## Assessment Cover Page

### Student Full Name: Ilia Grishkin

### Student Number: 2024139

### Module Title: Machine Learning

### Assessment Title: CA1 Project

### Assessment Due Date: 21st April 2024

### Date of Submission: 2nd April 2024

# Content

* [**Introduction**](#section1)
  + [**Literate review**](#section2)
  + [**Research question**](#section3)
  + [**Data sourse and methods**](#section4)
* [**EDA**](#section6)
  + [**Libraries and summary**](#section7)
  + [**Boxplots**](#section8)
  + [**Histograms**](#section9)
  + [**Heatmap**](#section10)
* [**Data preparation**](#section11)
  + [**Drop missing values**](#section12)
  + [**Outliers**](#section13)
  + [**Encoding**](#section14)
* [**Machine Learning Models**](#section230)
  + [**Define a target and split data**](#section231)
  + [**PCA**](#section232)
    - [**Scale dataset**](#section233)
    - [**Create pca object**](#section234)
    - [**Plot varietion ratio**](#section235)
    - [**Calculate number of features**](#section236)
    - [**List features**](#section237)
  + [**PCA Trasform**](#section238)
  + [**Define models**](#section239)
  + [**Hyperparameter Tuning**](#section240)
    - [**Parameters**](#section241)
    - [**GridSearch constructor)**](#section242)
  + [**Evaluation Models PCA+GridSearch**](#section243)
  + [**Evaluation Models GridSearch**](#section244)
* [**Visualisation final accuracy scores**](#section245)

# Housing Price Prediction Using Machine Learning Models

# Introduction

The main problem which may be associated with Ireland’s economy is currently experiencing a severe housing crisis. Housing prices are a big part of a country's economy and affect many other sectors. This issue has deep roots, as was mentioned by Nowlan (2016) and Scuffil (2022), this crisis is going back to the 1900s.

Lyons, R. (2018) point out that the property market is a regular topic of national attention in Ireland. For the majority of house owners, their house is the most valuable asset.

Ireland's economy has been suffering from one of the most severe housing busts of the global financial crisis (Norris and Byrne M. (2017), however in recent years been represented as having recovered economically (Nowicki et al., 2019). Since 2013, Irish house prices have increased by 50%, while rent rates have grown by over 60% according to Byrne M. (2020).

The pricing of houses is affected by many different factors, such as the location of the house, the features of the house and condition Phan, T. (2019).

Many factors may affect the prices of housing. These factors, as well as other parameters such as the materials used for the building, number of bedrooms, living area, location, upcoming projects and proximity were noted by Bourassa, Cantoni and Hoesli (2011).

The prediction of housing sale price may be considered an essential economic metric. The value of a house that grows with time requires the estimated value to be calculated as this value is required for sale, purchase or even mortgage. (Shinde and Gawande, 2018).

## Literate review

Before getting started, we should investigate recent research, methods and results. It will help us to understand which methods we can use and which results we should expect.

1. Aswin (2017) applied 6 different machine learning models to predict house prices in a data set with 2000 records and 10 features. The author used 6 machine learning algorithms, such as Random Forest, Neural Networks, Gradient Boosted, Bagging, Support Vector Machine and Multiple Regression. The best accuracy was performed Random Forest with R-squared value of 90%.
2. Hujia Yu and Jiafu Wu (2016), also were working on a price prediction model. They created regression and classification models which are able to estimate the price of the house given the features. It was concluded that for classification models the best model is the Support Vector Classifier with linear kernel. The model showed an accuracy of 0.6740 and after PCA was performed on the dataset it increased to 0.6913. For the regression problem, the best model is Support Vector Regression with a Gaussian kernel, with an RMSE of 0.5271.
3. Ng A. (2015) explored the use of machine learning methods for London house price prediction. The approach is used to create local models by comparing various regression methods. The Gaussian method was found to be most efficient because of its probabilistic approach to learning and model selection.

## Research question

Each dataset behaves differently. A machine learning model may work with high accuracy in one dataset, but perform poorly in another despite both of them being applied to similar data. Social and economic data is very dynamic in contrast to physical or chemical where approved theory can be reviewed only in unique circumstances. This implies that data related to the economy or social sector should be reviewed preferably as soon as new data has come. This study is going to focus on applying different machine learning algorithms to Irish housing prices datasets, in order to understand which model gives the best accuracy.

## Data sourse and methods

### Data Source

As a data source, we will use an available for public use dataset "daft.ie house price data" published on Kaggle (<https://www.kaggle.com/datasets/eavannan/daftie-house-price-data>). The dataset contains 3869 records about Irish property which were published in 2021 and 2022. Additionally data set includes accounts 22 features which reflect all essential property parameters for creating a machine learning model.

### EDA

Exploratory Data Analysis (EDA). It will help us to understand the structure of the dataset including the size, shape, properties and types of variables. Also, identify patterns and relationships between variables. Additionally, EDA allows us to select appropriate techniques and models for our further analysis.

### Data preparation

After we get the dataset explored using EDA, we can clearly understand which data preparation techniques should be applied.

1. Data Cleaning
2. Data Transformation
3. Handling outliers
4. Feature engineering
5. Feature Selection

### Machine Learning Models

We are going to apply multiple machine learning models to the dataset in order to understand which one gives the highest accuracy. Also, we will try to discover the advantages and disadvantages of an apply of each model to data the datasets related to the property market in Ireland.

1. Linear Regression
2. Ridge Regression
3. Lasso Regression
4. Elasticnet Regression
5. Decision tree Regression
6. Random Forest Regression
7. Gradient Boosting Regression
8. SVR
9. KNN Words in Intoduction section: 530

## EDA

### Libraries and summary

import warnings  
import numpy as np  
import pandas as pd  
import seaborn as sns  
from sklearn import tree  
from sklearn import metrics   
from sklearn.svm import SVC  
import matplotlib.pyplot as plt  
from sklearn import preprocessing  
from sklearn.decomposition import PCA  
from sklearn.linear\_model import Ridge  
from sklearn.linear\_model import Lasso  
from sklearn.linear\_model import ElasticNet  
from sklearn.tree import DecisionTreeRegressor  
from sklearn.preprocessing import MinMaxScaler   
from sklearn.preprocessing import LabelEncoder  
from sklearn.neural\_network import MLPRegressor  
from sklearn.model\_selection import GridSearchCV   
from sklearn.preprocessing import StandardScaler  
from sklearn.linear\_model import LinearRegression  
from sklearn.neighbors import KNeighborsRegressor  
from sklearn.ensemble import RandomForestRegressor  
from sklearn.model\_selection import cross\_val\_score  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import GradientBoostingRegressor  
from sklearn.tree import DecisionTreeClassifier, plot\_tree   
from sklearn.metrics import confusion\_matrix, classification\_report  
from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score  
warnings.filterwarnings("ignore")

df = pd.read\_csv("daft\_ie\_v1.csv")

df.head(2)

id title featuredLevel \  
0 3626025 11 Chestnut Crescent, Bridgemount, Carrigaline... featured   
1 3675175 58 The Glen, Kilnacourt Woods, Portarlington, ... featured   
  
 publishDate price numBedrooms numBathrooms propertyType \  
0 2022-01-28 290000 3 3 End of Terrace   
1 2022-01-28 225000 3 2 Semi-D   
  
 propertySize category ... seller\_name seller\_branch \  
0 96.0 Buy ... Roy Dennehy Dennehy Auctioneers   
1 93.0 Buy ... Marie Kiernan Tom McDonald & Associates   
  
 sellerType m\_totalImages m\_hasVideo m\_hasVirtualTour m\_hasBrochure \  
0 BRANDED\_AGENT 16.0 False False False   
1 BRANDED\_AGENT 33.0 False False False   
  
 ber\_rating longitude latitude   
0 C2 -8.382500 51.822940   
1 C1 -7.177098 53.157465   
  
[2 rows x 22 columns]

df.shape

(3967, 22)

df.describe(include="all")

id title featuredLevel \  
count 3.967000e+03 3967 3967   
unique NaN 3869 3   
top NaN Glebe Manor Estate, Whitegate, Co. Cork standard   
freq NaN 6 3473   
mean 3.629047e+06 NaN NaN   
std 2.502479e+05 NaN NaN   
min 1.092100e+04 NaN NaN   
25% 3.654066e+06 NaN NaN   
50% 3.673795e+06 NaN NaN   
75% 3.685990e+06 NaN NaN   
max 3.695402e+06 NaN NaN   
  
 publishDate price numBedrooms numBathrooms propertyType \  
count 3967 3.967000e+03 3967.000000 3967.000000 3967   
unique 53 NaN NaN NaN 11   
top 2022-01-28 NaN NaN NaN Detached   
freq 1220 NaN NaN NaN 1079   
mean NaN 3.416734e+05 3.126292 2.077136 NaN   
std NaN 2.703582e+05 1.230570 1.167772 NaN   
min NaN 2.000000e+04 1.000000 1.000000 NaN   
25% NaN 1.950000e+05 2.000000 1.000000 NaN   
50% NaN 2.750000e+05 3.000000 2.000000 NaN   
75% NaN 3.950000e+05 4.000000 3.000000 NaN   
max NaN 4.500000e+06 23.000000 23.000000 NaN   
  
 propertySize category ... seller\_name seller\_branch \  
count 3612.000000 3967 ... 3967 3967   
unique NaN 2 ... 1295 836   
top NaN Buy ... Isabelle Crean BidX1   
freq NaN 3869 ... 44 107   
mean 132.673865 NaN ... NaN NaN   
std 295.246580 NaN ... NaN NaN   
min 1.000000 NaN ... NaN NaN   
25% 78.000000 NaN ... NaN NaN   
50% 103.000000 NaN ... NaN NaN   
75% 137.000000 NaN ... NaN NaN   
max 8600.000000 NaN ... NaN NaN   
  
 sellerType m\_totalImages m\_hasVideo m\_hasVirtualTour \  
count 3967 3967.000000 3967 3967   
unique 3 NaN 2 2   
top BRANDED\_AGENT NaN False False   
freq 2929 NaN 3236 3606   
mean NaN 18.448954 NaN NaN   
std NaN 10.537226 NaN NaN   
min NaN 0.000000 NaN NaN   
25% NaN 12.000000 NaN NaN   
50% NaN 17.000000 NaN NaN   
75% NaN 23.000000 NaN NaN   
max NaN 104.000000 NaN NaN   
  
 m\_hasBrochure ber\_rating longitude latitude   
count 3967 3967 3967.000000 3967.000000   
unique 2 18 NaN NaN   
top False C2 NaN NaN   
freq 3835 475 NaN NaN   
mean NaN NaN -7.389964 53.133816   
std NaN NaN 1.865587 0.716435   
min NaN NaN -100.445882 39.783730   
25% NaN NaN -8.426627 52.666116   
50% NaN NaN -7.050391 53.303346   
75% NaN NaN -6.298945 53.438722   
max NaN NaN -6.028016 55.299693   
  
[11 rows x 22 columns]

df.dtypes

id int64  
title object  
featuredLevel object  
publishDate object  
price int64  
numBedrooms int64  
numBathrooms int64  
propertyType object  
propertySize float64  
category object  
AMV\_price int64  
sellerId float64  
seller\_name object  
seller\_branch object  
sellerType object  
m\_totalImages float64  
m\_hasVideo bool  
m\_hasVirtualTour bool  
m\_hasBrochure bool  
ber\_rating object  
longitude float64  
latitude float64  
dtype: object

df.isnull().sum()

id 0  
title 0  
featuredLevel 0  
publishDate 0  
price 0  
numBedrooms 0  
numBathrooms 0  
propertyType 0  
propertySize 355  
category 0  
AMV\_price 0  
sellerId 0  
seller\_name 0  
seller\_branch 0  
sellerType 0  
m\_totalImages 0  
m\_hasVideo 0  
m\_hasVirtualTour 0  
m\_hasBrochure 0  
ber\_rating 0  
longitude 0  
latitude 0  
dtype: int64

categorical\_indices = [1, 2, 3, 5, 6, 7, 9, 10, 12, 14, 15, 16, 17]  
rooms\_index = [5, 6]  
selected\_columns2 = df.iloc[:, rooms\_index]

def get\_unique\_values(df, categorical\_indices):  
 unique\_values\_dict = {}  
 for col\_index in categorical\_indices:  
 column\_name = df.columns[col\_index]  
 unique\_values = df.iloc[:, col\_index].unique()  
 unique\_values\_dict[column\_name] = unique\_values  
 return unique\_values\_dict  
  
unique\_values = get\_unique\_values(df, categorical\_indices)  
  
for column\_name, values in unique\_values.items():  
 print(f"Column '{column\_name}' unique values")  
 print(values)  
 print()

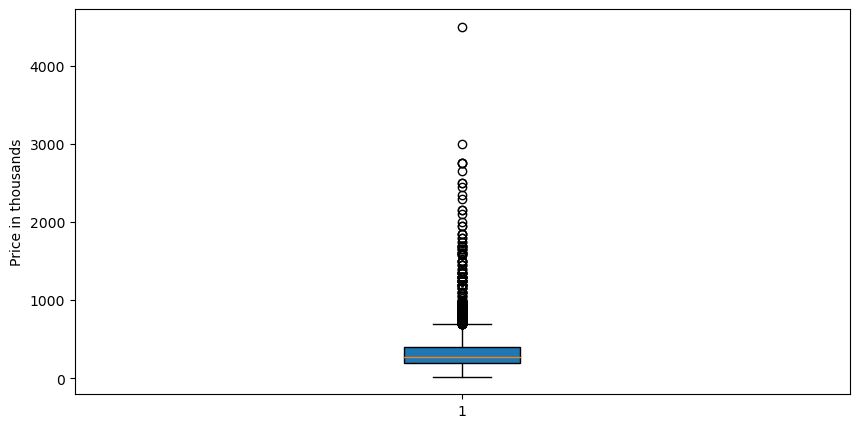
Column 'title' unique values  
['11 Chestnut Crescent, Bridgemount, Carrigaline, Co. Cork'  
 '58 The Glen, Kilnacourt Woods, Portarlington, Co. Laois'  
 '16 Dodderbrook Park, Ballycullen, Dublin 24' ...  
 '69 McAuley Drive, Artane, Artane, Dublin 5'  
 'School Land, Ballinalee, Co. Longford'  
 '14 Coolmagort Ave, Beaufort, Killarney, Co. Kerry']  
  
Column 'featuredLevel' unique values  
['featured' 'premium' 'standard']  
  
Column 'publishDate' unique values  
['2022-01-28' '2022-01-27' '2022-01-30' '2022-01-26' '2022-01-12'  
 '2022-01-14' '2022-01-11' '2022-01-25' '2022-01-10' '2022-01-07'  
 '2022-01-06' '2022-01-24' '2022-01-29' '2022-01-21' '2022-01-20'  
 '2022-01-19' '2022-01-18' '2022-01-17' '2021-12-17' '2022-01-05'  
 '2022-01-04' '2022-01-03' '2022-01-15' '2021-12-31' '2021-12-01'  
 '2022-01-23' '2022-01-09' '2022-01-01' '2021-12-15' '2021-12-30'  
 '2021-12-23' '2021-12-20' '2022-01-22' '2022-01-16' '2022-01-13'  
 '2022-01-08' '2021-12-29' '2021-12-28' '2021-12-27' '2021-12-24'  
 '2021-12-22' '2021-12-21' '2021-12-18' '2021-12-16' '2021-12-14'  
 '2021-12-13' '2021-12-11' '2021-12-10' '2021-12-09' '2021-12-08'  
 '2021-12-07' '2021-12-06' '2021-12-04']  
  
Column 'numBedrooms' unique values  
[ 3 4 6 2 5 7 1 9 8 13 10 16 14 12 23]  
  
Column 'numBathrooms' unique values  
[ 3 2 1 6 4 5 7 13 8 12 10 9 11 23]  
  
Column 'propertyType' unique values  
['End of Terrace' 'Semi-D' 'Terrace' 'Detached' 'Apartment' 'Bungalow'  
 'Townhouse' 'Duplex' 'Site' 'Studio' 'House']  
  
Column 'category' unique values  
['Buy' 'New Homes']  
  
Column 'AMV\_price' unique values  
[0 1]  
  
Column 'seller\_name' unique values  
['Roy Dennehy' 'Marie Kiernan' 'Moovingo' ... 'Rooney Auctioneers'  
 "Paul O'Shea" 'Jackie Horan']  
  
Column 'sellerType' unique values  
['BRANDED\_AGENT' 'UNBRANDED\_AGENT' 'PRIVATE\_USER']  
  
Column 'm\_totalImages' unique values  
[ 16. 33. 38. 22. 5. 20. 37. 21. 24. 15. 28. 14. 43. 25.  
 29. 42. 26. 8. 18. 27. 17. 13. 32. 12. 9. 23. 48. 31.  
 30. 35. 19. 40. 7. 39. 11. 34. 2. 10. 36. 4. 6. 1.  
 3. 51. 0. 50. 60. 44. 56. 63. 52. 47. 55. 46. 53. 41.  
 68. 71. 45. 66. 57. 69. 54. 104. 67. 59. 49. 93. 81. 75.  
 88. 80. 65. 87.]  
  
Column 'm\_hasVideo' unique values  
[False True]  
  
Column 'm\_hasVirtualTour' unique values  
[False True]

### Boxplots

First of all, let's take make a boxplot for our targer varieble "price". However, the outliers may affect clear understanding, so I would recommend to update this and other charts after running preparation section. As we can see in updated boxplot:

* the mean is between 200k and 300k approximately in the middle, that indicates a value of 250k.
* the second and third quartiles which include the majority of data are placed between 200k and 375k more or less. It gives us a range of prices for most of the properties.
* also we can see first and fourth quatiles which show us the whole range of prices except outliers left on the top of boxblot.

