Question 1 Cars Prediction - Regression

Team Members

Dhrish S Kumar - E0320008
A Ch Rohit - E0320022

Dataset - Cars Data

Importing Libraries

In [49]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:

```
df=pd.read_csv("Cars_data.csv")
df.head()
```

Out[2]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	en
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	
4	5	2	audi 100ls	gas	std	four	sedan	4wd	

5 rows × 26 columns

In [3]:

```
df.shape
```

Out[3]:

```
(205, 26)
```

Data Cleaning and Data Preparation

In [4]:

```
df.describe()
```

Out[4]:

	car_ID	symboling	wheelbase	carlength	carwidth	carheight	curbweight	e
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	2
mean	103.000000	0.834146	98.756585	174.049268	65.907805	53.724878	2555.565854	1
std	59.322565	1.245307	6.021776	12.337289	2.145204	2.443522	520.680204	
min	1.000000	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	
25%	52.000000	0.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	
50%	103.000000	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	1
75%	154.000000	2.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	1
max	205.000000	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	3

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   car_ID            205 non-null    int64  
 1   symboling         205 non-null    int64  
 2   CarName           205 non-null    object 
 3   fueltype          205 non-null    object 
 4   aspiration        205 non-null    object 
 5   doornumber        205 non-null    object 
 6   carbody           205 non-null    object 
 7   drivewheel        205 non-null    object 
 8   enginelocation    205 non-null    object 
 9   wheelbase         205 non-null    float64
 10  carlength         205 non-null    float64
 11  carwidth          205 non-null    float64
 12  carheight         205 non-null    float64
 13  curbweight        205 non-null    int64  
 14  enginetype        205 non-null    object 
 15  cylindernumber    205 non-null    object 
 16  enginesize        205 non-null    int64  
 17  fuelsystem        205 non-null    object 
 18  boreratio         205 non-null    float64
 19  stroke            205 non-null    float64
 20  compressionratio  205 non-null    float64
 21  horsepower        205 non-null    int64  
 22  peakrpm           205 non-null    int64  
 23  citympg           205 non-null    int64  
 24  highwaympg        205 non-null    int64  
 25  price             205 non-null    float64
dtypes: float64(8), int64(8), object(10)
memory usage: 41.8+ KB
```

In [6]:

```
df.isna().any()
```

Out[6]:

```
car_ID              False
symboling           False
CarName             False
fueltype            False
aspiration          False
doornumber          False
carbody             False
drivewheel          False
enginelocation      False
wheelbase           False
carlength           False
carwidth            False
carheight           False
curbweight          False
enginetype          False
cylindernumber      False
enginesize          False
fuelsystem          False
boreratio           False
stroke              False
compressionratio    False
horsepower          False
peakrpm             False
citympg             False
highwaympg          False
price               False
dtype: bool
```

In [7]:

```
CompanyName = df['CarName'].apply(lambda x : x.split(' ')[0])
df.insert(3,"CompanyName",CompanyName)
df=df.drop(["car_ID","CarName"],axis=1)
df.head()
```

Out[7]:

	symboling	CompanyName	fueltype	aspiration	doornumber	carbody	drivewheel	enginel
0	3	alfa-romero	gas	std	two	convertible	rwd	
1	3	alfa-romero	gas	std	two	convertible	rwd	
2	1	alfa-romero	gas	std	two	hatchback	rwd	
3	2	audi	gas	std	four	sedan	fwd	
4	2	audi	gas	std	four	sedan	4wd	

5 rows × 25 columns

In [8]:

```
df.columns
```

Out[8]:

```
Index(['symboling', 'CompanyName', 'fueltype', 'aspiration', 'doornumber',
       'carbody', 'drivewheel', 'enginelocation', 'wheelbase', 'carlength',
       'carwidth', 'carheight', 'curbweight', 'enginetype', 'cylindernumbe
r',
       'enginesize', 'fuelsystem', 'boreratio', 'stroke', 'compressionrati
o',
       'horsepower', 'peakrpm', 'citympg', 'highwaympg', 'price'],
      dtype='object')
```

In [9]:

```
x=df.iloc[:,:-1]
y=df.iloc[:,-1:]
x
```

Out[9]:

	symboling	CompanyName	fueltype	aspiration	doornumber	carbody	drivewheel	engi
0	3	alfa-romero	gas	std	two	convertible	rwd	
1	3	alfa-romero	gas	std	two	convertible	rwd	
2	1	alfa-romero	gas	std	two	hatchback	rwd	
3	2	audi	gas	std	four	sedan	fwd	
4	2	audi	gas	std	four	sedan	4wd	
...	...	...	...	...	...	...	...	
200	-1	volvo	gas	std	four	sedan	rwd	
201	-1	volvo	gas	turbo	four	sedan	rwd	
202	-1	volvo	gas	std	four	sedan	rwd	
203	-1	volvo	diesel	turbo	four	sedan	rwd	
204	-1	volvo	gas	turbo	four	sedan	rwd	

205 rows × 24 columns

In [10]:

```
y
```

Out[10]:

	price
0	13495.0
1	16500.0
2	16500.0
3	13950.0
4	17450.0
...	...
200	16845.0
201	19045.0
202	21485.0
203	22470.0
204	22625.0

205 rows × 1 columns

In [11]:

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
columns=['CompanyName', 'fueltype', 'aspiration', 'doornumber','carbody', 'drivewheel', 'en
for col in columns:
    x[col]=le.fit_transform(x[col])
x.head()
```

Out[11]:

	symboling	CompanyName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelo
0	3	1	1	0	1	0	2	
1	3	1	1	0	1	0	2	
2	1	1	1	0	1	2	2	
3	2	2	1	0	0	3	1	
4	2	2	1	0	0	3	0	

5 rows × 24 columns

In [56]:

```
from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
scaled_x=ss.fit_transform(x)
scaled_x=pd.DataFrame(scaled_x,columns=x.columns)
```

Data Splitting

In [13]:

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(scaled_x,y,test_size=0.25,random_state=0)
```

Building Models

Linear Regression Model

In [69]:

```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
lr_pred=lr.predict(x_test)
lr_pred
```

Out[69]:

```
array([[ 6382.93026691],
       [17389.92005472],
       [14147.89198415],
       [ 1055.21610004],
       [ 9643.5641032 ],
       [13324.89135827],
       [ 6668.40772215],
       [ 4562.86338541],
       [20462.78703906],
       [ 7906.1721239 ],
       [16331.05257632],
       [27464.71725046],
       [10677.49560596],
       [13866.34759394],
       [ 7162.72443096],
       [11821.54349773],
       [ 8414.97828177],
       [20233.93897444],
       [ 9680.96283176],
       [ 7075.97329649],
       [10673.71428038],
       [20838.10685109],
       [ 8719.80441669],
       [12445.8460476 ],
       [19924.73248781],
       [ 6674.49399411],
       [ 6283.73219484],
       [21349.08936839],
       [ 7416.12154699],
       [ 6652.02883374],
       [ 6902.91267079],
       [ 9526.67131   ],
       [16949.75397355],
       [ 8267.99378763],
       [ 6470.18037542],
       [24175.36763109],
       [ 9269.28879202],
       [14938.53639148],
       [ 7111.12186591],
       [37080.36185509],
       [ 5784.53938397],
       [15366.27116196],
       [33226.56994865],
       [18996.97787541],
       [ 9408.66324165],
       [ 7763.32106284],
       [ 8057.38599825],
       [13708.24870354],
       [ 8530.16769584],
       [ 8705.11656054],
```

```
       [22758.91239044],
       [ 4481.30828962]])
```

Polynomial Regression Model

In [82]:

```
from sklearn.preprocessing import PolynomialFeatures
poly=PolynomialFeatures(degree=3)
x_poly_train=poly.fit_transform(x_train)
x_poly_test=poly.fit_transform(x_test)
from sklearn.linear_model import LinearRegression
upd_lr=LinearRegression()
upd_lr.fit(x_poly_train,y_train)
upd_lr_pred=upd_lr.predict(x_poly_test)
upd_lr_pred
```

Out[82]:

```
array([[ 6.07325136e+03],
       [ 2.62191510e+04],
       [ 1.71296078e+04],
       [-6.39781867e+04],
       [ 8.54878371e+03],
       [-3.16181083e+04],
       [-2.07041402e+03],
       [ 6.07397857e+04],
       [-6.60222244e+03],
       [ 6.69676939e+03],
       [ 1.40068140e+04],
       [-8.64583369e+04],
       [ 1.18450000e+04],
       [ 1.49394497e+04],
       [ 5.65146020e+03],
       [-4.38075532e+04],
       [ 2.40088903e+04],
       [ 5.20296416e+03],
```

Ridge Model

In [16]:

```
# import ridge regression from sklearn library
from sklearn.linear_model import Ridge

# Train the model
ridgeR = Ridge(alpha = 1)
ridgeR.fit(x_train, y_train)
ridge_pred = ridgeR.predict(x_test)
ridge_pred
```

Out[16]:

```
array([[ 6358.44614549],
       [17440.39195225],
       [14155.4858299 ],
       [ 1094.29486588],
       [ 9444.28202073],
       [13259.22785382],
       [ 6744.67629258],
       [ 4518.68591609],
       [20456.34177558],
       [ 7834.28489457],
       [16192.09535612],
       [27555.47676115],
       [10741.10521359],
       [13894.93066317],
       [ 7125.90996635],
       [11783.96733862],
       [ 8305.4927869 ],
       [20223.25585318],
```

In [17]:

```
# get ridge coefficient and print them
ridge_coefficient = pd.DataFrame()
x_columns=np.array(x.columns)
x_columns
ridge_coefficient["Columns"]= scaled_x.columns
ridge_coefficient['Coefficient Estimate'] = pd.Series(ridgeR.coef_[0])
print(ridge_coefficient)
```

