# Prediktion av fastighetspriser

## Förarbete

### Datainsamling

Tidigare har jag skrivit ett skript för skrapning och tanken var att återanvända koden för att samla in data från Booli.se till detta projekt. Jag hittade dock ett tillägg till webbläsaren som gjorde ungefär samma sak så för att spara tid använde jag mig av ett tillägg vid namn ”Web Scraper”. Jag hade från början tänkt använda data från 2020-01-01 och framåt men det fanns begränsningar hos Booli som innebar att det bara gick att komma åt 35 objekt á 1000 sidor åt gången. Så jag delade upp sökningarna med datumfilter för att hämta mindre än 35000 poster per sökning och skrapade i flera ”mindre” stötar och bestämde mig då för att begränsa mig till 2024-01-01 och framåt. Detta gav mig drygt 313 000 st objekt.

### Databearbetning

När skrapningen var klar behandlade jag all data i Excel. Den innehöll en hel del fel där kolumnerna blandades, troligen för att olika mäklare knappat in olika saker i olika fält så jag fick manuellt genom filtreringar och sökningar flytta runt stora mängder data, vilket tog några dagar längre än planerat. All data sparade jag i en CSV-fil.

## Kodning

### Översikt

För Första delen av projektet gjorde jag ett ETL-flöde, som läser in en CSV-fil (data.csv), städar och transformerar data och sedan sparar resultaten i en SQLite-databas (fastigheter.db).

### Konfiguration av loggning

Skriptet skriver loggmeddelanden till en fil med namn fastighet\_log\_YYMMDD.log där datumet används utifrån dagens datum och konfigurerar loggnivå och format.

### Extrahera och transformera

Skriptet letar sen upp en fil som heter data.csv, som förväntas ligga i samma mapp som skriptet. Filen läses in med Pandas och använder en tolk som själv ska uppskatta vilken separator som använts i filen. Skriptet hanterar fel genom att logga och returnera en tom DataFrame och loggar även antal lästa rader.

Om data är tom loggas det och returnerar en tom dataframe. Annars börjar den transformera data och se till att den är användbar och har rätt format och kolumnnamn. Bearbetningen loggas.

### Dela upp, spara och testa

Här delar jag upp dataframen baserat på bostadstyp och lägger i en dict där nyckeln är bostadstypen och värdet är respektive databas. Sen sparas dataframen till en SQLite databas i samma mapp. Jag gjorde även ett test för att kontrollera de olika funktionerna genom att skicka in hittepåvärden och se om de hanterats korrekt.

### EDA

EDA gjorde jag löpande efter att databasen laddades in på nytt i ett jupyter script. Här fortsattes bearbetning utefter vilka behov varje boendetyp hade. Biarea kan t.ex lämnas tomt medan Boarea är ett krav, samtidigt är biarea generellt inte något som är relevant för lägenheter. Det som ligger kvar som kod är generellt resultatet av vad jag kommit fram av att lära känna min data snarare än själva EDAn.

### Machine learning

I filen fastighet\_ml.ipynb är koden för bearbetning och träning/test av själva modellerna. Till en början finns initieringen till loggningen, byggd på samma sätt som innan, och en import av databasen. Sen följer kod för att ersätta NaN med 0 i biarea då saknad data antas helt enkelt vara att det inte finns någon biarea samt formatering av datum. Skillnaden mellan radhus, kedjehus och parhus är ganska liten så dessa slog jag ihop. De värdena som fanns i ”hus” bestod mest av fel och var ganska få objekt så jag tog bort den dataframen (df) såväl som de hopslagna, ”tomt” samt ”övrigt”.

Sen itererade jag över dataframarna och satte nollvärden till NaN på Boarea och Rum. Nu minns jag inte exakt varför, men troligen för att de inte ska räknas som värden när jag tränar en modell för att fylla i saknade värden. Sen tog jag bort alla rader som saknade både boarea och rum, då minst en av dessa behövs, från alla df och de rader som saknade tomtarea för ”villa”.

Härnäst tar jag bort kolumnen ”våning” från alla df utom lägenhet då de är irrelevanta samt ”tomtarea” och ”biarea” från ”lägenhet” av samma anledning.

Efter detta följer en Random Forrest-modell som tränas att prediktera antal rum baserat på boarea, då dessa generellt hänger ihop, och fyller i saknade värden med predikterade värden. Här valde jag halvtal. Sen en som gör samma sak igen fast predikterar boarea efter antal rum, men med heltal. Sen för att göra densamma med ”fritidshus”, ”gård” och ”radhus” så itererade jag över dessa df med modellen tränad på ”villa”.

Sen städar jag datan där jag bland annat tar bort alla NaN från våning, boarea och rum för lägenheter då det fanns så många datapunkter i den dataframen. Nu i efterhand känns det kanske onödigt då en modell för att fylla i dessa redan gjorts men jag tror jag tog detta beslut redan innan jag kom på att fylla i predikterade värden och sen ifrågasatte jag inte beslutet. Jag tog även bort NaN-värden i tomtarea från ”gård” och ”radhus samt ersatte kommatering med punkt och konverterade kolumnen våning till numeriska värden med avrundning. Sen gjorde jag egentligen samma sak härnäst men itererade över övriga df för att konvertera tomtarea, boarea och biarea till tal.

