Adatbányászat, adatfeldolgozás gyakorlatok

Jegyzet Geoinformatika mesterszakos hallgatóknak

Dr. Ungvári Zsuzsanna
egyetemi adjunktus
ELTE IK Térképtudományi és Geoinformatikai Intézet
Budapest, 2023

Tartalom

–

Követelmények, előismeretek

A jegyzetben nem a klasszikus adatbányászati módszerekkel és szoftverekkel fogunk foglalkozni, hanem a térinformatikai célú adatnyerésre, adatok előfeldolgozására és térinformatikai formátumúvá alakítására és feldolgozására fogjuk fektetni.

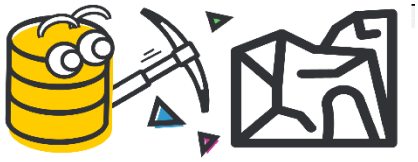
A tananyag tervezése során figyelembe vettem olyan térképészeti és geoinformatikai problémákat is, amelyek megvalósításához programozás lehet szükséges. Mivel a térinformatikai szoftverekben az utóbbi években egyértelműen a Python programozási nyelv lett az általánosan elterjedt, ezért minden példaprogram ezen a nyelven készült.

Az anyagban szereplő példákban szükség van egy térinformatikai szoftver pl. QGIS, valamint az SQL nyelv alapszintű ismertére. Az *Adatbányászat, felhő alapú adatok* című kurzus az ELTE Geoinformatika mesterképzésében párhuzamosan fut a *Térbeli adatbázisok* tárggyal. Az ott elsajátított ismeretekre szükség lesz a GTFS formátum című fejezetben.

Ajánlott olvasmány a témában:

Pang-Ning Tan, Michael Steinbach, Vipin Kumar: Bevezetés az adatbányászatba. Panem Könyvkiadó, 2011.

https://gyires.inf.unideb.hu/KMITT/a04/ch01.html#idp106326144



Lektorálta: Dr. Gede Mátyás.

1. fejezet: Bevezetés az adatbányászatba

Az adatbányászat fiatal tudomány: a nagyméretű adatbázisok hívták életre. Az adatbányászatba minden olyan eszköz és módszer beletartozik, amellyel nagyméretű adatbázisokat dolgozunk fel, illetve a feldolgozás bizonyos lépéseit automatizálhatjuk. Az adatbányászat eredményeként létrejövő anyagokból pedig új információk szűrhetők le.

Az adatbányászat a következő lépésekre bontható:

Bemenő adatok → Előfeldolgozás → Adatbányászat → Utófeldolgozás → Információ

Az előfeldolgozás célja, hogy a különböző formátumban tárolt adatokat egységes rendszerbe hozzuk, további feldolgozásra alkalmassá tegyük. Az előfeldolgozás tartalmazza a nyers adatok tisztítását, közös formátumúra hozását, a zajok csökkentését, hibák eltávolítását vagy javítását. Nem mindig van szükség minden adatra, így esetenként a különféle feltételek szerint is szűrni kell.

Az adatbányászat során az adatok tényleges feldolgozása történik, akár matematikai vagy egyéb módszerek igénybevételével. Az utófeldolgozás során az eredmények közlése, grafikus megjelenítése történik. Ezek az eredmények be tudnak épülni később a rendszerbe.

Adattudomány (Data Science)

Egy komplex új informatikai tudományterület, amely az adatok (jellemzően a big data) gyűjtésével, tárolásával, az adatok kutatásával és feldolgozásával, elemzésével, vizualizációjával, a feldolgozási algoritmusok és módszerek fejlesztésével, kutatásával, optimalizálásával, vizsgálatával foglalkozik (beleértve az AI és deep learning módszereket).

Big Data

Ehhez a fogalomhoz nem csak a nagymennyiségű adat, hanem a hardver-szoftver környezet és hálózatimodellek is hozzátartoznak. A *big data* nagy mennyiségű, gyakran változó adatok nagysebességű feldolgozását jelenti. A nagy adatmennyiséget tárolhatjuk felhőkben is, vagy több komputeren elosztva (mindenről valahol van egy másolat). Az adatok feldolgozásához sokszor olyan rendszereket használunk, amely több számítógépnek is kiadják, magyarul elosztják a feldolgozási folyamatot, így csökkentve a feldolgozási időt.

Mivel fogunk mi foglalkozni?

- Adatnyerés internetes adatbázisokból, különös tekintettel az előfeldolgozásra.
- Szöveges, illetve egyéb formátumok (helyre vonatkozó adatok esetében) térinformatikai adattípussá alakítása.
- Az előfeldolgozott adatokból információ nyerése néhány egyszerűbb statisztikai módszerrel.
- Utófeldolgozás, az adatok megjelenítése.
- Programozási feladatok Python környezetben, adatok előkészítése.
- Térinformatikai feladatok QGIS környezetben

2. fejezet: OVERPASS TURBO API

Az Overpass Turbo API egy webes adatbányász eszköz az OpenStreetMap adatbázisához. A webes felületen lehet lefuttatni a lekérdezést, és az interaktív térképes nézegetőablakban megjeleníthető, (később letölthető az eredmény).

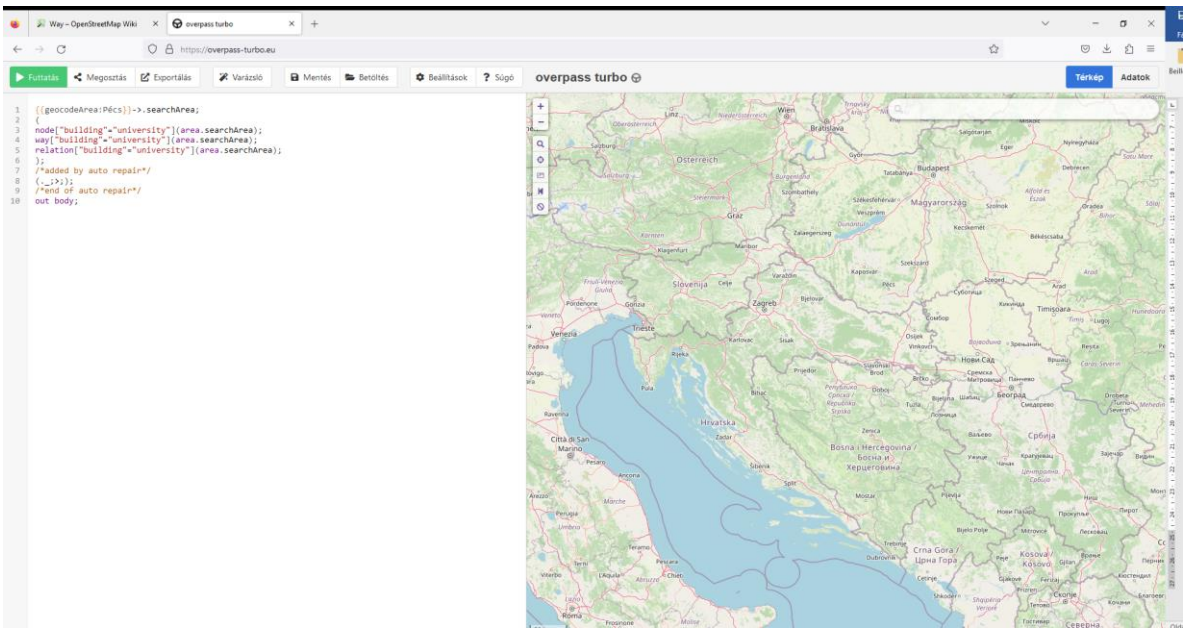
Dokumentáció:

https://wiki.openstreetmap.org/wiki/Overpass_turbo

Lekérdezések Overpass-ben

Lekérdezések végrehajtása Overpass QL (Query Language) vagy Overpass XML nyelvek segítségével lehetséges. Mindkét esetben egy HTTP GET kérés megy el a hálózaton keresztül. Az QL nyelv sokkal tömörebb szintaktikailag, mint az XML.

A legegyszerűbb talán az Overpass Turbo hivatalos weboldalát használni, amely így néz ki:



A weboldal két fő része a baloldalon látható szövegszerkesztő, amelybe a lekérdezéseinket tudjuk beírni, a jobboldalon pedig a válaszban visszatérő térbeli adatokat lehet megtekinteni.

A menürendszerből pedig a *Wizard (Varázsló)* és *Export (Exportálás)* funkciókat emelném ki. Az Overpass Turbo-ban a lekérdezések kétféleképpen futtathatók: az egyik módszer, hogy egy rövid QL szkriptet írunk, a másik, akár ezzel párhuzamos lehetőség, hogy használjuk a *Varázsló* (*Wizard*) funkciót. A következő példákban mindkettőt párhuzamosan alkalmazzuk.

Az Exportálás menüben lementhetők a saját gépünkre a visszatérő adatok GeoJSON, GPX, KML és nyers osm adatok (XML) formátumban.

Az Overpass XML egyébként a térkép ablak felső részén lévő *Data (Adatok)* menüre kattintva látható (milyen XML válasz tér vissza).

Overpass QL alapjai

Adattípusok az OSM adataiban

Az OpenStreetMap háromféle, egymásra épülő elemtípussal dolgozik: *node*-okkal, *way*ekkel és *relation*ökkel. Az elemek leíró attribútumait kulcs-érték párok, ún. *tag*ek tartalmazzák

Node. Egy koordinátapárból és egy *node id*-ből áll, ezek mellé attribútum adatok (*tag*) társulnak. Magassági adatot is tartalmazhat. A *node*-ok állhatnak önmagukban, vagy egy *way*-t alkotnak.

https://wiki.openstreetmap.org/wiki/Node

```
<node id="6454986742" lat="46.0725973" lon="18.2051926">
    <tag k="entrance" v="main"/>
```

```
<node id="1760195778" lat="46.0690616" lon="18.2160408">
    <tag k="amenity" v="restaurant"/>
    <tag k="dog" v="yes"/>
    <tag k="name" v="Szent György Fogadó"/>
    <tag k="toilets:wheelchair" v="yes"/>
    <tag k="website" v="http://www.szentgyorgyfogado.hu/"/>
    <tag k="wheelchair" v="yes"/>
```

Way. hétköznapi nyelven: vonallánc/polyline/poligon. A *way node*-ok rendezett sorozata alkotja, emellett attribútum adatokat is rendelünk hozzájuk, ezek a *tag*ek. A *way* kétféle lehet, zárt vagy nyílt vonallánc.

https://wiki.openstreetmap.org/wiki/Way

Open way: az első és az utolsó *node* nem egyezik meg, pl. utak, folyók. Ezeknek a *way*-eknek mindig van irányítottsága.

```
  <way id="375355485">
    <nd ref="27313594"/>
    <nd ref="4042642645"/>
    <nd ref="3787287460"/>
    <tag k="HU:ed_direction" v="forward"/>
    <tag k="highway" v="primary"/>
    <tag k="maxspeed" v="90"/>
    <tag k="ref" v="71"/>
    <tag k="ref:HU:edid" v="71u2k276m"/>
    <tag k="source:maxspeed" v="HU:rural"/>
    <tag k="surface" v="asphalt"/>
    <tag k="toll:hgv" v="yes"/>
  </way>
```

Closed way: az első és az utolsó *node* megegyezik. A *closed way* lehet zárt polyline, poligon (felület), a pontos mivoltát a *tag* határozza meg. Pl. zárt polyline lehet egy körforgalom.

Area: az első és az utolsó *node* megegyezik. Ez is egy *closed way*: a poligon (felület), a pontos mivoltát a tag határozza meg. vagyis nem mindig nekünk kell megadni, pl. a felület típusa erdő, akkor automatikusan felületként fog viselkedni. Egyébként bizonyos elemeknél megadható, hogy felületről, vagy önmagába visszatérő vonalláncról lesz-e szó, például highway=pedestrian + area=yes – gyalogos zóna/sétálóutca.

```
<way id="143854005">
    <nd ref="1574148556"/> <nd ref="1574148670"/>
    <nd ref="1574148682"/> <nd ref="2621432173"/>
    <nd ref="2621432172"/> <nd ref="2621432167"/>
    <nd ref="2621432168"/> <nd ref="1574148558"/>
    <nd ref="1574148556"/>
    <tag k="building" v="university"/>
    <tag k="name" v="K-épület"/>
  </way>
```

Rel (relation). A relációk vagy kapcsolatok tulajdonképpen az elemek (*node*-ok, *way*-ek, vagy *relation*ök) egy csoportja. Egy vagy több tag tartozik hozzájuk. *Relation*ök típusai: multipoligonok, *bus route* (útvonal), határok, közlekedési elemek, épület stb.

https://wiki.openstreetmap.org/wiki/Relation

```
<relation id="9684822">
    <member type="node" ref="1382931855" role="platform"/>
    …
    <member type="node" ref="1382931862" role="platform"/>
    <member type="way" ref="819269604" role=""/>
    …
    <member type="way" ref="819269609" role=""/>
    <tag k="from" v="Kerepes HÉV-állomás"/>
    <tag k="name" v="Helyi járat &quot;A&quot;: Kerepes HÉV-állomás
=&gt; Patkó Csárda =&gt; Iskola =&gt; Berzsenyi uca =&gt;
Szilasliget-Kemping"/>
    <tag k="operator" v="Regio 2007 Kft."/>
    <tag k="public_transport:version" v="2"/>
    <tag k="ref" v="Helyi járat"/>
    <tag k="route" v="bus"/>
    <tag k="to" v="Szilasliget-Kemping"/>
    <tag k="type" v="route"/>
    <tag k="via" v="Patkó Csárda -&gt; Iskola -&gt; Berzsenyi
uca"/>
  </relation>
```

Hogyan adjuk meg a lekérdezés célterületét?

Írjunk be lekérdezést az Overpass Turbo felületre, lásd 5. oldal. Miután megadtuk, hogy milyen elemet szeretnénk lekérdezni, definiálnunk kell, hogy milyen területről szeretnénk lekérni ezeket az adatokat. Többféle lehetőségünk van, a legegyszerűbb a nézet befoglaló téglalapja (*Canvas extent*).

Nézzünk egy példát! Írjuk be a varázslóba, hogy *amenity=restaurant*. Állítsuk a térképi nézetet egy nagyvárosra:

```
(
  // query part for: "amenity=restaurant"
  node["amenity"="restaurant"]({{bbox}});
  way["amenity"="restaurant"]({{bbox}});
  relation["amenity"="restaurant"]({{bbox}});
);
// print results
out body;
```

bbox (*bounding box*). Ebben az esetben a **nézet határoló koordinátáit** kérjük le, más szóval a nézet **befoglaló téglalap**ját. Kétféleképpen oldható meg ezen belül.

Ebben az esetben az éppen a képernyőn látható térképi területről kérdez le adatokat.

```
({{bbox}});
```

A nézet befoglalójának határoló parallelköreit és meridiánjait adjuk meg a következő sorrendben: déli határoló szélesség, nyugati határoló meridián, északi határoló szélességi kör, és keleti határoló meridián) sorrendben:

```
(47.4, 18.5, 47.8, 19.3);
```

Ha a lekérdezésben szereplő terület átmegy a 180°-os meridiánon, akkor két részre kell bontani a *bbox*ot: egy nyugati és egy keleti félre.

```
1   (
2     /*your query here*/(51.0,  7.0, 52.0, 180); /* West side of the antimeridian (up to the positive
longitude 180°E) */
3     /*your query here*/(51.0, -180, 52.0, 8.0); /* East side of the antimeridian (from  the negative
longitude 180°W) */
4   );
```

Egy pont körül valamilyen sugarú körben keresünk objektumokat.

Ebben az esetben legegyszerűbb két koordinátapárt megadni. Első paraméter: távolság az adott ponttól, majd földrajzi szélesség és hosszúság.

```
(around: 2000, 47.4866, 19.0567);
```

Persze, ha nem ismerjük a földrajzi szélességet és hosszúságot, megadható egy cím is. Keressünk éttermeket a Deák Ferenc tér 500 méteres körzetében. Ilyenkor a *geocodeCoords* paramétert kell használni.

```
{{radius=500}}
(
node["amenity"="restaurant"](around:{{radius}},{{geocodeCoords:Buda
pest Deák Ferenc tér 1}});
); out;
```

Lehetséges olyan eset, hogy **egy felület lesz a befoglaló**, például bármely igazgatási egység, (megye vagy településhatár, stb.)

Ebben az esetben, ha meg tudjuk adni az igazgatási egység nevével (pl. településnév) a területet, akkor lehet a geokódolási lehetőséget használni. Megjegyzendő, hogy ilyenkor a település területe megegyezik a település külterülethatárával.

```
{{geocodeArea:Ercsi}}->.searchArea
node["amenity"="restaurant"](area.searchArea);
```

A geokódolás településnév mellett más adatra is működik, pl. irányítószámra.

Geokódolás nélkül is megadható a terület. Definiálunk változót (itt: *area*), amiben lekérdezzük a területet, majd ez változó köszön vissza befoglalóként a lekérdezendő adatokra.

```
area
  [place=region]
  ["region:type"="mountain_area"]
  ["name:en"="Dolomites"];
out body;
  node["building"="castle"](area);
```

Ahhoz, hogy lekérdezzünk az OSM adatbázisából ismernünk kell, hogy milyen kulcs-értékpárok fordulhatnak elő benne.