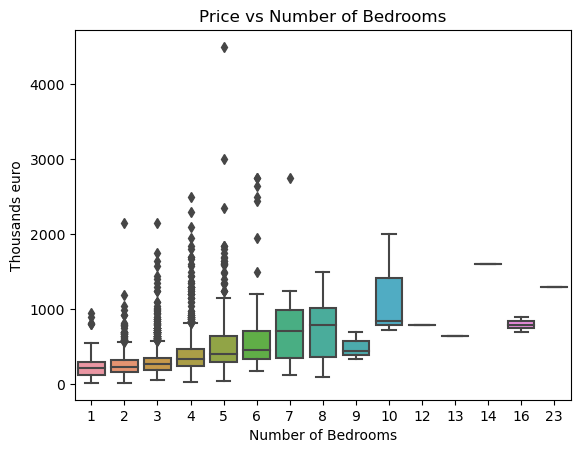
prices\_boxplot = df['price'] / 1000  
  
plt.figure(figsize=(10, 5))  
boxplot = plt.boxplot(prices\_boxplot, vert=True, patch\_artist=True)  
plt.ylabel('Price in thousands')  
plt.show()



We can see that two boxplots below show the distribution for properties with different number of badrooms and bathrooms. It is seen how price is changing according to the number of rooms.

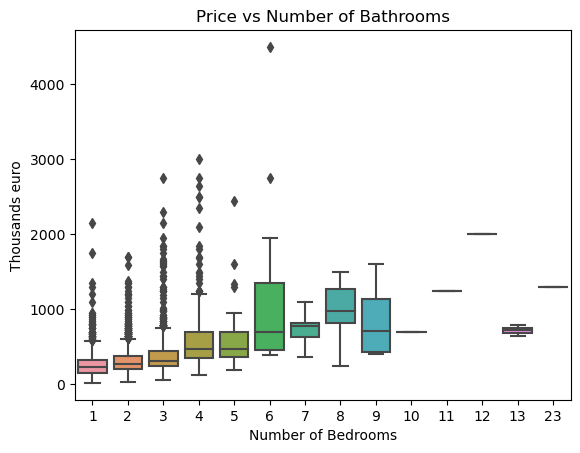
sns.boxplot(df.numBedrooms, prices\_boxplot)   
plt.title('Price vs Number of Bedrooms')  
plt.ylabel('Thousands euro')  
plt.xlabel('Number of Bedrooms')

Text(0.5, 0, 'Number of Bedrooms')



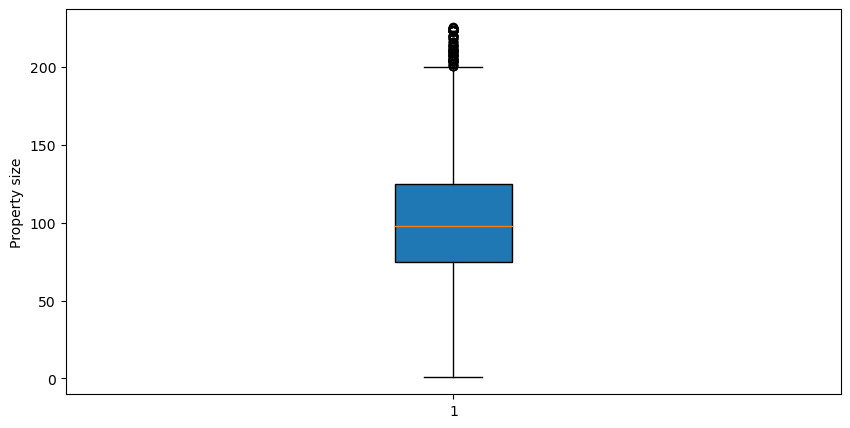
sns.boxplot(df.numBathrooms, prices\_boxplot)   
plt.title('Price vs Number of Bathrooms')  
plt.ylabel('Thousands euro')  
plt.xlabel('Number of Bedrooms')

Text(0.5, 0, 'Number of Bedrooms')



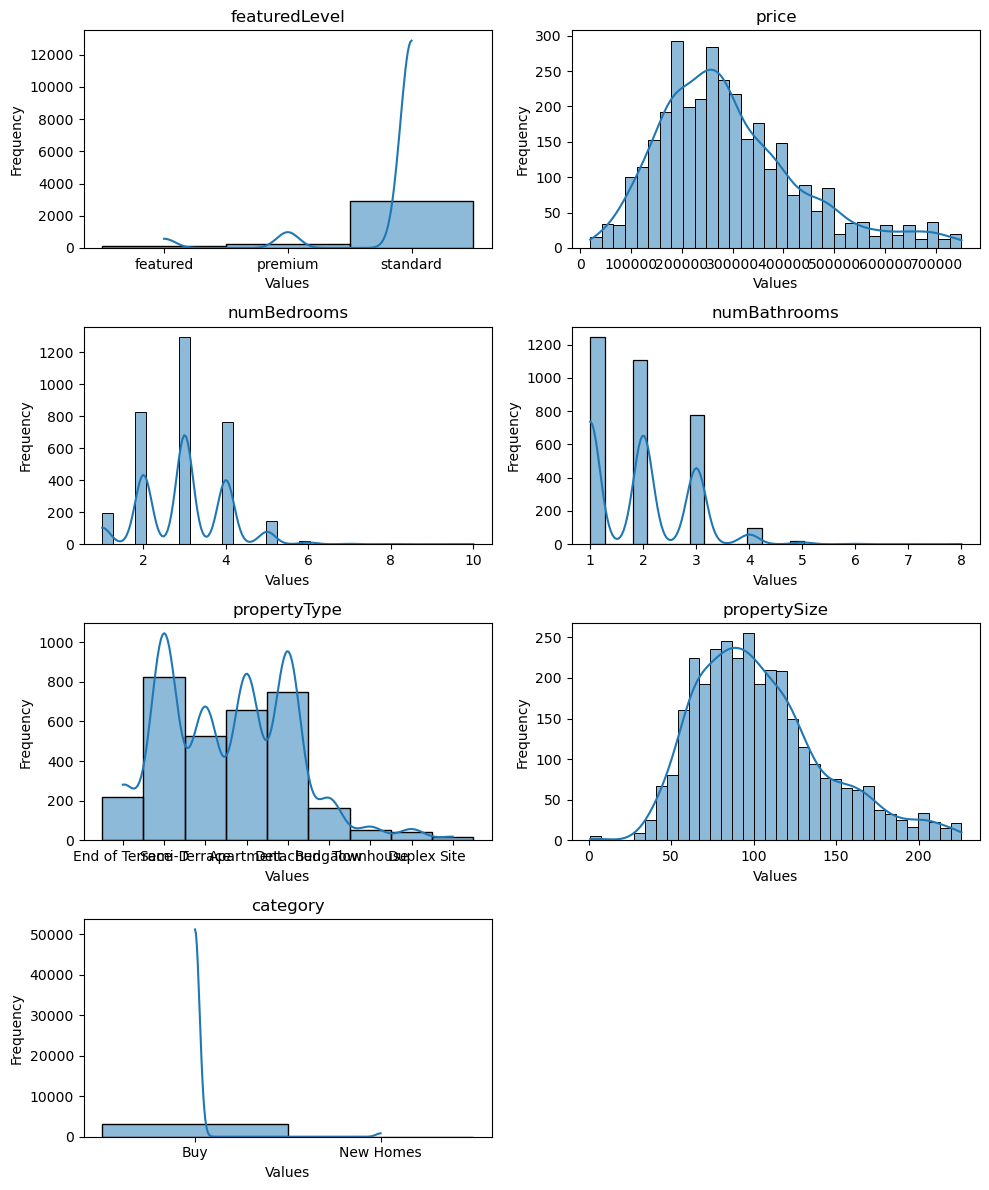
As we can see the avarage property size is 100. Additionally, the size for the most properties in range between 75 and 125.

prices\_boxplot = df['propertySize']  
  
plt.figure(figsize=(10, 5))  
boxplot = plt.boxplot(prices\_boxplot, vert=True, patch\_artist=True)  
plt.ylabel('Property size')  
plt.show()



### Histograms

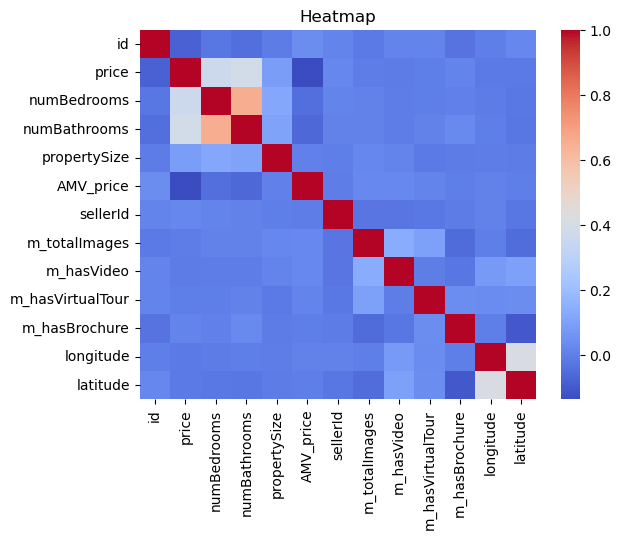
numeric\_index = [2, 4, 5, 6, 7, 8, 9]  
selected\_columns = df.iloc[:, numeric\_index]  
  
num\_cols = 2   
  
num\_features = len(numeric\_index)  
num\_rows = (num\_features - 1) // num\_cols + 1  
  
fig, axes = plt.subplots(num\_rows, num\_cols, figsize=(10, 3 \* num\_rows))  
  
axes = axes.flatten()  
  
for i, col\_index in enumerate(numeric\_index):  
 ax = axes[i]  
 col\_name = df.columns[col\_index]  
 sns.histplot(df.iloc[:, col\_index], ax=ax, kde=True)  
 ax.set\_title(col\_name)  
 ax.set\_xlabel('Values')  
 ax.set\_ylabel('Frequency')  
  
for i in range(num\_features, num\_rows \* num\_cols):  
 fig.delaxes(axes[i])  
   
plt.tight\_layout()  
plt.show()



### Heatmap

A heatmap shows correlation between all values in dataset. We can see that there are 4 feachers which have biggest impact on our target variable "price" such as number of bedrooms, number of bathrooms and property size.

sns.heatmap(df.corr(), cmap='coolwarm')  
plt.title('Heatmap')  
plt.show()



## Data preparation

#### Drop missing values

df = df.dropna()

df.isnull().sum()

id 0  
title 0  
featuredLevel 0  
publishDate 0  
price 0  
numBedrooms 0  
numBathrooms 0  
propertyType 0  
propertySize 0  
category 0  
AMV\_price 0  
sellerId 0  
seller\_name 0  
seller\_branch 0  
sellerType 0  
m\_totalImages 0  
m\_hasVideo 0  
m\_hasVirtualTour 0  
m\_hasBrochure 0  
ber\_rating 0  
longitude 0  
latitude 0  
dtype: int64

#### Outliers

##### Define percentage of dataset we want to keep which doesn't include outliers, in this case we keep 95% and cut just 5% whith outliers.

outlier = df.price.quantile(0.95)  
df = df.loc[df.price < outlier, :]

sns.distplot(df.price)

sns.distplot(df.propertySize)

outlier = df.propertySize.quantile(0.95)  
df = df.loc[df.propertySize < outlier, :]

#### Encoding

label\_encoder = LabelEncoder()  
df\_encoded = df  
  
df\_encoded['title'] = label\_encoder.fit\_transform(df['title'])  
df\_encoded['featuredLevel'] = label\_encoder.fit\_transform(df['featuredLevel'])  
df\_encoded['propertyType'] = label\_encoder.fit\_transform(df['propertyType'])  
df\_encoded['publishDate'] = label\_encoder.fit\_transform(df['publishDate'])  
df\_encoded['category'] = label\_encoder.fit\_transform(df['category'])  
df\_encoded['seller\_name'] = label\_encoder.fit\_transform(df['seller\_name'])  
df\_encoded['seller\_branch'] = label\_encoder.fit\_transform(df['seller\_branch'])  
df\_encoded['sellerType'] = label\_encoder.fit\_transform(df['sellerType'])  
df\_encoded['m\_hasVideo'] = label\_encoder.fit\_transform(df['m\_hasVideo'])  
df\_encoded['m\_hasVirtualTour'] = label\_encoder.fit\_transform(df['m\_hasVirtualTour'])  
df\_encoded['m\_hasBrochure'] = label\_encoder.fit\_transform(df['m\_hasBrochure'])  
df\_encoded['ber\_rating'] = label\_encoder.fit\_transform(df['ber\_rating'])  
df\_encoded['longitude'] = label\_encoder.fit\_transform(df['longitude'])  
df\_encoded['latitude'] = label\_encoder.fit\_transform(df['latitude'])

## Machine Learning Models

After we prepared our data and made sure it is ready for being analised, we are ready to set our models. However, before that we are going to make PCA.

#### Define a target and split data

X=df\_encoded.drop(columns=['price'],axis = 1)  
y=df\_encoded['price']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=42)

### PCA

#### Scale dataset

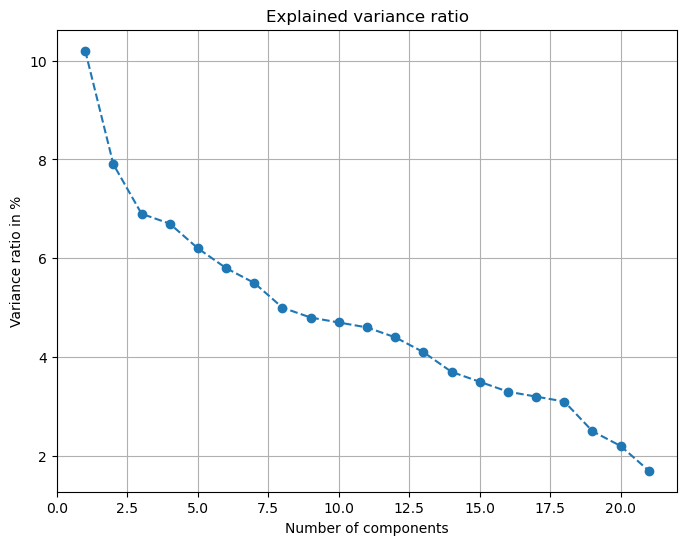
scaler = StandardScaler()  
X\_train\_scaled = scaler.fit\_transform(X)

#### Create pca object

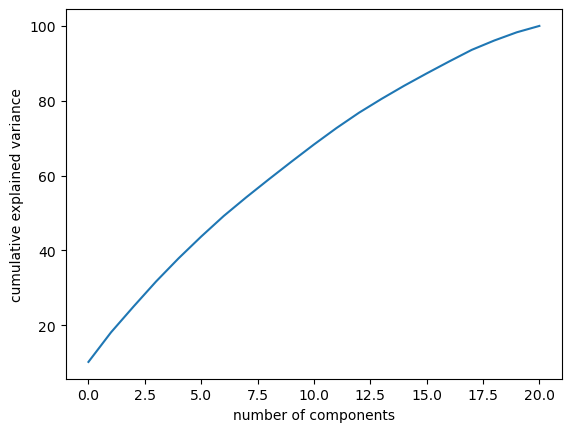
pca = PCA()  
pca.fit(X\_train\_scaled)  
pca\_data = pca.transform(X\_train\_scaled)

#### Plot varietion ratio for each component

variations = np.round(pca.explained\_variance\_ratio\_\* 100, decimals=1)  
labels = [str(x) for x in range (1, len(variations)+1)]  
  
plt.figure(figsize=(8, 6))  
plt.plot(range(1, len(variations) + 1),   
 variations, marker='o', linestyle='--')  
plt.xlabel('Number of components')  
plt.ylabel('Variance ratio in %')  
plt.title('Explained variance ratio')  
plt.grid(True)  
plt.show()



plt.plot(np.cumsum(variations))   
plt.xlabel('number of components')   
plt.ylabel('cumulative explained variance');



#### Calculate number of features which represent a sample with selected accuracy

variance\_ratio\_percentage = 99  
variance\_ratio = np.cumsum(variations)  
num\_components = np.argmax(variance\_ratio >= variance\_ratio\_percentage) + 1  
print("Percentage of variance:", variance\_ratio\_percentage)  
print("Number of components:", num\_components)

Percentage of variance: 99  
Number of components: 21

#### List features and their ratio from calculated number

abs\_loadings = np.abs(pca.components\_)  
importance\_scores = np.sum(abs\_loadings, axis=0)  
variance\_ratios = variations  
  
feature\_variance = {}  
for i, (feature\_name, variance\_ratio) in enumerate(zip(df.columns, variance\_ratios)):  
 feature\_variance[i, feature\_name] = variance\_ratio  
  
sorted\_feature\_variance = dict(sorted(feature\_variance.items(), key=lambda item: item[1], reverse=True))  
  
most\_important\_features = list(sorted\_feature\_variance.items())[:num\_components]  
  
for (index, feature), variance\_ratio in most\_important\_features:  
 print(f"Index: {index}, Feature: {feature}, Variance Ratio: {variance\_ratio:.1f}%")

Index: 0, Feature: id, Variance Ratio: 10.2%  
Index: 1, Feature: title, Variance Ratio: 7.9%  
Index: 2, Feature: featuredLevel, Variance Ratio: 6.9%  
Index: 3, Feature: publishDate, Variance Ratio: 6.7%  
Index: 4, Feature: price, Variance Ratio: 6.2%  
Index: 5, Feature: numBedrooms, Variance Ratio: 5.8%  
Index: 6, Feature: numBathrooms, Variance Ratio: 5.5%  
Index: 7, Feature: propertyType, Variance Ratio: 5.0%  
Index: 8, Feature: propertySize, Variance Ratio: 4.8%  
Index: 9, Feature: category, Variance Ratio: 4.7%  
Index: 10, Feature: AMV\_price, Variance Ratio: 4.6%  
Index: 11, Feature: sellerId, Variance Ratio: 4.4%  
Index: 12, Feature: seller\_name, Variance Ratio: 4.1%  
Index: 13, Feature: seller\_branch, Variance Ratio: 3.7%  
Index: 14, Feature: sellerType, Variance Ratio: 3.5%  
Index: 15, Feature: m\_totalImages, Variance Ratio: 3.3%  
Index: 16, Feature: m\_hasVideo, Variance Ratio: 3.2%  
Index: 17, Feature: m\_hasVirtualTour, Variance Ratio: 3.1%  
Index: 18, Feature: m\_hasBrochure, Variance Ratio: 2.5%  
Index: 19, Feature: ber\_rating, Variance Ratio: 2.2%  
Index: 20, Feature: longitude, Variance Ratio: 1.7%

### PCA Transform

pca = PCA(n\_components=20)  
X\_train\_pca = pca.fit\_transform(X\_train)  
X\_test\_pca = pca.transform(X\_test)  
print("original shape X\_train:", X\_train.shape)  
print("original shape X\_test:", X\_test.shape)  
print("transformed shape X\_train\_pca:", X\_train\_pca.shape)  
print("transformed shape X\_test\_pca:", X\_test\_pca.shape)

original shape X\_train: (2437, 21)  
original shape X\_test: (813, 21)  
transformed shape X\_train\_pca: (2437, 21)  
transformed shape X\_test\_pca: (813, 21)

scaler = StandardScaler()  
X\_train\_scaled = scaler.fit\_transform(X\_train\_pca)  
X\_test\_scaled = scaler.transform(X\_test\_pca)

### Defining models

#### Linear Regression

lr\_model = LinearRegression()

#### Ridge Regression

ridge\_model = Ridge()

#### Lasso Regression

lasso\_model = Lasso()

#### ElasticNet Regression

elasticnet\_model = ElasticNet()

#### Decision Tree Regression

dt\_model = DecisionTreeRegressor()

#### Random Forest Regression

rf\_model = RandomForestRegressor()

#### Gradient Boosting Regression

gb\_model = GradientBoostingRegressor()

#### Support Vector Regression

svr\_model = SVR(kernel='linear')

#### K-Nearest Neighbors Regression

knn\_model = KNeighborsRegressor()

## Hyperparameter Tuning

Each machine learning model has their own parameters by default. Hyperparameter tuning allows us to set multiple values of each parameter thereby creating new modified models. The more parameters the model has the more variation we can create and track the best parameters for our model.

