CCT College Dublin

Assessment Cover Page

Module Title:	Machine Learning
Assessment Title:	CA1: Machine Learning
Lecturer Name:	Marina Iantorno
Student Full Name:	Laercio Santos Lima
Student Number:	2022055
Assessment Due Date:	17/04/2022
Date of Submission:	17/04/2022

Declaration

By submitting this assessment, I confirm that I have read the CCT policy on Academic Misconduct and understand the implications of submitting work that is not my own or does not appropriately reference material taken from a third party or other source. I declare it to be my own work and that all material from third parties has been appropriately referenced. I further confirm that this work has not previously been submitted for assessment by myself or someone else in CCT College Dublin or any other higher education institution.

ABSTRACT

Machine Learning (ML) can be explained as a scientific study of statistical models and algorithms. It intends to capacity computer systems to conclude tasks that they were not originally programmed to do, keeping an acceptable level of accuracy. One of the good advantages of working with ML models is that once the algorithms understand the data patterns and learn with it, it achieves a level of automation that may be seen during the performed tasks.

In this paper, different types of regression models and classification models were used. There is also a brief review of the main purpose of the models used and their characteristics. Besides that, in order to be able to work with the dataset, it is also included in this paper some exploratory data analysis and data cleaning.

Keywords: Algorithm, Machine Learning, Supervised learning, Regression Models, Classification Models, Python, Jupyter Notebook

INTRODUCTION

Machine Learning is a tool that can be used to have a better idea of the data that has been constantly collected. By doing so, it is possible to understand this data, and after that, be able to transform it into information, and consequently, be able to predict future events.

This project intends to better understand some characteristics of supervised machine learning models. The dataset used in this project is about used cars. It contains 12 columns and over 6 thousand rows. One of the main goals of this CA is to perform Regression and Classification Models. Additionally, the features in the dataset will be treated in order to run the models.

As some features in the dataset are categorical and others are numerical, this project intends to demonstrate the performance of the different models and decide which one would be the best fit for the analysis.

In addition, after discovering which model performs better for the different features used, these models will be used to predict the price and the fuel type.

Moreover, besides doing exploratory data analysis, problems and difficulties such as missing values will be addressed.

Jupyter Notebook was used for all the calculations and pictures used in this report. It is important to clarify that even if this report contains some pictures of the Jupyter Notebook, not everything that was done there is included in this report. In other words, I just included in this report the images that I understand are important. However, as the Jupyter Notebook is going to be submitted with this report, all details regarding the code can be checked there.

To conclude, this project will provide details of some of the steps that are necessary in order to perform machine learning models. A short explanation of the main idea of different models is also included in this project.

VARIABLES DICTIONARY

	Name	Qualitative / Quantitative	type	Level of Measurement	Definition	Coding Options
0	name	Qualitative	string	nominal	name of the car	enter a string
1	year	Quantitative	integer	interval	year of the car	enter a year
2	selling_price	Quantitative	integer	ratio	price of the car	enter price
3	fuel	Qualitative	string	nominal	type of fuel	Diesel or Petrol
4	km_driven	Quantitative	integer	ratio	km the car was driven	enter a number
5	seller_type	Qualitative	string	nominal	type of the seller	Individual; Dealer; Trustmark Dealer
6	transmission	Qualitative	string	nominal	type of transmission	Manual; Automatic
7	owner	Qualitative	string	nominal	quantity of owners the car had	First Owner; Second Owner, Third Owner, Fourth...
8	seats	Quantitative	integer	interval	number of seats	enter a number
9	engine_value	Quantitative	integer	ratio	engine value in CC	enter a number
10	max_power_value	Quantitative	float	ratio	max power value in bhp	enter a number
11	mileage_value	Quantitative	float	ratio	mileage value in kmpl	enter a number

Variables Dictionary

CHARACTERISTICS OF THE DATASET

It contains 12 columns and 6353 rows. It also contains categorical and numerical data.

```
# showing the first rows
df.head()
```

	name	year	selling_price	fuel	km_driven	seller_type	transmission	owner	seats	engine_value	max_power_value	mileage_value
0	Maruti Alto 800 VXI	2020	250999	Petrol	30000	Individual	Manual	First Owner	5	796	47.30	22.05
1	Maruti Celerio X ZXI	2020	250000	Petrol	120000	Individual	Manual	Second Owner	5	998	67.00	21.63
2	Maruti Wagon R LXI	2020	265000	Petrol	70000	Individual	Manual	Second Owner	5	998	67.05	21.79
3	Volkswagen Polo 1.5 TDI Comfortline	2020	260000	Diesel	50000	Individual	Manual	First Owner	5	1498	88.50	20.14
4	Maruti Alto 800 LXI	2020	350000	Petrol	5000	Individual	Manual	First Owner	5	796	47.30	22.05

df

```
# Showing some info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6353 entries, 0 to 6352
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   name             6353 non-null   object 
 1   year             6353 non-null   int64  
 2   selling_price    6353 non-null   int64  
 3   fuel             6353 non-null   object 
 4   km_driven        6353 non-null   int64  
 5   seller_type      6353 non-null   object 
 6   transmission     6353 non-null   object 
 7   owner            6353 non-null   object 
 8   seats            6353 non-null   int64  
 9   engine_value     6353 non-null   int64  
 10  max_power_value  6353 non-null   float64
 11  mileage_value    6353 non-null   float64
dtypes: float64(2), int64(5), object(5)
memory usage: 595.7+ KB
```

info()

```
# Showing some info
df.describe()
```

	year	selling_price	km_driven	seats	engine_value	max_power_value	mileage_value
count	6353.000000	6.353000e+03	6.353000e+03	6353.000000	6353.000000	6353.000000	6353.000000
mean	2013.481820	4.496766e+05	7.455315e+04	5.419959	1398.478829	84.438516	19.623666
std	3.907949	2.592417e+05	5.941551e+04	0.978802	462.268275	25.904465	3.921903
min	1994.000000	2.999900e+04	1.000000e+03	2.000000	624.000000	34.200000	0.000000
25%	2011.000000	2.500000e+05	4.000000e+04	5.000000	1197.000000	67.100000	17.000000
50%	2014.000000	4.000000e+05	7.000000e+04	5.000000	1248.000000	81.800000	19.700000
75%	2017.000000	6.050000e+05	1.000000e+05	5.000000	1498.000000	98.600000	22.540000
max	2020.000000	1.325000e+06	2.360457e+06	14.000000	3498.000000	272.000000	28.400000

describe()

```
# looking for objects
df.describe(include = object)
```

	name	fuel	seller_type	transmission	owner
count	6353	6353	6353	6353	6353
unique	1801	2	3	2	4
top	Maruti Swift Dzire VDI	Diesel	Individual	Manual	First Owner
freq	118	3415	5776	5960	3899

describe()

OUTLIERS

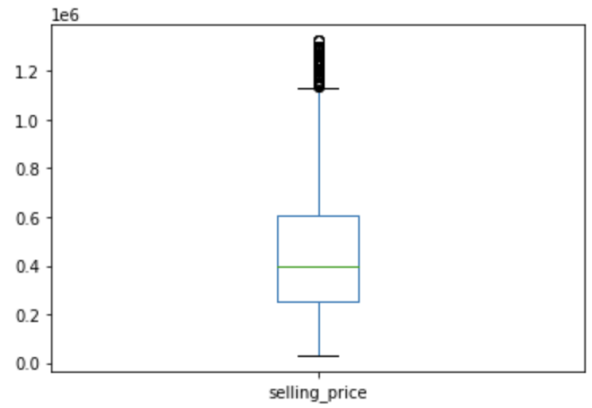
It is always important to be aware of any outliers in the dataset. For this project, I decided to look for them using a box plot. According to Galarnyk (2018), everything that was plotted out of the whiskers as points is considered outliers.

As it is possible to confirm below, there are some outliers in this dataset. However, I decided that they are important for the analysis, and consequently, I decided to keep them in the dataset.

The pictures below show 2 examples of outliers that can be found in the dataset.

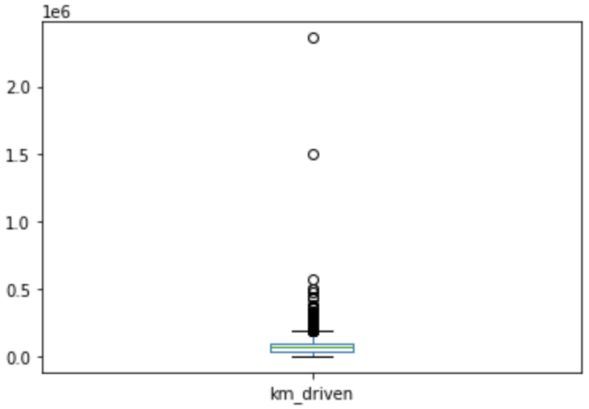
```
# Using Plot box to detect outliers
df["selling_price"].plot.box()
```

```
<AxesSubplot:>
```




```
# Using Plot box to detect outliers
df["km_driven"].plot.box()
```

```
<AxesSubplot:>
```



Looking for Outliers

DUPLICATES RECORDS

According to Bhutani (2018), another important part of data analysis is to look for duplicate values, and subsequently, decide to remove them or not.

Based on that, I looked for duplicated rows in the dataset. In order to do it, first I had to create a new dataframe. After that, I was able to show the amount of rows that are duplicated.

As the below picture demonstrates, there are no duplicate values in the dataset.

```
# looking for duplicated rows
dup_df = df[df.duplicated()]
```

```
# Showing the number of duplicated rows
print("Number of duplicated rows: ", dup_df.shape)
```

```
Number of duplicated rows:  (0, 12)
```

Looking for duplicate values

CHECKING MISSING VALUES

Identifying missing values in a dataset is also another important step when there is an intention to perform ML models. According to Chandradas (2021), NaN is a short form for Not A Number. In other words, it is a possible form to show a missing value in a dataset.