A részletekért látogassuk meg a következő linket, ahol minden térképi elem (az összes lehetséges tag) benne van.

https://wiki.openstreetmap.org/wiki/Map_features

Lekérdezés értékeinek megadása

A lekérdezésből ne hiányozzon lezárásként az **out;**

Ha egyértelműen egy kulcs-érték párra keresünk az OSM adatbázisban, akkor az egyenlőségjelet használjuk.

```
["building"="castle"]
```

Ha az értékben szeretnénk részleges vagy teljes szövegegyezést keresni (mint az SQL-ben a LIKE), akkor hullámvonalat használunk. Például tartalmazza a *bicycle* szövegrészletet, pl. *bicycle_parking, bicycle_repair_station, bicycle_rental.* (Megjegyzés, ha a Varázslót használjuk a lekérdezés felépítéséhez az *amenity:bicycle* is működik. Amikor felépítjük a lekérdezést, akkor átírhatjuk a kettőspontot hullámvonalra).

```
["amenity"~"bicycle"]
```

Ha szeretnénk lekérdezni, az adott kulcs értéknek milyen értékei vannak egy adott területen, akkor azt így fogalmazzuk meg a Varázslóban: pl. *amenity=**. Ebből ez lesz lekérdezésként:

```
["amenity"]
```

Ha azokat az elemeket szeretnénk lekérdezni, amelyeknek nem az az értéke, amit megadtunk, akkor használjuk a != műveletet. Kérdezzük le a Lágymányosi Campus környékéről azokat az elemeket, amelyek értéke nem *bicycle_rental*. Varázslóban: *amenity!=bicycle_rental.* Lekérdezésként:

```
["amenity"!="bicycle_rental"]
```

Vigyázzunk: kis területre is sok adat jön le, mert minden elemet lekérdez a megadotton kívül!

Hasonló a helyzet, ha szövegrészlettel nem egyező adatokat akarjuk leválogatni. Ilyenkor *amenity!~bicycle* a Varázslóban, lekérdezésként pedig:

```
["amenity"!~"bicycle"]
```

Vigyázzunk: ha csak ennyi a feltétel, kis területre is sok adat jön le, mert minden elemet lekérdez a megadotton kívül!

Ha például az *amenity* elemeken kívül mindent le akarunk kérdezni, akkor a Varázslóban: *amenity!=**, vagy lekérdezésként:

```
["amenity"!~".*"]
```

A timeout

```
[timeout:180]
```

A lekérdezésre maximálisan engedélyezett idő másodpercben. (Ha túllépi, leáll). Nem kötelező része a lekérdezésnek.

[out:json]

Ha a lekérdezést *[out:json]*-nel nyitjuk, a visszatérő adatok json formátumban lesznek, ha nem tesszük bele, akkor XML-t kapunk vissza:

```
out:json][timeout:25];
  (
   // query part for: "amenity=restaurant"
   node["amenity"="restaurant"]({{bbox}});
   way["amenity"="restaurant"]({{bbox}});
   relation["amenity"="restaurant"]({{bbox}});
);
out body;
```


```
 [timeout:25];
 (
  // query part for: "amenity=restaurant"
```

```
  node["amenity"="restaurant"]({{bbox}});
  way["amenity"="restaurant"]({{bbox}});
  relation["amenity"="restaurant"]({{bbox}});
);
out body;
```

Lekérdezés lezárása: out; és egyéb változataival

A lekérdezéseket le kell zárni. A legegyszerűbb és legrövidebb formája ennek az *out;*

Attól függően, hogy milyen részletesen – bőbeszédűen – szeretnénk kiíratni az eredményt, az *out* és a pontosvessző közé írhatunk még parancsokat. Az *out body;* az alapértelmezett, az *out;*-tal megegyező a válaszban visszatérő elemeket írja ki. Ezenkívül használhatjuk még az *out ids; tag*et, ekkor csak a visszatérő elem *id*-je íródik ki. Az *out skel;* csak a koordinátákat (+*id* is) írja ki.

Lásd itt részletesen:

https://wiki.openstreetmap.org/wiki/Overpass_API/Overpass_QL#Output_format_(out:)

Egyéb nyelvi finomságok az Overpass QL nyelvben:

https://wiki.openstreetmap.org/wiki/Overpass_API/Overpass_QL

Gyakorlatok

A következő gyakorlatok az Overpass Turbo API felületén oldhatók meg:

https://overpass-turbo.eu/

1. feladatcsoport: nézetek gyakorlása

1.1 feladat

Állítsuk az térképi nézetet Budapestre, és keressük meg a bárokat.

A bárok *amenity* objektumok (*amenity* ~ mint kényelmi szolgáltatások). A varázslóban: *amenity=bar*.

```
node[amenity="bar"]({{bbox}});
out;
```

A vizsgált területet zárójelben adjuk meg: ();

Ha az aktuális térképi nézet az, akkor *{{bbox}}*-szal helyettesítjük.

1.2 feladat

Most kérdezzük le Budapest és a Dunakanyar bárjait, a nézetet koordinátákkal adjuk meg.

```
node[„amenity”="bar"] (47.4, 18.7, 47.8, 19.2);
out;
```

A vizsgált területet koordinátákkal szeretnénk kifejezni, akkor a déli határoló szélesség, nyugati határoló meridián, északi határoló szélességi kör, és keleti határoló meridián a sorrend.

1.3 feladat

Tegyük bele a lekérdezésbe, hogy meg lehet-e közelíteni kerekesszékkel-e a bárt, vagy sem (akadálymentesség vizsgálata, ahol van adat).

```
node[„amenity”="bar"]
     [wheelchair=yes](47.4, 18.7, 47.8, 19.2);
out;
```

1.4 feladat

Kérdezzük le a címében budapestinek jelölt bárokat (*addr:city*) ugyanerre a nézetre. Ha a Varázslót használjuk, akkor *amenity=bar AND addr:city=Budapest*. (Egy objektum kétféle attribútumára keresünk).

```
node["amenity"="bar"]
      ["addr:city"="Budapest"]
     (47.4, 18.7, 47.8, 19.2);
out;
```

1.5 feladat

Keressünk az álláspontunkhoz közeli (2 km távolságon belüli) szórakozóhelyet (bárt) Budapest belvárosában. Álláspontunk 47.4866° és 19.0567°. Legyen akadálymentes.

```
node["amenity"="bar"]
     ["wheelchair"="yes"]
      (around:2000,47.4866, 19.0567);
 out body;
```

1.6 feladat

Mivel némely elemek nemcsak *node*-ként fordulnak elő, hanem *way*ként és *relation*ként is lehetnek definiálva, ezért ezeket is célszerű beleszőnünk a lekérdezésbe. A lekérdezés gyakorlatilag ugyanaz lesz, azzal a kis különbséggel, hogy *way* illetve *relatio*n is szerepel a *nod*e mellett, illetve zárójelbe fogjuk őket össze.

Álláspontunkhoz közeli (500 méteres sugarú körben) keressük meg az éttermeket. Álláspont: Budapest Deák Ferenc tér 1.

```
[out:json][timeout:25]; //hozzáadható, a lekérdezés lefutására
hagyott idő.
{{radius=500}}  // megadható így is. {{ }} jelenti a változó
kiíratását a későbbi kifejezésben.
 (
node["amenity"="restaurant"](around:{{radius}},{{geocodeCoords:Buda
pest Deák Ferenc tér 1}});
way["amenity"="restaurant"](around:{{radius}},{{geocodeCoords:Budap
est Deák Ferenc tér 1}});
relation["amenity"="restaurant"](around:{{radius}},{{geocodeCoords:
Budapest Deák Ferenc tér 1}});
);
out body;
```

1.7 feladat

Adjunk új stílust a pontoknak. Legyen mindnek kék kitöltése. A hely nevét írja ki attribútumként.

A stílusbeállítások elvégzéséhez a MapCSS leíró nyelvet fogom használni (egyelőre eléggé gyerekcipőben jár, és minimális formázást enged).

https://wiki.openstreetmap.org/wiki/MapCSS/Examples

```
 node["amenity"="bar"]
       ["wheelchair"="yes"]
       ["addr:city"="Budapest"]
     (47.4, 18.7, 47.8, 19.2);
{{style:
    node { fill-color: blue; color: blue, fill-opacity 1; font-
size:16; text: name; text-position: center; }
}}
 out body;
```

1.8 feladat

Kérdezzük le egyszerre kétféle objektumot a megadott nézetben (pl. Budapest belváros) lévő éttermeket és bárokat.

```
(
  node["amenity"="restaurant"]({{bbox}});
  node["amenity"="bar"]({{bbox}});
);
out;
```

1.9 feladat

A varázsló használata (Wizard). Keressük meg a bárokat és éttermeket. A varázslóba írjuk be, hogy *amenity=bar OR amenity=restaurant*. Vigyázzunk, mert itt az *OR*-t kell használni, mivel egyszerre egy objektum nem lehet kétféle attribútumú.

```
(
  node["amenity"="restaurant"]({{bbox}});
  way["amenity"="restaurant"]({{bbox}});
  relation["amenity"="restaurant"]({{bbox}});
  node["amenity"="bar"]({{bbox}});
  way["amenity"="bar"]({{bbox}});
  relation["amenity"="bar"]({{bbox}});
);
out body;
```

A varázsló segít összeállítani a lekérdezést: lehet olyan eset, hogy az adott elemet nem, vagy nemcsak pontként tároljuk, hanem *way*ként és/vagy *relation*ként.

2. feladatcsoport

2.1 Épületek lekérdezése

Az épületek lekérdezésénél ügyeljünk arra, hogy ezek az elemek *node*-ként, *way*ként és *relation*ként is előfordulhatnak. Kérdezzük le az adott nézetben pl. Budapest belvárosában lévő lakóépületeket. A Varázslóban *building=residential*.

```
(
  node["building"="residential"]({{bbox}});
  way["building"="residential"]({{bbox}});
  relation["building"="residential"]({{bbox}});
);
out body;
```

Kérdezzük le az egyetemeket! A Varázslóban *building=university*.

```
(
  node["building"="university"]({{bbox}});
  way["building"="university"]({{bbox}});
  relation["building"="university"]({{bbox}});
);
out body;
```

3. feladatcsoport

3.1. feladat

Kérdezzük le egy megadott nézetre az erdőket (a nézet legyen valahol a Budai-hegységben). Az erdők (*forest*) a *landuse* kategóriában vannak.

```
(
  node["landuse"="forest"]({{bbox}});
  way["landuse"="forest"]({{bbox}});
  relation["landuse"="forest"]({{bbox}});
);
out body;
```

3.2 feladat

Kérdezzük le az erdőket egy adott településre, pl. Ercsi. Ehhez érdemes lehet a Varázslót használni.

Varázsló: *landuse=forest in Ercsi*

```
{{geocodeArea:Ercsi}}->.searchArea;
 (
  node["landuse"="forest"](area.searchArea);
  way["landuse"="forest"](area.searchArea);
  relation["landuse"="forest"](area.searchArea);
);
out body;
```

3.3 feladat

Keressük meg a kastélyokat (*castle*) Nógrád megyében! Vigyázni kell, mert nemcsak a *building*, hanem a *historic* kategóriában is van egy *castle* érték!

building=castle or building historic=castle in Nógrád

```
{{geocodeArea:Nógrád}}->.searchArea;
 (
  node["building"="castle"](area.searchArea);
  way["building"="castle"](area.searchArea);
  relation["building"="castle"](area.searchArea);

  node["historic"="castle"](area.searchArea);
  way["historic"="castle"](area.searchArea);
  relation["historic"="castle"](area.searchArea);
);
out body;
```

Másképpen:

```
area
  ["boundary"="administrative"]
  ["admin_level"="6"]
  ["name"="Nógrád megye"];
out body;
 (
  node["historic"="castle"](area);
  way["historic"="castle"](area);
  relation["historic"="castle"](area);
  node["building"="castle"](area);
  way["building"="castle"](area);
  relation["building"="castle"](area);
);
out body;
```

4. feladatcsoport: Közigazgatási egységek

4.1 feladat

Kérdezzük le Magyarország adatait, állítsuk a nézetet a Kárpát-medencére. A *place tag*ben az ország egy pontként tér vissza (tehát nem a határt tartalmazza). Amire ez jó: rengeteg nyelven megvan benne attribútumként az egyes országok nevei különböző nyelveken.

```
(
  node["place"="country"]["name"="Magyarország"]({{bbox}});
  way["place"="country"]["name"="Magyarország"]({{bbox}});
  relation["place"="country"]["name"="Magyarország"]({{bbox}});
);
out body;
```

4.2. feladat

Menjünk végig a közigazgatási egységek határainak lekérdezésén. Ezekhez a *boundary=administrative and admin_level=?* lekérdezést kell beírni a Varázslóba. A kérdőjel

helyére egy szám kerül. Minden országnál a közigazgatási beosztástól függően eltérő lehet az, hogy melyik szám mit jelent. Lásd az összefoglaló táblázatot:

https://wiki.openstreetmap.org/wiki/Tag:boundary%3Dadministrative#admin_level=*_Country_specific_values

Magyarországra:

Admin_level	**Közigazgatási egység**	**Admin_level**	**Közigazgatási egység**
2	országhatárok	6	Vármegyék / főváros – NUTS 3
3	-	7	Járások
4	Országrészek – NUTS 1	8	Települések
5	Régiók – NUTS 2	9	Kerületek

Kezdjük a vármegyékkel:

```
(
node["boundary"="administrative"]["admin_level"="6"]({{bbox}});
way["boundary"="administrative"]["admin_level"="6"]({{bbox}});
relation["boundary"="administrative"]["admin_level"="6"]({{bbox}});
);
out body;
```

Folytassuk a járásokkal, amelyek az *admin_level 7*-es (járások) szinten vannak.

```
(
node["boundary"="administrative"]["admin_level"="7"]({{bbox}});
way["boundary"="administrative"]["admin_level"="7"]({{bbox}});
relation["boundary"="administrative"]["admin_level"="7"]({{bbox}});
);
out body;
```

A települések lekérdezése az *admin_level 8*-as szintről.

```
node["boundary"="administrative"]["admin_level"="8"]({{bbox}});
way["boundary"="administrative"]["admin_level"="8"]({{bbox}});
relation["boundary"="administrative"]["admin_level"="8"]({{bbox}});
);
out body;
```

Régiók lekérdezése. A régiókból Magyarországra vonatkozólag kétféle van. Az 5-ös *admin_level* szinten a hét régió szerepel, a 4-es szinten három régió van: Nyugat-Magyarország, Kelet-Magyarország, és a Központi-régió.

```
 (
node["boundary"="administrative"]["admin_level"="5"]({{bbox}});
way["boundary"="administrative"]["admin_level"="5"]({{bbox}});
relation["boundary"="administrative"]["admin_level"="5"]({{bbox}});
);
out body;
```

Az országhatárok az *admin_level* 2-es szintjén vannak tárolva.

```
(
node["boundary"="administrative"]["admin_level"="2"]({{bbox}});
way["boundary"="administrative"]["admin_level"="2"]({{bbox}});
```

```
relation["boundary"="administrative"]["admin_level"="2"]({{bbox}});
);
out body;
```

A bemutatott példák az adott nézetre kérdezik le határokat. Mi van akkor, ha egy konkrét megye határa kell?

```
{{geocodeArea:Nógrád}}->.searchArea;
 (

node["boundary"="administrative"]["admin level"="6"](area.searchAre
a);
way["boundary"="administrative"]["admin_level"="6"](area.searchArea
);
relation["boundary"="administrative"]["admin_level"="6"](area.searc
hArea);
);
out body;
```

Vagy másképpen, kissé nézet függően Nógrád vármegye határai, ekkor Nógrád vármegyének látszania kell a *bounding box*ban

```
 (
node["boundary"="administrative"]["admin_level"="6"]["name"="Nógrád
megye"]({{bbox}});

way["boundary"="administrative"]["admin_level"="6"]["name"="Nógrád
megye"]({{bbox}});
relation["boundary"="administrative"]["admin_level"="6"]["name"="Nó
grád megye"]({{bbox}});
);
out body;
```

Keressük meg Apc község határát. Nézet függően (a nézet legyen pl. Kelet-Magyarország). Vigyázzunk, a 8-as (településszinten) keresünk!

```
 (
node["boundary"="administrative"]["admin_level"="8"]["name"="Apc"](
{{bbox}});
way["boundary"="administrative"]["admin_level"="8"]["name"="Apc"]({
{bbox}});
relation["boundary"="administrative"]["admin_level"="8"]["name"="Ap
c"]({{bbox}});
);
/*added by auto repair*/
(._;>;);
/*end of auto repair*/
out body;
```

Másképpen, nézetfüggetlenül geokódolással.

```
[out:json][timeout:25];
{{geocodeArea:Apc}}->.searchArea;
 (

node["boundary"="administrative"]["admin_level"="8"](area.searchAre
a);
way["boundary"="administrative"]["admin_level"="8"](area.searchArea
);
relation["boundary"="administrative"]["admin_level"="8"](area.searc
hArea);
```

```
);
/*added by auto repair*/
(._;>;);
/*end of auto repair*/
out body;
```

5. feladatcsoport

5.1 feladat

Keressük meg a 1163-as irányítószámmal rendelkező éttermeket!

```
{{geocodeArea:1163}}->.searchArea;
(
  node["amenity"="restaurant"](area.searchArea);
  way["amenity"="restaurant"](area.searchArea);
  relation["amenity"="restaurant"](area.searchArea);
);
out body;
```

5.2 feladat

Keressük meg a Dolomitok hegycsúcsait!

```
area
  [place=region]
  ["region:type"="mountain_area"]
  ["name:en"="Dolomites"];
out body;

node
  [natural=peak]
  (area);
out body;
```

5.3 feladat

Kérdezzük le Nógrád vármegye hegycsúcsait!

```
area
  ["boundary"="administrative"]
  ["admin_level"="6"]
  ["name"="Nógrád megye"];
out body;
node
  [natural=peak]
  (area);
out body qt;
```

5.4 feladat

Jelöljük ki Nógrád vármegye településeit!

```
 [out:json][timeout:25];
{{geocodeArea:Nógrád}}->.searchArea;
 (

node["boundary"="administrative"]["admin_level"="8"](area.searchAre
a);
way["boundary"="administrative"]["admin_level"="8"](area.searchArea
);
relation["boundary"="administrative"]["admin_level"="8"](area.searc
hArea);
);
out body;
```

3. fejezet: GTFS adatformátum

Sok közlekedési társaság rendszeresen közzéteszi menetrendjeit egy ún. GTFS formátumban a weben. Ezáltal a fejlesztőknek lehetősége nyílik felhasználni saját térképes alkalmazásaikba ezeket az adatokat. „A rendszeresen frissített adatbázis néhány hónapra előre tartalmaz minden indulási és érkezési adatot a teljes nappali és éjszakai hálózatra vonatkozóan minden ágazatban (busz, trolibusz, villamos, metró, HÉV, hajó), így megjeleníthető a vonalhálózat, valamint minden megálló pontos helye a GPS-koordináták segítségével."[1]

Mi a GTFS?

