National College of Ireland

Configuration Manual

MSc Research
Project Data Analytics

Selin Uluturk
x23160373

School of Computing
National College of Ireland

Supervisor: Dr. Christian Horn

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Selin Uluturk
Student ID:	x23160373
Programme:	M.Sc. Data Analytics **Year:** 2024
Module:	Research Project
Lecturer:	Dr. Christian Horn
Submission Due Date:	12.08.2024
Project Title:	Regression Analysis for Predicting Prices of Used Cars: A Study Utilizing Data from Car Trading Website
Word Count:	1054..... **Page Count:** 15 .....

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Selin Uluturk.....

Date: 11.08.2024.....

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Selin Uluturk
x23160373

1 Introduction

This document provides a comprehensive overview of the hardware and software requirements, as well as the various steps taken to successfully implement the research project - Regression Analysis for Predicting Prices of Used Cars: A Study Utilizing Data from Car Trading Websites.

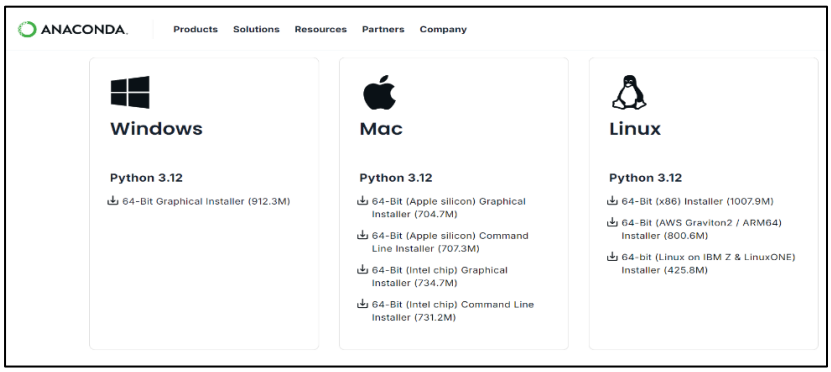
2 System Configuration

2.1 Hardware Requirements

- System OS: Windows 11
- Processor: AMD Ryzen 5 5500U
- RAM: 8 GB
- Graphics Card: Integrated Radeon Graphics

2.2 Software Requirements

This project uses Jupyter Notebook version 7.2.1 and Python version 3.11.5. Jupyter dependencies were installed using Anaconda Navigator.



3 Project Implementation

Since the analysis involves working with four datasets (df_h, df_l, df_ml, and df_mh) providing all the codes would result in excessively long documentation. Some codes are applied simultaneously across all four datasets; however, for tasks like outlier detection, different codes are used for each dataset and numeric variable. Given that the content of the

code is identical except for the variable names, a sample code is shared instead of 28 separate codes. Additionally, since different Jupyter files are used for each dataset in the modelling phase, only a sample code for one dataset is shared, as the codes are the same except for the dataset name.

3.1 Data Collection

In Cell 1, all lists of ads for a single search query are traversed, and simple information like the ad title, year, and URL for the complete ad are fetched. All this information is then dumped into a JSON file for enhancement later in Step 2. This script is executed separately for each brand (e.g., Fiat) for distinct search queries on the second-hand car website.

```
# Desired URL
url = "https://www.arabam.com/ikinci-el/otomobil/fiat-istanbul"

# create a new Firefox session
driver = webdriver.Firefox()
driver.implicitly_wait(30)
driver.get(url)

sleep(2)
#onetrust-accept-btn-handle
button = driver.find_element(By.CSS_SELECTOR, 'button#onetrust-accept-btn-handler')
button.click()

sleep(1)
button = driver.find_element(By.CSS_SELECTOR, 'button.fc-cta-consent')
button.click()

from selenium.webdriver.common.keys import Keys

def extracthotels():
        res = {}
        soup=BeautifulSoup(driver.page_source)
        cars = soup.select('tr.listing-list-item')
        print('found in page:', len(cars))
        for c in cars:
                name=c.select('.listing-modelname')[0].text.strip()
                url =c.select('.smallest-text-minus')[0]['href']
                items = c.select('td')
                k=0
                rr = {}
                for item in items:
                        rr[k] = item.text.strip()
                        k+=1
                res[url] =rr
                print(url)
        return res

ress = {}
prevlen = -1
typ = None
while True:
        try:
                driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
                sleep(1)
        except:
                pass
        try:
                driver.implicitly_wait(2)
                button = driver.find_elements(By.CSS_SELECTOR, 'a#pagingNext')
                if button[-1]:
                        button[-1].click()
                        sleep(2)
        except Exception as e:
                print(e)

        ress.update(extracthotels())
        if len(ress) == prevlen:
                break
        prevlen = len(ress)
        with open('arabam_fiat.json', 'w') as f:
            json.dump(ress, f)
        print('written unique:', len(ress))

print('DONE, find arabam.json for results')
```

```
found in page: 20
/ilan/sahibinden-satilik-fiat-palio-1-3-multijet-el/sahibinden-fiat-palio-1-3-multijet-el-2004-model-dusuk-km/24412919
```

