CAR RESALE VALUE PREDICTION

Project Report

1. INTRODUCTION

1.1 Project Overview

In this fast world, you don't have your own personal mode of transportation sort of an automobile, life will become even additional agitated. The public choose to obtain their automobile as a result of its convenience to commute between places, permits movement with an outsized cluster of individuals with fuel potency, and safe mode of transport. The used automobile marketplace is witnessing a boom in India, with the decision for luxurious vehicles sometimes increasing. Till a couple of years, owning a luxury automobile won't be a dream for varied shoppers, as a result of money hurdles, however, this is often bit by bit dynamic as shoppers can simply obtain used luxury vehicles. Machine Learning provides numerous ways through that it's easier to predict the worth of an automobile, by the previous information that is obtainable. We've enforced the model exploitation supervised Learning techniques of Machine Learning, which is outlined by its use of labeled information sets to coach algorithms to classify data or predict outcomes accurately. As the input file is fed into the model, it adjusts its weights till the model has been fitted fittingly, which happens as a part of the cross-validation method. If there is also further transparency within the marketplace and fewer intermediaries, the seller ought to get the next value for a vehicle and therefore the shopper ought to get one at a lower fee as margins get reduced on every facet

1.2 Purpose

The main idea of making a car resale value prediction system is to get hands-on practice for python using Data Science. Car resale value prediction is the system to predict the amount of resale value based on the parameters provided by the user. User enters the details of the car into the form given and accordingly the car resale value is predicted

2. LITERATURE SURVEY

2.1 Existing problem

(Gegic, Isakovic, Keco, Masetic, & Kevric, 2019) from the International Burch University in Sarajevo, used three different machine learning techniques to predict used car prices. Using data scrapped from a local Bosnian website for used cars totalled at 797 car samples after pre-processing, and proposed using these methods: Support Vector Machine, Random Forest and Artificial Neural network. Results have shown using only one machine learning algorithm achieved results less than 50%, whereas after combing the algorithms with pre calcification of prices using Random Forest, results with accuracies up to 87.38% was recorded.

2.2 References

1. https://www.kaggle.com/jpayne/852k-used-car-listings

2. N. Monburinon, P. Chertchom, T. Kaewkiriya, S. Rungpheung, S. Buya and P. Boonpou, "Prediction of prices for used car by using regression models," 2018 5th International Conference on Business and Industrial Research (ICBIR), Bangkok, 2018, pp. 115-119.

3. Listiani M. 2009. Support Vector Regression Analysis for Price Prediction in a Car Leasing Application. Master Thesis. Hamburg University of Technology

4. Chen, Tianqi, and Carlos Guestrin. "Xgboost: A scalable tree boosting system." Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. ACM, 2016.

5. Ke, Guolin, et al. "Lightgbm: A highly efficient gradient boosting decision tree." Advances in Neural Information Processing Systems. 2017.

6. Fisher, Walter D. "On grouping for maximum homogeneity." Journal of the American statistical Association 53.284 (1958): 789-798.

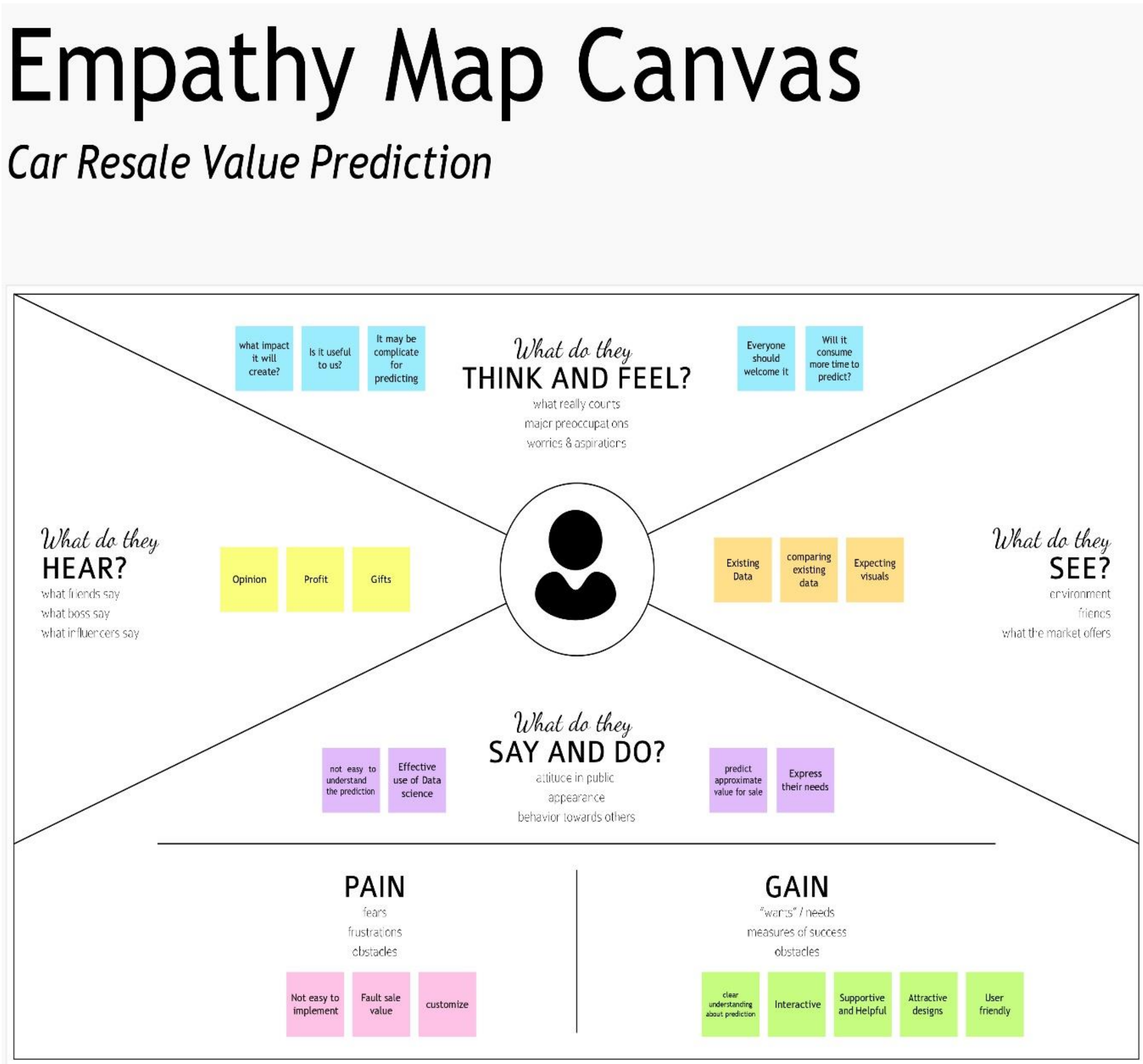
7. https://scikit-learn.org/stable/modules/classes.html: Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

2.3 Problem Statement Definition

It is easy for any company to price their new cars based on the manufacturing and marketing cost it involves. But when it comes to a used car it is quite difficult to define a price because it involves it is influenced by various parameters like car brand, manufactured year and etc. The goal of our project is to predict the best price for a pre-owned car in the Indian market based on the previous data related to sold cars using machine learning.