A GTFS (General Transit Feed Specification) a Google által kifejlesztett, teljesen nyilvános és ingyenes formátum, amely a földrajzi pozíciókat is felhasználó alkalmazásokba beépíthetővé teszi a közösségi közlekedés menetrendi adatbázisait.

A BKK GTFS adatainak elérhetősége:

https://bkk.hu/fejlesztesek/fejlesztoknek/

A GTFS angol nyelvű dokumentációja:

http://developers.google.com/transit/gtfs/

A GTFS valójában CSV formátumú szövegfájlok gyűjteménye, amelyek egy zip fájlban vannak összefogva. A fájlok neve kötött, pl.: stops, routes, trips, stb. Alább olvasható e szöveges fájlok szerkezete.

AGENCY.TXT

A járatüzementető adatai.

`agency_id`	Vállalat azonosítója pl. BKK, vagy HEV
`agency_name`	Vállalat neve
`agency_lang`	kétjegyű nyelvkód
`agency_phone`	Utastájékoztató telefonszám
`agency_email`	Utastájékoztató email
`agency_fare_url`	Utastájékoztató weboldal

STOPS.TXT

`stop_id`	Megálló ID, ha ugyanaz a tér, de két távolabbi pontján van a megálló, akkor különböző ID

[1] https://bkk.hu/fejlesztesek/fejlesztoknek/

stop_name	Nem feltétlenül egyezik meg a megálló utasoknak közzétett nevével. Pl. Örs vezér tere M+H, déli tárolótér vagy Örs vezér tere M+H, északi tároló
stop_lat & stop_lon	Szélesség és hosszúság, földrajzi koordináták
stop_code	A megálló utasok számára is megadott azonosítója
parent_station & location_type	Ha egy megállóban több felszállási lehetőség is van, a szülő és „gyermek” állomások neve, és mire lehet felszállni
wheelchair_boarding	Kerekesszékes megközelíthetőség
stop_direction	Menetirány azimutja

ROUTES.TXT

agency_id	Üzemeltető ID
route_short_name	max 12 karakterhosszúságú útvonal név: **Járatszám**!
route_long_name	Kiind. és érk. megálló nevei
route_id	Az útvonal ID-je (belső azonosító)
route_color & route_text_color	nyomatatási színek, betűtípusok

TRIPS.TXT

route_id	Útvonal azonosító (belső azonosító)
trip_id	Járatazonosító
service_id	Egy másik, hosszú járatazonosító
trip_headsign	A járaton éppen kijelzett célállomás neve
direction_id	Útvonal azonosítója 0: egyik irány, 1: visszafelé irány
block_id	
shape_id	A pontokból képzett vonalak, vagyis a járat nyomvonalának azonosítója)
wheelchair_accessible	Kerekesszékes használat
boarding_door	Felszálláshoz igénybe vehető ajtók
bikes_allowed	Engedélyezett-e kerékpárszállítás

STOP_TIMES.TXT

`trip_id`	Járatazonosító
`stop_id`	Megálló azonosító
`arrival_time` & `departure time`	Indulási és érkezési idő
`stop_headsign`	A járaton lévő célállomás neve (Busz elején)
`stop_sequence`	Hányadik megálló a végállomástól kezdve
`shape_dist_traveled`	Távolság a kiindulási állomástól méterben

SHAPES.TXT

`shape_id`	A járat nyomvonalának azonosítója
`shape_pt_sequence`	A járat nyomvonalát jelző töréspontok sorrendje
`shape_pt_lat` & `shape_pt_lon`	A pont földr. szélessége és hosszúsága
`shape_dist_traveled`	Távolság a kiind. állomástól méterben

FEED_INFO.TXT

Az adatszolgáltató adatai és az adatok érvényessége.

Mit lehet kihámozni az adatsorból?

Az GTFS adatsorban az egyes fájlokban (ha adatbáziskezelőben gondolkozunk, akkor nevezzük tábláknak a fájlokat) elsődleges és idegen kulcsok vannak elhelyezve, amelyek a fájlok összekapcsolását biztosítják. Ha ténylegesen lekérdezésre akarjuk használni ezeket a szöveges fájlokat, akkor érdemes importálni egy térinformatikai szoftverbe vagy egy adatbáziskezelőbe. Az alábbiakban mind a kettőre fogunk példát látni.

A fenti fájlok értelmezése: a járatok nyomvonalát a *shapes* tábla tartalmazza. Ennek minden sora egy-egy nyomvonal egy töréspontja; az egy nyomvonalhoz tartozó pontok *shape_id*-ja azonos, és a *pt_sequence* mező adja járat nyomvonalának csomópontjait, annak sorrendjét. Ezt érdemes térinformatikai szoftverben egy megfelelő algoritmussal vonalas réteggé konvertálni. Így járatok vonalai és azonosítók lesznek a *shapes* rétegen.

A *stops* fájlban a megállók szerepelnek (pontkoordinátákkal) és a megálló nevével.

A *routes.txt*-ben az egyes járatok száma és a viszonylat szöveges megnevezése található (*route_short_nam*e).

A „Jolly Joker” táblák a *trips* és a *stop_times*, ezek segítségével tudjuk összekötni az előzőeket. A *trips* táblában a járat éppen kijelzett célállomása olvasható a *trip_headsign*

mezőben, és a járat útvonalát a *shape_id*-vel lehet azonosítani (a vonalas elem). A *route_id*-t használva a járat számát (*route_short_name*) tudjuk visszanyerni.

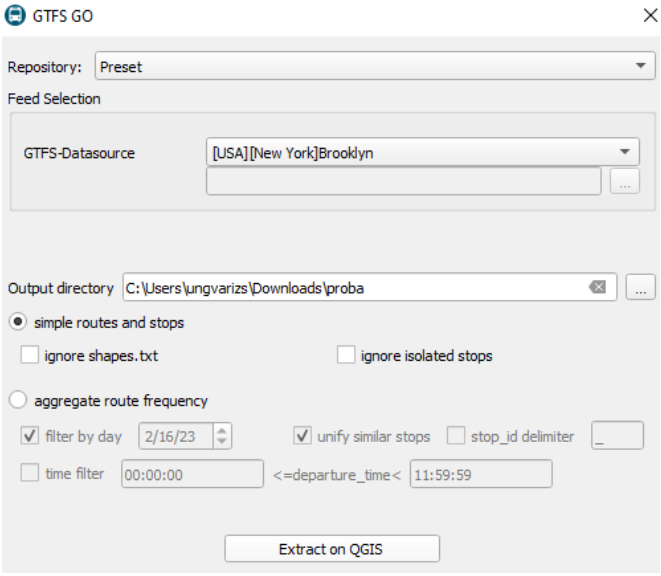
A *stop_times* összeköthető a *trips*-szel és a *stops* táblával: vagyis a járatok adatai és a megállók nevei is összekapcsolhatók. A megállóba érkezés és az onnan indulás időpontjai is láthatók.

Műveletek GTFS formátumú adatokkal: QGIS GTFS GO és GTFS Loader alkalmazás

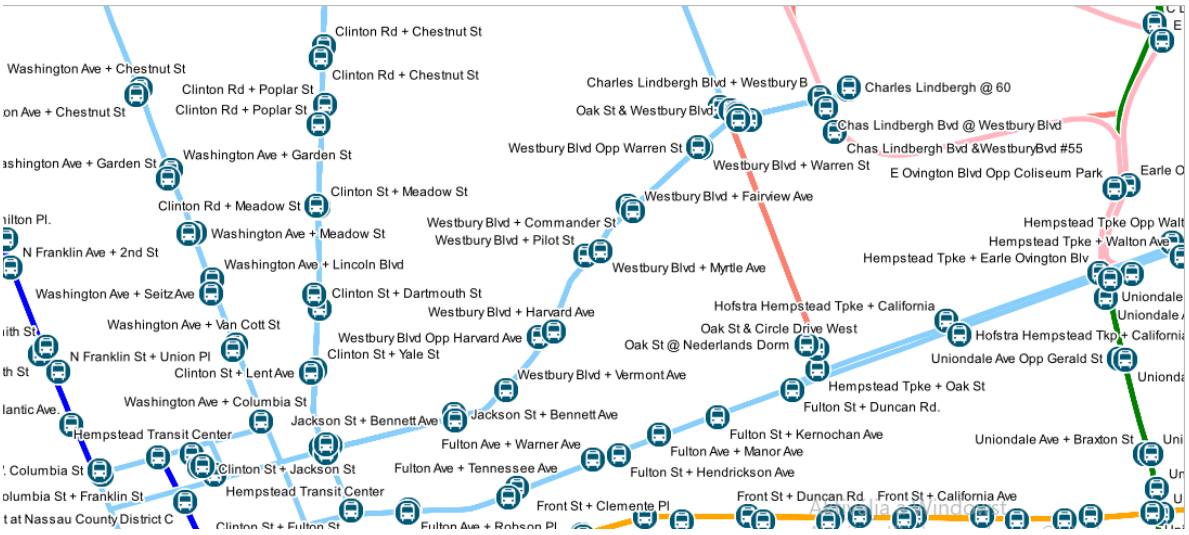
A QGIS-ben két plugin is rendelkezésre áll a GTFS adatok megjelenítésére. Sajnos mindkettőnek vannak hiányosságai.

A QGIS-ben *Plugins* → *Manage and Install Plugins* → *GTFS-GO*→*Install Plugin*

Töltsük be egy előre definiált adatsort a legördülő listából, pl. New York Brooklyn GTFS adatsort. Beállítások a képen.



Létrejön egy *Brooklyn* rétegcsoport két réteggel: *stops* és *routes*, az elsőn a megállók, a másodikon a járatok útvonala látható. Vizsgáljuk meg a *stops* és a *routes* rétegeket és attribútum-adataikat! A *stops* rétegen egy *stop_id*, egy *stop_name* (megálló neve) és egy *route_id* (járat azonosító) van. A *routes* rétegen a járat azonosítója és neve van. A QGIS a GTFS GO alkalmazás segítségével stílusokat rendel a rétegekhez, és a megállók neveit is kiírja.



Milyen vetületben vannak az adatok?

A GTFS fájlok földrajzi koordinátákat használnak (WGS-84, EPSG:4326).

Töltsük be a budapest_gtfs.zip állományt! (https://bkk.hu/gtfs/budapest_gtfs.zip)

Az alkalmazás segítségével egyszerűen lehet beolvasni GTFS fájlokat. A Local ZIP file opciót választva csupán ki kell választani a GTFS fájlt. Csak akkor működik, ha a szabványban megadott összes fájlt tartalmazza a ZIP. A BKK anyagából hiányzik a *calendar.txt*, így nem használható a modul. Viszont meglévő állományok a legördülő listából behívhatók, pl. new yorki adatok. Erre látunk egy példát a fenti képen.

QGIS GTFS Loader

A QGIS-ben *Plugins* → *Manage and Install Plugins* → *GTFS Loader* → *Install Plugin*

Ha behívja, a shapes réteg vonalként, a stops a megállók pont rétegként fognak szerepelni. A többi adat táblázatként jelenik meg. (Sajnos a GTFS Loaderrel is adódhatnak problémák).

Azonban pluginek nélkül is lehet használni a budapest_gtfs adatsort.

BKK adatok megnyitása

Importáljuk a két TXT fájlt, a *Data Source Manager*ben → *Delimited Textfile* (*stops.txt* és *shapes.txt*).

SHAPES.TXT

Elválasztó karakter: vessző, *Geometry definition: Point coordinates*. *X field* → *shape_pt_lon*, *Y field shape_pt_lat*. Vetület (CRS): 4326.

Ezután vonalakká kell alakítani a pontokat a *Processing* menüben → *Points to path*

Az *Order field*nél meg kell adni a pontok sorrendjét (*shape_pt_sequence*) és a *Group field*nél pedig a vonalak azonosító számát (*shape_id*).

Eredményül valami ilyesmit kapunk:

Hogyan tudom meg a *shape_id*-ből, hogy az micsoda, melyik busz, hova megy?

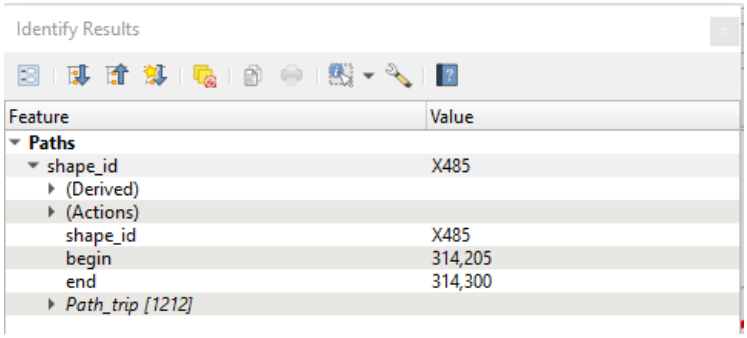
Hozzunk létre kapcsolatokat két tábla között.

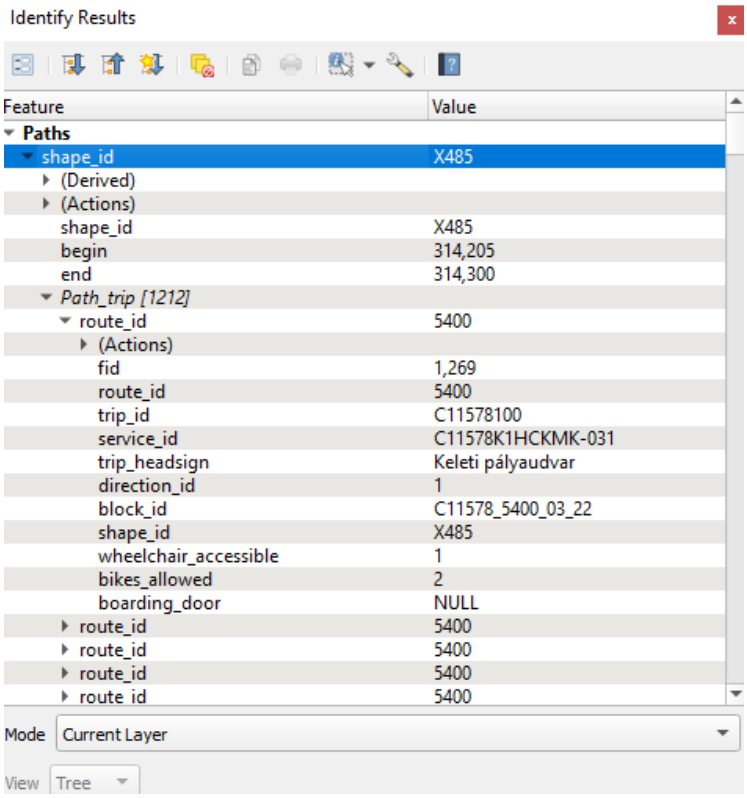
A *shapes* táblához (*shape_i*d elsődleges kulcs), kapcsoljuk a *trips* (*shape_id* idegen kulcs) táblát.

a) *Project* → *Properties* → *Relations* fül:→ + *Add relations*. Nevet adunk a kapcsolatnak, majd beállítjuk a kulcsokat. *SHAPES.shape id*- és *TRIP.shape_id*

*Trip*ek: hozzá lehet kapcsolni a *shape_id*-k alapján a *shape.shape_id*-t.