As it is possible to see below, I decided to look and count the amount of missing values in the dataset used for this project. In order to have it done, I used isna() and sum(). The image also shows that there are no missing values in this dataset.

```
#looking for missing values
df.isna().sum()
```

```
name               0
year               0
selling_price      0
fuel               0
km_driven          0
seller_type        0
transmission       0
owner              0
seats              0
engine_value       0
max_power_value    0
mileage_value      0
dtype: int64
```

Looking for missing values

EXPLORATORY DATA ANALYSIS

DEFINITION

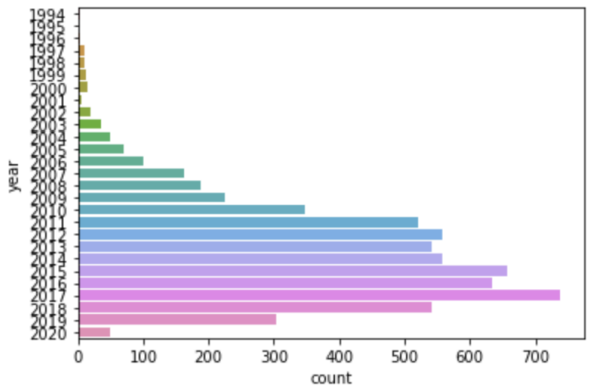
Exploratory Data Analysis (EDA) is an important step for this project since it is in this part where I am able to identify features patterns. According to Patil (2018), EDA can be defined as a critical process of performing initial investigations in a dataset.

As previously mentioned in this report, ML models will be performed. Consequently, here in the EDA I intend to discover the necessary patterns and characteristics that will help me to build better models in the future. Besides that, any extra anomalies spotted in this stage may be treated in future steps.

DETAILING THE CATEGORICAL VARIABLES

```
# Showing categorical variables
sns.countplot(y='year', data=df)
```

```
<AxesSubplot:xlabel='count', ylabel='year'>
```



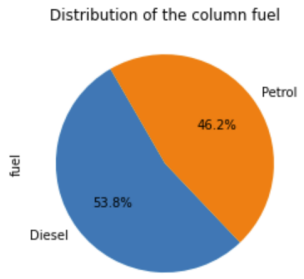

```
# Showing categorical in numbers
df["year"].value_counts()
```

```
2017    737
2015    656
2016    635
2014    558
2012    557
2013    542
2018    541
2011    521
2010    347
2019    304
2009    225
2008    188
2007    162
2006    101
2005     70
2004     49
2020     49
2003     36
2002     19
2000     14
1999     13
1997      9
1998      9
2001      6
1994      2
1996      2
1995      1
Name: year, dtype: int64
```

year

```
# Showing the percentage of Diesel and Petrol
mylabels_f = ["Diesel", "Petrol"]
pie_fuel = df["fuel"].value_counts().plot(kind='pie',
                                          autopct="%1.1f%%",
                                          startangle = 120,
                                          labels = mylabels_f)
pie_fuel.set_title("Distribution of the column fuel")
```

```
Text(0.5, 1.0, 'Distribution of the column fuel')
```



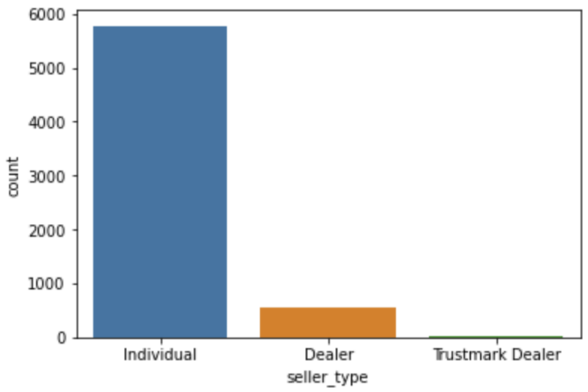

```
# Showing fuel in numbers
df["fuel"].value_counts()
```

```
Diesel    3415
Petrol    2938
Name: fuel, dtype: int64
```

fuel

```
# Showing categorical variables
sns.countplot(x='seller_type', data=df)
```

```
<AxesSubplot:xlabel='seller_type', ylabel='count'>
```



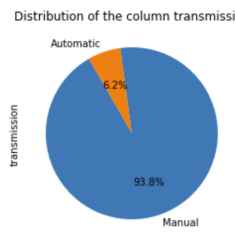

```
# Showing seller_type in numbers
df["seller_type"].value_counts()
```

```
Individual          5776
Dealer               551
Trustmark Dealer      26
Name: seller_type, dtype: int64
```

seller_type

```
# Showing the percentage of Diesel and Petrol
mylabels_t = ["Manual", "Automatic"]
pie_transmission = df["transmission"].value_counts().plot(kind='pie',
                                                          autopct="%1.1f%%",
                                                          startangle = 120,
                                                          labels = mylabels_t)
pie_transmission.set_title("Distribution of the column transmission")
```

```
Text(0.5, 1.0, 'Distribution of the column transmission')
```



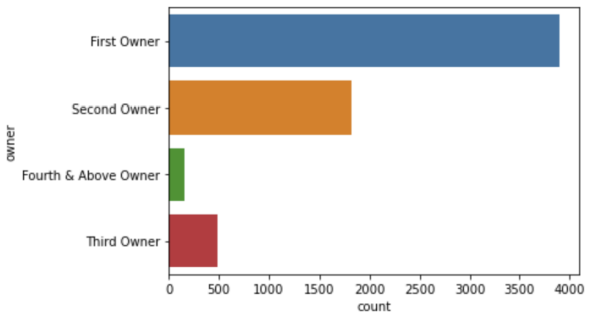

```
# Showing trnamission in numbers
df["transmission"].value_counts()
```

```
Manual       5960
Automatic     393
Name: transmission, dtype: int64
```

transmission

```
# Showing categorical variables
sns.countplot(y='owner', data=df)
```

```
<AxesSubplot:xlabel='count', ylabel='owner'>
```




```
# Showing owner in numbers
df["owner"].value_counts()
```

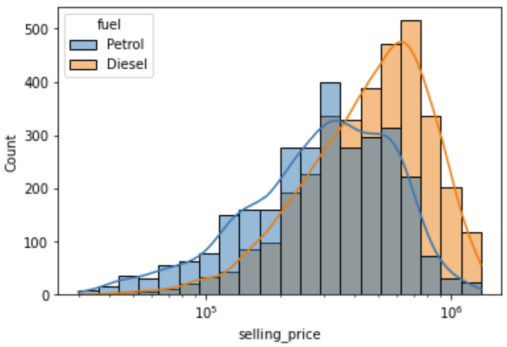
```
First Owner             3899
Second Owner            1817
Third Owner              484
Fourth & Above Owner     153
Name: owner, dtype: int64
```

owner

DETAILING THE NUMERICAL VARIABLES

```
# Detailing the numerical variables
sns.histplot(data=df, x="selling_price",
             kde=True, log_scale=True,
             bins=20, hue="fuel")
```

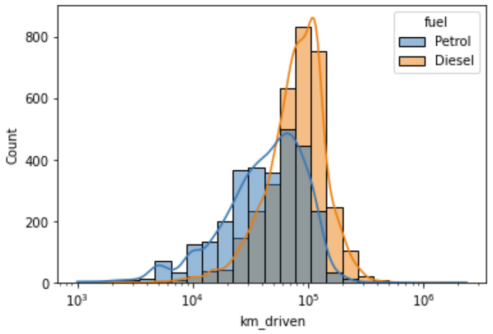
```
<AxesSubplot:xlabel='selling_price', ylabel='Count'>
```



selling_price

```
#Detailing the numerical variables
sns.histplot(data=df, x="km_driven",
             kde=True, log_scale=True,
             bins=25, hue="fuel")
```

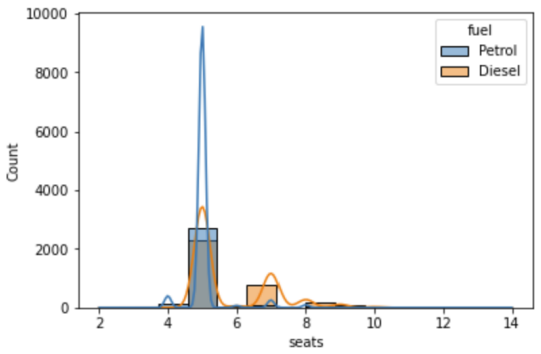
```
<AxesSubplot:xlabel='km_driven', ylabel='Count'>
```



km_driven

```
#Detailing the numerical variables
sns.histplot(data=df, x="seats", kde=True, hue="fuel")
```

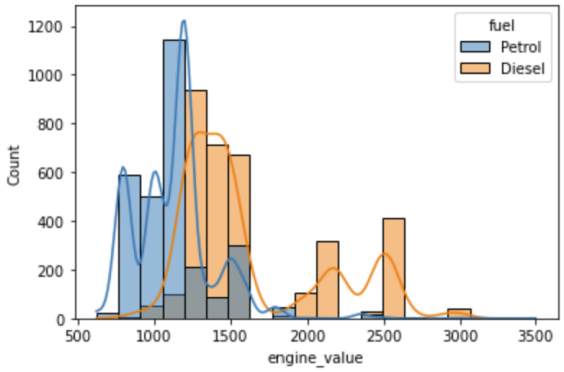
```
<AxesSubplot:xlabel='seats', ylabel='Count'>
```



seats

```
#Detailing the numerical variables
sns.histplot(data=df, x="engine_value",
             kde=True, bins=20, hue="fuel")
```

