Department of Computer Science
Technical University of Cluj-Napoca

Intelligent Systems
Machine Learning

Tools: JupyterLab, ScikitLearn

Name:Popa Alexandra
Group: 30234
Email:popa_ada_98@yahoo.com
Teacher: Loredana Coroama

Contents

1 Descriere proiect **4**

2 Descriere dataset **5**

3 Tool-uri folosite **6**

4 Detalii de implementare **7**

4.1 Reading dataset 7

4.2 Cleaning dataset 7

4.3 Data selection 8

4.4 Split dataset 8

4.5 Machine Learning Algorithm 8

4.6 Linear Regression 9

4.7 Decision Tree 9

4.8 Support Vector Machine 9

4.9 K-Nearest-Neighbour 9

4.10 Multi-Layer Perceptron 10

4.11 Ensemble Learning:Voting Regressor 10

5 Grafuri si experimente **11**

5.1 Metrici utilizate 11

5.2 Rezultate 12

6 Munca si documentare **16**

7 Concluzii si imbunatatiri **17**

8 Ghid de rulare a proiectului **18**

9 Codul original **20**

Bibliography **23**

Chapter 1

Descriere proiect

Începand cu 2008, oaspetii și gazdele folosesc Airbnb pentru a se extinde asupra posibilitatilor de calatorie și pentru a prezenta un mod mai unic, personalizat de a experimenta lumea. Setul de date ales descrie activitatea si valorile de aparitie in aplicatie a proprietatilor de inchiriat din NYC, NY pentru anul 2019. [1] Scenariul propus este creat pentru prezicerea preturilor pentru apartamentele inchiriate in regim AirBnb din orasul NewYork. Prezicerea se poate realiza transformand scenariul ales intr-o problema de regresie.

Chapter 2

Descriere dataset

Datele folosite pentru realiarea proiectului au fost luate de pe site-ul www.kaggle.com.[1] Dataset-ul preluat include toate informațiile necesare pentru a afla mai multe informatii despre gazde, disponibilitatea geografică,si mai multe valori necesare pentru a face predicții și a trage concluzii. Acest set de date publice face parte din Airbnb, iar sursa originală poate fi găsită pe site-ul web insideairbnb.com.[2]

Dataset-ul utilizat prezinta urmaroarele caracteristici:

id → id-ul afisarii pe site

name → titlul afisarii pe site

host_id → ID -ul gazdei

host_name → numele gazdei

neighbourhood_group → locatia

neighbourhood → zona

latitude → coordonatele latitudinii

longitude → coordonatele longitudinii

room_type → tipul camerei;tipuri: Entire home/apt, Private room, others

price → pretul in dolari

minimum_nights → numarul de nopti minime de sedere

number_of_reviews → numarul de review-uri

last_review → data celui mai recent review

reviews_per_month → numarul de review pe luna

calculated_host_listings_count → cantitatea de vizualizari pe gazda

availability_365 → numarul de zile din an in care oferta de inchiriere este valabila

Chapter 3

Tool-uri folosite

Pentru realizarea acestui proiect am folosit Scikit-learn, librarie care ofera instrumente simple și eficiente pentru analiza datelor predictive. Cu ajutotul acestui tool, an facut spit pe date, am prelucrat coloanele tabelului, am curatat setul de date initiale, dar totodata am aplicat si analiat algoritmii necesari rezolvarii ploblemei propuse.

Alte doua librarii utilizate sunt pandas, numpy si graphziz cu ajutorul carora am extras, analizat si vizualizat datele.[3]

Un alt tool utilizat ca mediu de lucru este JupyterLab, o interfata utilizator web care permite dezvoltarea programelor intr-un mod flexibil. [4]

Chapter 4

Detalii de implementare

4.1 Reading dataset

Pentru inceput am extras tot setul de date citind fisierul de tip .csv, urmand ca apoi sa prelucrez si utilizez datele de intrare initiale. Datele initiale cuprind: 48895 rows x 16 columns.

```
[123]: data_df.columns

[123]: Index(['name', 'host_name', 'neighbourhood_group', 'neighbourhood', 'latitude',
              'longitude', 'room_type', 'price', 'minimum_nights',
              'number_of_reviews', 'last_review', 'reviews_per_month',
              'calculated_host_listings_count', 'availability_365'],
             dtype='object')
```

Figure 4.1: Reading dataset

4.2 Cleaning dataset

Desi dataset-ul ales vine cu o varietate de features, in cazul in care eu doresc sa prezic pretul unei proprietati inchiriate, doar anumite atribute sunt utile pentru determinarea rezultatului final. Astfel, am decis sa aleg doar atributele pe care le consier mai importante pentru determinarea pretului, asa ca am renuntat la urmatoarele coloane : id si host_id.

```
[4]: data_df.drop(columns=["id"],inplace=True)
     data_df.drop(columns=["host_id"],inplace=True)
```

Figure 4.2: Cleaning dataset

De asemenea in aceasta sectiune am verificat coloanele astfel incat sa nu contina valori nule (NaN), iar in cazul in care am descoperit date lipsa, s-au utilizat diferire strategii de inlocuire a acestora: cu media celorlate date (in cazul unor valori numerice), cu ce-a mai frecventa aparitie (in cazul datelor non numerice).

```
[74]: data_df["last_review"].unique()

[74]: array(['2018-10-19', '2019-05-21', nan, ..., '2017-12-23', '2018-01-29',
             '2018-03-29'], dtype=object)
```

Figure 4.3: Vizualizare date dintr-o coloana: am observat ca exista valori nan

```
[15]: #AICI FACE SI PARTE DE CLEANING DATASET
preprocessor = ColumnTransformer([
    ("num", Pipeline([("imputer", SimpleImputer(missing_values=np.nan, strategy="mean")),
                      ("scaler", StandardScaler())]), numeric_features),
    ("nom", Pipeline([("imputer", SimpleImputer(missing_values=np.nan, strategy="most_frequent")),
                      ("binarizer", OneHotEncoder(handle_unknown="ignore"))]), nominal_features)],
    remainder="drop")
```