Majd bezárjuk és kikeressük az *Identify feature* gombot (kék körben *i* betű ikon), és rákattintunk az egyik *shape*-re, vagyis buszútvonalra.





A buszra kiírandó szöveget tudjuk (*trip_headsign*), de a járat számát még mindig nem. Ehhez a TRIP.*route_id* és a ROUTE.*route_id*-t kellene még kapcsolnunk.

Mi a helyzet, ha azt is szeretnénk megtudni, hogy melyik megállóban melyik busz áll meg és mikor?

Innentől már érdemesebb adatbázisba szervezni az adatokat, és SQL lekérdezésekkel kinyerni az adatokat.

A QGIS DB Manager

Indítsuk el a *Database* → *DB Manager*t (a Telepített pluginok között kell keresni, ha ott már aktiválva van, a menüsoron található). A *DB Manager* SQL lekérdezések írására alkalmas, vagyis behívhatunk SQL adattípust és bármilyen térinformatikai adatotformátumot.

Ha *Shapefile/Delimited textfile* rétegekkel dolgozunk, akkor a jobb oldali menüben *Virtual Layers* → *Project Layers*nél kell keresünk a rétegeket. Nyissunk egy új SQL ablakot, és írjunk lekérdezéseket. Írassuk ki azokat a megállókat, amelyek nevében szerepel az Örs vezér tere!

1.) Csoportosítsuk az Örs vezér téri megállókat nevük szerint!
2.) Rajzoltassuk is ki a megállók helyét QGIS-ben (Örs vezér téri megállók)!
3.) Mi a Csepel-Királyerdő feliratú járat száma? (*trips* és *routes*, *trip_headsign*)

Ha a rétegekhez nem használunk indexeket és térbeli indexeket, akkor nagyon lassúvá válhatnak a lekérdezések. Éppen ezért innentől használjunk adatbázis kezelőt a további feladatokhoz (DBeaver PostgreSQL és PostGIS adatbáziskezelővel).

A táblákat importálni PostgreSQL-be a következőképpen tudjuk: a legegyszerűbb, ha a *stops* és *shapes* fájlokat SHP-ként mentjük, majd a PostGIS Bundle-lel importáljuk. Shapefile-ok importálását PosGIS Bundle-lel lásd részletesebben a *Térbeli adatbázisok* jegyzetben, dióhéjban ennek a fejezetnek a végén. A többi táblát kétféleképpen importálhatjuk. Az egyik lehetőség, a QGIS-be már behívott táblát (szöveges fájlt pl. *stop_times*) elmentjük DBF formátumban (Jobb egérgombbal kattintás a rétegen → *Export* → *Save as* → *Shapefile*), majd a PostGIS Bundle-lel importáljuk a DBF fájlt. A másik lehetőség a szövegfájl importálása DBeaver-be.

További segítség lehet a lekérdezések gyorsítására, hogy a QGIS-ben indexeket adunk hozzá az egyes rétegekhez.

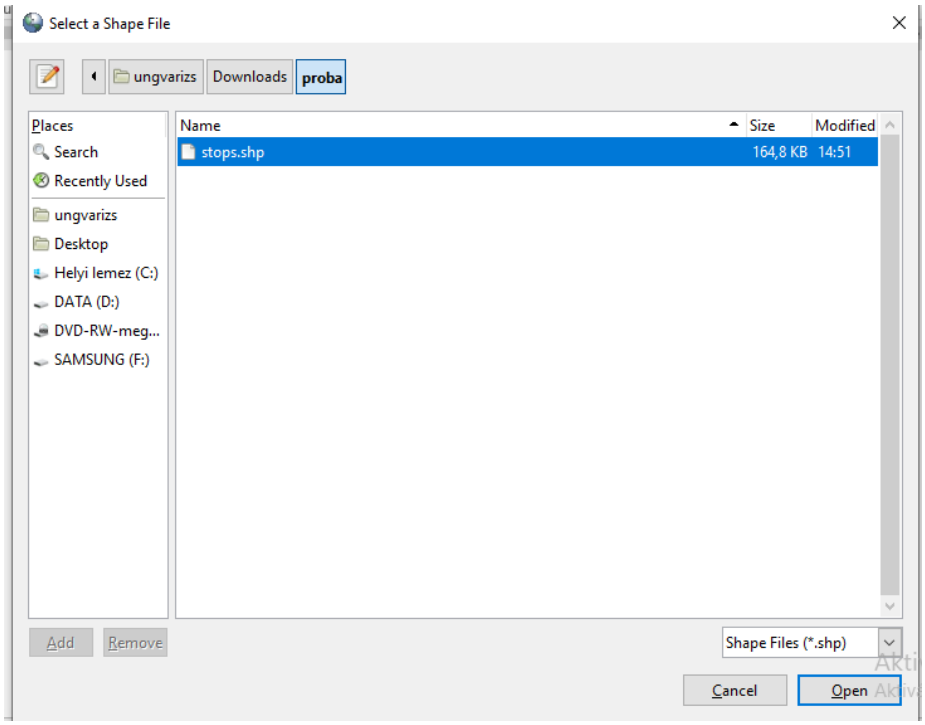
4.) Mely járatok mennek a Göncz Árpád városközpontba (és környékére)? (*trip és routes*). Rendezzük növekvő sorrendbe! /hosszabban futó lekérdezés/
5.) Mikor érkeznek járatok az Egyenes utcai lakótelepre?
6.) Melyek azok a járatok, amelyek megállnak az 'Egyenes utcai lakótelepen'? Listázzuk ki!
7.) Listázza ki a 276E busz indulási idejeit az Egyenes utcai megállóból!
---------- QGIS-ben elvégezhető ----------
8.) Rajzoljuk ki a 276-os busz vonalát egy új rétegre!
10.) Színezzük ki a vonalakat (*shapes path*) a *routes.route_color* alapján!

Megoldások

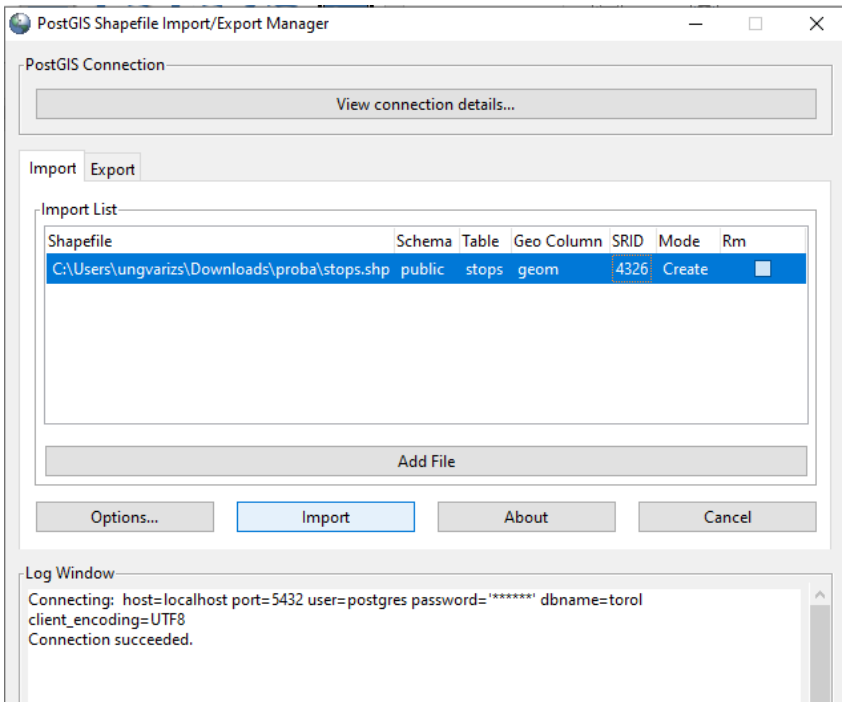
```
1.) select * from stops where stop_name like "Örs vezér tere%"
2.) select stop_name, count(*) from stops where stop_name like "Örs
vezér tere%" group by 1
3.) select stop_name, count(*), geometry from stops where stop_name
like "Örs vezér tere%" group by 1
4.) select * from trips  join routes on
trips.route_id=routes.route_id where trip_headsign='Csepel-
Királyerdő' → D14
5.) select route_short_name from trips  join routes on
trips.route_id=routes.route_id where trip_headsign like 'Göncz
Árpád városközpont%' group by 1 order by 1
6.) select arrival_time from stops join stop_times on
stops.stop_id=stop_times.stop_id where stop_name='Egyenes utcai
lakótelep' order by 1
7.) select distinct routes.route_short_name from stop_times join
stops on stops.stop_id=stop_times.stop_id join trips on
trips.trip_id=stop_times.trip_id join routes on
routes.route_id=trips.route_id where stop_name like 'Egyenes utcai
lakótelep%'
8.) select distinct arrival_time from stop_times join stops on
stops.stop_id=stop_times.stop_id join trips on
trips.trip_id=stop_times.trip_id join routes on
routes.route_id=trips.route_id where stop_name like 'Egyenes utcai
lakótelep%' and route_short_name='276E'
9.) select geometry from shapes_path join trips on
trips.shape_id=shapes_path.shape_id join routes on
routes.route_id=trips.route_id where route_short_name='276E'
majd Load as new layer.
10.) Nem kell lekérdezés. Properties → Joins. Először kapcsoljuk a
trips táblát a shapes_pathoz a shape_id-vel, majd újabb kapcsolat:
ekkor látszik a trips is. trips.route_id és route.route_id
kapcsolása. Végül a Symbology-ban Kategorizáltan osztályzunk.
```

Töltsük fel a rétegeket a PostGIS Bundle program segítségével!

1. Adjuk meg a céladatbázist (ne feledkezzünk meg a PostGIS kiterjesztést hozzárendelni az adatbázishoz). Válasszuk ki az importálandó fájlokat. Ez lehet SHP fájl, de lehetőség van importálni csak a DBF-it. Ilyenkor nem hozza be a geometriát, csak az attribútum táblát.



A SRID-nél adjuk meg a vetületet, majd import. Vigyázzunk, mert a fájlban található a hosszabb mezők neveit levághatja az importálás során!



4. fejezet: A geokódolók használata

Mi az a geokódolás (geocoding)?

Egy megadott címhez vagy földrajzi névhez rendelünk koordinátákat.

Mi az inverz geokódolás (reverse geocoding)?

A megadott koordináta párost azonosítjuk egy címmel.

A geokódolás eredményessége függ egyrészt a mi címadataink pontosságától, és a felhasznált címadatbázis részletességétől. Fontos még megemlíteni, ha részlegesen vagy hiányosan írjuk be a címet, akkor akár több találat is lehet. Ilyenkor a geokódoló a legelső visszatérő címet fogja választani, ami nem biztos, hogy az, amit szerettünk volna. Próbáljuk megkeresni a következő címet OSM Nominatimban vagy a Google adatbázisában:

Budapest Jókai utca 6.

Eredmények:
https://nominatim.openstreetmap.org/ui/search.html?q=Budapest+J%C3%B3kai+utca+6.&exclude_place_ids=71360719%2C173383115%2C24319190%2C259704147%2C226370159

Pontosítsunk: Jókai utca 6., Budapest, XVI. kerület

https://nominatim.openstreetmap.org/ui/search.html?q=+J%C3%B3kai+utca+6.%2C+Budapest%2C+XVI.+ker%C3%BClet

A geokódolásnál a cím megadásához lehetőleg a következő adatok álljanak rendelkezésre: ország, város, postai irányítószám, pontos cím. Az ország megadásával gyorsabban és pontosabban geokódolhatunk.

Elsőként kezdjük a QGIS pluginjeivel. A Plugin Tárházból telepíthetjük őket.

Geocoding

1 db címet lehet egyszerre beadni. pl. Budapest Pázmány Péter sétány 1/a→ válasszuk ki a nekünk kellő találatot a listából. A plugin beállításainál kétféle geokódoló szolgáltatás van: a Nominatim és a Google. A Google-höz API Key szükséges, amit csak bankkártya-adatok megadásával lehet igényelni, ezért ezzel a lehetőséggel itt most nem foglalkozunk.

MMQGIS

Az Oktatási Hivatal honlapjáról töltsük le a Működő köznevelési intézmények listáját (vagy használjuk a jegyzethez mellékeltet). https://dari.oktatas.hu/

Ahhoz, hogy az MMQGIS pluginnel geokódolást futtassunk egy Excel állományon, néhány formai átalakításon kell átesni fájlunknak, hogy egyáltalán importálni tudjuk, illetve, hogy a geokódolás eredményességét növeljük. Adjunk hozzá egy ország mezőt, és töltsük is fel adattal.

A fájlt el kell menteni CSV formátumban (CSV pontosvesszővel tagolt). Majd Notepad++-ban eszközöljük a következő változásokat: UTF-8 kódolásra állítsuk át (Excelből az

alapbeállítás szerint ANSI vagy más néven windows-1250-es kódolással menti el, de az MMQGIS UTF-8-at vár) majd a pontosvesszőt cseréljük át sima vesszőre.

(*Notepad++* → *Keresés* → *Csere*).

Ha ezzel elkészültünk, nyissuk meg az MMQGIS plugint és geokódoljuk az állományt. Próbaként nem muszáj a teljes, több ezer rekordos állományt használni. Vágjuk ki az első száz sort, és csak azt geokódoljuk!

Menüsoron *MMQGIS* → *Geocode* → *Geocode CSV with web service*

Megadjuk a CSV fájlt (az elválasztó karakternek vesszőt vár):

- Address: utca és házszám
- City: város
- State: megye
- Country: ország

Ki kell választani a geokódoló szolgáltatást: OpenStreetMap / Nominatim ez az OSM adatbázisát használja, ezért ingyenes a szolgáltatás.

Lehetőség van a Google és az ESRI geokódolókat is választani, de ezekhez azonosító kulcs szükséges.

Output file name és *not found output list* → az eredmény Shapefile pontokkal és ez CSV lista a kimaradt címekkel (amelyeket nem tud geokódolni, ebbe menti el).

Eredményként megjelennek a geokódolt intézmények pontkoordinátái.

Az OSM Nominatim weboldala

A címadatbázis az OpenStreetMap adatain alapul. Az alább megjelölt oldalon egyszerű és strukturált formában is el lehet küldeni a címeket feldolgozásra.

https://nominatim.openstreetmap.org/ui/search.html

Python geoPy

A feladat további részét Pythonban fogjuk megoldani.

A `geoPy` egy olyan modul, amellyel egyszerűen lehet geokódoltatni és inverz geokódoltatni adatainkat Pythonban. A `geoPy` előnye, hogy egy modulban összegzi a geokódoló szolgáltatásokat (30+) és ezek meghívása is egy-két soros kóddal megoldható. Próbáljuk ki a geokódolást a Nominatim-mal. A feladat a korábban letöltött működő oktatási intézmények Excel tábla geokódolása lesz (egyszerűrég kedvéért ennek első száz rekordja). Mivel most nem CSV-t, hanem Excel táblát olvasunk be, szükségünk van egy modulra, amely segít ennek megvalósításában, ez pedig a `pandas` (a `pandas` alatt az `openpyxl` modulnak is telepítve kell lennie!).

Segítségként itt található a `geoPy` dokumentáció:

https://geopy.readthedocs.io/en/stable/

Python modulok telepítéséről a jegyzet utolsó fejezetében olvashat.

Feladat

Olvassuk be az XLSX fájlt. Geokódoljuk a címeket (intézmény székhelye). Hozzunk létre belőle egy Shapefile-t a következő attribútumokkal: Intézmény neve, és teljes címe egy mezőben. Kerüljenek külön fájlba az óvodák, általános iskolák és gimnáziumok (a többi-t most elhanyagoljuk). A folyamathoz használjuk a `GDAL/OGR` modult, a `pandas`t és a `geoPy`t. Készüljön lista a sikertelen geokódolásokról is.

A feladat részletezése, felépítése:

1. Nyissuk meg és férjünk hozzá az Excel tábla megfelelő oszlopaihoz. Képezzünk egy olyan *string*et a különböző oszlopokban használható címadatokból, hogy az elküldhető legyen a geokódolónak.
2. Küldjük el geokódolásra a címet.
3. Ha van visszaérkező cím, annak koordinátáit írjuk ki egy Shapefile-ba (ezt persze először hozzuk létre).

Importáljuk a szükséges modulokat a Python kód fejlécében. Az osgeo modulból a térbeli osr (vetületeket kezelő könyvtár) és az ogr (vektoros adatokat kezelő könyvtár) szükséges. A Nominatim geokódolót alkalmazzuk.

```
from geopy.geocoders import Nominatim
from osgeo import ogr
from osgeo import osr
import pandas as pd
geolocator = Nominatim(user_agent="None")
```

Olvassuk be az `intézmények100.xlsx` táblát. Töltsük be az adatokat a *DataFrame*-be. A *DataFrame* egy táblázatos adatok tárolására és kezelésére kitalált kétdimenziós, változtatható méretű adatstruktúra. Ezen később akár különféle műveletek is végezhetők.

```
df=pd.read_excel('intezmenyek100.xlsx')
```

Nevezzük át az oszlopokat (*Series*) a fejlécük szerint, vagyis az első sorban található név legyen az oszlop indexe (dobjuk el a jelenlegit, ami az oszlop sorszáma, majd definiáljuk az újat).

```
df.rename(columns=df.iloc[0]).drop(df.index[0])
```

Mielőtt nekiállnánk a geokódoló ciklust definiálni, meg kell adni azt a részt, amellyel a Shapefile-ba írást készítjük elő.