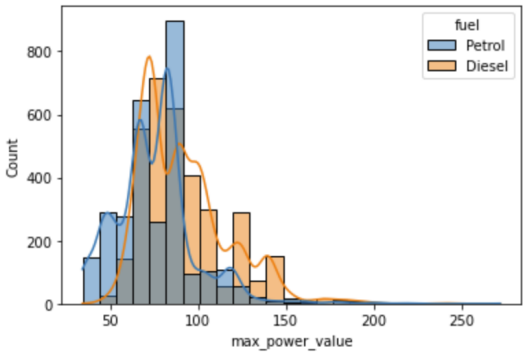
```
<AxesSubplot:xlabel='engine_value', ylabel='Count'>
```



engine_value

```
#Detailing the numerical variables
sns.histplot(data=df, x="max_power_value",
             kde=True, bins=25, hue="fuel")
```

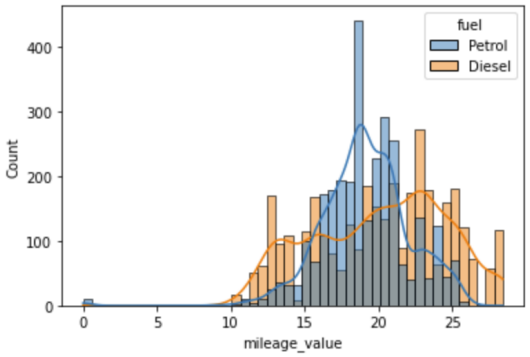
```
<AxesSubplot:xlabel='max_power_value', ylabel='Count'>
```



max_power_value

```
#Detailing the numerical variables
sns.histplot(data=df, x="mileage_value",
             kde=True, hue="fuel")
```

```
<AxesSubplot:xlabel='mileage_value', ylabel='Count'>
```



miliage_value

ENCODING

According to Team (2020), one good practice for data science projects is to encode categorical variables in order to transform them into a numerical variable. For this project, I used two techniques.

The first one is called get_dummies. I decided to do this one in the columns that have more the two values, for example, the columns “Owner” which shows if the car had one, two, three or four or more owners before being put on sale.

The second technique is called Label Encoder. I used this one in the column “fuel”, since it has only two values (Diesel or Petrol).

```
# Encoding the categorical variables
df_encoded = pd.get_dummies(df, columns = ['seller_type', 'transmission', 'owner'])
df_encoded.head(3)
```

_type_Dealer	seller_type_Individual	seller_type_Trustmark Dealer	transmission_Automatic	transmission_Manual	owner_First Owner	owner_Fourth & Above Owner	owner_Second Owner	owner_Third Owner
0	1	0	0	1	1	0	0	0
0	1	0	0	1	0	0	1	0
0	1	0	0	1	0	0	1	0

get_dummies()

```
# creating the instance
labelencoder = LabelEncoder()
# Assingning and storing the values
df_encoded["fuel"] = labelencoder.fit_transform(df_encoded["fuel"]) # Disel = 0 // Petrol = 1
# displaying
df_encoded.head(3)
```

	name	year	selling_price	fuel	km_driven	seats	engine_value	max_power_value	mileage_value	seller_type_Dealer
0	Maruti Alto 800 VXI	2020	250999	1	30000	5	796	47.30	22.05	0
1	Maruti Celerio X ZXI	2020	250000	1	120000	5	998	67.00	21.63	0
2	Maruti Wagon R LXI	2020	265000	1	70000	5	998	67.05	21.79	0

Label encoder

DROPPING AND RENAMING COLUMNS

After doing the encoding, I decided to drop some columns in order to make the dataset simpler.

First, instead of having 4 columns to show previous owners, I decided to keep just one, that will show if it is the first “owner” or not. I used the same idea to show “manual” or not and “individual seller” or not.

Finally, I renamed the columns that had spaces in their names.

```
# Dropping unnecessary columns
df_encoded.drop(['seller_type_Dealer',
                 "seller_type_Trustmark Dealer",
                 "transmission_Automatic",
                 "owner_Second Owner",
                 "owner_Third Owner",
                 "owner_Fourth & Above Owner"],
                axis=1, inplace=True)
df_encoded.head(3)
```

	name	year	selling_price	fuel	km_driven	seats	engine_value	max_power_value	mileage_value	seller_type_Individual	transmission_Manual	owner_First Owner
0	Maruti Alto 800 VXI	2020	250999	1	30000	5	796	47.30	22.05	1	1	1
1	Maruti Celerio X ZXI	2020	250000	1	120000	5	998	67.00	21.63	1	1	0
2	Maruti Wagon R LXI	2020	265000	1	70000	5	998	67.05	21.79	1	1	0

Dropping columns after encoding

```
# Renaming columns
df_encoded.rename(columns = {'seller_type_Individual':'individual_seller',
                             'transmission_Manual':'manual',
                             'owner_First Owner':'first_owner'},
                  inplace = True)
df_encoded.head(3)
```

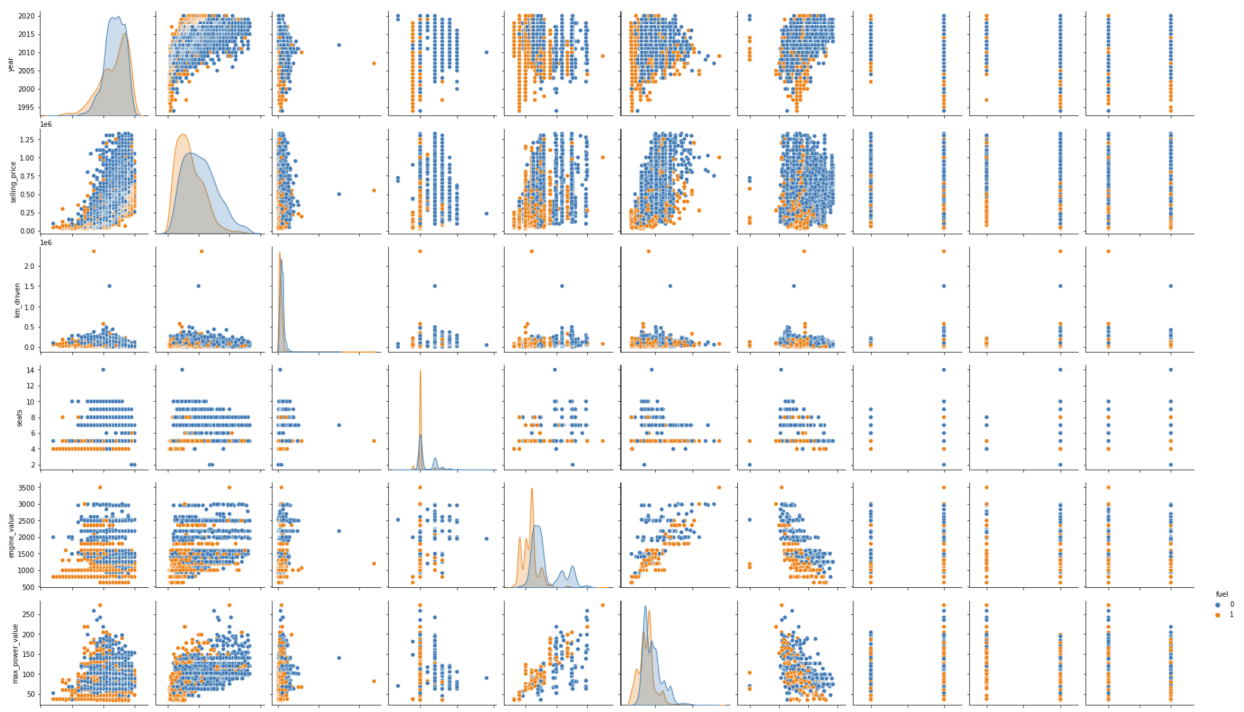
	name	year	selling_price	fuel	km_driven	seats	engine_value	max_power_value	mileage_value	individual_seller	manual	first_owner
0	Maruti Alto 800 VXI	2020	250999	1	30000	5	796	47.30	22.05	1	1	1
1	Maruti Celerio X ZXI	2020	250000	1	120000	5	998	67.00	21.63	1	1	0
2	Maruti Wagon R LXI	2020	265000	1	70000	5	998	67.05	21.79	1	1	0

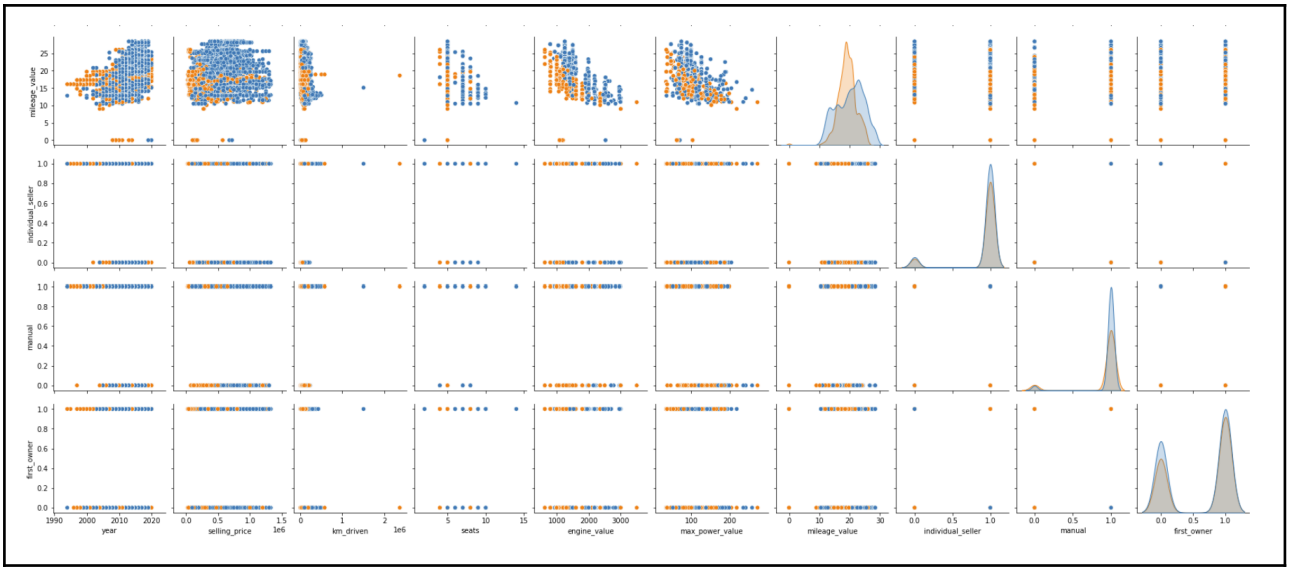
Renaming columns

CHECKING VARIABLES' RELATION

```
# Checking variables' relation

sns.pairplot(df_encoded, hue="fuel")
```

```
<seaborn.axisgrid.PairGrid at 0x1b4fd384a60>
```