Figure 4.4: Inlocuire date lipsa

4.3 Data selection

Dupa curatarea dataset-ului urmeaza selectarea datelor care vor fi utilizate in continuare, iar pentru aceasta am folosit un procent de 5% din datele initiale, pentru a eficientiza timpul de prelucrare a datelor cat si pentru o mai buna vizualizare a rezultatelor obtinute. In urma acestu proces, s-a obtinut un dataset de dimensiunea 2445 rows x 14 columns.

```
[5]: data_sel=data_df.sample(frac=0.05, replace=True, random_state=1, axis=0)
```

Figure 4.5: Data selection

4.4 Split dataset

Setul de date a fost divizat intr-un set de date de test (test set) si un set de date de antrenament(training set). Astfel, deoarece doresc ca programul sa imi prezica pretul,am extras pretul din setul de date si am impartit astfel 2 seturi de date pe care le-am divizat in 2 categorii(X_train, X_test,y_train, y_test).

```
[7]: y=data_sel["price"]

[8]: data_sel.drop(columns=["price"],inplace=True)

[9]: X=data_sel

[10]: X_train,X_test,y_train,y_test=train_test_split(X,y, test_size=0.20, random_state=42)
```

Figure 4.6: Split dataset

4.5 Machine Learning Algorithm

Pentru aplicarea algoritmilor de Machine Learning este necesar sa impart datele in : date numerice si date nominale. Aceste date impartite in doua categorii se vor trece printr-un pipeline de transformare a coloanelor, cu ajutorul clasei "Pipeline" ce are ca scop standardizarea seturilor ce contin ambele tipuri de date.[5][6] Prin standardizare se intelege maparea valorilor intr-o anumita distributie pentru a putea fi prelucrate ulterior de algoritm. Tot in acest pipeline s-au utilizat diferire strategii de inlocuire a datelor lipsa (cleaning code).

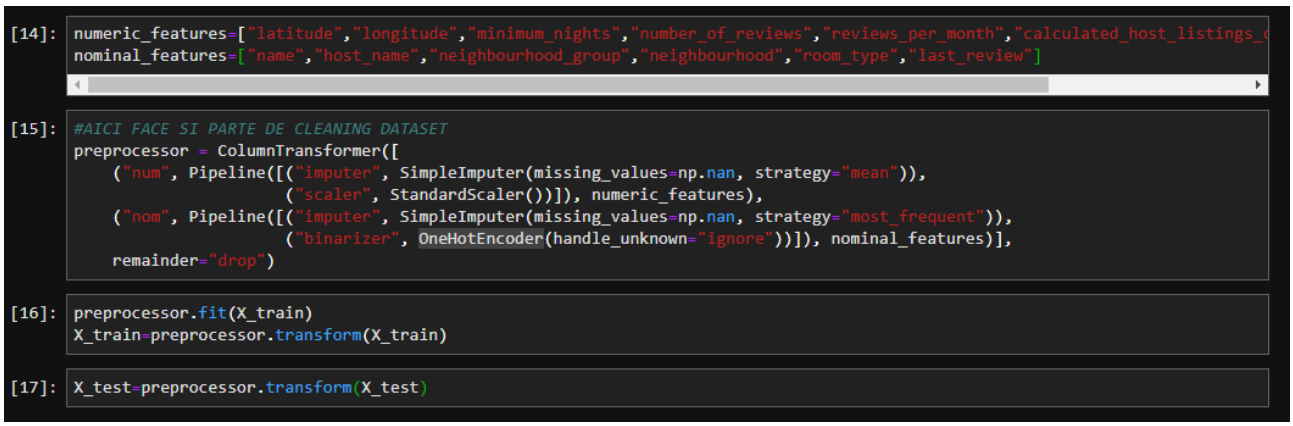


Figure 4.7: Pipeline transformare coloane

4.6 Linear Regression

LinearRegression se potrivește unui model liniar cu coeficienții w = (w1,..., wp) pentru a minimiza suma reziduală a pătratelor dintre obiectivele observate din setul de date și obiectivele prevăzute de aproximarea liniară.[7]

Acesta algoritm este unul simplu si a fost utilizat fara a se folosi vre-un parametru. Pentru determinarea scorului in urma aplicarii acestui algoritm, am folosit cross_val_score, ce returneaza matricea scorurilor estimatorului pentru fiecare rundă de validare încrucișată.[8] Pentru determinarea acestei matrici s-a folosit mertrica r2_score.

4.7 Decision Tree

Acestata metoda a fost aplicata de doua ori pe setul de date, utilizand parametrii diferiti. Prima data s-a folosit acest algoritm fara nici un parametru, iar mai apoi s-a adaugat parametrul ce specifica numarul maxim al adancimii arborelui, pentru a se putea efectua compararea scorului obtinut in urma utilizarii metricii r2_score pe cele doua variante si a decide care metoda este mai eficienta pentru a face o predictie cat mai buna.

Cu ajutorul librariei graphviz s-a putut face vizualizarea datelor in urma procesului de selectie a feature-ilor, desfasurat de algoritm .[3]

4.8 Support Vector Machine

Acest algoritm a fost, de asemenea aplicat de doua ori, in moduri diferite pentru a se putea face o comparatie intre scorurile obtinute. Astfel, am aplicat SVR() fara nici un parametru si SVR(kernel='linear',C=100) specificand Kernel.[9] De asemenea, si pentru acest algoritm se utilizaeza metrica r2_score.

4.9 K-Nearest-Neighbour

Clasa utiliata "KNeighborsRegressor" iar scopul este prezicerea prin interpolarea locala a obiectivelor asociate celor mai apropiati vecini din setul de training. [10] Pentru a observa comportamentul algoritmului in functie de scorul obtinut am modificat parametrul "n_neighbors", care

reprezinta numarul de vecini folositi pentru interogari. De asemenea, pentru calcularea scorului am folosit merica "r2_score".

4.10 Multi-Layer Perceptron

Pentru aplicarea acestui algoritm, am utilizat clasa "MLPRegressor" schimband parametrul "hidden_layer_sizes" ce reprezinta numarul de layere ascunse si numarul de neuroni continuti de fiecare layer ascuns in parte, pentru a observa influenta modificarilor facute asupra scorul rezultat. [11]

4.11 Ensemble Learning:Voting Regressor

Un regresor de vot este un meta-estimator de ansamblu care se potrivește mai multor regresori de baza, fiecare pe intregul set de date. Apoi face media predictiilor individuale pentru a forma o predictie finala. Pentru aceasta clasa am utilizat ca parametru "estimators" ce reprezinta o lista de tuple de estimatori, in cazul utilizat acestia fiind: LinearRegression, Support Vector Machine si KNeighborsRegressor. [12]

Chapter 5

Grafuri si experimente

5.1 Metrici utilizate

Pentru calcularea scorurilor obtinute in urma aplicarii algoritmilor, am ales metrica r2_score. R2_score este o metrica pentru care : cel mai bun scor posibil este 1.0 și poate fi negativ (deoarece modelul poate fi arbitrar mai slab). Un model constant care prezice întotdeauna valoarea așteptată de y, fără a ține cont de caracteristicile de intrare, ar obține un scor R $\hat{2}$ de 0,0. [13] Totodata, predictia s-a facut utilizand aceiasi metrica: r2_score.