Under EDAn så hittade jag bland annat att på vissa villor hade sålts för 1/20 av priset som generellt gällde på den gatan, vilket kan vara köp av en förälder, skilsmässa eller liknande men oavsett anledning så är de felaktiga vid prediktion av marknadspriser. Så härnäst itererade jag över df och kollade om adress och ort var densamma och priset diffade med mer än 50% så skulle datapunkten droppas.

Här är förarbetet klart och nu följer en mängt träningar. Jag började med ”villa” och testade CatBoost (CB), LightGBM (LGBM) och Random Forrest (RF). CB var först ut och här tog jag bort 5% av de översta och understa för att få bort lite outliers. Enligt koden så log-transformerar jag här, men första gången jag körde modellen gjorde jag inte det utan det uppdaterade jag med senare för att testa. Jag hade tänkt använda Totalarea, men den var bara en addering av bo och biarea så jag valde sen att exkludera den, tillsammans med andra onödiga kategorier som pris, datum, bostadstyp och nyckel då ingen av dessa tillför något i en prediktion. Sen identifierar jag kategoriska kolumner och börjar sen träna modellen. Till en början skrev jag ut värdena men jag upptäckte att det var helt grymt att logga allt i stället så var det enklare att följa. Jag loggade bland annat R², RMSE och viktigast features. Härnäst följer en modell utan log-priser som uppdaterats med parametrar från en Grid Search (GS) som ligger nedanför. Detta var modellen som gav bäst resultat efter många omkörningar och i slutet på modellens kod dumpade jag den till en fil. Sen återanvände jag koden till nästa df. Jag körde GS några gånger på olika modeller men den längsta tog drygt 8 timmar så jag gick över till Randomized Search (RS) i stället. Nu när jag ser efter så är det ju en RS jag kört även på CB på villadata, men jag måste råkat ersätta GS med en RS även om jag tänkte ha båda kvar, även om jag inte skulle köra GS flera gånger.

Jag gjorde en RF härnäst utifrån samma struktur som CB men den tog väldigt lång tid och gav ganska dåligt resultat så jag släppte RF efter detta. I stället hittade jag modellen LGBM som var extremt snabb och gav nästan lika bra (teoretiska) resultat som CB. Även denna byggde jag på samma sätt som CB. Sen följer många upprepningar där jag kör LGBM, CB och sen RS för att förbättra parametrarna som jag petade in i modellerna igen och körde om. Detta upprepade jag med ”lägenhet”, ”fritidshus” och till slut ”radhus”. ”Gårdar” släppte jag då det fanns lite för få datapunkter. Nu i efterhand kanske det hade varit snyggt att iterera över df i vid träningen av modellerna och kanske använda funktioner, men det får jag titta på vid senare tillfälle.

### Streamlit

Sista koden ligger lite fantasilöst i filen Streamlit.py. Där importerar jag LGBM och CB med de bästa parametrarna från respektive bostadstyp. Sen lägger jag upp olika val där man får välja vilken bostadstyp man är intresserad av och sen knappa in egen data i fält. Beroende på vilken bostadstyp man klickat i så körs en av fyra if-satser, men som alla i princip ser likadana ut. Först läggs de angivna värdena i en dict och bygger en df av värdena i samma struktur som modellerna tränats på. Sen anges vilka av dessa som är kategoriska. Därefter initieras en variabel i streamlits sessionstillstånd. Den används för att spara resultatet av föregående sökning så att den kan visas tillsammans med den nuvarande beräkningen, och bevaras även mellan olika körningar i appen. Sen följer kod som aktiveras genom att man klickar på ”Beräkna pris” som då kör två prisprognoser, en med LGBM och en med CB och visar resultatet i gränssnittet tillsammans med föregående sökning (som är lite genomskinlig för tydlighets skull), om det finns en.

Till sist så finns kod för att rösta på vilken prediktion som var närmast (om man vet). Denna sparas i en csv-fil som skapas om den inte redan finns sen tidigare och läses in. Sen följer boendeformen man tidigare valt med och här kommer röstningsknappar som sparar rösten för respektive modell och boendeform. Sen visas resultatet, vilken modell som presterat bäst flest gånger (i procent), per boendeform.

### Diskussion och slutsats

Det var över lag ett kul projekt. För att göra bättre prediktioner så skulle det behövas mer och bättre data. Dels tänker jag att data från 2020 och framåt hade varit bra, åtminstone data från efter pandemin. Kanske med någon viktning på åldern för datan. En gammal datapunkt tror jag är bättre än ingen, speciellt utanför städer. Sen så finns det parametrar som påverkar bostadspriser kraftigt som inte fanns med. Sjönära läge gör lätt en skillnad på en miljon, kanske mer. Månadsavgift för lägenhet påverkar priset kraftigt, en avgift på 4000 för en 4:a jämfört med 8000 innebär motsvarande räntekostnad på ett 2 miljoner högre lån. Driftkostnad i villor är också en sådan variabel som påverkar stort då 20 000 i drift med effektiv värmepump jämfört med 45 000 med olja motsvarar räntekostnad för omkring 1 miljon. Byggår, renoveringar etc.

Min uppfattning är att det ändå vore fullt möjligt att göra en modell som predikterar bostadspriser bra, men då behövs en mer gedigen grunddata med fler parametrar. Själva koden är helt okej men jag tycker ML-delen är lite rörig. Jag tror den skulle bli mycket snyggare om man la den upprepade träningen i funktioner på något sätt, men samtidigt är inte datan till alla modeller likadan, men det skulle nog gå att lösa på något sätt.