3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas



3.2 Ideation & Brainstorming

3.3 Proposed Solution

S.No:	Parameter	Description
1.	Problem Statement (Problem to be solved)	User needs a way to buy recommended used cars on online through all the used cars available in the platform so that they can save time on surfing through the Internet and different platforms!
2.	Idea / Solution description	To develop a efficient and effective model which predicts the price of a used car according to user's inputs. To develop a User Interface(UI) which is user-friendly and takes input from the user and predicts the price.
3.	Novelty / Uniqueness	Accuracy in Price Prediction. Variety of car collections
4.	Social Impact / Customer Satisfaction	A car price prediction has been a high-interest research area, as it requires noticeable effort and knowledge of the field expert. Considerable number of distinct attributes are examined for the reliable and accurate prediction. The final prediction model was integrated into Java application. Furthermore, the model was evaluted using test data and the accuracy of 87.38% was obtained
5.	Business Model (Revenue Model)	By using this system, the users can predict and analyze the picture of the Model and price . In which it results to the visualizing the description of the Model taken as input.

6.	Scalability of the Solution	In future this machine learning model may bind with various website which can provide real time data for price prediction. Also we may add large historical data of car price which can help to improve accuracy of the machine learning model. We can build an android app as user interface for interacting with user. For better performance, we plan to judiciously design deep learning network structures, use adaptive learning rates and train on clusters of data rather than the whole dataset

3.4 Problem Solution fit

Project Title: Car Resale Value Prediction

Project Design Phase-I - Solution Fit Template

Team ID: PNT2022TMID07691

Define CS, fit into CC

1. CUSTOMER SEGMENT(S) CS

Who is your customer?
i.e. working parents of 0-5 y.o. kids

Used car sellers

6. CUSTOMER CONSTRAINTS

What constraints prevent your customers from taking action or limit their choices of solutions? i.e. spending power, budget, no cash, network connection, available devices.

- To determine the worth of the car by their own within a minutes
- Customer's loss of money can be optimized by spending money for dealers, brokers to buy or sell a car.

5. AVAILABLE SOLUTIONS AS

Which solutions are available to the customers when they face the problem

or What pros & cons do these solutions have? i.e. pen and paper is an alternative to digital notetaking need to get the job done? What have they tried in the past?

- In the past User can't find the value of their car's selling price without prior knowledge about cars.
- A person with lack of knowledge about the car can also easily predict the used car price easily.

Explore AS, differentiate

Focus on J&P, tap into BE, understand RC

2. JOBS-TO-BE-DONE / PROBLEMS J&P

Which jobs-to-be-done (or problems) do you address for your customers? There could be more than one; explore different sides.

To build a supervised machine learning model using regression algorithms for forecasting the value of a vehicle based on multiple attributes such as

- Condition of Engine
- Year of Registration
- Kilometers
- Number of Owner

9. PROBLEM ROOT CAUSE RC

What is the real reason that this problem exists? What is the back story behind the need to do this job?
i.e. customers have to do it because of the change in regulations.

- The price predicted by the dealers or brokers for used car is not trustful
- Users can predict the correct value of the car remotely without human intervention like car dealers.
- User can eliminate the valuation predicted by the dealer

7. BEHAVIOUR BE

What does your customer do to address the problem and get the job done?
i.e. directly related: find the right solar panel installer, calculate usage and benefits; indirectly associated: customers spend free time on volunteering work (i.e. Greenpeace)

- The History of Your Car's condition and documents produced by them will be Suspicious.
- The model that is built would give the nearest value of the vehicle by eliminating anonymous value predicted by using humans.

Focus on J&P, tap into BE, understand RC

3. TRIGGERS TR

What triggers customers to act? i.e. seeing their neighbour installing solarpanels, reading about a more efficient solution in the news.

Users can predict the correct valuation of the car by their own like OLX cars, Cars24 and other car resale value prediction websites by using model, year, owner, etc.

10. YOUR SOLUTION SL

If you are working on an existing business, write down your current solution first, fill in the canvas, and check how much it fits reality.
If you are working on a new business proposition, then keep it blank until you fill in the canvas and come up with a solution that fits within customer limitations, solves a problem and matches customer behaviour.

- The main aim of this project is to predict the price of used cars using the Machine Learning (ML) algorithms and collection of data's about different cars.

8. CHANNELS of BEHAVIOUR CH

8.1 ONLINE
What kind of actions do customers take online? Extract online channels from #7

8.2 OFFLINE
What kind of actions do customers take offline? Extract offline channels from #7 and use them for customer development.

- Customer should predict the worth of the carby using different parameters given by the owner.

Identify strong TR & EM

4. EMOTIONS: BEFORE / AFTER EM

How do customers feel when they face a problem or a job and afterwards?
i.e. lost, insecure > confident, in control - use it in your communication strategy & design.

Before:

- User will be in fear about the biased values predicted by the humans based on the condition of the car.

After:

- User can determine the almost accurate value of the car by their own without human intervention.

The project should take parameters related to used car as inputs and enable the customers to make decisions by their own.

- User Should confirm the details provided about the vehicle in RTO online.
- By seeing the exterior and interior condition of the car, user can decide their decisions.
- User can test the performance of the car and to buy it up in a affordable price based on its condition.