De asemenea pentru calcularea scorului, am utilizat clasa "cross_val_score", care imparte datele de training in flod-uri si apoi masoara performanta raportata prin k-fold cross-validation , performanta care este defapt media valorilor obtinute.[8] [14]

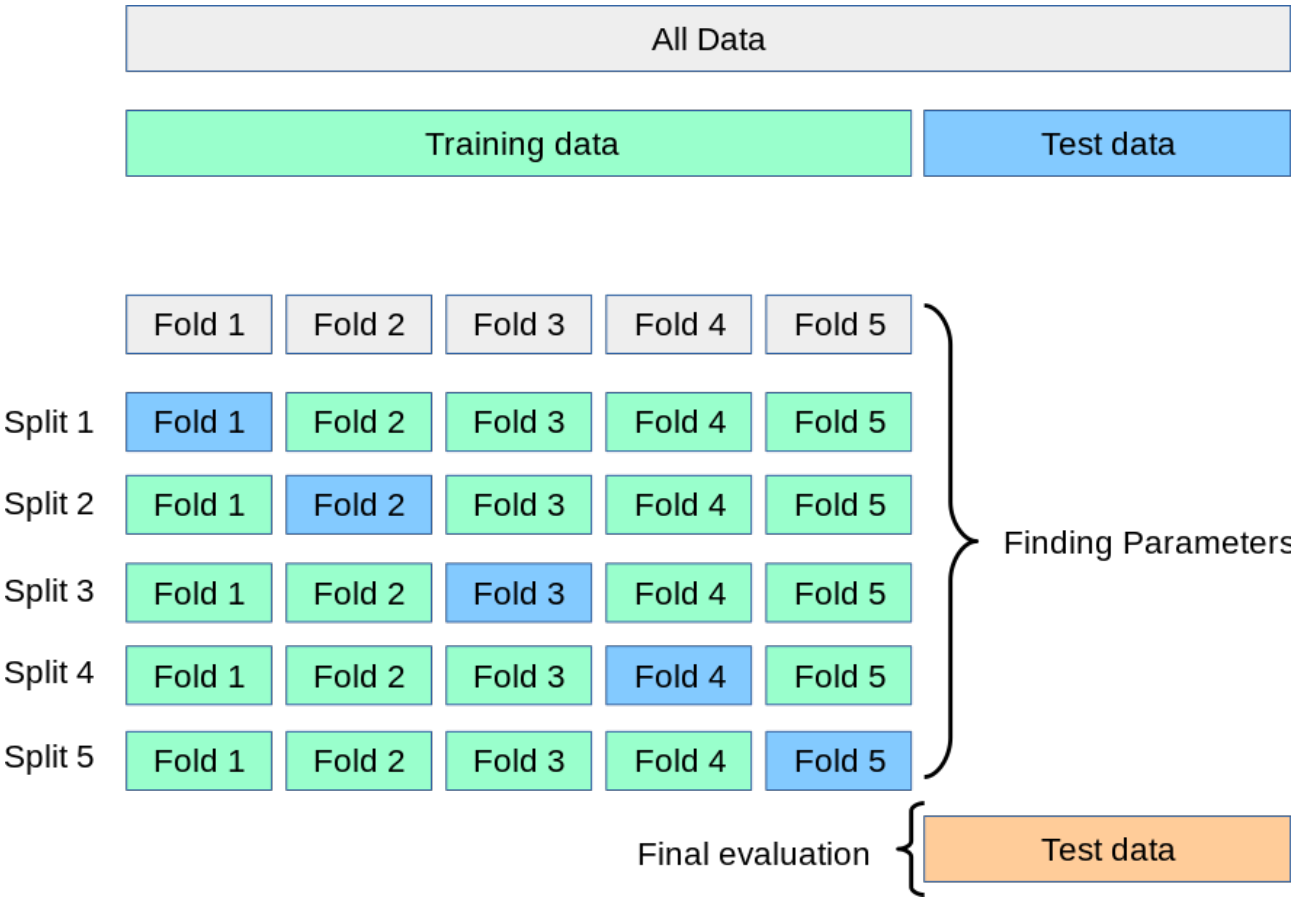


Figure 5.1: Principiul de functionare "cross_val_score"