As a result, we should get some information about which combination of parameters gives us the best accuracy. So, after we will be able to change the default parameters for our model.

At this stage we are going to define parametrs for each model which we want to test. Paramaters for each model may vary, so before set parametrs we will check documentantion for the library. Links are attached.

### Parameters

#### Linear Regression

[**https://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.LinearRegression.html**](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html) Accessed on April 4, 2024.

hyp\_params\_linear\_reg = {}

#### Ridge Regression

[**https://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.Ridge.html**](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html) Accessed on April 4, 2024.

hyp\_params\_ridge\_reg = {  
 'alpha': [0.1, 1, 10]  
}

#### Lasso Regression

<https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html> Accessed on April 4, 2024.

hyp\_params\_lasso\_reg = {  
 'alpha': [0.1, 1, 10]  
}

#### ElasticNet Regression

<https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html> Accessed on April 4, 2024.

hyp\_params\_elasticnet\_reg = {  
 'alpha': [0.1, 1, 10],  
 'l1\_ratio': [0.1, 0.5, 0.9]  
}

#### Decision Tree Regression

<https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html> Accessed on April 4, 2024.

hyp\_params\_decision\_tree\_reg = {  
 'max\_depth': [3, 4, 5],  
 'min\_samples\_split': [2, 5, 10]  
}

#### Random Forest Regression

<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html> Accessed on April 4, 2024.

hyp\_params\_random\_forest\_reg = {  
 'n\_estimators': [50, 100, 200],  
 'max\_depth': [3, 5, 7]  
}

#### Gradient Boosting Regression

<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html> Accessed on April 4, 2024.

hyp\_params\_gradient\_boosting\_reg = {  
 'n\_estimators': [100, 200],  
 'learning\_rate': [0.1, 0.5],  
 'max\_depth': [ 5, 7]  
}

#### Support Vector Regression

<https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html> Accessed on April 4, 2024.

hyp\_params\_svr = {  
 'C': [0.1],  
 'epsilon': [0.01, 0.1]  
}

#### KNN Regression

<https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html> Accessed on April 4, 2024.

hyp\_params\_knn\_reg = {  
 'n\_neighbors': [3, 5, 7],  
 'weights': ['uniform', 'distance']  
}

### Set Grid Search

After we defined our target, models and parametrs for search, we can set our GridSearch constructor:

* **estimator** - as a estimator we are going to use models which we created before.
* **param\_grid** - it parametrs which we want to test, they were set in a previous section.
* **scoring** - scoring metrics we are want to include in the report.
* **refit** - metric we want to use for extracting the best score acroos all tests.
* **cv** - defines the number of cross validation folds.
* **verbose** - indicates verbosity of the output during the grid search process.

# Linear Regression  
GS\_lr\_model = GridSearchCV(estimator = lr\_model,   
 param\_grid = hyp\_params\_linear\_reg,  
 scoring = ["r2", 'neg\_root\_mean\_squared\_error'],  
 refit = "r2",  
 cv = 5,  
 verbose = 4)  
# Ridge Regression  
GS\_ridge\_model = GridSearchCV(estimator = ridge\_model,   
 param\_grid = hyp\_params\_ridge\_reg,  
 scoring = ["r2", 'neg\_root\_mean\_squared\_error'],  
 refit = "r2",  
 cv = 5,  
 verbose = 4)  
# Lasso Regression  
GS\_lasso\_model = GridSearchCV(estimator = lasso\_model,   
 param\_grid = hyp\_params\_lasso\_reg,  
 scoring = ["r2", 'neg\_root\_mean\_squared\_error'],  
 refit = "r2",  
 cv = 5,  
 verbose = 4)  
# ElasticNet Regression  
GS\_elasticnet\_model = GridSearchCV(estimator = elasticnet\_model,   
 param\_grid = hyp\_params\_elasticnet\_reg,  
 scoring = ["r2", 'neg\_root\_mean\_squared\_error'],  
 refit = "r2",  
 cv = 5,  
 verbose = 4)  
# Decision Tree Regression  
GS\_dt\_model = GridSearchCV(estimator = dt\_model,   
 param\_grid = hyp\_params\_decision\_tree\_reg,  
 scoring = ["r2", 'neg\_root\_mean\_squared\_error'],  
 refit = "r2",  
 cv = 5,  
 verbose = 4)  
# Random Forest Regression  
GS\_rf\_model = GridSearchCV(estimator = rf\_model,   
 param\_grid = hyp\_params\_random\_forest\_reg,  
 scoring = ["r2", 'neg\_root\_mean\_squared\_error'],  
 refit = "r2",  
 cv = 5,  
 verbose = 4)  
# Gradient Boosting Regression  
GS\_gb\_model = GridSearchCV(estimator = gb\_model,   
 param\_grid = hyp\_params\_gradient\_boosting\_reg,  
 scoring = ["r2", 'neg\_root\_mean\_squared\_error'],  
 refit = "r2",  
 cv = 3,  
 verbose = 4)  
# SVR  
GS\_svr\_model = GridSearchCV(estimator = svr\_model,   
 param\_grid = hyp\_params\_svr,  
 scoring = ["r2", 'neg\_root\_mean\_squared\_error'],  
 refit = "r2",  
 cv = 3,  
 verbose = 4)  
# KNN Regression  
GS\_knn\_model = GridSearchCV(estimator = knn\_model,   
 param\_grid = hyp\_params\_knn\_reg,  
 scoring = ["r2", 'neg\_root\_mean\_squared\_error'],  
 refit = "r2",  
 cv = 5,  
 verbose = 4)

## Evaluation Models with PCA and Grid Search

Now we are ready to evaluate our model. To record results we are going to create an empty data frame, so we can add result in it after each evaluation. Additionally, we will define variables to store metrics which we want to analise.

final\_accuracy\_PCA\_GRID = pd.DataFrame(columns=['Model', 'Accuracy(R2)', 'Mean Squared Error', 'Mean Absolute Error', 'Best estimator', 'Best parametrs'])

final\_accuracy\_GRID = pd.DataFrame(columns=['Model', 'Accuracy(R2)', 'Mean Squared Error', 'Mean Absolute Error', 'Best estimator', 'Best parametrs'])

#### Grid Search (Linear Regression)

GS\_lr\_model.fit(X\_train\_scaled, y\_train)  
y\_pred = GS\_lr\_model.predict(X\_test\_scaled)  
  
best\_accuracy\_score = GS\_lr\_model.best\_score\_  
best\_estimator = GS\_lr\_model.best\_estimator\_  
best\_parametrs = GS\_lr\_model.best\_params\_  
mse = mean\_squared\_error(y\_test, y\_pred)  
mae = mean\_absolute\_error(y\_test, y\_pred)  
final\_accuracy\_PCA\_GRID = final\_accuracy\_PCA\_GRID.append({'Model': 'Linear Regression', 'Accuracy(R2)': best\_accuracy\_score, 'Mean Squared Error': mse, 'Mean Absolute Error': mae, 'Best estimator': best\_estimator, 'Best parametrs': best\_parametrs }, ignore\_index=True)

Fitting 5 folds for each of 1 candidates, totalling 5 fits  
[CV 1/5] END neg\_root\_mean\_squared\_error: (test=-128061.824) r2: (test=0.217) total time= 0.0s  
[CV 2/5] END neg\_root\_mean\_squared\_error: (test=-127857.977) r2: (test=0.190) total time= 0.0s  
[CV 3/5] END neg\_root\_mean\_squared\_error: (test=-122256.146) r2: (test=0.182) total time= 0.0s  
[CV 4/5] END neg\_root\_mean\_squared\_error: (test=-126491.510) r2: (test=0.104) total time= 0.0s  
[CV 5/5] END neg\_root\_mean\_squared\_error: (test=-135614.084) r2: (test=0.131) total time= 0.0s

#### Grid Search (Ridge Regression)

GS\_ridge\_model.fit(X\_train\_scaled, y\_train)  
y\_pred = GS\_ridge\_model.predict(X\_test\_scaled)  
  
best\_accuracy\_score = GS\_ridge\_model.best\_score\_  
best\_estimator = GS\_ridge\_model.best\_estimator\_  
best\_parametrs = GS\_ridge\_model.best\_params\_  
mse = mean\_squared\_error(y\_test, y\_pred)  
mae = mean\_absolute\_error(y\_test, y\_pred)  
final\_accuracy\_PCA\_GRID = final\_accuracy\_PCA\_GRID.append({'Model': 'Ridge Regression', 'Accuracy(R2)': best\_accuracy\_score, 'Mean Squared Error': mse, 'Mean Absolute Error': mae, 'Best estimator': best\_estimator, 'Best parametrs': best\_parametrs }, ignore\_index=True)

Fitting 5 folds for each of 3 candidates, totalling 15 fits  
[CV 1/5] END alpha=0.1; neg\_root\_mean\_squared\_error: (test=-128062.062) r2: (test=0.217) total time= 0.0s  
[CV 2/5] END alpha=0.1; neg\_root\_mean\_squared\_error: (test=-127858.071) r2: (test=0.190) total time= 0.0s  
[CV 3/5] END alpha=0.1; neg\_root\_mean\_squared\_error: (test=-122256.103) r2: (test=0.182) total time= 0.0s  
[CV 4/5] END alpha=0.1; neg\_root\_mean\_squared\_error: (test=-126490.991) r2: (test=0.104) total time= 0.0s  
[CV 5/5] END alpha=0.1; neg\_root\_mean\_squared\_error: (test=-135613.878) r2: (test=0.131) total time= 0.0s  
[CV 1/5] END alpha=1; neg\_root\_mean\_squared\_error: (test=-128064.209) r2: (test=0.217) total time= 0.0s  
[CV 2/5] END alpha=1; neg\_root\_mean\_squared\_error: (test=-127858.923) r2: (test=0.190) total time= 0.0s  
[CV 3/5] END alpha=1; neg\_root\_mean\_squared\_error: (test=-122255.716) r2: (test=0.182) total time= 0.0s  
[CV 4/5] END alpha=1; neg\_root\_mean\_squared\_error: (test=-126486.334) r2: (test=0.104) total time= 0.0s  
[CV 5/5] END alpha=1; neg\_root\_mean\_squared\_error: (test=-135612.022) r2: (test=0.131) total time= 0.0s  
[CV 1/5] END alpha=10; neg\_root\_mean\_squared\_error: (test=-128085.888) r2: (test=0.217) total time= 0.0s  
[CV 2/5] END alpha=10; neg\_root\_mean\_squared\_error: (test=-127867.703) r2: (test=0.190) total time= 0.0s  
[CV 3/5] END alpha=10; neg\_root\_mean\_squared\_error: (test=-122252.204) r2: (test=0.182) total time= 0.0s  
[CV 4/5] END alpha=10; neg\_root\_mean\_squared\_error: (test=-126440.422) r2: (test=0.104) total time= 0.0s  
[CV 5/5] END alpha=10; neg\_root\_mean\_squared\_error: (test=-135593.902) r2: (test=0.131) total time= 0.0s

#### Grid Search (Lasso Regression)

GS\_lasso\_model.fit(X\_train\_scaled, y\_train)  
y\_pred = GS\_lasso\_model.predict(X\_test\_scaled)  
  
best\_accuracy\_score = GS\_lasso\_model.best\_score\_  
best\_estimator = GS\_lasso\_model.best\_estimator\_  
best\_parametrs = GS\_lasso\_model.best\_params\_  
mse = mean\_squared\_error(y\_test, y\_pred)  
mae = mean\_absolute\_error(y\_test, y\_pred)  
final\_accuracy\_PCA\_GRID = final\_accuracy\_PCA\_GRID.append({'Model': 'Lasso Regression', 'Accuracy(R2)': best\_accuracy\_score, 'Mean Squared Error': mse, 'Mean Absolute Error': mae, 'Best estimator': best\_estimator, 'Best parametrs': best\_parametrs }, ignore\_index=True)

Fitting 5 folds for each of 3 candidates, totalling 15 fits  
[CV 1/5] END alpha=0.1; neg\_root\_mean\_squared\_error: (test=-128061.839) r2: (test=0.217) total time= 0.0s  
[CV 2/5] END alpha=0.1; neg\_root\_mean\_squared\_error: (test=-127857.975) r2: (test=0.190) total time= 0.0s  
[CV 3/5] END alpha=0.1; neg\_root\_mean\_squared\_error: (test=-122256.120) r2: (test=0.182) total time= 0.0s  
[CV 4/5] END alpha=0.1; neg\_root\_mean\_squared\_error: (test=-126491.451) r2: (test=0.104) total time= 0.0s  
[CV 5/5] END alpha=0.1; neg\_root\_mean\_squared\_error: (test=-135614.075) r2: (test=0.131) total time= 0.0s  
[CV 1/5] END alpha=1; neg\_root\_mean\_squared\_error: (test=-128061.972) r2: (test=0.217) total time= 0.0s  
[CV 2/5] END alpha=1; neg\_root\_mean\_squared\_error: (test=-127857.956) r2: (test=0.190) total time= 0.0s  
[CV 3/5] END alpha=1; neg\_root\_mean\_squared\_error: (test=-122255.884) r2: (test=0.182) total time= 0.0s  
[CV 4/5] END alpha=1; neg\_root\_mean\_squared\_error: (test=-126490.922) r2: (test=0.104) total time= 0.0s  
[CV 5/5] END alpha=1; neg\_root\_mean\_squared\_error: (test=-135613.992) r2: (test=0.131) total time= 0.0s  
[CV 1/5] END alpha=10; neg\_root\_mean\_squared\_error: (test=-128063.313) r2: (test=0.217) total time= 0.0s  
[CV 2/5] END alpha=10; neg\_root\_mean\_squared\_error: (test=-127857.776) r2: (test=0.190) total time= 0.0s  
[CV 3/5] END alpha=10; neg\_root\_mean\_squared\_error: (test=-122253.535) r2: (test=0.182) total time= 0.0s  
[CV 4/5] END alpha=10; neg\_root\_mean\_squared\_error: (test=-126485.654) r2: (test=0.104) total time= 0.0s  
[CV 5/5] END alpha=10; neg\_root\_mean\_squared\_error: (test=-135613.168) r2: (test=0.131) total time= 0.0s

#### Grid Search (Elasticnet Regression)

GS\_elasticnet\_model.fit(X\_train\_scaled, y\_train)  
y\_pred = GS\_elasticnet\_model.predict(X\_test\_scaled)  
  
best\_accuracy\_score = GS\_elasticnet\_model.best\_score\_  
best\_estimator = GS\_elasticnet\_model.best\_estimator\_  
best\_parametrs = GS\_elasticnet\_model.best\_params\_  
mse = mean\_squared\_error(y\_test, y\_pred)  
mae = mean\_absolute\_error(y\_test, y\_pred)  
final\_accuracy\_PCA\_GRID = final\_accuracy\_PCA\_GRID.append({'Model': 'Elasticnet Regression', 'Accuracy(R2)': best\_accuracy\_score, 'Mean Squared Error': mse, 'Mean Absolute Error': mae, 'Best estimator': best\_estimator, 'Best parametrs': best\_parametrs }, ignore\_index=True)