Cell1: Jupyter Notebook Name: Research1 Notebook

```
from selenium import webdriver
from bs4 import BeautifulSoup
from selenium.webdriver.common.by import By
from time import sleep
import json

# Desired URL
url = "https://www.arabam.com/ikinci-el/otomobil?searchText=istanbul"

# create a new Firefox session
driver = webdriver.Firefox()
driver.implicitly_wait(30)
driver.get(url)

sleep(2)
#onetrust-accept-btn-handle
button = driver.find_element(By.CSS_SELECTOR, 'button#onetrust-accept-btn-handler')
button.click()

sleep(1)
button = driver.find_element(By.CSS_SELECTOR, 'button.fc-cta-consent')
button.click()

from selenium.webdriver.common.keys import Keys

with open('arabam_fiat.json', 'r') as f:
        ff = json.loads(f.read())
print(f"read {len(ff)} items")
num = 0
for url, v in ff.items():
        driver.get("https://www.arabam.com/" +url)
        soup=BeautifulSoup(driver.page_source)
        props = soup.select('.property-item')
        for prop in props[1:]:
                key = prop.select('.property-key')[0].text.strip()
                value = prop.select('.property-value')[0].text.strip()
                v[key]=value
        num+=1
        sleep(0.5)
        # if num == 5:
        #       break
        print(f"done {num}/{len(ff)}")
with open('arabam_fiat_all.json', 'w') as f:
    json.dump(ff, f)
print(f"done all {len(ff)}!")
```

```
read 980 items
done 1/980
done 2/980
done 3/980
```

Cell2: Jupyter Notebook Name: Research2 Notebook

In Cell 2, each of the ads found in the previously generated JSON file is enriched. The initial JSON file contained simple information like the ad title and URL, and this URL is used to navigate to the complete listing page to enrich the data. After navigating to the full listing page, all information related to the ad such as engine type, color, make, model, and premium features of the car is scraped. All of this fetched complete information is then aggregated on top of the first JSON file to get the full list of ads with various features/metadata.

It would be more useful to read JSON files instead of running this code, because prices may have changed on the date the data was collected and may change the results. To test the functionality of these codes, you can change the name of the JSON file in the code and save it as a different JSON file. The code that needs to be changed is: "with open('arabam_fiat_all.json', 'w') as f:"

3.2 Data Cleaning

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.model_selection import train_test_split, cross_val_score, cross_val_predict
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.neighbors import KNeighborsRegressor
import xgboost as xgb
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error

Fiat=pd.read_json("arabam_fiat_all.json")
```

Cell3: All libraries used in the study and reading JSON files - Research3 Notebook

JSON files belonging to 15 different car brands are read. In the example, there is a code sample that reads a file belonging to a brand. The same process is applied to other brands with the same code (with the file name changed). After the variable names of all data sets were synchronized, they were combined and saved as a single data set with the name "arabam_dataset".

```
all_rows.to_json('arabam_dataset.json', orient='records')
```

Cell 4: Saving as a JSON - Research3 Notebook

Every time I started working on a new Jupyter Notebook, the last saved JSON-CSV files were re-read. So, after running Cell4, I switch to the Jupyter notebook named "Research4".