5.2 Rezultate

Linear Regression

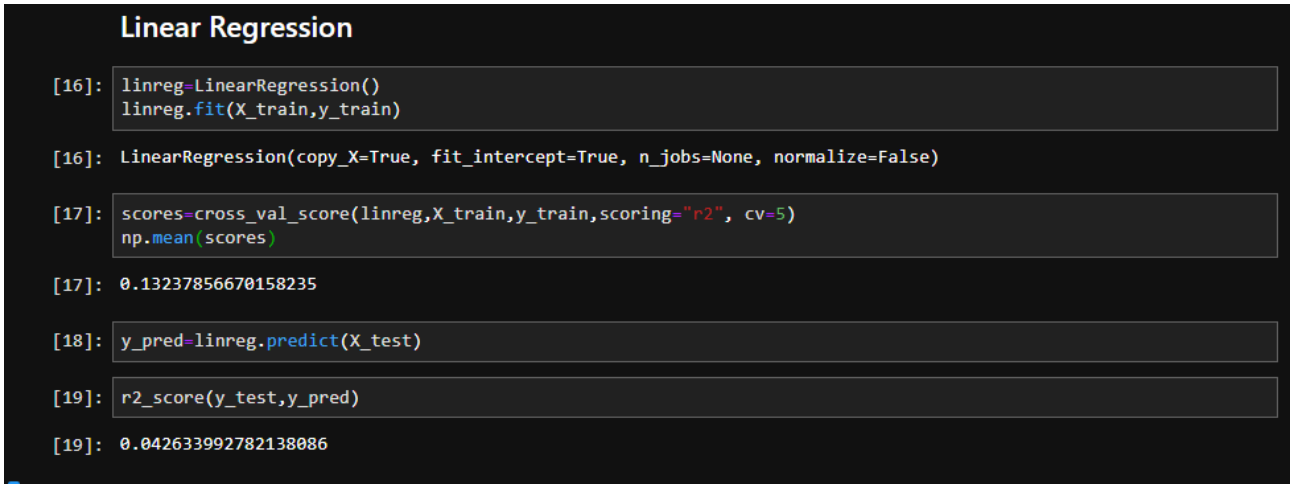


Figure 5.2: Linear Regression results

In urma aplicarii algoritmului s-a obtinut un scor de 0.13237856670158235, ceea ce inseamna ca modelul antrenat este unul constant, mai mare ca 0 ce tinde spre 1(cel mai bun scor posibil). In urma predictiei facute s-a obtinut un scor de 0.042633992782138086.

Decision Tree

Figure 5.3: Decision Tree results

In urma aplicarii algoritmului fara parametrii s-a obtinut un scor de -0.9837772288576939, iar dupa aplicarea folosind parametrul max_depth=3 s-a obtinut un scor de 0.12191035804007921, ceea ce inseamna ca scorul obtinut in urma antrenarii celui de al doilea model este mai bun, pentru ca este pozitiv, mai mare ca 0 ce tinde spre 1, si totodata constant. In urma obtinerii acestui rezultat, am facut o predictie cu ajutorul celui de al doilea model(fiind mai bun) si am obtinut un scor de 0.1594205675889433.

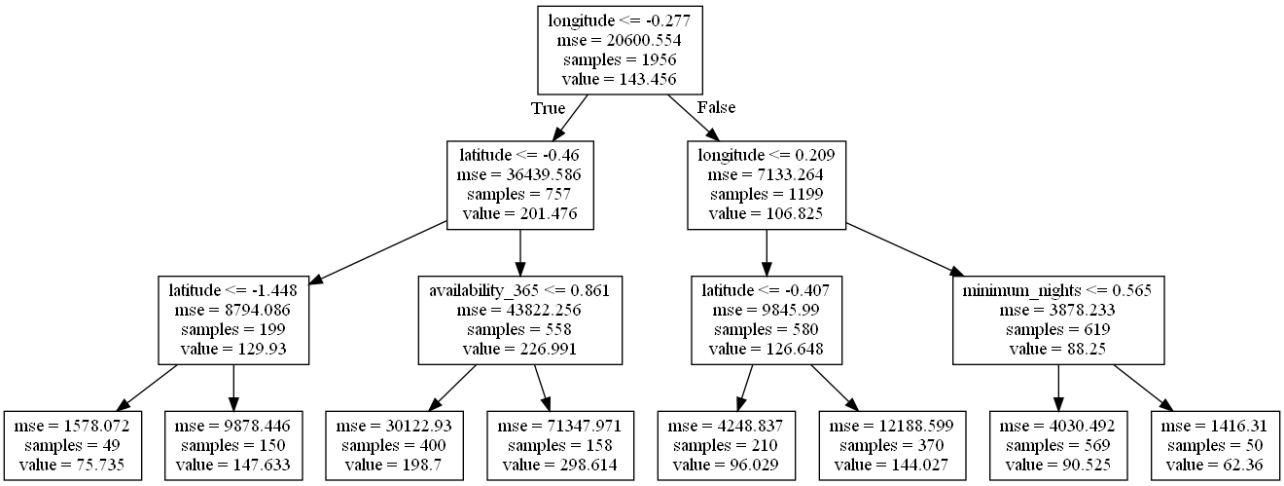


Figure 5.4: Tree results

In urma generarii acestui arbore se poate observa care au fost atributele cele mai relevante selectate de catre algoritm pentru a obtine un rezultat.

Support Vector Machine

```
[27]: regr = svm.SVR()
      regr.fit(X_train,y_train)

      •••

[28]: scores=cross_val_score(regr,X_train,y_train,scoring="r2", cv=5)
      np.mean(scores)

[28]: 0.06668643952748284

[29]: regr1 = svm.SVR(kernel='linear',C=100)
      regr1.fit(X_train,y_train)

      •••

[30]: scores=cross_val_score(regr1,X_train,y_train,scoring="r2", cv=5)
      np.mean(scores)

[30]: 0.2981083052719263

[31]: y_pred=regr1.predict(X_test)

[32]: r2_score(y_test, y_pred)

[32]: 0.43674000148232883
```

Figure 5.5: Support Vector Machine results

In urma aplicarii algoritmului fara parametrii s-a obtinut un scor de 0.06668643952748284, iar dupa aplicarea folosind parametrul kernel=’linear’ s-a obtinut un scor de 0.2981083052719263, ceea ce inseamna ca scorul obtinut in urma antrenarii celui de al doilea model este mai bun, si totodata constant, mai mare ca 0 ce tinde spre 1. In urma obtinerii acestui rezultat, am facut o predictie cu ajutorul celui de al doilea model(fiind mai bun) si am obtinut un scor de 0.43674000148232883, si mai apropiat de valoarea 1.

K-Nearest-Neighbour

```
[33]: neigh = KNeighborsRegressor(n_neighbors=2)
      neigh.fit(X_train,y_train)
      ...
[34]: scores=cross_val_score(neigh,X_train,y_train,scoring="r2", cv=5)
      np.mean(scores)
[34]: -0.12237906999362638
[35]: neigh1 = KNeighborsRegressor(n_neighbors=10)
      neigh1.fit(X_train,y_train)
      ...
[36]: scores=cross_val_score(neigh1,X_train,y_train,scoring="r2", cv=5)
      np.mean(scores)
[36]: 0.28029316717340896
[37]: y_pred=neigh1.predict(X_test)
[38]: r2_score(y_test, y_pred)
[38]: 0.22588109214802599
```

Figure 5.6: K-Nearest-Neighbour results

In urma aplicarii algoritmului cu parametrul n_neighbors=2 s-a obtinut un scor de -0.12237906999362638, iar dupa aplicarea folosind parametrul n_neighbors=10 s-a obtinut un scor de 0.28029316717340896, ceea ce inseamna ca scorul obtinut in urma antrenarii celui de al doilea model este mai bun, deoarece este mai mare ca 0, tinde spre 1 si totodata constant. In urma obtinerii acestui rezultat, am facut o predictie cu ajutorul celui de al doilea model(fiind mai bun) si am obtinut un scor de 0.22588109214802599.

