University of Regina

A Used Car Price Prediction System (UCPPS)

Final Project Report

Submitted to Dr. Alireza Manashty

Department of Computer Science

CS 890ES - Data Science Fundamentals

Winter 2020

By

Tanu Nanda Prabhu - 200409072 - tnb735@uregina.ca

Regina, Saskatchewan

April, 2020

TABLE OF CONTENTS

Chapter 1

Data Analytic Life Cycle

1.1 Introduction to data analytic lifecycle

Data Science is an emerging interdisciplinary field. Altogether, data science is the science involved in studying the data. Data Analysis which is a part of data science has a life cycle composed of 6 phases [1]. These phases include: **Discovery**, **Data Preparation**, **Model Planning**, **Model Building**, **Communicate Results** and **Operationalize**. The detailed data analytic life cycle can be found below:

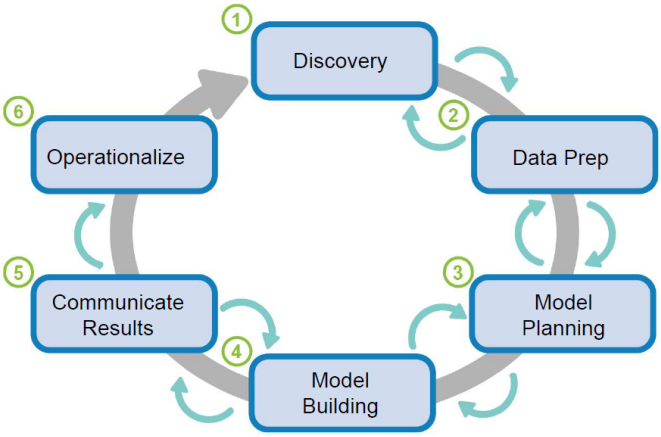

Figure 1.1: Data Analytic life cycle

The above life cycle was completely followed while developing the project named a used car price prediction system.

Chapter 2

Data Discovery

The discovery is the first phase in the data analytic life cycle. In this chapter the introduction followed by problem statement along with the current situation and the desired situation would be discussed.

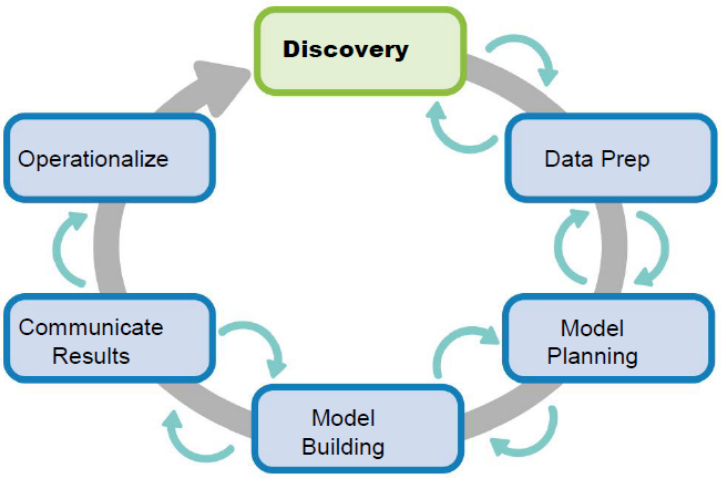

Figure 2.1: Data Discovery

2.1 Introduction

Vehicle value forecast is both a basic and a significant errand, particularly when the vehicle is used or coming legitimately from the production line. It is also a

fascinating and a prominent issue. Often the vehicle value will always depend on its features or specifications. Better features increase the value of the vehicle. Wouldn't it be awesome to build a model that would predict the price of the vehicle given the features? Here, the vehicle would be a **"Used car"**. Because most of the people prefer to buy a used car than a brand-new car because of various reasons, and one of them is poor financial conditions. In this project, a machine learning model is developed with the help of which of you can predict the price of a used car. The value of a used car depends on several factors. Some main factors are the make (model) of the car, the origin, the number of kilometers it has run, power, year of registration, fuel type, and gearbox. Unfortunately, all the above feature information is not always available to the buyer online. The buyer then must buy the car at a certain price and end up thinking if the car is worth the price or not? Similarly, the seller here won't prefer to sell their car for a very cheap price. Through setting unfair costs for the used cars, consumers can be subjugated easily, and much might fall into the trap. There is a need for a used car price prediction system to determine the worthiness of the car using a variety of features.

2.2 Problem Statement

Used car prices are an important reflection of the economy, and they interest both buyers and sellers. The used car market is large and strategically very important for car manufacturers. Since the secondhand car market is closely related to the new car business in various aspects. A prediction model that estimates resale price based on car attributes and features is much more needed today. This prediction system would be helpful for a buyer who doesn't have any idea how much to spend on a used car. Similarly, the seller who does not understand about the price to sell his/her car for. As we all know, many factors contribute to predicting the price of the used

car based on the features of the car. This analysis aims to determine which features of the car in the may have the strongest statistical correlation with the price of the car. Also, determining the relationship between the outcome (price) and the input variables (features). With the help of the model we can come to know the variables on which the price depends, and to what degree those variables justify a car's price.

2.3 Analyzing the situations

2.3.1 Current Situation

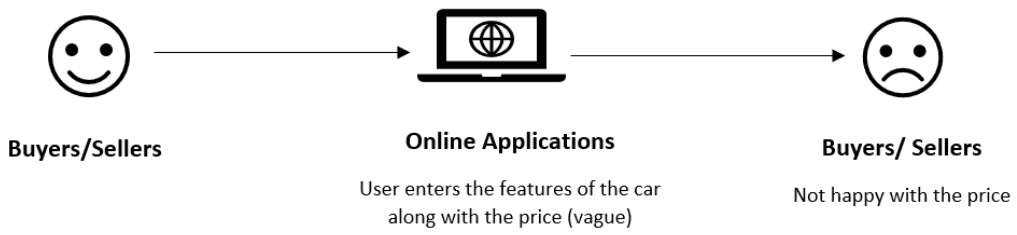

Figure 2.2: Current Situation

Now let us analyze the current situation from the above shown figure. Currently, the users such as Buyers/Sellers can access an online application to buy or sell their car, but the problem is they must manually decide and set the price of the car. Sometimes in this case both the users might face a tremendous loss regarding the price. So, at the end of the day both the users end up getting disappointed because their price estimation was not worthy.

2.3.2 Desired Situation

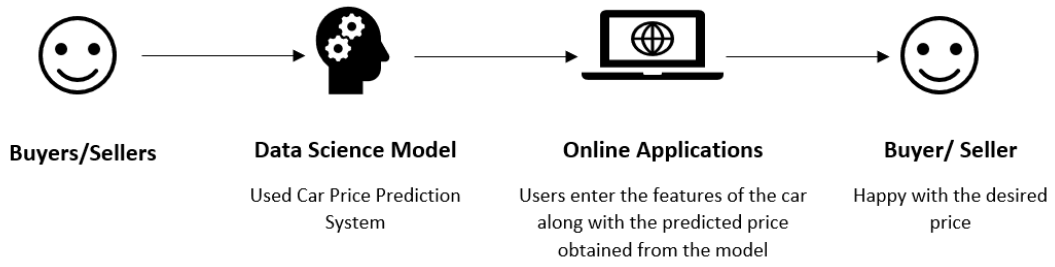

Figure 2.3: Desired Situation

In this desired situation as shown in the above figure, now an extra phase or step is being added which is **"Data Science Model"** which is also known as **"Used Car Price Prediction System"**. Here, rather than entering vague price of the car, the users can get an estimated price predicted by the system and then they can use the generated price for the online application. Now in the data science model, the user must enter all the features of their car and then get the predicted price. The users can now trust the price generated by the system and end up not getting disappointed. With the help of predicted price by the model, the both the users can buy/sell their cars online or offline.

Chapter 3

Data Preparation

In this chapter, all the necessary steps taken to prepare the data such as extraction, transformation, loading the data would be performed. Further the data would be explored and conditioned by visualizing the results.

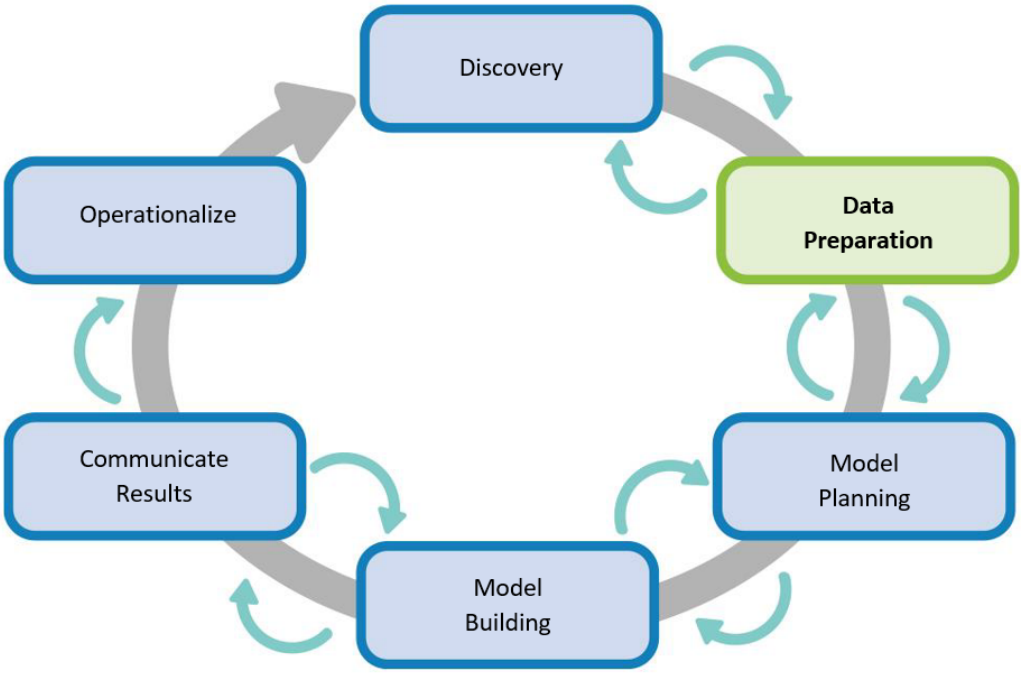

Figure 3.1: Data Preparation

3.1 Data

3.1.1 Source of the data

The dataset titled **Used Cars Database** was chosen from the website data world. This data set was scraped from an online e-commerce website called E-bay (Germany) using scrapy as a tool. The author of the data set is Orges Leka [2]. Scrapy is an open-source collaborative tool that is used to extract the data [3]. The contents of the data are in the German language. So, all the contents of the data must be translated into English. Also, the original raw dataset can be found here.

3.1.2 Storing the data

As this is not a big data set in terms of size, no cloud service would be used as external storage to store the data. Rather, the data was stored inside my drive (Hard disk). Initially, it took a lot of time to upload the data on to the Google Colab notebook, and sometimes it crashed due to some issues. But an alternate option called **Mount** was performed which mounted my google drive to the Google Colab environment and then the dataset could be easily access from my drive.

3.1.3 Accessing the data

Accessing the data set was trivial, the data set can be found in the website named data world. You can also use this link to access the data. The dataset was downloaded from the website and the file which has an extension as .csv (autos.csv) was the actual data set file. The original size of the CSV file was 69 MB (Megabytes). Later the data was formatted into a data frame the data because in the Jupyter notebook or Google Colab environment we cannot perform any operations on the raw CSV file, so we need to convert the raw data file and store it in a data frame.

3.1.4 Format of the data

The formatting did not take much time because the data was already stored as a CSV (Comma Seperated Value) file. By importing the pandas library, we can read the CSV file and then store data into a data frame. Shown below is the data in a raw CSV format.

dateCrawled	name	seller	offerType	price	abtest	vehicleTyp	yearOfReg	gearbox	powerPS	model
3/24/2016 11:52	Golf_3_1.6	privat	Angebot	480	test		1993	manuell	0	golf
3/24/2016 10:58	A5_Sportback_2.7_Tdi	privat	Angebot	18300	test	coupe	2011	manuell	190	
3/14/2016 12:52	Jeep_Grand_Cherokee_"Overland"	privat	Angebot	9800	test	suv	2004	automatik	163	grand
3/17/2016 16:54	GOLF_4_1_4__3TÜRER	privat	Angebot	1500	test	kleinwage	2001	manuell	75	golf
3/31/2016 17:25	Skoda_Fabia_1.4_TDI_PD_Classic	privat	Angebot	3600	test	kleinwage	2008	manuell	69	fabia
4/4/2016 17:36	BMW_316i___e36_Limousine___Bastlerfahrzeug__Export	privat	Angebot	650	test	limousine	1995	manuell	102	3er
4/1/2016 20:48	Peugeot_206_CC_110_Platinum	privat	Angebot	2200	test	cabrio	2004	manuell	109	2_reihe
3/21/2016 18:54	VW_Derby_Bj_80__Scheunenfund	privat	Angebot	0	test	limousine	1980	manuell	50	andere
4/4/2016 23:42	Ford_C___Max_Titanium_1_0_L_EcoBoost	privat	Angebot	14500	control	bus	2014	manuell	125	c_max
3/17/2016 10:53	VW_Golf_4_5_tuerig_zu_verkaufen_mit_Anhaengerkupplung	privat	Angebot	999	test	kleinwage	1998	manuell	101	golf
3/26/2016 19:54	Mazda_3_1.6_Sport	privat	Angebot	2000	control	limousine	2004	manuell	105	3_reihe
4/7/2016 10:06	Volkswagen_Passat_Variant_2.0_TDI_Comfortline	privat	Angebot	2799	control	kombi	2005	manuell	140	passat
3/15/2016 22:49	VW_Passat_Facelift_35i__"7Sitzer"	privat	Angebot	999	control	kombi	1995	manuell	115	passat
3/21/2016 21:37	VW_PASSAT_1.9_TDI_131_PS_LEDER	privat	Angebot	2500	control	kombi	2004	manuell	131	passat
3/21/2016 12:57	Nissan_Navara_2.5DPF_SE4x4_Klima_Sitzheizg_Bluetooth.Doppelkabine	privat	Angebot	17999	control	suv	2011	manuell	190	navara
3/11/2016 21:39	KA_Lufthansa_Edition_450€_VB	privat	Angebot	450	test	kleinwage	1910		0	ka
4/1/2016 12:46	Polo_6n_1_4	privat	Angebot	300	test		2016		60	polo
3/20/2016 10:25	Renault_Twingo_1.2_16V_Aut.	privat	Angebot	1750	control	kleinwage	2004	automatik	75	twingo
3/23/2016 15:48	Ford_C_MAX_2.0_TDCi_DPF_Titanium	privat	Angebot	7550	test	bus	2007	manuell	136	c_max

Figure 3.2: Raw CSV format of the dataset

After formatting the data from CSV to the data frame, the data looks cleaner and easier to understand which can be found below:

```
import pandas as pd
import time
start_time = time.time()
df = pd.read_csv("/content/drive/My Drive/Dataset/autos.csv", sep = ',', header = 0, encoding='cp1252')
print("--- %s seconds ---" % (time.time() - start_time))
df.head(5)
```

```
--- 9.454026222229004 seconds ---
```

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

Figure 3.3: Dataset formatted as a pandas dataframe

To avoid the `utf-8 error (can't decode byte 0xdc in position 14)`, a dif-

ferent way was followed to read the CSV file such as using the parameters such as **sep as** ",", **header** as 0 and **encoding** as cp1252 were used in the `read_csv` method. It is mainly because the actual data set CSV file doesn't contain utf-8 encoded data, it contains some other encoding. We manually have to encode it using cp1252.