```
             Columns  Coefficient Estimate
0          symboling            371.472095
1        CompanyName          -1006.589840
2           fueltype            588.528405
3         aspiration            309.658564
4         doornumber           -745.679839
5            carbody           -865.223825
6         drivewheel            771.085001
7     enginelocation           1581.530307
8          wheelbase           1004.809047
9          carlength           -217.506844
10          carwidth           1360.346452
11         carheight            257.126638
12        curbweight           1889.529768
13        enginetype            121.348868
14    cylindernumber             54.226443
15        enginesize           2909.356938
16        fuelsystem            -50.353578
17         boreratio           -761.628260
18            stroke          -1032.832219
19  compressionratio            839.516966
20        horsepower            632.390614
21           peakrpm            577.911330
22           citympg           -452.168134
23        highwaympg            336.357234
```

Lasso Model

In [18]:

```
# import Lasso regression from sklearn library
from sklearn.linear_model import Lasso

# Train the model
lasso = Lasso(alpha = 1)
lasso.fit(x_train, y_train)
lasso_pred = lasso.predict(x_test)

lasso_pred
```

Out[18]:

```
array([ 6377.55309742, 17382.13008318, 14141.00301575,  1089.91874323,
        9584.23068332, 13293.19257242,  6696.30576976,  4566.02365282,
       20463.05058781,  7885.26484324, 16299.49455651, 27298.28151421,
       10681.75838449, 13837.29208503,  7153.15456727, 11793.23190651,
        8410.26291931, 20240.57970209,  9677.12587005,  7045.2286475 ,
       10705.98403645, 20861.71092231,  8713.82127365, 12411.181727  ,
       19911.7906416 ,  6686.70665468,  6278.90654725, 21383.985045  ,
        7412.54171916,  6640.70342394,  6908.41744962,  9551.91644129,
       16865.01985886,  8287.40286761,  6451.98115969, 24246.17203295,
        9277.73375605, 14887.22980761,  7080.29349087, 37112.43729015,
        5785.24092399, 15381.45418627, 33243.9677781 , 19043.89828864,
        9444.87980973,  7751.65479677,  8007.87215583, 13733.26367419,
        8612.65410185,  8695.19537122, 22747.11148844,  4615.76087672])
```

In [19]:

```
lasso_coeff = pd.DataFrame()
lasso_coeff["Columns"] = scaled_x.columns
lasso_coeff['Coefficient Estimate'] = pd.Series(lasso.coef_)

print(lasso_coeff)
```

```
             Columns  Coefficient Estimate
0          symboling            374.344469
1        CompanyName          -1019.497880
2           fueltype           1316.135024
3         aspiration            430.921041
4         doornumber           -762.548167
5            carbody           -869.580259
6         drivewheel            768.999903
7     enginelocation           1596.444838
8          wheelbase           1022.602230
9          carlength           -334.539522
10          carwidth           1379.135682
11         carheight            276.438827
12        curbweight           1969.131978
13        enginetype            105.401678
14    cylindernumber             61.361518
15        enginesize           3074.864863
16        fuelsystem            -27.384499
17         boreratio           -759.951935
18            stroke          -1032.923690
19  compressionratio           1498.747855
20        horsepower            386.765744
21           peakrpm            598.805692
22           citympg           -713.907245
23        highwaympg            532.627150
```

Performance Evaluation

Calculating and displaying MSE, RMSE, R^2 for each model

In [48]:

```
# calculate mean square error linear regression
from sklearn.metrics import r2_score
linear_mse = np.mean((lr_pred - y_test)**2)
linear_rmse=linear_mse**0.5
linear_r2=r2_score(y_test,lr_pred)

print("MSE for Linear Regression is ",linear_mse)
print("RMSE for Linear Regression is",linear_rmse)
print("R square for Linear Regression is",linear_r2)
```

```
MSE for Linear Regression is  price    1.271327e+07
dtype: float64
RMSE for Linear Regression is price    3565.567027
dtype: float64
R square for Linear Regression is 0.8295077818465942
```

In [47]:

```
# calculate mean square error polynomial regression
poly_mse = np.mean((upd_lr_pred - y_test)**2)
poly_rmse=poly_mse**0.5
poly_r2=r2_score(y_test,upd_lr_pred)

print("MSE for Polynomial Regression is ",poly_mse)
print("RMSE for Polynomial Regression is",poly_rmse)
print("R square for Polynomial Regression is",poly_r2)
```

```
MSE for Polynomial Regression is  price    1.716145e+09
dtype: float64
RMSE for Polynomial Regression is price    41426.379119
dtype: float64
R square for Polynomial Regression is -22.014487173526938
```

In [45]:

```
# calculate mean square error ridge
ridge_mse = np.mean((ridge_pred - y_test)**2)
ridge_rmse=ridge_mse**0.5
ridge_r2=r2_score(y_test,ridge_pred)


print("MSE for Ridge Regression is ",ridge_mse)
print("RMSE for Ridge Regression is",ridge_rmse)
print("R square for Ridge Regression is",ridge_r2)
```