Multi-Layer Perceptron

```
[40]: scores=cross_val_score(mlp,X_train,y_train,scoring="r2", cv=5)
      np.mean(scores)
[40]: 0.20943439431848718
```

Figure 5.7: Multi-Layer Perceptron results

```
[42]: scores=cross_val_score(mlp1,X_train,y_train,scoring="r2", cv=5)
      np.mean(scores)
[42]: 0.04122786261813431
```

Figure 5.8: Multi-Layer Perceptron results

```
[104]: y_pred=mlp.predict(X_test)
[105]: r2_score(y_test, y_pred)
[105]: 0.3092284911538161
```

Figure 5.9: Multi-Layer Perceptron results

In urma aplicarii algoritmului cu parametrii random_state=1,max_iter=100 s-a obtinut un scor de 0.20943439431848718, iar dupa aplicarea folosind si parametrul hidden_layer_sizes=(32,32,32)

s-a obtinut un scor de 0.04122786261813431, ceea ce inseamna ca scorul obtinut in urma antrenarii primului model este mai bun, deoarece este mai mare, tinde spre 1 si e totodata constant. In urma obtinerii acestui rezultat, am facut o predictie cu ajutorul primului model(fiind mai bun) si am obtinut un scor de 0.3092284911538161, si mai apropiat de valoarea 1.

Ensemble Learning:Voting Regressor

Ensemble Learning:Voting Regressor

```
[44]: ereg = VotingRegressor(estimators=[('gb', linreg), ('rf', regr), ('lr', neigh)])
      ereg = ereg.fit(X_train, y_train)

[45]: scores=cross_val_score(ereg,X_train,y_train,scoring="r2", cv=5)
      np.mean(scores)

[45]: 0.23172616584956734

[46]: y_pred=ereg.predict(X_test)

[47]: r2_score(y_test, y_pred)

[47]: 0.2951452836455678
```

Figure 5.10: Ensemble Learning:Voting Regressor results

In urma aplicarii algoritmului cu parametrul estimators=[('gb', linreg), ('rf', regr), ('lr', neigh) s-a obtinut un scor de 0.23172616584956734, ceea ce inseamna ca modelul antrenat este unul constant, mai mare ca 0 ce tinde spre 1(cel mai bun scor posibil). In urma predictiei facute s-a obtinut un scor de 0.2951452836455678.

Rezultatele obtinute in urma predictiilor pentru diferiti algoritmi pot fi comparate cu ajutorul tabelului urmator:

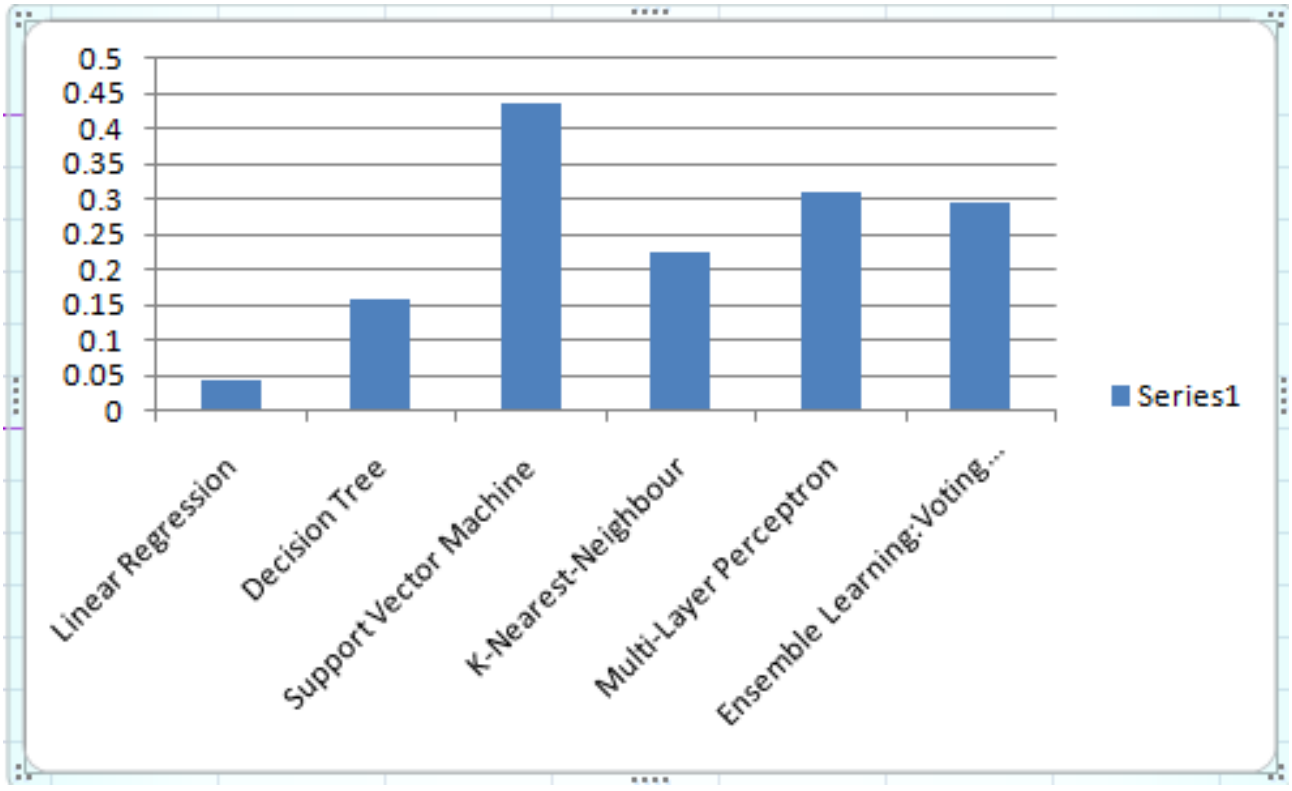


Figure 5.11: Rezultate predictii

Chapter 6

Munca si documentare

Pentru documentarea si realizarea acestui proiect am utilizat surse precum www.kaggle.com de unde am preluat setul de date pe care l-am prelucrat ulterior, dar si insideairbnb.com din care am obtinut informatii cu privire la politica de functionare a sistemului de inchiriere AirBnb precum si informatii privind principalele avantaje si principalii factori care influenteaza pretul si inchirierea.[1] [2]

Chapter 7

Concluzii si imbunatatiri

In concluzie, proiectul propus a avut ca scop analizarea rezultatelor obtinute in urma aplicarii mai multor algoritmi asupra setului de date. Se poate observa faptul ca anumiti algoitmi conduc la un scor mai bun si automat la o prezicerea pretului mai buna, iar alti algoritmi nu dau rezultate la fel de bune(Fig. 5.11).