3.2 Statistics of data

3.2.1 Determining the number of instances and features of the data set

As we all know that the dataset is a collection of labelled examples i.e.

$$\{(x_i, y_i)\}_{i=1}^{N}$$

Each element xi among N is called a feature vector. The feature vector is a value that describes the example in a dataset [4]. This value is called as a feature which is represented as shown below:

$$x^j$$

A feature vector is a vector in which each dimension `j = 1, . . . , D` contains a value that describes the dataset somehow [4]. Similarly, this data set comprises **371,528** instances and **20** features. The pandas data frame library has a method called shape, which helps in determining the dimensions of the data set. The shape method returns the number of rows and columns of the data frame.

```
df.shape                    # The shape method returns the total number of rows and columns of the dataframe
```

```
(371528, 20)
```

Figure 3.4: Shape of the dataframe

The instances here represent different individual record of a used car along with

their details. The features represent different attributes or features of the used car such as the name, gearbox, vehicle type, year of registration, power ps, model, kilometer, fuel type, brand and 11 more features as shown below.

```
features = list(df.columns.values)     # Displaying all the values of the columns
features
```

```
['dateCrawled',
 'name',
 'seller',
 'offerType',
 'price',
 'abtest',
 'vehicleType',
 'yearOfRegistration',
 'gearbox',
 'powerPS',
 'model',
 'kilometer',
 'monthOfRegistration',
 'fuelType',
 'brand',
 'notRepairedDamage',
 'dateCreated',
 'nrOfPictures',
 'postalCode',
 'lastSeen']
```

Figure 3.5: Features list

Below are the details of each features

- **dateCrawled** : when this ad was first crawled, all field-values are taken from this date
- **name** : "name" of the car
- **seller** : private or dealer
- **offerType**: With offer or without offer
- **price** : the price on the ad to sell the car
- **abtest**: Test on the car
- **vehicleType**: Type of the car (Sedan, truck, etc.)
- **yearOfRegistration** : at which year the car was first registered
- **gearbox**: Automatic or manual transmission
- **powerPS** : power of the car in PS
- **model**: Model of the car
- **kilometer** : how many kilometers the car has driven
- **monthOfRegistration** : at which month the car was first registered
- **fuelType**: Gas, Petrol, Diesel, etc.
- **brand**: Mercedes, Audi, BMW, etc.
- **notRepairedDamage** : if the car has a damage which is not repaired yet
- **dateCreated** : the date for which the ad at ebay was created
- **nrOfPictures** : number of pictures in the ad (unfortunately this field * contains everywhere a 0 and is thus useless (bug in crawler!))
- **postalCode**: Area wise postal code
- **lastSeenOnline** : when the crawler saw this ad last online

Figure 3.6: Details of features of the dataframe

3.2.2 Determining the dimensions of the data set

Also, the pandas `ndim` method was used to determine the dimension of the data frame which resulted in 2 meaning the data frame is of 2 dimensions.

```
df.ndim      # Returns the dimensions of the dataframe
```

```
2
```

Figure 3.7: Dimensions of the dataframe

3.2.3 Performing a 5-number summary (min, low quartile, median, upper quartile, max)

The 5-number summary technique works only on numeric type data, it doesn't work on categorical data. It is used to display the median, 1st Quartile (25th percentile), 3rd Quartile (75th percentile), minimum and maximum values of each features. The result of the 5-number summary technique is shown in the below.

```
df.describe()    # Getting descriptive statistics
```

	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

Figure 3.8: 5 number summary of the data frame

3.2.4 Determining the data types

Determining the data type is one of the most important steps performed in data processing. This can be done with the help of the pandas method called `dtypes` or `info` as shown below.

```
df.info()                                    # Getting information about the datatypes alternative to df.dtypes()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 371528 entries, 0 to 371527
Data columns (total 20 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   dateCrawled          371528 non-null  object
 1   name                 371528 non-null  object
 2   seller               371528 non-null  object
 3   offerType            371528 non-null  object
 4   price                371528 non-null  int64
 5   abtest               371528 non-null  object
 6   vehicleType          333659 non-null  object
 7   yearOfRegistration   371528 non-null  int64
 8   gearbox              351319 non-null  object
 9   powerPS              371528 non-null  int64
 10  model                351044 non-null  object
 11  kilometer            371528 non-null  int64
 12  monthOfRegistration  371528 non-null  int64
 13  fuelType             338142 non-null  object
 14  brand                371528 non-null  object
 15  notRepairedDamage    299468 non-null  object
 16  dateCreated          371528 non-null  object
 17  nrOfPictures         371528 non-null  int64
 18  postalCode           371528 non-null  int64
 19  lastSeen             371528 non-null  object
dtypes: int64(7), object(13)
memory usage: 56.7+ MB
```

Figure 3.9: Data type of the features in the data frame

As seen above, there are two types of data such as the numeric (integer) 7 and categorical (object) 13 data. Most of the machine learning algorithms don't accept the data to be categorical in such cases we have to typecast it or encode it.

3.3 Data Exploration

3.3.1 Removing Outliers

In this phase, two most important techniques such as manually removing the outliers by using threshold values and IQR technique would be followed.

Using threshold values to manually remove the outliers

When we see the above results got from the summary there are few things that catch our attentions and these things need to be fixed immediately such as

- How can the minimum value of a price be 0?

- How can the year of registration of a vehicle be 1000?
- How can the minimum power of a vehicle be 0?
- How can the month of registration of a vehicle be 0?

So, there is no point in keeping these equivalent rows, removing them is the only option. Let's set a threshold value for all these above features and remove them accordingly. For example

- Retaining the used cars whose year of registration is between 1950 and 2019 and removing the rest.
- Retaining the power of the vehicle between 100, 1500. This is because the one of lowest powered car is Mitsubishi Mirage, its power is 78 ps and the car with the highest power is Lotus Evija 2000 ps power. Setting the threshold according to these cars will help us in our exploration.
- Keeping the price of the vehicle between 100 to 200,000. Because one of the most expensive cars in Europe is Maybach 62 which is 400,000 euros. And there are no such cars in this data frame
- All the vehicles whose month of registration is 0 was removed. Because the month can never be 0

We can also visualize these with the help of a density histogram plot. First all the values will be normalized to get a better plot result. And the range with the probability density function curve would be plotted with the range (0, 1) (bell curve).

Now all the negative values along with the 0 values must be removed. Below is the code snippet which uses threshold values to remove the outliers.

```
import numpy as np
import pandas as pd
from scipy.stats import norm
import matplotlib.pyplot as plt
df = pd.DataFrame({'price': np.random.normal(size = 100)})
df.price.plot(kind = 'hist', density = True)
range = np.arange(-3, 3, 0.001)
plt.plot(range, norm.pdf(range, 0, 1))
```

```
[<matplotlib.lines.Line2D at 0x7f7e144e4208>]
```

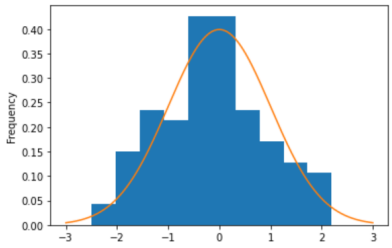

Figure 3.10: Price density normal histogram plot

```
import numpy as np
import pandas as pd
from scipy.stats import norm
import matplotlib.pyplot as plt
df = pd.DataFrame({'powerPS': np.random.normal(size = 100)})
df.powerPS.plot(kind = 'hist', density = True)
range = np.arange(-3, 3, 0.001)
plt.plot(range, norm.pdf(range, 0, 1))
```

```
[<matplotlib.lines.Line2D at 0x7f7e14337cc0>]
```

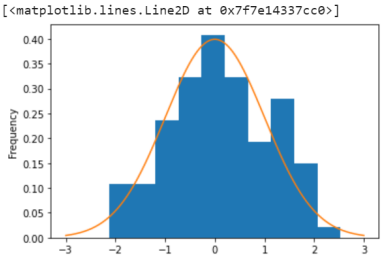

Figure 3.11: Power density normal histogram plot

```
import numpy as np
import pandas as pd
from scipy.stats import norm
import matplotlib.pyplot as plt
df = pd.DataFrame({'yearOfRegistration': np.random.normal(size = 100)})
df.yearOfRegistration.plot(kind = 'hist', density = True)
range = np.arange(-3, 3, 0.001)
plt.plot(range, norm.pdf(range, 0, 1))
```

```
[<matplotlib.lines.Line2D at 0x7f7e145d2e48>]
```

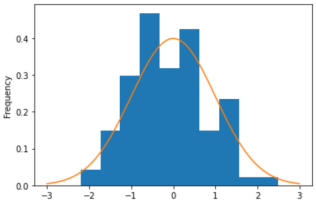

Figure 3.12: Year of registration density normal histogram plot

```
import numpy as np
import pandas as pd
from scipy.stats import norm
import matplotlib.pyplot as plt
df = pd.DataFrame({'monthOfRegistration': np.random.normal(size = 100)})
df.monthOfRegistration.plot(kind = 'hist', density = True)
range = np.arange(-3, 3, 0.001)
plt.plot(range, norm.pdf(range, 0, 1))
```

```
[<matplotlib.lines.Line2D at 0x7f7e1e371898>]
```

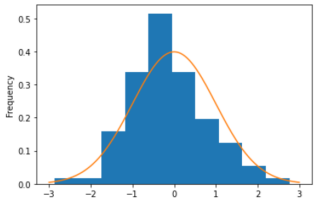

Figure 3.13: Month of registration density normal histogram plot

```
# Cleaning the expectional values from the dataframe. The threshold values can be set using the df[].between methods.
df_clean = df[
                (df["yearOfRegistration"].between(1950, 2019, inclusive=True)) &
                (df["powerPS"].between(100, 1500, inclusive=True)) &
                (df["price"].between(100, 200000, inclusive=True))
            ]
```

```
# Storing all the indexes who's month of registration is 0
monthOfRegistration = df_clean[df_clean['monthOfRegistration'] == 0 ].index
# Removing all the desired index values using drop()
df_clean.drop(monthOfRegistration , inplace=True)
```

Figure 3.14: Setting the threshold to remove the outliers

Removing the outliers using IQR (Inter Quartile Range) technique

One of the best ways to detect outliers is using a box plot. An outlier will be plotted as point in boxplot whereas other points will be grouped together and display as

boxes as shown below.

```
df_clean['price'].plot.box()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7faac4817940>
```

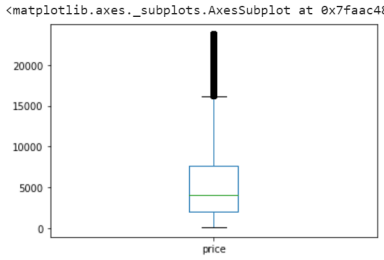

Figure 3.15: Box plot - Price

```
df_clean['powerPS'].plot.box()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7faac46e6fd0>
```

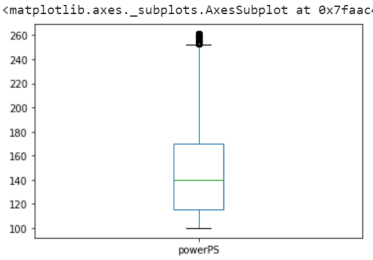

Figure 3.16: Box plot - Power

```
df_clean['yearOfRegistration'].plot.box()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7faac465f208>
```

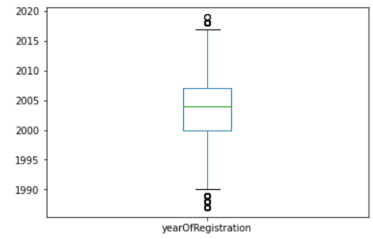

Figure 3.17: Box plot - Year of Registration

A commonly used technique known as IQR suggests that a data point is an outlier if it is more than `1.5 * IQR` above the third quartile or below the first quartile. Low outliers are below `Q1 - 1.5 * IQR` and high outliers are above `Q3 + 1.5 * IQR`. The IQR score of each column is calculated as shown below [5].

```
Q1 = df_clean.quantile(0.25)
Q3 = df_clean.quantile(0.75)
IQR = Q3 - Q1
IQR
```

```
price                8540.5
yearOfRegistration      9.0
powerPS                58.0
kilometer           25000.0
monthOfRegistration     5.0
nrOfPictures            0.0
postalCode          41679.0
dtype: float64
```

Figure 3.18: Calculating the IQR score for all the columns

We can use previously calculated IQR score to filter out the outliers by keeping only valid values. By doing so all the outliers were completely removed.

```
df_clean = df_clean[~((df_clean < (Q1-1.5 * IQR)) |(df_clean > (Q3 + 1.5 * IQR))).any(axis=1)]
```

Figure 3.19: Cleaning all the columns values based on the IQR scores generated above

3.3.2 Removing irrelevant columns

In this section all the irrelevant column will be removed that do not contribute for the price prediction.

Checking if the sum of any columns is returning zero, if yes then there it is certain we need to remove it because of its null contents. For some reason the code `df.sum(axis = 0)` was not working, it was taking more time and yet not executing, making the notebook to crash.

```
# Taking the sum to see if any columns has 0 values in it
print(df_clean['kilometer'].sum())
print(df_clean['price'].sum())
print(df_clean['yearOfRegistration'].sum())
print(df_clean['powerPS'].sum())
print(df_clean['monthOfRegistration'].sum())
print(df_clean['nrOfPictures'].sum())
print(df_clean['postalCode'].sum())
```

```
25393270000
1669773090
401813515
31318400
1286709
0
10555500224
```

Figure 3.20: Checking to see if any columns have sum as zero

As seen above the sum of `nrOfPictures` resulting to 0, this is an indication that there are no pictures. Removing it, was the only option.

Using "groupby" method to remove the irrelevant columns

Seller

There was only one instance of the seller **"gewerblich"** which is **"commercial"** in english. Since all the values in this coloumns seller are private, there are no other mixed values, removing this feature was the only option.

```
df_clean.groupby('seller').count()
```

seller	dateCrawled	name	offerType	price	abtest	vehicleType	yearOfRegistration	gearbox	powerPS	model	kilometer	monthOfRegistration
gewerblich	1	1	1	1	1	1	1	1	1	1	1	1
privat	200458	200458	200458	200458	200458	191784	200458	198592	200458	193902	200458	200458

Figure 3.21: Grouping the values in the seller column

Offer Type

This column indicated whether the vehicle has any offer or not. Since there is only one category of offerType callde **Angebot** which is **offer**. There was no point in keeping this column.

```
df_clean.groupby('offerType').count()
```

offerType	dateCrawled	name	price	abtest	vehicleType	yearOfRegistration	gearbox	powerPS	model	kilometer	monthOfRegistration
Angebot	200458	200458	200458	200458	191784	200458	198592	200458	193902	200458	200458

Figure 3.22: Grouping the values in the offer type column

Removing other columns

The reason for dropping all the below columns is that no insights could be drawn from those columns. Such as the date crawled and created contribute nothing because these are used indicated the scraping details of the data. Similarly, the **abtest** columns is not known. For this reason the original was contacted but no information was provided on this. Last, the **lastseen** indicated the status of the vehicle on the e-bay website, so this will not help in the price prediction. Similarly, the name of the car was removed, because we already have the model, brand of the cars which are

the supporting features. As there were more null values in the **notRepairedDamage** columns, rather than filling it with an instance called **nein** (No) it was removed too.

```
# dropping columns
df_clean = df_clean.drop(columns= ['dateCrawled', 'name', 'abtest', 'dateCreated', 'lastSeen', 'notRepairedDamage'])
```

Figure 3.23: Dropping all the irrelevant columns

Finally, the mostly cleaned data with all the irrelevant removed columns is as shown below:

```
df_clean.head(5)
```

	price	vehicleType	yearOfRegistration	gearbox	powerPS	model	kilometer	monthOfRegistration	fuelType	brand	postalCode
1	18300	coupe	2011	manuell	190	NaN	125000	5	diesel	audi	66954
2	9800	suv	2004	automatik	163	grand	125000	8	diesel	jeep	90480
5	650	limousine	1995	manuell	102	3er	150000	10	benzin	bmw	33775
6	2200	cabrio	2004	manuell	109	2_reihe	150000	8	benzin	peugeot	67112
8	14500	bus	2014	manuell	125	c_max	30000	8	benzin	ford	94505

Figure 3.24: Mostly cleaned data frame

3.3.3 Dealing with the missing values

In this section, a technique named one-to-one mapping and max frequency was used to find and deal with missing values.