```
MSE for Ridge Regression is  price    1.252677e+07
dtype: float64
RMSE for Ridge Regression is price    3539.317707
dtype: float64
R square for Ridge Regression is 0.8320088322071963
```

In [46]:

```
# Calculate Mean Squared Error Lasso
lasso_mse = np.mean((lasso_pred - np.array(y_test))**2)
lasso_rmse=lasso_mse**0.5
lasso_r2=r2_score(y_test,lasso_pred)

print("MSE for Lasso Regression is ",lasso_mse)
print("RMSE for Lasso Regression is",lasso_rmse)
print("R square for Lasso Regression is",lasso_r2)
```

```
MSE for Lasso Regression is  129204982.17189582
RMSE for Lasso Regression is 11366.836946657404
R square for Lasso Regression is 0.8298246115693888
```

In [ ]:

```

```

Question 2 Horse racing tips classification

Team Members

Dhrish S Kumar - E0320008
A Ch Rohit - E0320022

Dataset - Horse racing tips

Importing Libraries

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:

```
df=pd.read_csv("tips.csv")
df.head()
```

Out[2]:

	UID	ID	Tipster	Date	Track	Horse	Bet Type	Odds	Result	TipsterActive
0	1	1	Tipster A	24/07/2015	Ascot	Fredricka	Win	8.00	Lose	True
1	2	2	Tipster A	24/07/2015	Thirsk	Spend A Penny	Win	4.50	Lose	True
2	3	3	Tipster A	24/07/2015	York	Straightothepoint	Win	7.00	Lose	True
3	4	4	Tipster A	24/07/2015	Newmarket	Miss Inga Sock	Win	5.00	Lose	True
4	5	5	Tipster A	25/07/2015	Ascot	Peril	Win	4.33	Win	True

In [3]:

```
df.shape
```

Out[3]:

```
(38248, 10)
```

Data Cleaning and Data Preparation

In [4]:

```
df.describe()
```

Out[4]:

	UID	ID	Odds
count	38248.000000	38248.000000	38248.000000
mean	19124.500000	1013.308251	10.994968
std	11041.390885	917.941098	11.004589
min	1.000000	1.000000	1.070000
25%	9562.750000	318.000000	5.000000
50%	19124.500000	749.000000	8.000000
75%	28686.250000	1419.000000	13.000000
max	38248.000000	4383.000000	407.000000

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 38248 entries, 0 to 38247
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   UID            38248 non-null  int64  
 1   ID             38248 non-null  int64  
 2   Tipster        38248 non-null  object 
 3   Date           38248 non-null  object 
 4   Track          38248 non-null  object 
 5   Horse          38248 non-null  object 
 6   Bet Type       38248 non-null  object 
 7   Odds           38248 non-null  float64
 8   Result         38248 non-null  object 
 9   TipsterActive  38248 non-null  bool   
dtypes: bool(1), float64(1), int64(2), object(6)
memory usage: 2.7+ MB
```

In [6]:

```
df.isna().any()
```

Out[6]:

```
UID              False
ID               False
Tipster          False
Date             False
Track            False
Horse            False
Bet Type         False
Odds             False
Result           False
TipsterActive    False
dtype: bool
```

In [7]:

```
x=df.iloc[:,[2,3,4,5,6,7,9]]
y=df.iloc[:,8:9]
x
```

Out[7]:

	Tipster	Date	Track	Horse	Bet Type	Odds	TipsterActive
0	Tipster A	24/07/2015	Ascot	Fredricka	Win	8.00	True
1	Tipster A	24/07/2015	Thirsk	Spend A Penny	Win	4.50	True
2	Tipster A	24/07/2015	York	Straightothepoint	Win	7.00	True
3	Tipster A	24/07/2015	Newmarket	Miss Inga Sock	Win	5.00	True
4	Tipster A	25/07/2015	Ascot	Peril	Win	4.33	True
...	...	...	...	...	...	...	...
38243	Tipster E1	02/04/2016	Kempton	Solar Flair	Win	7.00	False
38244	Tipster E1	02/04/2016	Doncaster	Express Himself	Each Way	12.00	False
38245	Tipster E1	02/04/2016	Doncaster	Jack Dexter	Win	7.00	False
38246	Tipster E1	02/04/2016	Kelso	Just Cameron	Win	4.33	False
38247	Tipster E1	31/05/2016	Redcar	Dream Farr	Win	5.00	False

38248 rows × 7 columns

In [8]:

```
y
```

Out[8]:

	Result
0	Lose
1	Lose
2	Lose
3	Lose
4	Win
...	...
38243	Lose
38244	Lose
38245	Lose
38246	Lose
38247	Lose

38248 rows × 1 columns

In [9]:

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
columns=["Tipster","Date","Track","Horse","Bet Type","TipsterActive"]
for col in columns:
    x[col]=le.fit_transform(x[col])
x.head()
```