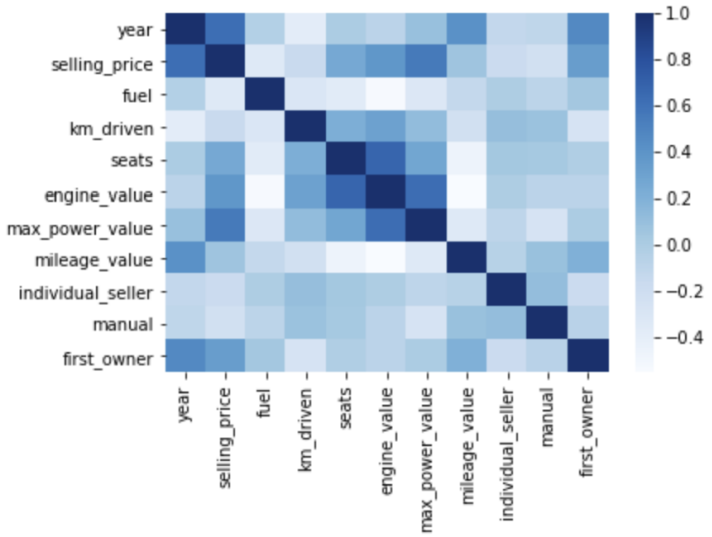




Showing relation

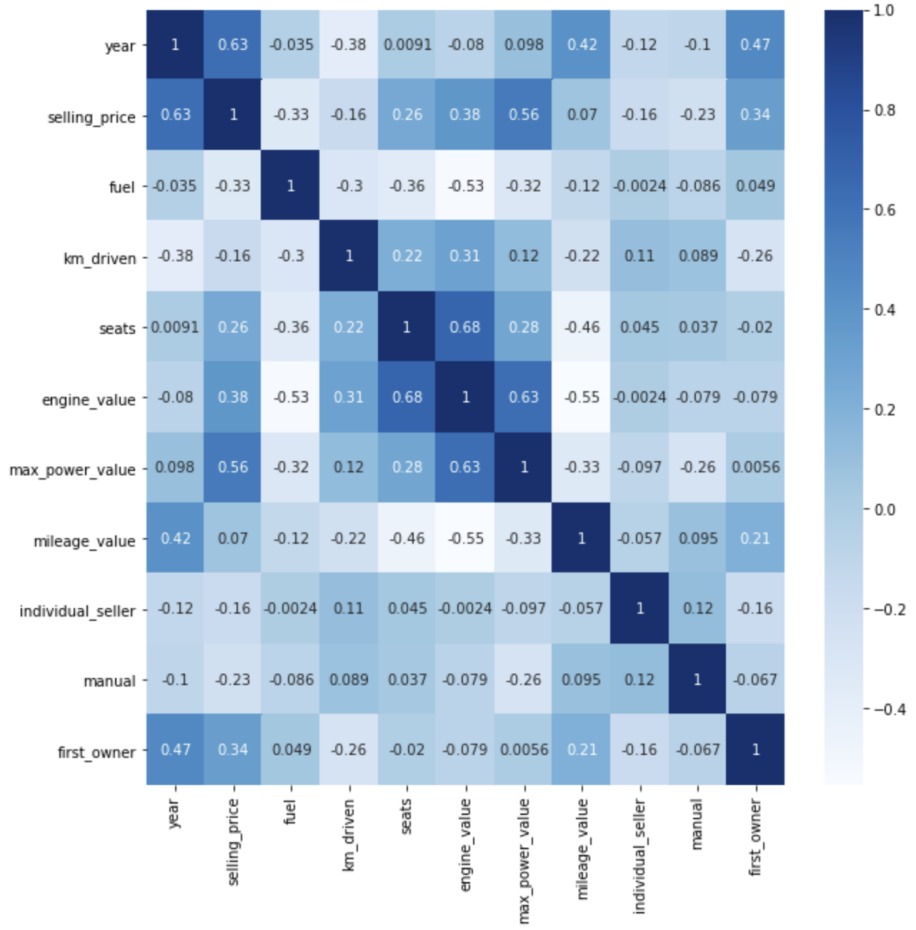
```
# Checking variables' relation
sns.heatmap(df_encoded.corr(), cmap="Blues")
```

```
<AxesSubplot:>
```



Heatmap 1 - just colors

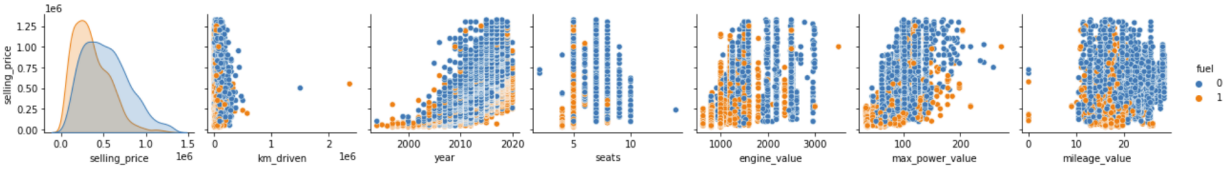
```
# Checking variables' relation
f, ax = plt.subplots(1, 1, figsize=(10, 10))
ax = sns.heatmap(df_encoded.corr(), annot=True, cmap='Blues')
```



Heatmap 2

```
# Checking variables' relation

sns.pairplot(df_encoded,
             x_vars=["selling_price", "km_driven", "year", "seats", "engine_value", "max_power_value", "mileage_value"],
             y_vars=["selling_price"],
             hue="fuel")
```

```
<seaborn.axisgrid.PairGrid at 0x1b4862e1a60>
```



Pairplot - selling_price

MACHINE LEARNING

DEFINITION

To start off with, it is important to understand the concept of Machine Learning. According to Brown (2021), Machine Learning can be explained as the capability of a machine to imitate human behavior in order to solve problems by performing complex tasks and calculations.

In other words, a Machine Learning model may be able to make predictions after using a variety of algorithms and statistical models. It means that computer systems may be able to learn, grow and/or adapt new data in order to imitate how humans acquire new knowledge, gradually improving its accuracy.

For this project, different models of machine learning will be used. The main idea of running the models in this CA is to better understand the differences and applications of regression and classification models.

REGRESSION VS CLASSIFICATION

It may be confusing to understand the differences between regression and classification in Machine Learning. Firstly, it is important to clarify that both regression and classification are types of supervised machine learning algorithms. It means that both models are somehow trained using part of the existing data in order to make predictions.

Knowing the main differences of these two types of algorithms may be important so as to have better accuracy at the end.

In this project, I am going to use 5 regression models and 4 classification models in different features of the dataset.

JUSTIFYING WHY I CHOSE THE MODELS

Originally, I was only requested to perform 5 models for this CA (3 Regression and 2 Classification). However, during the research for this project I ended up finding different models that made me curious, and I ended up deciding to compare their performance.

In the end, I used 5 regression and 4 classification models, in which 4 of them (2 regression and 2 classification) have not covered up to this point at CCT. The regression ones are Lasso and Ridge and the classification ones are KNeighbors and logistic regression.

In addition, for some of the 9 models used in this project, I decided to experiment with different parameters in order to better understand the way they work.

I also included before each model a short description of its functionality.

REGRESSION MODELS

CONCEPT

In general, regression models tend to provide a function to show and/or determine any relationship between a dependent and an independent variable. In other words, after a regression analysis it may be possible to make predictions on the dependent variable based on the independent variable.

After analyzing the dataset used in this project, I decided that the continuous feature “selling_price” would be the best one to make predictions. I took this decision because not only does it make more sense to try to predict the selling price, but also because this feature has some level of interaction with the other columns.

In order to prepare the dataset to perform the regression models, I decided to create a copy of the encoded dataset from previous steps. The main reason for this is because I wanted to keep a saved copy of the dataset, since in the future, I may do different things with it, for example, perform classification models.

As it is possible to see below, I also renamed the column “selling_price”. Its new name is “price”. Besides that, for better visualization, I moved the column for the last position in the dataset.

```
# Making a copy
df_encoded_r = df_encoded.copy()
df_encoded_r.head(3)
```

	name	year	selling_price	fuel	km_driven	seats	engine_value	max_power_value	mileage_value	individual_seller	manual	first_owner
0	Maruti Alto 800 VXI	2020	250999	1	30000	5	796	47.30	22.05	1	1	1
1	Maruti Celerio X ZXI	2020	250000	1	120000	5	998	67.00	21.63	1	1	0
2	Maruti Wagon R LXI	2020	265000	1	70000	5	998	67.05	21.79	1	1	0

```
# Renaming and moving the y
df_encoded_r[['price']] = df_encoded_r['selling_price']
df_encoded_r.drop(['selling_price'], axis=1, inplace=True)
df_encoded_r.head(3)
```

	name	year	fuel	km_driven	seats	engine_value	max_power_value	mileage_value	individual_seller	manual	first_owner	price
0	Maruti Alto 800 VXI	2020	1	30000	5	796	47.30	22.05	1	1	1	250999
1	Maruti Celerio X ZXI	2020	1	120000	5	998	67.00	21.63	1	1	0	250000
2	Maruti Wagon R LXI	2020	1	70000	5	998	67.05	21.79	1	1	0	265000

Preparing for regression

SPLITTING AND SCALING THE DATASET

In order to test the performance of the regression models, I decided to split the dataset into a train and test set. My test size is 20% of the dataset and I am using random_state=0.

It is important to notice that I splitted that dataset before running all the regression models. The main idea for that is because I wanted all the models to use the same test size and random state, since my intention is to compare their accuracy at the end.

```
# Splititng the dataset
from sklearn.model_selection import train_test_split
X = df_encoded_r.iloc[:, 1:-1].values
y = df_encoded_r.iloc[:, -1].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=0)
```

Splitting the dataset

Sometimes, it is necessary to change the data in a way to make it easier for machine learning models to run without any problems. In other words, scaling a dataset helps its features to be kept in a certain finite range, which may increase the performance of machine learning models. According to Roy(2020), scaling is crucial because the analysis seeks out the features with the highest variance, and high variance features skew the analysis towards high magnitude features. For this project I used Standard Scaling.

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

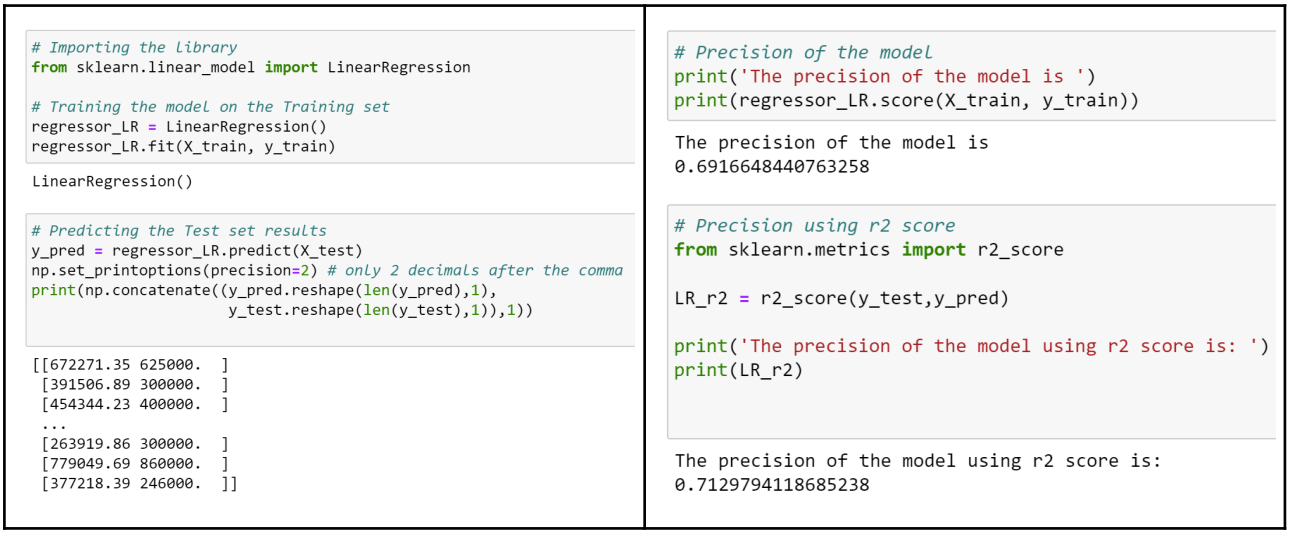
Scaling the dataset

LINEAR REGRESSION

Linear regression models represent the proportional relationship between the dependent and the independent variable. Maily, it shows that both variables tend to increase or decrease together.

Graphically speaking, linear regression models have a line trying to demonstrate the pattern among the points that represent the variables. It is important to notice that those points are regularly not on the line and that they do not need to be. The main idea is that this straight line gives an adequate representation of the variable's distribution.

The performance of this model in this dataset was relatively OK. Using score(), its accuracy was 69% and when using r2_score() it was 71%. During the data preparation process, the plots showed that the column “selling_price” may not have a linear relation with the other columns. These scores may be a reflection of that.



```
# Importing the library
from sklearn.linear_model import LinearRegression

# Training the model on the Training set
regressor_LR = LinearRegression()
regressor_LR.fit(X_train, y_train)
```

```
LinearRegression()
```

```
# Predicting the Test set results
y_pred = regressor_LR.predict(X_test)
np.set_printoptions(precision=2) # only 2 decimals after the comma
print(np.concatenate((y_pred.reshape(len(y_pred),1),
                      y_test.reshape(len(y_test),1)),1))
```

```
[[672271.35 625000.  ]
 [391506.89 300000.  ]
 [454344.23 400000.  ]
 ...
 [263919.86 300000.  ]
 [779049.69 860000.  ]
 [377218.39 246000.  ]]
```

```
# Precision of the model
print('The precision of the model is ')
print(regressor_LR.score(X_train, y_train))
```

```
The precision of the model is
0.6916648440763258
```

```
# Precision using r2 score
from sklearn.metrics import r2_score

LR_r2 = r2_score(y_test,y_pred)

print('The precision of the model using r2 score is: ')
print(LR_r2)
```

```
The precision of the model using r2 score is:
0.7129794118685238
```

Linear regression

RANDOM FOREST REGRESSION

Random Forest Regression is one example of a supervised learning algorithm based on ensemble learning method for regression. In other words, it combines different predictions from different machine learning models in order to make better predictions than the single models alone. The output of this model generally is the average prediction of the individual trees.

The performance of this model in this dataset was the best found among all the regressors. When using score(), its accuracy was 93% and when using r2_score() it was 86%. I decided to use max_depth=10 since I wanted to reduce the complexity of the model, as well as, reduce the risk of overfitting.

```python
# Importing the library
from sklearn.ensemble import RandomForestRegressor

# Training the model on the Training set
regressor_RFR = RandomForestRegressor(max_depth=10, random_state=0)
regressor_RFR.fit(X_train, y_train)
```

```
RandomForestRegressor(max_depth=10, random_state=0)
```

```python
# Predicting the Test set results
y_pred = regressor_RFR.predict(X_test)
np.set_printoptions(precision=2) # only 2 decimals after the comma
print(np.concatenate((y_pred.reshape(len(y_pred),1),
                      y_test.reshape(len(y_test),1)),1))
```

```
[[689067.26 625000.  ]
 [285927.1  300000.  ]
 [359439.71 400000.  ]
 ...
 [242283.87 300000.  ]
 [946105.34 860000.  ]
 [221462.08 246000.  ]]
```

```python
# Precision of the model
print('The precision of the model is ')
print(regressor_RFR.score(X_train, y_train))
```

```
The precision of the model is
0.9391914909818211
```

```python
# Precision using r2 score
# from sklearn.metrics import r2_score

RFR_r2 = r2_score(y_test,y_pred)

print('The precision of the model using r2 score is: ')
print(RFR_r2)
```

```
The precision of the model using r2 score is:
0.8689751306795716
```

Random forest regression

DECISION TREE REGRESSION

Decision tree regression models are models that make predictions using a set of binary rules in order to determine a specific value. Being more specific, these models divide the dataset in small subsets while an associated decision tree is developed, resulting in decision nodes and leaf nodes.

As expected, the performance of this model in this dataset was rather similar to the Random Forest Regression. When using score(), its accuracy was 88% and when using r2_score() it was 83%. Differently from the Random Forest Regression case, I decided to use max_depth=8 in order to try different parameters. The main purpose was the same (reduce the complexity of the model, and reduce the risk of overfitting).

```python
# Importing the library
from sklearn.tree import DecisionTreeRegressor

# Training the model on the Training set
regressor_DTR = DecisionTreeRegressor(max_depth=8)
regressor_DTR.fit(X_train, y_train)
```

```
DecisionTreeRegressor(max_depth=8)
```

```python
# Predicting the Test set results
y_pred = regressor_DTR.predict(X_test)
np.set_printoptions(precision=2) # only 2 decimals after the comma
print(np.concatenate((y_pred.reshape(len(y_pred),1),
                      y_test.reshape(len(y_test),1)),1))
```

```
[[659132.63 625000.  ]
 [272666.63 300000.  ]
 [318624.09 400000.  ]
 ...
 [239722.17 300000.  ]
 [895677.39 860000.  ]
 [244378.01 246000.  ]]
```

```python
# Precision of the model
print('The precision of the model is ')
print(regressor_DTR.score(X_train, y_train))
```

```
The precision of the model is
0.8870421921641337
```

```python
# Precision using r2 score
# from sklearn.metrics import r2_score

DTR_r2 = r2_score(y_test,y_pred)

print('The precision of the model using r2 score is: ')
print(DTR_r2)
```

```
The precision of the model using r2 score is:
0.8301380226366681
```

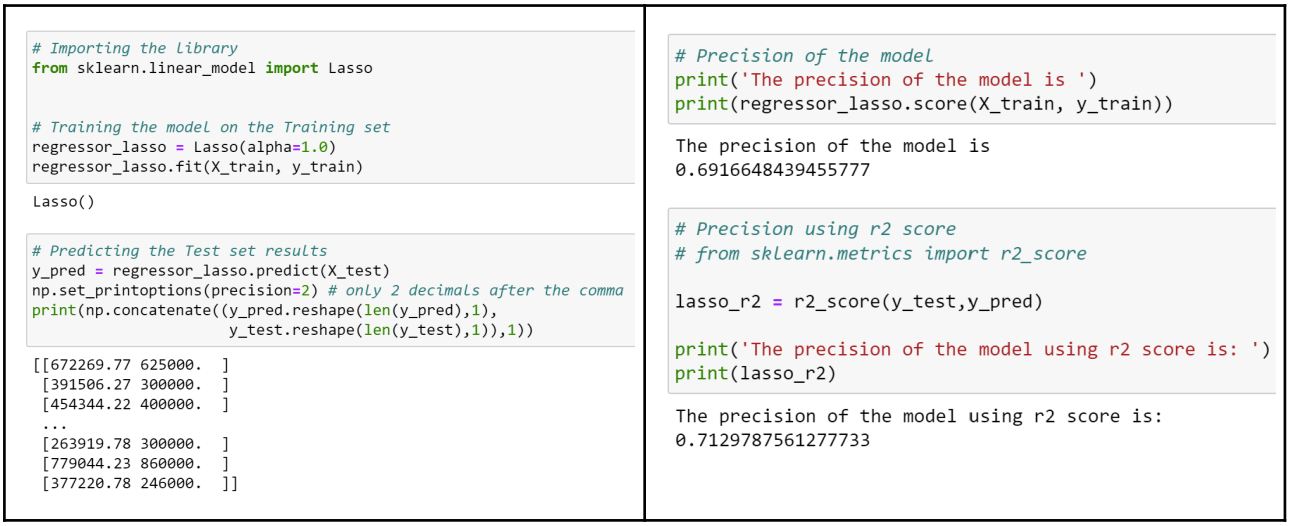
Decision tree regression

LASSO REGRESSION

Lasso regression can be simply defined as a type of regularized linear regression that includes an L1 penalty. The acronym “LASSO” stands for Least Absolute Shrinkage and Selection Operator. In other words, it is a type of linear regression that uses shrinkage. It has the effect of shrinking the coefficients of variables that do not contribute much to the prediction task.

Lasso regression is mainly recommended for datasets with some level of multicollinearity or for when it is necessary to automate some specific parts of the model.

As expected, the performance of this model in this dataset was pretty similar to the Linear Regression. When using score(), its accuracy was 69% and when using r2_score() it was 71%.



```
# Importing the library
from sklearn.linear_model import Lasso


# Training the model on the Training set
regressor_lasso = Lasso(alpha=1.0)
regressor_lasso.fit(X_train, y_train)
```

```
Lasso()
```

```
# Predicting the Test set results
y_pred = regressor_lasso.predict(X_test)
np.set_printoptions(precision=2) # only 2 decimals after the comma
print(np.concatenate((y_pred.reshape(len(y_pred),1),
                      y_test.reshape(len(y_test),1)),1))
```

```
[[672269.77 625000.  ]
 [391506.27 300000.  ]
 [454344.22 400000.  ]
 ...
 [263919.78 300000.  ]
 [779044.23 860000.  ]
 [377220.78 246000.  ]]
```

```
# Precision of the model
print('The precision of the model is ')
print(regressor_lasso.score(X_train, y_train))
```

```
The precision of the model is
0.6916648439455777
```

```
# Precision using r2 score
# from sklearn.metrics import r2_score

lasso_r2 = r2_score(y_test,y_pred)

print('The precision of the model using r2 score is: ')
print(lasso_r2)
```

```
The precision of the model using r2 score is:
0.7129787561277733
```

Lasso regression

RIDGE REGRESSION

Ridge regression is a technique used to make analysis and predictions in datasets that suffer from multicollinearity. The best case scenario to use this model would be in a dataset that contains a greater number of predictor variables than number of rows.

In other words, this model may not be the best one for the dataset used in this project. However, in order to have some experience and have a way to compare different models, I decided to perform this model. My expectations were that this model performance may be rather similar to linear models.

As expected, the performance of this model in this dataset was pretty similar to the Linear Regression and Lasso Regression. When using score(), its accuracy was 69% and when using r2_score() it was 71%.

```
# Importing the library
from sklearn.linear_model import Ridge

regressor_ridge = Ridge()
regressor_ridge.fit(X_train, y_train)
```

```
Ridge()
```

```
# Predicting the Test set results
y_pred = regressor_ridge.predict(X_test)
np.set_printoptions(precision=2) # only 2 decimals after the comma
print(np.concatenate((y_pred.reshape(len(y_pred),1),
                      y_test.reshape(len(y_test),1)),1))
```

```
[[672229.48 625000.  ]
 [391519.54 300000.  ]
 [454351.03 400000.  ]
 ...
 [263963.2  300000.  ]
 [779021.83 860000.  ]
 [377217.42 246000.  ]]
```

```
# Precision of the model
print('The precision of the model is ')
print(regressor_ridge.score(X_train, y_train))
```

```
The precision of the model is
0.6916648251851827
```

```
# Precision using r2 score
# from sklearn.metrics import r2_score

ridge_r2 = r2_score(y_test,y_pred)

print('The precision of the model using r2 score is: ')
print(ridge_r2)
```

```
The precision of the model using r2 score is:
0.7129714477594963
```

Ridge regression

WHICH REGRESSION MODEL PERFORMED BETTER

In general, tree based models performed better in this dataset, even when comparing score() and r2_score(). Once again, it may be a reflection of what was seen in the plots.

To conclude, for now, Random Forest Regression had the best performance with over 93% (86%) of accuracy.

```
print("Linear Regression:          ", regressor_LR.score(X_train, y_train))
print("Random Forest Regression:   ", regressor_RFR.score(X_train, y_train))
print("Decision Tree Regressor:    ", regressor_DTR.score(X_train, y_train))
print("Lasso Regression:           ", regressor_lasso.score(X_train, y_train))
print("Ridge Regression:           ", regressor_ridge.score(X_train, y_train))
```

```
Linear Regression:          0.6916648440763258
Random Forest Regression:   0.9391914909818211
Decision Tree Regressor:    0.8870421921641337
Lasso Regression:           0.6916648439455777
Ridge Regression:           0.6916648251851827
```

score() - regression models

```
print("(R2 Score) Linear Regression:          ", LR_r2)
print("(R2 Score) Random Forest Regression:   ", RFR_r2)
print("(R2 Score) Decision Tree Regressor:    ", DTR_r2)
print("(R2 Score) Lasso Regression:           ", lasso_r2)
print("(R2 Score) Ridge Regression:           ", ridge_r2)
```

```
(R2 Score) Linear Regression:          0.7129794118685238
(R2 Score) Random Forest Regression:   0.8689751306795716
(R2 Score) Decision Tree Regressor:    0.8301380226366681
(R2 Score) Lasso Regression:           0.7129787561277733
(R2 Score) Ridge Regression:           0.7129714477594963
```

R^2 score - regression models

MAKING PREDICTIONS USING REGRESSION MODELS

Once the best model for this project is known, it is possible now to make some predictions.

First, I compared the real price with the price that the Random Forest model predicted. The real price is 250999 and the price the model predicted is 291563. As expected, the result was rather similar since the accuracy of the model is 93%.

Finally, I decided to experiment with the variables: “km_driven”, “year” and “engine value”, expecting the price to decrease and increase according to common sense. And that was exactly what happened.

```
df_encoded_r.head(3)
```

	name	year	fuel	km_driven	seats	engine_value	max_power_value	mileage_value	individual_seller	manual	first_owner	price
0	Maruti Alto 800 VXI	2020	1	30000	5	796	47.30	22.05	1	1	1	250999
1	Maruti Celerio X ZXI	2020	1	120000	5	998	67.00	21.63	1	1	0	250000
2	Maruti Wagon R LXI	2020	1	70000	5	998	67.05	21.79	1	1	0	265000

```
# Predicting using real data

print(regressor_RFR.predict(sc.transform([[2020, 1, 30000, 5, 796, 47.30, 22.05, 1, 1, 1]])))
```

```
[291563.18]
```

```
# Making predictions
# Changing km_driven

print(regressor_RFR.predict(sc.transform([[2020, 1, 60000, 5, 796, 47.30, 22.05, 1, 1, 1]])))
```

```
[291117.77]
```

```
# Making predictions
# Changing year

print(regressor_RFR.predict(sc.transform([[2015, 1, 30000, 5, 796, 47.30, 22.05, 1, 1, 1]])))
```

```
[262396.74]
```

```
# Making predictions
# Changing engine_value

print(regressor_RFR.predict(sc.transform([[2020, 1, 30000, 5, 900, 47.30, 22.05, 1, 1, 1]])))
```

```
[338001.26]
```

Making predictions - Regression

CLASSIFICATION MODELS

CONCEPT

Classification models, in general, are prediction models for discrete variables. They mainly create mapping functions in order to predict labels and/or categories. In other words, according to Brownlee (2020) a classification model is able to predict a class label after being given some input data..

After analyzing the dataset used in this project, I decided that the categorical feature “fuel” would be the best one to make predictions.

I took this decision because among the categorical variables in the dataset, this feature is the one more balanced, which would proportionate a better evaluation for the classification models that will be performed.

Similarly as done with the regression models, I decided to create a copy of the encoded dataset from previous steps.

I also renamed the column “fuel”. Its new name is “fuel_e”. Besides that, for better visualization, I moved the column for the last position in the dataset.

```
# Making a copy
df_encoded_c = df_encoded.copy()
df_encoded_c.head(3)
```

	name	year	selling_price	fuel	km_driven	seats	engine_value	max_power_value	mileage_value	individual_seller	manual	first_owner
0	Maruti Alto 800 VXI	2020	250999	1	30000	5	796	47.30	22.05	1	1	1
1	Maruti Celerio X ZXI	2020	250000	1	120000	5	998	67.00	21.63	1	1	0
2	Maruti Wagon R LXI	2020	265000	1	70000	5	998	67.05	21.79	1	1	0

```
# Renaming and moving fuel
df_encoded_c[['fuel_e']] = df_encoded_c['fuel']
df_encoded_c.drop(['fuel'], axis=1, inplace=True)
df_encoded_c.head(3)
```

	name	year	selling_price	km_driven	seats	engine_value	max_power_value	mileage_value	individual_seller	manual	first_owner	fuel_e
0	Maruti Alto 800 VXI	2020	250999	30000	5	796	47.30	22.05	1	1	1	1
1	Maruti Celerio X ZXI	2020	250000	120000	5	998	67.00	21.63	1	1	0	1
2	Maruti Wagon R LXI	2020	265000	70000	5	998	67.05	21.79	1	1	0	1

Preparing for classification

SPLITTING THE DATASET

It is believed that classification models tend to perform better when the test size is 25% of the dataset. Moreover, I also splitted and used random_state=0 before running the models in order to compare the models accuracy at the end.

```
# Spliting the dataset
# from sklearn.model_selection import train_test_split
X1 = df_encoded_c.iloc[:, 1:-1].values
y1 = df_encoded_c.iloc[:, -1].values
X_train1, X_test1, y_train1, y_test1 = train_test_split(X1, y1, test_size=0.25, random_state=0)
```

Splitting the dataset

RANDOM FOREST CLASSIFIER

According to Yiu (2019), Random Forest Classifier is an algorithm that contains many decision trees that are more accurate than any individual tree.

Similarly to the Random Forest Regression, the Random Forest Classifier is an ensemble learning method; however, it is used for classification. Maily, it is possible to affirm that the output of this model is the class chosen by most of the trees.

The performance of this model in this dataset was the best found among all the classifiers, with 97%. I decided to use max_depth=5 and criterion = “entropy” since I wanted to reduce the complexity of the model, as well as, reduce the risk of overfitting.

```
# Training the Random Forest Classifier model on the Training set
from sklearn.ensemble import RandomForestClassifier
classifier_RFC = RandomForestClassifier(criterion = 'entropy',
                                        max_depth=5, random_state = 0)
classifier_RFC.fit(X_train1, y_train1)
```

```
RandomForestClassifier(criterion='entropy', max_depth=5, random_state=0)
```

```
# Predicting the Test set results
y_pred = classifier_RFC.predict(X_test1)
print(np.concatenate((y_pred.reshape(len(y_pred),1),
                      y_test1.reshape(len(y_test1),1)),1))
```

```
[[0 0]
 [1 1]
 [0 0]
 ...
 [1 1]
 [1 1]
 [0 0]]
```

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test1, y_pred)
score_RFC = accuracy_score(y_test1, y_pred)

print(cm)
```

```
[[853  14]
 [ 31 691]]
```

```
# Precision of the model
print('The precision of the model is ')
score_RFC
```

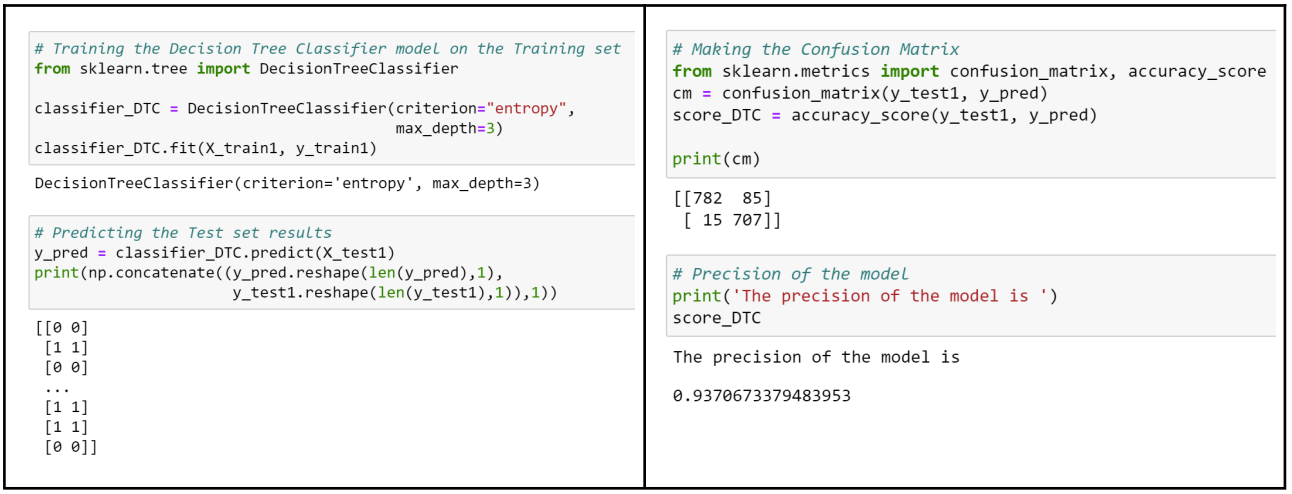
```
The precision of the model is

0.9716803020767778
```

Random forest classifier

DECISION TREE CLASSIFIER

One of the main differences between the decision tree classifier and the regressor is that in the regressor the leaves show the labels and/or classes while the classifier shows a group of features that lead to the labels.

This model had 93% of accuracy. As expected, it was as good as the Random Forest Classifier.



```
# Training the Decision Tree Classifier model on the Training set
from sklearn.tree import DecisionTreeClassifier

classifier_DTC = DecisionTreeClassifier(criterion="entropy",
                                        max_depth=3)
classifier_DTC.fit(X_train1, y_train1)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=3)
```

```
# Predicting the Test set results
y_pred = classifier_DTC.predict(X_test1)
print(np.concatenate((y_pred.reshape(len(y_pred),1),
                      y_test1.reshape(len(y_test1),1)),1))
```

```
[[0 0]
 [1 1]
 [0 0]
 ...
 [1 1]
 [1 1]
 [0 0]]
```

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test1, y_pred)
score_DTC = accuracy_score(y_test1, y_pred)

print(cm)
```

```
[[782  85]
 [ 15 707]]
```

```
# Precision of the model
print('The precision of the model is ')
score_DTC
```

```
The precision of the model is

0.9370673379483953
```

Decision tree classifier

K NEIGHBORS CLASSIFIER

Basically, K Neighbors Classifier models store and classify the data and new data based on the similarity that is shown. In other words, it is possible to classify points in the dataset based on how the neighbors are classified.

This model had 79% of accuracy. Different values for n_neighbors were used, and 5 was the value when the accuracy stopped having a significant change.

```
# Training the KNN model on the Training set
from sklearn.neighbors import KNeighborsClassifier

classifier_KNC = KNeighborsClassifier(n_neighbors = 5,
                                      metric = 'minkowski', p = 2)
classifier_KNC.fit(X_train1, y_train1)
```

```
KNeighborsClassifier()
```

```
# Predicting the Test set results
y_pred = classifier_KNC.predict(X_test1)
print(np.concatenate((y_pred.reshape(len(y_pred),1),
                      y_test1.reshape(len(y_test1),1)),1))
```

```
[[1 0]
 [1 1]
 [0 0]
 ...
 [0 1]
 [1 1]
 [1 0]]
```

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test1, y_pred)
score_KNC = accuracy_score(y_test1, y_pred)

print(cm)
```

```
[[729 138]
 [189 533]]
```

```
# Precision of the model
print('The precision of the model is ')
score_KNC
```

```
The precision of the model is

0.7942101950912523
```

K Neighbors Classifier

LOGISTIC REGRESSION

This is another algorithm used for classification. According to Garg (2018), Logistic Regression models use a logistic function in order to describe and/or predict possible results or outcomes.

This model had 81% of accuracy.

```
# Training the Logistic Regression model on the Training set
from sklearn.linear_model import LogisticRegression

classifier_LogR = LogisticRegression(solver='liblinear',
                                     multi_class='ovr')
classifier_LogR.fit(X_train1, y_train1)
```

```
LogisticRegression(multi_class='ovr', solver='liblinear')
```

```
# Predicting the Test set results
y_pred = classifier_LogR.predict(X_test1)
print(np.concatenate((y_pred.reshape(len(y_pred),1),
                      y_test1.reshape(len(y_test1),1)),1))
```

```
[[0 0]
 [1 1]
 [0 0]
 ...
 [0 1]
 [1 1]
 [1 0]]
```

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test1, y_pred)
score_LogR = accuracy_score(y_test1, y_pred)

print(cm)
```

```
[[731 136]
 [155 567]]
```

```
# Precision of the model
print('The precision of the model is ')
score_LogR
```

```
The precision of the model is

0.81686595342983
```