Finding the count of missing values

```
df_clean.isnull().sum()
```

```
price                   0
vehicleType          8674
yearOfRegistration      0
gearbox              1866
powerPS                 0
model                6556
kilometer               0
monthOfRegistration     0
fuelType             8419
brand                   0
postalCode              0
dtype: int64
```

Figure 3.25: Total count of null values in the data frame

Also we can use a searborn plot to find all the null values. This plot uses the heatmap to plot all the null values in the dataframe. The horizontal lines inside the plot are the null values.

```
sns.heatmap(df_clean.isnull(), cbar=False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8b02cae400>
```

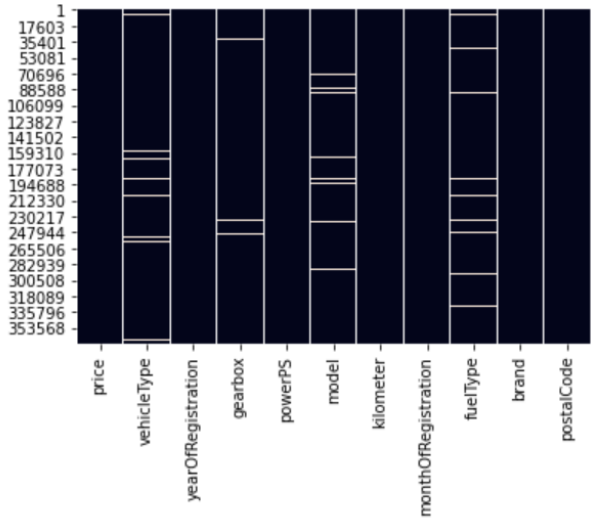

Figure 3.26: Using a seaborn plot to detect null values

As seen from the above results, there were many missing values in vehicleType, gearbox, model, fuelType columns. Now rather than deleting all the null values, an alternative approach was followed. Because deleting would a lame and useless method. Also, one of the data Impuatation techniques mentioned in the 100 page machine learning book [4] such as replacing the mean values, using a learning algorithm, or even interpolating cannot be applied. This is because of the datatyes of the dataframe. All the datatypes were not integer values, they were strings. So below is the approach that was followed to handle the missing values.

One to One Mapping

Using the column brand to fill the gearbox values

Since the column brand has no missing values, we can use this as a reference to fill all the gearbox values.

```
# Using brand values to fill the gearbox values
gearbox = df_clean["gearbox"].unique()
brand = df_clean["brand"].unique()
d = {}
for i in brand :
    m = 0
    for j in gearbox :
        if df_clean[(df_clean.gearbox == j) & (df_clean.brand == i)
   ].shape[0] > m :
            m = df_clean[(df_clean.gearbox == j) & (df_clean.brand
   == i)].shape[0]
            d[i] = j

for i in brand:
    df_clean.loc[(df_clean.brand == i) & (df_clean.gearbox.isnull())
    ,"gearbox" ] = d[i]
```

Code Listing 3.1: One to one mapping of brand to gearbox values

Using the column fuel type to fill the vehicle type values

Since both the column fuel type and the vehicle type are closely related we can use the vehicle type values to map the fuel type values.

```
df_clean.groupby("fuelType")["vehicleType"].value_counts()
```

```
fuelType  vehicleType
andere    limousine          9
          kombi              8
          bus                4
          suv                4
          andere             3
          cabrio             2
          kleinwagen         2
          coupe              1
benzin    limousine      42195
          kombi          21284
          cabrio         14724
          coupe          11946
          bus             7087
          kleinwagen      6435
          suv             3924
          andere           625
cng       bus              118
          kombi             93
          limousine         25
          andere             7
          kleinwagen         3
          cabrio             1
          coupe              1
          suv                1
diesel    kombi          29424
          limousine      22632
          bus            13572
          suv             7759
```

Figure 3.27: Grouping the vehicle type with the fuel type

```
# Using the vehicle type column to fill the fuel type column
d = {}
for i in fuelType :
    m = 0
    for j in vehicleType :
```

```
        if df_clean[(df_clean.vehicleType == j) & (df_clean.fuelType
    == i)].shape[0] > m :
            m = df_clean[(df_clean.vehicleType == j) & (df_clean.
   fuelType == i)].shape[0]
            d[i] = j

for i in fuelType :
    df_clean.loc[(df_clean.fuelType == i) & (df_clean.vehicleType.
   isnull()) ,"vehicleType" ] = d[i]
```

Code Listing 3.2: One to one mapping of fuel type to vehicle type values

Max frequency technique

Using the feature value petrol to fill all the missing fuel type values

The most common value is `benzin` and `diesel`. Petrol (benzin) would be replaced for the null values.

```
print(df_clean["fuelType"].value_counts())
df_clean["fuelType"].fillna("benzin",inplace = True)
print("The total null values of fuelType is = ",df_clean['fuelType'].isnull().sum())
```

```
benzin     114917
diesel      81277
lpg          3831
cng           262
hybrid        113
andere         35
elektro        23
Name: fuelType, dtype: int64
The total null values of fuelType is =  0
```

Figure 3.28: Using the value petrol to fill all the missing fuel type values

Filling the model values with of '3er' (3 series) values

In the model column the majority of the values were 3er (3 series). So we can use the max frequency technique to fill all the null values with **3er**

```
print(df_clean['model'].value_counts())
df_clean["model"].fillna("3er",inplace =True)
print("The total null values of model is = ",df_clean['model'].isnull().sum())
```

```
3er                23708
andere             14756
golf               14084
a4                  8692
passat              7917
                   ...
up                     1
cuore                  1
kalos                  1
discovery_sport        1
charade                1
Name: model, Length: 235, dtype: int64
The total null values of model is =  0
```

Figure 3.29: Filling the model columns with "3er"

3.3.4 Dealing with duplicate values

As this data frame has many rows, there were many duplicated rows. One of the important steps in data exploration is dealing with duplicate values. Since there were no unique identifier values like ID or License plate number of a car, it was tricky to filter the duplicate values. The important columns such as brand, model, price, kilometer and postal code were grouped and assigned a unique identifier. This was done with the help of `ngroup()` in pandas, basically this method returns the unique number of each groups.

```
df_clean['id'] =  df_clean.groupby(['postalCode', 'brand', 'model', 'powerPS', 'kilometer', 'price']).ngroup()
df_clean.sort_values("id", inplace = True)
df_clean['id']
```

```
317154         0
84265          1
345995         2
357801         3
268203         4
           ...
298829    181185
82342     181186
251479    181187
231158    181188
331008    181189
Name: id, Length: 200458, dtype: int64
```

Figure 3.30: Assigning a unique identifier for the groups

As seen above the unique values have been assigned to the column **id**. Now the column was sorted as seen above. Since there were over 1000 duplicate values all of them had to be removed. An example of such duplicate value is shown below:

```
df_clean.loc[df_clean['id'] == 14]
```

	price	vehicleType	yearOfRegistration	gearbox	powerPS	model	kilometer	monthOfRegistration	fuelType	brand	postalCode	id
57167	12999	coupe	2006	manuell	218	3er	125000	12	benzin	bmw	1067	14
307326	12999	coupe	2006	manuell	218	3er	125000	12	benzin	bmw	1067	14

Figure 3.31: Duplicate rows with the same id number

3.3.5 Translating the data frame from German to English

In this section, details about translation of the data frame from German to English language will be provided. Rather than manually translating the values using Google Translate service one by one, it might take years because of the size of the data frame. With the help of the translator function shown below, the entire data frame can be converted into the desired language of our choice, in this case it's English. First, to use the function we need to manually install the **google trans python** library and then invoke the translator function. Basically, the working of the translator function is simple all the unique value of the columns in a data frame must be passed into the built-in translator functions and then the function maps the original data with the newly generated translated data.

```
!pip install googletrans
import googletrans
from googletrans import Translator
import pandas as pd

translator = Translator()
translations = {}

for column in df.columns:
    # unique elements of the column
    unique_elements = df[column].unique()
    for element in unique_elements:
        # add translation to the dictionary
        translations[element] = translator.translate(element).text
```

Code Listing 3.3: Translating dataframe from German to English

```
df.head(10)
```

	seller	offerType	abtest	vehicleType	gearbox	model	fuelType	brand	notRepairedDamage
3	private	offer	test	small car	manually	golf	gasoline	volkswagen	No
4	private	offer	test	small car	manually	fabia	diesel	skoda	No
5	private	offer	test	limousine	manually	Presentation	gasoline	bmw	yes
6	private	offer	test	cabrio	manually	2_reihe	gasoline	peugeot	No
7	private	offer	test	limousine	manually	Others	gasoline	volkswagen	No
10	private	offer	control	limousine	manually	3_reihe	gasoline	mazda	No
11	private	offer	control	combi	manually	passat	diesel	volkswagen	yes
14	private	offer	control	suv	manually	navara	diesel	nissan	No
17	private	offer	control	small car	automatic	twingo	gasoline	renault	No
18	private	offer	test	bus	manually	c_max	diesel	ford	No

Figure 3.32: Displaying top 10 rows the translated data

```
df.tail(10)
```

	seller	offerType	abtest	vehicleType	gearbox	model	fuelType	brand	notRepairedDamage
371512	private	offer	test	limousine	automatic	e_klasse	diesel	Mercedes Benz	yes
371513	private	offer	control	limousine	manually	leon	diesel	seat	No
371516	private	offer	control	small car	manually	wolf	gasoline	volkswagen	No
371517	private	offer	test	limousine	manually	golf	diesel	volkswagen	No
371518	private	offer	test	combi	manually	Presentation	diesel	bmw	No
371520	private	offer	control	limousine	manually	leon	gasoline	seat	yes
371521	private	offer	control	bus	manually	zafira	gasoline	opel	No
371524	private	offer	test	cabrio	automatic	fortwo	gasoline	smart	No
371525	private	offer	test	bus	manually	transporter	diesel	volkswagen	No
371527	private	offer	control	limousine	manually	m_reihe	gasoline	bmw	No

Figure 3.33: Displaying last 10 rows the translated data

As a result, around 95% of the German data were successful translated into English. There are some of the tradeoff or restrictions that was dealt during implementing the translating function.

- There must be no null, or missing values in the data frame. Otherwise the functions returns an error.
- The translator function could not fully translate the data when it was followed by a numeric value.
- The translator function doesn't work on numeric data. It only works on string or categorical data.
- The translator function takes a lot of time to execute, the maximum time that was recorded was 33 minutes. Sometimes if we run it for more than once, it crashes then we must manually clear or restart the runtime in a Jupyter notebook environment. Apart from these minute drawbacks the translator function works as intended.

3.3.6 Dealing with the datatypes

In this section, all the object data types was converted to the categorical data type, and also all the categorical data columns would be assigned a code called categorical codes. This step is very important because most of the machine learning algorithms doesnot support string type data, meaning all the value should not be string rather they must be numeric. The conversion is as shown below:

```
for col in ['brand','vehicleType', 'gearbox', 'model', 'fuelType']:
    df_clean[col] = df_clean[col].astype('category')

# Assign codes to categorical attribues instead of strings
df_cloumns = df_clean.select_dtypes(['category']).columns
df_clean[df_cloumns] = df_clean[df_cloumns].apply(lambda x: x.cat.
   codes)
```

Code Listing 3.4: Converting the datatypes into categorical data and assigning codes

Below shown is the data frame where all the string values converted to numeric values with assign codes by the cat code (pandas method).

```
df_clean.head(5)
```

	price	vehicleType	yearOfRegistration	gearbox	powerPS	model	kilometer	monthOfRegistration	fuelType	brand	postalCode
1	18300	3	2011	1	190	11	125000	5	3	1	66954
2	9800	7	2004	0	163	114	125000	8	3	14	90480
5	650	6	1995	1	102	11	150000	10	1	2	33775
6	2200	2	2004	1	109	8	150000	8	1	25	67112
8	14500	1	2014	1	125	58	30000	8	1	10	94505

Figure 3.34: Values converted from string to numeric

One Hot Encoding

Initially one hot encoding was implemented on the entire data frame, but at the end there were 301 columns. It was very hard to process and render this data into the model. As it was taking a lot of time. So a decision not to encode all the columns using

onehot encoding was made. Further the one hot encoding will be only implemented only on **vehicleType** and **gearbox**.

```
df_clean=pd.get_dummies(data=df_clean,columns=['notRepairedDamage','vehicleType','model','brand','gearbox','fuelType'])
# cars_dummies = cars_updated.drop(columns=['notRepairedDamage','vehicleType','model','brand','gearbox','fuelType'])
```

	name	offerType	price	yearOfRegistration	powerPS	kilometer	monthOfRegistration	postalCode	notRepairedDamage_0	notRepairedDamage_1	vehicleT
1	1949	0	18300	2011	190	125000	5	66954	1	0	
2	52968	0	9800	2004	163	125000	8	90480	0	1	
5	19474	0	650	1995	102	150000	10	33775	1	0	
6	83138	0	2200	2004	109	150000	8	67112	0	1	
8	41838	0	14500	2014	125	30000	8	94505	0	1	

5 rows × 301 columns

Figure 3.35: One hot encoding implemented on all the columns

```
df_clean=pd.get_dummies(data=df_clean,columns=['vehicleType','gearbox'])
df_clean.head(5)
```

	price	yearOfRegistration	powerPS	model	kilometer	monthOfRegistration	fuelType	brand	postalCode	vehicleType_0	vehicleType_1	vehicleType_2
1	18300	2011	190	11	125000	5	3	1	66954	0	0	0
2	9800	2004	163	114	125000	8	3	14	90480	0	0	0
5	650	1995	102	11	150000	10	1	2	33775	0	0	0
6	2200	2004	109	8	150000	8	1	25	67112	0	0	1
8	14500	2014	125	58	30000	8	1	10	94505	0	1	0

Figure 3.36: One hot encoding implemented on vehicle type and gearbox columns

3.3.7 Feature Selection

Feature selection techniques are easy to use and also give good results. In feature selection, a technique called uni-variate selection was performed. Statistical tests can be used to select these features that have the strongest relationship with the output variable. The **scikit-learn** library provides the **SelectKBest** class that can be used with a suite of different statistical tests to select a specific number of features.