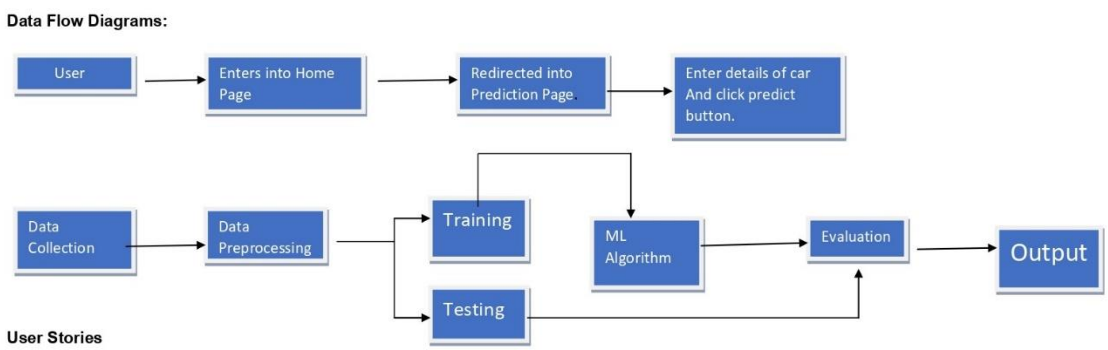
4. REQUIREMENT ANALYSIS

4.1 Functional requirement

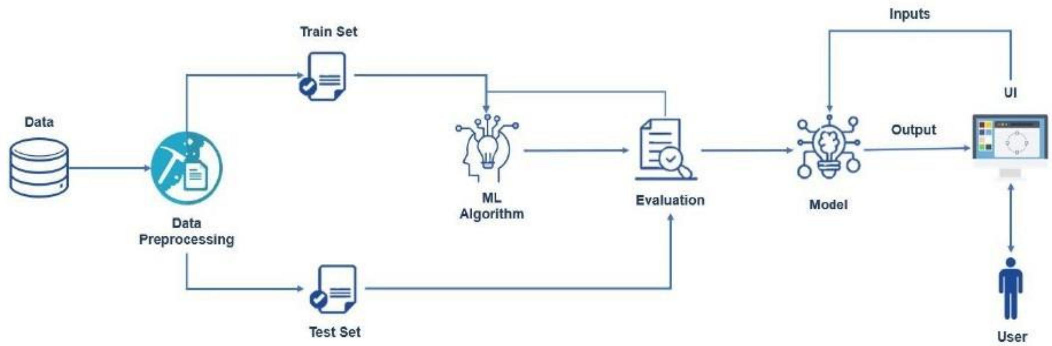
FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Website
FR-2	User Confirmation	Confirmation via Website
FR-3	Car Registration	Registration through Website
FR-4	Car Information	Getting the car details through Website
FR-5	Value Prediction	Shows the resale value of the car through website

5. PROJECT DESIGN

5.1 Data Flow Diagrams



5.2 Solution & Technical Architecture



5.3 User Stories

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Web user)	Web browser	USN-1	As a user, I can visit to the website directly	I can access the website by simply clicking available link	High	Sprint -1
		USN-2	I can move to the homepage	can able to visit the webpage without any acceptance	High	Sprint -1
		UNS-3	After reading the description of the model I can move to the prediction page by clicking the prediction button(RESALE VALUE OF YOUR CAR)	I can move to prediction page without any acceptance	High	Sprint -2

		USN-4	After filling the details in the prediction page the accurate value should be shown in the webpage.	I can get the result without any Accept ance	High	Sprint -3
Custo mer(M obile User)	Mobile app (Sign up)	USN-1	As a user can register for the application by giving email as a username and setting a password .	I can register if the username is in the correct format.	Mediu m	Sprint -4
	(Sign in)	USN-2	As a user I can login to the app by filling the username and password field	I can login to the app if the username and password matches with database	Mediu m	Sprint -5
Custo mer(M obile User)	Dashboard	USN-3	As a user I can move to the dashboard after successful login and navigate to next page	Without any acceptance I can move to the next page.	Mediu m	Sprint -5
Custo mer	Searching	USN-4	After filling the required details click predict button to get the result.	Without any acceptance I can get the result	Mediu m	Sprint -6
Custo mer Care Executi ve						
Admini strator						

6. PROJECT PLANNING & SCHEDULING

6.1 Sprint Planning & Estimation

Sprint	Functional Requirement (Epic)	Task	Team Members
Sprint-1	Dataset reading and Pre-processing	Cleaning the dataset and splitting to dependent and independent variables	R.Nandha kumar S.Manoj kumar G.Gowtham J.Stanly
Sprint-2	Building the model	Choosing the appropriate model for building and saving the model as pickle file	R.Nandha kumar S.Manoj kumar G.Gowtham J.Stanly
Sprint-3	Application building	Using flask deploying the ML model	R.Nandha kumar S.Manoj kumar G.Gowtham J.Stanly
Sprint-4	Train the model in IBM	Finally train the model on IBM cloud and deploy the application	R.Nandha kumar S.Manoj kumar G.Gowtham J.Stanly

1.1 Sprint Delivery Schedule

Sprint	**Total story points**	**Duration**	**Sprint start Date**	**Sprint End Date(planned**	**Story points completed(as on Planned End Date**	**Sprint Release Date (Actual)**
Sprint-1	15	5 Days	5 Nov 2022	29 Oct 2022	15	11 Nov 2022
Sprint-2	15	5 Days	5 Nov 2022	05 Nov 2022	15	11 Nov 2022
Sprint-3	15	5 Days	5 Nov 2022	12 Nov 2022	15	11 Nov 2022
Sprint -4	15	5 Days	5 Nov 2022	19 Nov 2022	25	11 Nov 2022

1.2 Reports from JIRA

2. CODING & SOLUTIONING (Explain the features added in the project along with code)

2.1 Feature 1

App.py

```
port flask
from flask import request, render_template
from flask_cors import CORS
import joblib

app = flask.Flask(__name__, static_url_path='')
CORS(app)

@app.route('/', methods=['GET'])
def sendHomePage():
    return render_template('index1.html')

@app.route('/predict', methods=['POST'])
def predictSpecies():
    A=float(request.form['A'])
    B=float(request.form['B'])
    C=float(request.form['C'])
    D=float(request.form['D'])
    E=float(request.form['E'])
    F=float(request.form['F'])
    G=float(request.form['G'])
    H=float(request.form['H'])
```

```
    I=float(request.form['I'])
    J=float(request.form['J'])
    K=float(request.form['K'])
    L=float(request.form['L'])
    X=[[A,B,C,D,E,
    F,G,H,I,J,K,L]]
    model = joblib.load('CRF.pkl')
    species = model.predict(X)[0]
    return render_template('predict.html',predict=species)

if __name__ == '__main__':
    app.run(debug= True)
```

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>IRIS Prediction</title>
</head>
<body>
    <h1>CAR RESALE VALUE PREDICTION</h1>
    <h2> Using Random Forest</h2>
    <h3>Made by Manoj</h3>
    <form methods="POST" action='/predict'>
        <p>abtest</p>
        <input name="abtest"required>
        <p>vechicleType</p>
        <input name="vehicleType"required>
        <p>yearOfRegistration</p>
        <input name="yearOfRegistration"required>
        <p>gearbox</p>
        <input name="gearbox"required>
        <p>powerPS</p>
        <input name="powerPS"required>
        <p>model</p>
        <input name="model"required>
        <p>kilometer</p>
        <input name="kilometer"required>
        <p>monthOfRegistration</p>
```

```
        <input name="monthOfRegistration"required>
        <p>fuelType</p>
        <input name="fuelType"required>
        <p>brand</p>
        <input name="brand"required>
        <p>notRepairedDamage</p>
        <input name="notRepairedDamage"required>
        <p>postalcode</p>
        <input name="postalCode"required>
        <br>
        <br>
        <button type="submit">Submit</button>