Fájltípus, driver megadása

```
driver = ogr.GetDriverByName("ESRI Shapefile")
```

Hozzunk létre egy Shapefile-t

```
ds = driver.CreateDataSource("intezmenyek.shp")
```

Adjuk meg a vetületet, mivel a geokódolás alkalmával földrajzi koordinátákat kapunk vissza, érdemes négyzetes hengervetületet használni WGS84-es felületen. Ennek az EPSG száma 4326.

```
srs =  osr.SpatialReference()
srs.ImportFromEPSG(4326)
```

Réteg létrehozása, adattípus definiálása.

```
layer = ds.CreateLayer("pontok", srs, ogr.wkbPoint)
```

Hozzuk létre az egyes mezők definícióját az attribútum táblázatba. Pl. mező neve: id, adattípus: egész szám.

```
idField = ogr.FieldDefn("id", ogr.OFTInteger)
layer.CreateField(idField)
featureDefn = layer.GetLayerDefn()
```

Utána következik a ciklus, amellyel geokódoltatom a címeket, majd fájlba iratom. Menjünk végig mint a 100 rekordon. Hozzunk létre egy sztringet a címmel.

A `df.loc[i]['Intézmény települése']` az *i* növelésével kiírható.

```
for i in range(1,99):
     szoveg=' Magyarország, '+df.loc[i]['Intézmény székhelyének
megyéje']+' megye '+df.loc[i]['Intézmény székhelyének
települése']+', '+str(df.loc[i]['Intézmény székhelyének
irányítószáma'])+' '+df.loc[i]['Intézmény székhelyének pontos
címe']+' '
```

Hívjuk meg a geokódolást, majd tegyük bele egy változóba a geokódolt cím visszatérési értékét.

```
    location = geolocator.geocode(szoveg)
```

Írassuk ki a Shapefile-ba a két koordinátapárt, amely az intézmény címét jelöli. Ha van visszatérő koordinátapár, akkor írjuk csak a fájlba, ha nincs írjuk ki a listát.

```
if location!=None:
        feature = ogr.Feature(featureDefn) #létrehozzuk a
geometriai elemet (feature-t)
        pont = ogr.Geometry(ogr.wkbPoint) #megadjuk a geometria
típusát
        pont.AddPoint(location.longitude, location.latitude)
#hozzádjuk a koordinátákat
        feature.SetGeometry(pont) #hozzáadjuk a geometriát az
elemhez
        feature.SetField("id", i) #hozzáadjuk az attribútumokat az
elemhez
        layer.CreateFeature(feature) #hozzáadjuk a réteghez az
elemet
ds = None #lezárjuk a fájlt
```

Szorgalmi/gyakorló feladatok:

- Hogyan lehet azt megoldani, hogy a Python kód felülírja a már meglévő Shapefile-t (mert ha nincs ilyen kódrészlet a programban, akkor nem íródik felül)?

- Írassuk az attribútum táblázatba bele a címet. Külön oszlopba kerüljön az ország, település, postai cím és irányítószám!

5. fejezet: Statisztikai adatok feldolgozása pandas és matplotlib modulokkal: Érettségi adatok

Ebben a fejezetben térvonatkozású statisztikai adatok előkészítését és feldolgozását tekintjük át Python `pandas` modul segítségével. Gyakoroljuk még a diagramkészítést is. A kétszintű érettségi statisztikái elérhetők itt:

https://www.ketszintu.hu/publicstat.php

Válasszuk ki és töltsük például a 2018 őszi közép szintű matematika érettségi eredményeit közlő CSV táblát.
https://www.ketszintu.hu/publicstat.php?stat=_2018_1&reszletes=1&eta_id=3&etj_szint=K

- Számoljunk átlagpontszámot az első és második matematika feladatra, az összpontszámra és az elért jegyre! Csak azok a rekordok szerepeljenek az összegzésben, ahol a vizsgázó megjelent és egyik mezőben sincs – jel.
- Írassuk ki az elért összpontszámok (1.és 2. feladat alapján) megyék szerinti átlagát!
- Rajzoltassuk ki ezt oszlopdiagramba!
- Írjuk ki egy CSV táblázatba ezeket a számokat megyénként.
- Hívjuk be ezt egy QGIS-be, csatoljuk hozzá a megye réteghez, és készítsünk belőle diagramot.

Importáljuk a `pandas` modult, és a *`matplotlib`* függvénykönyvtárból a diagram kirajzolásra alkalmas *`plt`* modult. Olvassuk be a CSV fájlt a *DataFrame*-be (Fájlnév, határoló, karakterkódolás és fejléc megadása).

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('matek_2017_2.csv',
delimiter=";",encoding="cp1250", header=0)  #A DataFramebe tölti az
adatokat
```

Dobjuk el az oszlopok jelenlegi nevét és az első sor alapján nevezzük őket. Így nem az oszlopok sorszámával, hanem a táblázatban lévő oszlopnevekkel tudok hivatkozni az oszlopokra. Emellett hozzunk létre új üres oszlopokat (`pandas` *Series*), egész szám adattípussal az első, a második és az összpontszám mezőkből. Ide csak azokat a pontszámokat fogjuk bemásolni, amelyeknél a vizsgázó részvétele megjelent státuszú és az írásbeli pontszám nem kihúzott (-). Ezáltal a teljes mezőre foguk tudni meghívni a csoportfüggvényeket, nem lesznek üres mezők.

```
df.rename(columns=df.iloc[0]).drop(df.index[0])
df['1.rész']=pd.Series(dtype='int')
df['2.rész']=pd.Series(dtype='int')
df['ossz']=pd.Series(dtype='int')
```

Menjünk végig az érettségi eredmények tábláján és hajtsuk végre az előbb leírtakat. Írjuk ki az átlagos pontszámot, a maximális és minimális pontszámot. A kódban ezen adatok kiíratására többféleképpen is van lehetőség.

```
v=0;
for i in range(1,len(df)-1):
```

```
    if df['vizsgázó részvétele'][i]=='megjelent' and df['írásbeli
pontszám'][i]!='-' :
        v=v+1
        df['1.rész'][i]=int(df['I. rész'][i])
        df['2.rész'][i]=int(df['II. rész'][i])
        df['ossz'][i]=df['1.rész'][i]+df['2.rész'][i]
print(df['ossz'].sum()/v)
print(df['ossz'].aggregate('mean'))
print(df['ossz'].aggregate('max'))
print(df['ossz'].aggregate('min'))
print(df['ossz'].max())
print(df['ossz'].min())
```

A csoportfüggvények (hasonlóan egy SQL-hez) itt úgy is működnek, hogy táblán belül egy másik oszlop alapján lehet kisebb csoportokat is képezni. Például csoportosítsuk az átlagpontszámokat megyénként. Írassuk ki az összpontszámra és az első és második részfeladatra.

```
print (df.groupby('intézmény megyéje')['ossz'].aggregate('mean'))
print (df['1.rész'].sum()/v)
print (df['2.rész'].sum()/v)
```

Rajzoljunk ki megyénként az átlagos pontszámot! Tegyük bele egy változóba. A *groupby*-nál meg kell adni a csoportképzés alapjául szolgáló oszlopot. Utána szögletes zárójelben szerepel az oszlop neve, amelyből számítjuk az átlagokat. Az „*aggregate*” aggregáló függvények típusának megadására alkalmas. A *mean* az átlagot képezi. A *plot* függvénnyel paraméterezhető a diagram, pl. tengelyek címkéi, egységei, színei stb. A kirajzolást pedig a *plt.show()* függvény végzi el.

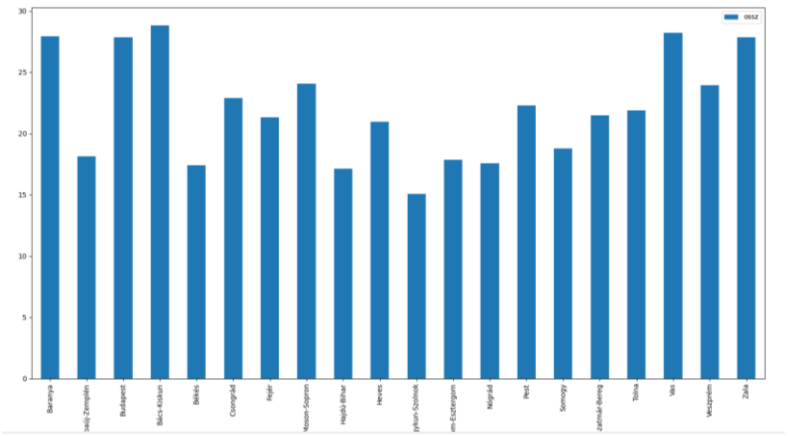
```
df_diagram=df.groupby('intézmény
megyéje')['ossz'].aggregate('mean')
df_diagram.plot(x='intézmény megyéje', y='ossz', kind='bar')
plt.show()
```

Végül írassuk ki egy CSV fájlba megyénként az átlagos pontszámot!

```
df_diagram.to_csv('ujmatek.csv')
```

Ha valaki szeretné fájlba íratni az ábrát, pl. PDF-be, ezen kód hozzáadásával lesz lehetséges. De '`figure.jpg`' fájlnév megadásával akár JPEG formátumú kép is készíthető.

```
fig = df_diagram.plot(x='intézmény megyéje', y='ossz',
kind='bar').get_figure()
fig.savefig('figure.pdf')
```



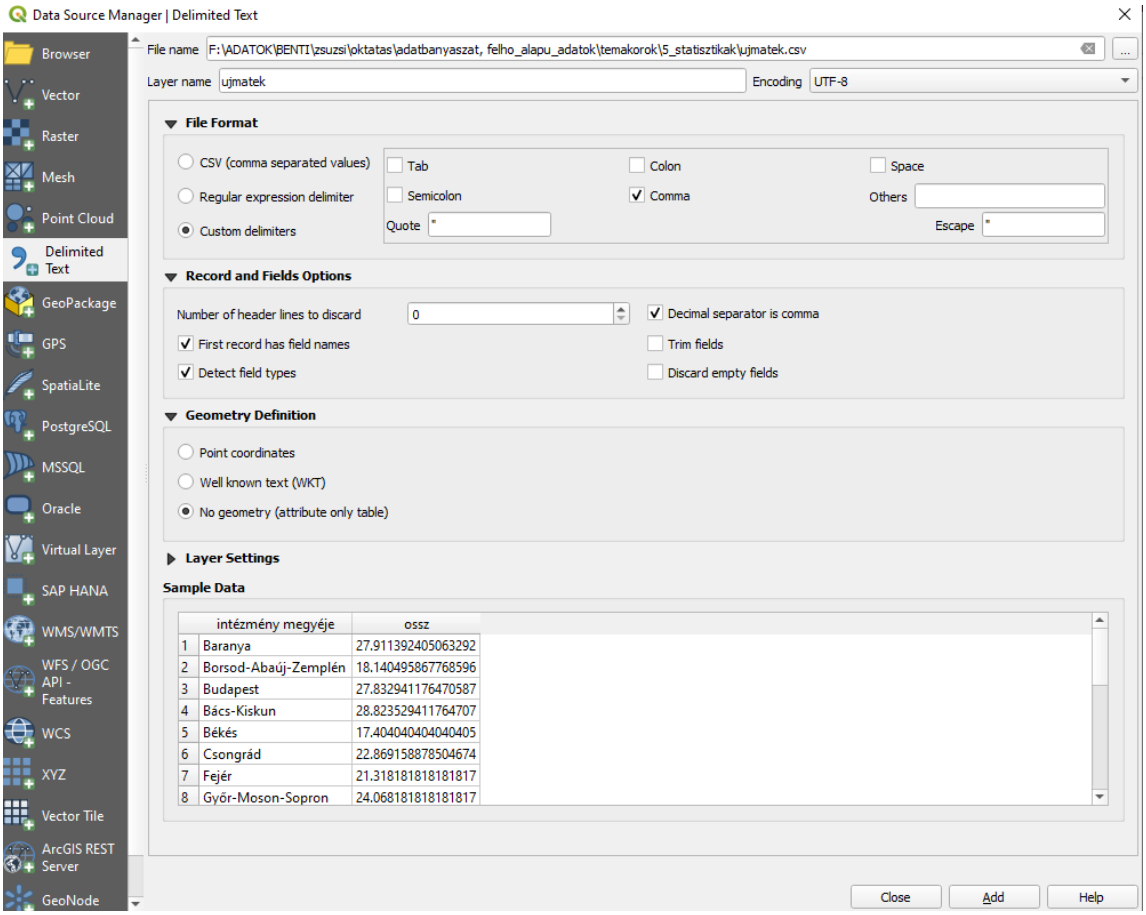
Gyakorló feladat: Ahol van szóbeli pontszám, ott adja hozzá azokat az írásbeli pontszámához (1. és 2. feladat). Így nézze meg, hogy mennyi lesz az új összpontszám.

Rajzoltassa ki diagramon az új átlagos pontszámokat!

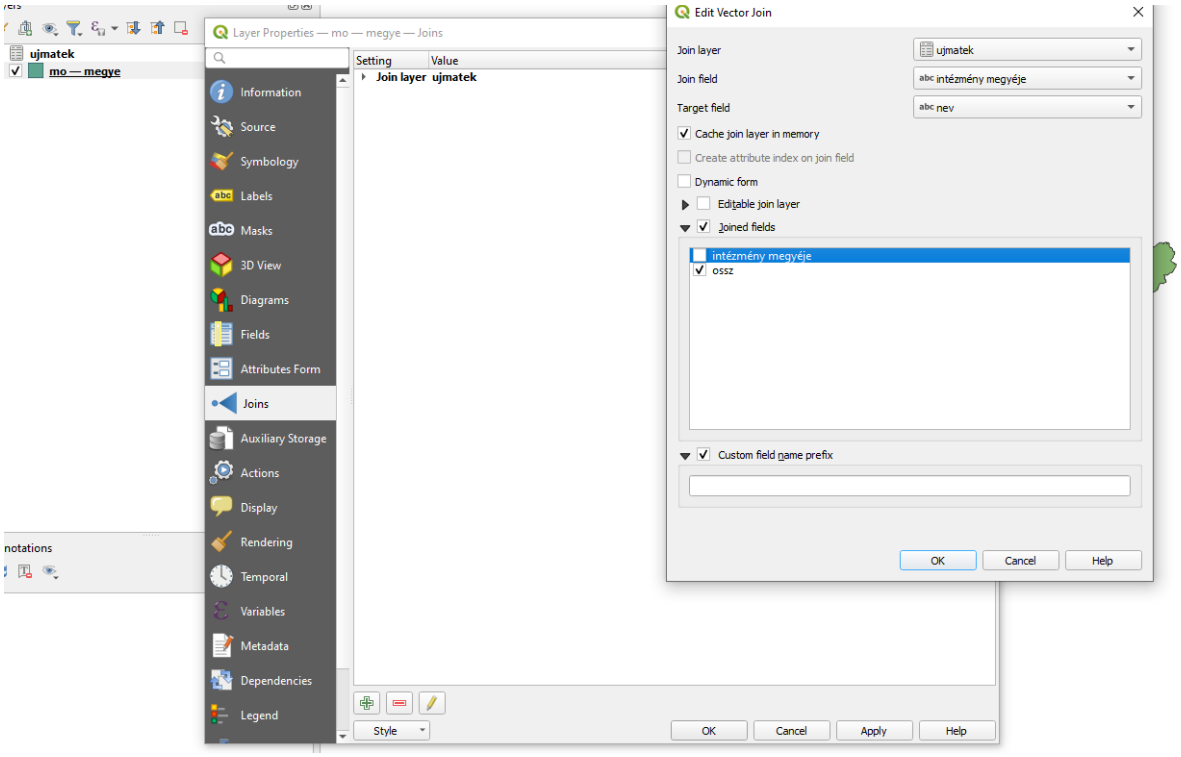
Rajzoltasson egy olyan oszlopdiagramot is, amelyen az írásbeli (1. és 2. rész összege) és a javító szóbeli rész közötti átlagpontszám változásokat mutatja be! (Vagyis mennyivel növekedett megyénként az átlagpontszám?)

Miután megvannak a megyénkénti adatok CSV-ben, készítsünk belőle kartogramtérképet QGIS-ben. Használjuk a mo.gpkg állományból a megye réteget. (Hasonló feladat részletesebb magyarázattal olvasható a QGIS feladatgyűjteményben: http://mercator.elte.hu/~ungvarizs/).

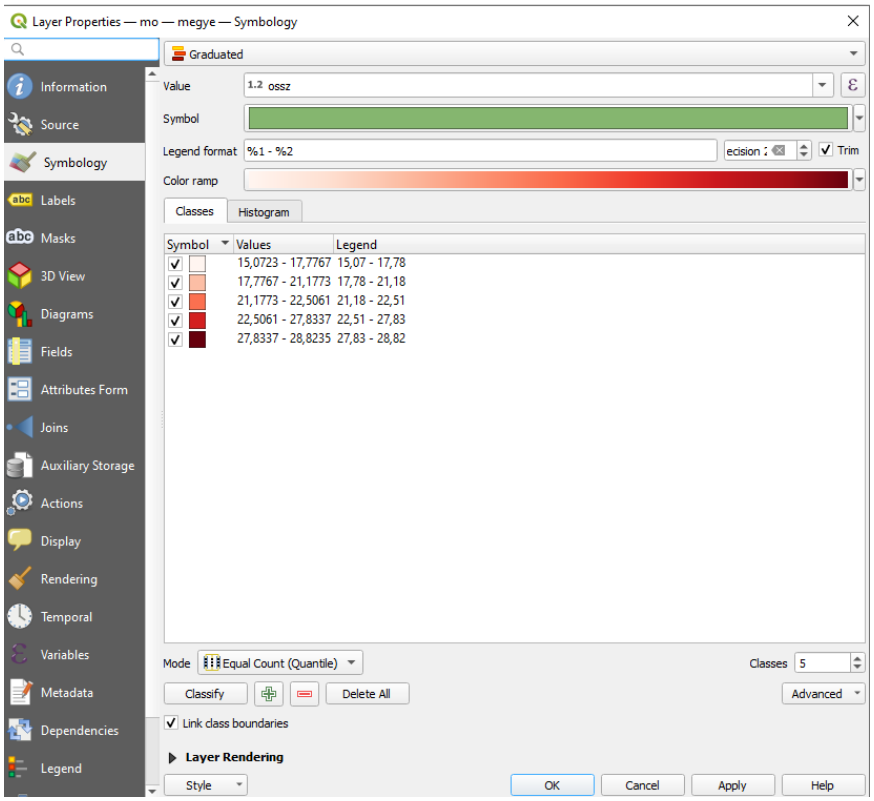
A CSV fájlt *Delimited Text File*-ként hívjuk be a következő beállításokkal:

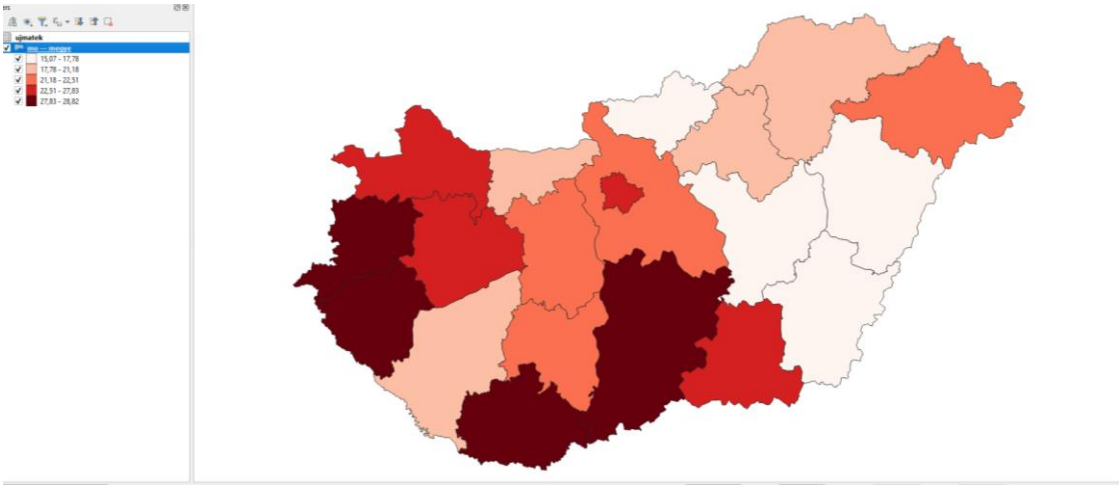


Majd hozzákapcsoljuk a megye réteghez a megye neve alapján.



Végül színezzük ki a kartogramtérképet! Ehhez most az *Equal Count* módszert használtam.




sijmatek
15,07 - 17,78
17,78 - 21,18
21,18 - 22,51
22,51 - 27,83
27,83 - 28,82

6. fejezet: Statisztikai adatok feldolgozása pandas és matplotlib modulokkal 2: Vízállás adatok

A Hydroinfo.hu weboldalon többek között folyóink és tavaink vízállásával, jégborítottságával kapcsolatos információkat találunk. https://www.hydroinfo.hu/

A honlap archívuma a Központi Hidrológiai Adattár néven elérhető, és HTML formátumban jeleníti meg az egyes vízmércéken mért vízállásokat. Ezekből fogunk most letölteni, és a vízállásokból diagramot készítünk.

A Balatonnak az 2000-es évek elején volt egy jelentős vízszint csökkenése, amely később aztán rendeződött. Ezeket az éveket szeretnénk bemutatni oszlopdiagram segítségével.

Az adattárban válasszuk a Balaton átlag vízmércét, és a szöveges mezőbe írjuk be az évet. Töltsük le az adatokat, és mentsük el egy szöveges fájlba (2000-2004 között).

Észlelt VÍZÁLLÁSOK Évszám: 2000

adatok jégkóddal /interpolációval/ [cm] Kinyomtatva: 2004-Jún-24 11:42 Időpont: 7:00 ń 3:00 KEI

Állomás kód: 142300 Vízgyüjtö terület: 5774.0 km2
Állomás név: BALATON ATLAG Távolság a torkolattól:
Vízfolyás: BALATON A nullpont magassága:
Érvényes: 2000-Dec-31

Nap	Jan	Feb	Már	Ápr	Máj	Jún	Júl	Aug	Sze	Okt	Nov	Dec
1	110	102	102	111	104	98	85	78	64	59	60	62
2	111	102	102	111	105	98	85	78	65	60	61	62
3	111	101	103	111	104	97	85	78	65	60	60	62
4	111	101	103	111	104	97	84	78	65	60	60	62
5	111	101	104	111	104	97	83	77	64	60	61	62
6	111	101	104	111	104	97	83	76	64	60	61	62
7	110	101	104	111	103	96	83	76	63	59	61	62
8	110	100	104	111	103	96	82	75	63	59	62	62
9	110	100	104	110	103	96	82	74	62	59	62	62
10	110	101	105	109	103	95	82	74	62	59	62	62

Ahhoz, hogy az adatsort minél rövidebb és egyszerűbb programkóddal tudjuk feldolgozni, csak a *'Nap Jan Feb'* stb. sortól kezdve mentettem le szöveges fájlként az adatokat.

Jobban megvizsgálva az így lementett TXT fájlt Notepad++-ban, a mért adatok látszólag tabulátorral vannak elválasztva. Sajnos ez a tabulátor nem mindenhol ugyanolyan hosszú, valahol több szóköz egymás mögött. Éppen ezért előfeldolgozásként cseréljük ezeket a

tabokat, vagy halmozott szóközöket pl. vesszőre vagy pontosvesszőre. *Notepad++* → *Keresés* → *Csere*

Az üres sorokat töröljük. Végül valami ilyesmit fogunk kapni az adatsor tisztítása után:

vizallas_balaton_2000.txt – Jegyzettömb

Fájl Szerkesztés Formátum Nézet Súgó