Chi-Squared Statistical Test

Below shown is the chi-squared (chi^2) statistical test for non-negative features to select 10 of the best features. Basically in order to perform the chi-squared test we

need to divide the features into dependent and independent features. In this case the price and the features of the car.

```
X = df_clean.drop('price', axis=1)
y = df_clean['price']

#apply SelectKBest class to extract top 10 best features
bestfeatures = SelectKBest(score_func=chi2, k=10)
fit = bestfeatures.fit(X,y)

dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)

#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Features','Score']  #naming the dataframe
    columns

featureScores.nlargest(10,'Score')  #print 10 best features
```

Code Listing 3.5: Performing Chi-Sqaured Test on features

So the most dependent features on the price is as shown below:

Features	Score
kilometer	8.093982e+08
postalCode	7.993496e+07
powerPS	1.354623e+06
model	3.623605e+05
brand	6.458665e+04
fuelType	9.892546e+03
monthOfRegistration	8.880040e+03
gearbox	6.577627e+03
vehicleType	5.282056e+03
yearOfRegistration	1.261282e+03

Figure 3.37: Chi square test for getting the important features

Correlation Matrix with Heatmap

Correlation states the features that are closely related to each other or the target variable. Correlation can be positive (increase in one value of feature increases the value of the target variable) or negative (increase in one value of feature decreases the value of the target variable). Heatmap makes it easy to identify which features are most related to the target variable, we will plot heatmap of correlated features using the **seaborn** library.

```
import seaborn as sns
#get correlations of each features in dataset
corrmat = df_clean.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(10,10))
g=sns.heatmap(df_clean[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```

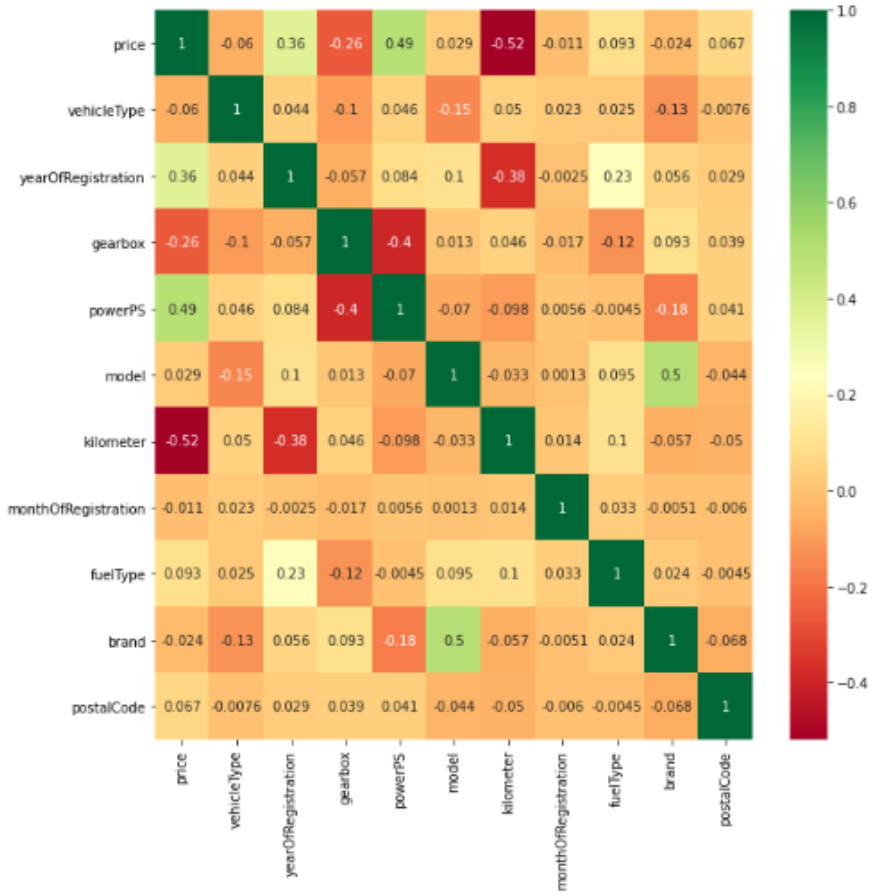

Figure 3.38: Heat Map

3.4 Visualization

3.4.1 Hexagonal binning plot

Plotting the price features against all other features. Hexbin plots can be a useful alternative to scatter plots if the data points are too dense to plot each point individually. A useful keyword argument was gridsize; it controls the number of hexagons in the x-direction, and defaults to 100. A larger criticize means more smaller bins.

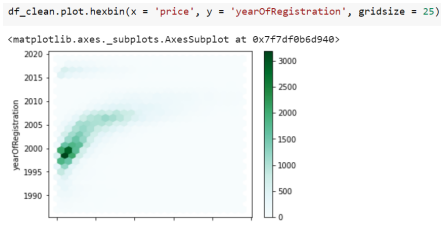

Figure 3.39: Hexbin plot - Price vs Year of Registration

Figure 3.40: Hexbin plot - Price vs Month of Registration

Figure 3.41: Hexbin plot - Price vs Model

Figure 3.42: Hexbin plot - Price vs Brand

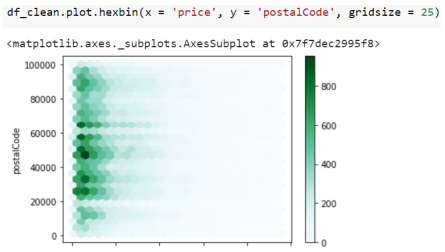

Figure 3.43: Hexbin plot - Price vs Postal code

Figure 3.44: Hexbin plot - Price vs Kilometer

3.4.2 Scatter Matrix Plot

The scatter matrix plot is a plot invoked from the pandas plot function. Firstly it divides all the values into a matrix and then plots them as shown below.

```
num_attributes = ["price", "yearOfRegistration", "powerPS", "kilometer"]
pd.plotting.scatter_matrix(df_clean[num_attributes], figsize = (12,8), alpha = 0.1)
```

Figure 3.45: Scatter matrix

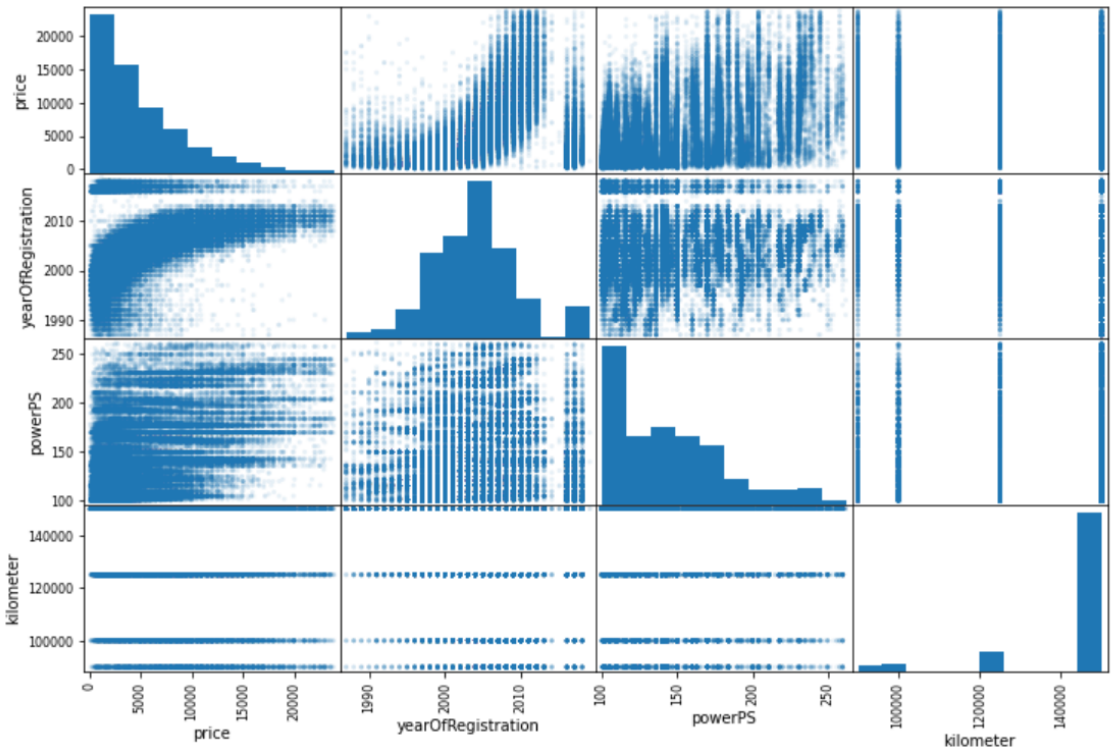

Figure 3.46: Scatter plot of all the features

The thoroughly cleaned data set can be found here

Chapter 4

Model Planning

In this chapter, the model planning phase will be discussed, which includes variable selection, model selection, category of techniques.

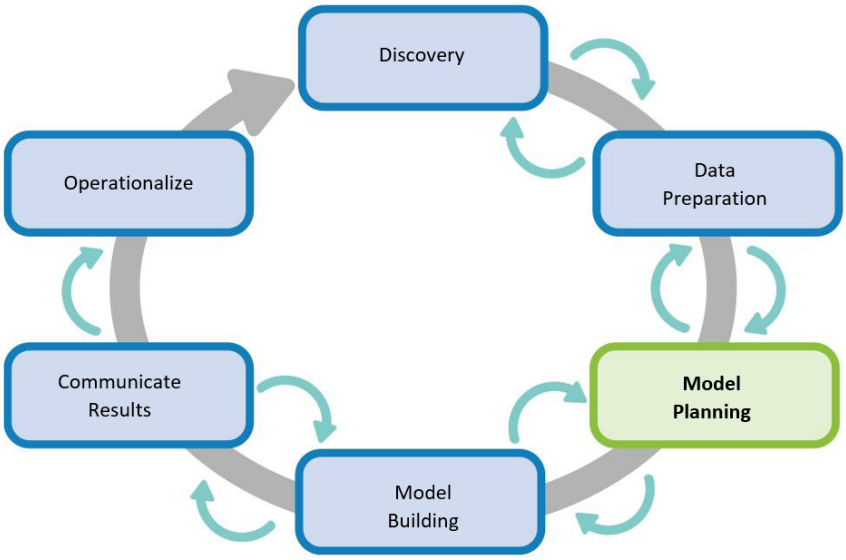

Figure 4.1: Model Planning

4.1 Variable Selection

As the data was thoroughly explored and conditioned in the previous stage, the variables i.e features must be selected in this phase. As we all know there are two types of variables: independent and dependent variables. Below is the variable selection code snippet. Also, the cleaned data set can be found here

```
selectedFeatures =
  [
  'yearOfRegistration','powerPS','model','kilometer','
   monthOfRegistration','fuelType','brand','postalCode','
   vehicleType_0','vehicleType_1','vehicleType_2','vehicleType_3','
   vehicleType_4','vehicleType_5','vehicleType_6','vehicleType_7','
   gearbox_0','gearbox_1'
  ]

X = df[selectedFeatures]
y = df['price']
```

Code Listing 4.1: Variable selection

Here the feature separation was prepared in a particular order because wrong mismatch order of data can cause an error. The **'X'** variable contains all the features except the price and it of 2 dimensions. Similarly, the **'y'** variable contains only price values and its of 1 dimension.

4.2 Model Selection

There are 4 types of models in machine learning

- Classification
- Regression
- Association rules
- Text/Image/Video Analysis

Since the problem analysis was to determine the relationship between the outcome and the input variables, Hence only regression was used the model selection.

Chapter 5

Model Building

In this chapter, the model building phase will be discussed, which includes splitting the data sets into 3 sets, choosing the model, checking for overfitting and underfitting conditions and the hyperparameter tuning.

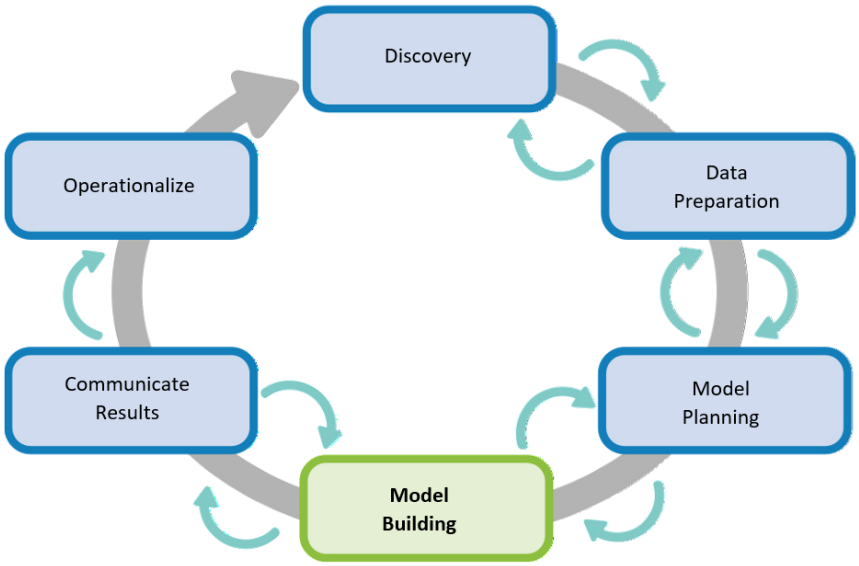

Figure 5.1: Model Building

5.1 Splitting the dataset into three sets

The data set will be splitted into three distinct sets of labeled examples as shown below:

- Training set
- Validation set
- Testing set

The holdout sets in this case will be the validation and testing set. Since this is not a big data, 80% will be the training set, 10% will be validation and 10% will be testing set [4]. In this case,

- The training will be only used for building (training) the model. The training set comprises of 72029 samples and it can be located here
- Validation set will be used to choose the learning algorithm, find the best values of hyper-parameters. The validation set comprises of 9416 samples and it can be located here
- Testing set will be used when the model is deployed in production, or for testing purpose. The testing set comrises 9416 samples and it can be located here

The `sklearn.model_selection.train_test_split` twice to split the data set into three sets. First to split to train, test and then split train again into validation and train.

```
from sklearn.model_selection import train_test_split

# Training set = 90%, Testing set = 10%, Validation set = 10%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=1)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.1, random_state=1)
```

Figure 5.2: Splitting the data into three sets

The learning algorithm chosen below will not be using examples from these two subsets to build the model validation and test sets.

5.2 Choosing the best learning algorithm by using validation set

5.2.1 Pipelining

The main reason for using pipelining is that training different models and testing on the validation sets was easy to perform. Here, at a time 4 different regression models were builded (trained) and tested. All the models were together placed inside the pipeline function such that they were trained and tested simultaneously. This prevents us training and testing manually the models one by one, which takes a lot of time. Below is the time difference in executing all the models.

	Total time taken to execute
With Pipeline	368.26367020606995 seconds (6.13 minutes)
Without Pipeline	1002.0538923740387 seconds (16.7 minutes)

The two important steps that were followed in the pipeline creation was

- **Data Preprocessing by using Standard Scaler**: Here all the features will be standardised by removing the mean and scaling to unit variance.
- **Applying Regression algorithms**: Here the regression algorithms mentioned below were invoked. The more we invoke the algorithm, the execution will increase. We can invoke maximum 5 algorithms at a time.

Different regression algorithms shown below was implemented by using the validation set. The validation set will be specifically used here to choose the best alogorithm (score).

- Linear Regression
- Random Forest Regressor
- Decision Tree Regressor
- Support Vector Regressor

Below shown is the implementation of pipelining with all the 4 models

Creating pipelines with different regression algorithms

```
# Importing necessary libraries
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn import svm
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score

# Creating the pipelines
pipeline_rf=Pipeline([('scalar1', StandardScaler()),
                      ('rf_regressor',RandomForestRegressor())])

pipeline_dt=Pipeline([('scalar2', StandardScaler()),
                      ('dt_regressor',DecisionTreeRegressor())])

pipeline_lr=Pipeline([('scalar3', StandardScaler()),
                      ('lr_regressor',LinearRegression())])

pipeline_svm=Pipeline([('scalar4', StandardScaler()),
                      ('svr_regressor',svm.SVR())])

# List to store all the model results
pipelines = [pipeline_rf, pipeline_dt, pipeline_lr, pipeline_svm]
```

Figure 5.3: Creating pipelines for different regressors

Building (Fit) the pipelines and predicting the validation set labels

```
start_time = time.time()

# Dictionary of pipelines and Regression types for ease of reference
pipe_dict = {0: 'Random Forest Regression', 1: 'Decision Tree Regressor', 2: 'Linear Regression', 3: 'Support Vector Regression'}
# Fit the pipelines
for pipe in pipelines:
    pipe.fit(X_train, y_train)

for i,model in enumerate(pipelines):
    pred = model.predict(X_val)
    print("{} Model Accuracy: {}".format(pipe_dict[i],r2_score(y_val, pred)* 100))

print("--- %s seconds ---" % (time.time() - start_time))   # Displaying the time in seconds
```

```
Random Forest Regression Model Accuracy: 84.81584796937892
Decision Tree Regressor Model Accuracy: 73.02462727685325
Linear Regression Model Accuracy: 56.095065680128066
Support Vector Regression Model Accuracy: 23.623095422425923
--- 379.20802187919617 seconds ---
```

Figure 5.4: Building the pipelines

As seen from the above accuracy results, Random Forest Regression and Decision Tree Regression have far better results. In fact, Random Forest Regression has the highest and better accuracy.

5.3 Underfitting

Usually a model has a low bias if it predicts well the labels of the training data. If the same model makes many mistakes on the training data, then the model is prone to have a high bias or the model is definitely under-fitting [4]. Here both the models such as Decision Tree Regression and Random Forest Regression will be used to check for underfitting condition.

5.3.1 Random forest regression - Predicting the labels of training data

The model will be trained on the training set, and will be tested on the training set itself as shown below.

```
from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor().fit(X_train, y_train)
pred = rfr.predict(X_train)
print(r2_score(y_train, pred)* 100)
```

```
97.85146144913742
```

Figure 5.5: Random Forest Regression

Below is the actual versus predicted price values comparison along with a graph.