    </form>
</body>
</html>
```

Predict.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>IRIS Predicted category</title>
</head>
<body>
    <h1>The predicted species is</h1>
    <h1>{{predict}}</h1>
    <a href="/">Go back</a>
</body>
</html>
```

2.2 Feature 2

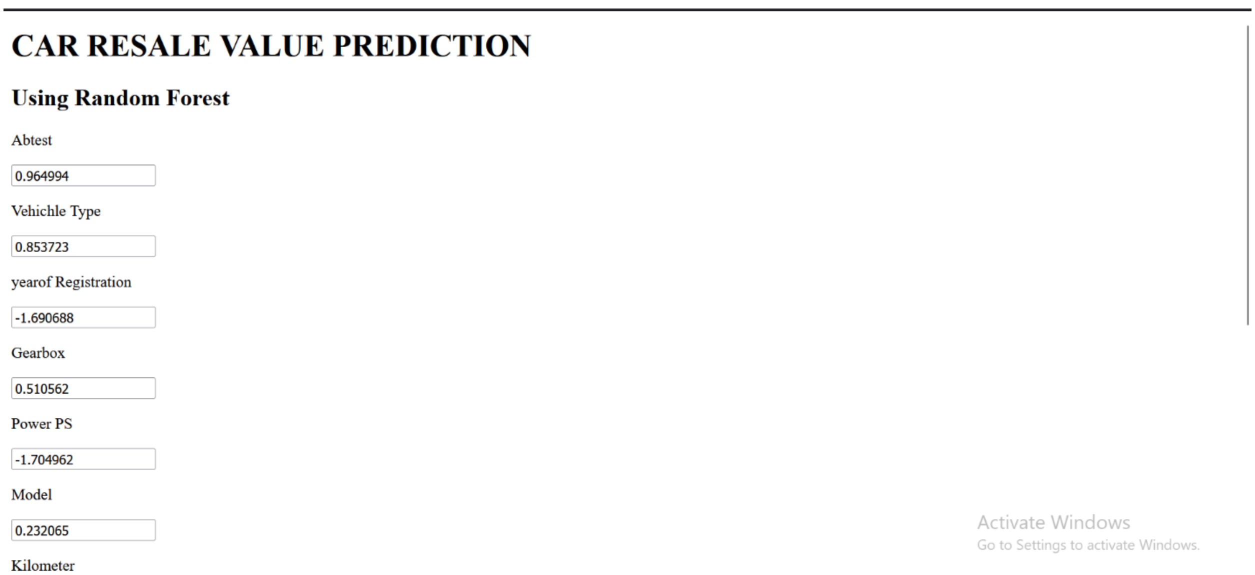
App_ibm.py

```
import flask
from flask import request, render_template
from flask_cors import CORS
import requests

import requests
```

```
# NOTE: you must manually set API_KEY below using information retrieved from your IBM Cloud account.
API_KEY = "<L0Tb5y6cXGzzoPYEMXM-XXZTmAgzVQgLJ2HTlRUlkVn9>"
token_response = requests.post('https://iam.cloud.ibm.com/identity/token', data={"apikey":
API_KEY, "grant_type": 'urn:ibm:params:oauth:grant-type:apikey'})
mltoken = token_response.json()["access_token"]

header = {'Content-Type': 'application/json', 'Authorization': 'Bearer ' + mltoken}


app = flask.Flask(__name__, static_url_path='')
CORS(app)
@app.route('/', methods=['GET'])
def sendHomePage():
   return render_template('index1.html')
@app.route('/predict', methods=['POST'])
def predictSpecies():
   a1 = float(request.form['a1'])
   b1 = float(request.form['b1'])
   c1 = float(request.form['c1'])
   d1 = float(request.form['d1'])
   e1 = float(request.form['e1'])
   f1 = float(request.form['f1'])
   g1 = float(request.form['g1'])
   h1 = float(request.form['h1'])
   i1 = float(request.form['i1'])
   j1 = float(request.form['j1'])
   k1 = float(request.form['k1'])
   l1 = float(request.form['l1'])

   X = [[a1, b1, c1, d1, e1, f1, g1, h1, i1, j1, k1, l1]]


# NOTE: manually define and pass the array(s) of values to be scored in the next line
   payload_scoring = {"input_data": [{"field": [['a1','b1','c1','d1','e1','f1','g1','h1','i1','j1','k1','l1']],
"values": X}]}

   response_scoring = requests.post('https://us-
south.ml.cloud.ibm.com/ml/v4/deployments/96c39faa-abfd-47bd-ad20-
884c3c9472ef/predictions?version=2022-11-
16',json=payload_scoring,header={'Authorization':'Bearer' + mltoken})

   print("Scoring response")
```

```
print(response_scoring)
predictions = response_scoring.json()
predict = predictions['predictions'][0]['values'][0][0]
print("Final prediction :",predict)

# showing the prediction results in a UI# showing the prediction results in a UI
return render_template('predict.html', predict=predict)

if __name__ == '__main__' :
    app.run(debug= False)
```

2.3 Database Schema (if Applicable)



3. TESTING

3.1 Test Cases

Car Resale Value Prediction jupyter notebook

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

import os, types
```

```
import pandas as pd
from botocore.client import Config
import ibm_boto3

def __iter__(self): return 0

# @hidden_cell
# The following code accesses a file in your IBM Cloud Object Storage. It includes your credentials.
# You might want to remove those credentials before you share the notebook.
cos_client = ibm_boto3.client(service_name='s3',
    ibm_api_key_id='Uq1L_fr0mGem56BJl58Ro0C5ks-jybAjXNehBzPseAZ1',
    ibm_auth_endpoint="https://iam.cloud.ibm.com/oidc/token",
    config=Config(signature_version='oauth'),
    endpoint_url='https://s3.private.us.cloud-object-storage.appdomain.cloud')

bucket = 'carresalevalue-donotdelete-pr-2lv9juqbtt5eqf'
object_key = 'autos.csv'

body = cos_client.get_object(Bucket=bucket,Key=object_key)['Body']
# add missing __iter__ method, so pandas accepts body as file-like object
if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType( __iter__, body )

df = pd.read_csv(body)
df.head()

## Read dataset

df.tail()

df.shape

## Cleaning the dataset
```

```
df.columns

# Droping the Unwanted Columns
df.drop(columns= ['seller', 'offerType', 'nrOfPictures'], inplace = True)



df.drop(columns= ['dateCrawled', 'dateCreated', 'name','lastSeen'], inplace = True)

## Missing Values

#check missing values
df.isnull().sum()