Fitting 5 folds for each of 9 candidates, totalling 45 fits  
[CV 1/5] END alpha=0.1, l1\_ratio=0.1; neg\_root\_mean\_squared\_error: (test=-128537.950) r2: (test=0.211) total time= 0.0s  
[CV 2/5] END alpha=0.1, l1\_ratio=0.1; neg\_root\_mean\_squared\_error: (test=-128099.476) r2: (test=0.187) total time= 0.0s  
[CV 3/5] END alpha=0.1, l1\_ratio=0.1; neg\_root\_mean\_squared\_error: (test=-122284.783) r2: (test=0.182) total time= 0.0s  
[CV 4/5] END alpha=0.1, l1\_ratio=0.1; neg\_root\_mean\_squared\_error: (test=-125781.281) r2: (test=0.114) total time= 0.0s  
[CV 5/5] END alpha=0.1, l1\_ratio=0.1; neg\_root\_mean\_squared\_error: (test=-135383.147) r2: (test=0.134) total time= 0.0s  
[CV 1/5] END alpha=0.1, l1\_ratio=0.5; neg\_root\_mean\_squared\_error: (test=-128313.898) r2: (test=0.214) total time= 0.0s  
[CV 2/5] END alpha=0.1, l1\_ratio=0.5; neg\_root\_mean\_squared\_error: (test=-127975.376) r2: (test=0.189) total time= 0.0s  
[CV 3/5] END alpha=0.1, l1\_ratio=0.5; neg\_root\_mean\_squared\_error: (test=-122248.605) r2: (test=0.182) total time= 0.0s  
[CV 4/5] END alpha=0.1, l1\_ratio=0.5; neg\_root\_mean\_squared\_error: (test=-126051.567) r2: (test=0.110) total time= 0.0s  
[CV 5/5] END alpha=0.1, l1\_ratio=0.5; neg\_root\_mean\_squared\_error: (test=-135456.050) r2: (test=0.133) total time= 0.0s  
[CV 1/5] END alpha=0.1, l1\_ratio=0.9; neg\_root\_mean\_squared\_error: (test=-128109.162) r2: (test=0.217) total time= 0.0s  
[CV 2/5] END alpha=0.1, l1\_ratio=0.9; neg\_root\_mean\_squared\_error: (test=-127877.466) r2: (test=0.190) total time= 0.0s  
[CV 3/5] END alpha=0.1, l1\_ratio=0.9; neg\_root\_mean\_squared\_error: (test=-122249.161) r2: (test=0.182) total time= 0.0s  
[CV 4/5] END alpha=0.1, l1\_ratio=0.9; neg\_root\_mean\_squared\_error: (test=-126393.184) r2: (test=0.105) total time= 0.0s  
[CV 5/5] END alpha=0.1, l1\_ratio=0.9; neg\_root\_mean\_squared\_error: (test=-135575.625) r2: (test=0.131) total time= 0.0s  
[CV 1/5] END alpha=1, l1\_ratio=0.1; neg\_root\_mean\_squared\_error: (test=-133195.516) r2: (test=0.153) total time= 0.0s  
[CV 2/5] END alpha=1, l1\_ratio=0.1; neg\_root\_mean\_squared\_error: (test=-131631.107) r2: (test=0.141) total time= 0.0s  
[CV 3/5] END alpha=1, l1\_ratio=0.1; neg\_root\_mean\_squared\_error: (test=-125066.510) r2: (test=0.144) total time= 0.0s  
[CV 4/5] END alpha=1, l1\_ratio=0.1; neg\_root\_mean\_squared\_error: (test=-125744.856) r2: (test=0.114) total time= 0.0s  
[CV 5/5] END alpha=1, l1\_ratio=0.1; neg\_root\_mean\_squared\_error: (test=-136981.804) r2: (test=0.113) total time= 0.0s  
[CV 1/5] END alpha=1, l1\_ratio=0.5; neg\_root\_mean\_squared\_error: (test=-131084.977) r2: (test=0.180) total time= 0.0s  
[CV 2/5] END alpha=1, l1\_ratio=0.5; neg\_root\_mean\_squared\_error: (test=-129919.174) r2: (test=0.164) total time= 0.0s  
[CV 3/5] END alpha=1, l1\_ratio=0.5; neg\_root\_mean\_squared\_error: (test=-123572.829) r2: (test=0.165) total time= 0.0s  
[CV 4/5] END alpha=1, l1\_ratio=0.5; neg\_root\_mean\_squared\_error: (test=-125136.163) r2: (test=0.123) total time= 0.0s  
[CV 5/5] END alpha=1, l1\_ratio=0.5; neg\_root\_mean\_squared\_error: (test=-135909.413) r2: (test=0.127) total time= 0.0s  
[CV 1/5] END alpha=1, l1\_ratio=0.9; neg\_root\_mean\_squared\_error: (test=-128596.479) r2: (test=0.211) total time= 0.0s  
[CV 2/5] END alpha=1, l1\_ratio=0.9; neg\_root\_mean\_squared\_error: (test=-128133.903) r2: (test=0.186) total time= 0.0s  
[CV 3/5] END alpha=1, l1\_ratio=0.9; neg\_root\_mean\_squared\_error: (test=-122298.520) r2: (test=0.182) total time= 0.0s  
[CV 4/5] END alpha=1, l1\_ratio=0.9; neg\_root\_mean\_squared\_error: (test=-125722.996) r2: (test=0.115) total time= 0.0s  
[CV 5/5] END alpha=1, l1\_ratio=0.9; neg\_root\_mean\_squared\_error: (test=-135371.155) r2: (test=0.134) total time= 0.0s  
[CV 1/5] END alpha=10, l1\_ratio=0.1; neg\_root\_mean\_squared\_error: (test=-142287.760) r2: (test=0.034) total time= 0.0s  
[CV 2/5] END alpha=10, l1\_ratio=0.1; neg\_root\_mean\_squared\_error: (test=-139661.010) r2: (test=0.034) total time= 0.0s  
[CV 3/5] END alpha=10, l1\_ratio=0.1; neg\_root\_mean\_squared\_error: (test=-132769.304) r2: (test=0.036) total time= 0.0s  
[CV 4/5] END alpha=10, l1\_ratio=0.1; neg\_root\_mean\_squared\_error: (test=-131521.742) r2: (test=0.031) total time= 0.0s  
[CV 5/5] END alpha=10, l1\_ratio=0.1; neg\_root\_mean\_squared\_error: (test=-143374.057) r2: (test=0.029) total time= 0.0s  
[CV 1/5] END alpha=10, l1\_ratio=0.5; neg\_root\_mean\_squared\_error: (test=-140622.223) r2: (test=0.056) total time= 0.0s  
[CV 2/5] END alpha=10, l1\_ratio=0.5; neg\_root\_mean\_squared\_error: (test=-138141.468) r2: (test=0.054) total time= 0.0s  
[CV 3/5] END alpha=10, l1\_ratio=0.5; neg\_root\_mean\_squared\_error: (test=-131270.039) r2: (test=0.057) total time= 0.0s  
[CV 4/5] END alpha=10, l1\_ratio=0.5; neg\_root\_mean\_squared\_error: (test=-130256.082) r2: (test=0.050) total time= 0.0s  
[CV 5/5] END alpha=10, l1\_ratio=0.5; neg\_root\_mean\_squared\_error: (test=-142084.408) r2: (test=0.046) total time= 0.0s  
[CV 1/5] END alpha=10, l1\_ratio=0.9; neg\_root\_mean\_squared\_error: (test=-133646.365) r2: (test=0.148) total time= 0.0s  
[CV 2/5] END alpha=10, l1\_ratio=0.9; neg\_root\_mean\_squared\_error: (test=-132008.447) r2: (test=0.137) total time= 0.0s  
[CV 3/5] END alpha=10, l1\_ratio=0.9; neg\_root\_mean\_squared\_error: (test=-125408.511) r2: (test=0.140) total time= 0.0s  
[CV 4/5] END alpha=10, l1\_ratio=0.9; neg\_root\_mean\_squared\_error: (test=-125934.273) r2: (test=0.112) total time= 0.0s  
[CV 5/5] END alpha=10, l1\_ratio=0.9; neg\_root\_mean\_squared\_error: (test=-137244.772) r2: (test=0.110) total time= 0.0s

#### Grid Search (Decision tree Regression)

GS\_dt\_model.fit(X\_train\_scaled, y\_train)  
y\_pred = GS\_dt\_model.predict(X\_test\_scaled)  
  
best\_accuracy\_score = GS\_dt\_model.best\_score\_  
best\_estimator = GS\_dt\_model.best\_estimator\_  
best\_parametrs = GS\_dt\_model.best\_params\_  
mse = mean\_squared\_error(y\_test, y\_pred)  
mae = mean\_absolute\_error(y\_test, y\_pred)  
final\_accuracy\_PCA\_GRID = final\_accuracy\_PCA\_GRID.append({'Model': 'Decision tree Regression', 'Accuracy(R2)': best\_accuracy\_score, 'Mean Squared Error': mse, 'Mean Absolute Error': mae, 'Best estimator': best\_estimator, 'Best parametrs': best\_parametrs }, ignore\_index=True)

Fitting 5 folds for each of 9 candidates, totalling 45 fits  
[CV 1/5] END max\_depth=3, min\_samples\_split=2; neg\_root\_mean\_squared\_error: (test=-137813.507) r2: (test=0.094) total time= 0.0s  
[CV 2/5] END max\_depth=3, min\_samples\_split=2; neg\_root\_mean\_squared\_error: (test=-131831.745) r2: (test=0.139) total time= 0.0s  
[CV 3/5] END max\_depth=3, min\_samples\_split=2; neg\_root\_mean\_squared\_error: (test=-126529.006) r2: (test=0.124) total time= 0.0s  
[CV 4/5] END max\_depth=3, min\_samples\_split=2; neg\_root\_mean\_squared\_error: (test=-127932.346) r2: (test=0.083) total time= 0.0s  
[CV 5/5] END max\_depth=3, min\_samples\_split=2; neg\_root\_mean\_squared\_error: (test=-139346.902) r2: (test=0.082) total time= 0.0s  
[CV 1/5] END max\_depth=3, min\_samples\_split=5; neg\_root\_mean\_squared\_error: (test=-137813.507) r2: (test=0.094) total time= 0.0s  
[CV 2/5] END max\_depth=3, min\_samples\_split=5; neg\_root\_mean\_squared\_error: (test=-131831.745) r2: (test=0.139) total time= 0.0s  
[CV 3/5] END max\_depth=3, min\_samples\_split=5; neg\_root\_mean\_squared\_error: (test=-126529.006) r2: (test=0.124) total time= 0.0s  
[CV 4/5] END max\_depth=3, min\_samples\_split=5; neg\_root\_mean\_squared\_error: (test=-127932.346) r2: (test=0.083) total time= 0.0s  
[CV 5/5] END max\_depth=3, min\_samples\_split=5; neg\_root\_mean\_squared\_error: (test=-139346.902) r2: (test=0.082) total time= 0.0s  
[CV 1/5] END max\_depth=3, min\_samples\_split=10; neg\_root\_mean\_squared\_error: (test=-137813.507) r2: (test=0.094) total time= 0.0s  
[CV 2/5] END max\_depth=3, min\_samples\_split=10; neg\_root\_mean\_squared\_error: (test=-131831.745) r2: (test=0.139) total time= 0.0s  
[CV 3/5] END max\_depth=3, min\_samples\_split=10; neg\_root\_mean\_squared\_error: (test=-126529.006) r2: (test=0.124) total time= 0.0s  
[CV 4/5] END max\_depth=3, min\_samples\_split=10; neg\_root\_mean\_squared\_error: (test=-127932.346) r2: (test=0.083) total time= 0.0s  
[CV 5/5] END max\_depth=3, min\_samples\_split=10; neg\_root\_mean\_squared\_error: (test=-139346.902) r2: (test=0.082) total time= 0.0s  
[CV 1/5] END max\_depth=4, min\_samples\_split=2; neg\_root\_mean\_squared\_error: (test=-137896.473) r2: (test=0.092) total time= 0.0s  
[CV 2/5] END max\_depth=4, min\_samples\_split=2; neg\_root\_mean\_squared\_error: (test=-130723.263) r2: (test=0.153) total time= 0.0s  
[CV 3/5] END max\_depth=4, min\_samples\_split=2; neg\_root\_mean\_squared\_error: (test=-126319.415) r2: (test=0.127) total time= 0.0s  
[CV 4/5] END max\_depth=4, min\_samples\_split=2; neg\_root\_mean\_squared\_error: (test=-128874.269) r2: (test=0.070) total time= 0.0s  
[CV 5/5] END max\_depth=4, min\_samples\_split=2; neg\_root\_mean\_squared\_error: (test=-137832.695) r2: (test=0.102) total time= 0.0s  
[CV 1/5] END max\_depth=4, min\_samples\_split=5; neg\_root\_mean\_squared\_error: (test=-137896.473) r2: (test=0.092) total time= 0.0s  
[CV 2/5] END max\_depth=4, min\_samples\_split=5; neg\_root\_mean\_squared\_error: (test=-130723.263) r2: (test=0.153) total time= 0.0s  
[CV 3/5] END max\_depth=4, min\_samples\_split=5; neg\_root\_mean\_squared\_error: (test=-126319.415) r2: (test=0.127) total time= 0.0s  
[CV 4/5] END max\_depth=4, min\_samples\_split=5; neg\_root\_mean\_squared\_error: (test=-128874.269) r2: (test=0.070) total time= 0.0s  
[CV 5/5] END max\_depth=4, min\_samples\_split=5; neg\_root\_mean\_squared\_error: (test=-137832.695) r2: (test=0.102) total time= 0.0s  
[CV 1/5] END max\_depth=4, min\_samples\_split=10; neg\_root\_mean\_squared\_error: (test=-137896.473) r2: (test=0.092) total time= 0.0s  
[CV 2/5] END max\_depth=4, min\_samples\_split=10; neg\_root\_mean\_squared\_error: (test=-130723.263) r2: (test=0.153) total time= 0.0s  
[CV 3/5] END max\_depth=4, min\_samples\_split=10; neg\_root\_mean\_squared\_error: (test=-126319.415) r2: (test=0.127) total time= 0.0s  
[CV 4/5] END max\_depth=4, min\_samples\_split=10; neg\_root\_mean\_squared\_error: (test=-128874.269) r2: (test=0.070) total time= 0.0s  
[CV 5/5] END max\_depth=4, min\_samples\_split=10; neg\_root\_mean\_squared\_error: (test=-137832.695) r2: (test=0.102) total time= 0.0s  
[CV 1/5] END max\_depth=5, min\_samples\_split=2; neg\_root\_mean\_squared\_error: (test=-136628.741) r2: (test=0.109) total time= 0.0s  
[CV 2/5] END max\_depth=5, min\_samples\_split=2; neg\_root\_mean\_squared\_error: (test=-133394.774) r2: (test=0.118) total time= 0.0s  
[CV 3/5] END max\_depth=5, min\_samples\_split=2; neg\_root\_mean\_squared\_error: (test=-130829.236) r2: (test=0.064) total time= 0.0s  
[CV 4/5] END max\_depth=5, min\_samples\_split=2; neg\_root\_mean\_squared\_error: (test=-133398.740) r2: (test=0.003) total time= 0.0s  
[CV 5/5] END max\_depth=5, min\_samples\_split=2; neg\_root\_mean\_squared\_error: (test=-137893.449) r2: (test=0.101) total time= 0.0s  
[CV 1/5] END max\_depth=5, min\_samples\_split=5; neg\_root\_mean\_squared\_error: (test=-136440.771) r2: (test=0.111) total time= 0.0s  
[CV 2/5] END max\_depth=5, min\_samples\_split=5; neg\_root\_mean\_squared\_error: (test=-133523.979) r2: (test=0.117) total time= 0.0s  
[CV 3/5] END max\_depth=5, min\_samples\_split=5; neg\_root\_mean\_squared\_error: (test=-128884.739) r2: (test=0.091) total time= 0.0s  
[CV 4/5] END max\_depth=5, min\_samples\_split=5; neg\_root\_mean\_squared\_error: (test=-133486.371) r2: (test=0.002) total time= 0.0s  
[CV 5/5] END max\_depth=5, min\_samples\_split=5; neg\_root\_mean\_squared\_error: (test=-137893.449) r2: (test=0.101) total time= 0.0s  
[CV 1/5] END max\_depth=5, min\_samples\_split=10; neg\_root\_mean\_squared\_error: (test=-136759.249) r2: (test=0.107) total time= 0.0s  
[CV 2/5] END max\_depth=5, min\_samples\_split=10; neg\_root\_mean\_squared\_error: (test=-133455.595) r2: (test=0.118) total time= 0.0s  
[CV 3/5] END max\_depth=5, min\_samples\_split=10; neg\_root\_mean\_squared\_error: (test=-128928.562) r2: (test=0.091) total time= 0.0s  
[CV 4/5] END max\_depth=5, min\_samples\_split=10; neg\_root\_mean\_squared\_error: (test=-133430.617) r2: (test=0.003) total time= 0.0s  
[CV 5/5] END max\_depth=5, min\_samples\_split=10; neg\_root\_mean\_squared\_error: (test=-137771.324) r2: (test=0.103) total time= 0.0s

#### Grid Search (Random Forest Regression)

GS\_rf\_model.fit(X\_train\_scaled, y\_train)  
y\_pred = GS\_rf\_model.predict(X\_test\_scaled)  
  
best\_accuracy\_score = GS\_rf\_model.best\_score\_  
best\_estimator = GS\_rf\_model.best\_estimator\_  
best\_parametrs = GS\_rf\_model.best\_params\_  
mse = mean\_squared\_error(y\_test, y\_pred)  
mae = mean\_absolute\_error(y\_test, y\_pred)  
final\_accuracy\_PCA\_GRID = final\_accuracy\_PCA\_GRID.append({'Model': 'Random Forest Regression', 'Accuracy(R2)': best\_accuracy\_score, 'Mean Squared Error': mse, 'Mean Absolute Error': mae, 'Best estimator': best\_estimator, 'Best parametrs': best\_parametrs }, ignore\_index=True)

