Data Science Project Screenshots

Tanu Nanda Prabhu

University of Regina, Canada

1 Introduction to data analytic lifecycle

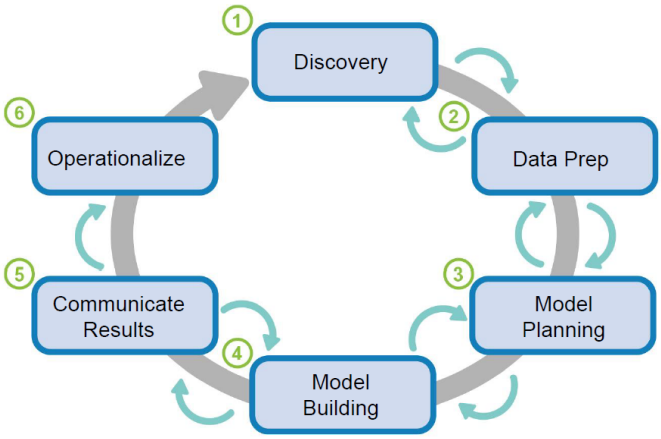

Figure 1: Data Analytic life cycle

2 Data Discovery

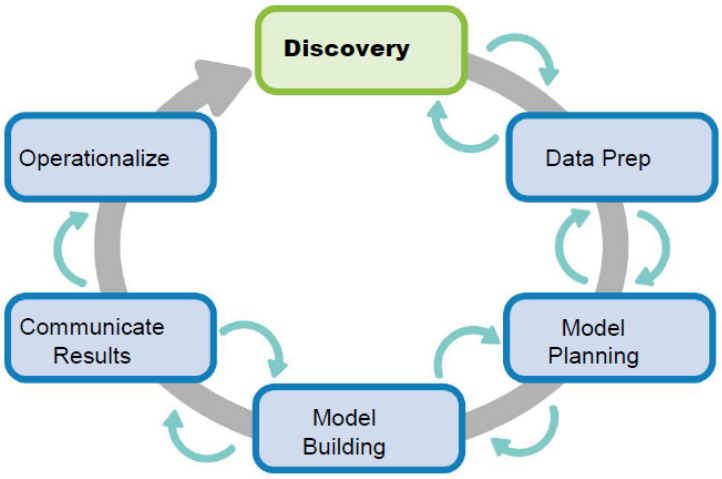

Figure 2: Data Discovery

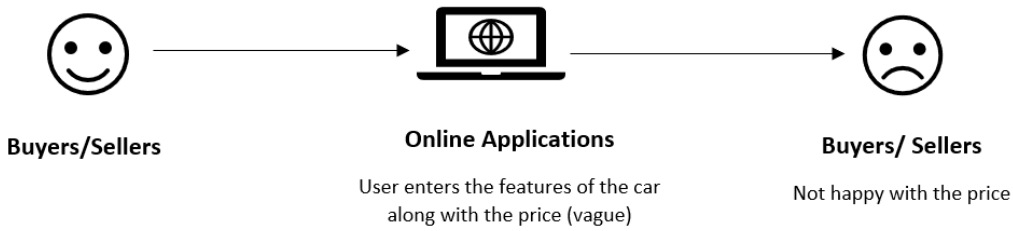

Figure 3: Current Situation

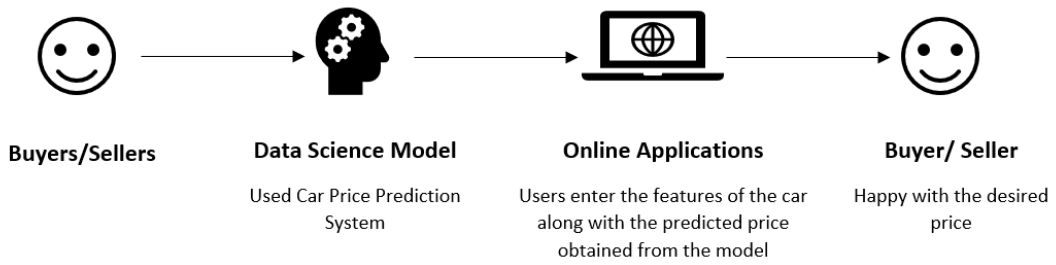

Figure 4: Desired Situation

3 Data Preparation

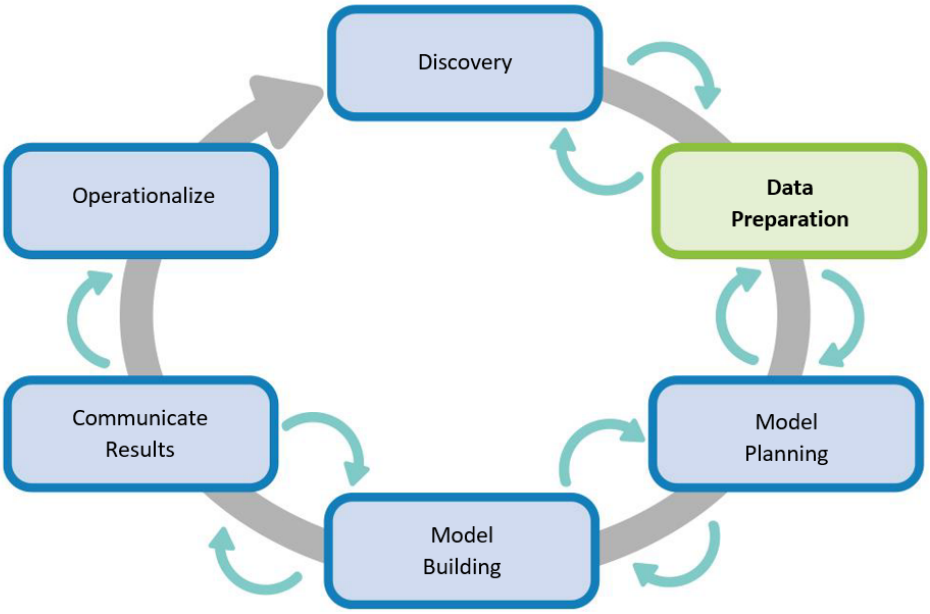

Figure 5: Data Preparation

```
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 6: Dataset formatted as a pandas dataframe