```
Semmi,Nap,Jan,Feb,Már,Ápr,Máj,Jún,Júl,Aug,Sze,Okt,Nov,Dec
,1,110,102,102,111,104,98,85,78,64,59,60,62
,2,111,102,102,111,105,98,85,78,65,60,61,62
,3,111,101,103,111,104,97,85,78,65,60,60,62
,4,111,101,103,111,104,97,84,78,65,60,60,62
,5,111,101,104,111,104,97,83,77,64,60,61,62
,6,111,101,104,111,104,97,83,76,64,60,61,62
,7,110,101,104,111,103,96,83,76,63,59,61,62
,8,110,100,104,111,103,96,82,75,63,59,62,62
,9,110,100,104,110,103,96,82,74,62,59,62,62
,10,110,101,105,109,103,95,82,74,62,59,62,62
,11,110,101,105,108,103,95,81,74,62,60,62,63
,12,110,100,106,109,103,95,81,74,62,61,62,63
,13,109,100,106,109,102,95,81,73,62,60,62,63
,14,109,100,106,109,102,95,81,73,62,60,62,63
,15,108,100,106,109,102,95,80,73,62,60,62,63
,16,108,100,106,109,101,94,81,73,62,59,62,63
,17,108,100,106,109,101,92,81,72,63,59,61,63
.18.108.100.107.109.101.91.81.71.63.59.61.64
```

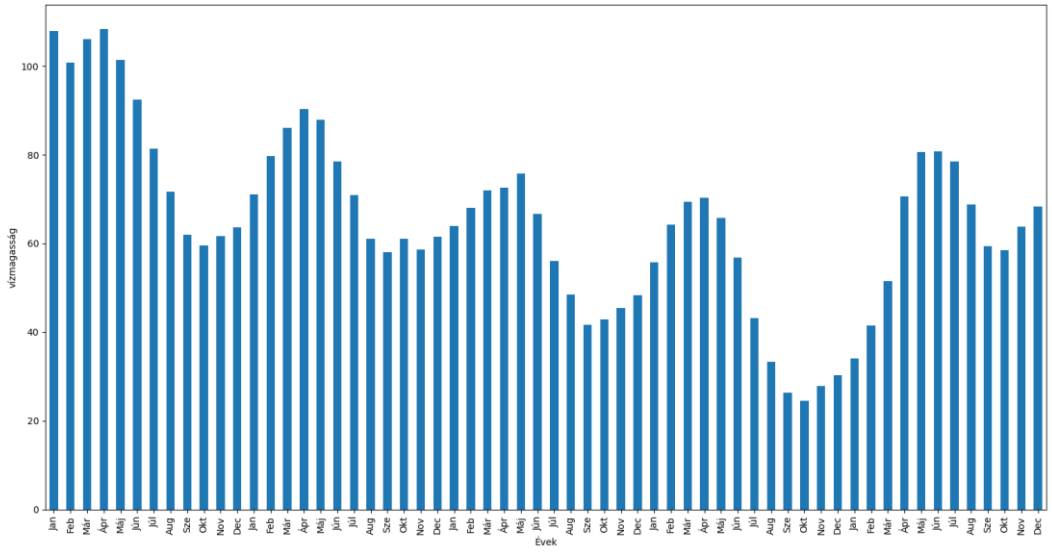
Sor: 1, oszl.: 1 120% Windows (CRLF) UTF-8

Az átalakításnál fontos az, hogy végül minden cella ugyanúgy nézzen ki (tehát ne maradjanak benne felesleges szóközök), minden sornak ugyanannyi eleme legyen (ha üres a cella pl. a hónapok 31. napján, ott két vessző követi egymást). A fejlécben lévő adatok számossága megegyezik a többi soréval.

Feladat

Készítsünk egy oszlopdiagramot, amely a havi átlagos vízállásokat írja ki. Dolgozzuk fel a 2000-2004 közötti időszakot.

Figure 1



A megvalósítás lépési

Töltsük le a Hidrológia Adattár archívumából az adatokat, és tisztítsuk le (`vizallas_balaton_2000.txt` stb).

- Importáljuk a `pandas`-t és a `matplotlib`-et.
- A TXT fájlokban az elválasztó vessző, kódolás UTF-8, és a táblázat fejléce (*head*er) a 0. sor.
- Töröljük az üres oszlopokat.

```
import pandas
import pandas as pd

#2000-2004
df0=pd.read_csv('vizallas_balaton_2000.txt',
delimiter=",",encoding="utf8", header=0)

df1=pd.read_csv('vizallas_balaton_2001.txt',
delimiter=",",encoding="utf8", header=0)
df2=pd.read_csv('vizallas_balaton_2002.txt',
delimiter=",",encoding="utf8", header=0)
df3=pd.read_csv('vizallas_balaton_2003.txt',
delimiter=",",encoding="utf8", header=0)
df4=pd.read_csv('vizallas_balaton_2004.txt',
delimiter=",",encoding="utf8", header=0)
del df0['Semmi']
del df0['Nap']
del df1['Semmi']
del df1['Nap']
del df2['Semmi']
del df2['Nap']
del df3['Semmi']
del df3['Nap']
del df4['Semmi']
del df4['Nap']
```

Aggregáljuk az egyes fájlokat a hónapok szerint (számítsuk ki a havi átlagot).

```
df_diagram0=df0.aggregate('mean')
df_diagram1=df1.aggregate('mean')
df_diagram2=df2.aggregate('mean')
df_diagram3=df3.aggregate('mean')
df_diagram4=df4.aggregate('mean')
```

Egy print utasítással megnézhető az eredmény:

```
print (df_diagram0)
```

```
>>
Jan    107.967742
Feb    100.724138
Már    106.129032
Ápr    108.400000
Máj    101.419355
Jún     92.400000
Júl     81.322581
Aug     71.741935
```

```
Sze      62.033333
Okt      59.548387
Nov      61.600000
Dec      63.677419
dtype: float64
```

Majd fűzzük össze ezeket az adatokat egy tömbbe. Egyesítsük őket egy *pd.concat()* függvénnyel azért, hogy megszűnjenek a tömbök, és folytonos szöveget kapjunk. Ezáltal a diagramok is folytonosak lesznek. Végül rajzoltassuk ki a diagramot. Lásd a feladatnál.