```
<ipython-input-9-91da4de41028>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
  x[col]=le.fit_transform(x[col])
```

Out[9]:

	Tipster	Date	Track	Horse	Bet Type	Odds	TipsterActive
0	0	818	2	5157	1	8.00	1
1	0	818	96	13103	1	4.50	1
2	0	818	114	13406	1	7.00	1
3	0	818	74	8974	1	5.00	1
4	0	851	2	10550	1	4.33	1

In [10]:

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
y=le.fit_transform(y)
y=pd.DataFrame(y,columns=["Result"])
y.head()
```

```
C:\Users\Rohit Anand\anaconda3\lib\site-packages\sklearn\utils\validation.p
y:63: DataConversionWarning: A column-vector y was passed when a 1d array wa
s expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
```

Out[10]:

	Result
0	0
1	0
2	0
3	0
4	1

In [11]:

```
from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
scaled_x=ss.fit_transform(x)
scaled_x=pd.DataFrame(scaled_x,columns=x.columns)
```

Data Splitting

In [12]:

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(scaled_x,y,test_size=0.30,random_state=0)
```

Building Models

Logisitc Regression Model

In [13]:

```
from sklearn.linear_model import LogisticRegression
log=LogisticRegression()
log.fit(x_train,y_train)
log_pred=log.predict(x_test)
log_pred=log_pred.reshape(len(log_pred),1)
log_predicted=pd.DataFrame(log_pred,columns=['Predicted'])
log_predicted
```

```
C:\Users\Rohit Anand\anaconda3\lib\site-packages\sklearn\utils\validation.p
y:63: DataConversionWarning: A column-vector y was passed when a 1d array wa
s expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
```

Out[13]:

	Predicted
0	0
1	0
2	0
3	0
4	0
...	...
11470	0
11471	0
11472	0
11473	0
11474	0

11475 rows × 1 columns

KNN (K Nearest Neighbors) Model

In [14]:

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=5,metric='minkowski',p=2)
knn.fit(x_train,y_train)
knn_pred=knn.predict(x_test)
knn_pred
```

```
C:\Users\Rohit Anand\anaconda3\lib\site-packages\sklearn\neighbors\_classifi
cation.py:179: DataConversionWarning: A column-vector y was passed when a 1d
array was expected. Please change the shape of y to (n_samples,), for exampl
e using ravel().
  return self._fit(X, y)
```

Out[14]:

```
array([0, 0, 1, ..., 0, 1, 0])
```

Decision Tree Model

In [16]:

```
from sklearn.tree import DecisionTreeClassifier
decision_tree=DecisionTreeClassifier(criterion='entropy',random_state=0)
decision_tree.fit(x_train,y_train)
dt_pred=decision_tree.predict(x_test)
print(dt_pred)
print(np.concatenate((dt_pred.reshape(-1,1),np.array(y_test).reshape(-1,1)),1))
```

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

Naive Bayes Model

In [17]:

```
from sklearn.naive_bayes import GaussianNB
NB = GaussianNB()
NB.fit(x_train, y_train)
NB_pred = NB.predict(x_test)
NB_pred
```

```
C:\Users\Rohit Anand\anaconda3\lib\site-packages\sklearn\utils\validation.p
y:63: DataConversionWarning: A column-vector y was passed when a 1d array wa
s expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  return f(*args, **kwargs)
```

Out[17]:

```
array([0, 0, 0, ..., 0, 0, 0])
```

Performance Evaluation

Calculating and displaying classification report, confusion report, accuracy for Logistic Regression model

In [18]:

```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
log_cr=classification_report(y_test,log_pred)
log_conf=confusion_matrix(y_test,log_pred)
log_ac=accuracy_score(y_test,log_pred)
print("\t\tLogistic Regression's Classification Report\n\n",log_cr)
print("\t\tLogistic Regression's Confusion Matrix\n\n",log_conf)
print("\nLogistic Regression's Accuracy Score is ",log_ac)
```

```
               Logistic Regression's Classification Report

              precision    recall  f1-score   support

           0       0.81      1.00      0.89      9237
           1       0.52      0.01      0.01      2238

    accuracy                           0.81     11475
   macro avg       0.66      0.50      0.45     11475
weighted avg       0.75      0.81      0.72     11475

               Logistic Regression's Confusion Matrix

 [[9222   15]
 [2222   16]]

Logistic Regression's Accuracy Score is  0.8050544662309368
```

Calculating and displaying classification report, confusion report, accuracy for KNN Classification model

In [19]:

```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
knn_cr=classification_report(y_test,knn_pred)
knn_conf=confusion_matrix(y_test,knn_pred)
knn_ac=accuracy_score(y_test,knn_pred)
print("\t\tKNN's Classification Report\n\n",knn_cr)
print("\t\tKNN's Confusion Matrix\n\n",knn_conf)
print("\nKNN's Accuracy Score is ",knn_ac)
```

```
               KNN's Classification Report

              precision    recall  f1-score   support

           0       0.82      0.92      0.87      9237
           1       0.30      0.14      0.19      2238

    accuracy                           0.77     11475
   macro avg       0.56      0.53      0.53     11475
weighted avg       0.72      0.77      0.73     11475

               KNN's Confusion Matrix

 [[8538  699]
 [1934  304]]

KNN's Accuracy Score is  0.7705446623093682
```

Calculating and displaying classification report, confusion report, accuracy for Decision Tree Classification model

In [20]:

```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
dt_cr=classification_report(y_test,dt_pred)
dt_conf=confusion_matrix(y_test,dt_pred)
dt_ac=accuracy_score(y_test,dt_pred)
print("\t\tDecision Tree's Classification Report\n\n",dt_cr)
print("\t\tDecision Tree's  Confusion Matrix\n\n",dt_conf)
print("\nDecision Tree's Accuracy Score is ",dt_ac)
```

```
                Decision Tree's Classification Report

               precision    recall  f1-score   support

           0       0.83      0.81      0.82      9237
           1       0.29      0.31      0.30      2238

    accuracy                           0.71     11475
   macro avg       0.56      0.56      0.56     11475
weighted avg       0.72      0.71      0.72     11475

                Decision Tree's  Confusion Matrix

 [[7493 1744]
 [1539  699]]

Decision Tree's Accuracy Score is  0.7138997821350762
```

Calculating and displaying classification report, confusion report, accuracy for Naive Bayes Classification model

In [21]:

```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
NB_cr=classification_report(y_test,NB_pred)
NB_conf=confusion_matrix(y_test,NB_pred)
NB_ac=accuracy_score(y_test,NB_pred)
print("\t\tNaive Bayes's Classification Report\n\n",NB_cr)
print("\t\tNaive Bayes's  Confusion Matrix\n\n",NB_conf)
print("\nNaive Bayes's Accuracy Score is ",NB_ac)
```

```
                Naive Bayes's Classification Report

               precision    recall  f1-score   support

           0       0.81      1.00      0.89      9237
           1       0.41      0.01      0.02      2238

    accuracy                           0.80     11475
   macro avg       0.61      0.50      0.46     11475
weighted avg       0.73      0.80      0.72     11475

                Naive Bayes's  Confusion Matrix

 [[9200   37]
 [2212   26]]

Naive Bayes's Accuracy Score is  0.8040087145969499
```

In [ ]:

```
```

Question 3 Patient Charges - Clustering

In machine learning too, we often group examples as a first step to understand a subject (data set) in a machine learning system. Grouping unlabeled examples is called clustering. As the examples are unlabeled, clustering relies on unsupervised machine learning.

Team Members

Dhrish S Kumar - E0320008
A Ch Rohit - E0320022

Dataset - Patient Charges Clustering

Importing the Libraries

In [1]:

```
import numpy as np
import pandas as pd


# Matplotlib and Seaborn
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
```

Reading the file and printing the top 5 columns

In [2]:

```

#Reading Csv File
df=pd.read_csv("insurance.csv")

#Top 5 Rows
df.head()
```

Out[2]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

In [3]:

```
# Storing the orginal dataframe in another variable
original_df = df.copy()
```

Data cleaning and Data preparation

In [28]:

```
df.isnull().any()
```

Out[28]:

```
age         False
sex         False
bmi         False
children    False
smoker      False
region      False
charges     False
dtype: bool
```

Build clusters using K-means; choose K=3, metric as Eucledian.

In [5]:

```
X = df[["bmi", "charges"]]
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
```

Out[5]:

```
KMeans(n_clusters=3)
```

In [6]:

```
# Printing the Centroids
print(kmeans.cluster_centers_)
```

```
[[2.90733333e+01 1.86465264e+04]
 [3.04345491e+01 6.34535967e+03]
 [3.48454321e+01 4.07613086e+04]]
```

In [7]:

```
#t receives a label as the index of the cluster it gets assigned to. We can see these label
print(kmeans.labels_)
```