Pentru imbunatatiri ulterioare as recomanda, testarea algoritmilor utilizand o varietate mai mare de parametrii si valori atribuite acestor parametrii de intrare. Totodata, o influenta pozitiva ar avea selectarea cat mai amanuntita a altor feature-uri si adaugarea acestora la cele existente pentru a obtine rezultate cat mai bune.

Chapter 8

Ghid de rulare a proiectului

Pentru a putea rula proiectul, AnacondaNavigator pune la dispozitie JupyterLab, necesar pentru a deschide fisierul project.ipynb.

AnacondaNavigator se instaleaza utilizand link-ul de pe pagina web anaconda.com.[4]

Dupa instalare, se va deschide fereastra de mai jos:

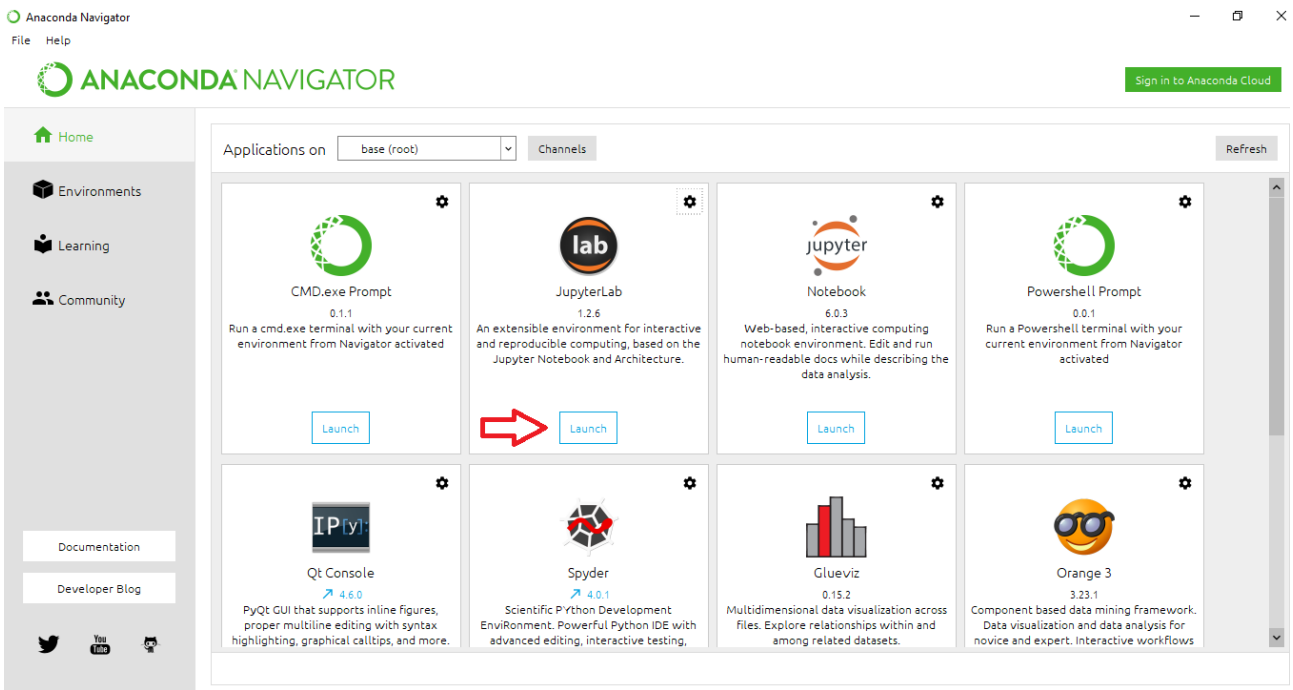


Figure 8.1: AnacondaNavigator

Dupa apasarea butonului "Launch" se va descide mediul de lucru, si astfel se poate selecta numele proiectului din folderul in care se gaseste si astfel se poate deschide.