Logistic Regression

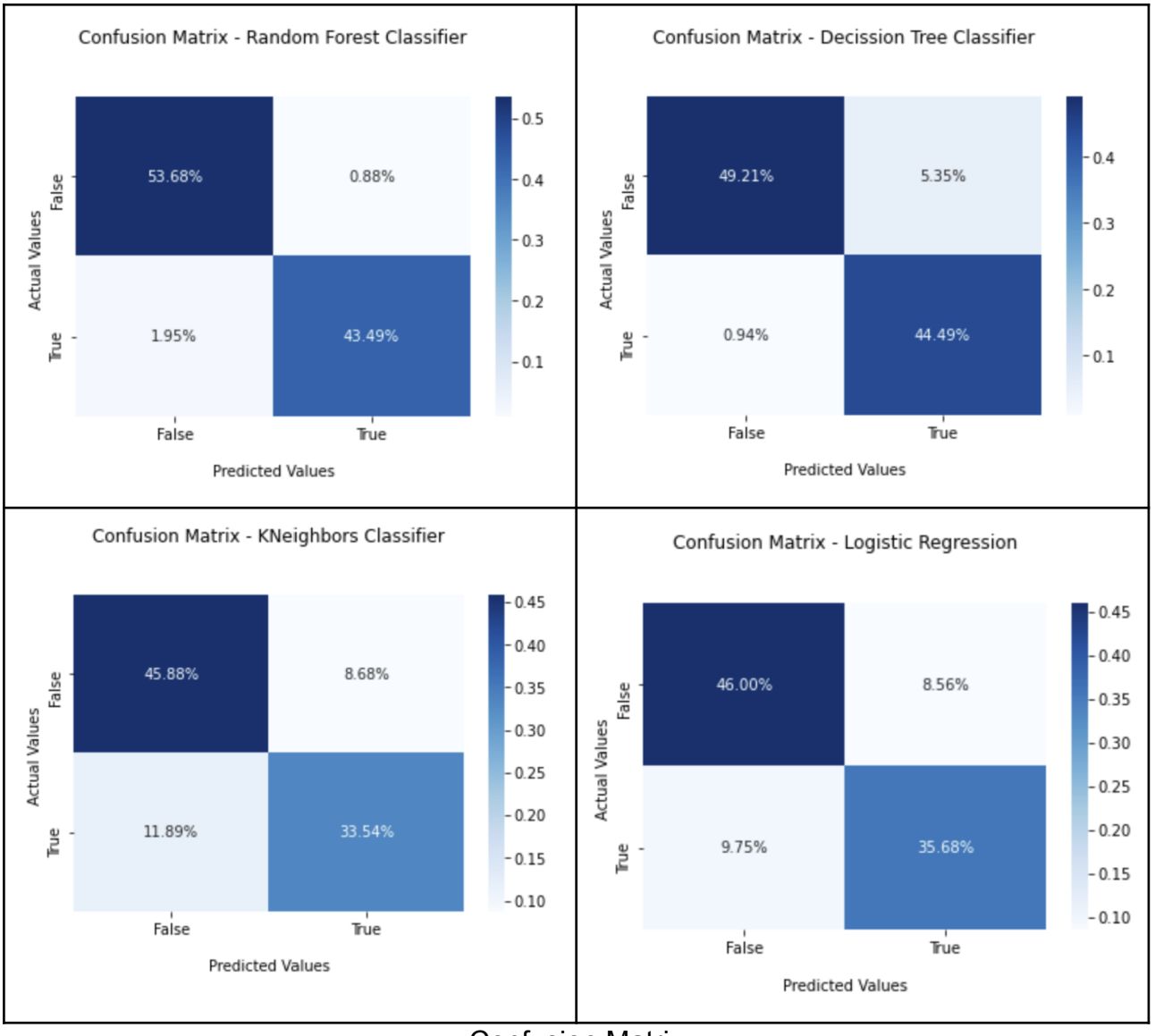
WHICH CLASSIFICATION MODEL PERFORMED BETTER

In general, all the models had good performance in this dataset. However, Random Forest Classifier had the best performance with over 93% of accuracy.

```
print("Random Forest Classifier:   ", score_RFC)
print("Decission Tree Classifier:  ", score_DTC)
print("KNeighbors Classifier:      ", score_KNC)
print("Logistic Regression:        ", score_LogR)
```

```
Random Forest Classifier:    0.9716803020767778
Decission Tree Classifier:   0.9370673379483953
KNeighbors Classifier:       0.7942101950912523
Logistic Regression:         0.81686595342983
```

Classification models accuracy



Confusion Matrix

MAKING PREDICTIONS USING CLASSIFICATION MODELS

It is possible now to make some predictions using classification models. As all models had good performance, I decided to make predictions using the best and the worst classifier.

First, I compared the real fuel type with the fuel type that the models predicted. As expected, the predictions were right.

Finally, I decided to change some inputs in order to verify the algorithm's prediction.

```
df_encoded_c.head(3)
```

	name	year	selling_price	km_driven	seats	engine_value	max_power_value	mileage_value	individual_seller	manual	first_owner	fuel_e
0	Maruti Alto 800 VXI	2020	250999	30000	5	796	47.30	22.05	1	1	1	1
1	Maruti Celerio X ZXI	2020	250000	120000	5	998	67.00	21.63	1	1	0	1
2	Maruti Wagon R LXI	2020	265000	70000	5	998	67.05	21.79	1	1	0	1

```
# Predicting using real data - Random Forest Classifier

print(classifier_RFC.predict(sc.transform([[2020, 250999, 30000, 5, 796, 47.30, 22.05, 1, 1, 1]])))
```

```
[1]
```

```
# Making predictions - Random Forest Classifier
# Changing engine_value, max_power_value, mileage_value

print(classifier_RFC.predict(sc.transform([[2020, 250999, 30000, 5, 1000, 28.50, 10.20, 1, 1, 1]])))
```

```
[1]
```

```
# Predicting using real data - KNeighbors Classifier


print(classifier_KNC.predict(sc.transform([[2020, 480000, 44665, 6, 1198, 77.00, 25.32, 1, 1, 1]])))
```

```
[0]
```

```
# Making predictions - KNeighbors Classifier
# Changing year and selling price

print(classifier_KNC.predict(sc.transform([[2015, 250000, 44665, 6, 1198, 77.00, 25.32, 1, 1, 1]])))
```

```
[1]
```

Making predictions - Classification

SUGGESTIONS OF FUTURE STEPS

Different regression and classification models were used in this project. In general, they had a satisfactory performance regarding accuracy. However, it would be a good idea to return to the data preparation part and review some steps and decisions taken, since it may reflect in the model's performance.

Besides that, by experimenting with different parameters, it may be possible not only to improve this dataset analysis, but also, it may help me to develop my knowledge as a data analyst.

CONCLUSION

During the research period that was needed to conclude this CA, I had the opportunity to better understand some concepts of Exploratory data analysis and Machine Learning. It was a great experience to learn the similarities of different machine learning models.

As I was able to perform all the models in the same dataset, I could practice and experience some real life problems, such as trying to find different models to answer different questions.

Besides that, after having studied about different types of Machine Learning models, I could have a better practice of topics studied in the subject in class. In addition, as I have read about models that were not taught in class, once their time arrives, my understanding may be simpler.

During the prediction part, I could have a better visualization in how independent variables can affect dependent variables. By trying different inputs and parameters, I could have a better understanding of the dataset. Moreover, the experience obtained after dealing with scaling, splitting and modeling were very educational.

Finally, the research helped me to better understand concepts of regression and classification, and these will surely be a great help, not only for my future projects, but also for my professional career.

To conclude, all the acquired knowledge after studying Python to conclude this CA may accompany me during my career as a data analyst.

REFERENCE LIST

Bhutani, K. (2018). *Python | Pandas dataframe.drop_duplicates()*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/python-pandas-dataframe-drop_duplicates/ [Accessed 11 Apr. 2022].

Brown, S. (2021). *Machine learning, explained*. [online] MIT Sloan. Available at: https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained [Accessed 8 Apr. 2022].

Brownlee, J. (2020). *4 Types of Classification Tasks in Machine Learning*. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/types-of-classification-in-machine-learning/ [Accessed 15 Apr. 2022].

Chandradas, A. (2021). *5 Methods to Check for NaN values in in Python*. [online] Medium. Available at: https://towardsdatascience.com/5-methods-to-check-for-nan-values-in-in-python-3f21ddd17eed#:~:text=NaN%20stands%20for%20Not%20A [Accessed 14 Apr. 2022].

Galarnyk, M. (2018). *Understanding Boxplots*. [online] Medium. Available at: https://towardsdatascience.com/understanding-boxplots-5e2df7bcbd51 [Accessed 14 Apr. 2022].

Garg, R. (2018). *7 Types of Classification Algorithms*. [online] Analytics India Magazine. Available at: https://analyticsindiamag.com/7-types-classification-algorithms/ [Accessed 8 Apr. 2022].

Gupta, S. (2021). *Regression vs. Classification in Machine Learning: What's the Difference?* [online] Springboard Blog. Available at: https://www.springboard.com/blog/data-science/regression-vs-classification/ [Accessed 8 Apr. 2022].

Patil, P. (2018). *What is Exploratory Data Analysis?* [online] Towards Data Science. Available at: https://towardsdatascience.com/exploratory-data-analysis-8fc1cb20fd15 [Accessed 15 Apr. 2022].

Roy, B. (2020). *All about Feature Scaling*. [online] Medium. Available at: https://towardsdatascience.com/all-about-feature-scaling-bcc0ad75cb35 [Accessed 14 Apr. 2022].

Team, G.L. (2020a). *Ridge Regression Definition & Examples | What is Ridge Regression?* [online] GreatLearning Blog: Free Resources what Matters to shape your Career! Available at: https://www.mygreatlearning.com/blog/what-is-ridge-regression/ [Accessed 16 Apr. 2022].

Team, G.L. (2020b). *What is Label Encoding in Python | Great Learning*. [online] GreatLearning Blog: Free Resources what Matters to shape your Career! Available at: https://www.mygreatlearning.com/blog/label-encoding-in-python/ [Accessed 15 Apr. 2022].

Vadapalli, P. (2020). *6 Types of Regression Models in Machine Learning You Should Know about*. [online] UpGrad Blog. Available at: https://www.upgrad.com/blog/types-of-regression-models-in-machine-learning/ [Accessed 8 Apr. 2022].

Yiu, T. (2019). *Understanding Random Forest*. [online] Medium. Available at: https://towardsdatascience.com/understanding-random-forest-58381e0602d2 [Accessed 11 Apr. 2022].