Fitting 5 folds for each of 9 candidates, totalling 45 fits  
[CV 1/5] END max\_depth=3, n\_estimators=50; neg\_root\_mean\_squared\_error: (test=-134633.184) r2: (test=0.135) total time= 0.3s  
[CV 2/5] END max\_depth=3, n\_estimators=50; neg\_root\_mean\_squared\_error: (test=-129451.978) r2: (test=0.170) total time= 0.3s  
[CV 3/5] END max\_depth=3, n\_estimators=50; neg\_root\_mean\_squared\_error: (test=-123724.242) r2: (test=0.162) total time= 0.3s  
[CV 4/5] END max\_depth=3, n\_estimators=50; neg\_root\_mean\_squared\_error: (test=-123853.939) r2: (test=0.141) total time= 0.3s  
[CV 5/5] END max\_depth=3, n\_estimators=50; neg\_root\_mean\_squared\_error: (test=-136688.876) r2: (test=0.117) total time= 0.3s  
[CV 1/5] END max\_depth=3, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-134358.783) r2: (test=0.138) total time= 0.5s  
[CV 2/5] END max\_depth=3, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-129255.429) r2: (test=0.172) total time= 0.5s  
[CV 3/5] END max\_depth=3, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-123830.475) r2: (test=0.161) total time= 0.5s  
[CV 4/5] END max\_depth=3, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-123981.059) r2: (test=0.139) total time= 0.5s  
[CV 5/5] END max\_depth=3, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-136928.272) r2: (test=0.114) total time= 0.5s  
[CV 1/5] END max\_depth=3, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-134252.197) r2: (test=0.140) total time= 1.0s  
[CV 2/5] END max\_depth=3, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-129254.164) r2: (test=0.172) total time= 1.0s  
[CV 3/5] END max\_depth=3, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-124038.303) r2: (test=0.158) total time= 1.0s  
[CV 4/5] END max\_depth=3, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-123863.750) r2: (test=0.141) total time= 1.1s  
[CV 5/5] END max\_depth=3, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-136517.219) r2: (test=0.119) total time= 1.3s  
[CV 1/5] END max\_depth=5, n\_estimators=50; neg\_root\_mean\_squared\_error: (test=-131203.230) r2: (test=0.178) total time= 0.4s  
[CV 2/5] END max\_depth=5, n\_estimators=50; neg\_root\_mean\_squared\_error: (test=-127217.642) r2: (test=0.198) total time= 0.4s  
[CV 3/5] END max\_depth=5, n\_estimators=50; neg\_root\_mean\_squared\_error: (test=-122840.550) r2: (test=0.174) total time= 0.4s  
[CV 4/5] END max\_depth=5, n\_estimators=50; neg\_root\_mean\_squared\_error: (test=-123168.162) r2: (test=0.150) total time= 0.4s  
[CV 5/5] END max\_depth=5, n\_estimators=50; neg\_root\_mean\_squared\_error: (test=-135522.621) r2: (test=0.132) total time= 0.4s  
[CV 1/5] END max\_depth=5, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-131711.952) r2: (test=0.172) total time= 0.8s  
[CV 2/5] END max\_depth=5, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-126446.313) r2: (test=0.208) total time= 0.9s  
[CV 3/5] END max\_depth=5, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-123030.085) r2: (test=0.172) total time= 0.8s  
[CV 4/5] END max\_depth=5, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-123744.533) r2: (test=0.142) total time= 0.8s  
[CV 5/5] END max\_depth=5, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-135197.543) r2: (test=0.136) total time= 0.8s  
[CV 1/5] END max\_depth=5, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-131947.780) r2: (test=0.169) total time= 1.6s  
[CV 2/5] END max\_depth=5, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-127464.304) r2: (test=0.195) total time= 1.6s  
[CV 3/5] END max\_depth=5, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-123137.208) r2: (test=0.170) total time= 1.6s  
[CV 4/5] END max\_depth=5, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-123511.507) r2: (test=0.145) total time= 1.6s  
[CV 5/5] END max\_depth=5, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-135068.083) r2: (test=0.138) total time= 1.6s  
[CV 1/5] END max\_depth=7, n\_estimators=50; neg\_root\_mean\_squared\_error: (test=-131348.600) r2: (test=0.177) total time= 0.5s  
[CV 2/5] END max\_depth=7, n\_estimators=50; neg\_root\_mean\_squared\_error: (test=-127130.742) r2: (test=0.199) total time= 0.5s  
[CV 3/5] END max\_depth=7, n\_estimators=50; neg\_root\_mean\_squared\_error: (test=-123646.611) r2: (test=0.164) total time= 0.5s  
[CV 4/5] END max\_depth=7, n\_estimators=50; neg\_root\_mean\_squared\_error: (test=-124155.527) r2: (test=0.137) total time= 0.5s  
[CV 5/5] END max\_depth=7, n\_estimators=50; neg\_root\_mean\_squared\_error: (test=-135945.741) r2: (test=0.127) total time= 0.5s  
[CV 1/5] END max\_depth=7, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-130850.794) r2: (test=0.183) total time= 1.1s  
[CV 2/5] END max\_depth=7, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-126664.825) r2: (test=0.205) total time= 1.2s  
[CV 3/5] END max\_depth=7, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-123580.857) r2: (test=0.164) total time= 1.2s  
[CV 4/5] END max\_depth=7, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-124668.724) r2: (test=0.129) total time= 1.1s  
[CV 5/5] END max\_depth=7, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-135158.001) r2: (test=0.137) total time= 1.2s  
[CV 1/5] END max\_depth=7, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-130489.348) r2: (test=0.187) total time= 2.1s  
[CV 2/5] END max\_depth=7, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-126276.926) r2: (test=0.210) total time= 2.3s  
[CV 3/5] END max\_depth=7, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-122733.554) r2: (test=0.176) total time= 2.6s  
[CV 4/5] END max\_depth=7, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-123825.416) r2: (test=0.141) total time= 2.5s  
[CV 5/5] END max\_depth=7, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-134827.590) r2: (test=0.141) total time= 2.4s

#### Grid Search (Gradient Boosting Regression)

# my apology to your CPU  
GS\_gb\_model.fit(X\_train\_scaled, y\_train)  
y\_pred = GS\_gb\_model.predict(X\_test\_scaled)  
  
best\_accuracy\_score = GS\_gb\_model.best\_score\_  
best\_estimator = GS\_gb\_model.best\_estimator\_  
best\_parametrs = GS\_gb\_model.best\_params\_  
mse = mean\_squared\_error(y\_test, y\_pred)  
mae = mean\_absolute\_error(y\_test, y\_pred)  
final\_accuracy\_PCA\_GRID = final\_accuracy\_PCA\_GRID.append({'Model': 'Gradient Boosting Regression', 'Accuracy(R2)': best\_accuracy\_score, 'Mean Squared Error': mse, 'Mean Absolute Error': mae, 'Best estimator': best\_estimator, 'Best parametrs': best\_parametrs }, ignore\_index=True)

Fitting 3 folds for each of 8 candidates, totalling 24 fits  
[CV 1/3] END learning\_rate=0.1, max\_depth=5, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-133827.593) r2: (test=0.157) total time= 1.1s  
[CV 2/3] END learning\_rate=0.1, max\_depth=5, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-125219.917) r2: (test=0.136) total time= 1.1s  
[CV 3/3] END learning\_rate=0.1, max\_depth=5, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-133893.702) r2: (test=0.088) total time= 1.1s  
[CV 1/3] END learning\_rate=0.1, max\_depth=5, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-136335.241) r2: (test=0.125) total time= 2.2s  
[CV 2/3] END learning\_rate=0.1, max\_depth=5, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-126894.806) r2: (test=0.113) total time= 2.2s  
[CV 3/3] END learning\_rate=0.1, max\_depth=5, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-136555.364) r2: (test=0.052) total time= 2.2s  
[CV 1/3] END learning\_rate=0.1, max\_depth=7, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-133085.375) r2: (test=0.166) total time= 1.5s  
[CV 2/3] END learning\_rate=0.1, max\_depth=7, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-130049.977) r2: (test=0.069) total time= 1.5s  
[CV 3/3] END learning\_rate=0.1, max\_depth=7, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-134054.279) r2: (test=0.086) total time= 1.5s  
[CV 1/3] END learning\_rate=0.1, max\_depth=7, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-133385.441) r2: (test=0.162) total time= 3.0s  
[CV 2/3] END learning\_rate=0.1, max\_depth=7, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-131805.699) r2: (test=0.043) total time= 3.0s  
[CV 3/3] END learning\_rate=0.1, max\_depth=7, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-134810.242) r2: (test=0.076) total time= 2.9s  
[CV 1/3] END learning\_rate=0.5, max\_depth=5, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-148047.069) r2: (test=-0.032) total time= 1.1s  
[CV 2/3] END learning\_rate=0.5, max\_depth=5, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-145730.404) r2: (test=-0.170) total time= 1.1s  
[CV 3/3] END learning\_rate=0.5, max\_depth=5, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-148558.331) r2: (test=-0.122) total time= 1.1s  
[CV 1/3] END learning\_rate=0.5, max\_depth=5, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-148879.306) r2: (test=-0.043) total time= 2.2s  
[CV 2/3] END learning\_rate=0.5, max\_depth=5, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-148078.005) r2: (test=-0.208) total time= 2.3s  
[CV 3/3] END learning\_rate=0.5, max\_depth=5, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-148814.222) r2: (test=-0.126) total time= 2.3s  
[CV 1/3] END learning\_rate=0.5, max\_depth=7, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-144802.896) r2: (test=0.013) total time= 1.5s  
[CV 2/3] END learning\_rate=0.5, max\_depth=7, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-147377.121) r2: (test=-0.196) total time= 1.5s  
[CV 3/3] END learning\_rate=0.5, max\_depth=7, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-149309.313) r2: (test=-0.134) total time= 1.5s  
[CV 1/3] END learning\_rate=0.5, max\_depth=7, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-143953.704) r2: (test=0.024) total time= 2.9s  
[CV 2/3] END learning\_rate=0.5, max\_depth=7, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-144892.197) r2: (test=-0.156) total time= 2.9s  
[CV 3/3] END learning\_rate=0.5, max\_depth=7, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-152714.413) r2: (test=-0.186) total time= 3.0s

#### Grid Search (SVR)

GS\_svr\_model.fit(X\_train\_scaled, y\_train)  
y\_pred = GS\_svr\_model.predict(X\_test\_scaled)  
  
best\_accuracy\_score = GS\_svr\_model.best\_score\_  
best\_estimator = GS\_svr\_model.best\_estimator\_  
best\_parametrs = GS\_svr\_model.best\_params\_  
mse = mean\_squared\_error(y\_test, y\_pred)  
mae = mean\_absolute\_error(y\_test, y\_pred)  
final\_accuracy\_PCA\_GRID = final\_accuracy\_PCA\_GRID.append({'Model': 'SVR', 'Accuracy(R2)': best\_accuracy\_score, 'Mean Squared Error': mse, 'Mean Absolute Error': mae, 'Best estimator': best\_estimator, 'Best parametrs': best\_parametrs }, ignore\_index=True)

Fitting 3 folds for each of 2 candidates, totalling 6 fits  
[CV 1/3] END C=0.1, epsilon=0.01; neg\_root\_mean\_squared\_error: (test=-148852.257) r2: (test=-0.043) total time= 0.2s  
[CV 2/3] END C=0.1, epsilon=0.01; neg\_root\_mean\_squared\_error: (test=-136064.806) r2: (test=-0.020) total time= 0.2s  
[CV 3/3] END C=0.1, epsilon=0.01; neg\_root\_mean\_squared\_error: (test=-141409.384) r2: (test=-0.017) total time= 0.2s  
[CV 1/3] END C=0.1, epsilon=0.1; neg\_root\_mean\_squared\_error: (test=-148852.257) r2: (test=-0.043) total time= 0.2s  
[CV 2/3] END C=0.1, epsilon=0.1; neg\_root\_mean\_squared\_error: (test=-136064.806) r2: (test=-0.020) total time= 0.1s  
[CV 3/3] END C=0.1, epsilon=0.1; neg\_root\_mean\_squared\_error: (test=-141409.384) r2: (test=-0.017) total time= 0.1s

#### Grid Search (KNN)

GS\_knn\_model.fit(X\_train\_scaled, y\_train)  
y\_pred = GS\_knn\_model.predict(X\_test\_scaled)   
  
best\_accuracy\_score = GS\_knn\_model.best\_score\_  
best\_estimator = GS\_knn\_model.best\_estimator\_  
best\_parametrs = GS\_knn\_model.best\_params\_  
mse = mean\_squared\_error(y\_test, y\_pred)  
mae = mean\_absolute\_error(y\_test, y\_pred)  
final\_accuracy\_PCA\_GRID = final\_accuracy\_PCA\_GRID.append({'Model': 'KNN', 'Accuracy(R2)': best\_accuracy\_score, 'Mean Squared Error': mse, 'Mean Absolute Error': mae, 'Best estimator': best\_estimator, 'Best parametrs': best\_parametrs }, ignore\_index=True)

Fitting 5 folds for each of 6 candidates, totalling 30 fits  
[CV 1/5] END n\_neighbors=3, weights=uniform; neg\_root\_mean\_squared\_error: (test=-146806.681) r2: (test=-0.029) total time= 0.0s  
[CV 2/5] END n\_neighbors=3, weights=uniform; neg\_root\_mean\_squared\_error: (test=-147108.609) r2: (test=-0.072) total time= 0.0s  
[CV 3/5] END n\_neighbors=3, weights=uniform; neg\_root\_mean\_squared\_error: (test=-142299.298) r2: (test=-0.108) total time= 0.0s  
[CV 4/5] END n\_neighbors=3, weights=uniform; neg\_root\_mean\_squared\_error: (test=-144721.953) r2: (test=-0.173) total time= 0.0s  
[CV 5/5] END n\_neighbors=3, weights=uniform; neg\_root\_mean\_squared\_error: (test=-152264.959) r2: (test=-0.096) total time= 0.0s  
[CV 1/5] END n\_neighbors=3, weights=distance; neg\_root\_mean\_squared\_error: (test=-146579.682) r2: (test=-0.025) total time= 0.0s  
[CV 2/5] END n\_neighbors=3, weights=distance; neg\_root\_mean\_squared\_error: (test=-147394.287) r2: (test=-0.076) total time= 0.0s  
[CV 3/5] END n\_neighbors=3, weights=distance; neg\_root\_mean\_squared\_error: (test=-142632.560) r2: (test=-0.113) total time= 0.0s  
[CV 4/5] END n\_neighbors=3, weights=distance; neg\_root\_mean\_squared\_error: (test=-144872.288) r2: (test=-0.176) total time= 0.0s  
[CV 5/5] END n\_neighbors=3, weights=distance; neg\_root\_mean\_squared\_error: (test=-152159.184) r2: (test=-0.094) total time= 0.0s  
[CV 1/5] END n\_neighbors=5, weights=uniform; neg\_root\_mean\_squared\_error: (test=-139202.561) r2: (test=0.075) total time= 0.0s  
[CV 2/5] END n\_neighbors=5, weights=uniform; neg\_root\_mean\_squared\_error: (test=-141884.075) r2: (test=0.003) total time= 0.0s  
[CV 3/5] END n\_neighbors=5, weights=uniform; neg\_root\_mean\_squared\_error: (test=-134702.888) r2: (test=0.007) total time= 0.0s  
[CV 4/5] END n\_neighbors=5, weights=uniform; neg\_root\_mean\_squared\_error: (test=-137638.700) r2: (test=-0.061) total time= 0.0s  
[CV 5/5] END n\_neighbors=5, weights=uniform; neg\_root\_mean\_squared\_error: (test=-142747.349) r2: (test=0.037) total time= 0.0s  
[CV 1/5] END n\_neighbors=5, weights=distance; neg\_root\_mean\_squared\_error: (test=-138804.486) r2: (test=0.080) total time= 0.0s  
[CV 2/5] END n\_neighbors=5, weights=distance; neg\_root\_mean\_squared\_error: (test=-142009.511) r2: (test=0.001) total time= 0.0s  
[CV 3/5] END n\_neighbors=5, weights=distance; neg\_root\_mean\_squared\_error: (test=-134707.923) r2: (test=0.007) total time= 0.0s  
[CV 4/5] END n\_neighbors=5, weights=distance; neg\_root\_mean\_squared\_error: (test=-137857.278) r2: (test=-0.065) total time= 0.0s  
[CV 5/5] END n\_neighbors=5, weights=distance; neg\_root\_mean\_squared\_error: (test=-142766.761) r2: (test=0.037) total time= 0.0s  
[CV 1/5] END n\_neighbors=7, weights=uniform; neg\_root\_mean\_squared\_error: (test=-138049.104) r2: (test=0.090) total time= 0.0s  
[CV 2/5] END n\_neighbors=7, weights=uniform; neg\_root\_mean\_squared\_error: (test=-138495.010) r2: (test=0.050) total time= 0.0s  
[CV 3/5] END n\_neighbors=7, weights=uniform; neg\_root\_mean\_squared\_error: (test=-131092.536) r2: (test=0.060) total time= 0.0s  
[CV 4/5] END n\_neighbors=7, weights=uniform; neg\_root\_mean\_squared\_error: (test=-134463.907) r2: (test=-0.013) total time= 0.0s  
[CV 5/5] END n\_neighbors=7, weights=uniform; neg\_root\_mean\_squared\_error: (test=-140612.154) r2: (test=0.066) total time= 0.0s  
[CV 1/5] END n\_neighbors=7, weights=distance; neg\_root\_mean\_squared\_error: (test=-137464.567) r2: (test=0.098) total time= 0.0s  
[CV 2/5] END n\_neighbors=7, weights=distance; neg\_root\_mean\_squared\_error: (test=-138582.047) r2: (test=0.048) total time= 0.0s  
[CV 3/5] END n\_neighbors=7, weights=distance; neg\_root\_mean\_squared\_error: (test=-131197.068) r2: (test=0.058) total time= 0.0s  
[CV 4/5] END n\_neighbors=7, weights=distance; neg\_root\_mean\_squared\_error: (test=-134607.316) r2: (test=-0.015) total time= 0.0s  
[CV 5/5] END n\_neighbors=7, weights=distance; neg\_root\_mean\_squared\_error: (test=-140536.598) r2: (test=0.067) total time= 0.0s

## Evaluation Models GridSearch only

#### Grid Search (Linear Regression)

GS\_lr\_model.fit(X\_train, y\_train)  
y\_pred = GS\_lr\_model.predict(X\_test)  
  
best\_accuracy\_score = GS\_lr\_model.best\_score\_  
best\_estimator = GS\_lr\_model.best\_estimator\_  
best\_parametrs = GS\_lr\_model.best\_params\_  
mse = mean\_squared\_error(y\_test, y\_pred)  
mae = mean\_absolute\_error(y\_test, y\_pred)  
final\_accuracy\_GRID = final\_accuracy\_GRID.append({'Model': 'Linear Regression', 'Accuracy(R2)': best\_accuracy\_score, 'Mean Squared Error': mse, 'Mean Absolute Error': mae, 'Best estimator': best\_estimator, 'Best parametrs': best\_parametrs }, ignore\_index=True)