```
df=pd.read_json("arabam_dataset.json")
```

Cell 5: Reading JSON file- Research4 Notebook

```
df.rename(columns={'3': 'Product_year'}, inplace=True)
df.rename(columns={'4': 'Mileage'}, inplace=True)
df.rename(columns={'5': 'Color'}, inplace=True)
df.rename(columns={'6': 'Price'}, inplace=True)
df.rename(columns={'Vites Tipi': 'Gear_Type'}, inplace=True)
df.rename(columns={'Yakıt Tipi': 'Fuel_Type'}, inplace=True)
df.rename(columns={'Kasa Tipi': 'Category'}, inplace=True)
df.rename(columns={'Motor Hacmi': 'Engine_Volume'}, inplace=True)
df.rename(columns={'Motor Gücü': 'Engine_Power'}, inplace=True)
df.rename(columns={'Seri': 'Model'}, inplace=True)
df.rename(columns={'Ort. Yakıt Tüketimi': 'Avr_fuel_consumption'}, inplace=True)
df.rename(columns={'Çekiş': 'Drive_Type'}, inplace=True)
df.rename(columns={'Yakıt Deposu': 'Fuel_Tank'}, inplace=True)
df.rename(columns={'Brand': 'Manufacturer'}, inplace=True)
```

Cell 6: Rename Columns-Research4 Notebook

```
color_translation = {
    'Beyaz': 'White',
    'Siyah': 'Black',
    'Gri': 'Gray',
    'Gri (Gümüş)': 'Gray (Silver)',
    'Füme': 'Smoke',
    'Kırmızı': 'Red',
    'Mavi': 'Blue',
    'Gri (metalik)': 'Gray (Metallic)',
    'Lacivert': 'Navy Blue',
    'Mavi (metalik)': 'Blue (Metallic)',
    'Kahverengi': 'Brown',
    'Bordo': 'Burgundy',
    'Bej': 'Beige',
    'Gri (titanyum)': 'Gray (Titanium)',
    'Yeşil': 'Green',
    'Yeşil (metalik)': 'Green (Metallic)',
    'Şampanya': 'Champagne',
    'Sarı': 'Yellow',
    'Diğer': 'Other',
    'Turuncu': 'Orange',
    'Altın': 'Gold',
    'Mor': 'Purple',
    'Turkuaz': 'Turquoise'
}
```

Cell 7: Translating Datas-Research4 Notebook

```
df.to_csv('arabam_last.csv', index=False)
```

Cell 8: Saving as CSV file- Research4 Notebook

After running Cell8, I switch to the Jupyter notebook named "Research5".

```
df = pd.read_csv("arabam_last.csv")
```

Cell 9: Reading CSV file-Research5 Notebook

```
columns_to_check = ['Model', 'Category', 'Gear_type', 'Fuel_type']
df.dropna(subset=columns_to_check, inplace=True)
print(df.isnull().sum())
```

```
Product_year            0
Mileage                 0
Price                   0
Model                   0
Category                0
Engine_Volume           0
Engine_Power            0
Avr_fuel_consumption    0
Fuel_Tank               0
Manufacturer            0
color                   0
Drive_type              0
Gear_type               0
Fuel_type               0
dtype: int64
```

Cell 10: Handling Null Values- Research5 Notebook

```
df = df[df['Product_year'] >= 2004]
#There are some very very old years and they don't represent the dataset. So i prefer to use last 20 years
```

Cell 11: Removing more than 20 ages cars-Research5 Notebook

```
manufacturers = cleaned_df['Manufacturer'].unique()

for i, manufacturer in enumerate(manufacturers, 1):
    globals()[f'df{i}'] = cleaned_df[cleaned_df['Manufacturer'] == manufacturer]
    print(f'The subset for {manufacturer} is named df{i}.')

print(df1.head())
print(df2.head())
```

```
The subset for Fiat is named df1.
The subset for Renault is named df2.
The subset for Opel is named df3.
The subset for Citroen is named df4.
The subset for Mercedes is named df5.
The subset for Bmw is named df6.
The subset for Ford is named df7.
The subset for Volkswagen is named df8.
The subset for Peugeot is named df9.
The subset for Toyota is named df10.
The subset for Hyundai is named df11.
The subset for Dacia is named df12.
The subset for Skoda is named df13.
The subset for Kia is named df14.
The subset for Audi is named df15.
```

Cell 12: Splitting dataset into 15 according to Manufacturers (Brands)- Research5 Notebook

Grouping Dataframes