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

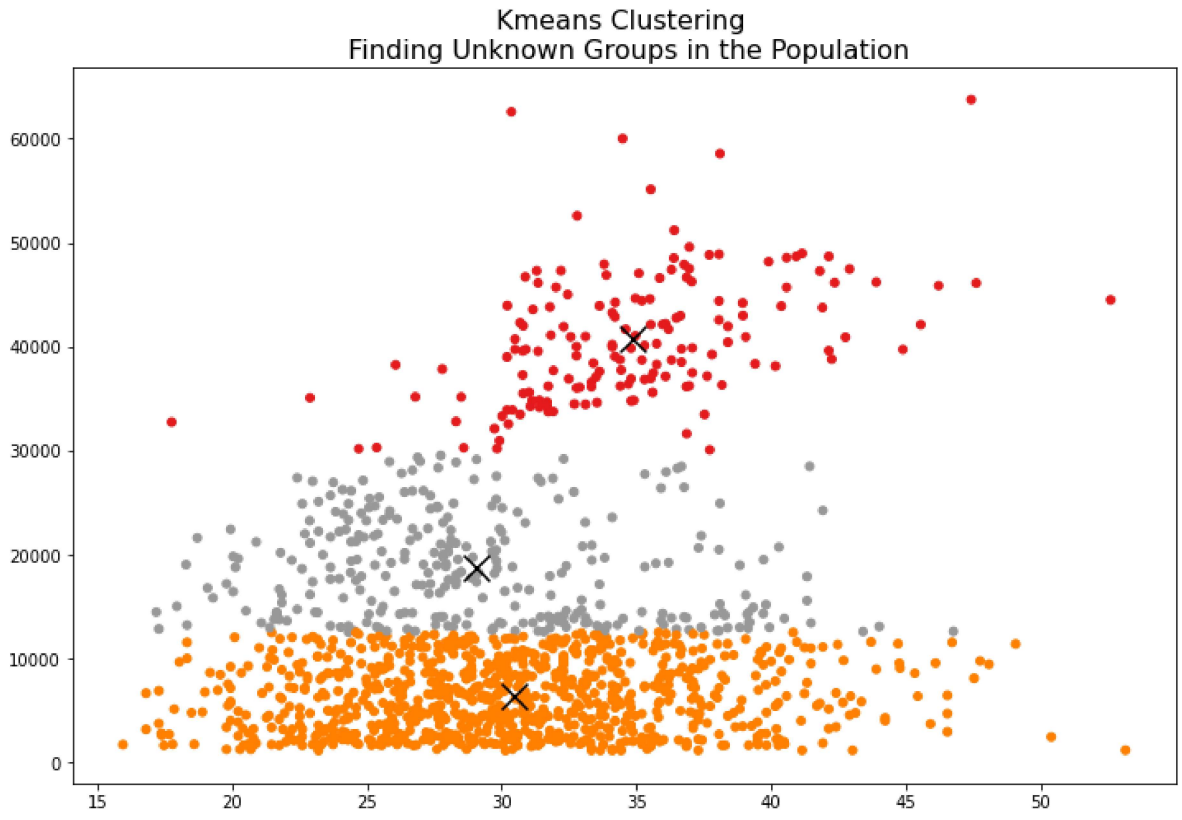
Explaining the Elbow Method: The elbow method finds the average sum of squares distance between the cluster centroid and the data observations. As the number of cluster increases the average sum of squares decreases. Basically, as the number of clusters increases, the distance between the data points and the

centroids decreases as well. Whenever, we see the "elbow" that is a rule of thumb to consider the optimal number of clusters.

In [8]:

```
import sklearn.cluster as KMeans
fig = plt.figure(figsize=(12,8))

plt.scatter(X.values[:,0], X.values[:,1], c=kmeans.labels_, cmap="Set1_r", s=25)
plt.scatter(kmeans.cluster_centers_[:,0] ,kmeans.cluster_centers_[:,1], color='black', mark
plt.title("Kmeans Clustering \n Finding Unknown Groups in the Population", fontsize=16)
plt.show()
```



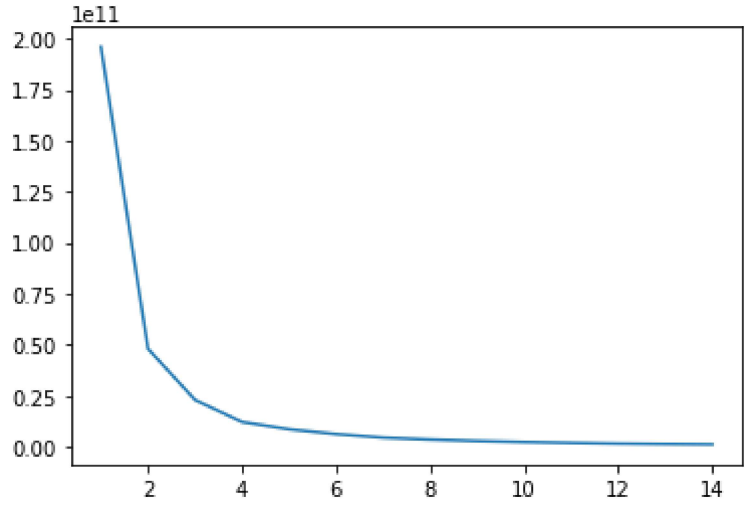
Print the optimal k value and within cluster SSE for that k value

In [14]:

```
from sklearn.cluster  import KMeans
# ELBOW METHOD
a=[]
for i in range(1,15):
    kmeans=KMeans(i)
    kmeans.fit(X)
    a.append(kmeans.inertia_)
    print(i,":",kmeans.inertia_)
plt.plot(range(1,15),a)
plt.show()
```

```
C:\Users\Rohit Anand\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:
881: UserWarning: KMeans is known to have a memory leak on Windows with MKL,
when there are less chunks than available threads. You can avoid it by setti
ng the environment variable OMP_NUM_THREADS=6.
  warnings.warn(

1 : 196074271288.56775
2 : 48014969101.122986
3 : 22958613268.22045
4 : 12124777274.518297
5 : 8540496641.444865
6 : 6167798545.289857
7 : 4434619533.806842
8 : 3496462584.3295546
9 : 2776483573.644193
10 : 2259108759.1467156
11 : 1841948151.3900037
12 : 1496421554.6398735
13 : 1281508793.1142607
14 : 1099543460.3421926
```



