How much is your car worth? A **U**sed **C**ar **P**rice **P**rediction **S**ystem (UCPPS)

Faculty of Graduate Studies & Research
Instructor: Dr. Alireza Manashty
Student Name: Tanu Nanda Prabhu

4/8/2020

University of Regina

Table of Contents

DATA ANALYTIC LIFECYCLE

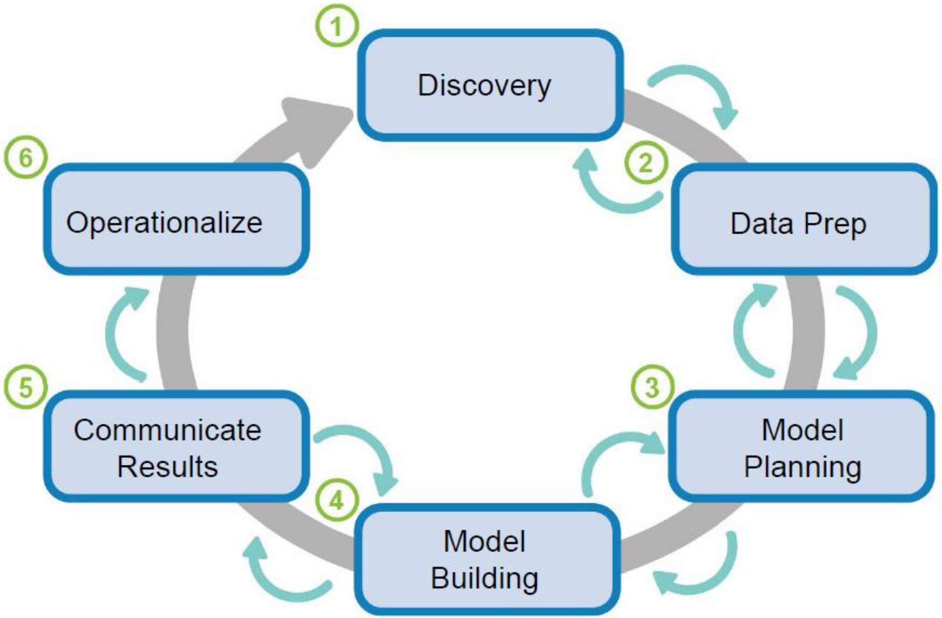


Image Credits: Manashty, 2020 [1]

INTRODUCTION

- Vehicle value forecast is a significant errand particularly when the vehicle is used.
- The value of the car depends on several factors:
  - Make (brand of the car)
  - Power
  - Number of kilometers it has been run
  - Year of registration, and many more
- Better the features higher the price

Image Credits: Manashty, 2020 [2]

University of Regina

PROBLEM STATEMENT

•Used Car Prices are important reflection of the economy and they greatly interest both buyers and sellers.

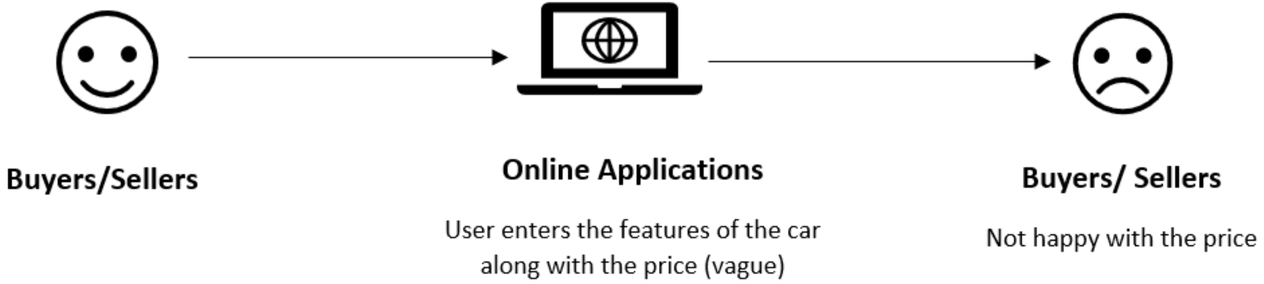
•A prediction model that estimates resale price based on car's attributes or features is much more needed today.

•My analysis aims to determine which features of the car that may have the strongest statistical correlation with the price of the car.

University of Regina

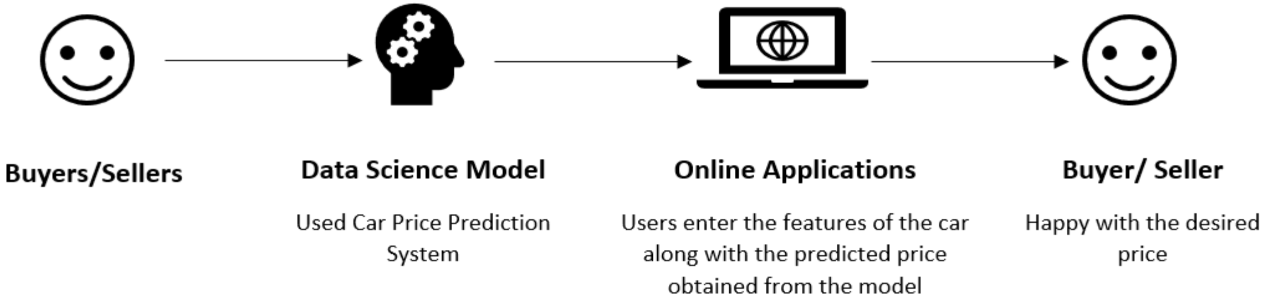
PROBLEM STATEMENT

- Current Situation



PROBLEM STATEMENT

- Desired Situation



DATA

• The data set was chosen from data.world, which was originally scraped from e-bay [3]

```
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

DATA STATISTICS

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

```
<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
```

- **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

```
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

University of Regina

DATA EXPLORATION

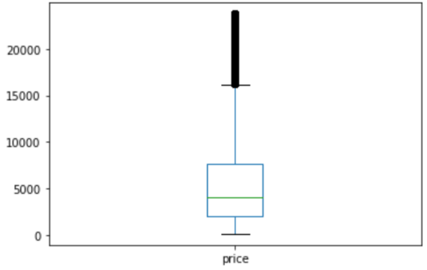
Examining distribution of all variables

- Analyzing the outliers using the box-plot
- Using histogram density by plotting a bell curve
- Removing the outliers using IQR and Manual removing technique

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

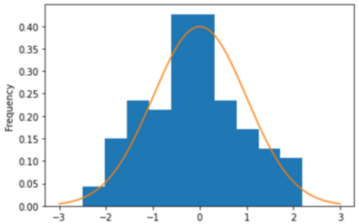
```
df_clean['price'].plot.box()
```

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




```
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>]
```



DATA EXPLORATION

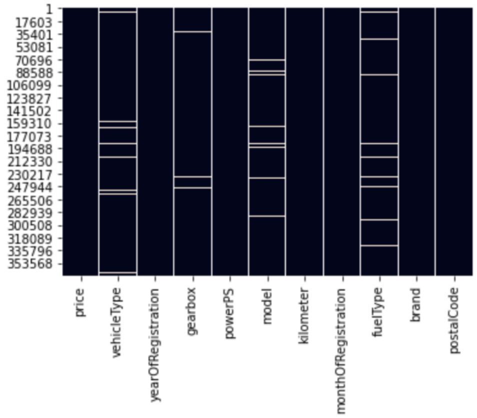
Detecting missing values using plot

```
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
```

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

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



DATA EXPLORATION

- Translating the data frame from German to English

```
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
```

```
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

DATA EXPLORATION

- **One hot encoding**
  - One hot encoding will be only implemented only on **vehicleType** and **gearbox** columns

```
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

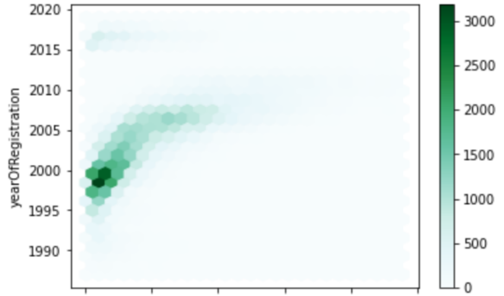
DATA VISUALIZATION

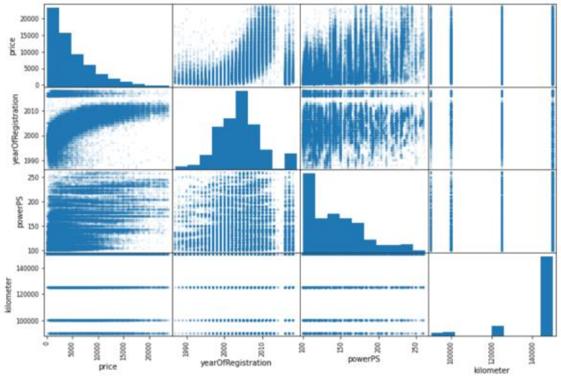
Data was visualized using different plots

- Hexagonal bin plots

- Scatter matrix plot

```
df_clean.plot.hexbin(x = 'price', y = 'yearOfRegistration', gridsize = 25)
<matplotlib.axes._subplots.AxesSubplot at 0x7f7df0b6d940>
```





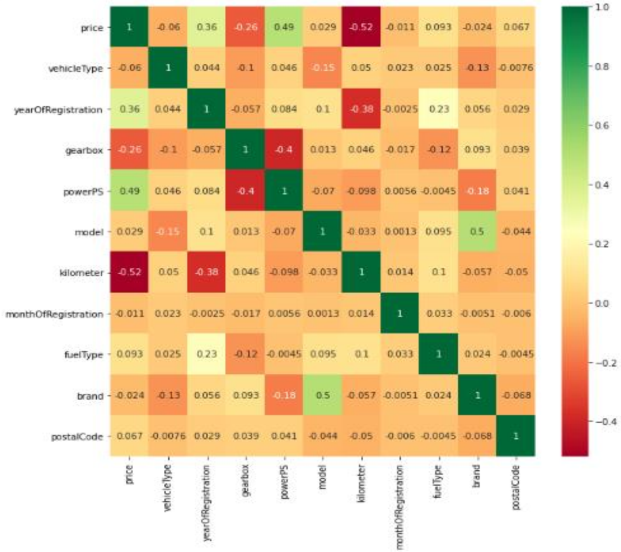
FEATURE SELECTION

- Chi Squared test

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

- Heat map

```
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")
```



Model Planning

- Variable selection

```
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']
```

- Model Selection
    - Classification
    - **Regression**
    - Association Rules
    - Text/Image/Video Analysis

University of Regina

Model Building

- Splitting the dataset into three sets
  - **Training set** - 80%
  - **Validation set** - 10%
  - **Testing set** - 10%

```python
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)
```

- Choosing the learning algorithm with validation set

```python
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 ---
```

Model Building

- **Underfitting**
  - No underfitting

Regression Algorithms	Model Accuracy
Random Forest	97.8514614913742
Decision Tree	99.36287739304052

- **Overfitting**
  - Slightly overfitting

```
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
```

University of Regina

Model Building

- **Hyper parameter tuning**
  - Manually
  - 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

University of Regina

Model Building

- **Manual Hyper Parameter Tuning 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

Model 10 shown above performed better compared to others

Model Building

- **Grid Search CV**

```python
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)
```

```
--- 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}
```

University of Regina

Model Building

- **L1 – Regularization**
  - **Lasso**

```
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)
```

L1 - Regularization	Accuracy	Time taken to execute(s)
Lasso Regression - alpha = 10000	-0.004225239923694	0.018745422363281125
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

Model Building

- **L2 - Regularization**
    - **Ridge**

```
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)
```

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	0.020066738128662 11
Ridge Regression - alpha = 1	56.88618326663119	0.018297672271728516
Ridge Regression - alpha = 0.1	56.88619615287895	0.021454572677612305

University of Regina

Model Building

- MSE train

```
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
```

- MSE test

```
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
```

- T-test
  - $t = -124.323$, $p = 0.02$

Model Building

- Model Performance Assessment
  - Final Model with the best parameters

```
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 ---
```

Model Building

- Model Performance Assessment
  - Summary of performance metrics

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

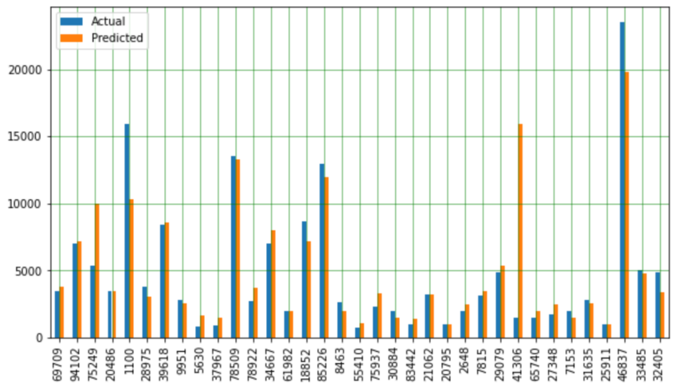
Model Building

- Comparing the actual values vs predicted values

```
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

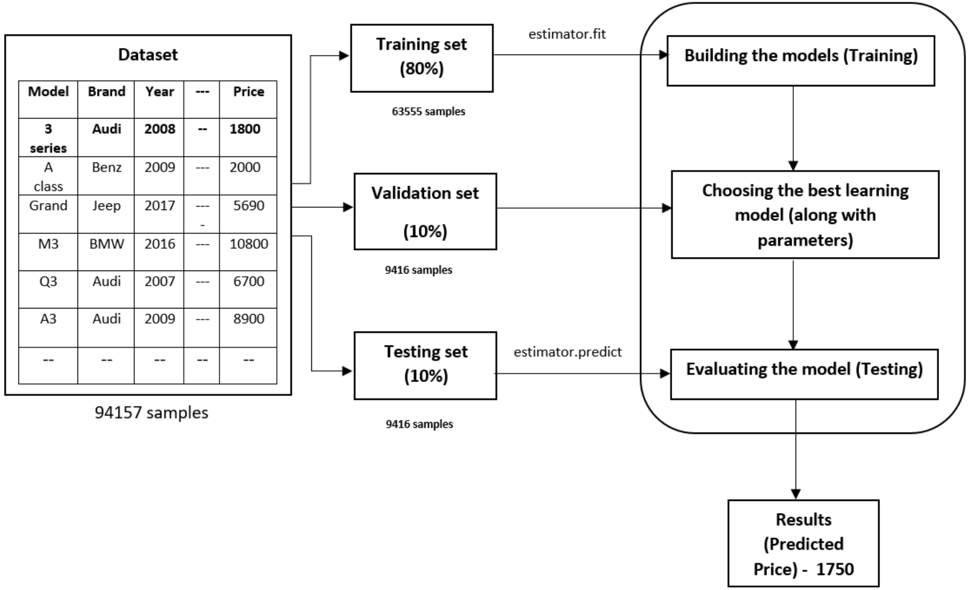
```
import matplotlib.pyplot as plt
df1 = df.head(35)
df1.plot(kind='bar',figsize=(10,5.5))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



University of Regina

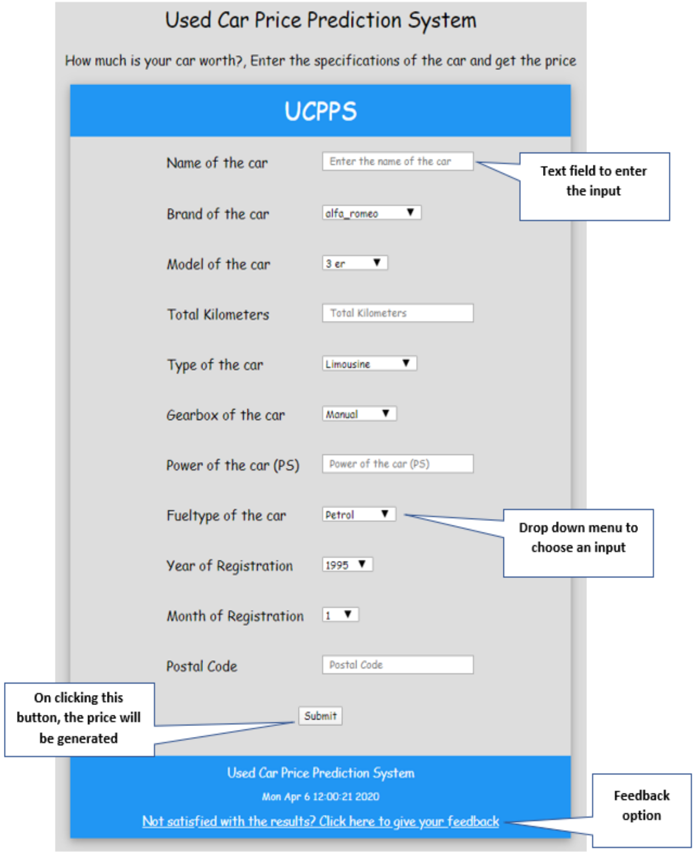
SOLUTION OVERVIEW

- Category of data falls under supervised machine learning: Regression Estimator



Operationalize

- Component Of Project
  - Fully developed and Operational Cloud-based published website. (Copied from [UR Courses])

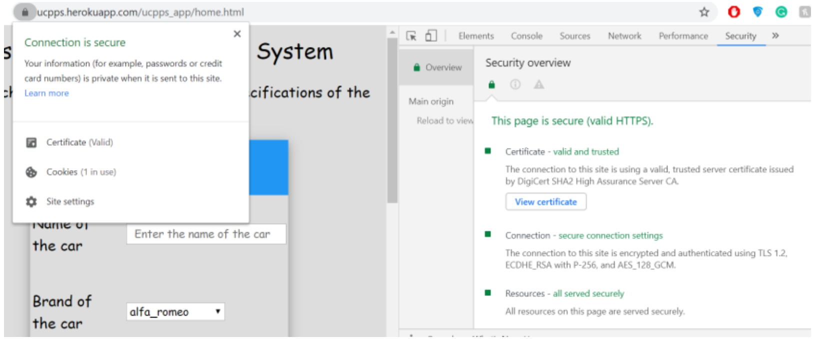


- Link: https://ucpps.herokuapp.com/ucpps_app/home.html

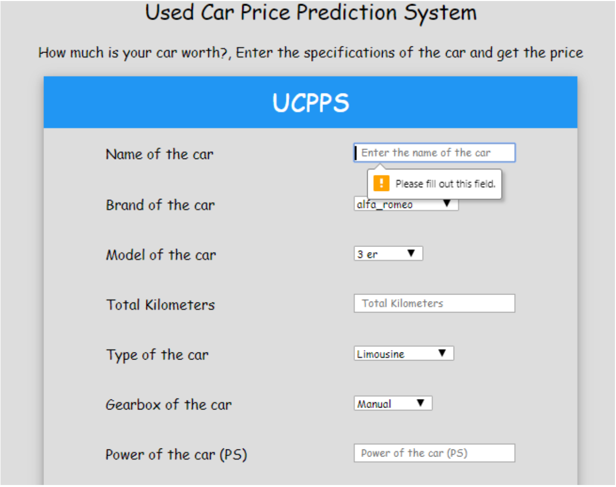
University of Regina

Operationalize

- Features of the application



Connection link is secure

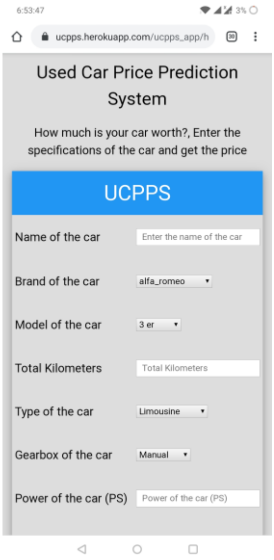


Application is robourst

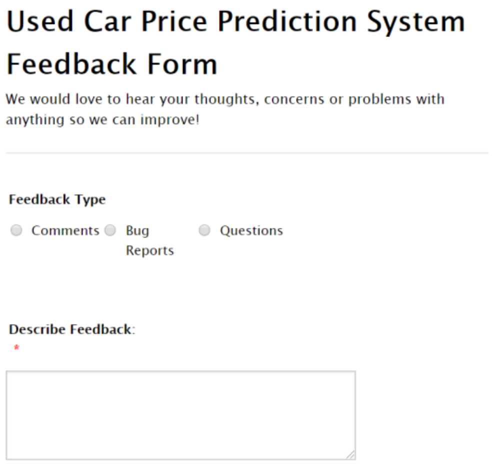
University of Regina

Operationalize

- Features of the application



Responsive (Mobile Screen)

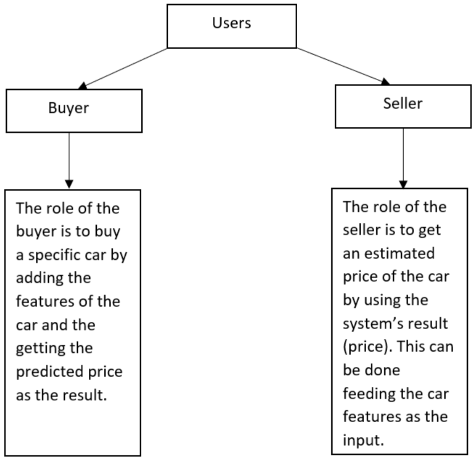


Option to give feedback

University of Regina

USERS

- Two Types of Users
  - Buyer
  - Seller
- Both of the users don’t have to worry about paying excess or end getting less paid.
- Canada Used Car Dealer Retail Sales is at a level of 1.056B CAD for Nov 2019 [2]

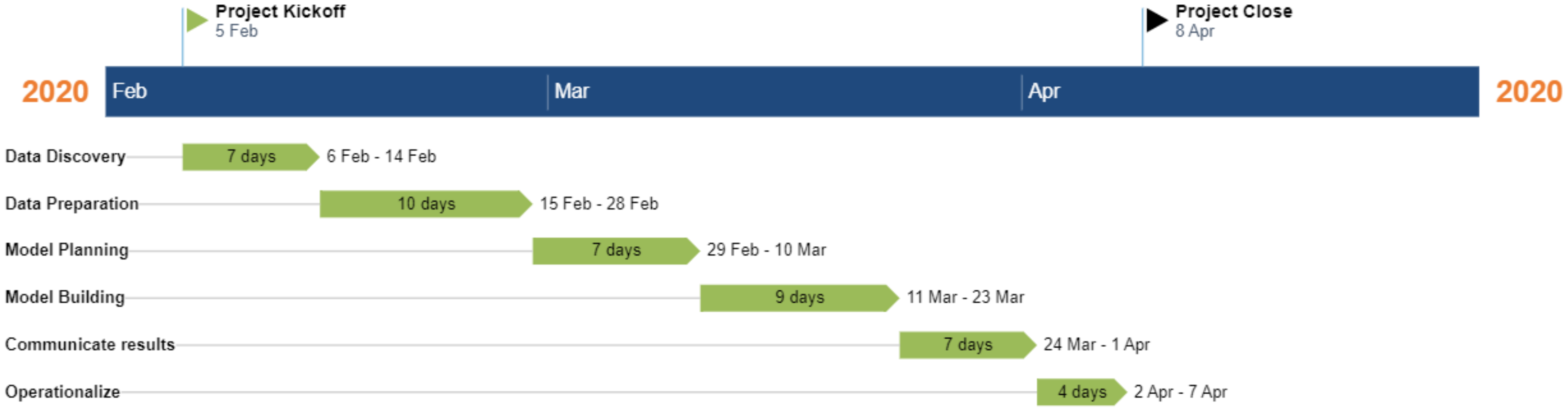


University of Regina

TOOLS

- Google Colab
- Anaconda Jupyter Notebook
- Python Programming (Obviously)
- Python Libraries (Pandas, NumPy, Matplotlib, scikit learn)
- Django Web framework (HTML, CSS, Bootstrap)
- GitHub
- Visual Studio Code
- Heroku

TIMELINE



University of Regina

TEAM ROLES

- Data Collection, data understanding
- Model Design, model evaluation
- Code Documentation
- Deployment and building a functional website.

- 2 new things that took place recently
  - Uploaded my first YouTube video (Live coding) about code deployment
  - Deployed my first project on to the cloud

Image Credits: Medium

Important links

- **Application Link**: https://ucpps.herokuapp.com/ucpps_app/home.html

- **GitHub General files link**: https://github.com/Tanu-N-Prabhu/UsedCarPricePredictionSystem-Files

- **GitHub deployment code link**: https://github.com/Tanu-N-Prabhu/Used_Car_Price_Prediction_System1

University of Regina

REFERENCES

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

[2] *The all-new BMW 5-series (G30) launched – All You Need to Know*. (2020). [Image]. Retrieved 8 April 2020, from http://kensomuse.com/blog/2017/03/30/new-bmw-5-series-g30-launched-need-know/

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

[4] Canada Used Car Dealers Retail Sales. Retrieved 5 February 2020, from https://ycharts.com/indicators/canada_used_car_dealers_retail_sales

Thank You