```
sumdf1=[df_diagram0,df_diagram1,df_diagram2,df_diagram3,df_diagram4
]
result=pd.concat(sumdf1)
result.plot(xlabel='Évek', ylabel='vízmagasság',kind='bar')
plt.show()
```

7. fejezet: Flickr fotómegosztó oldalról letöltött metaadatok feldolgozása

Kötelező olvasmány a lecke elvégzése előtt:

Gede Mátyás: Fényképek térképei – geotaggelt fotók térbeli eloszlásának térképes vizsgálata.

http://geodezia.elte.hu/sites/default/files/sites/dokumentumok/feltoltes/admin/01_02.pdf

Feladat

Töltsük le egy kijelölt területen készült fotók metaadatait a Flickr fotómegosztó oldal adatbázisából. Ezek tartalmazzák a fénykép készítésének helyét, idejét stb. Alakítsunk ki a területre egy hálózatot, és számoljuk meg, hogy az egyes cellákban hány fotó van. Jelenítsük meg Google Earth-ön a fotók számát oszlopdiagramként 3D-ben. Vagyis ha az adott cellában több fotó van, akkor az oszlop magasabb lesz, ha nincs fotó, ne kerüljön fel oszlop.

A fotók metaadatainak letöltése

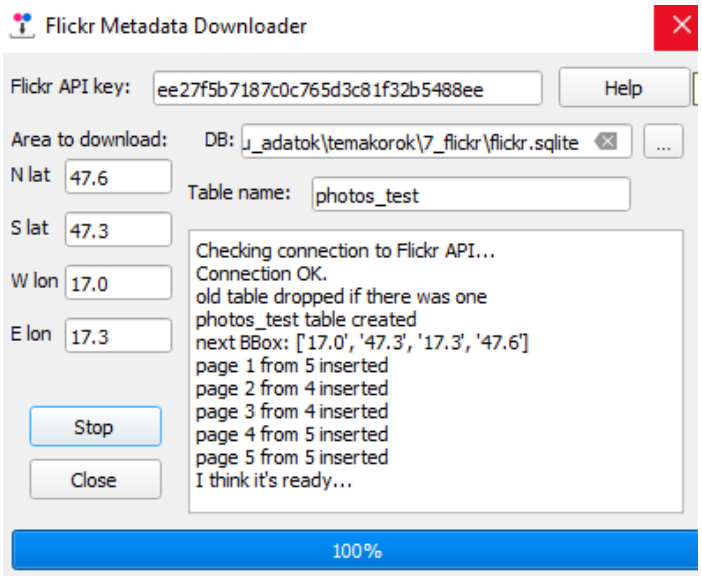
QGIS → *Plugins* → *Manage and Install Plugins* → Telepítsük a *Flickr Metadata Downloader* (FMD).

Adatok letöltése

Hozzunk létre egy sqlite formátumú térbeli adatbázis fájlt (*Layer* → *Create Layer* → *New SpatiaLite layer*). Megadjuk a fájlnevet, a réteg nevét és pont geometriát készítünk. *Autoincrementing primary key*. Bár nem erre a rétegre írunk, de létre kell hozni egy fájlt, mert a FMD-vel ezt nem tudjuk megcsinálni. Ebbe a fájlba kerülnek majd a fotók adatai egy másik rétegen (táblában).

Indítsuk el az FMD-t.

Adjuk meg a saját felhasználói azonosítónkat a Flickr adatbázishoz, ez a honlapjukon regisztrációval készíthető. Válasszuk ki az sqlite adatbázis fájlt, a tábla/réteg nevét, és adjuk meg a terület határoló koordinátáit is.



Letöltés után a *photo_test layer* behívható, és a rétegek lévő adatok pontokként jelennek meg. Földrajzi koordinátáink lesznek. Az attribútum táblázatot is tanulmányozzuk!

Generáljunk egy gridet, számoljuk meg cellákban lévő fotókat

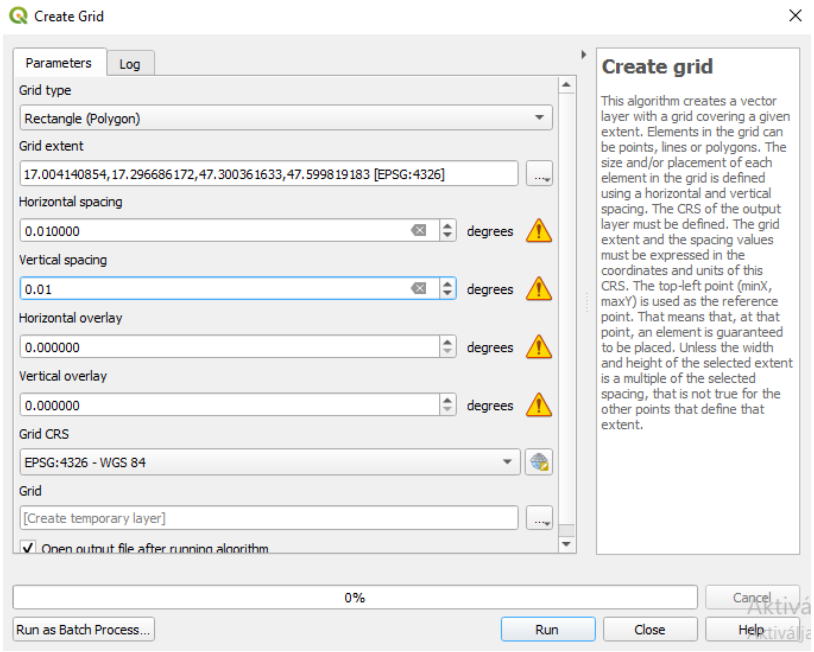
Készítsünk egy gridet a *Create grid* eszközzel (*Processing*). A cellák mérete opcionális, igyekezzünk a területen lévő fotók sűrűségéhez igazítani a cellaméretet. Értelemszerűen pl. Budapest belvárosában kisebb cellaméretet érdemes használni, míg a képen ábrázolt nyugat-magyarországi területen nagyobb cellákkal dolgoztam, mivel kevésbé népszerű desztináció.

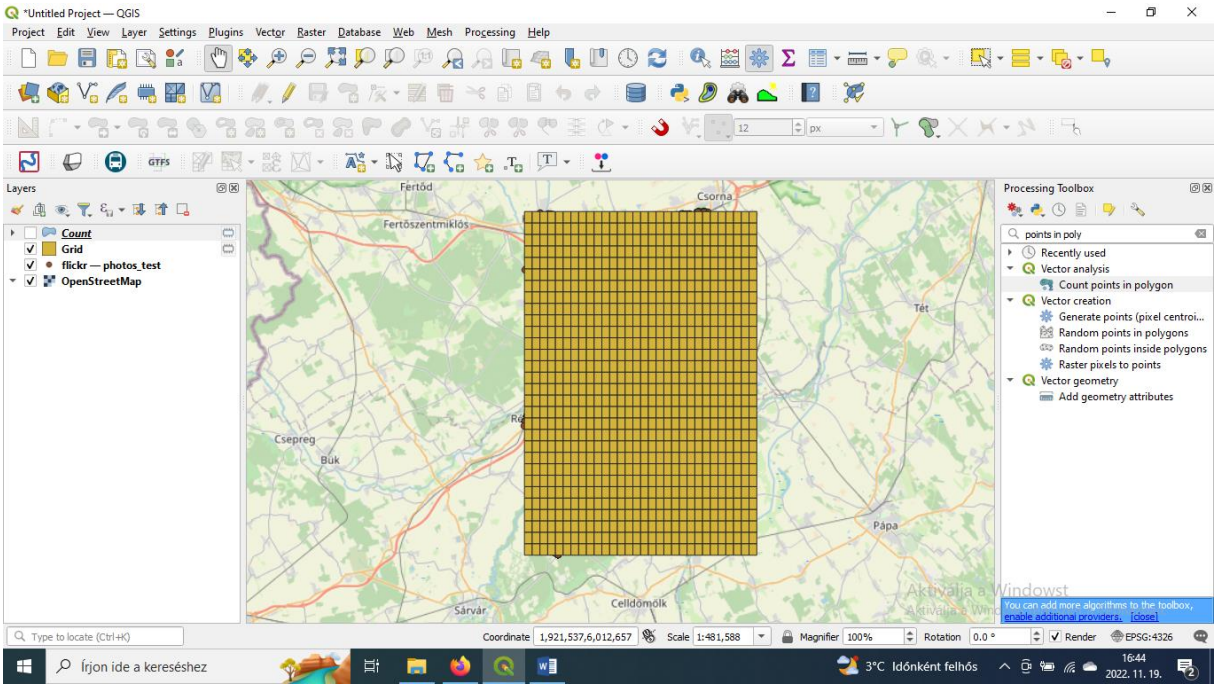
A grid *Rectangle* (*Polygon*), a réteg a kiterjedését (*grid extent*) örökölheti a fotók rétegből. A *horizontal* és *vertical spacing*nél megadjuk a cellák méretét. Ügyeljünk a vetületre. Itt földrajzi koordinátákkal dolgoztam, ezért a 4326-os földrajzi koordinátákkal dolgozó négyzetes hengervetületet használom.

Segítség a cellaméret közelítő becsléséhez:

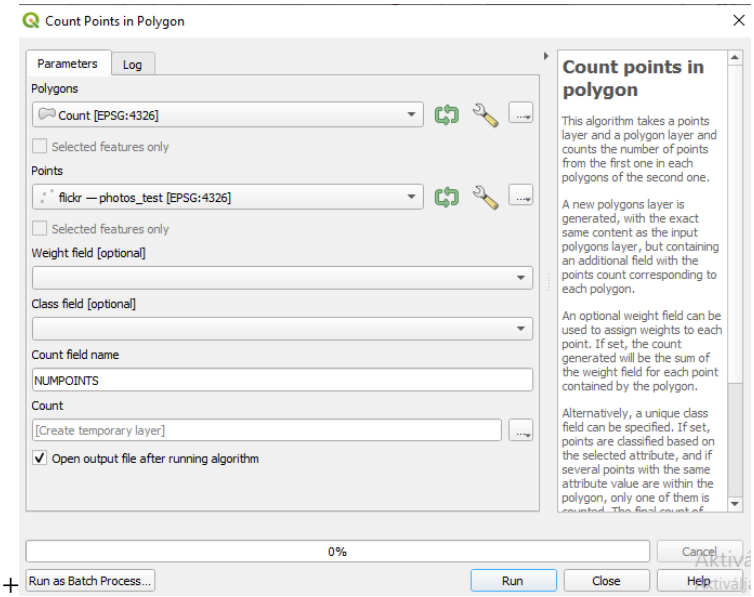
1°~> 111.1 km 0.01°~> 1.1 km

0.1°~> 11.1 km 0.001°~> 111 m





Ezután számoljuk meg a pontokat a cellákban: *Count points in polygon*. Az eszköz beleírja az attribútum táblázatba a cellák azonosítója mellé a képek számosságát. Írjuk Shapefileba az eredményt, azt könnyen olvashatjuk Pythonnal.



Pythonban írjunk egy programot: olvassuk be az shapefile-t, ha a *NUMPOINTS* oszlop nem 0, rajzoljunk egy négyszög alapterületű oszlopot, a benne lévő értéknek megfelelően.

A feladat megoldásához az OSGEO GDAL/OGR modult fogom használni. A 4. fejezetben már használtuk ezt a modult.

```
from osgeo import ogr
from osgeo import osr
```

A KML fájlt a hagyományos módon fogom elkészíteni, vagyis szöveges fájlként fogjuk megírni. Ehhez létre kell hozni a fájlt, előkészíteni a fejlécet, stílusokat megadni.

A fényképek számát bemutató oszlopok magasságuk szerint háromféle színűek lehetnek. Ehhez definiálok már most egy *colors* listát a hexadecimálisan megadott színkódokkal. Ezekből kitöltési stílust készítek.

A KML fájlok szerkezetének ismeretét feltételezem a programkód további részében, így ehhez nem fűznék hozzá további magyarázatot. Ha nem ismeri az olvasó, a jegyzet végén talál erről egy összefoglalót.

```
colors=['ff00ffff', 'ff0000ff', 'ff336699']
f = open("pelda.kml", "w")
f.write('<?xml version="1.0" encoding="utf-8" ?><kml
xmlns="http://earth.google.com/kml/2.2"><Document> <name>Pontok a
polyban</name>')
for i in range(0, len(colors)):
    f.write('<Style id="PolyColor'+str(i)+'"> \n ')
    f.write(' <LineStyle> \n ')
    f.write('<width>1.5</width> \n ')
    f.write('</LineStyle> \n ')
    f.write('<PolyStyle>')
    f.write('<color>'+colors[i]+'</color> \n ')
    f.write(' </PolyStyle> \n ')
    f.write(' </Style> \n ')
```

Definiáljuk a drivert, nyissuk meg olvasásra a fájlt, kapjuk el a réteget, és számoljuk meg a rétegen lévő elemeket.

```
driver = ogr.GetDriverByName("ESRI Shapefile")
dataSource = driver.Open('pointsingrid.shp', 0)
layer = dataSource.GetLayer()
featureCount = layer.GetFeatureCount()
```

Menjünk végig a réteg elemein. Ha a *NUMPOINTS* attribútum értéke nem 0, akkor foglalkozni kell az elemmel (oszlopot kell létrehozni).

Írjuk meg KML-ben a *Placemarkot*, adjuk hozzá a fényképek számossága alapján a három stílus egyikét (ezt a beosztást a jelen példa alapján készítettem, értelemszerűen más is lehet más területen). Van benne egy konstans 20-as szorzó. Mivel ezen a területen nincs túl sok fénykép, ezt túlmagasításként alkalmazom (*vertical exaggeration*). Más számok esetén természetesen átírható/elhagyható.

```
for feature in layer:
    geom = feature.GetGeometryRef()
    numPoints=feature.GetField("NUMPOINTS")
    if numPoints!=0:

        f.write('<Placemark>')
        if numPoints*20<=20:
            f.write('<styleUrl>#PolyColor0</styleUrl> ')
        elif numPoints*20>20 and numPoints*20<=100:
            f.write('<styleUrl>#PolyColor1</styleUrl> ')
```

```
        elif numPoints*20>100:
            f.write('<styleUrl>#PolyColor2</styleUrl> ')
        else:
            pass
```

Adjuk hozzá a *Placema*rk lényegi részét, a geometriát.

A poligon térbeli test, össze van kötve a felszínnel, így az *extrude* értéke 1.

Mivel egyszerű négyszög alakú területeink vannak (egy gyűrűből álló egyszerű poligon=a poligon körvonala), ezért amikor a poligonok geometriáját akarjuk kiolvasni a Shapefile-ból és menteni KML-be, elég a ring változó töréspontjain végig futni.

A gyűrű töréspontjait a *GetPointCount()* függvénnyel számoltatom le.

A `ring.GetPoint(p)[0]` és `ring.GetPoint(p)[1]` a pont két koordinátáját olvassa ki. A KML fájlba íráskor adjuk meg túlmagasítással együttesen az oszlop Z koordinátáját.

```
        f.write('<Polygon><extrude>1</extrude><tessellate>1
</tessellate><altitudeMode>relativeToGround
     </altitudeMode><outerBoundaryIs><LinearRing><coordinates>')
        geom=feature.GetGeometryRef()
        ring=geom.GetGeometryRef(0)
        points=ring.GetPointCount()

        for p in range(0,points):
            lon= ring.GetPoint(p)[0]
            lat= ring.GetPoint(p)[1]
            f.write(str(lon)+','+str(lat)+','+str(numPoints*20)+'
\n')

f.write('</coordinates></LinearRing></outerBoundaryIs></Polygon></P
lacemark>')
```

Lezárjuk a fájlokat.

```
f.write('</Document></kml>')
f.close()
dataSource.Destroy()
```

8. fejezet Vektoros adatok olvasása és írása a Python GDAL/OGR (OSGEO) modulban

Ebben a fejezetben részletesebben megismerkedünk a GDAL vektoros adatok feldolgozására szolgáló moduljával, az OGR-rel. Azért tartom fontosnak az alapokat átvenni, mert szükség lehet arra, hogy saját Python programot kell fejleszteni, amely olyan dolgokat hajt végre, amelyeket egy térinformatikai szoftverben nem, vagy csak részben vagyunk képesek megcsinálni. Ehhez jó alapnak szolgál a legegyszerűbb Shapefile-ok beolvasása/írása és néhány geometriai vagy *geoprocessing* függvény meghívása.

Ajánlott irodalmak a témában:

GDAL/OGR Cookbook

https://pcjericks.github.io/py-gdalogr-cookbook/index.html

GDAL Documentation

https://gdal.org/

Python OGR

https://gdal.org/api/python/osgeo.ogr.html

A fejezet első felében vektoros adatok beolvasását fogom programkódokkal illusztrálni. A legfontosabb adattípusokra lesz látható egy-egy példa. Egészen a koordinátákhoz való hozzáférésig adom meg. A példákban Shapefile-okkal fogok dolgozni.

A fájlbeolvasás mindig ugyanaz, adattípustól függetlenül. Az OSGEO modulból meghívjuk az OGR könyvtárat. Definiáljuk a *driver*t, ami a fájltípus beolvasásáért felelős.

Megnyitjuk a fájl-t olvasásra, majd hozzáférünk a réteghez. Végül nézzük meg a réteg elemszámát.

```
from osgeo import ogr
driver = ogr.GetDriverByName("ESRI Shapefile")
dataSource = driver.Open('grid.shp', 0)
layer = dataSource.GetLayer()
featureCount = layer.GetFeatureCount()
```

Pontok (Point) koordinátáinak beolvasása

Végig megyünk a réteg elemein. A *GetGeometryRef()* függvénnyel férjünk hozzá a geometriához, tegyük egy *geom* nevű változóba. Olvassuk ki a *GetField('xxx')* függvénnyel az egyik mezőben található adatokat. Ezután a *GetX()* és *GetY()* függvénnyel közvetlenül lekérhetjük a két koordinátát.

```
for feature in layer:
    geom = feature.GetGeometryRef()
    field1=feature.GetField("p_id")
    lon= geom.GetX()
    lat= geom.GetY()
```

Vonal (LineString) adattípus beolvasása

Végig megyünk a réteg elemein. A *GetGeometryRef()* függvénnyel férjünk hozzá a geometriához, tegyük egy geom változóba. Olvassuk ki a *GetField('xxx')* függvénnyel az egyik mezőben található adatokat. Számoljuk ki, hány töréspontja van a vonalnak. Menjünk végig rajta és írjuk ki a koordinátáit.

```
for feature in layer:
    geom = feature.GetGeometryRef()
    field1=feature.GetField("id")
    points=geom.GetPointCount()
    for p in range(0,points):
        lon= geom.GetPoint(p)[0]
        lat= geom.GetPoint(p)[1]
```

Felület (Polygon) beolvasása

Végig megyünk a réteg elemein. A *GetGeometryRef()* függvénnyel férjünk hozzá a geometriához, tegyük egy geom változóba. Olvassuk ki a *GetField('xxx')* függvénnyel az egyik mezőben található adatokat. Nézzük meg a poligon gyűrűinek számát (*GetGeometryCount()*), menjünk végig egyesével a gyűrűkön. Vegyük a gyűrű töréspontjait, majd írassuk ki azok hosszúságát és szélességét.

```
for feature in layer:
    geom = feature.GetGeometryRef()
    field1=feature.GetField("id")
    for i in range(0,geom.GetGeometryCount()):
        ring=geom.GetGeometryRef(i)
        points=ring.GetPointCount()
        for p in range(0,points):
            lon= ring.GetPoint(p)[0]
            lat= ring.GetPoint(p)[1]
```

Pontcsoport (MultiPoint) típusú elemek koordinátáinak beolvasása

Végig megyünk a réteg elemein. A *GetGeometryRef()* függvénnyel férjünk hozzá a geometriához, tegyük egy geom változóba. Olvassuk ki a *GetField('xxx')* függvénnyel az egyik mezőben található adatokat. Számoljuk meg, hogy hány eleme van a geometriának, majd menjünk ezeken végig. Az egyes részgeometriákból pedig ezután a *GetX()* és *GetY()* függvénnyel közvetlenül lekérhetjük a két koordinátát.

```
for feature in layer:
    geom = feature.GetGeometryRef()
    field1=feature.GetField("id")
    numgeom=geom.GetGeometryCount()
    print (numgeom)
    for i in range(0,numgeom):
        lon=geom.GetGeometryRef(i).GetX()
        lat=geom.GetGeometryRef(i).GetY()
```

Vonalcsoport (MultiLineString) elemek koordinátájának beolvasása

Végig megyünk a réteg elemein. A *GetGeometryRef()* függvénnyel férjünk hozzá a geometriához, tegyük egy geom változóba. Olvassuk ki a *GetField('xxx')* függvénnyel az egyik mezőben található adatokat. Számoljuk meg, hogy hány eleme van a geometriának, majd menjünk ezeken végig. Számoljuk ki, hány töréspontja van a vonalnak. Menjünk végig rajta és írjuk ki a koordinátáit.