```
# i will countinue in 4 groups according to Prices

df_high = pd.concat([df5, df6, df15]) # Mercedes, BMW, Audi

df_medium_high = pd.concat([df2, df8 ,df10, df13]) # Renault, Volkswagen, Toyota, Skoda

df_low = pd.concat([df1, df4, df9, df11, df12]) # Fiat, Citroen, Peugeot, Hyundai, Dacia

df_medium_low = pd.concat([df3, df7, df14]) # Opel, Ford, Kia

#Saving Json Files

df_high.to_json('df_high.json', orient='records')

df_medium_high.to_json('df_medium_high.json', orient='records')

df_low.to_json('df_low.json', orient='records')

df_medium_low.to_json('df_medium_low.json', orient='records')
```

Cells 13: Grouping datasets by Price averages and saving 4 JSON files-Research5 Notebook

After running Cell13, I switch to the Jupyter notebook named "Research6".

```
df_h=pd.read_json("df_high.json")

df_l=pd.read_json("df_low.json")

df_mh=pd.read_json("df_medium_high.json")

df_ml=pd.read_json("df_medium_low.json")
```

Cells 14: Reading 4 JSON files- Research6 Notebook

```
class_counts12 = df_l['Category'].value_counts()
print(class_counts12)
```

```
Category
Sedan            1311
Hatchback/5      1101
MPV                80
Station wagon      43
Hatchback/3        22
SUV                12
Cabrio              2
-                   2
Roadster            1
Pick-up             1
Name: count, dtype: int64
```

```
categories_to_remove8= ['-','Cabrio','Roadster','Pick-up']

df_l = df_l[~df_l['Category'].isin(categories_to_remove8)]
```

Cells 15: Determining categorical variable classes and deleting those with less than 10 data-Research6 Notebook

Saving Final Version as a JSon File

```
df_h.to_json('df_h.json', orient='records')
df_mh.to_json('df_mh.json', orient='records')
df_l.to_json('df_l.json', orient='records')
df_ml.to_json('df_ml.json', orient='records')
```

Cells 16: Saving as JSON files-Research6

After running Cell 16, I switch to the Jupyter notebook named "Research7".

```
df_h=pd.read_json("df_h.json")
df_l=pd.read_json("df_l.json")
df_mh=pd.read_json("df_mh.json")
df_ml=pd.read_json("df_ml.json")
```

Cells 17: Reading JSON files-Research7 Notebook

```
class_counts= df_mh['Drive_type'].value_counts()
print(class_counts) # Drop This variable (Because categorical data has 2 classes but a class has only 4 data)
```

```
Drive_type
Front-Wheel Drive (FWD)    2669
4WD (Full-Time)               4
Name: count, dtype: int64
```

```
df_mh.drop(columns=['Drive_type'], inplace=True)
```

```
class_counts1 = df_l['Drive_type'].value_counts()
print(class_counts1)#drop this variable (Because categorical data has only 1 class)
```

```
Drive_type
Front-Wheel Drive (FWD)    2569
Name: count, dtype: int64
```

```
df_l.drop(columns=['Drive_type'], inplace=True)
```

```
class_counts2 = df_ml['Drive_type'].value_counts()
print(class_counts2) #Drop this variable (Because categorical data has 2 classes but a class has only 2 data)
```

```
Drive_type
Front-Wheel Drive (FWD)    1661
4WD (Full-Time)               2
Name: count, dtype: int64
```

```
df_ml.drop(columns=['Drive_type'], inplace=True)
```

Cells 18: Droping “Drive Type” variable- Research7 Notebook

3.3 Data Pre-Processing

```
#This code is used to identify outliers in the 'Mileage' column.
#The standard deviation to be used in the code is determined separately for each variable,
#depending on the status of the outliers in the box-plot.
#These steps were performed separately for 4 different data sets and 7 numerical variables.
mean_mileage1 = df_h['Mileage'].mean()
std_mileage1 = df_h['Mileage'].std()
lower_bound_mileage1 = mean_mileage1 - 3 * std_mileage1
upper_bound_mileage1 = mean_mileage1 + 2 * std_mileage1
outliers_mileage1 = df_h[(df_h['Mileage'] < lower_bound_mileage1) | (df_h['Mileage'] > upper_bound_mileage1)]
len(outliers_mileage1)
```