#replacing the missing values
df['vehicleType'].fillna(df['vehicleType'].mode()[0], inplace = True)
df['gearbox'].fillna(df['gearbox'].mode()[0], inplace = True)
df['model'].fillna(df['model'].mode()[0], inplace = True)
df['fuelType'].fillna(df['fuelType'].mode()[0], inplace = True)
df['notRepairedDamage'].fillna(df['notRepairedDamage'].mode()[0], inplace = True)

df.head()

df.tail()

df.isnull().sum()

## Remove the duplicates values

# Checking for Duplicates
df.duplicated().sum()

# Removing Duplicates
df = df.drop_duplicates()
```

```
df.duplicated().sum()

## label Encoding

df.info()

from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['abtest'] = le.fit_transform(df['abtest'])
df['vehicleType'] = le.fit_transform(df['vehicleType'])
df['gearbox'] = le.fit_transform(df['gearbox'])
df['model'] = le.fit_transform(df['model'])
df['fuelType'] = le.fit_transform(df['fuelType'])
df['brand'] = le.fit_transform(df['brand'])
df['notRepairedDamage'] = df['notRepairedDamage'].replace({'nein' : 0, 'ja' : 1})

df.info()

df.head()

df.hist(figsize=(20,20))

df.plot()

## Replacing the Outliers

sns.boxplot(x = df['vehicleType'])

q1=df["vehicleType"].quantile(0.25)
q3=df["vehicleType"].quantile(0.75)
q1



q3
```

```
IQR=q3-q1

upper_limit= q3 + 1.5*IQR

lower_limit= q1 - 1.5*IQR

upper_limit

lower_limit

df.median()

df["vehicleType"]= np.where(df["vehicleType"]<lower_limit,5.0,df["vehicleType"])

sns.boxplot(df["vehicleType"])

sns.boxplot(df['price'])

q1=df["price"].quantile(0.25)
q3=df["price"].quantile(0.75)
q1

q3

IQR=q3-q1

upper_limit= q3 + 1.5*IQR

lower_limit= q1 - 1.5*IQR

upper_limit
```

```
lower_limit

df["price"]= np.where(df["price"]>upper_limit,16150.0,df["price"])

sns.boxplot(df['price'])

sns.boxplot(x = df['yearOfRegistration'])

q1=df["yearOfRegistration"].quantile(0.25)
q3=df["yearOfRegistration"].quantile(0.75)
q1

q3

IQR=q3-q1

upper_limit= q3 + 1.5*IQR

lower_limit= q1 - 1.5*IQR

upper_limit

lower_limit

df["yearOfRegistration"]=
np.where(df["yearOfRegistration"]<lower_limit,2003.0,df["yearOfRegistration"])

sns.boxplot(x = df['yearOfRegistration'])

df["yearOfRegistration"]=
np.where(df["yearOfRegistration"]>upper_limit,2003.0,df["yearOfRegistration"])

sns.boxplot(x = df['yearOfRegistration'])
```

```
sns.boxplot(df['powerPS'])

q1=df["powerPS"].quantile(0.25)
q3=df["powerPS"].quantile(0.75)
q1

q3

IQR=q3-q1

upper_limit= q3 + 1.5*IQR

lower_limit= q1 - 1.5*IQR

upper_limit

lower_limit

df["powerPS"]= np.where(df["powerPS"]>upper_limit,270.0,df["powerPS"])

sns.boxplot(df['powerPS'])

sns.boxplot(df['kilometer'])

q1=df["kilometer"].quantile(0.25)
q3=df["kilometer"].quantile(0.75)
q1

q3

IQR=q3-q1

upper_limit= q3 + 1.5*IQR
```

```
lower_limit= q1 - 1.5*IQR

upper_limit

lower_limit

df["kilometer"]= np.where(df["kilometer"]<lower_limit,87500.0,df["kilometer"])

sns.boxplot(df['kilometer'])

df.head()

# Split the Data into Dependent and Independent variables.

x=df.drop(columns=['price'],axis=1)
x

y = df['price']
y

## Scaling the independent variables

from sklearn.preprocessing import scale

dfN=pd.DataFrame(scale(x),columns=x.columns)
dfN.head()

dfN.shape

plt.figure(figsize=(20,20))
sns.heatmap(dfN.corr(), annot = True)
plt.show()

sns.pairplot(dfN)
```

```
plt.show()

## Descriptive statistics

dfN.nunique()

dfN.describe()

dfN.skew()

dfN.kurt()

# Split the data into training and testing

dfN.head()

# Splitting into test and train
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(dfN, y, test_size=0.2, random_state=0)

## BUILDING MODELS

# LINEAR REGRESSION
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(x_train, y_train)

# LASSO

from sklearn.linear_model import Lasso
lasso = Lasso(alpha=0.01, normalize=True)
lasso.fit(x_train, y_train)

# RIDGE
```

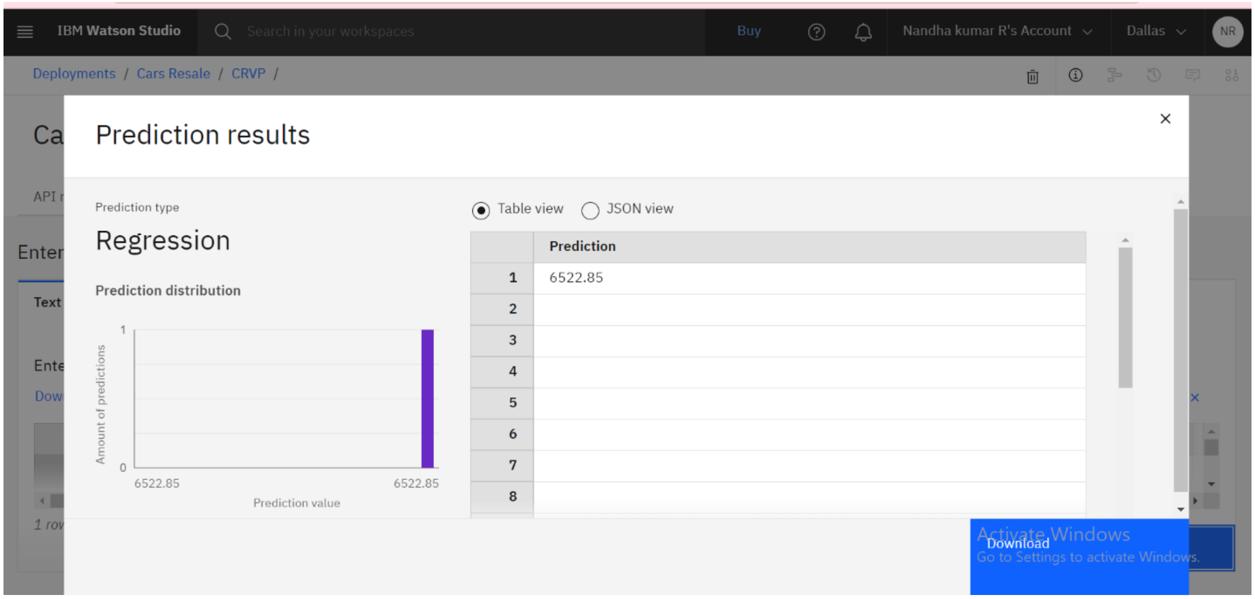
```
from sklearn.linear_model import Ridge
ridge = Ridge(alpha=0.01, normalize=True)
ridge.fit(x_train, y_train)

# Decision Tree

from sklearn.tree import DecisionTreeRegressor
DT = DecisionTreeRegressor()
DT.fit(x_train, y_train)

# KNN

from sklearn.neighbors import KNeighborsRegressor
knn = KNeighborsRegressor()
knn.fit(x_train, y_train)

# Random Forest

from sklearn.ensemble import RandomForestRegressor
RF = RandomForestRegressor()
RF.fit(x_train, y_train)

# Checking the Metrics of the models

# Linear Regression
lr.score(x_test, y_test)

from sklearn.metrics import mean_squared_error
np.sqrt(mean_squared_error(y_test,lr.predict(x_test)))

# Lasso Regression
lasso.score(x_test, y_test)
```

```
np.sqrt(mean_squared_error(y_test,lasso.predict(x_test)))

# Ridge Regression
ridge.score(x_test, y_test)

np.sqrt(mean_squared_error(y_test,ridge.predict(x_test)))

# K Nearest Neighbour
knn.score(x_test, y_test)

np.sqrt(mean_squared_error(y_test,knn.predict(x_test)))

# Decision Tree
DT.score(x_test, y_test)

np.sqrt(mean_squared_error(y_test,DT.predict(x_test)))

# Random Forest
RF.score(x_test, y_test)

np.sqrt(mean_squared_error(y_test,RF.predict(x_test)))

## IBM DEPLOYEMENT

URLS Dallas: https://us-south.ml.cloud.ibm.com

!pip install -U ibm-watson-machine-learning

from ibm_watson_machine_learning import APIClient
import json


## Authenticate and Set Space
```

```
wml_credentials = {
    "apikey":"Krx4DSuPf5HF7OqwE6HfopUaFxstdLSoFu4QzEo-ELfo",
    "url":"https://us-south.ml.cloud.ibm.com"
}

wml_client = APIClient(wml_credentials)
wml_client.spaces.list()

SPACE_ID="4e36baae-6a85-430b-b35b-d5e7876724e3"

wml_client.set.default_space(SPACE_ID)

wml_client.software_specifications.list(500)

## Save and Deploy the model

import sklearn
sklearn.__version__

MODEL_NAME = 'CRVP'
DEPLOYMENT_NAME = 'Cars Resale'
DEMO_MODEL = RF

# Set Python Version
software_spec_uid = wml_client.software_specifications.get_id_by_name('runtime-22.1-py3.9')

# Setup model meta
model_props = {
    wml_client.repository.ModelMetaNames.NAME: MODEL_NAME,
    wml_client.repository.ModelMetaNames.TYPE: 'scikit-learn_1.0',
    wml_client.repository.ModelMetaNames.SOFTWARE_SPEC_UID: software_spec_uid
}
```

```
#Save model
model_details = wml_client.repository.store_model(
    model=DEMO_MODEL,
    meta_props=model_props,
    training_data=x_train,
    training_target=y_train
)

model_details

model_id = wml_client.repository.get_model_id(model_details)
model_id

# Set meta
deployment_props = {
    wml_client.deployments.ConfigurationMetaNames.NAME:DEPLOYMENT_NAME,
    wml_client.deployments.ConfigurationMetaNames.ONLINE: {}
}

# Deploy
deployment = wml_client.deployments.create(
    artifact_uid=model_id,
    meta_props=deployment_props
)
```

3.2 User Acceptance Testing



4. RESULTS

4.1 Performance Metrics

Result of Models:

Model	MSLE	RMSLE	Accuracy
Linear regression	0.00243399	0.04933557	59.3051%
Ridge regression:	0.00243399	0.04933553	59.3051%
Lasso regression	0.00243400	0.04933566	59.305%
KNN	0.00144004	0.03794796	76.4681%
Random Forest	0.00077811	0.00077811	87.5979%
Bagging Regressor	0.00143192	0.03784080	76.809%
Adaboost Regressor	0.00084475	0.02906475	86.4084%
XGBoost Regressor	0.00065047	0.02550431	89.6623%

5. ADVANTAGES & DISADVANTAGES

ADVANTAGES

Highly Effective

DISADVANTAGES

1. Not accurate
2. Not effective

6. CONCLUSION

Using data mining and machine learning approaches, this project proposed a scalable framework for Dubai based used cars price prediction. Buyanycar.com website was scraped using the Parse Hub scraping tool to collect the benchmark data. An efficient machine learning model is built by training, testing, and evaluating three machine learning regressors named Random Forest Regressor, Linear Regression, and Bagging Regressor. As a result of pre-processing and transformation, Random Forest Regressor came out on top with 95% accuracy followed by Bagging Regressor with 88%. Each experiment was performed in real-time within the Google Colab environment. In comparison to the system's integrated Jupyter notebook and Anaconda's platform, algorithms took less training time in Google Colab.

7. FUTURE SCOPE

In future this machine learning model may bind with various website which can provide real time data for price prediction. Also we may add large historical data of car price which can help to improve accuracy of the machine learning model. We can build an android app as user interface for interacting with user. For better performance, we plan to judiciously design deep learning network structures, use adaptive learning rates and train on clusters of data rather than the whole dataset

8. APPENDIX

Source Code## Importing libraries

App.py

```
port flask
from flask import request, render_template
from flask_cors import CORS
import joblib

app = flask.Flask(__name__, static_url_path='')
CORS(app)

@app.route('/', methods=['GET'])
def sendHomePage():
    return render_template('index1.html')

@app.route('/predict', methods=['POST'])
def predictSpecies():
```

```
    A=float(request.form['A'])
    B=float(request.form['B'])
    C=float(request.form['C'])
    D=float(request.form['D'])
    E=float(request.form['E'])
    F=float(request.form['F'])
    G=float(request.form['G'])
    H=float(request.form['H'])
    I=float(request.form['I'])
    J=float(request.form['J'])
    K=float(request.form['K'])
    L=float(request.form['L'])
    X=[[A,B,C,D,E,
    F,G,H,I,J,K,L]]
    model = joblib.load('CRF.pkl')
    species = model.predict(X)[0]
    return render_template('predict.html',predict=species)

if __name__ == '__main__':
    app.run(debug= True)
```

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>IRIS Prediction</title>
</head>
<body>
    <h1>CAR RESALE VALUE PREDICTION</h1>
    <h2> Using Random Forest</h2>
    <h3>Made by Manoj</h3>
    <form methods="POST" action='/predict'>
        <p>abtest</p>
        <input name="abtest"required>
        <p>vechicleType</p>
        <input name="vehicleType"required>
        <p>yearOfRegistration</p>
        <input name="yearOfRegistration"required>
        <p>gearbox</p>
```

```
        <input name="gearbox"required>
        <p>powerPS</p>
        <input name="powerPS"required>
        <p>model</p>
        <input name="model"required>
        <p>kilometer</p>
        <input name="kilometer"required>
        <p>monthOfRegistration</p>
        <input name="monthOfRegistration"required>
        <p>fuelType</p>
        <input name="fuelType"required>
        <p>brand</p>
        <input name="brand"required>
        <p>notRepairedDamage</p>
        <input name="notRepairedDamage"required>
        <p>postalcode</p>
        <input name="postalCode"required>
        <br>
        <br>
        <button type="submit">Submit</button>

    </form>
</body>
</html>
```