Fitting 5 folds for each of 1 candidates, totalling 5 fits  
[CV 1/5] END neg\_root\_mean\_squared\_error: (test=-128061.824) r2: (test=0.217) total time= 0.0s  
[CV 2/5] END neg\_root\_mean\_squared\_error: (test=-127857.977) r2: (test=0.190) total time= 0.0s  
[CV 3/5] END neg\_root\_mean\_squared\_error: (test=-122256.146) r2: (test=0.182) total time= 0.0s  
[CV 4/5] END neg\_root\_mean\_squared\_error: (test=-126491.510) r2: (test=0.104) total time= 0.0s  
[CV 5/5] END neg\_root\_mean\_squared\_error: (test=-135614.084) r2: (test=0.131) total time= 0.0s

#### Grid Search (Ridge Regression)

GS\_ridge\_model.fit(X\_train, y\_train)  
y\_pred = GS\_ridge\_model.predict(X\_test)  
  
best\_accuracy\_score = GS\_ridge\_model.best\_score\_  
best\_estimator = GS\_ridge\_model.best\_estimator\_  
best\_parametrs = GS\_ridge\_model.best\_params\_  
mse = mean\_squared\_error(y\_test, y\_pred)  
mae = mean\_absolute\_error(y\_test, y\_pred)  
final\_accuracy\_GRID = final\_accuracy\_GRID.append({'Model': 'Ridge Regression', 'Accuracy(R2)': best\_accuracy\_score, 'Mean Squared Error': mse, 'Mean Absolute Error': mae, 'Best estimator': best\_estimator, 'Best parametrs': best\_parametrs }, ignore\_index=True)

Fitting 5 folds for each of 3 candidates, totalling 15 fits  
[CV 1/5] END alpha=0.1; neg\_root\_mean\_squared\_error: (test=-128064.801) r2: (test=0.217) total time= 0.0s  
[CV 2/5] END alpha=0.1; neg\_root\_mean\_squared\_error: (test=-127856.178) r2: (test=0.190) total time= 0.0s  
[CV 3/5] END alpha=0.1; neg\_root\_mean\_squared\_error: (test=-122255.139) r2: (test=0.182) total time= 0.0s  
[CV 4/5] END alpha=0.1; neg\_root\_mean\_squared\_error: (test=-126491.671) r2: (test=0.104) total time= 0.0s  
[CV 5/5] END alpha=0.1; neg\_root\_mean\_squared\_error: (test=-135611.328) r2: (test=0.131) total time= 0.0s  
[CV 1/5] END alpha=1; neg\_root\_mean\_squared\_error: (test=-128091.571) r2: (test=0.217) total time= 0.0s  
[CV 2/5] END alpha=1; neg\_root\_mean\_squared\_error: (test=-127841.281) r2: (test=0.190) total time= 0.0s  
[CV 3/5] END alpha=1; neg\_root\_mean\_squared\_error: (test=-122247.187) r2: (test=0.182) total time= 0.0s  
[CV 4/5] END alpha=1; neg\_root\_mean\_squared\_error: (test=-126493.307) r2: (test=0.104) total time= 0.0s  
[CV 5/5] END alpha=1; neg\_root\_mean\_squared\_error: (test=-135587.805) r2: (test=0.131) total time= 0.0s  
[CV 1/5] END alpha=10; neg\_root\_mean\_squared\_error: (test=-128348.787) r2: (test=0.214) total time= 0.0s  
[CV 2/5] END alpha=10; neg\_root\_mean\_squared\_error: (test=-127776.715) r2: (test=0.191) total time= 0.0s  
[CV 3/5] END alpha=10; neg\_root\_mean\_squared\_error: (test=-122237.864) r2: (test=0.182) total time= 0.0s  
[CV 4/5] END alpha=10; neg\_root\_mean\_squared\_error: (test=-126521.849) r2: (test=0.103) total time= 0.0s  
[CV 5/5] END alpha=10; neg\_root\_mean\_squared\_error: (test=-135444.107) r2: (test=0.133) total time= 0.0s

#### Grid Search (Lasso Regression)

GS\_lasso\_model.fit(X\_train, y\_train)  
y\_pred = GS\_lasso\_model.predict(X\_test)  
  
best\_accuracy\_score = GS\_lasso\_model.best\_score\_  
best\_estimator = GS\_lasso\_model.best\_estimator\_  
best\_parametrs = GS\_lasso\_model.best\_params\_  
mse = mean\_squared\_error(y\_test, y\_pred)  
mae = mean\_absolute\_error(y\_test, y\_pred)  
final\_accuracy\_GRID = final\_accuracy\_GRID.append({'Model': 'Lasso Regression', 'Accuracy(R2)': best\_accuracy\_score, 'Mean Squared Error': mse, 'Mean Absolute Error': mae, 'Best estimator': best\_estimator, 'Best parametrs': best\_parametrs }, ignore\_index=True)

Fitting 5 folds for each of 3 candidates, totalling 15 fits  
[CV 1/5] END alpha=0.1; neg\_root\_mean\_squared\_error: (test=-128061.882) r2: (test=0.217) total time= 0.0s  
[CV 2/5] END alpha=0.1; neg\_root\_mean\_squared\_error: (test=-127857.919) r2: (test=0.190) total time= 0.0s  
[CV 3/5] END alpha=0.1; neg\_root\_mean\_squared\_error: (test=-122256.119) r2: (test=0.182) total time= 0.0s  
[CV 4/5] END alpha=0.1; neg\_root\_mean\_squared\_error: (test=-126491.459) r2: (test=0.104) total time= 0.0s  
[CV 5/5] END alpha=0.1; neg\_root\_mean\_squared\_error: (test=-135613.970) r2: (test=0.131) total time= 0.0s  
[CV 1/5] END alpha=1; neg\_root\_mean\_squared\_error: (test=-128062.402) r2: (test=0.217) total time= 0.0s  
[CV 2/5] END alpha=1; neg\_root\_mean\_squared\_error: (test=-127857.402) r2: (test=0.190) total time= 0.0s  
[CV 3/5] END alpha=1; neg\_root\_mean\_squared\_error: (test=-122255.879) r2: (test=0.182) total time= 0.0s  
[CV 4/5] END alpha=1; neg\_root\_mean\_squared\_error: (test=-126491.003) r2: (test=0.104) total time= 0.0s  
[CV 5/5] END alpha=1; neg\_root\_mean\_squared\_error: (test=-135612.946) r2: (test=0.131) total time= 0.0s  
[CV 1/5] END alpha=10; neg\_root\_mean\_squared\_error: (test=-128067.658) r2: (test=0.217) total time= 0.0s  
[CV 2/5] END alpha=10; neg\_root\_mean\_squared\_error: (test=-127852.282) r2: (test=0.190) total time= 0.0s  
[CV 3/5] END alpha=10; neg\_root\_mean\_squared\_error: (test=-122253.536) r2: (test=0.182) total time= 0.0s  
[CV 4/5] END alpha=10; neg\_root\_mean\_squared\_error: (test=-126486.504) r2: (test=0.104) total time= 0.0s  
[CV 5/5] END alpha=10; neg\_root\_mean\_squared\_error: (test=-135602.754) r2: (test=0.131) total time= 0.0s

#### Grid Search (Elasticnet Regression)

GS\_elasticnet\_model.fit(X\_train, y\_train)  
y\_pred = GS\_elasticnet\_model.predict(X\_test)  
  
best\_accuracy\_score = GS\_elasticnet\_model.best\_score\_  
best\_estimator = GS\_elasticnet\_model.best\_estimator\_  
best\_parametrs = GS\_elasticnet\_model.best\_params\_  
mse = mean\_squared\_error(y\_test, y\_pred)  
mae = mean\_absolute\_error(y\_test, y\_pred)  
final\_accuracy\_GRID = final\_accuracy\_GRID.append({'Model': 'Elasticnet Regression', 'Accuracy(R2)': best\_accuracy\_score, 'Mean Squared Error': mse, 'Mean Absolute Error': mae, 'Best estimator': best\_estimator, 'Best parametrs': best\_parametrs }, ignore\_index=True)

Fitting 5 folds for each of 9 candidates, totalling 45 fits  
[CV 1/5] END alpha=0.1, l1\_ratio=0.1; neg\_root\_mean\_squared\_error: (test=-130387.087) r2: (test=0.189) total time= 0.0s  
[CV 2/5] END alpha=0.1, l1\_ratio=0.1; neg\_root\_mean\_squared\_error: (test=-128788.317) r2: (test=0.178) total time= 0.0s  
[CV 3/5] END alpha=0.1, l1\_ratio=0.1; neg\_root\_mean\_squared\_error: (test=-123511.561) r2: (test=0.165) total time= 0.0s  
[CV 4/5] END alpha=0.1, l1\_ratio=0.1; neg\_root\_mean\_squared\_error: (test=-127092.746) r2: (test=0.095) total time= 0.0s  
[CV 5/5] END alpha=0.1, l1\_ratio=0.1; neg\_root\_mean\_squared\_error: (test=-135979.820) r2: (test=0.126) total time= 0.0s  
[CV 1/5] END alpha=0.1, l1\_ratio=0.5; neg\_root\_mean\_squared\_error: (test=-129794.220) r2: (test=0.196) total time= 0.0s  
[CV 2/5] END alpha=0.1, l1\_ratio=0.5; neg\_root\_mean\_squared\_error: (test=-128325.645) r2: (test=0.184) total time= 0.0s  
[CV 3/5] END alpha=0.1, l1\_ratio=0.5; neg\_root\_mean\_squared\_error: (test=-122991.373) r2: (test=0.172) total time= 0.0s  
[CV 4/5] END alpha=0.1, l1\_ratio=0.5; neg\_root\_mean\_squared\_error: (test=-126908.797) r2: (test=0.098) total time= 0.0s  
[CV 5/5] END alpha=0.1, l1\_ratio=0.5; neg\_root\_mean\_squared\_error: (test=-135696.862) r2: (test=0.130) total time= 0.0s  
[CV 1/5] END alpha=0.1, l1\_ratio=0.9; neg\_root\_mean\_squared\_error: (test=-128590.190) r2: (test=0.211) total time= 0.0s  
[CV 2/5] END alpha=0.1, l1\_ratio=0.9; neg\_root\_mean\_squared\_error: (test=-127790.545) r2: (test=0.191) total time= 0.0s  
[CV 3/5] END alpha=0.1, l1\_ratio=0.9; neg\_root\_mean\_squared\_error: (test=-122293.034) r2: (test=0.182) total time= 0.0s  
[CV 4/5] END alpha=0.1, l1\_ratio=0.9; neg\_root\_mean\_squared\_error: (test=-126565.000) r2: (test=0.103) total time= 0.0s  
[CV 5/5] END alpha=0.1, l1\_ratio=0.9; neg\_root\_mean\_squared\_error: (test=-135395.738) r2: (test=0.134) total time= 0.0s  
[CV 1/5] END alpha=1, l1\_ratio=0.1; neg\_root\_mean\_squared\_error: (test=-132521.994) r2: (test=0.162) total time= 0.0s  
[CV 2/5] END alpha=1, l1\_ratio=0.1; neg\_root\_mean\_squared\_error: (test=-131820.924) r2: (test=0.139) total time= 0.0s  
[CV 3/5] END alpha=1, l1\_ratio=0.1; neg\_root\_mean\_squared\_error: (test=-126363.556) r2: (test=0.126) total time= 0.0s  
[CV 4/5] END alpha=1, l1\_ratio=0.1; neg\_root\_mean\_squared\_error: (test=-127794.208) r2: (test=0.085) total time= 0.0s  
[CV 5/5] END alpha=1, l1\_ratio=0.1; neg\_root\_mean\_squared\_error: (test=-137504.928) r2: (test=0.107) total time= 0.0s  
[CV 1/5] END alpha=1, l1\_ratio=0.5; neg\_root\_mean\_squared\_error: (test=-131987.093) r2: (test=0.169) total time= 0.0s  
[CV 2/5] END alpha=1, l1\_ratio=0.5; neg\_root\_mean\_squared\_error: (test=-130881.616) r2: (test=0.151) total time= 0.0s  
[CV 3/5] END alpha=1, l1\_ratio=0.5; neg\_root\_mean\_squared\_error: (test=-125516.289) r2: (test=0.138) total time= 0.0s  
[CV 4/5] END alpha=1, l1\_ratio=0.5; neg\_root\_mean\_squared\_error: (test=-127560.802) r2: (test=0.089) total time= 0.0s  
[CV 5/5] END alpha=1, l1\_ratio=0.5; neg\_root\_mean\_squared\_error: (test=-136996.090) r2: (test=0.113) total time= 0.0s  
[CV 1/5] END alpha=1, l1\_ratio=0.9; neg\_root\_mean\_squared\_error: (test=-130494.019) r2: (test=0.187) total time= 0.0s  
[CV 2/5] END alpha=1, l1\_ratio=0.9; neg\_root\_mean\_squared\_error: (test=-128886.393) r2: (test=0.177) total time= 0.0s  
[CV 3/5] END alpha=1, l1\_ratio=0.9; neg\_root\_mean\_squared\_error: (test=-123616.690) r2: (test=0.164) total time= 0.0s  
[CV 4/5] END alpha=1, l1\_ratio=0.9; neg\_root\_mean\_squared\_error: (test=-127124.137) r2: (test=0.095) total time= 0.0s  
[CV 5/5] END alpha=1, l1\_ratio=0.9; neg\_root\_mean\_squared\_error: (test=-136034.499) r2: (test=0.126) total time= 0.0s  
[CV 1/5] END alpha=10, l1\_ratio=0.1; neg\_root\_mean\_squared\_error: (test=-134405.009) r2: (test=0.138) total time= 0.0s  
[CV 2/5] END alpha=10, l1\_ratio=0.1; neg\_root\_mean\_squared\_error: (test=-134858.171) r2: (test=0.099) total time= 0.0s  
[CV 3/5] END alpha=10, l1\_ratio=0.1; neg\_root\_mean\_squared\_error: (test=-129152.901) r2: (test=0.087) total time= 0.0s  
[CV 4/5] END alpha=10, l1\_ratio=0.1; neg\_root\_mean\_squared\_error: (test=-128992.566) r2: (test=0.068) total time= 0.0s  
[CV 5/5] END alpha=10, l1\_ratio=0.1; neg\_root\_mean\_squared\_error: (test=-139642.385) r2: (test=0.079) total time= 0.0s  
[CV 1/5] END alpha=10, l1\_ratio=0.5; neg\_root\_mean\_squared\_error: (test=-134049.239) r2: (test=0.142) total time= 0.0s  
[CV 2/5] END alpha=10, l1\_ratio=0.5; neg\_root\_mean\_squared\_error: (test=-134341.748) r2: (test=0.106) total time= 0.0s  
[CV 3/5] END alpha=10, l1\_ratio=0.5; neg\_root\_mean\_squared\_error: (test=-128688.238) r2: (test=0.094) total time= 0.0s  
[CV 4/5] END alpha=10, l1\_ratio=0.5; neg\_root\_mean\_squared\_error: (test=-128758.793) r2: (test=0.071) total time= 0.0s  
[CV 5/5] END alpha=10, l1\_ratio=0.5; neg\_root\_mean\_squared\_error: (test=-139225.845) r2: (test=0.084) total time= 0.0s  
[CV 1/5] END alpha=10, l1\_ratio=0.9; neg\_root\_mean\_squared\_error: (test=-132622.701) r2: (test=0.161) total time= 0.0s  
[CV 2/5] END alpha=10, l1\_ratio=0.9; neg\_root\_mean\_squared\_error: (test=-131998.045) r2: (test=0.137) total time= 0.0s  
[CV 3/5] END alpha=10, l1\_ratio=0.9; neg\_root\_mean\_squared\_error: (test=-126524.491) r2: (test=0.124) total time= 0.0s  
[CV 4/5] END alpha=10, l1\_ratio=0.9; neg\_root\_mean\_squared\_error: (test=-127846.285) r2: (test=0.084) total time= 0.0s  
[CV 5/5] END alpha=10, l1\_ratio=0.9; neg\_root\_mean\_squared\_error: (test=-137609.838) r2: (test=0.105) total time= 0.0s

#### Grid Search (Decision tree Regression)

GS\_dt\_model.fit(X\_train, y\_train)  
y\_pred = GS\_dt\_model.predict(X\_test)  
  
best\_accuracy\_score = GS\_dt\_model.best\_score\_  
best\_estimator = GS\_dt\_model.best\_estimator\_  
best\_parametrs = GS\_dt\_model.best\_params\_  
mse = mean\_squared\_error(y\_test, y\_pred)  
mae = mean\_absolute\_error(y\_test, y\_pred)  
final\_accuracy\_GRID = final\_accuracy\_GRID.append({'Model': 'Decision tree Regression', 'Accuracy(R2)': best\_accuracy\_score, 'Mean Squared Error': mse, 'Mean Absolute Error': mae, 'Best estimator': best\_estimator, 'Best parametrs': best\_parametrs }, ignore\_index=True)

Fitting 5 folds for each of 9 candidates, totalling 45 fits  
[CV 1/5] END max\_depth=3, min\_samples\_split=2; neg\_root\_mean\_squared\_error: (test=-136791.481) r2: (test=0.107) total time= 0.0s  
[CV 2/5] END max\_depth=3, min\_samples\_split=2; neg\_root\_mean\_squared\_error: (test=-133257.499) r2: (test=0.120) total time= 0.0s  
[CV 3/5] END max\_depth=3, min\_samples\_split=2; neg\_root\_mean\_squared\_error: (test=-127049.613) r2: (test=0.117) total time= 0.0s  
[CV 4/5] END max\_depth=3, min\_samples\_split=2; neg\_root\_mean\_squared\_error: (test=-128110.806) r2: (test=0.081) total time= 0.0s  
[CV 5/5] END max\_depth=3, min\_samples\_split=2; neg\_root\_mean\_squared\_error: (test=-140849.888) r2: (test=0.063) total time= 0.0s  
[CV 1/5] END max\_depth=3, min\_samples\_split=5; neg\_root\_mean\_squared\_error: (test=-136791.481) r2: (test=0.107) total time= 0.0s  
[CV 2/5] END max\_depth=3, min\_samples\_split=5; neg\_root\_mean\_squared\_error: (test=-133257.499) r2: (test=0.120) total time= 0.0s  
[CV 3/5] END max\_depth=3, min\_samples\_split=5; neg\_root\_mean\_squared\_error: (test=-127049.613) r2: (test=0.117) total time= 0.0s  
[CV 4/5] END max\_depth=3, min\_samples\_split=5; neg\_root\_mean\_squared\_error: (test=-128110.806) r2: (test=0.081) total time= 0.0s  
[CV 5/5] END max\_depth=3, min\_samples\_split=5; neg\_root\_mean\_squared\_error: (test=-140849.888) r2: (test=0.063) total time= 0.0s  
[CV 1/5] END max\_depth=3, min\_samples\_split=10; neg\_root\_mean\_squared\_error: (test=-136791.481) r2: (test=0.107) total time= 0.0s  
[CV 2/5] END max\_depth=3, min\_samples\_split=10; neg\_root\_mean\_squared\_error: (test=-133257.499) r2: (test=0.120) total time= 0.0s  
[CV 3/5] END max\_depth=3, min\_samples\_split=10; neg\_root\_mean\_squared\_error: (test=-127049.613) r2: (test=0.117) total time= 0.0s  
[CV 4/5] END max\_depth=3, min\_samples\_split=10; neg\_root\_mean\_squared\_error: (test=-128110.806) r2: (test=0.081) total time= 0.0s  
[CV 5/5] END max\_depth=3, min\_samples\_split=10; neg\_root\_mean\_squared\_error: (test=-140849.888) r2: (test=0.063) total time= 0.0s  
[CV 1/5] END max\_depth=4, min\_samples\_split=2; neg\_root\_mean\_squared\_error: (test=-135674.173) r2: (test=0.121) total time= 0.0s  
[CV 2/5] END max\_depth=4, min\_samples\_split=2; neg\_root\_mean\_squared\_error: (test=-133193.411) r2: (test=0.121) total time= 0.0s  
[CV 3/5] END max\_depth=4, min\_samples\_split=2; neg\_root\_mean\_squared\_error: (test=-126576.008) r2: (test=0.123) total time= 0.0s  
[CV 4/5] END max\_depth=4, min\_samples\_split=2; neg\_root\_mean\_squared\_error: (test=-130110.624) r2: (test=0.052) total time= 0.0s  
[CV 5/5] END max\_depth=4, min\_samples\_split=2; neg\_root\_mean\_squared\_error: (test=-140867.973) r2: (test=0.062) total time= 0.0s  
[CV 1/5] END max\_depth=4, min\_samples\_split=5; neg\_root\_mean\_squared\_error: (test=-135674.173) r2: (test=0.121) total time= 0.0s  
[CV 2/5] END max\_depth=4, min\_samples\_split=5; neg\_root\_mean\_squared\_error: (test=-133193.411) r2: (test=0.121) total time= 0.0s  
[CV 3/5] END max\_depth=4, min\_samples\_split=5; neg\_root\_mean\_squared\_error: (test=-126576.008) r2: (test=0.123) total time= 0.0s  
[CV 4/5] END max\_depth=4, min\_samples\_split=5; neg\_root\_mean\_squared\_error: (test=-130110.624) r2: (test=0.052) total time= 0.0s  
[CV 5/5] END max\_depth=4, min\_samples\_split=5; neg\_root\_mean\_squared\_error: (test=-140867.973) r2: (test=0.062) total time= 0.0s  
[CV 1/5] END max\_depth=4, min\_samples\_split=10; neg\_root\_mean\_squared\_error: (test=-135674.173) r2: (test=0.121) total time= 0.0s  
[CV 2/5] END max\_depth=4, min\_samples\_split=10; neg\_root\_mean\_squared\_error: (test=-133193.411) r2: (test=0.121) total time= 0.0s  
[CV 3/5] END max\_depth=4, min\_samples\_split=10; neg\_root\_mean\_squared\_error: (test=-126576.008) r2: (test=0.123) total time= 0.0s  
[CV 4/5] END max\_depth=4, min\_samples\_split=10; neg\_root\_mean\_squared\_error: (test=-130110.624) r2: (test=0.052) total time= 0.0s  
[CV 5/5] END max\_depth=4, min\_samples\_split=10; neg\_root\_mean\_squared\_error: (test=-140803.014) r2: (test=0.063) total time= 0.0s  
[CV 1/5] END max\_depth=5, min\_samples\_split=2; neg\_root\_mean\_squared\_error: (test=-136150.053) r2: (test=0.115) total time= 0.0s  
[CV 2/5] END max\_depth=5, min\_samples\_split=2; neg\_root\_mean\_squared\_error: (test=-137841.534) r2: (test=0.059) total time= 0.0s  
[CV 3/5] END max\_depth=5, min\_samples\_split=2; neg\_root\_mean\_squared\_error: (test=-130239.100) r2: (test=0.072) total time= 0.0s  
[CV 4/5] END max\_depth=5, min\_samples\_split=2; neg\_root\_mean\_squared\_error: (test=-127740.305) r2: (test=0.086) total time= 0.0s  
[CV 5/5] END max\_depth=5, min\_samples\_split=2; neg\_root\_mean\_squared\_error: (test=-141753.149) r2: (test=0.050) total time= 0.0s  
[CV 1/5] END max\_depth=5, min\_samples\_split=5; neg\_root\_mean\_squared\_error: (test=-135457.212) r2: (test=0.124) total time= 0.0s  
[CV 2/5] END max\_depth=5, min\_samples\_split=5; neg\_root\_mean\_squared\_error: (test=-137751.722) r2: (test=0.060) total time= 0.0s  
[CV 3/5] END max\_depth=5, min\_samples\_split=5; neg\_root\_mean\_squared\_error: (test=-130085.363) r2: (test=0.074) total time= 0.0s  
[CV 4/5] END max\_depth=5, min\_samples\_split=5; neg\_root\_mean\_squared\_error: (test=-127740.305) r2: (test=0.086) total time= 0.0s  
[CV 5/5] END max\_depth=5, min\_samples\_split=5; neg\_root\_mean\_squared\_error: (test=-141714.264) r2: (test=0.051) total time= 0.0s  
[CV 1/5] END max\_depth=5, min\_samples\_split=10; neg\_root\_mean\_squared\_error: (test=-135838.998) r2: (test=0.119) total time= 0.0s  
[CV 2/5] END max\_depth=5, min\_samples\_split=10; neg\_root\_mean\_squared\_error: (test=-137841.534) r2: (test=0.059) total time= 0.0s  
[CV 3/5] END max\_depth=5, min\_samples\_split=10; neg\_root\_mean\_squared\_error: (test=-129957.808) r2: (test=0.076) total time= 0.0s  
[CV 4/5] END max\_depth=5, min\_samples\_split=10; neg\_root\_mean\_squared\_error: (test=-127690.418) r2: (test=0.087) total time= 0.0s  
[CV 5/5] END max\_depth=5, min\_samples\_split=10; neg\_root\_mean\_squared\_error: (test=-140853.104) r2: (test=0.062) total time= 0.0s

#### Grid Search (Random Forest Regression)

GS\_rf\_model.fit(X\_train, y\_train)  
y\_pred = GS\_rf\_model.predict(X\_test)  
  
best\_accuracy\_score = GS\_rf\_model.best\_score\_  
best\_estimator = GS\_rf\_model.best\_estimator\_  
best\_parametrs = GS\_rf\_model.best\_params\_  
mse = mean\_squared\_error(y\_test, y\_pred)  
mae = mean\_absolute\_error(y\_test, y\_pred)  
final\_accuracy\_GRID = final\_accuracy\_GRID.append({'Model': 'Random Forest Regression', 'Accuracy(R2)': best\_accuracy\_score, 'Mean Squared Error': mse, 'Mean Absolute Error': mae, 'Best estimator': best\_estimator, 'Best parametrs': best\_parametrs }, ignore\_index=True)

Fitting 5 folds for each of 9 candidates, totalling 45 fits  
[CV 1/5] END max\_depth=3, n\_estimators=50; neg\_root\_mean\_squared\_error: (test=-132407.167) r2: (test=0.163) total time= 0.2s  
[CV 2/5] END max\_depth=3, n\_estimators=50; neg\_root\_mean\_squared\_error: (test=-129518.998) r2: (test=0.169) total time= 0.2s  
[CV 3/5] END max\_depth=3, n\_estimators=50; neg\_root\_mean\_squared\_error: (test=-123655.726) r2: (test=0.163) total time= 0.1s  
[CV 4/5] END max\_depth=3, n\_estimators=50; neg\_root\_mean\_squared\_error: (test=-124408.215) r2: (test=0.133) total time= 0.1s  
[CV 5/5] END max\_depth=3, n\_estimators=50; neg\_root\_mean\_squared\_error: (test=-137006.260) r2: (test=0.113) total time= 0.2s  
[CV 1/5] END max\_depth=3, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-132345.402) r2: (test=0.164) total time= 0.3s  
[CV 2/5] END max\_depth=3, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-129834.197) r2: (test=0.165) total time= 0.3s  
[CV 3/5] END max\_depth=3, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-123149.422) r2: (test=0.170) total time= 0.3s  
[CV 4/5] END max\_depth=3, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-123850.505) r2: (test=0.141) total time= 0.3s  
[CV 5/5] END max\_depth=3, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-136828.232) r2: (test=0.115) total time= 0.3s  
[CV 1/5] END max\_depth=3, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-132513.701) r2: (test=0.162) total time= 0.5s  
[CV 2/5] END max\_depth=3, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-129886.424) r2: (test=0.164) total time= 0.5s  
[CV 3/5] END max\_depth=3, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-122935.997) r2: (test=0.173) total time= 0.5s  
[CV 4/5] END max\_depth=3, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-123574.564) r2: (test=0.145) total time= 0.5s  
[CV 5/5] END max\_depth=3, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-136895.945) r2: (test=0.114) total time= 0.5s  
[CV 1/5] END max\_depth=5, n\_estimators=50; neg\_root\_mean\_squared\_error: (test=-130526.143) r2: (test=0.187) total time= 0.2s  
[CV 2/5] END max\_depth=5, n\_estimators=50; neg\_root\_mean\_squared\_error: (test=-127116.988) r2: (test=0.199) total time= 0.2s  
[CV 3/5] END max\_depth=5, n\_estimators=50; neg\_root\_mean\_squared\_error: (test=-122379.023) r2: (test=0.181) total time= 0.2s  
[CV 4/5] END max\_depth=5, n\_estimators=50; neg\_root\_mean\_squared\_error: (test=-123154.352) r2: (test=0.150) total time= 0.2s  
[CV 5/5] END max\_depth=5, n\_estimators=50; neg\_root\_mean\_squared\_error: (test=-136085.705) r2: (test=0.125) total time= 0.2s  
[CV 1/5] END max\_depth=5, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-130300.229) r2: (test=0.190) total time= 0.4s  
[CV 2/5] END max\_depth=5, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-126848.032) r2: (test=0.203) total time= 0.4s  
[CV 3/5] END max\_depth=5, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-121761.615) r2: (test=0.189) total time= 0.4s  
[CV 4/5] END max\_depth=5, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-123221.722) r2: (test=0.150) total time= 0.4s  
[CV 5/5] END max\_depth=5, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-134966.409) r2: (test=0.139) total time= 0.4s  
[CV 1/5] END max\_depth=5, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-129806.886) r2: (test=0.196) total time= 0.8s  
[CV 2/5] END max\_depth=5, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-126551.058) r2: (test=0.206) total time= 0.9s  
[CV 3/5] END max\_depth=5, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-121921.783) r2: (test=0.187) total time= 0.9s  
[CV 4/5] END max\_depth=5, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-122860.041) r2: (test=0.154) total time= 0.9s  
[CV 5/5] END max\_depth=5, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-135300.068) r2: (test=0.135) total time= 0.8s  
[CV 1/5] END max\_depth=7, n\_estimators=50; neg\_root\_mean\_squared\_error: (test=-129797.282) r2: (test=0.196) total time= 0.3s  
[CV 2/5] END max\_depth=7, n\_estimators=50; neg\_root\_mean\_squared\_error: (test=-126126.213) r2: (test=0.212) total time= 0.3s  
[CV 3/5] END max\_depth=7, n\_estimators=50; neg\_root\_mean\_squared\_error: (test=-122085.286) r2: (test=0.185) total time= 0.3s  
[CV 4/5] END max\_depth=7, n\_estimators=50; neg\_root\_mean\_squared\_error: (test=-124413.101) r2: (test=0.133) total time= 0.3s  
[CV 5/5] END max\_depth=7, n\_estimators=50; neg\_root\_mean\_squared\_error: (test=-135356.873) r2: (test=0.134) total time= 0.3s  
[CV 1/5] END max\_depth=7, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-128398.337) r2: (test=0.213) total time= 0.5s  
[CV 2/5] END max\_depth=7, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-125809.191) r2: (test=0.216) total time= 0.5s  
[CV 3/5] END max\_depth=7, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-121836.978) r2: (test=0.188) total time= 0.5s  
[CV 4/5] END max\_depth=7, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-123092.957) r2: (test=0.151) total time= 0.5s  
[CV 5/5] END max\_depth=7, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-134214.681) r2: (test=0.149) total time= 0.5s  
[CV 1/5] END max\_depth=7, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-128554.495) r2: (test=0.211) total time= 1.0s  
[CV 2/5] END max\_depth=7, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-125206.921) r2: (test=0.223) total time= 1.2s  
[CV 3/5] END max\_depth=7, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-121888.977) r2: (test=0.187) total time= 1.1s  
[CV 4/5] END max\_depth=7, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-122689.577) r2: (test=0.157) total time= 1.2s  
[CV 5/5] END max\_depth=7, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-134616.274) r2: (test=0.144) total time= 1.2s

#### Grid Search (Gradient Boosting Regression)

GS\_gb\_model.fit(X\_train, y\_train)  
y\_pred = GS\_gb\_model.predict(X\_test)  
  
best\_accuracy\_score = GS\_gb\_model.best\_score\_  
best\_estimator = GS\_gb\_model.best\_estimator\_  
best\_parametrs = GS\_gb\_model.best\_params\_  
mse = mean\_squared\_error(y\_test, y\_pred)  
mae = mean\_absolute\_error(y\_test, y\_pred)  
final\_accuracy\_GRID = final\_accuracy\_GRID.append({'Model': 'Gradient Boosting Regression', 'Accuracy(R2)': best\_accuracy\_score, 'Mean Squared Error': mse, 'Mean Absolute Error': mae, 'Best estimator': best\_estimator, 'Best parametrs': best\_parametrs }, ignore\_index=True)

Fitting 3 folds for each of 8 candidates, totalling 24 fits  
[CV 1/3] END learning\_rate=0.1, max\_depth=5, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-131242.435) r2: (test=0.189) total time= 0.5s  
[CV 2/3] END learning\_rate=0.1, max\_depth=5, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-123622.453) r2: (test=0.158) total time= 0.5s  
[CV 3/3] END learning\_rate=0.1, max\_depth=5, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-132008.352) r2: (test=0.114) total time= 0.5s  
[CV 1/3] END learning\_rate=0.1, max\_depth=5, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-133430.886) r2: (test=0.162) total time= 1.0s  
[CV 2/3] END learning\_rate=0.1, max\_depth=5, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-126935.715) r2: (test=0.113) total time= 1.0s  
[CV 3/3] END learning\_rate=0.1, max\_depth=5, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-133731.014) r2: (test=0.090) total time= 1.0s  
[CV 1/3] END learning\_rate=0.1, max\_depth=7, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-133811.615) r2: (test=0.157) total time= 0.7s  
[CV 2/3] END learning\_rate=0.1, max\_depth=7, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-128222.730) r2: (test=0.095) total time= 0.7s  
[CV 3/3] END learning\_rate=0.1, max\_depth=7, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-135392.228) r2: (test=0.068) total time= 0.7s  
[CV 1/3] END learning\_rate=0.1, max\_depth=7, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-136072.276) r2: (test=0.128) total time= 1.3s  
[CV 2/3] END learning\_rate=0.1, max\_depth=7, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-129522.899) r2: (test=0.076) total time= 1.3s  
[CV 3/3] END learning\_rate=0.1, max\_depth=7, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-135655.658) r2: (test=0.064) total time= 1.4s  
[CV 1/3] END learning\_rate=0.5, max\_depth=5, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-147072.464) r2: (test=-0.018) total time= 0.5s  
[CV 2/3] END learning\_rate=0.5, max\_depth=5, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-143734.320) r2: (test=-0.138) total time= 0.5s  
[CV 3/3] END learning\_rate=0.5, max\_depth=5, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-151427.148) r2: (test=-0.166) total time= 0.5s  
[CV 1/3] END learning\_rate=0.5, max\_depth=5, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-149283.918) r2: (test=-0.049) total time= 1.0s  
[CV 2/3] END learning\_rate=0.5, max\_depth=5, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-145035.084) r2: (test=-0.158) total time= 1.0s  
[CV 3/3] END learning\_rate=0.5, max\_depth=5, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-151435.904) r2: (test=-0.166) total time= 1.0s  
[CV 1/3] END learning\_rate=0.5, max\_depth=7, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-149493.349) r2: (test=-0.052) total time= 0.7s  
[CV 2/3] END learning\_rate=0.5, max\_depth=7, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-145715.278) r2: (test=-0.169) total time= 0.7s  
[CV 3/3] END learning\_rate=0.5, max\_depth=7, n\_estimators=100; neg\_root\_mean\_squared\_error: (test=-155494.686) r2: (test=-0.230) total time= 0.7s  
[CV 1/3] END learning\_rate=0.5, max\_depth=7, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-148941.555) r2: (test=-0.044) total time= 1.3s  
[CV 2/3] END learning\_rate=0.5, max\_depth=7, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-144201.895) r2: (test=-0.145) total time= 1.5s  
[CV 3/3] END learning\_rate=0.5, max\_depth=7, n\_estimators=200; neg\_root\_mean\_squared\_error: (test=-155238.919) r2: (test=-0.226) total time= 1.3s