In [15]:

```
kmeans=KMeans(3)
kmeans.fit(X)
```

Out[15]:

```
KMeans(n_clusters=3)
```

In [16]:

```
pred=kmeans.fit_predict(X)
pred
```

Out[16]:

```
array([2, 0, 0, ..., 0, 0, 2])
```

Build clusters using agglomerative clustering for the random 10 datapoints of that dataset and display dendrogram tree

In [20]:

```
# Hierarchial CLustering
from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import StandardScaler,LabelEncoder
df1=df[:10]
```

In [22]:

```
le=LabelEncoder()
columns=['sex','smoker','region']
for col in columns:
    df1[col]=le.fit_transform(df1[col])
df1
```

```
<ipython-input-22-e9b5083998eb>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
  df1[col]=le.fit_transform(df1[col])
```

Out[22]:

	age	sex	bmi	children	smoker	region	charges
0	19	0	27.900	0	1	3	16884.92400
1	18	1	33.770	1	0	2	1725.55230
2	28	1	33.000	3	0	2	4449.46200
3	33	1	22.705	0	0	1	21984.47061
4	32	1	28.880	0	0	1	3866.85520
5	31	0	25.740	0	0	2	3756.62160
6	46	0	33.440	1	0	2	8240.58960
7	37	0	27.740	3	0	1	7281.50560
8	37	1	29.830	2	0	0	6406.41070
9	60	0	25.840	0	0	1	28923.13692

In [23]:

```
from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
scaled_x=ss.fit_transform(df1)
scaled_x=pd.DataFrame(scaled_x,columns=df1.columns)
```

Agglomerative Clustering

In [24]:

```
from sklearn.cluster import AgglomerativeClustering
cluster=AgglomerativeClustering(n_clusters=3,affinity='euclidean',linkage='ward')
cluster
```

Out[24]:

```
AgglomerativeClustering(n_clusters=3)
```

In [25]:

```
y=cluster.fit_predict(scaled_x)
y
```

Out[25]:

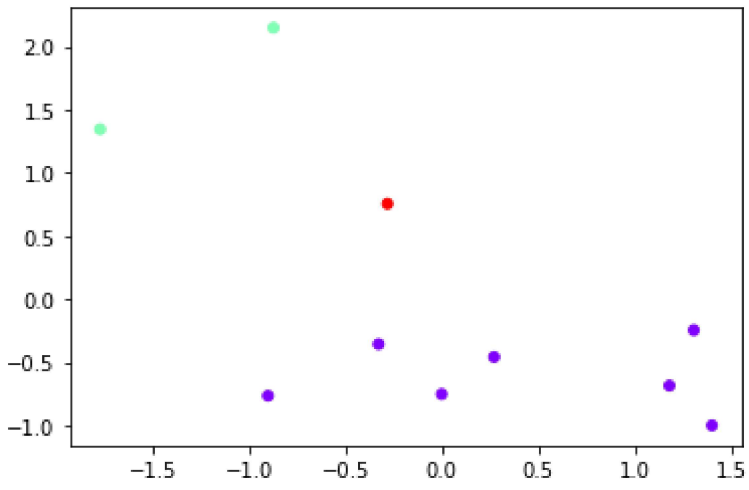
```
array([2, 0, 0, 1, 0, 0, 0, 0, 0, 1], dtype=int64)
```

In [26]:

```
plt.scatter(scaled_x['bmi'], scaled_x['charges'], c=y, cmap="rainbow", s=25)
```

Out[26]:

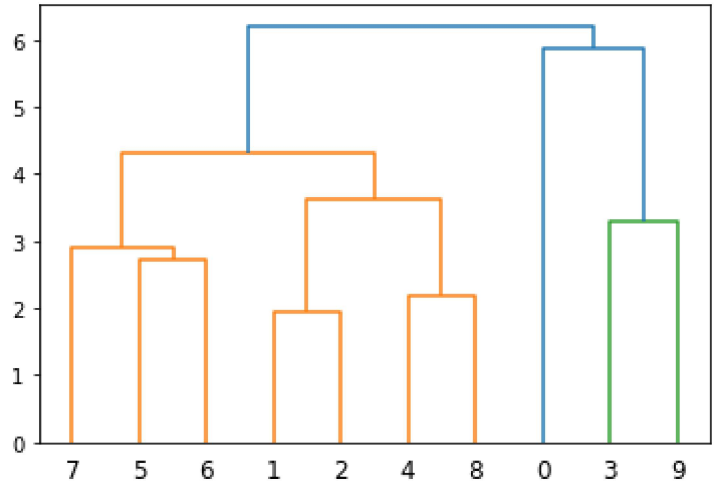
```
<matplotlib.collections.PathCollection at 0x238dad05a60>
```



Dendrogram

In [27]:

```
import scipy.cluster.hierarchy as shc
dend=shc.dendrogram(shc.linkage(scaled_x,method='ward'))
```



In [ ]:

```
```