Predict.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>IRIS Predicted category</title>
</head>
<body>
    <h1>The predicted species is</h1>
    <h1>{{predict}}</h1>
    <a href="/">Go back</a>
</body>
</html>
```

App_ibm.py

```
import flask
from flask import request, render_template
from flask_cors import CORS
import requests

import requests

# NOTE: you must manually set API_KEY below using information retrieved from your IBM
Cloud account.
API_KEY = "<L0Tb5y6cXGzzoPYEMXM-XXZTmAgzVQgLJ2HTlRUlkVn9>"
token_response = requests.post('https://iam.cloud.ibm.com/identity/token', data={"apikey":
API_KEY, "grant_type": 'urn:ibm:params:oauth:grant-type:apikey'})
mltoken = token_response.json()["access_token"]

header = {'Content-Type': 'application/json', 'Authorization': 'Bearer ' + mltoken}


app = flask.Flask(__name__, static_url_path='')
CORS(app)
@app.route('/', methods=['GET'])
def sendHomePage():
    return render_template('index1.html')
@app.route('/predict', methods=['POST'])
def predictSpecies():
    a1 = float(request.form['a1'])
    b1 = float(request.form['b1'])
    c1 = float(request.form['c1'])
    d1 = float(request.form['d1'])
    e1 = float(request.form['e1'])
    f1 = float(request.form['f1'])
    g1 = float(request.form['g1'])
    h1 = float(request.form['h1'])
    i1 = float(request.form['i1'])
    j1 = float(request.form['j1'])
    k1 = float(request.form['k1'])
    l1 = float(request.form['l1'])

    X = [[a1, b1, c1, d1, e1, f1, g1, h1, i1, j1, k1, l1]]


# NOTE: manually define and pass the array(s) of values to be scored in the next line
```

```
    payload_scoring = {"input_data": [{"field": [['a1','b1','c1','d1','e1','f1','g1','h1','i1','j1','k1','l1']], "values": X}]}

    response_scoring = requests.post('https://us-south.ml.cloud.ibm.com/ml/v4/deployments/96c39faa-abfd-47bd-ad20-884c3c9472ef/predictions?version=2022-11-16',json=payload_scoring,header={'Authorization':'Bearer' + mltoken})

    print("Scoring response")
    print(response_scoring)
    predictions = response_scoring.json()
    predict = predictions['predictions'][0]['values'][0][0]
    print("Final prediction :",predict)

  # showing the prediction results in a UI# showing the prediction results in a UI
    return render_template('predict.html', predict=predict)

if __name__ == '__main__' :
    app.run(debug= False)
```

Predict.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>IRIS Predicted category</title>
</head>
<body>
<h1>The predicted species is</h1>
<h1>{{predict}}</h1>
<a href="/">Go back</a>
</body>
</html>
```