```
 for feature in layer:
    geom = feature.GetGeometryRef()
    field1=feature.GetField("id")
    numgeom=geom.GetGeometryCount()
    for i in range(0,numgeom):
        line=geom.GetGeometryRef(i)
        points=line.GetPointCount()
        for p in range(0,points):
            lon= line.GetPoint(p)[0]
            lat= line.GetPoint(p)[1]
```

Felület (Polygon) és felületcsoport (MultiPolygon) elemek koordinátájának beolvasása

Végig megyünk a réteg elemein. A *GetGeometryRef()* függvénnyel férjünk hozzá a geometriához, tegyük egy geom változóba. Olvassuk ki a *GetField('xxx'*) függvénnyel az egyik mezőben található adatokat. Számoljuk meg, hogy hány eleme van a geometriának, majd menjünk ezeken végig. Nézzük meg a poligon gyűrűinek számát (*GetGeometryCount()*), menjünk végig egyesével a gyűrűkön. Vegyük a gyűrű töréspontjait, majd írassuk ki azok hosszúságát és szélességét.

Azokban az esetekben, ahol a fájlban vegyesen található poligon és multipoligon, szükség lesz egy elágazásra, amely a geometria számossága alapján kettéválasztja a kétféle elemet. (Pont és pontcsoport, illetve vonal és vonalcsoportot vegyesen tartalmazó fájloknál ugyanígy működik).

```
for feature in layer:
    geom = feature.GetGeometryRef()
    field1=feature.GetField("id")
    numgeom=geom.GetGeometryCount()
    if numgeom>1:
        for ng in range(0,numgeom):
            poly=geom.GetGeometryRef(ng)
            for i in range(0,poly.GetGeometryCount()):
                ring=poly.GetGeometryRef(i)
                points=ring.GetPointCount()
                for p in range(0,points):
                    lon= ring.GetPoint(p)[0]
                    lat= ring.GetPoint(p)[1]
    else:
        for i in range(0,geom.GetGeometryCount()):
            ring=geom.GetGeometryRef(i)
            points=ring.GetPointCount()

            for p in range(0,points):
                lon= ring.GetPoint(p)[0]
                lat= ring.GetPoint(p)[1]
```

Shapefile-ok írása

Ebben a fejezetrészben a pontok, vonalak és felületek (egyszerű elemek) fájlba írásával fogunk foglalkozni.

Pontok

Olvassunk be a points.shp pontokat tartalmazó Shapefile-t, majd attribútum adatok alapján válogassuk le bizonyos elemeket és a feltételnek megfelelő pontokat írjuk ki egy új Shapefile-ba.

A program bizonyos részei megegyeznek a pontok fájlbeolvasásánál leírtakkal. Az srs változóban megadhatjuk a vetületet. Hozzunk létre egy új adatforrást a már megadott (Shapefile) *driver*rel. Adjuk meg a réteg adattípusát (*ogr.wkbPoint, ogr.wkbLineString, ogr.wkbPolygon* stb.). Hozzunk létre az attribútum táblázatot most csak egy oszloppal. A lehetséges adattípusok (*OFTInteger, OFTString, OFTDateTime* stb.)

Ezen adattípusokról a https://gdal.org/java/org/gdal/ogr/ogrConstants.html linken lehet olvasni.

```
from osgeo import ogr
from osgeo import osr
driver = ogr.GetDriverByName("ESRI Shapefile")
dataSource = driver.Open('points.shp', 0)
layer = dataSource.GetLayer()
featureCount = layer.GetFeatureCount()

srs =  osr.SpatialReference()
srs.ImportFromEPSG(4326)
ds = driver.CreateDataSource("selected_points.shp")
layeruj = ds.CreateLayer("pontok", srs, ogr.wkbPoint)
dateField = ogr.FieldDefn("datum", ogr.OFTDateTime)
layeruj.CreateField(dateField)
featureDefn = layeruj.GetLayerDefn()
```

Szűrjük le az adatokat egy elágazással: azok a pontok fognak majd szerepelni az új fájlban, amelyek készítési dátuma 2010-01-01 utáni. Hozzunk létre egy új *feature*-t (elemet). Hozzunk létre egy új pontot. Adjuk meg a pont koordinátáit. A *feature*-höz adjuk hozzá a létrehozott pontot, majd vigyük át a dátumot is, és adjuk a réteghez. Végül zárjuk be a fájlt.

```
for feature in layer:
    geom = feature.GetGeometryRef()
    field1=feature.GetField("p_date")
    if field1>'2010-01-01':
        lon= geom.GetX()
        lat= geom.GetY()
        feature = ogr.Feature(featureDefn)
        pont = ogr.Geometry(ogr.wkbPoint)
        pont.AddPoint(lon, lat)
        feature.SetGeometry(pont)
        feature.SetField("datum", field1)
        layeruj.CreateFeature(feature)

ds.Destroy()
```

Vonalak egyszerűsítése

Olvassuk be a `lines.shp` fájlt. Hívjuk meg a vonalakon a *Simplify()* függvényt. A *SimpliFy(0.01)* függvény a Douglas–Peucker módszerrel egyszerűsíti a vonalakat. A zárójelben megadott paraméter mindig a réteg vetületének mértékegységében, (tehát itt fokban) értendő. A fájlbeolvasás megegyezik a korábban leírtakkal.

```
from osgeo import ogr
from osgeo import osr
driver = ogr.GetDriverByName("ESRI Shapefile")
dataSource = driver.Open('lines.shp', 0)
layer = dataSource.GetLayer()
featureCount = layer.GetFeatureCount()


srs =  osr.SpatialReference()
srs.ImportFromEPSG(4326)
ds = driver.CreateDataSource("simplified_line.shp")
layeruj = ds.CreateLayer("line", srs, ogr.wkbLineString)
idField = ogr.FieldDefn("id", ogr.OFTInteger)
layeruj.CreateField(idField)
featureDefn = layeruj.GetLayerDefn()
```

A ciklusban most nem szükséges csomópontonként hozzáadni a vonalat (bár úgy is lehet, de felesleges bonyolítása a feladatnak), mivel már az egyszerűsítés után egy új geometria jön létre, tehát elég a feature-nek beállítani az új geometriát.

```
for feature in layer:
    geom = feature.GetGeometryRef()
    newgeom=geom.Simplify(0.01)
    field1=feature.GetField("id")
    feature = ogr.Feature(featureDefn)
    feature.SetGeometry(newgeom)
    feature.SetField("id", field1)
    layeruj.CreateFeature(feature)

ds.Destroy()
```

Poligonok írása

Képezzünk a vonalak (`lines.shp`) köré befoglaló téglalapokat. Ezeket írjuk ki fájlba.

A fájl eleje megegyezik a korábban leírtakkal – egy különbség van, hogy itt az adattípus az poligon.

```
 from osgeo import ogr
from osgeo import osr
driver = ogr.GetDriverByName("ESRI Shapefile")
dataSource = driver.Open('lines.shp', 0)
layer = dataSource.GetLayer()
featureCount = layer.GetFeatureCount()


srs =  osr.SpatialReference()
srs.ImportFromEPSG(4326)
ds = driver.CreateDataSource("bounding_boxes.shp")
layeruj = ds.CreateLayer("line", srs, ogr.wkbPolygon)
idField = ogr.FieldDefn("id", ogr.OFTInteger)
layeruj.CreateField(idField)
```

```
featureDefn = layeruj.GetLayerDefn()
```

Menjünk végig az elemeken, hívjuk meg a *GetEnvelope()* függvényt. A függvény egy listával tér vissza, ez a négy határoló koordináta: *[West, East, South, North]*. Hozzunk létre először egy gyűrűt (LinearRing), adjuk a gyűrűkhöz a megfelelő sorrendben a koordinátákat (az első és az utolsó megegyezik!). Hozzuk létre a poligont, a gyűrűt adjuk a poligonhoz, majd a poligont definiáljuk feature-ként. A többi rész szintén megegyezik.

```
for feature in layer:
    geom = feature.GetGeometryRef()
    env=geom.GetEnvelope()
    ring = ogr.Geometry(ogr.wkbLinearRing)
    ring.AddPoint(env[0],env[3])
    ring.AddPoint(env[0],env[2])
    ring.AddPoint(env[1],env[2])
    ring.AddPoint(env[1],env[3])
    ring.AddPoint(env[0],env[3])
    poly = ogr.Geometry(ogr.wkbPolygon)
    poly.AddGeometry(ring)

    field1=feature.GetField("id")
    feature = ogr.Feature(featureDefn)
    feature.SetGeometry(poly)
    feature.SetField("id", field1)
    layeruj.CreateFeature(feature)

ds.Destroy()
```

Egyéb linkek, olvasnivaló a témában:

https://www.gis.usu.edu/~chrisg/python/2009/lectures/ospy_slides1.pdf

https://www.gis.usu.edu/~chrisg/python/2009/lectures/ospy_slides2.pdf

https://www.gis.usu.edu/~chrisg/python/2009/lectures/ospy_slides3.pdf

https://www.gis.usu.edu/~chrisg/python/2009/lectures/ospy_slides4.pdf

https://www.gis.usu.edu/~chrisg/python/2009/lectures/ospy_slides5.pdf

https://www.gis.usu.edu/~chrisg/python/2009/lectures/ospy_slides6.pdf

https://www.gis.usu.edu/~chrisg/python/2009/lectures/ospy_slides7.pdf

KML fájlok lényeges szerkezeti elemei

További leírások, és példák a hivatalos dokumentációban:

https://developers.google.com/kml/documentation

Illetve Gede Mátyás jegyzetében:

https://mercator.elte.hu/~saman/hu/ge_alk/kml.html

A KML fájlok XML állományok, amelyekben tulajdonság-érték párokat találunk:

```
<tulajdonság> érték </tulajdonság>
```

Minden elemnek van nyitó és záró *tag*je, közte pedig az elem értéke olvasható.

A fájl fejléce attól függően, hogy van-e benne animáció vagy sem – lehet rövidebb vagy hosszabb. Érdemes felkészíteni a fájlt mindenre a későbbi tartalomtól függetlenül. A gx-típusú elemek megadásához elengedhetetlen az erre utaló link.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2"
xmlns:gx="http://www.google.com/kml/ext/2.2">
```

A KML fájl felépítése a következő. Általában a fájl elejére helyezzük a stílusdefiníciókat, majd utána megadjuk az egyes geometriai elemeket. Bármely geometriai elem a *Placemark tag*be kerül.

A stílusdefiníciónál meg kell adni a stílus azonosítóját (*id*), ezután megadjuk a paramétereket.

```
<Style id="poligoncsoport1">
<PolyStyle>
        <color>64B40014</color>
        <fill>1</fill>
        <outline>0</outline>
</PolyStyle>
</Style>
```

Majd az elemnél erre a stílusra a következőképpen hivatkozunk.

```
<styleUrl>#poligoncsoport1</styleUrl>
```

A KML-ben a következő geometriai elemek lehetnek (a *Placemarkok* típusai): *Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, GeometryCollection*. Bármelyik geometriai elem esetében a harmadik koordináta megadásával lehet térben elhelyezni az egyes elemeket. A harmadik koordinátát többféleképpen is értelmezhetjük (*altitudeMode*). Ezek jelentése: A *relativeToGround* a felszín felett a megadott magasságban (pl. 100 m), *absolute* a 0 tengerszint feletti magasságban (ha a megadott magasság kisebb, mint a földfelszín magassága, akkor a föld alá kerül az elem és nem látszik), *clampToGround* a felszínhez tűzve, *relativeToSeaFloor* a tengerfenék feletti magasságban, és a *clampToSeaFloor* pedig a tengerfenékhez tűzve.

Az elemet lebegtethetjük a föld felett, vagy összeköthetjük azzal – *extrude* érték 0 vagy 1.

A *tessellate* az elemeknél meghatározza, hogy követi-e a felszín görbületeit (1) vagy inkább egyenes (0).

Az egyes elemekhez név a *name tag*gel, leírás a *description tag*gel, stílus a *styleUrl tag*gel rendelhető.

```
<name>Név</név>
<description>Leírás</description>
<styleUrl>#Stilus1</styleUrl>
```

Animációkból két fajtát különböztetünk meg: a *TimeStamp* és a *TimeSpan*.

Az egyszerűbb a *TimeSpan*. Itt egy kezdő (*begin*) és ha szükséges záróidőponttal (*end*) ritmizálhatjuk az egyes elemek megjelenését és eltűnését. Például, ha olyan animációt készítünk, hogy valamely attribútum mennyiségi változását ábrázoljuk pl. népesség változása 10 évente, akkor látszólag ugyanaz a felület (a megye területéből a lakosságszám szerint kihúzott poligon) először eltűnik, majd a nagyobb értékkel megjelenik. Ez persze a fájlban két külön elem lesz, mindkettőnek más a magassága és a kezdő és végdátuma.

```
<TimeSpan>
      <begin>1986</begin>
      <end>2023</end>
</TimeSpan>
```

A *gx:track* animációt útvonalak bemutatására használjuk. Itt egy koordináta párt (hely) és egy időpillanatot (*TimeStamp*) kell megadni (mindig párban, különben hibás a fájl). Jó példa rá pl. egy túra útvonalának megadása (melyik pillanatban hol jártunk). A mozgást pedig pl. egy sétáló ember ikonnal (vagy bármilyen képpel vagy modellel) tudjuk méginkább szemléltetni.

A *gx:track* animáció-t a vonalhoz, vagy a mozgatandó ikonhoz (modellhez) kapcsoljuk. Kicsit módosul koordináták megadása, a *gx:coord*-ban nincs vessző, csak szóköz.

Az időbélyegző év-hónap-napTóra:perc:másodpercZ formátumú. Feltéve, ha megadjuk az időt, nemcsak a dátumot.

```
<Placemark>
<gx:Track>
      <when>2017-01-02T17:00:22Z</when>
      <when>2017-01-02T17:03:22Z</when>
      …
      <gx:coord>19.03814980042824 47.46165145965755 0</gx:coord>
      <gx:coord>19.03837381154784 47.46229927027417 0</gx:coord>
      …
</gx:Track>
      <name>Take a virtual walk together!</name>
      <description> Lets explore the Buda Castle quarter!
</description>
      <Style>
            <IconStyle>
               <Icon><href>traveller.png</href></Icon>
            </IconStyle>
      </Style>
</Placemark>
```

Python modulok telepítése WHEEL fájlokból

A Python modulok telepítése az éppen használt környezettől függően változhat. Azonban a következő módszerrel egy Python Shell felület vagy bármely más eszközhöz tudjuk telepíteni a modulokat.

Hogyan telepítsük csomagkezelőből?

`PIP INSTALL` 'a modul neve'

```
pip install geopy
```

Lehetséges közvetlenül a wheel fájlokból telepíteni. Ehhez letöltjük a modult a saját gépünkön egy mappába, majd a Windows Parancssorban meghívjuk a telepítést.

A Python Package Index honlapon rengeteg Python modul közül válogathatunk.

https://pypi.org/

Van egy másik nem hivatalos gyűjtemény is:

https://www.lfd.uci.edu/~gohlke/pythonlibs/

Mi az a WHEEL fájl?

A WHEEL fájlok lefordított bináris formátumú archívumok, amelyekben Python modulok és könyvtárak telepítéshez szükséges kódjait tároljuk. A WHEEL fájlok egyszerűsítik a telepítést, mivel nem a felhasználónak kell a nyers programkódból lefordítni a fájlt.

Hogyan telepítsük pl. a geoPy-t wheel fájlból?

1. Letöltjük a WHEEL fájlt például a C:/Letöltések mappába.
2. Windows-on a cmd.exe-ben (Parancssor), válasszuk ki a felhasználó mappáját a cd parancs segítségével, majd adjuk meg a letöltött fájl helyét.

```
C:\Users\ungvarizs>
```

3. Hívjuk meg a pip csomagkezelő telepítő funkcióját. A WHEEL archívum teljes fájlnevét meg kell adni.

```
py -m pip install C:\Users\ungvarizs\Downloads\geopy-2.2.0-py3-
none-any.whl
```

Olyan eset is fennállhat, hogy a kiszemelt Python modulnak van függősége. Pl. OSGEO modulnak a `numpy`. Ilyenkor elég meghívni az `osgeo` telepítőjét, a `pip install` elintézi nekünk a `numpy` modul telepítését is.

Vannak olyan modulok, pl. `pandas`, amely bizonyos funkciói igényelhetik más modulok jelenlétét. De mivel nem létszükséglet ezek jelenléte (gyenge függőség van), így a felhasználónak kell telepítenie, automatikusan nem fog települni. Ilyen az Excel fájlok megnyitására használatos `openpyxl`.

A következő listában a jegyzetben használt modulok beszerzési helyét gyűjtöttem össze:

`GDAL`: https://pypi.org/project/GDAL/#files

`pandas`: https://pypi.org/project/pandas/#files

`openpyxl`: https://pypi.org/project/openpyxl/

`geoPy`: https://pypi.org/project/geopy/