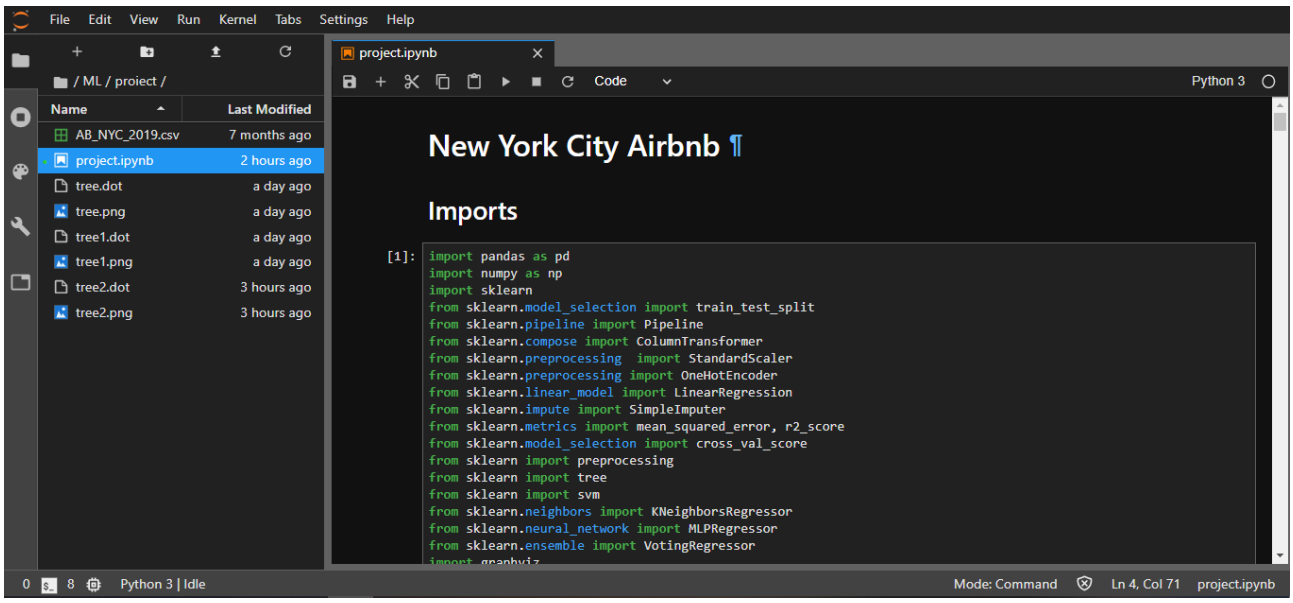
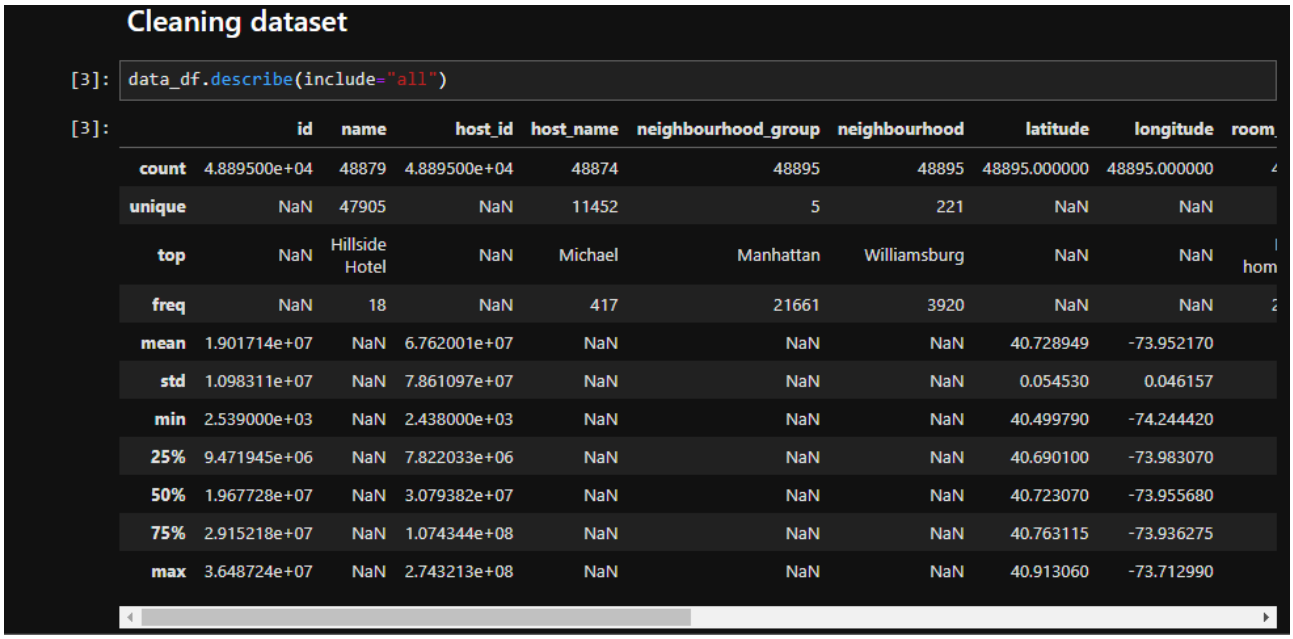


Figure 8.2: Deschiderea proiectului

Dupa deschiderea proiectului, se va rula fiecare celula de cod in parte utilizand SHIFT+ENTER si se pot observa rezultatele obtinute in sectiunea de sub celula rulata.



	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room
count	4.889500e+04	48879	4.889500e+04	48874	48895	48895	48895.000000	48895.000000	4
unique	NaN	47905	NaN	11452	5	221	NaN	NaN	
top	NaN	Hillside Hotel	NaN	Michael	Manhattan	Williamsburg	NaN	NaN	hom
freq	NaN	18	NaN	417	21661	3920	NaN	NaN	2
mean	1.901714e+07	NaN	6.762001e+07	NaN	NaN	NaN	40.728949	-73.952170	
std	1.098311e+07	NaN	7.861097e+07	NaN	NaN	NaN	0.054530	0.046157	
min	2.539000e+03	NaN	2.438000e+03	NaN	NaN	NaN	40.499790	-74.244420	
25%	9.471945e+06	NaN	7.822033e+06	NaN	NaN	NaN	40.690100	-73.983070	
50%	1.967728e+07	NaN	3.079382e+07	NaN	NaN	NaN	40.723070	-73.955680	
75%	2.915218e+07	NaN	1.074344e+08	NaN	NaN	NaN	40.763115	-73.936275	
max	3.648724e+07	NaN	2.743213e+08	NaN	NaN	NaN	40.913060	-73.712990	

Figure 8.3: Rezultat rulare cod dintr-o celula

Chapter 9

Codul original

```
Imports
import pandas as pd
import numpy as np
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing  import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LinearRegression
from sklearn.impute import SimpleImputer
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import cross_val_score
from sklearn import preprocessing
from sklearn import tree
from sklearn import svm
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.ensemble import VotingRegressor
import graphviz
from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering
Data preparation
Reading dataset
data_df = pd.read_csv("AB_NYC_2019.csv")
data_df
data_df.columns
data_df.head
data_df.dtypes
len(data_df["name"].unique())
len(data_df["room_type"].unique())
data_df["reviews_per_month"].unique()
Cleaning dataset
data_df.describe(include="all")
data_df.drop(columns=["id"],inplace=True)
data_df.drop(columns=["host_id"],inplace=True)
data_df["last_review"].unique()
data_df["name"].unique()
Data selection
data_sel=data_df.sample(frac=0.05, replace=True, random_state=1, axis=0)
data_sel
Split dataset
y=data_sel["price"]
```

```
data_sel.drop(columns=["price"],inplace=True)
X=data_sel
X_train,X_test,y_train,y_test=train_test_split(X,y, test_size=0.20,
random_state=42)
X_train
y_train
```