```
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 7: Dataset formatted as a pandas dataframe

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

```
(371528, 20)
```

Figure 8: Shape of the dataframe

```
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 9: 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 10: Details of features of the dataframe

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

```
2
```

Figure 11: Dimensions of the dataframe

```
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 12: 5 number summary of the data frame

```
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 13: Data type of the features in the data frame

```
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>]
```

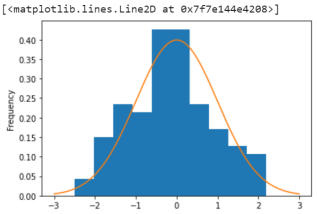

Figure 14: Price density normal histogram plot

```
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>]
```

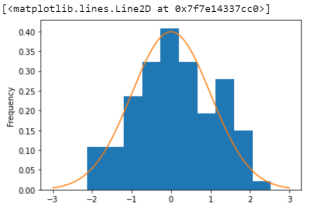

Figure 15: Power density normal histogram plot

```
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>]
```

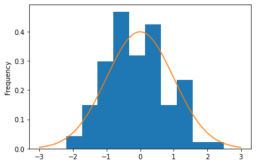

Figure 16: Year of registration density normal histogram plot

```
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>]
```

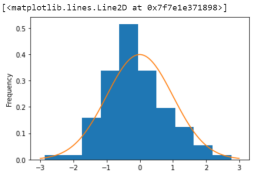

Figure 17: Month of registration density normal histogram plot

```
# 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 18: Setting the threshold to remove the outliers

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

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

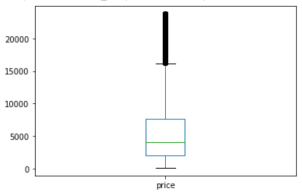

Figure 19: Box plot - Price

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

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

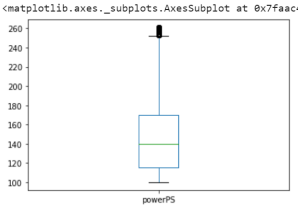

Figure 20: Box plot - Power

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

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

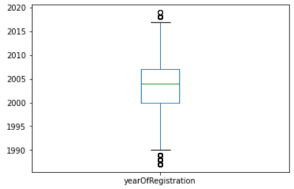

Figure 21: Box plot - Year of Registration

```
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 22: Calculating the IQR score for all the columns

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

Figure 23: Cleaning all the columns values based on the IQR scores generated above

```
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 24: Grouping the values in the offer type column

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

Figure 25: Dropping all the irrelevant columns

```
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 26: Mostly cleaned data frame

```
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 27: Total count of null values in the data frame

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

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

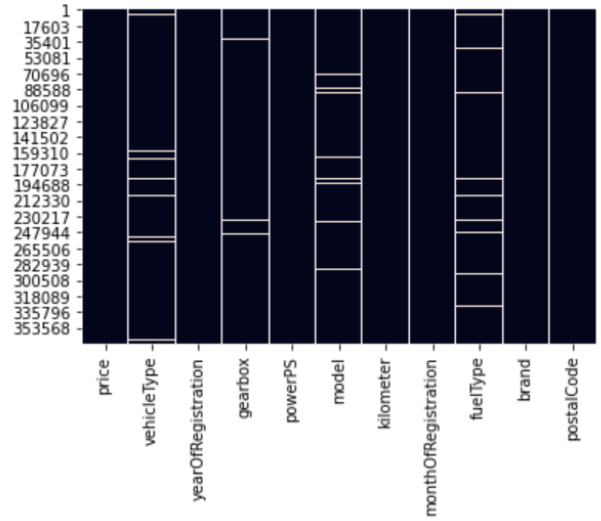

Figure 28: Using a seaborn plot to detect null values

```
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 29: Grouping the vehicle type with the fuel type

```
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 30: Using the value petrol to fill all the missing fuel type values

```
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 31: Filling the model columns with 3er

```
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 32: Assigning a unique identifier for the groups

```
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 33: Duplicate rows with the same id number

```
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 34: 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 35: Displaying last 10 rows the translated data

```
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 36: Values converted from string to numeric

```
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 37: One hot encoding implemented on vehicle type and gearbox columns

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 38: Chi square test for getting the important features

```
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")
```