Car Resale Value Prediction jupyter notebook

```
import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

import os, types
import pandas as pd
from botocore.client import Config
import ibm_boto3

def __iter__(self): return 0

# @hidden_cell
# The following code accesses a file in your IBM Cloud Object Storage. It includes your credentials.
# You might want to remove those credentials before you share the notebook.
cos_client = ibm_boto3.client(service_name='s3',
    ibm_api_key_id='Uq1L_fr0mGem56BJl58Ro0C5ks-jybAjXNehBzPseAZ1',
    ibm_auth_endpoint="https://iam.cloud.ibm.com/oidc/token",
    config=Config(signature_version='oauth'),
    endpoint_url='https://s3.private.us.cloud-object-storage.appdomain.cloud')

bucket = 'carresalevalue-donotdelete-pr-2lv9juqbtt5eqf'
object_key = 'autos.csv'

body = cos_client.get_object(Bucket=bucket,Key=object_key)['Body']
# add missing __iter__ method, so pandas accepts body as file-like object
if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType( __iter__, body )

df = pd.read_csv(body)
df.head()

## Read dataset
```

```
df.tail()

df.shape

## Cleaning the dataset

df.columns

# Droping the Unwanted Columns
df.drop(columns= ['seller', 'offerType', 'nrOfPictures'], inplace = True)



df.drop(columns= ['dateCrawled', 'dateCreated', 'name','lastSeen'], inplace = True)

## Missing Values

#check missing values
df.isnull().sum()

#replacing the missing values
df['vehicleType'].fillna(df['vehicleType'].mode()[0], inplace = True)
df['gearbox'].fillna(df['gearbox'].mode()[0], inplace = True)
df['model'].fillna(df['model'].mode()[0], inplace = True)
df['fuelType'].fillna(df['fuelType'].mode()[0], inplace = True)
df['notRepairedDamage'].fillna(df['notRepairedDamage'].mode()[0], inplace = True)

df.head()

df.tail()

df.isnull().sum()

## Remove the duplicates values
```

```
# Checking for Duplicates
df.duplicated().sum()

# Removing Duplicates
df = df.drop_duplicates()

df.duplicated().sum()

## label Encoding

df.info()

from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['abtest'] = le.fit_transform(df['abtest'])
df['vehicleType'] = le.fit_transform(df['vehicleType'])
df['gearbox'] = le.fit_transform(df['gearbox'])
df['model'] = le.fit_transform(df['model'])
df['fuelType'] = le.fit_transform(df['fuelType'])
df['brand'] = le.fit_transform(df['brand'])
df['notRepairedDamage'] = df['notRepairedDamage'].replace({'nein' : 0, 'ja' : 1})

df.info()

df.head()

df.hist(figsize=(20,20))

df.plot()

## Replacing the Outliers

sns.boxplot(x = df['vehicleType'])
```

```
q1=df["vehicleType"].quantile(0.25)
q3=df["vehicleType"].quantile(0.75)
q1


q3

IQR=q3-q1

upper_limit= q3 + 1.5*IQR

lower_limit= q1 - 1.5*IQR

upper_limit

lower_limit

df.median()

df["vehicleType"]= np.where(df["vehicleType"]<lower_limit,5.0,df["vehicleType"])

sns.boxplot(df["vehicleType"])

sns.boxplot(df['price'])

q1=df["price"].quantile(0.25)
q3=df["price"].quantile(0.75)
q1

q3

IQR=q3-q1

upper_limit= q3 + 1.5*IQR
```

```
lower_limit= q1 - 1.5*IQR

upper_limit


lower_limit

df["price"]= np.where(df["price"]>upper_limit,16150.0,df["price"])

sns.boxplot(df['price'])

sns.boxplot(x = df['yearOfRegistration'])

q1=df["yearOfRegistration"].quantile(0.25)
q3=df["yearOfRegistration"].quantile(0.75)
q1

q3

IQR=q3-q1

upper_limit= q3 + 1.5*IQR

lower_limit= q1 - 1.5*IQR

upper_limit

lower_limit

df["yearOfRegistration"]=
np.where(df["yearOfRegistration"]<lower_limit,2003.0,df["yearOfRegistration"])

sns.boxplot(x = df['yearOfRegistration'])
```

```
df["yearOfRegistration"]=
np.where(df["yearOfRegistration"]>upper_limit,2003.0,df["yearOfRegistration"])

sns.boxplot(x = df['yearOfRegistration'])

sns.boxplot(df['powerPS'])

q1=df["powerPS"].quantile(0.25)
q3=df["powerPS"].quantile(0.75)
q1

q3

IQR=q3-q1

upper_limit= q3 + 1.5*IQR

lower_limit= q1 - 1.5*IQR

upper_limit

lower_limit

df["powerPS"]= np.where(df["powerPS"]>upper_limit,270.0,df["powerPS"])

sns.boxplot(df['powerPS'])

sns.boxplot(df['kilometer'])

q1=df["kilometer"].quantile(0.25)
q3=df["kilometer"].quantile(0.75)
q1
```

```
q3

IQR=q3-q1

upper_limit= q3 + 1.5*IQR

lower_limit= q1 - 1.5*IQR

upper_limit

lower_limit

df["kilometer"]= np.where(df["kilometer"]<lower_limit,87500.0,df["kilometer"])

sns.boxplot(df['kilometer'])

df.head()

# Split the Data into Dependent and Independent variables.

x=df.drop(columns=['price'],axis=1)
x

y = df['price']
y

## Scaling the independent variables

from sklearn.preprocessing import scale

dfN=pd.DataFrame(scale(x),columns=x.columns)
dfN.head()

dfN.shape
```

```
plt.figure(figsize=(20,20))
sns.heatmap(dfN.corr(), annot = True)
plt.show()

sns.pairplot(dfN)
plt.show()

## Descriptive statistics

dfN.nunique()

dfN.describe()

dfN.skew()

dfN.kurt()

# Split the data into training and testing

dfN.head()

# Splitting into test and train
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(dfN, y, test_size=0.2, random_state=0)

## BUILDING MODELS

# LINEAR REGRESSION
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(x_train, y_train)

# LASSO
```

```
from sklearn.linear_model import Lasso
lasso = Lasso(alpha=0.01, normalize=True)
lasso.fit(x_train, y_train)

# RIDGE

from sklearn.linear_model import Ridge
ridge = Ridge(alpha=0.01, normalize=True)
ridge.fit(x_train, y_train)

# Decision Tree

from sklearn.tree import DecisionTreeRegressor
DT = DecisionTreeRegressor()
DT.fit(x_train, y_train)

# KNN

from sklearn.neighbors import KNeighborsRegressor
knn = KNeighborsRegressor()
knn.fit(x_train, y_train)

# Random Forest

from sklearn.ensemble import RandomForestRegressor
RF = RandomForestRegressor()
RF.fit(x_train, y_train)

# Checking the Metrics of the models

# Linear Regression
lr.score(x_test, y_test)
```

```
from sklearn.metrics import mean_squared_error
np.sqrt(mean_squared_error(y_test,lr.predict(x_test)))

# Lasso Regression
lasso.score(x_test, y_test)

np.sqrt(mean_squared_error(y_test,lasso.predict(x_test)))

# Ridge Regression
ridge.score(x_test, y_test)

np.sqrt(mean_squared_error(y_test,ridge.predict(x_test)))

# K Nearest Neighbour
knn.score(x_test, y_test)

np.sqrt(mean_squared_error(y_test,knn.predict(x_test)))

# Decision Tree
DT.score(x_test, y_test)

np.sqrt(mean_squared_error(y_test,DT.predict(x_test)))

# Random Forest
RF.score(x_test, y_test)

np.sqrt(mean_squared_error(y_test,RF.predict(x_test)))

## IBM DEPLOYEMENT

URLS Dallas: https://us-south.ml.cloud.ibm.com

!pip install -U ibm-watson-machine-learning
```

```
from ibm_watson_machine_learning import APIClient
import json


## Authenticate and Set Space

wml_credentials = {
    "apikey":"Krx4DSuPf5HF7OqwE6HfopUaFxstdLSoFu4QzEo-ELfo",
    "url":"https://us-south.ml.cloud.ibm.com"
}

wml_client = APIClient(wml_credentials)
wml_client.spaces.list()

SPACE_ID="4e36baae-6a85-430b-b35b-d5e7876724e3"

wml_client.set.default_space(SPACE_ID)

wml_client.software_specifications.list(500)

## Save and Deploy the model

import sklearn
sklearn.__version__

MODEL_NAME = 'CRVP'
DEPLOYMENT_NAME = 'Cars Resale'
DEMO_MODEL = RF

# Set Python Version
software_spec_uid = wml_client.software_specifications.get_id_by_name('runtime-22.1-py3.9')

# Setup model meta
```

```
model_props = {
    wml_client.repository.ModelMetaNames.NAME: MODEL_NAME,
    wml_client.repository.ModelMetaNames.TYPE: 'scikit-learn_1.0',
    wml_client.repository.ModelMetaNames.SOFTWARE_SPEC_UID: software_spec_uid
}

#Save model
model_details = wml_client.repository.store_model(
    model=DEMO_MODEL,
    meta_props=model_props,
    training_data=x_train,
    training_target=y_train
)

model_details

model_id = wml_client.repository.get_model_id(model_details)
model_id

# Set meta
deployment_props = {
    wml_client.deployments.ConfigurationMetaNames.NAME:DEPLOYMENT_NAME,
    wml_client.deployments.ConfigurationMetaNames.ONLINE: {}
}

# Deploy
deployment = wml_client.deployments.create(
    artifact_uid=model_id,
    meta_props=deployment_props
)
```

GitHub Link : https: //github.com/IBM-EPBL/IBM-Project-29766-1660129290

Project Demo Link: https://youtu.be/94ZRph7b3A4