```
df = pd.DataFrame({'Actual': y_train, 'Predicted': pred})
df
```

	Actual	Predicted
91177	13700	12979.650000
40705	10500	9566.580000
52135	6500	7231.250000
70690	2000	2087.480000
75994	5300	5589.413333
...	...	...
16346	10490	9601.850000
85585	12990	12459.875000
70331	850	1502.950000
46354	500	678.490000
66110	5006	5292.587917

Figure 5.6: Actual vs predicted train values

```
import matplotlib.pyplot as plt
df1 = df.head(35)
df1.plot(kind='bar',figsize=(10,5.5))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```

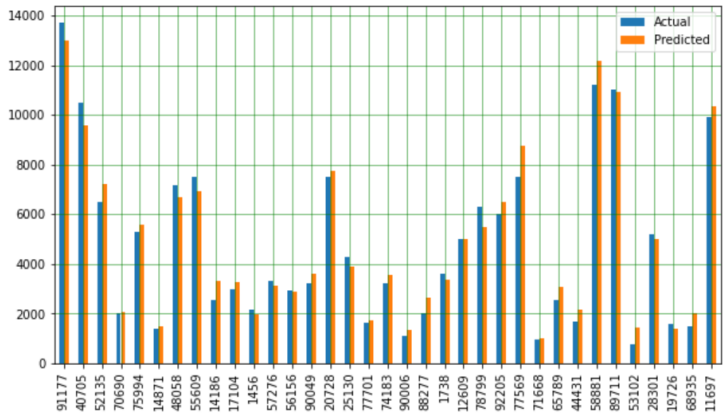

Figure 5.7: Actual vs predicted train values plot

Since the accuracy is almost 98%, it means that the model is not making many mistakes while predicting the training labels. Because there is not much of a difference between the actual and the predicted values as seen above. Hence the random forest model is not under fitting

5.3.2 Decision Tree Regression - Predicting the labels of training data

```
from sklearn.tree import DecisionTreeRegressor
dtr = DecisionTreeRegressor()
dtr.fit(X_train, y_train)
pred = dtr.predict(X_train)
print(r2_score(y_train, pred)* 100)
```

```
99.36216624335579
```

Figure 5.8: Decision Tree Regression

Below is the actual versus predicted train value comparison along with a plot

```
df = pd.DataFrame({'Actual': y_train, 'Predicted': pred})
df
```

	Actual	Predicted
91177	13700	13700.0
40705	10500	10500.0
52135	6500	6500.0
70690	2000	2000.0
75994	5300	5300.0
...	...	...
16346	10490	10490.0
85585	12990	12990.0
70331	850	850.0
46354	500	500.0
66110	5006	5006.0

Figure 5.9: Actual vs predicted values of decision tree regression

```
import matplotlib.pyplot as plt
df1 = df.head(35)
df1.plot(kind='bar',figsize=(10,5.5))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```

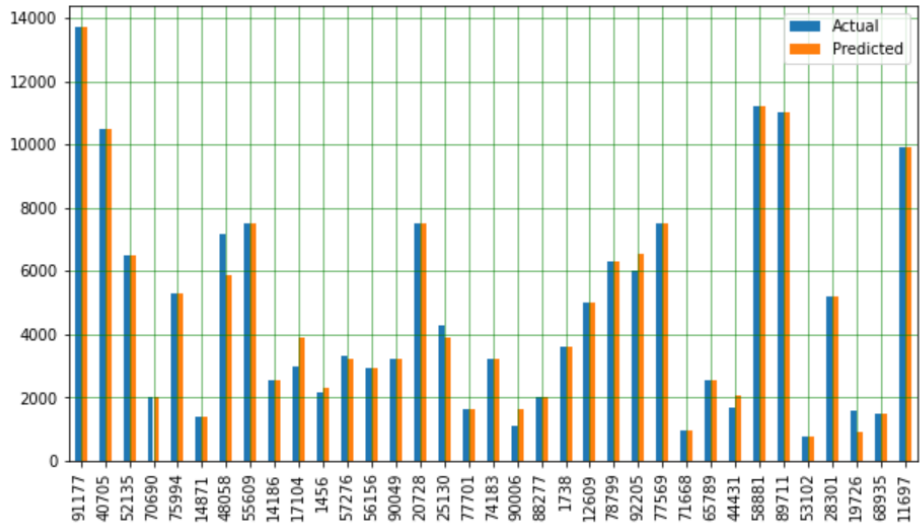

Figure 5.10: Actual vs predicted train values plot

The accuracy of the decision tree regression is 99%, it means that the model is not making any mistakes while predicting the training labels (Almost a perfect score). Because there hardly a difference between the actual and the predicted values as seen above. Hence the decision tree regression is not under fitting.

Overall as seen from the above results both the model did not underfit. Both the models were trained well on the training sets and both predicted the training labels very well.

Regression Algorithms	**Model Accuracy**
Random Forest	97.85146144913742
Decision Tree	99.36287739304052

5.4 Overfitting

In the case of overfitting the model that overfits predicts very well the training data but poorly the data from at least one of the two holdout sets [4].

5.4.1 Checking whether the model overfits or not

In this case let us train the model with the training set, and then try to predict the labels from one of the holdout sets (test and validation).

Random Forest Regression

```
from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor().fit(X_train, y_train)
pred = rfr.predict(X_test)
print(r2_score(y_test, pred)* 100)
```

```
84.65840335933866
```

Figure 5.11: Random Forest Regression

Decision Tree Regression

```
from sklearn.tree import DecisionTreeRegressor
dtr = DecisionTreeRegressor()
dtr.fit(X_train, y_train)
pred = dtr.predict(X_test)
print(r2_score(y_test, pred)* 100)
```

```
73.77756505301102
```

Figure 5.12: Decision Tree Regression

As seen above the results of decision tree is lower than the Random forest regression. Further hyper parameter tuning will be performed to increase the accuracy as well as choose the best parameters for random forest regression.

5.4.2 Hyper parameter tuning

Hyperparameter optimization or tuning is the problem of choosing a set of optimal hyperparameters for a learning algorithm. Usually a hyper parameter's value is used to control a learning algorithm. Just by choosing a good hyper parameter the overall performance of the model can be improved. Show below are two techniques to choose a good parameter for random forest regression.

Manually checking with every possible parameter

Here manually checking of the parameters will be done. Meaning the model will be tested on various parameters. Usually, the main parameters in the random forest regression that would be tuned below are:

- `n_estimators` : Indicates the number of trees in the forest. The higher the value (1000) the more chances of overfit [1].
- `max_depth`: The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples [6].
- `max_features`: The number of features to consider when looking for the best split [6].
- `min_samples_leaf`: The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression [6].

Other parameter are set to their default values. Below are the test results.

Random Forest Regression Models	n_estimators	max_depth	max_features	min_samples_leaf	Mean Absolute Error	Mean Squared Error	Accuracy	Time (s)
Model 1	100	5	10	2	1485.5237501232773	4474156.97292161	76.81429170476318	4.6896379224567344
Model 2	150	6	11	3	1388.5920299852435	3939498.416712833	79.58496725254449	7.769359588623047
Model 3	200	7	12	4	1320.321085333402	3616265.9053388224	81.26000341368321	12.595654726028442
Model 4	210	8	13	5	1265.814598626482	3379373.521901157	82.48761293498316	15.94746470451355
Model 5	220	9	14	6	1229.3222644591006	3225296.746048596	83.28606037471509	19.716230869293213
Model 6	230	10	15	7	1199.9552515372477	3098057.1830257615	83.94543361747694	23.898204803466797
Model 7	240	11	16	8	1180.8174800140378	3020335.8454557112	84.34819647808308	28.439677476882935
Model 8	250	12	17	9	1169.3047368943987	2971898.26229558	84.5992068204725	33.26568913459778
Model 9	260	13	18	10	1165.4628144585083	2967316.141040257	84.62295201480572	37.88083338737488
Model 10	270	14	18	11	1163.9861317221678	2964894.919932388	84.63549912181718	40.37581658363342
Model 11	280	15	18	12	1165.9423374829814	2981429.490132762	84.54981466242668	42.32461333274841

Figure 5.13: Parameters of random forest regression (Manual hyperparameter tuning)

As seen from the above results, at one point the accuracy of the model started to be same and the gradually decrease. This was the point where the model was stopped from further predicting the test labels. So by performing manual tuning of hyper parameters the best parameters are n_estimators = 270, max_dept = 14, max_features = 18, min_samples_leaf = 11. The best performance metrics are as shown below:

Random forest reg model name	n_estimators	max_depth	max_features	min_samples_leaf	MAE	MSE	Accuracy	Time (s)
Model 10	270	14	18	11	1163.9861317221678	2964894.919932388	84.63549912181718	40.37581658363342

Figure 5.14: The model with the best parameters

Using Grid search CV to choose the best parameters

The hyper parameter tuning is performed using Grid Search cv which is an exhaustive search over specified parameter values for an estimator. With the help of grid search cv we can tune and find the best estimator (parameter) for the model. As we saw earlier, random forest gave better results when predicting the labels from the holdout sets, we will perform hyper parameter tuning on random forest.

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

start_time = time.time()
rfr = RandomForestRegressor()
# Parameter of Random Forest Regression
parameters = {
                    "n_estimators":[5,50,150,250,350],
                    "max_depth":[2,4,8,16,None],
                    "max_features":[10, 11, 12, 13, 14],
                    "min_samples_leaf": [2, 3, 4, 5, 6]
             }
cv = GridSearchCV(rfr,parameters,cv=2)
cv.fit(X_train, y_train.values.ravel())
print("--- %s seconds ---" % (time.time() - start_time))

def display(results):
    print(f'Best parameters are: {results.best_params_}')
    print("\n")
    mean_score = results.cv_results_['mean_test_score']
    std_score = results.cv_results_['std_test_score']
    params = results.cv_results_['params']
    for mean,std,params in zip(mean_score,std_score,params):
        print(f'{round(mean,3)} + or -{round(std,3)} for the {params}')
display(cv)
```

Figure 5.15: Grid Search CV

One of the main disadvantages of grid search cv is longer execution time. The more the parameter is passed the longer it takes to execute. Below is the results returned by grid search cv along with the best parameters. The whole tuning of parameter using grid search cv took almost 2-3 hours. Initially there were more parameters but it took forever to execute and lastly the notebook crashed. So the parameters were then reduced.

```
--- 6988.722146034241 seconds ---
Best parameters are: {'max_depth': 16, 'max_features': 10, 'min_samples_leaf': 2, 'n_estimators': 350}


0.55 + or -0.009 for the {'max_depth': 2, 'max_features': 10, 'min_samples_leaf': 2, 'n_estimators': 5}
0.572 + or -0.001 for the {'max_depth': 2, 'max_features': 10, 'min_samples_leaf': 2, 'n_estimators': 50}
0.572 + or -0.0 for the {'max_depth': 2, 'max_features': 10, 'min_samples_leaf': 2, 'n_estimators': 150}
0.572 + or -0.001 for the {'max_depth': 2, 'max_features': 10, 'min_samples_leaf': 2, 'n_estimators': 250}
0.57 + or -0.0 for the {'max_depth': 2, 'max_features': 10, 'min_samples_leaf': 2, 'n_estimators': 350}
0.56 + or -0.003 for the {'max_depth': 2, 'max_features': 10, 'min_samples_leaf': 3, 'n_estimators': 5}
0.569 + or -0.003 for the {'max_depth': 2, 'max_features': 10, 'min_samples_leaf': 3, 'n_estimators': 50}
0.571 + or -0.001 for the {'max_depth': 2, 'max_features': 10, 'min_samples_leaf': 3, 'n_estimators': 150}
0.571 + or -0.002 for the {'max_depth': 2, 'max_features': 10, 'min_samples_leaf': 3, 'n_estimators': 250}
0.573 + or -0.001 for the {'max_depth': 2, 'max_features': 10, 'min_samples_leaf': 3, 'n_estimators': 350}
0.532 + or -0.011 for the {'max_depth': 2, 'max_features': 10, 'min_samples_leaf': 4, 'n_estimators': 5}
0.57 + or -0.004 for the {'max_depth': 2, 'max_features': 10, 'min_samples_leaf': 4, 'n_estimators': 50}
0.573 + or -0.001 for the {'max_depth': 2, 'max_features': 10, 'min_samples_leaf': 4, 'n_estimators': 150}
0.571 + or -0.001 for the {'max_depth': 2, 'max_features': 10, 'min_samples_leaf': 4, 'n_estimators': 250}
0.572 + or -0.0 for the {'max_depth': 2, 'max_features': 10, 'min_samples_leaf': 4, 'n_estimators': 350}
0.538 + or -0.03 for the {'max_depth': 2, 'max_features': 10, 'min_samples_leaf': 5, 'n_estimators': 5}
0.573 + or -0.001 for the {'max_depth': 2, 'max_features': 10, 'min_samples_leaf': 5, 'n_estimators': 50}
0.577 + or -0.002 for the {'max_depth': 2, 'max_features': 10, 'min_samples_leaf': 5, 'n_estimators': 150}
0.571 + or -0.001 for the {'max_depth': 2, 'max_features': 10, 'min_samples_leaf': 5, 'n_estimators': 250}
0.572 + or -0.0 for the {'max_depth': 2, 'max_features': 10, 'min_samples_leaf': 5, 'n_estimators': 350}
0.546 + or -0.007 for the {'max_depth': 2, 'max_features': 10, 'min_samples_leaf': 6, 'n_estimators': 5}
0.566 + or -0.002 for the {'max_depth': 2, 'max_features': 10, 'min_samples_leaf': 6, 'n_estimators': 50}
0.572 + or -0.003 for the {'max_depth': 2, 'max_features': 10, 'min_samples_leaf': 6, 'n_estimators': 150}
0.57 + or -0.0 for the {'max_depth': 2, 'max_features': 10, 'min_samples_leaf': 6, 'n_estimators': 250}
0.571 + or -0.0 for the {'max_depth': 2, 'max_features': 10, 'min_samples_leaf': 6, 'n_estimators': 350}
0.549 + or -0.009 for the {'max_depth': 2, 'max_features': 11, 'min_samples_leaf': 2, 'n_estimators': 5}
0.572 + or -0.002 for the {'max_depth': 2, 'max_features': 11, 'min_samples_leaf': 2, 'n_estimators': 50}
```

Figure 5.16: Best parameters chosen by grid search cv

So by tuning of hyper parameters using grid search the best parameters are n_estimators = 350, max_dept = 16, max_features = 10, min_samples_leaf = 2. The comparison of the performance of the model both by manually checking and using grid search cv for tuning parameters is as shown below.

Random forest reg	n_estimators	max_depth	max_features	min_samples_leaf	MAE	MSE	Accuracy	Time (s)
Manual tuning	270	14	18	11	1163.9861317221678	2964894.919932388	84.63549912181718	40.37581658363342
Grid Search CV	350	16	10	2	1108.0748354948025	2697919.915430242	86.01900100026474	39.78496479988098

Figure 5.17: Performance comparison between manual tuning and grid search cv

As seen from the above results the hyper parameter tuning using grid search cv gave very good results. There is an increase in accuracy of 2% (86%) which is a very big difference and good sign of improvement. Hence the parameter given by the grid search cv would be used for further process.

5.4.3 Regularization

Further the concept of regularization was implemented for testing purpose to see if there is any new findings. The two most widely used types of regularization are called L1 and L2 regularization. L1 is knows as Lasso and L2 is known as Ridge regression.

L1 Regularization (Lasso)

Below shown is the implementation of lasso regularization along with the comparison of coefficients.

```
from sklearn.linear_model import Lasso
rr = Lasso(alpha= 10000)
rr.fit(X_train, y_train)
pred = rr.predict(X_test)
print(r2_score(y_test, pred) * 100)
```

Figure 5.18: Lasso Regression

L1 - Regularization	**Accuracy**	**Time taken to execute(s)**
Lasso Regression - alpha = 10000	-0.004225239923694	0.01874542236328125
Lasso Regression - alpha = 1000	40.188960421043895	0.015836477279663086
Lasso Regression - alpha = 100	56.53602625199379	0.024311065673828125
Lasso Regression - alpha = 10	56.872121216214076	0.027382612228393555
Lasso Regression - alpha = 1	56.88514458882089	0.028135061264038086
Lasso Regression - alpha = 0.1	56.88610417645656	0.031087160110473633

L2 Regularization (Ridge)

Below shown is the implementation of ridge regression

```
from sklearn.linear_model import Ridge
rr = Ridge(alpha= 10000)
rr.fit(X_train, y_train)
pred = rr.predict(X_test)
print(r2_score(y_test, pred) * 100)
```

Figure 5.19: Ridge Regression

The alpha value is varied from 1000 to 0.1. The results can be found below:

L2 - Regularization	**Accuracy**	**Time taken to execute(s)**
Ridge Regression - alpha = 10000	56.005916173458516	0.021957874298095703
Ridge Regression - alpha = 1000	56.862126442879735	0.024925947189331055
Ridge Regression - alpha = 100	56.88466608809992	0.017815828323364258
Ridge Regression - alpha = 10	56.88605350312452	0.02006673812866211
Ridge Regression - alpha = 1	56.88618326663119	0.018297672271728516
Ridge Regression - alpha = 0.1	56.88619615287895	0.021454572677612305

As seen from the above results there were no major improvement in varying the hyper parameter (alpha).

5.4.4 MSE test vs MSE Train

This is one of the test used to check overfitting. The mean squared error (MSE) for the training, and, separately, for the test data is computed. If the MSE of the model on the test data is substantially higher than the MSE obtained on the training data, this is a sign of overfitting [4].

Training MSE

```
from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor().fit(X_train, y_train)
pred = rfr.predict(X_train)

print("Mean Absolute Error is :", mean_absolute_error(y_train, pred))
print(" - - - - - - - - - - - - - - - - - - - - - - - - ")
print("Mean Squared Error is :", mean_squared_error(y_train, pred))
print(" - - - - - - - - - - - - - - - - - - - - - - - - ")
print("The R2 square value of Random Forest Regression is :",rfr.score(X_train, y_train)* 100)
```

```
Mean Absolute Error is : 426.01334188659473
 - - - - - - - - - - - - - - - - - - - - - - - -
Mean Squared Error is : 410579.12151505775
 - - - - - - - - - - - - - - - - - - - - - - - -
The R2 square value of Random Forest Regression is : 97.8539998222856
```

Figure 5.20: Ridge Regression

Testing MSE

```
from sklearn.ensemble import RandomForestRegressor
clf = RandomForestRegressor()
clf.fit(X_test, y_test)
pred = clf.predict(X_test)
```

```
print("Mean Absolute Error is :", mean_absolute_error(y_test, pred))
print(" - - - - - - - - - - - - - - - - - - - - - - - - ")
print("Mean Squared Error is :", mean_squared_error(y_test, pred))
print(" - - - - - - - - - - - - - - - - - - - - - - - - ")
print("The R2 square value of Random Forest Regression is :",clf.score(X_test, y_test)* 100)
```

```
Mean Absolute Error is : 453.6230156679866
 - - - - - - - - - - - - - - - - - - - - - - - -
Mean Squared Error is : 457486.05584306625
 - - - - - - - - - - - - - - - - - - - - - - - -
The R2 square value of Random Forest Regression is : 97.62924316153588
```

Figure 5.21: Ridge Regression

As seen from above both the MSE and MAE of training set is not significantly higher than the training set. Also, to support let's perform T-test to check the difference between the MSE values of training and testing set.

T - test

A t-test is a type of inferential statistic used to determine if there is a significant difference between the means of two groups, which may be related in certain features [7].

```
## Import the packages
import numpy as np
from scipy import stats

## Define 2 random distributions
#Sample Size
N = 10
#Gaussian distributed data with mean = 2 and var = 1
a = np.array([409055.7933, 411257.5183, 410579.1215])
#Gaussian distributed data with with mean = 0 and var = 1
b = np.array([455373.9379, 455426.4436, 455794.1234])

## Calculate the Standard Deviation
#Calculate the variance to get the standard deviation
#For unbiased max likelihood estimate we have to divide the var by N
    -1, and therefore the parameter ddof = 1
var_a = a.var(ddof=1)
var_b = b.var(ddof=1)

#std deviation
s = np.sqrt((var_a + var_b)/2)

## Calculate the t-statistics
t = (a.mean() - b.mean())/(s*np.sqrt(2/N))

## Compare with the critical t-value
#Degrees of freedom
```

```
df = 2*N - 2

#p-value after comparison with the t
p = 1 - stats.t.cdf(t,df=df)

print("t = " + str(t))
print("p = " + str(2*p/100))
```

Code Listing 5.1: Performing stastistical T-test

Here as seen above, the model was tested and trainied multiple times (3), hence all the three values from each set was recorded.

```
t = -124.32337778426184
p = 0.02
```

Figure 5.22: Results of T-test

As seen from the above results the t-test and p value are significantly low. This indicates that the model is not overfitting.

5.5 Model Performance Assessment

The model shown above is trained and tested well. The test set contains the examples that the learning algorithm has never seen before, so if our model performs well on predicting the labels of the examples from the test set, we can say that our model generalizes well or, simply, that it's good. All the parameters were chosen from the grid search cv results.

```
from sklearn.ensemble import RandomForestRegressor
start_time = time.time()
rfr = RandomForestRegressor(max_depth = 16, max_features = 10, min_samples_leaf = 2, n_estimators = 350).fit(X_train, y_train)
pred = rfr.predict(X_test)
print(r2_score(y_test, pred)* 100)
print("--- %s seconds ---" % (time.time() - start_time))
```

```
86.020619788918
--- 39.57201290130615 seconds ---
```

Figure 5.23: Random Forest Regression

The below shown are the performance metrics of random forest regression

- R2 score: It is regression score function. The best or the highest score value is 1.0 (100%), similarly the lowest score can be a negative value which indicates the model can be worst. Basically, the higher the value the better the model.

- MSE: Mean squared error is a risk metric corresponding to the expected value of the squared (quadratic) error or loss [8]. Basically it is

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

  If $\hat{Y}_i$ is the predicted value of the i-th sample, and Y_i is the corresponding true value, then the mean squared error (MSE) estimated over $n_{samples}$ [8]

- MAE: Mean absolute error is the risk metric corresponding to the expected value of the absolute loss [9].

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|Y_i - \hat{Y}_i|^2.$$

  If $\hat{Y}_i$ is the predicted value of the i-th sample, and Y_i is the corresponding true value, then the mean absolute error (MSE) estimated over $n_{samples}$ [9]

The summary of the performance metrics of the random forest regression is shown below

	Accuracy	**MSE**	**MAE**	**Time (seconds)**
Random forest Regressor	86.02061	2697919.9154302	1108.0748354	39.57201

Below shown is the comparison between actual and the predicted values along with the plot.

```
df = pd.DataFrame({'Actual': y_test, 'Predicted': pred})
df.head(10)
```

	Actual	Predicted
69709	3500	3768.383020
94102	6990	7164.811422
75249	5350	10014.683847
20486	3499	3465.002190
1100	15900	10315.966681
28975	3799	3031.635670
39618	8450	8553.548221
9951	2800	2587.234036
5630	790	1657.373327
37967	900	1516.382849

Figure 5.24: Actual vs predicted values

```
import matplotlib.pyplot as plt
df1 = df.head(35)
df1.plot(kind='bar',figsize=(10,5.5))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```

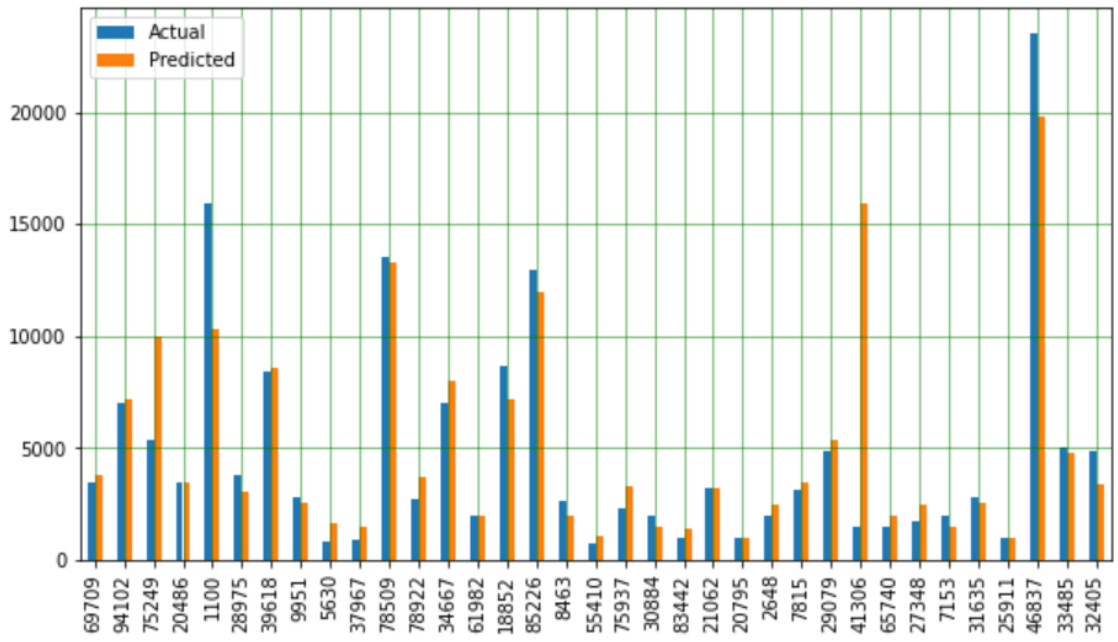

Figure 5.25: Actual vs predicted values plot

Chapter 6

Communicate Results

In this chapter, all the findings, necessary steps to enhance the model along with the technical documentation, tool and users of the application will be discussed

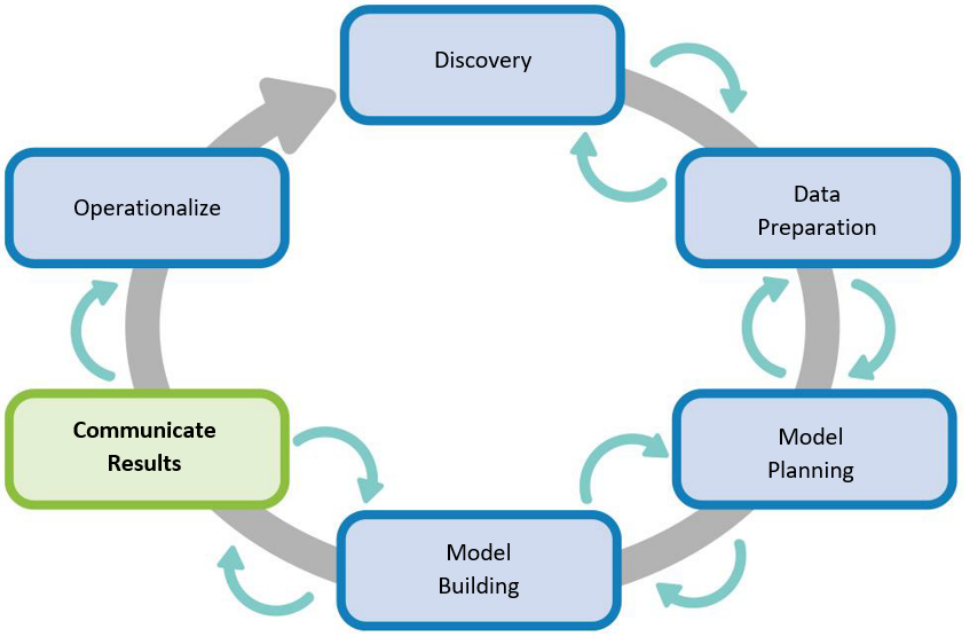

Figure 6.1: Communicate Results

6.1 Users

The potential users of the system are of two types: buyers/sellers and admin as shown in below figure. As you all know there are a lot of people in this world who want to buy and sell cars spontaneously. Hence the user rank would be very high because both technical people such as the dealers and the non-technical such as the buyers can use this software in ease.

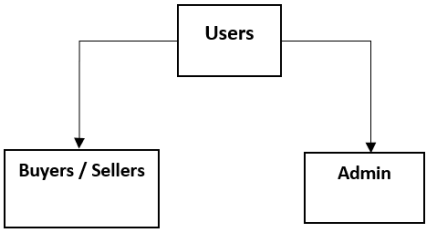

Figure 6.2: Users of the application

6.1.1 Buyer

- There are heaps of people keen on the trade-in vehicle showcase at certain focuses in their life since they needed to sell their vehicle or purchase a trade-in vehicle. Right now, a major corner to pay excessively or sell not as much as it's estimated worth.

- The potential value of this system would be great because when a buyer needs to buy a used car say a Mercedes Benz A -Class his/her main intention would be great features and cheap price obviously. Without having to know how much to pay, they can use this software to get an accurate estimation of the price of the car and then decide what to do further. Here, they can use this software to get the price of the used car by feeding the features of the attributes of the car.

- Giving feedback if the buyer is not satisfied with the price or suggesting any improvements

6.1.2 Seller

- These users are the ones who might use this software very much. They are one of the largest target groups that would depend on the results of the system.
- Whenever used vehicle vendors better comprehend what makes a vehicle alluring, what the significant highlights are for a trade-in vehicle, at that point they may consider this information and over superior help.
- In the seller's case let's picture that the seller has this amazing Volkswagen Golf GT 2017 model, and they are not sure about the price to sell, they are looking for a price which is probably not too less and more profit. Rather than selling the car for vague numbers they can use the system accurate price predictions to solve this issue.
- Giving feedback if the buyer is not satisfied with the price or suggesting any improvements

6.1.3 Admin

- The admin is the one who mainly implemented, builded and developed the model. The software designing and deploying was done by the admin. The admin in this case is Tanu Nanda Prabhu.
- Regularly updating the website by adding more features. Managing and deleting the users feedbacks. Improving the system according to users suggestions.

6.2 Technical Documentation

In this section, the technical documentation of the proposed application is provided, which includes a list of programming languages, reused algorithms, software tools and environment.

6.2.1 List of Programming Languages

The languages into three parts as programming, web-designing languages and Django's template language which can be found below

Programming Language

- Python
  - Since we all know Python is one of the most commonly used programming language throughout the world. The entire backend of the application was coded in Python.
  - This includes data processing, model building and evaluation and many more were code in python

Source	https://docs.python.org/3/
Version used	3.7
Filename examples	Data Preparation, Model Planning, Discovery

Web design language

- **HTML**
  - HTML as a markup language was very useful in building the front-end of the application.

- All the forms such as the name, power and the drop downs such as vehicle type, fuel type including the submit button were designed using HTML
- The HTML pages is located in the templates folder.

Source	https://www.w3schools.com/html/
Version used	5 (HTML5)
Filename examples	home.html, messages.html

- **CSS**
  - Although only using HTML doesn't make the application look colorful, different styles and layouts were incorporated for the application. Hence, the CSS (Cascading Style Sheets) was used to serve the purpose.
  - CSS is specifically used for implementing the card view for displaying the information, grid view for the layouts, external fonts for the text and many more. The CSS files can be found in the static/css folder.

Source	https://www.w3.org/Style/CSS/
Version used	4 (CSS4)
Filename examples	home.html, messages.html

- **Bootstrap**
  - Last, Bootstrap was used for making the application responsive. Every single page in the application is responsive meaning every page can and will perfectly fit into any screen irrespective of their aspect ratios.
  - There is no separate file written for the bootstrap code, we must place the bootstrap code inside the HTML code. To make the page responsive all that was done was using external bootstrap libraries and placed all the

HTML code inside the div tag. By doing so automatically, the pages would be responsive.

Source	https://getbootstrap.com/
Version used	4.0.1
Filename examples	home.html, messages.html

- **Django Template Language**
  - The Django template language provides a platform to render python variables onto the HTML pages. Suppose in this case, when all the values in the forms were entered the hardest part was displaying it in the dashboard (HTML) page. With the help of Django template language we could do this in one step by using {{% *forms.name* %%}}. The code was placed in the HTML page and then the forms values were rendered on the portal page.

Source	https://djangoproject.com/template-language-intro
Version used	3.0
Filename examples	home.html, messages.hmtl

6.2.2 List of reused algorithms

Below are the reused algorithm and program that was used in building the application. Also exact sources are provided.

CSS Card View

The card view was incorporated to display beautiful card layout for the front-end.

Source: https://www.w3schools.com/w3css/w3css_cards.asp

T-test

The stastistical t-test was performed to determine the difference between two mean groups, in this case difference between training MSE and testing MSE of the random forest regression algorithm was determined form the t-test.

Source: https://towardsdatascience.com/inferential-statistics-series-t-test-using-numpy-2718f8f9bf2f

6.3 List of software tools and environment

- **Google Colab**: Google Colab environment was used as a primary coding platform. All the data exploration, model selection and model building were done on Google Colab. One of the main reasons of using Google Colab was documenting the code was far easier. Because of the markdown language feature, all the code could be easily documented.

  Source: Google Colab

- **Anaconda - Jupyter Notebook** The main reason for installing the anoconda - jupyter notebook was that because of its execution speed. Because the Google Colab was connected to the hosted runtime which took more time to execute the cells. Whereas the Jupyter notebook connects to local runtime (system) execution time was faster than Colab. Basically, all the training, testing the model, tuning the hyper-parameters were done on anoconda's jupyter notebook environment.

  Source: Anoconda - Jupyter Notebook

- **Visual Studio Code**: This IDE was used to develop the front end of the application. After the model was builded and tested, the last step was to develop a website (front-end). Visual studio code IDE was very helpful even in

integration with the GitHub. Because with the help of its commands the codes could be easily push and committed to GitHub. Also, lastly with the help of its command terminal all the external python libraries was easily installed inside visual studio code.

Source: Visual Studio Code

- **GitHub**: All the code of the application was uploaded on GitHub. In the future, if we or anyone wants to improve the application, then they can easily get the source code. This is one of the main reasons of the code being an open source, such that this would be helpful for others. The main reason that the GitHub was used to store the code and commit all the changes. Also, deploying the code on Heroku cloud was very easy, it directly pulled the code from GitHub.

  Source https://desktop.github.com/

- **Django:** Django is a python based web framework and also provides MVT (Model View Template) architecture. Since one of the project component involved in building a cloud based web application, so Django framework was to build a website. Django forms made the effort less while creating name, model, kilometers and more forms. The main model was placed inside the views folder and the front-end code was placed inside the templates folder.

  Source https://www.djangoproject.com or pip install Django

- **Heroku** Heroku is a platform as a service (Paas), that allows us to build and develop the application and then deploy in the cloud [10]. This service was free to use (no credit card details were asked during account creation). With, the help of this service the application was hosted both on the cloud. Since for using heroku, the application got default SSL certificate from DigiCert, so that the communication or the data sent through the application is secure. Deployment

through GitHub repository was very helpful. Heroku deployment logs was very useful for debugging purpose.

Source https://www.heroku.com/

6.4 List of important libraries

Some of the important libraries that were used in developing the application was.

- **Data Preparation**:
  - Pandas, NumPy for storing the data into dataframe, and performing mathematical computations on the dataframe.
  - time, warning for calculating execution time and filtering warnings.
  - selectKBoost and Chi2 from sklearn for selecting the best features and performing Chi-squared test on the features.
  - matplotlib and seaborn for visualizing the findings
- **Model Selection**:
  - model selection from scikit learn for spliting the dataset into three sets.
  - preprocesing from sklearn to standardize the values.
  - ensemble from scikit learn for implenting random forest regression
  - tree from scikit learn for implementing decision tree regression
  - linear model from scikit learn for implementing linear regression
  - svm from scikit learn for implementing support vectore regression
  - ridge and lasso from linear model of scikit learn for performing regularization

- pipeline and gridsearch cv for performing pipelining and tuning the hyper parameters

- **Deployment**

  - pickle to save the model and the load anytime and anywhere

    Also below is the link to the requirements file that has all the necessary installations libraries and dependencies that was installed during the implementation requirements file. Also the model file can be located here

6.5 Team Members with role of each student

6.5.1 Team roles

The data science project **"Used Car Price Prediction System"** was solely done by me (Tanu Nanda Prabhu). No teammates were involved in developing, building and deploying the project. Basically, all the phases as shown in the below data-analytic life cycle were implemented by me.

- **Data discovery**

  - Understanding and drafting the business problem statement.
  - Framing the issue as a challenge to analytics.
  - Assessing the tools, resources and selection or checking the availability of data.

- **Data Preparation**

  - Storing the data in the external hard disk drive
  - Extracting, transforming, loading and transforming (ETLT) the data

  - Data Exploration and conditioning which involves removing outliers, missing and duplicate values, etc.
  - Finally summarizing and visualizing the data.

- **Model Planning**
  - Variable selection
  - Model selection choosing out of Regression, Classification, Association Rules and Text/Image/Video Analysis.

- **Model Building**
  - Building the training and testing datasets.
  - Training the selected model.
  - Testing the accuracy of the model, this phase includes implementing different models and choosing the best one with the highest score.
  - Visualizing the results and reporting the metrics

- **Communicating results**
  - Communicating the results which include giving a presentation which include all the necessary findings, future enhancements, etc and making the code as open source.
  - Documenting code (GitHub) along with technical specifications and writing project report

- **Operationalization**
  - Providing the code and technical documentation
  - Deployment of the code on the cloud and hosting it on a web service platform.

6.5.2 Introduction about me

Figure 6.3: Tanu Nanda Prabhu

My name is Tanu Nanda Prabhu, I am a graduate student pursuing master's in computer science (Course-Based) at the University of Regina. My expertise in the field of computer science is as shown below:

- **Programming languages**: C (data structures), Python (Intermediate level) and Kotlin (Intermediate Level)
- **Web Programming**: HTML, CSS, JavaScript, Bootstrap
- **Web Frameworks**: Django
- Also, I am very well in using Git and GitHub and I am an author on Medium platform. Mainly I write articles, tutorials on Python, Kotlin, Data Science, and Web designing.
  - GitHub profile link: https://github.com/Tanu-N-Prabhu
  - Medium profile: https://medium.com/@tanunprabhu95
- Contact Details- Email: tanuprabhu96@gmail.com

6.6 Solution Overview

Since this category of the data set falls under supervised machine learning algorithm, the regression estimator was applied on this dataset. As seen above out of the four regression algorithms namely: Linear Regression, Decision Tree Regression, Random Forest Regression and Support Vector Regression, the Random Forest Regression gave better results in terms of prediction accuracy, mean squared error and mean absolute error. More details of the performance can be found in the above chapter 5 under model performance assessment. Before applying the regression algorithms, the correlation between two variables (independent and dependent variables) were detected and then the dependent variables which are the price of the used car was later predicted. Basically, the categorization of the variables which is the dependent are the price and the independent variables are the features of the used car except the price was done in this stage. And then the regression algorithms were applied until a good accuracy score which was presentable. In this case, Random Forest Regression gave an accuracy score of around 86.019%.

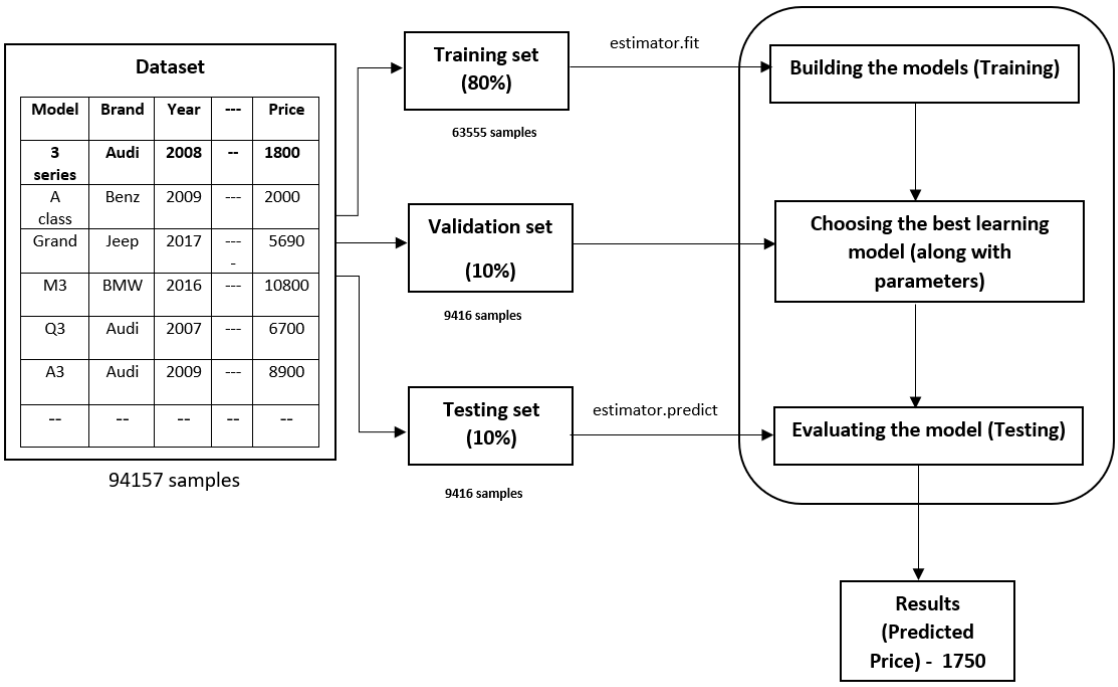

Figure 6.4: Block diagram of model

Shown above is the block diagram of which depicts splitting of datasets into three parts. Finally applying the best machine learning algorithm to get the predicted outcome. The expected outcome is the price of the used car. This can be achieved by giving the features of the car as the input variables to the machine learning algorithm (Random Forest Regression), and then the model will predict the price. For example, consider the user who wants to sell a car BMW M3, now he can give the features of the car such as Model, Brand, Power, Gearbox, fuel type and then predict the car. With the help of this software, the seller can know the price of the car in advance without having to spend much time on guessing the price.

Also, this application is deployed on the cloud platform named Heroku, a beautiful and responsive web-based software for the front end is created. Here Django was used as the web framework. Also, since this project was done solely, one of the components project was to **Fully developed and Operational Cloud-based app or website**. Here, the main focus was building a fully functional cloud-based website as a result.

More details of the web application and details about the cloud can be found in the next section.

6.6.1 Outcome

The expected outcome of the project is developing a web-based application that could predict the price of the used car given the features of the car. As discussed earlier the model was deployed on the cloud named heroku. A simple dashboard UI was designed which can be seen below. The application has two types of input options: Number field, where the user has to enter the desired number of their choice with respect to their used car and drop down menu where the user has to select a provided option given in the menu. Later when the user clicks on the submit button the model predicts the price based on the values given by the user and then displays underneath the submit button.

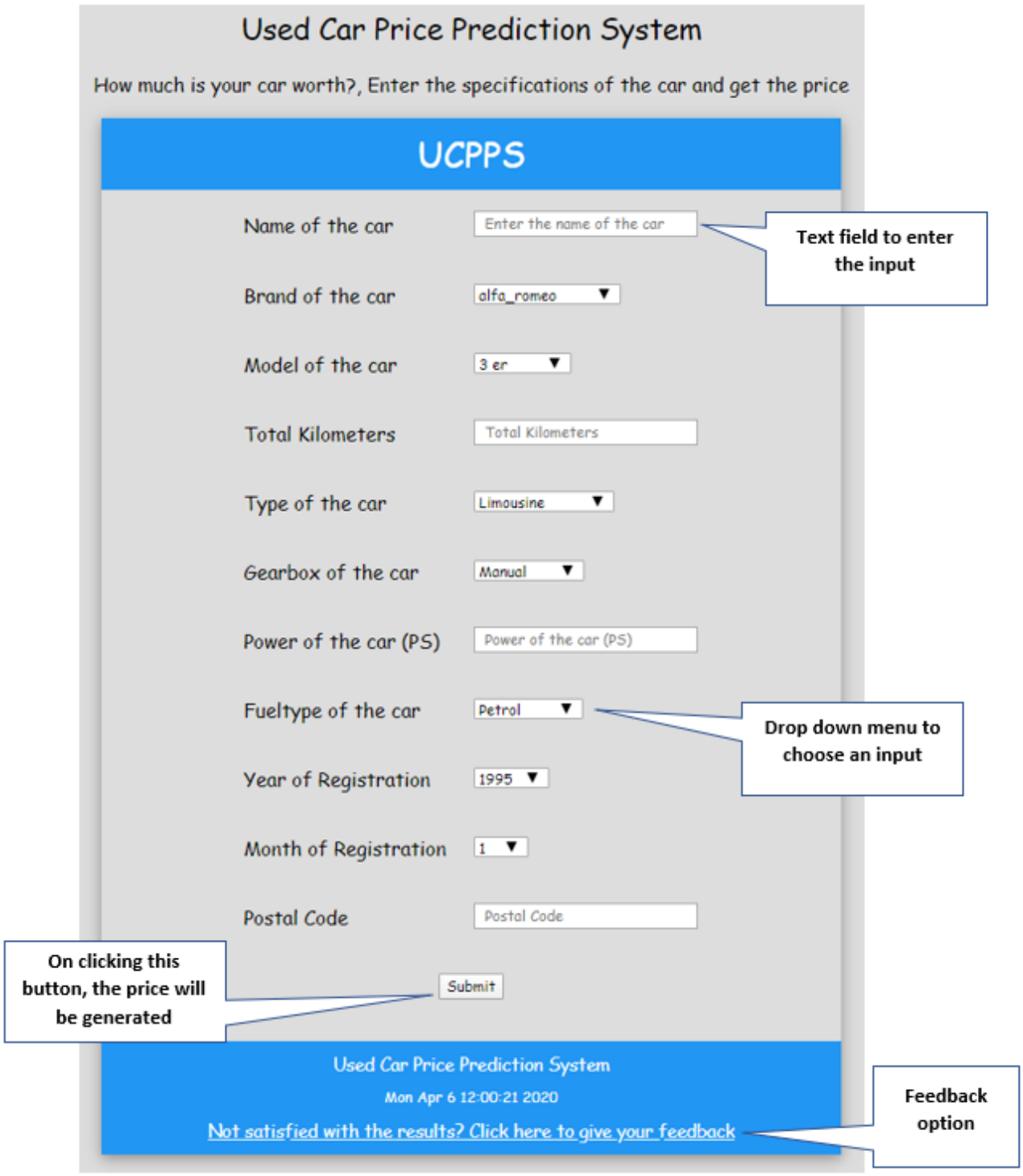

Figure 6.5: Used Car Price Prediction Application

6.7 Time line

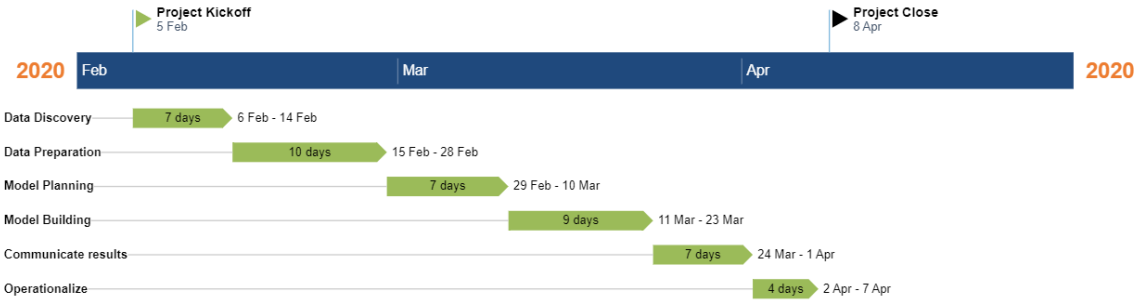

Figure 6.6: Time line of the project

The above is the timeline of the project. Throughout the project the data analytics phases were followed. Every phase of the project was met and completed with the allocated time. One of the hardest task was data preparation phase because initially all lot of techniques were applied on the data to condition, explore and translate the data. Later the same data was visualised to find out the patterns. Secondly, more time was utilized for model building phase too. Since initially many algorithms were applied on the data set and the best one was chosen. And on the selected one of the tuning of hyper-parameters was performed to choose the best estimators. In this phase regularization along with t-test was also performed to avoid overfitting. Basically, through the project a lot of trial and error methods and techniques were performed and out of those the best one was selected and documented as shown above in Model building section.

Chapter 7

Operationalize

In this chapter more details about the deployment process will be give along with elucidating the prominent features and working of the application.

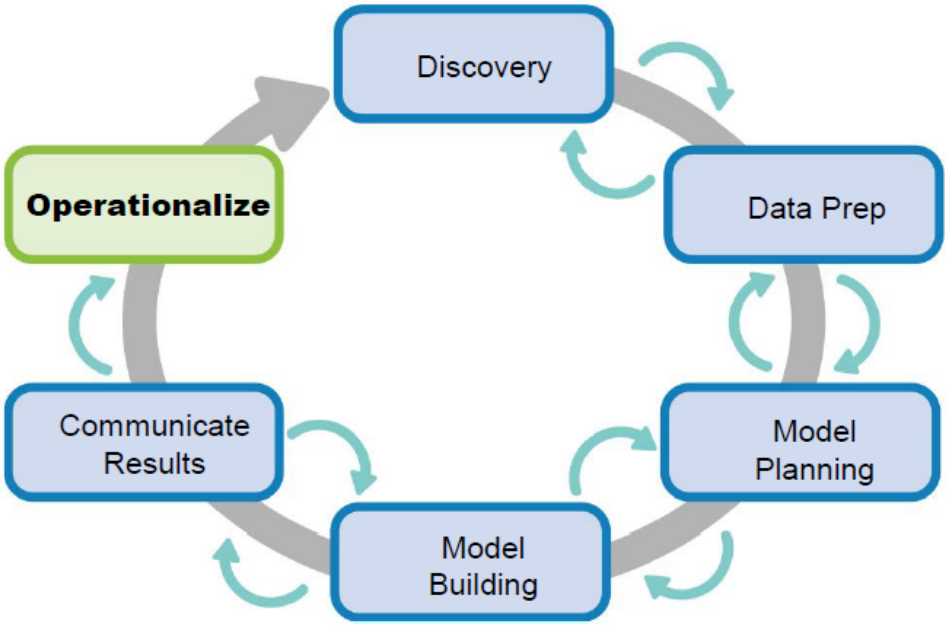

Figure 7.1: Operationalize

7.1 Working of the application

The working of the application is pretty trivial. First, the user must enter all the desired feature values of their used car as inputs to the form field as shown below:

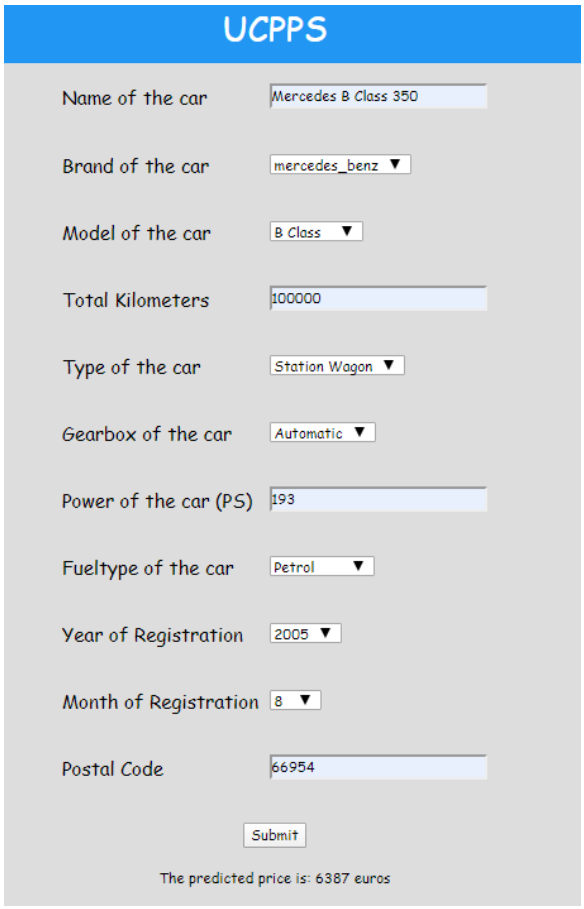

Figure 7.2: User entering input

As seen above the predicted price for the Mercedes Benz B class model is about 6387 euros where the original price was 6500 shown below. Just for comparison from this we can see that the prediction rate of the model is quite accurate.

```
df.iloc[20486]

dateCrawled                                  2016-03-14 22:49:58
name                 Mercedes_Benz_B_KlasseTurbo_SHZ_Klima_193PS
seller                                                     privat
offerType                                                Angebot
price                                                       6500
abtest                                                   control
vehicleType                                                  bus
yearOfRegistration                                          2005
gearbox                                                  manuell
powerPS                                                      193
model                                                  b_klasse
kilometer                                                100000
monthOfRegistration                                           8
fuelType                                                 benzin
brand                                              mercedes_benz
notRepairedDamage                                           nein
dateCreated                                 2016-03-14 00:00:00
nrOfPictures                                                  0
postalCode                                                66954
lastSeen                                    2016-03-24 08:17:29
Name: 20486, dtype: object
```

Figure 7.3: Original Price of the car

7.1.1 Features of the application

- **Secure link**: Firstly the application is secure. Heroku Cloud application platform by default provides SSL certificate, issued by DigiCert. The application uses HTTPS instead of HTTP, so data sent through our application will be safe. Below figure indicates the secure connection of our application and the certificate information is also depicted.

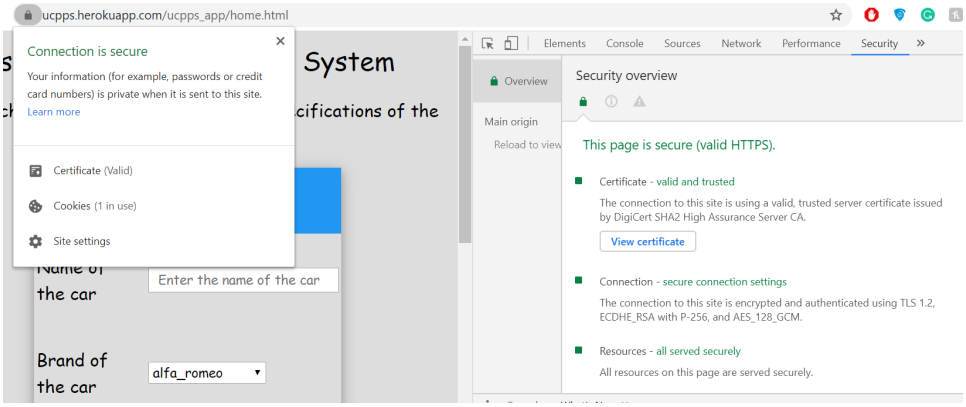

Figure 7.4: HTTPS secure connection of the application

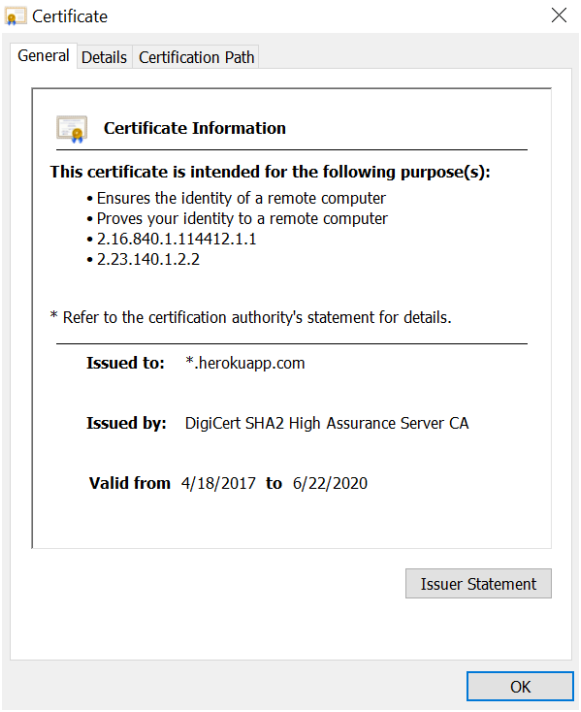

Figure 7.5: The certificate information of the application provided by DigiCert

- **The application is very roboust**:
  - **Incomplete form submission is not allowed**: All the values in the forms (textfields, dropdown) must be entered. Skipping the forms will lead to a warning (Please Fill this field). A user cannot submit a empty form or a incomplete form. To allow smooth performance of the application all the values has to entered only then the form can be submitted. Show below are the use cases for the both conditions.
  - **Character values cannot be entered in the interger fields**: In some of the forms such as kilometer the user has to enter the numeric values, other than numeric the user cannot enter any values. This is the speciality of Django's integer field method which only accepts integers. Since the model accepts only numeric values this feature was very helpful.

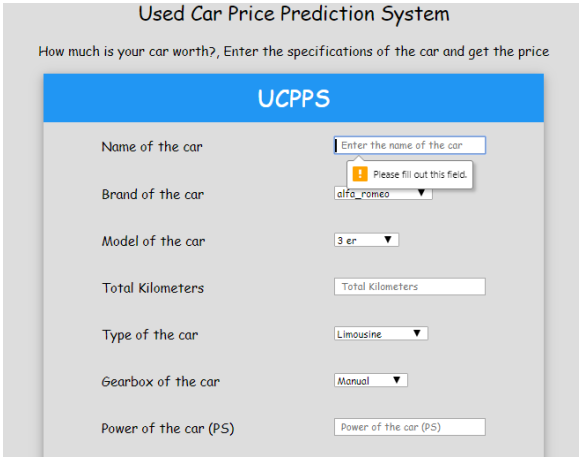

Figure 7.6: Empty form submission

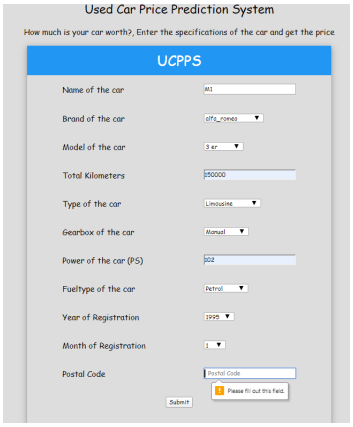

Figure 7.7: Incomplete form submission

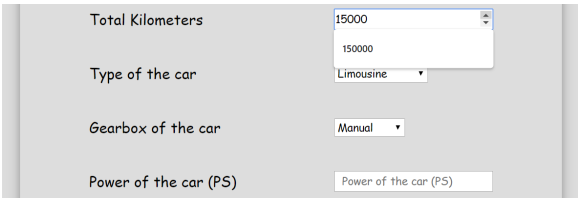

Figure 7.8: Only numeric values can be entered in certain fields

- **Responsive** The whole application is mobile-friendly. The application adjusts itself to different aspect rations. This is the functionality of using bootstrap. Below is the screenshot take from my mobile phone (One Plus 6T).

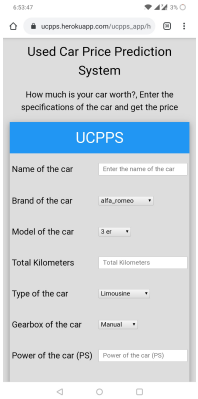

Figure 7.9: Mobile friendly application

- **Feedback option**: This application has a feedback option. The user if they are not satisfied with the resuts or they wanted the application to be improved

then they can give the feedback which is located below the submit button. The feedback form was not design from scratch, meanwhile it was extended from jotform.com.

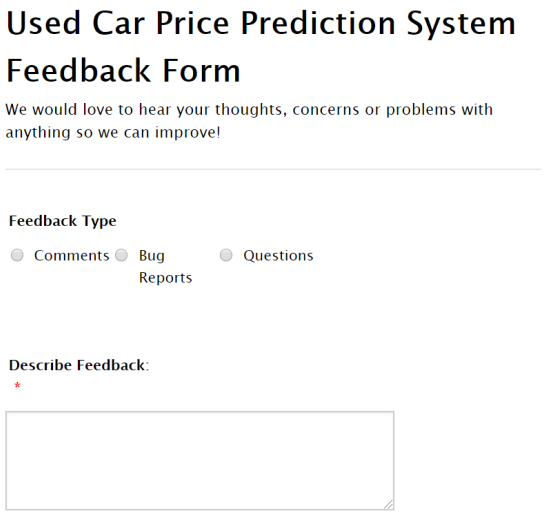

Figure 7.10: Feedback form

7.2 Deployment logs of the application

Below shown is the deployment log generated while deploying the used car price prediction application on Heroku. These logs help in understanding the deployment process of our application.

```
-----> Python app detected
-----> Installing SQLite3
Sqlite3 successfully installed.
-----> Installing requirements with pip
-----> $ python manage.py collectstatic --noinput
       146 static files copied to '/tmp/
   build_bf9922d1c1d6dba410b1ffe54daf6c4d/assets'.
-----> Discovering process types
       Procfile declares types -> web
-----> Compressing...
```

```
10        Done: 335.4M
11 -----> Launching...
12  !     Warning: Your slug size (335 MB) exceeds our soft limit (300
       MB) which may affect boot time.
13        Released v9
14        https://ucpps.herokuapp.com/ deployed to Heroku
```

Code Listing 7.1: Heroku deployment logs

7.3 Limitations of the application

Below are some of the limitations of the application deployed on the cloud

- **The application runs slow**: Since heroku is a free cloud hosting platform the speed of the application is restricted.
- **Limited space on the cloud**: As seen from the above logs code, this application has consumed 335MB of space due to this sometimes the application crashes. This is because the random forest algorithm takes some time to execute and if at all it exceeds 30 seconds then the request time out error will be displayed. The model was saved as a .sav file using pickle it was unable to upload the whole file to GitHub due to its size (277MB). Because GitHub only supports (100MB) maximum. Due to this the whole model will be executed from the beginning in the backend. Sometimes the speed also depends on internet connection.
- **Limited availability**: This application would is on free trial, hence the trial expires after 6 months. After that a domain needs to be purchased.

7.4 Link of the application

Link: https://ucpps.herokuapp.com/ucpps_app/home.html

7.5 Source Code Link

GitHub (Deployment) link: https://github.com/Tanu-N-Prabhu/UCPPS

GitHub (General Files) link: https://github.com/Tanu-N-Prabhu/UCPPS_files

7.6 Life cycle continuation

Finally the cycle does not end here with operationalize. It's a loop again if any new insights from the data are discovered then the whole process is restarted. Also, the web application would checked thoroughly and updates the changes regularly with new features.

Chapter 8

References

[1] Manashty, Dr. (2020). Data Science Fundamentals - Chapter 1. Presentation, University of Regina, Canada.

[2] Leka, O. Used Cars Data - dataset by data-society. Retrieved 8 April 2020, from https://data.world/data-society/used-cars-data

[3]Scrapy—A Fast and Powerful Scraping and Web Crawling Framework. Retrieved 8 April 2020, from https://scrapy.org/

[4] Burkov, A. (2019). The hundred-page machine learning book. Andriy Burkov.

[5] Sharma, N. (2018). Ways to Detect and Remove the Outliers [Blog]. Retrieved 9 April 2020, from https://towardsdatascience.com/ways-to-detect-and-remove-the-outliers-404d16608dba

[6] 3.2.4.3.1.sklearn.ensemble.RandomForestClassifier — scikit-learn 0.22.2 documentation. Retrieved 7 April 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[7] Aankul, A. (2017). T-test using Python and Numpy [Blog]. Retrieved 9 April 2020, from https://towardsdatascience.com/inferential-statistics-series-t-test-using-numpy-2718f8f9bf2f

[8] sklearn.metrics.mean_squared_error — scikit-learn 0.22.2 documentation. Scikit-

learn.org. Retrieved 9 April 2020, from http://scikit-learn.org/metrics

[9] sklearn.metrics.mean_absolute_error — scikit-learn 0.22.2 documentation. Scikit-learn.org. Retrieved 9 April 2020, from http://scikit-learn.org/metrics

[10] Cloud Application Platform—Heroku.

Retrieved 8 April 2020, from https://www.heroku.com/