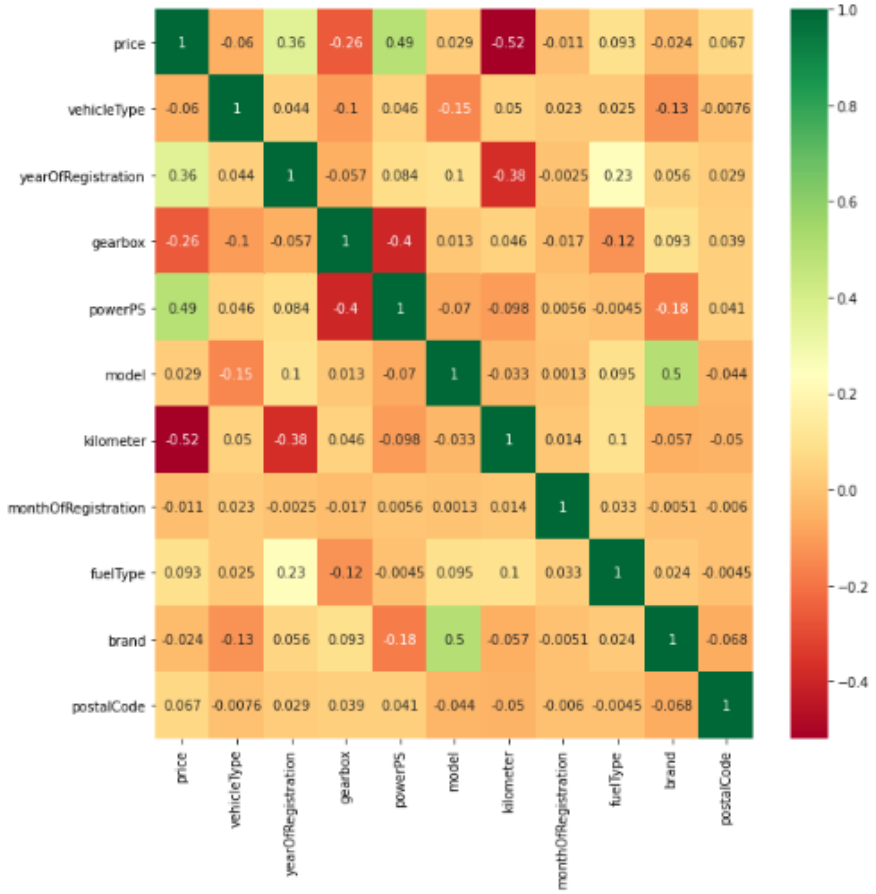

Figure 39: Heat Map

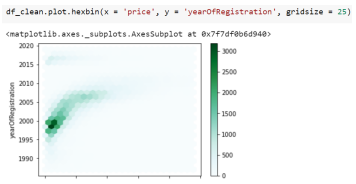

Figure 40: Hexbin plot - Price vs Year of Registration

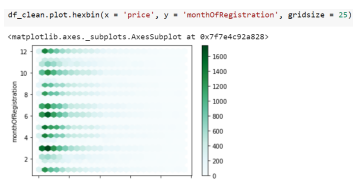

Figure 41: Hexbin plot - Price vs Month of Registration

Figure 42: Hexbin plot - Price vs Model

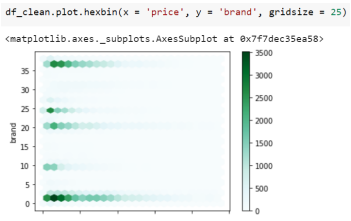

Figure 43: Hexbin plot - Price vs Brand

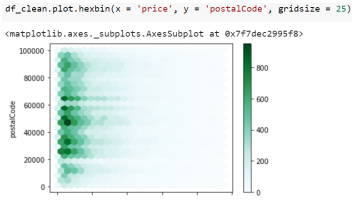

Figure 44: Hexbin plot - Price vs Postal code

Figure 45: Hexbin plot - Price vs Kilometer

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

Figure 46: Scatter matrix

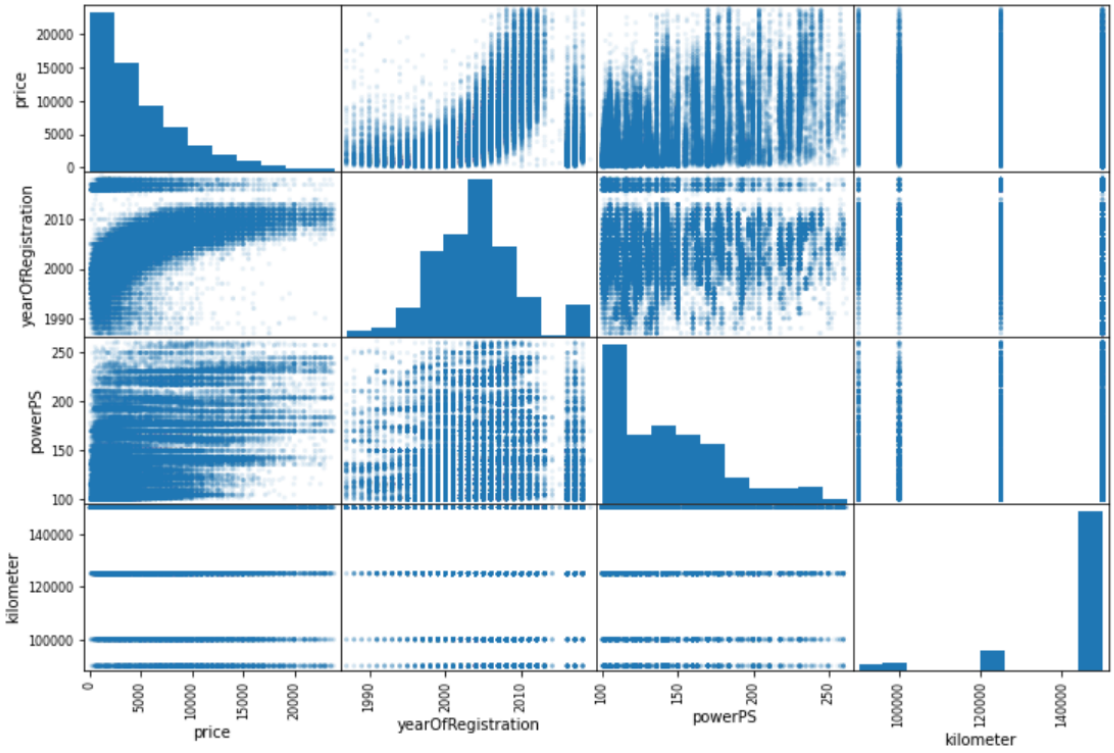

Figure 47: Scatter plot of all the features

4 Model Planning

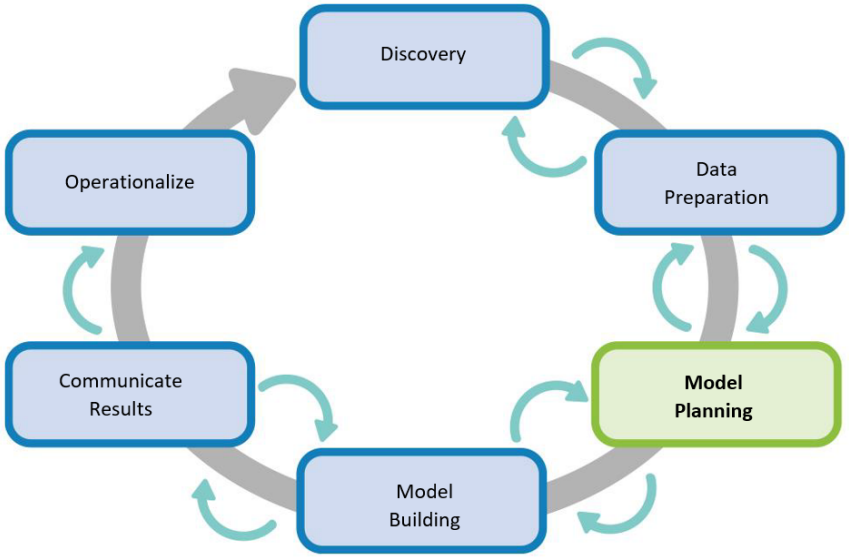

Figure 48: Model Planning

5 Model Building

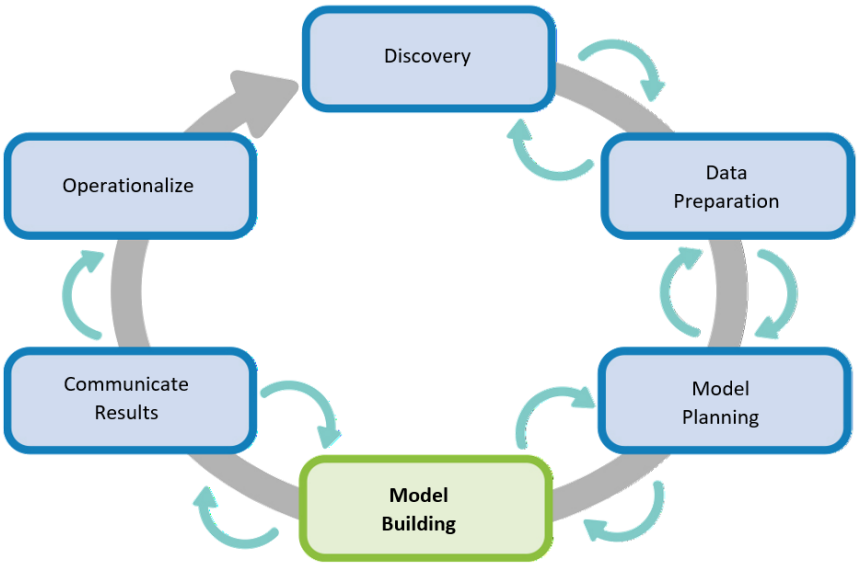

Figure 49: Model Building

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

Figure 50: Splitting the data into three sets

```
# 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 51: Creating pipelines for different regressors

```
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 52: Building the pipelines

```
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 53: Random Forest Regression

```
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 54: 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()
```

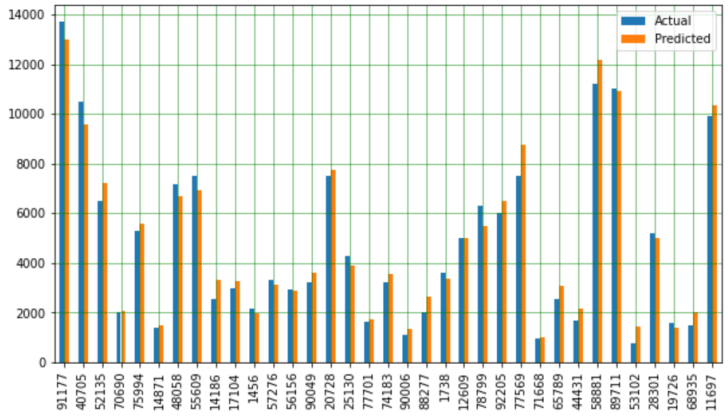

Figure 55: Actual vs predicted train values plot

```
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 56: Decision Tree Regression

```
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 57: 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()
```

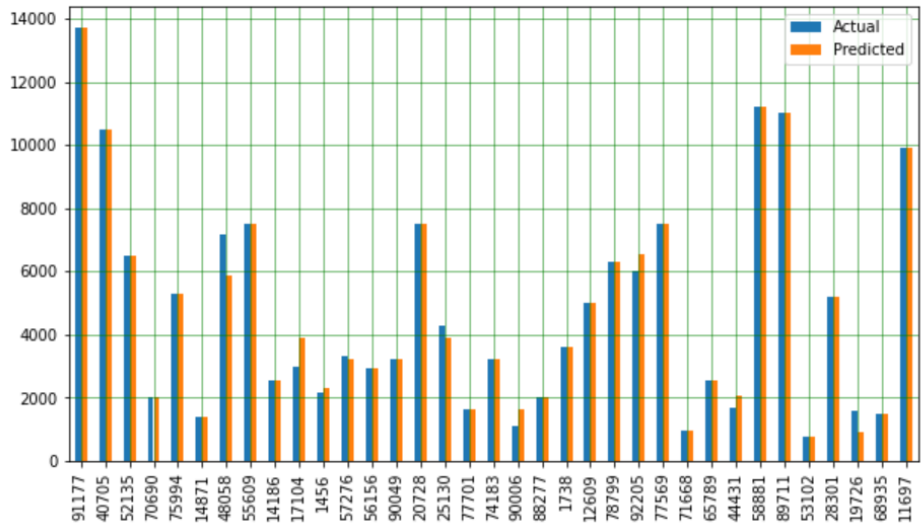

Figure 58: Actual vs predicted train values plot

Regression Algorithms	Model Accuracy
Random Forest	97.85146144913742
Decision Tree	99.36287739304052

Figure 59: Random Forest vs Decision Tree

```
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 60: Random Forest 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 61: Decision Tree Regression

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 62: Parameters of random forest regression (Manual hyperparameter tuning)

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 63: The model with the best parameters

```
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 64: Grid Search CV

```
--- 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 65: Best parameters chosen by grid search cv

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 66: Performance comparison between manual tuning and grid search cv

```python
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 67: 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

Figure 68: Lasso 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 69: Ridge Regression

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

Figure 70: Ridge Regression

```
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 71: Ridge Regression

```
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 72: Ridge Regression

```
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 73: Random Forest Regression

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

Figure 74: Random Forest Regression

```
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 75: 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()
```

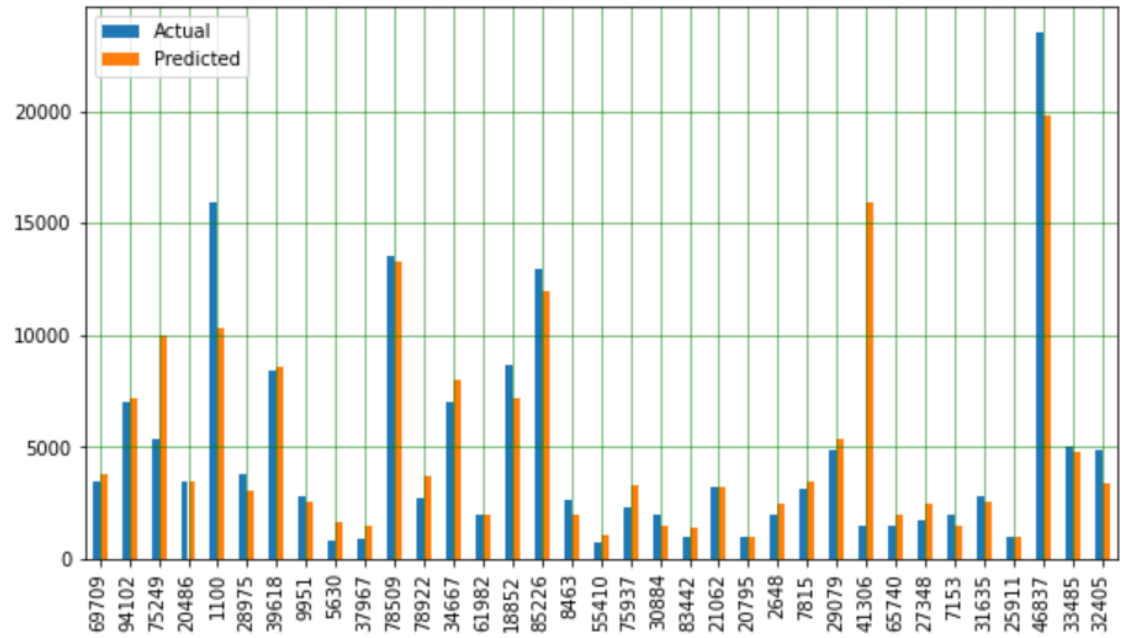

Figure 76: Actual vs predicted values plot

6 Communicate Results

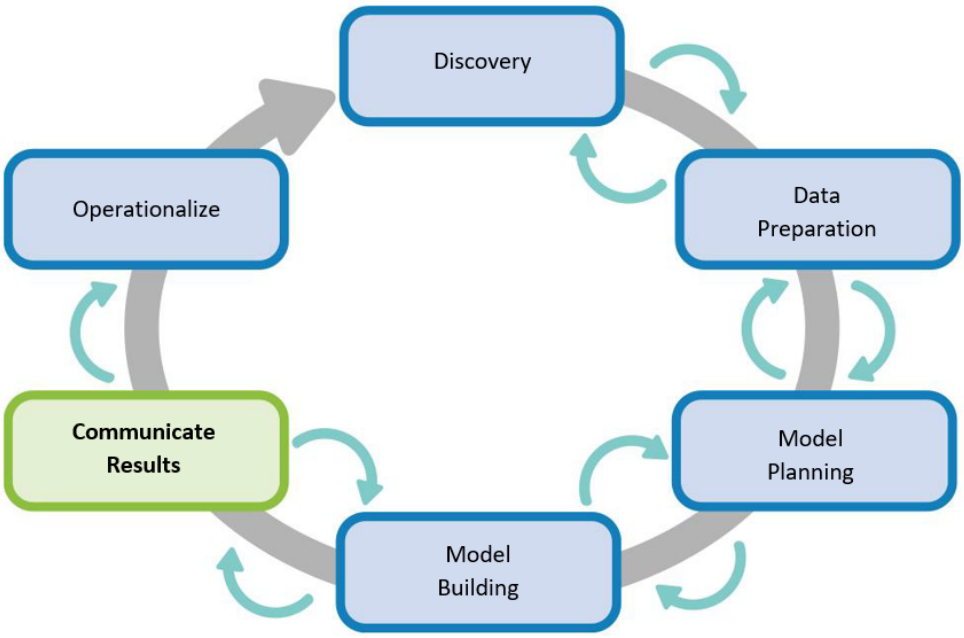

Figure 77: Communicate Results

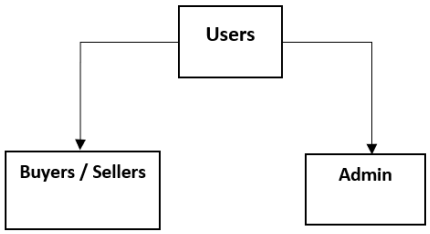

Figure 78: Users of the application

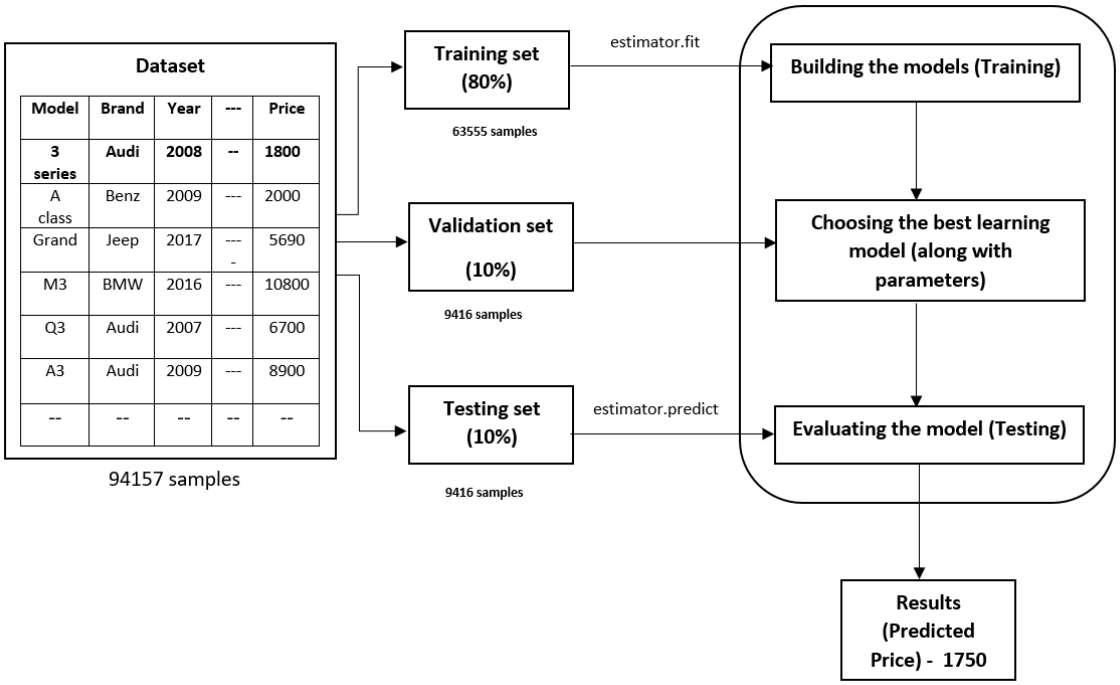

Figure 79: Block diagram of model

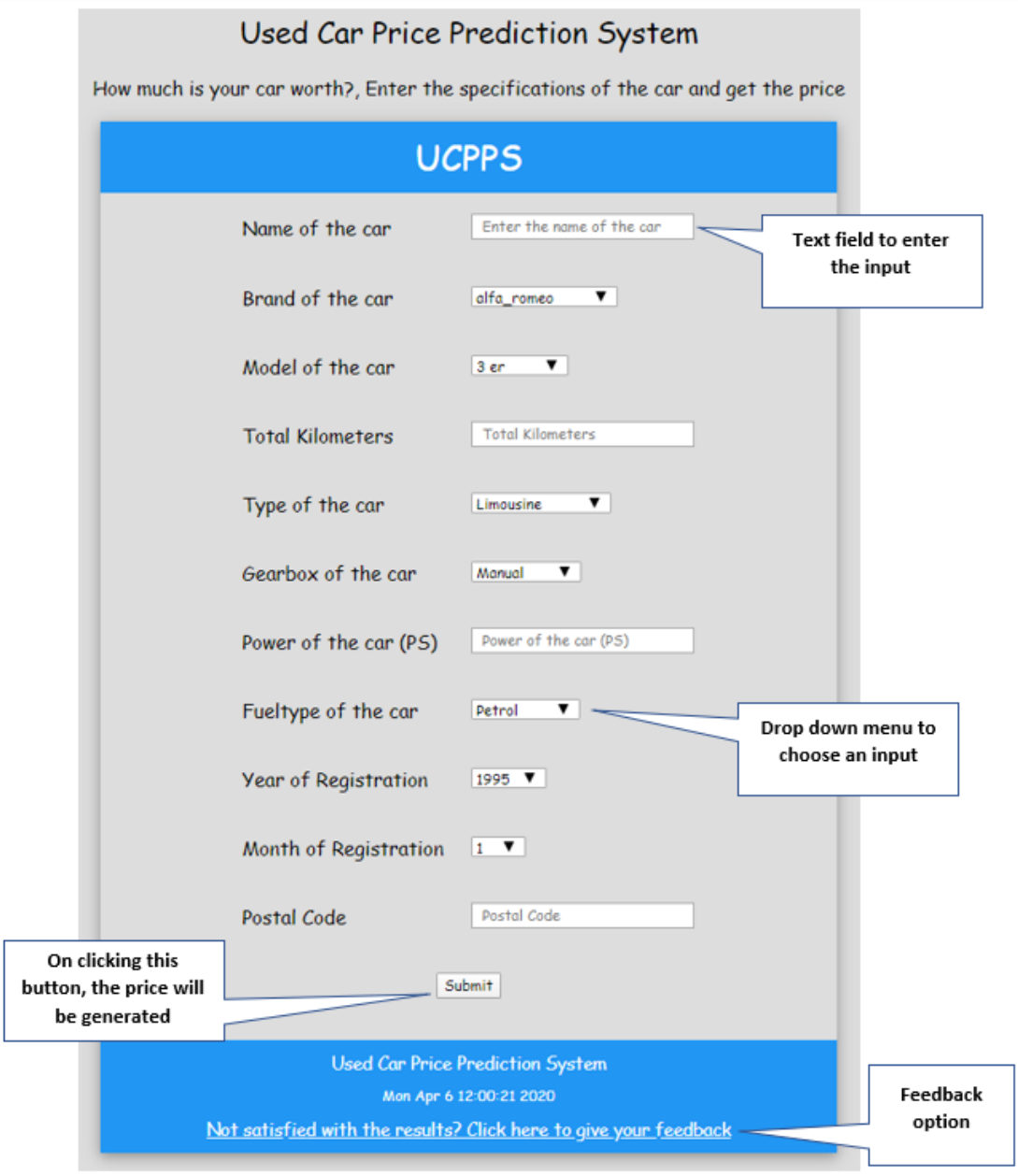

Figure 80: Used Car Price Prediction Application

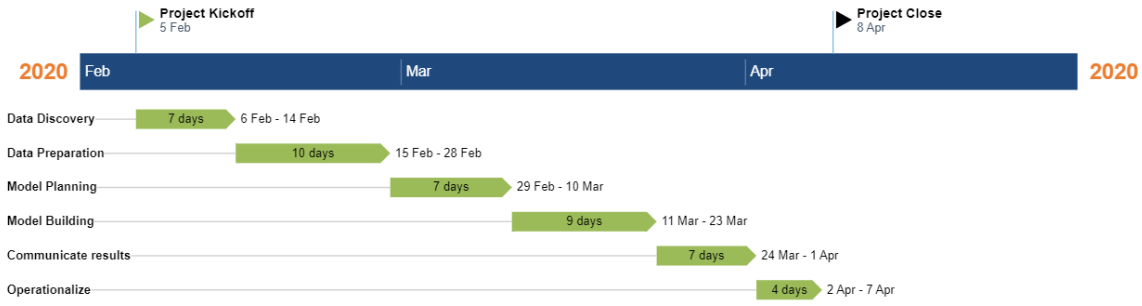

Figure 81: Time line of the project

7 Operationalize

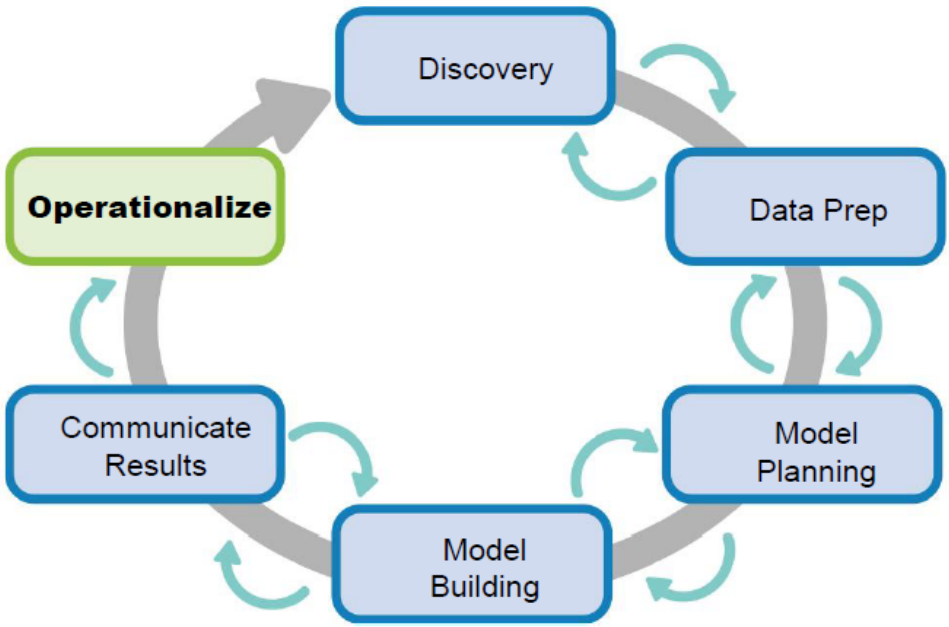

Figure 82: Operationalize

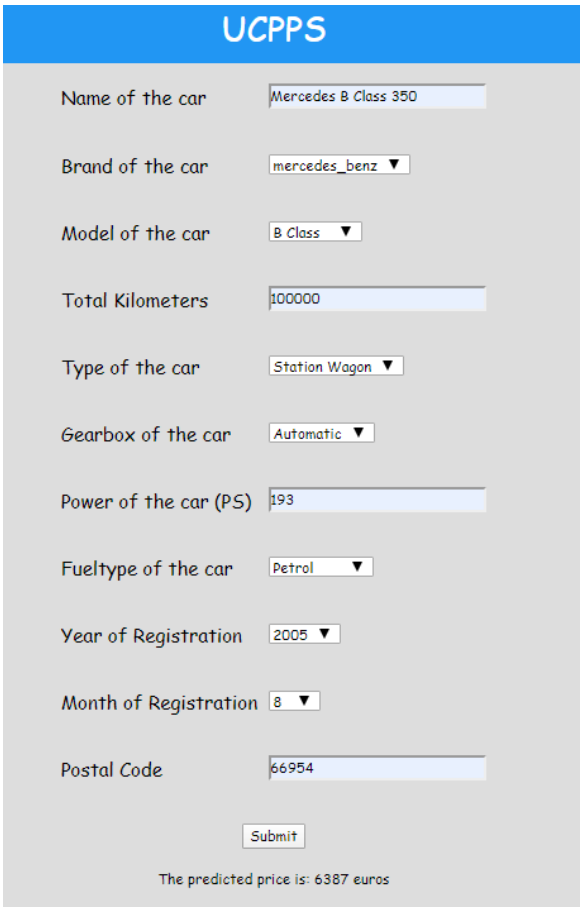

Figure 83: User entering input

```
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 84: Original Price of the car

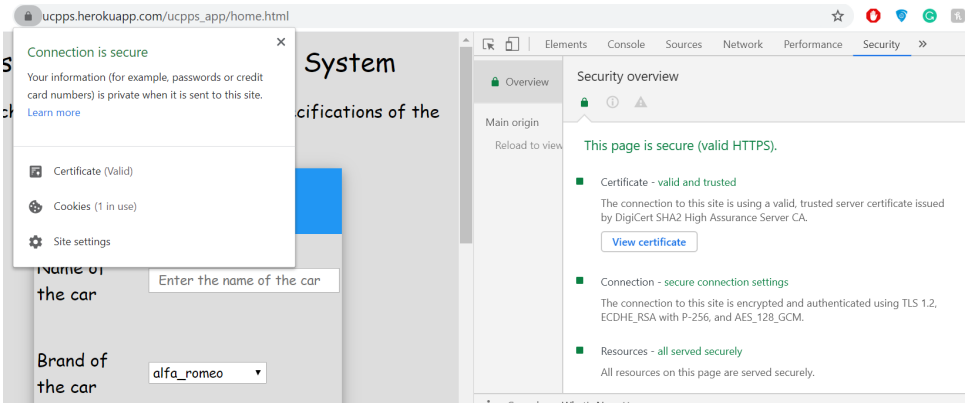

Figure 85: HTTPS secure connection of the application

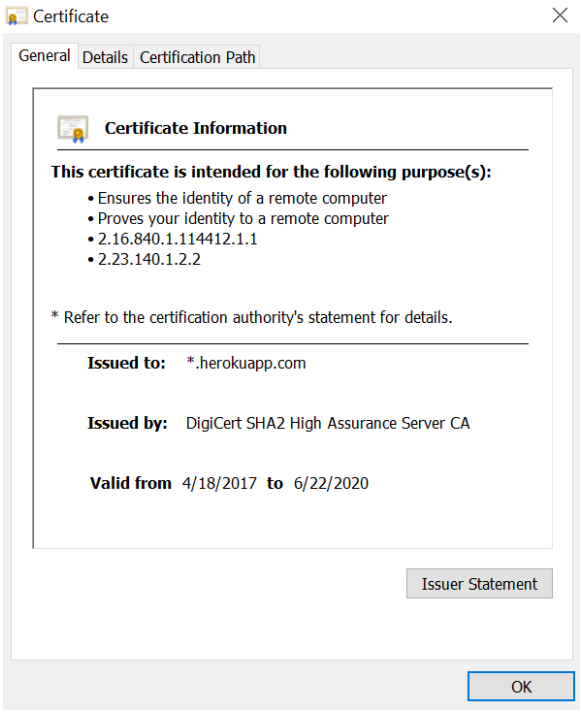

Figure 86: The certificate information of the application provided by DigiCert

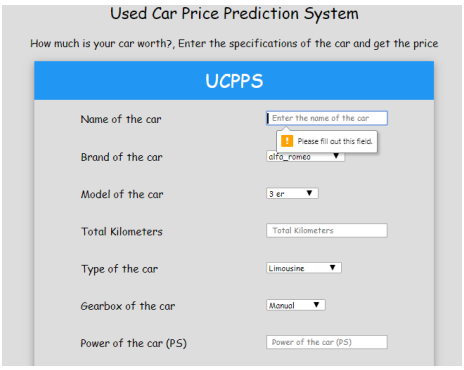

Figure 87: Empty form submission

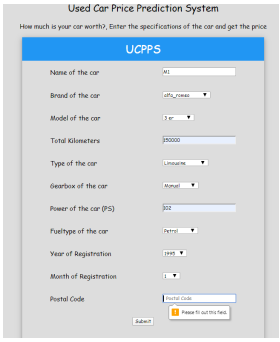

Figure 88: Incomplete form submission

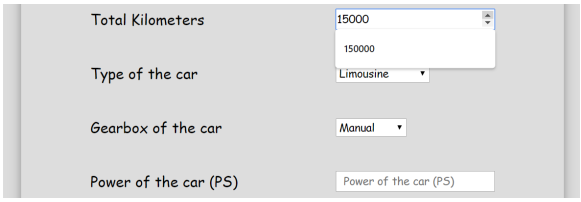

Figure 89: Only numeric values can be entered in certain fields

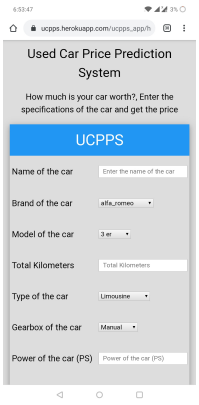

Figure 90: Mobile friendly application

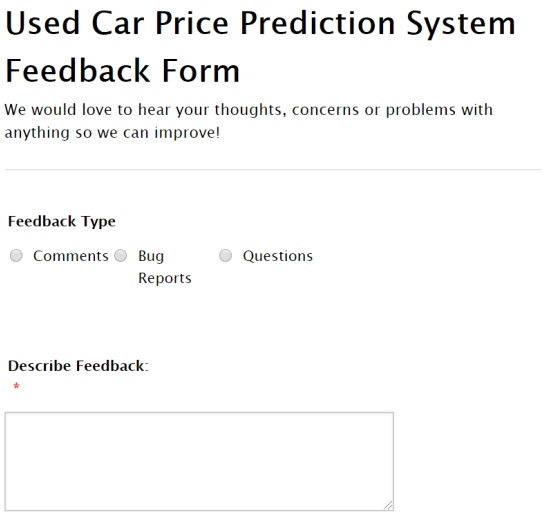

Figure 91: Feedback form