INT - 404

Artificial Intelligence Assignment

TASK:

To make a program in Python to predict the price of Used Cars.

Submitted to: Shabnam Ma'am	
Submitted by: ● Sumit Kumar Rajput 11802025 ● Rhituraj Sarkar 11802139 ● Siddharth Sahoo 11802128	
Section: K18GA	
GitHub link : https://github.com/sumit-rk/AI-Project	

Techniques Employed in Solving the problem :

- Linear Regression
- Decision Tree
- Random Forest Regression

The problem is being solved using the above techniques, and then the results are compared based on the accuracy of each method.

The training technique used is test and split, so that a particular part of the data is used to train the model and the other part of the data is used for the predictions.

And then based on the Root Mean Squared Error, the best model is selected.

Solving Methodology:

- First, the data is taken from the CSV file.
- Then, the data is inspected in order to determine the attributes that have a significant role in determining the price, like
- How many kilometers is the vehicle used for?
- When was the vehicle purchased?
- If there is any damage? etcetera...

Then based on these attributes the model is trained. And the predictions are made using the models.

<u>CODE:</u>

In [1]:
```
import matplotlib.pyplot as plt
import pandas as pd
#from pandas.tools.plotting import scatter_matrix
import pylab as pl
import numpy as np
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.preprocessing import StandardScaler
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
#from sklearn.preprocessing import Imputer
from sklearn.preprocessing import OneHotEncoder
import os
%matplotlib inline
```

In [2]:
```
data_frame = pd.read_csv("autos.csv",encoding="latin-1")
```

In [3]:
```
data_frame.head()
```

In [3]:
```
data_frame.head()
```


Out[3]:

	dateCrawled	name	seller	offerType	price	abtest	vehicleType	yearOfRegistration	gearbox	powerPS	model	kilometer	mont
0	2016-03-24 11:52:17	Golf_3_1.6	privat	Angebot	480	test	NaN	1993	manuell	0	golf	150000	
1	2016-03-24 10:58:45	A5_Sportback_2.7_Tdi	privat	Angebot	18300	test	coupe	2011	manuell	190	NaN	125000	
2	2016-03-14 12:52:21	Jeep_Grand_Cherokee_"Overland"	privat	Angebot	9800	test	suv	2004	automatik	163	grand	125000	
3	2016-03-17 16:54:04	GOLF_4_1_4__3TÜRER	privat	Angebot	1500	test	kleinwagen	2001	manuell	75	golf	150000	
4	2016-03-31 17:25:20	Skoda_Fabia_1.4_TDI_PD_Classic	privat	Angebot	3600	test	kleinwagen	2008	manuell	69	fabia	90000	

In [4]:

```
data_frame.describe()
```

Out[4]:

	price	yearOfRegistration	powerPS	kilometer	monthOfRegistration	nrOfPictures	postalCode
count	3.715280e+05	371528.000000	371528.000000	371528.000000	371528.000000	371528.0	371528.00000
mean	1.729514e+04	2004.577997	115.549477	125618.688228	5.734445	0.0	50820.66764
std	3.587954e+06	92.866598	192.139578	40112.337051	3.712412	0.0	25799.08247
min	0.000000e+00	1000.000000	0.000000	5000.000000	0.000000	0.0	1067.00000
25%	1.150000e+03	1999.000000	70.000000	125000.000000	3.000000	0.0	30459.00000
50%	2.950000e+03	2003.000000	105.000000	150000.000000	6.000000	0.0	49610.00000
75%	7.200000e+03	2008.000000	150.000000	150000.000000	9.000000	0.0	71546.00000
max	2.147484e+09	9999.000000	20000.000000	150000.000000	12.000000	0.0	99998.00000

In [5]:

```
missing_values = data_frame.isnull().sum()
missing_values
```

Out[5]:

```
dateCrawled              0
name                     0
seller                   0
offerType                0
price                    0
abtest                   0
vehicleType          37869
yearOfRegistration       0
gearbox              20209
powerPS                  0
model                20484
kilometer                0
monthOfRegistration      0
fuelType             33386
brand                    0
notRepairedDamage    72060
dateCreated              0
nrOfPictures             0
postalCode               0
lastSeen                 0
dtype: int64
```

In [6]:

```
cat_val = ["seller","offerType","abtest","gearbox","fuelType","notRepairedDamage","nrOfPictures"]

for col in cat_val:
    print ([col]," : ",data_frame[col].unique())
```

```
['seller']  :  ['privat' 'gewerblich']
['offerType']  :  ['Angebot' 'Gesuch']
['abtest']  :  ['test' 'control']
['gearbox']  :  ['manuell' 'automatik' nan]
['fuelType']  :  ['benzin' 'diesel' nan 'lpg' 'andere' 'hybrid' 'cng' 'elektro']
['notRepairedDamage']  :  [nan 'ja' 'nein']
['nrOfPictures']  :  [0]
```

In [7]:

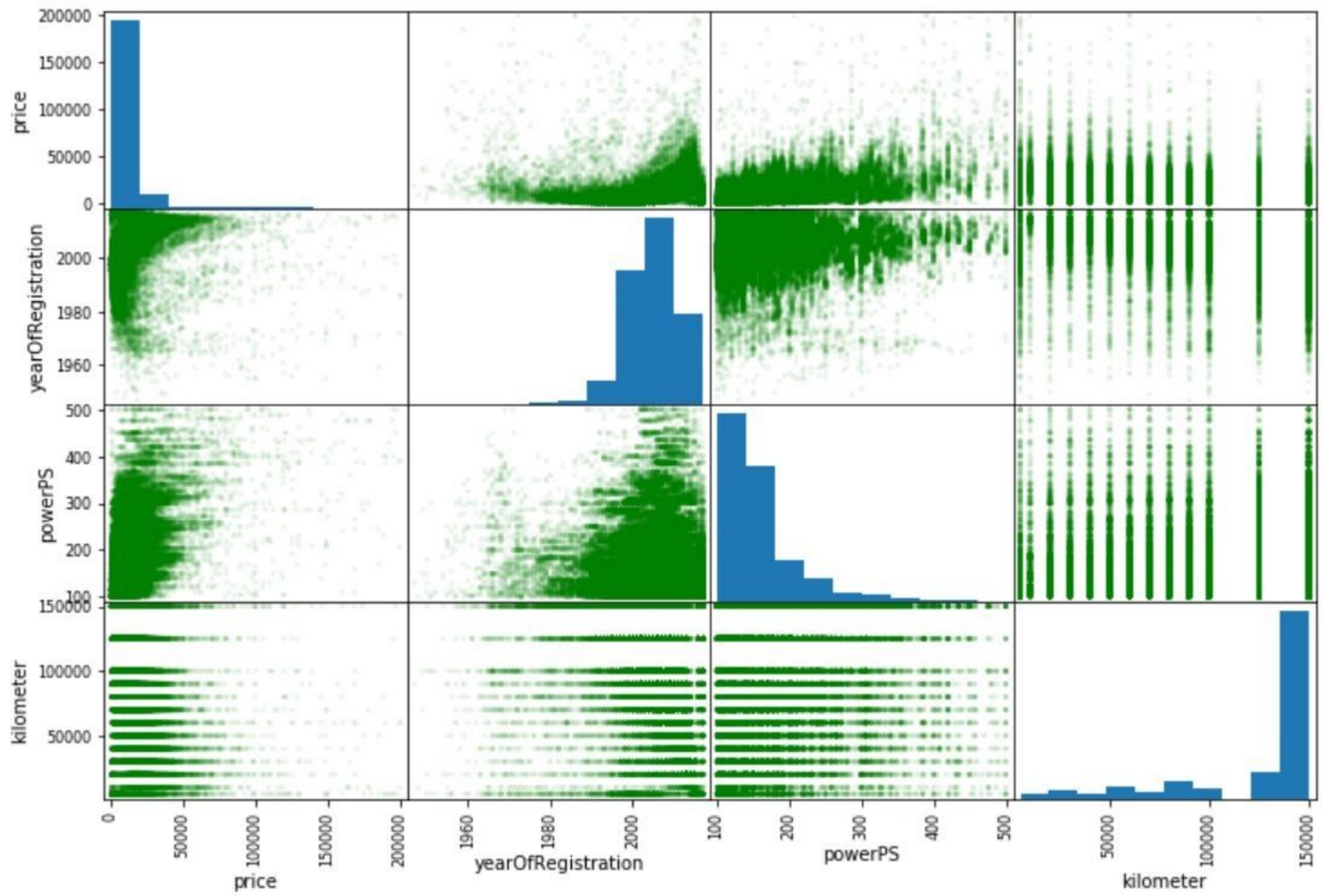
```
cars_cl = data_frame.copy()
cars_cl = cars_cl [

                    (cars_cl["yearOfRegistration"].between(1945,2017,inclusive=True)) &
                    (cars_cl["powerPS"].between(100,500,inclusive=True)) &
                    (cars_cl["price"].between(100,200000,inclusive=True))
                  ]
```

In [8]:

```
num_attir = ["price" , "yearOfRegistration" , "powerPS" , "kilometer"]
%matplotlib inline
pd.plotting.scatter_matrix(cars_cl[num_attir],figsize=(12,8),alpha = 0.08,color='green')
```

Out[8]:

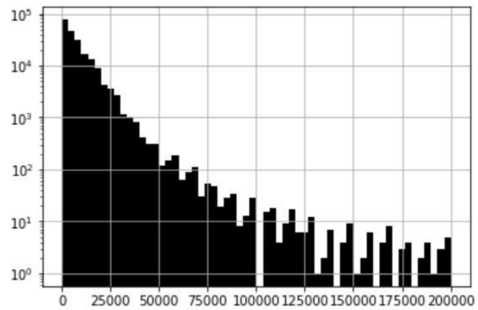
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001CEFC7C1608>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001CEF0BE8F88>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001CEF0C29048>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001CEF0C5F148>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001CEF0C96248>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001CEF0CCF348>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001CEF0D07448>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001CEF0D40548>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001CEF0D4C148>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001CEF0D84348>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001CEF0DE9848>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001CEF0E20988>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001CEF0E59A88>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001CEF0E90B88>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001CEF0EC8CC8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001CEF0F02D48>]],
      dtype=object)
```



In [9]:

```
cars_cl["price"].hist(bins = 60, log = True,color='black')
```

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x1cef0646548>



In [10]:

```
cars_clean = data_frame.copy()

# Filtering the irrevelent data.

cars_clean = cars_clean[
    (cars_clean["yearOfRegistration"].between(1945, 2017, inclusive=True)) &
    (cars_clean["powerPS"].between(100, 500, inclusive=True)) &
    (cars_clean["price"].between(100, 200000, inclusive=True)) &
    (cars_clean["offerType"] == "Angebot")
]

# Replacing the empty/ NaN values in  the data.

cars_clean['vehicleType'].fillna(value='blank', inplace=True)
cars_clean['gearbox'].fillna(value='blank', inplace=True)
cars_clean['model'].fillna(value='blank', inplace=True)
cars_clean['fuelType'].fillna(value='blank', inplace=True)
cars_clean['notRepairedDamage'].fillna(value='blank', inplace=True)

# Change categorical attributes dtype to category

for col in cars_clean:
    if cars_clean[col].dtype == "object":
        cars_clean[col] = cars_clean[col].astype('category')

# Assign codes to categorical attribues instead of strings

cat_columns = cars_clean.select_dtypes(['category']).columns

cars_clean[cat_columns] = cars_clean[cat_columns].apply(lambda x: x.cat.codes)


# Dropping useless columns

drop_cols = ["dateCrawled", "abtest", "dateCreated", "nrOfPictures", "lastSeen"]

cars_clean = cars_clean.drop(drop_cols, axis=1)
```

In [11]:
```
cars_clean.head()
```

Out[11]:

	name	seller	offerType	price	vehicleType	yearOfRegistration	gearbox	powerPS	model	kilometer	monthOfRegistration	fuelType	brand	notRepairedDa
1	2136	1	0	18300	4	2011	2	190	50	125000	5	4	1	
2	56312	1	0	9800	8	2004	0	163	115	125000	8	4	14	
5	20652	1	0	650	7	1995	2	102	11	150000	10	1	2	
6	87933	1	0	2200	3	2004	2	109	8	150000	8	1	25	
8	44386	1	0	14500	2	2014	2	125	59	30000	8	1	10	

In [12]:
```
# Getting the train and test sets
train_set, test_set = train_test_split(cars_clean, test_size =0.2, random_state = 42)

# Separation of Features and Labels

cars_price = train_set["price"].copy()
cars = train_set.drop("price", axis=1)
#cars_price
```

In [13]:
```

# Create a class to select numerical or categorical columns
# since Scikit-Learn doesn't handle DataFrames yet
class DFSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attribute_names].values
```

In [14]:
```
# Setting categorical and numerical attributes

cat_attribs = ["name", "seller", "offerType", "vehicleType", "fuelType", "brand", "notRepairedDamage"]
num_attribs = list(cars.drop(cat_attribs, axis=1))

# Building the Pipelines

num_pipeline = Pipeline([
    ("selector", DFSelector(num_attribs)),
    ("std_scaler", StandardScaler())
])

cat_pipeline = Pipeline([
    ("selector", DFSelector(cat_attribs)),
    ("encoder", OneHotEncoder(sparse=True))
])

full_pipeline = FeatureUnion(transformer_list =[
    ("num_pipeline", num_pipeline),
    ("cat_pipeline", cat_pipeline)
])

```

In [15]:
```
cars_prepared = full_pipeline.fit_transform(cars)
```

In [16]:
```
lin_reg = LinearRegression()
lin_reg.fit(cars_prepared,cars_price)
```

Out[16]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

In [17]:
```python
from sklearn.metrics import mean_squared_error
```

In [19]:
```python
cars_predictions = lin_reg.predict(cars_prepared)
lin_mse = mean_squared_error(cars_price, cars_predictions)
lin_rmse = np.sqrt(lin_mse)
lin_rmse
```

Out[19]: 1827.3421981985368

In [20]:
```python
cars_predictions[0:4]
```

Out[20]: array([25749.93597856, 4000.11431413, 1118.64918339, 2599.98254437])

In [21]:
```python
list(cars_price[0:4])
```

Out[21]: [25750, 4000, 800, 2600]

In [23]:
```python
# Decision Tree Approach
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor(random_state=42)
tree_reg.fit(cars_prepared, cars_price)
```

Out[23]:
```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=42, splitter='best')
```

In [ ]:
```python
cars_predictions = tree_reg.predict(cars_prepared)
tree_mse = mean_squared_error(cars_price, cars_predictions)
tree_rmse = np.sqrt(tree_mse)
tree_rmse
```

In [*]:
```python
# Random Forest Approach
from sklearn.ensemble import RandomForestRegressor

forest_reg = RandomForestRegressor(random_state=42, n_jobs =-1, max_depth = 30 )
forest_reg.fit(cars_prepared, cars_price)
```

In [*]:
```python
cars_predictions = forest_reg.predict(cars_prepared)
forest_mse = mean_squared_error(cars_price, cars_predictions)
forest_rmse = np.sqrt(forest_mse)
forest_rmse
```

In [*]:
```python
# Cross Validation

from sklearn.model_selection import cross_val_score

def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())
```

In [*]:
```python
# LinReg - CrossValidation
scores = cross_val_score(lin_reg, cars_prepared, cars_price,
                         scoring="neg_mean_squared_error", cv=4)
lin_rmse_scores = np.sqrt(-scores)

display_scores(lin_rmse_scores)
```

In [*]:
```python
# Decision Tree - CrossValidation

scores = cross_val_score(tree_reg, cars_prepared, cars_price,
                         scoring="neg_mean_squared_error", cv=4)
tree_rmse_scores = np.sqrt(-scores)

display_scores(tree_rmse_scores)
```

In [*]:

```
#Random Forest- CrossValidation

from sklearn.model_selection import cross_val_score

scores = cross_val_score(forest_reg, cars_prepared, cars_price,
                         scoring="neg_mean_squared_error", cv=2)
forest_rmse_scores = np.sqrt(-scores)

display_scores(forest_rmse_scores)
```

In [*]:

```
# To see the importance of the features
feature_importances = forest_reg.feature_importances_
feature_importances
```

In [*]:

```
cat_encoder = cat_pipeline.named_steps["encoder"]
#cat_one_hot_attribs = list(cat_encoder.categories_[0])
attributes = num_attribs #+ cat_encoder
sorted(zip(feature_importances, attributes), reverse=True)
```

In [*]:

```
# Final Model
final_model = forest_reg

cars_test = test_set.drop("price", axis = 1)
cars_price_test = test_set["price"].copy()

cars_test_prepared = full_pipeline.transform(cars_test) ## call transform NOT fit_transform


from sklearn.metrics import mean_squared_error
final_predictions = final_model.predict(cars_test_prepared)

final_mse = mean_squared_error(cars_price_test, final_predictions)

final_rmse = np.sqrt(final_mse)
#to see the results.
#final_predictions
```

Using test and split technique, we can say that the **Forest Regression Approach** is most accurate, but time consuming.