#### Grid Search (SVR)

GS\_svr\_model.fit(X\_train, y\_train)  
y\_pred = GS\_svr\_model.predict(X\_test)  
  
best\_accuracy\_score = GS\_svr\_model.best\_score\_  
best\_estimator = GS\_svr\_model.best\_estimator\_  
best\_parametrs = GS\_svr\_model.best\_params\_  
mse = mean\_squared\_error(y\_test, y\_pred)  
mae = mean\_absolute\_error(y\_test, y\_pred)  
final\_accuracy\_GRID = final\_accuracy\_GRID.append({'Model': 'SVR', 'Accuracy(R2)': best\_accuracy\_score, 'Mean Squared Error': mse, 'Mean Absolute Error': mae, 'Best estimator': best\_estimator, 'Best parametrs': best\_parametrs }, ignore\_index=True)

Fitting 3 folds for each of 2 candidates, totalling 6 fits  
[CV 1/3] END C=0.1, epsilon=0.01; neg\_root\_mean\_squared\_error: (test=-210259.587) r2: (test=-1.081) total time= 4.3s  
[CV 2/3] END C=0.1, epsilon=0.01; neg\_root\_mean\_squared\_error: (test=-153288.076) r2: (test=-0.294) total time= 1.2s  
[CV 3/3] END C=0.1, epsilon=0.01; neg\_root\_mean\_squared\_error: (test=-181559.954) r2: (test=-0.677) total time= 6.1s  
[CV 1/3] END C=0.1, epsilon=0.1; neg\_root\_mean\_squared\_error: (test=-279774.421) r2: (test=-2.685) total time= 6.8s  
[CV 2/3] END C=0.1, epsilon=0.1; neg\_root\_mean\_squared\_error: (test=-187406.451) r2: (test=-0.934) total time= 2.5s  
[CV 3/3] END C=0.1, epsilon=0.1; neg\_root\_mean\_squared\_error: (test=-152180.909) r2: (test=-0.178) total time= 5.1s

#### Grid Search (KNN)

GS\_knn\_model.fit(X\_train, y\_train)  
y\_pred = GS\_knn\_model.predict(X\_test)   
  
best\_accuracy\_score = GS\_knn\_model.best\_score\_  
best\_estimator = GS\_knn\_model.best\_estimator\_  
best\_parametrs = GS\_knn\_model.best\_params\_  
mse = mean\_squared\_error(y\_test, y\_pred)  
mae = mean\_absolute\_error(y\_test, y\_pred)  
final\_accuracy\_GRID = final\_accuracy\_GRID.append({'Model': 'KNN', 'Accuracy(R2)': best\_accuracy\_score, 'Mean Squared Error': mse, 'Mean Absolute Error': mae, 'Best estimator': best\_estimator, 'Best parametrs': best\_parametrs }, ignore\_index=True)

Fitting 5 folds for each of 6 candidates, totalling 30 fits  
[CV 1/5] END n\_neighbors=3, weights=uniform; neg\_root\_mean\_squared\_error: (test=-160360.546) r2: (test=-0.227) total time= 0.0s  
[CV 2/5] END n\_neighbors=3, weights=uniform; neg\_root\_mean\_squared\_error: (test=-160037.999) r2: (test=-0.269) total time= 0.0s  
[CV 3/5] END n\_neighbors=3, weights=uniform; neg\_root\_mean\_squared\_error: (test=-161896.029) r2: (test=-0.434) total time= 0.0s  
[CV 4/5] END n\_neighbors=3, weights=uniform; neg\_root\_mean\_squared\_error: (test=-150367.756) r2: (test=-0.267) total time= 0.0s  
[CV 5/5] END n\_neighbors=3, weights=uniform; neg\_root\_mean\_squared\_error: (test=-153065.226) r2: (test=-0.107) total time= 0.0s  
[CV 1/5] END n\_neighbors=3, weights=distance; neg\_root\_mean\_squared\_error: (test=-160209.235) r2: (test=-0.225) total time= 0.0s  
[CV 2/5] END n\_neighbors=3, weights=distance; neg\_root\_mean\_squared\_error: (test=-159560.128) r2: (test=-0.261) total time= 0.0s  
[CV 3/5] END n\_neighbors=3, weights=distance; neg\_root\_mean\_squared\_error: (test=-161492.634) r2: (test=-0.427) total time= 0.0s  
[CV 4/5] END n\_neighbors=3, weights=distance; neg\_root\_mean\_squared\_error: (test=-151164.934) r2: (test=-0.280) total time= 0.0s  
[CV 5/5] END n\_neighbors=3, weights=distance; neg\_root\_mean\_squared\_error: (test=-153860.150) r2: (test=-0.119) total time= 0.0s  
[CV 1/5] END n\_neighbors=5, weights=uniform; neg\_root\_mean\_squared\_error: (test=-153632.444) r2: (test=-0.127) total time= 0.0s  
[CV 2/5] END n\_neighbors=5, weights=uniform; neg\_root\_mean\_squared\_error: (test=-151374.096) r2: (test=-0.135) total time= 0.0s  
[CV 3/5] END n\_neighbors=5, weights=uniform; neg\_root\_mean\_squared\_error: (test=-151358.929) r2: (test=-0.253) total time= 0.0s  
[CV 4/5] END n\_neighbors=5, weights=uniform; neg\_root\_mean\_squared\_error: (test=-145543.175) r2: (test=-0.187) total time= 0.0s  
[CV 5/5] END n\_neighbors=5, weights=uniform; neg\_root\_mean\_squared\_error: (test=-147621.003) r2: (test=-0.030) total time= 0.0s  
[CV 1/5] END n\_neighbors=5, weights=distance; neg\_root\_mean\_squared\_error: (test=-153226.078) r2: (test=-0.121) total time= 0.0s  
[CV 2/5] END n\_neighbors=5, weights=distance; neg\_root\_mean\_squared\_error: (test=-151063.485) r2: (test=-0.131) total time= 0.0s  
[CV 3/5] END n\_neighbors=5, weights=distance; neg\_root\_mean\_squared\_error: (test=-151973.000) r2: (test=-0.264) total time= 0.0s  
[CV 4/5] END n\_neighbors=5, weights=distance; neg\_root\_mean\_squared\_error: (test=-146272.055) r2: (test=-0.198) total time= 0.0s  
[CV 5/5] END n\_neighbors=5, weights=distance; neg\_root\_mean\_squared\_error: (test=-149028.055) r2: (test=-0.049) total time= 0.0s  
[CV 1/5] END n\_neighbors=7, weights=uniform; neg\_root\_mean\_squared\_error: (test=-150846.169) r2: (test=-0.086) total time= 0.0s  
[CV 2/5] END n\_neighbors=7, weights=uniform; neg\_root\_mean\_squared\_error: (test=-148656.074) r2: (test=-0.095) total time= 0.0s  
[CV 3/5] END n\_neighbors=7, weights=uniform; neg\_root\_mean\_squared\_error: (test=-147592.437) r2: (test=-0.192) total time= 0.0s  
[CV 4/5] END n\_neighbors=7, weights=uniform; neg\_root\_mean\_squared\_error: (test=-143036.090) r2: (test=-0.146) total time= 0.0s  
[CV 5/5] END n\_neighbors=7, weights=uniform; neg\_root\_mean\_squared\_error: (test=-149929.796) r2: (test=-0.062) total time= 0.0s  
[CV 1/5] END n\_neighbors=7, weights=distance; neg\_root\_mean\_squared\_error: (test=-150178.264) r2: (test=-0.076) total time= 0.0s  
[CV 2/5] END n\_neighbors=7, weights=distance; neg\_root\_mean\_squared\_error: (test=-148032.333) r2: (test=-0.086) total time= 0.0s  
[CV 3/5] END n\_neighbors=7, weights=distance; neg\_root\_mean\_squared\_error: (test=-148851.130) r2: (test=-0.212) total time= 0.0s  
[CV 4/5] END n\_neighbors=7, weights=distance; neg\_root\_mean\_squared\_error: (test=-143130.492) r2: (test=-0.148) total time= 0.0s  
[CV 5/5] END n\_neighbors=7, weights=distance; neg\_root\_mean\_squared\_error: (test=-150431.385) r2: (test=-0.069) total time= 0.0s

## Visualisation final accuracy scores

At this stage we are ready to analise the results of our models: Below are displayed:

1. \*\*final\_accuracy\_\_PCA\_GRID\*\* - the data frame which we were using to store data about models' accuracy using data after PCA and Scaling.
2. **final\_accuracy\_GRID** - the data frame that was used for storing the same data but without PCA and Scaling. Both tables include:
   * Model's name
   * Accuracy score
   * Mean Squared Error
   * Mean Absolute Error
   * Best estimator
   * Best parametrs
3. Diagram which shows how each model behaves in each condition.

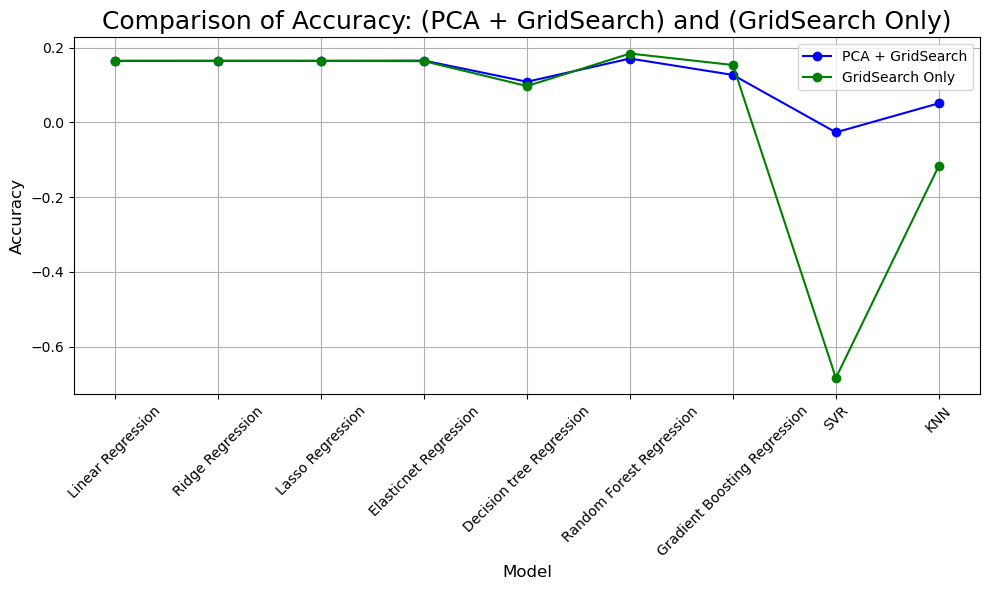
final\_accuracy\_PCA\_GRID.head(10)

Model Accuracy(R2) Mean Squared Error \  
0 Linear Regression 0.164840 1.706682e+10   
1 Ridge Regression 0.164964 1.706425e+10   
2 Lasso Regression 0.164863 1.706601e+10   
3 Elasticnet Regression 0.165595 1.704552e+10   
4 Decision tree Regression 0.108928 1.836247e+10   
5 Random Forest Regression 0.171037 1.686074e+10   
6 Gradient Boosting Regression 0.127201 1.743018e+10   
7 SVR -0.026538 2.098145e+10   
8 KNN 0.051309 1.947006e+10   
  
 Mean Absolute Error Best estimator \  
0 101427.373443 LinearRegression()   
1 101422.652841 Ridge(alpha=10)   
2 101424.611482 Lasso(alpha=10)   
3 101408.710235 ElasticNet(alpha=0.1)   
4 104208.562579 DecisionTreeRegressor(max\_depth=4)   
5 99843.049697 (DecisionTreeRegressor(max\_depth=7, max\_featur...   
6 101652.207899 ([DecisionTreeRegressor(criterion='friedman\_ms...   
7 109878.343648 SVR(C=0.1, epsilon=0.01, kernel='linear')   
8 107808.619689 KNeighborsRegressor(n\_neighbors=7, weights='di...   
  
 Best parametrs   
0 {}   
1 {'alpha': 10}   
2 {'alpha': 10}   
3 {'alpha': 0.1, 'l1\_ratio': 0.5}   
4 {'max\_depth': 4, 'min\_samples\_split': 2}   
5 {'max\_depth': 7, 'n\_estimators': 200}   
6 {'learning\_rate': 0.1, 'max\_depth': 5, 'n\_esti...   
7 {'C': 0.1, 'epsilon': 0.01}   
8 {'n\_neighbors': 7, 'weights': 'distance'}

final\_accuracy\_GRID.head(10)

Model Accuracy(R2) Mean Squared Error \  
0 Linear Regression 0.164840 1.706682e+10   
1 Ridge Regression 0.164896 1.707098e+10   
2 Lasso Regression 0.164890 1.706790e+10   
3 Elasticnet Regression 0.163969 1.717312e+10   
4 Decision tree Regression 0.097422 1.753796e+10   
5 Random Forest Regression 0.184424 1.648870e+10   
6 Gradient Boosting Regression 0.153750 1.714854e+10   
7 SVR -0.683891 5.818365e+10   
8 KNN -0.116213 2.312502e+10   
  
 Mean Absolute Error Best estimator \  
0 101427.373443 LinearRegression()   
1 101440.019034 Ridge(alpha=1)   
2 101429.883796 Lasso(alpha=10)   
3 101853.467572 ElasticNet(alpha=0.1, l1\_ratio=0.9)   
4 102031.607492 DecisionTreeRegressor(max\_depth=3)   
5 99235.575632 (DecisionTreeRegressor(max\_depth=7, max\_featur...   
6 101817.958356 ([DecisionTreeRegressor(criterion='friedman\_ms...   
7 142584.558870 SVR(C=0.1, epsilon=0.01, kernel='linear')   
8 119175.495168 KNeighborsRegressor(n\_neighbors=7)   
  
 Best parametrs   
0 {}   
1 {'alpha': 1}   
2 {'alpha': 10}   
3 {'alpha': 0.1, 'l1\_ratio': 0.9}   
4 {'max\_depth': 3, 'min\_samples\_split': 2}   
5 {'max\_depth': 7, 'n\_estimators': 200}   
6 {'learning\_rate': 0.1, 'max\_depth': 5, 'n\_esti...   
7 {'C': 0.1, 'epsilon': 0.01}   
8 {'n\_neighbors': 7, 'weights': 'uniform'}

plt.figure(figsize=(10, 6))  
plt.plot(final\_accuracy\_PCA\_GRID['Model'], final\_accuracy\_PCA\_GRID['Accuracy(R2)'], marker='o', linestyle='-', color='blue', label='PCA + GridSearch')  
plt.plot(final\_accuracy\_GRID['Model'], final\_accuracy\_GRID['Accuracy(R2)'], marker='o', linestyle='-', color='green', label='GridSearch Only')  
plt.xlabel('Model', fontsize=12)   
plt.ylabel('Accuracy', fontsize=12)  
plt.title('Comparison of Accuracy: (PCA + GridSearch) and (GridSearch Only)', fontsize=18)  
plt.xticks(rotation=45, fontsize=10)   
plt.yticks(fontsize=10)   
plt.legend(fontsize=10)   
plt.grid(True)  
plt.tight\_layout()  
plt.show()



# References

1. Nowlan, B., (2015). Housing Supply in Ireland: Perennial Problems and Sustainable Solutions. Sunday Business Property Conference Paper 25.09.15
2. Scuffil, C., (2022). Dublin’s Housing Crisis in Troubled Times. Dublin City Council Libraries Blog. Accessed on 03.09.22 at <https://www.dublincity.ie/library/blog/dublins-housing-crisis-troubled-times>
3. Phan, T. D. (2019) ‘Housing price prediction using machine learning algorithms: The case of Melbourne city, Australia’, Proceedings - International Conference on Machine Learning and Data Engineering, iCMLDE 2018. IEEE, pp. 8–13. DOI: 10.1109/iCMLDE.2018.00017.
4. Norris, M. & Byrne, M. (2017) A tale of two busts (and a boom): Irish social housing before and after the global financial crisis, Critical Housing Analysis, 4, pp. 19–28.
5. Nowicki, M., Brickell, K. & Harris, E. (2019) The hotelisation of the housing crisis: Experiences of family homelessness in Dublin hotels, The Geographical Journal, 185, pp. 313–324.
6. Byrne, M. (2020) Generation rent and the financialization of housing: A comparative exploration of the growth of the private rental sector in Ireland, the UK and Spain, Housing Studies, 35, pp. 743–765.
7. Bourassa, S. C., Cantoni, E. and Hoesli, M. (2011) ‘Predicting House Prices with Spatial Dependence:Impacts of Alternative Submarket Definitions’, SSRN Electronic Journal. DOI: 10.2139/ssrn.1090147.
8. Aswin S. R (2017), ‘Real Estate Price Prediction Using Machine Learning’ <https://norma.ncirl.ie/3096/1/aswinsivamravikumar.pdf>
9. Hujia Yu and Jiafu Wu., (2016), ' Real Estate Price Prediction with Regression and Classification' <https://cs229.stanford.edu/proj2016/report/WuYu_HousingPrice_report.pdf>
10. Ng, A. (2015). Machine Learning for a London Housing Price Prediction Mobile Application. Imperial College London. <http://www.doc.ic.ac.uk/~mpd37/theses/2015_beng_aaron-ng.pdf>

### Author: Ilia Grishkin

#### <https://github.com/Ilia-Grishkin/Machine_learning>

### Change log

| Date | Change Description | Changed By | Status |
| --- | --- | --- | --- |
| 19.03.2024 | Created title, structure | Ilia |  |
| 20.03.2024 | Created intoduction | Ilia | Pushed |
| 21.03.2024 | Created literate review, content | Ilia | Pushed |
| 22.03.2024 | Add data sourse | Ilia | Pushed |
| 24.03.2024 | Methods selection | Ilia | Pushed |
| 24.03.2024 | Liniar regressions | Ilia | Pushed |
| 24.03.2024 | SVM | Ilia | Planned |
| 24.03.2024 | Random forest | Ilia | Pushed |
| 31.03.2024 | GridSearchCV | Ilia | Pushed |
| 01.04.2024 | KNN | Ilia | Pushed |