```
52
```

```
#Replacing outliers with mean values
#These steps were performed separately for 4 different data sets and 7 numerical variables.
df_h.loc[df_h['Mileage'] < lower_bound_mileage1, 'Mileage'] = mean_mileage1
df_h.loc[df_h['Mileage'] > upper_bound_mileage1, 'Mileage'] = mean_mileage1
```

Cells 19: Handling Outliers- Research7 Notebook

```
#Log Transfor for all datasets and numeric variables
columns_to_transform = ['Mileage' ,'Engine_Volume', 'Engine_Power', 'Avr_fuel_consumption', 'Fuel_Tank']

for column in columns_to_transform:
    df_h[column] = np.log1p(df_h[column])
```

```
for column in columns_to_transform:
    df_l[column] = np.log1p(df_l[column])
```

```
for column in columns_to_transform:
    df_ml[column] = np.log1p(df_ml[column])
```

```
for column in columns_to_transform:
    df_mh[column] = np.log1p(df_mh[column])
```

Cells 20: Log Transformation for 4 datasets - Research7 Notebook

```
categorical_columns = ['Model', 'Category', 'Manufacturer', 'color', 'Drive_type', 'Gear_type', 'Fuel_type']

def apply_one_hot_encoding(df, all_columns):

    columns = [col for col in all_columns if col in df.columns]
    df_encoded = pd.get_dummies(df, columns=columns)

    df_encoded = df_encoded.astype(int)
    return df_encoded

df_h = apply_one_hot_encoding(df_h, categorical_columns)
df_l = apply_one_hot_encoding(df_l, categorical_columns)
df_ml = apply_one_hot_encoding(df_ml, categorical_columns)
df_mh = apply_one_hot_encoding(df_mh, categorical_columns)
```

Cell 21: One-Hot-Encoding for 4 datasets- Research7 Notebook

```
#Since there are 4 separate data sets, continuing in a single Jupyter notebook is confusing and since I have difficulties in
#I switch to different notebooks.
#For this reason, I saved the latest versions as a JSON file.

df_h.to_json('df_h2.json', orient='records')

df_mh.to_json('df_mh2.json', orient='records')

df_l.to_json('df_l2.json', orient='records')

df_ml.to_json('df_ml2.json', orient='records')
```

Cells 22: Saving JSON files- Research7 Notebook

After running Cell 22, I switch to the Jupyter notebook named:

- "Research8" for High Priced Dataset
- "Research9" for Medium-High Priced Dataset
- "Research10" for Medium-Low Priced Dataset
- "Research11" for Low Priced Dataset

I will continue with the Jupyter Notebook named Research8 because the same operations were applied to all the datasets belonging to the other price groups I listed above.

3.4 Regression Models

```
df_h=pd.read_json("df_h2.json")
```

Splitting Test and Train

```
X = df_h.drop("Price", axis = 1)
y = df_h["Price"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state=2316)
```

Cell 23: Splitting Test and Train- Research8 Notebook

Multiple Linear Regression

```
lm = sm.OLS(y_train, X_train)
```

```
model_mlr = lm.fit()
model_mlr.summary()
```

t[8]:

OLS Regression Results

Dep. Variable:	Price	**R-squared:**	0.829
Model:	OLS	**Adj. R-squared:**	0.821

Cells 23: Multiple Linear Regression- Research8 Notebook

```
def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

mape_mlr = mean_absolute_percentage_error(y_test, model_mlr.predict(X_test))
print(f'MAPE: {mape_mlr}%')
```

```
MAPE: 11.721268581052936%
```

Cell 24: MLR- MAPE value- Research8 Notebook

```
rf_model = RandomForestRegressor(random_state=2316)
rf_model.fit(X_train, y_train)

y_pred_train_rf= rf_model.predict(X_train)
y_pred_test_rf= rf_model.predict(X_test)

print(f'R2 Score: {r2_score(y_train, y_pred_train_rf)}')
print(f'Mean Squared Error: {mean_squared_error(y_test, y_pred_test_rf)}')
```

Cell 25: Random Forest and R2 Value- Research8 Notebook

```
def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

mape_rf = mean_absolute_percentage_error(y_test, y_pred_test_rf)
print(f'MAPE: {mape_rf}%')
```

```
MAPE: 10.696707617507464%
```

Cell 26: Cell 24: Random Forest- MAPE value- Research8 Notebook

XGBoost Model

```
DM_train = xgb.DMatrix(data = X_train, label = y_train)
DM_test = xgb.DMatrix(data = X_test, label = y_test)
```

```
xgb_model = XGBRegressor().fit(X_train, y_train)
```

Cells 27: XGBoost Model- Research8 Notebook

```
r2_xgb= r2_score(y_train, y_train_pred_xgb)
r2_xgb
```

```
0.95055291246971
```

Cells 28: XGBoost Model - R2- Research8 Notebook

```python
def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

mape_xgb = mean_absolute_percentage_error(y_test, y_pred_xgb)
print(f'MAPE: {mape_xgb}%')
```

```
MAPE: 11.229226472312268%
```

Cell 29: XGBoost- MAPE value- Research8 Notebook

SVR-Non Linear

```python
param_grid = {
    'C': [0.1, 1, 10, 100],
    'epsilon': [0.01, 0.1, 1, 10]
}

# SVR model (linear kernel)
svr = SVR(kernel='linear')

# Finding optimal hyperparameters with GridSearchCV
svr_cv_model = GridSearchCV(svr, param_grid, cv=5)
svr_cv_model.fit(X_train, y_train)
# Creating and training the SVR model with optimal parameters
best_params = svr_cv_model.best_params_

svr_tuned = SVR(kernel='linear', C=best_params['C'], epsilon=best_params['epsilon'])
svr_tuned.fit(X_train, y_train)
# Prediction - train
y_train_pred_svr = svr_tuned.predict(X_train)

# R squared
r2_train_svr = r2_score(y_train, y_train_pred_svr)
print(f"R-squared: {r2_train_svr}")
# Prediction - test
y_test_pred_svr = svr_tuned.predict(X_test)
mse_test_svr= mean_squared_error(y_test, y_test_pred_svr)
print(f"Mean Squared Error: {mse_test_svr}")
```

```
R-squared: 0.5303291026885577
Mean Squared Error: 50034447233.161835
```

Cell 30: SVR Model and R2- Research8 Notebook

```python
def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

mape_svr = mean_absolute_percentage_error(y_test, y_test_pred_svr)
print(f'MAPE: {mape_svr}%')
```

```
MAPE: 16.434638674156236%
```

Cell 31: SVR - MAPE Value- Research8 Notebook

3.5 Visualizations and Tables used in the Report

```
#All datasets "BEFORE" pre-processing steps
df_h=pd.read_json("df_h.json")
df_ml=pd.read_json("df_ml.json")
df_mh=pd.read_json("df_mh.json")
df_l=pd.read_json("df_l.json")
```

Cell 32: Reading JSON files (version of before pre-processing) for visualizations- Research12 Notebook

```
df_ml['Group'] = 'Medium-Low'
df_mh['Group'] = 'Medium-High'
df_l['Group'] = 'Low'
df_h['Group'] = 'High'
combined_df = pd.concat([df_ml, df_mh, df_l, df_h])
avg_price_df = combined_df.groupby(['Manufacturer', 'Group'], as_index=False).agg({'Price': 'mean'})
avg_price_df = avg_price_df.sort_values(by='Price')
palette = {"Low": "green", "Medium-Low": "blue", "Medium-High": "orange", "High": "red"}
# Plots
plt.figure(figsize=(14, 8))
sns.barplot(x="Manufacturer", y="Price", hue="Group", data=avg_price_df, palette=palette, ci=None, dodge=True, linewidth=9)
plt.xlabel("Manufacturer")
plt.ylabel("Price")
plt.title("Average Price by Manufacturer in Different Price Groups")
plt.xticks(rotation=45)
plt.legend(title='Price Group')
plt.show()
```

Figure 3: Average Price by Manufacturer in Different Price Groups- Research12 Notebook

```
plt.figure(figsize=(10, 6))
sns.histplot(df_h['Price'], kde=True)
plt.title('High Price Distribution')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.show()
```

Figure 4: Distribution of Price – High- Research12 Notebook

```
plt.figure(figsize=(10, 6))
sns.histplot(df_l['Price'], kde=True)
plt.title('Low Price Distribution')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.show()
```

Figure 5: Distribution of Price – Low- Research12 Notebook

```
plt.figure(figsize=(10, 6))
sns.histplot(df_mh['Price'], kde=True)
plt.title('Med-High Price Distribution')
plt.xlabel('Frequency')
plt.ylabel('Price')
plt.show()
```

Figure 6: Distribution of Price - Meduim High- Research12 Notebook

```
plt.figure(figsize=(10, 6))
sns.histplot(df_ml['Price'],kde=True,)
plt.title('Med-Low Price Distribution')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.show()
```

Figure 7: Distribution of Price - Medium-Low- Research12 Notebook

```
df_h['Dataset'] = 'High'
df_mh['Dataset'] = 'Medium-High'
df_ml['Dataset'] = 'Medium-Low'
df_l['Dataset'] = 'Low'

combined_df = pd.concat([df_h[['Price', 'Dataset']], df_mh[['Price', 'Dataset']], df_ml[['Price', 'Dataset']], df_l[['Price', 'Dataset']]])

# Plot the box plots
plt.figure(figsize=(12, 8))
sns.boxplot(x='Dataset', y='Price', data=combined_df)
plt.title('Box Plots of Prices for Different Datasets')
plt.xlabel('Dataset')
plt.ylabel('Price')
plt.show()
```

Figure 8: Box Plots of Prices of Different Datasets- Research12 Notebook

```
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix_df_h, annot=True, cmap='coolwarm', fmt='.2f')
plt.title(' High Price Correlation Matrix')
plt.show()
```

Figure 9: Heatmap-High Price- Research12 Notebook

```
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix_df_l, annot=True, cmap='coolwarm', fmt='.2f')
plt.title(' Low Price Correlation Matrix')
plt.show()
```

Figure 10: Heatmap-Low Price- Research12 Notebook

```
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix_df_ml, annot=True, cmap='coolwarm', fmt='.2f')
plt.title(' Medium-Low Price Correlation Matrix')
plt.show()
```

Figure 11: Heatmap-Med-Low Price- Research12 Notebook

```
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix_df_mh, annot=True, cmap='coolwarm', fmt='.2f')
plt.title(' Medium-High Price Correlation Matrix')
plt.show()
```

Figure 12: Heatmap-Med-High Price- Research12 Notebook

```
df_peugeot.head()
```

Table 4 : First Version of Dataset – Research3 Notebook

```
df_h.head()
```

Table 5: Last Version of Dataset-Research7 Notebook

```
importances_xgb = xgb_model.feature_importances_
feature_names = X_train.columns
importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances_xgb
})
importance_df = importance_df.sort_values(by='Importance', ascending=False).head(10)
plt.figure(figsize=(12, 8))
sns.barplot(x='Importance', y='Feature', data=importance_df)
plt.title('Top 10 Feature Importance for XGBoost')
```

Figure 15: Most Important Features by Price Groups- Research8 Notebook

```
def calculate_mape(y_true, y_pred):
    return np.abs((y_true - y_pred) / y_true) * 100

results_df = pd.DataFrame({
    'Actual': y_test,
    'Predicted': y_pred_test_rf
})

results_df['MAPE'] = calculate_mape(results_df['Actual'], results_df['Predicted'])

random_sample = results_df.sample(n=10, random_state=2316)
print(random_sample)
```

Table 8: Actual-Predicted Prices(Turkish Lira) and PE-Low Priced- Research11 Notebook

```
def calculate_mape(y_true, y_pred):
    return np.abs((y_true - y_pred) / y_true) * 100

results_df = pd.DataFrame({
    'Actual': y_test,
    'Predicted': y_pred_test_rf
})

results_df['MAPE'] = calculate_mape(results_df['Actual'], results_df['Predicted'])

random_sample = results_df.sample(n=10, random_state=992316)
print(random_sample)
```

Table 9: Actual-Predicted Prices(Turkish Lira) and PE-High Priced- Research8 Notebook