Machine Learning Alg.

```
data_sel.dtypes
numeric_features=["latitude","longitude","minimum_nights",
"number_of_reviews","reviews_per_month","calculated_host_listings_count",
"availability_365"]
nominal_features=["name","host_name","neighbourhood_group",
"neighbourhood","room_type","last_review"]
#AICI FACE SI PARTE DE CLEANING DATASET
preprocessor = ColumnTransformer([
    ("num", Pipeline([("imputer", SimpleImputer(missing_values=np.nan,
    strategy="mean")),
    ("scaler", StandardScaler())]), numeric_features),
    ("nom", Pipeline([("imputer", SimpleImputer(missing_values=np.nan,
    strategy="most_frequent")),
    ("binarizer", OneHotEncoder(handle_unknown="ignore"))]), nominal_features)],
    remainder="drop")
preprocessor.fit(X_train)
X_train=preprocessor.transform(X_train)
X_test=preprocessor.transform(X_test)
```

Linear Regression

```
linreg=LinearRegression()
linreg.fit(X_train,y_train)
scores=cross_val_score(linreg,X_train,y_train,scoring="r2", cv=5)
np.mean(scores)
y_pred=linreg.predict(X_test)
r2_score(y_test,y_pred)
```

Decision Tree

```
X1=data_sel.copy()
X1.drop(columns=["neighbourhood_group"],inplace=True)
X1.drop(columns=["neighbourhood"],inplace=True)
X1.drop(columns=["room_type"],inplace=True)
X1.drop(columns=["last_review"],inplace=True)
X1.drop(columns=["name"],inplace=True)
X1.drop(columns=["host_name"],inplace=True)
X_train1,X_test1,y_train1,y_test1=train_test_split(X1,y, test_size=0.20,
random_state=42)
preprocessor1 = ColumnTransformer([
    ("num", Pipeline([("imputer", SimpleImputer(missing_values=np.nan,
    strategy="mean")),
                      ("scaler", StandardScaler())]), numeric_features)],
    remainder="passthrough"
    )
preprocessor1.fit(X_train1)
X_train1=preprocessor1.transform(X_train1)
X_test1=preprocessor1.transform(X_test1)
clf=tree.DecisionTreeRegressor()
clf.fit(X_train1,y_train1)
scores=cross_val_score(clf,X_train1,y_train1,scoring="r2", cv=5)
np.mean(scores)
clf_depth=tree.DecisionTreeRegressor(max_depth=3)
clf_depth.fit(X_train1,y_train1)
scores=cross_val_score(clf_depth,X_train1,y_train1,scoring="r2", cv=5)
np.mean(scores)
y_pred=clf_depth.predict(X_test1)
```

```
r2_score(y_test1, y_pred)
dot_data = tree.export_graphviz(clf_depth, out_file = "tree2.dot",
feature_names=X1.columns)
graph2 = graphviz.Source(dot_data)
!dot -Tpng tree2.dot -o tree2.png
```

Support Vector Machine

```
regr = svm.SVR()
regr.fit(X_train,y_train)
scores=cross_val_score(regr,X_train,y_train,scoring="r2", cv=5)
np.mean(scores)
regr1 = svm.SVR(kernel='linear',C=100)
regr1.fit(X_train,y_train)
scores=cross_val_score(regr1,X_train,y_train,scoring="r2", cv=5)
np.mean(scores)
y_pred=regr1.predict(X_test)
r2_score(y_test, y_pred)
```

K-Nearest-Neighbour

```
neigh = KNeighborsRegressor(n_neighbors=2)
neigh.fit(X_train,y_train)
scores=cross_val_score(neigh,X_train,y_train,scoring="r2", cv=5)
np.mean(scores)
neigh1 = KNeighborsRegressor(n_neighbors=10)
neigh1.fit(X_train,y_train)
scores=cross_val_score(neigh1,X_train,y_train,scoring="r2", cv=5)
np.mean(scores)
y_pred=neigh1.predict(X_test)
r2_score(y_test, y_pred)
```

Multi-Layer Perceptron

```
mlp = MLPRegressor(random_state=1, max_iter=100)
mlp.fit(X_train, y_train)
scores=cross_val_score(mlp,X_train,y_train,scoring="r2", cv=5)
np.mean(scores)
mlp1 = MLPRegressor(random_state=1, max_iter=100,hidden_layer_sizes=(32,32,32))
mlp1.fit(X_train, y_train)
scores=cross_val_score(mlp1,X_train,y_train,scoring="r2", cv=5)
np.mean(scores)
y_pred=mlp.predict(X_test)
r2_score(y_test, y_pred)
```

Ensemble Learning:Voting Regressor

```
ereg = VotingRegressor(estimators=[('gb', linreg), ('rf', regr), ('lr', neigh)])
ereg = ereg.fit(X_train, y_train)
scores=cross_val_score(ereg,X_train,y_train,scoring="r2", cv=5)
np.mean(scores)
y_pred=ereg.predict(X_test)
r2_score(y_test, y_pred)
```

Bibliography

[1] URL: https://www.kaggle.com/dgomonov/new-york-city-airbnb-open-data.

[2] URL: http://insideairbnb.com/.

[3] URL: https://scikit-learn.org/stable/modules/generated/sklearn.tree.export_graphviz.html?highlight=export_graphviz#sklearn.tree.export_graphviz.

[4] URL: https://docs.anaconda.com/anaconda/install/.

[5] URL: https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html?highlight=pipeline#sklearn.pipeline.Pipeline.

[6] URL: https://scikit-learn.org/stable/modules/generated/sklearn.compose.ColumnTransformer.html?highlight=column%20transformer#sklearn.compose.ColumnTransformer.

[7] URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html?highlight=linear%20regression#sklearn.linear_model.LinearRegression.

[8] URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html?highlight=cros%20val%20score#sklearn.model_selection.cross_val_score.

[9] URL: https://scikit-learn.org/stable/auto_examples/svm/plot_svm_regression.html#support-vector-regression-svr-using-linear-and-non-linear-kernels.

[10] URL: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html?highlight=kneighborsregressor#sklearn.neighbors.KNeighborsRegressor.

[11] URL: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html?highlight=mlpregresso#sklearn.neural_network.MLPRegressor.

[12] URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingRegressor.html?highlight=votingregressor#sklearn.ensemble.VotingRegressor.

[13] URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html?highlight=r2_score#sklearn.metrics.r2_score.

[14] URL: https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation.