PROBLEM STATEMENT 3:

Collect the data of used cars and bikes and try to predict the price using the user
input information of car or bike data.

Now-a-days, with the technological advancement, Techniques like Machine Learning, etc are being used on a large scale in many organisations. These models usually work with a set of predefined data-points available in the form of datasets. These datasets contain the past/previous information on a specific domain. Organising these datapoints before it is fed to the model is very important. This is where we use Data Analysis. If the data fed to the machine learning model is not well organised, it gives out false or undesired output. This can cause major losses to the organisation. Hence making use of proper data analysis is very important.

About Dataset:

The data that we are going to use in this example is about cars and bikes. Specifically containing various information datapoints about the used cars and used bikes, like their price, color, etc. Here we need to understand that simply collecting data isn't enough. Raw data isn't useful. Here data analysis plays a vital role in unlocking the information that we require and to gain new insights into this raw data.

Modules needed:

pandas:

Pandas is an opensource library that allows you to perform data manipulation in Python. Pandas provide an easy way to create, manipulate and wrangle the data.

numpy:

Numpy is the fundamental package for scientific computing with Python. numpy can be used as an efficient multi-dimensional container of generic data.

matplotlib:

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of formats.

seaborn:

Seaborn is a Python data-visualization library that is based on matplotlib. Seaborn provides a high-level interface for drawing attractive and informative statistical graphics.

scipy:

Scipy is a Python-based ecosystem of open-source software for mathematics, science, and engineering.

Steps that are used in the following code (Short description):

- Install all the packages
- Import the packages
- Set the path to the data file(.csv file)

- Find if there are any null data or NaN data in our file. If any, remove them
- Perform various data cleaning and data visualisation operations on your data. These steps are illustrated beside each line of code in the form of comments for better understanding, as it would be better to see the code side by side than explaining it entirely here, would be meaningless.
- Obtain the result!

Lets start analyzing the data.

Step 1: Import the modules needed.

```
# importing section
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy as sp
```

Step 2: Let's check the first five entries of dataset.

```
# using the Csv file
df = pd.read_csv('output.csv')

# Checking the first 5 entries of dataset
df.head()
```

Output:

```
In [54]: df = pd.read_csv('output.csv') #using the Csv file
         df.head() #Checking the first 5 entries of dataset
```

Out[54]:

	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.60	...	130	mpfi	3.47	2.68	9.00	111	5000	21	27	13495
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	16500
1	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	154	5000	19	26	16500
2	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	102	5500	24	30	13950
3	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	115	5500	18	22	17450
4	2	?	audi	gas	std	two	sedan	fwd	front	99.8	...	136	mpfi	3.19	3.40	8.5	110	5500	19	25	15250

5 rows × 26 columns

Step 3: Defining headers for our dataset.

```
headers = ["symboling", "normalized-losses", "make",
           "fuel-type", "aspiration","num-of-doors",
           "body-style","drive-wheels", "engine-location",
           "wheel-base","length", "width","height", "curb-weight",
           "engine-type","num-of-cylinders", "engine-size",
           "fuel-system","bore","stroke", "compression-ratio",
           "horsepower", "peak-rpm","city-mpg","highway-mpg","price"]

df.columns=headers
df.head()
```

Output:

In [28]:
```
headers = ["symboling", "normalized-losses", "make", "fuel-type",
           "aspiration","num-of-doors", "body-style","drive-wheels",
           "engine-location","wheel-base","length", "width","height",
           "curb-weight","engine-type","num-of-cylinders", "engine-size",
           "fuel-system","bore","stroke", "compression-ratio","horsepower",
           "peak-rpm","city-mpg","highway-mpg","price"]

df.columns=headers
df.head()
```

Out[28]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepowe
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	11
1	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	15
2	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	10
3	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	11
4	2	?	audi	gas	std	two	sedan	fwd	front	99.8	...	136	mpfi	3.19	3.40	8.5	11

5 rows × 26 columns

Step 4: Finding the missing value if any.

data = df

# Finding the missing values
data.isna().any()

# Finding if missing values
data.isnull().any()

In [56]:
```
data=df
data.isna().any() #Finding the missing values
data.isnull().any() #Finding if missing values
```

Out[56]:
```
3               False
?               False
alfa-romero     False
gas             False
std             False
two             False
convertible     False
rwd             False
front           False
88.60           False
168.80          False
64.10           False
48.80           False
2548            False
dohc            False
four            False
130             False
mpfi            False
3.47            False
2.68            False
9.00            False
111             False
5000            False
21              False
27              False
13495           False
dtype: bool
```

Output:

Step 5: Converting mpg to L/100km and checking the data type of each column.

```
# converting mpg to L / 100km
data['city-mpg'] = 235 / df['city-mpg']
data.rename(columns = {'city_mpg': "city-L / 100km"}, inplace = True)
```

```
print(data.columns)
```

```
# checking the data type of each column
data.dtypes
```

Output:

```
In [30]: # converting mpg to L/100km
         data['city-mpg'] = 235/df['city-mpg']
         data.rename(columns = {'city_mpg': "city-L/100km"}, inplace = True)

         print(data.columns)
         data.dtypes #checking the data type of each column

         Index(['symboling', 'normalized-losses', 'make', 'fuel-type', 'aspiration',
                'num-of-doors', 'body-style', 'drive-wheels', 'engine-location',
                'wheel-base', 'length', 'width', 'height', 'curb-weight', 'engine-type',
                'num-of-cylinders', 'engine-size', 'fuel-system', 'bore', 'stroke',
                'compression-ratio', 'horsepower', 'peak-rpm', 'city-mpg',
                'highway-mpg', 'price'],
               dtype='object')

Out[30]: symboling              int64
         normalized-losses     object
         make                  object
         fuel-type             object
         aspiration            object
         num-of-doors          object
         body-style            object
         drive-wheels          object
         engine-location       object
         wheel-base           float64
         length               float64
         width                float64
         height               float64
         curb-weight            int64
         engine-type           object
         num-of-cylinders      object
         engine-size            int64
         fuel-system           object
         bore                  object
         stroke                object
         compression-ratio    float64
         horsepower            object
         peak-rpm              object
         city-mpg             float64
         highway-mpg            int64
         price                 object
         dtype: object
```

Step 6: Here, price is of object type(string), it should be int or float, so we need to change it

```
data.price.unique()
```

```
# Here it contains '?', so we Drop it
data = data[data.price != '?']
```

```
# checking it again
data.dtypes
```

Output:

```
In [70]: #Here, price is of object type(string), it should be int or float, so we need to change it
         data.price.unique()

Out[70]: array([16500, 13950, 17450, 15250, 17710, 18920, 23875, 16430, 16925,
                20970, 21105, 24565, 30760, 41315, 36880,  5151,  6295,  6575,
                 5572,  6377,  7957,  6229,  6692,  7609,  8558,  8921, 12964,
                 6479,  6855,  5399,  6529,  7129,  7295,  7895,  9095,  8845,
                10295, 12945, 10345,  6785, 11048, 32250, 35550, 36000,  5195,
                 6095,  6795,  6695,  7395, 10945, 11845, 13645, 15645,  8495,
                10595, 10245, 10795, 11245, 18280, 18344, 25552, 28248, 28176,
                31600, 34184, 35056, 40960, 45400, 16503,  5389,  6189,  6669,
                 7689,  9959,  8499, 12629, 14869, 14489,  6989,  8189,  9279,
                 5499,  7099,  6649,  6849,  7349,  7299,  7799,  7499,  7999,
                 8249,  8949,  9549, 13499, 14399, 17199, 19699, 18399, 11900,
                13200, 12440, 13860, 15580, 16900, 16695, 17075, 16630, 17950,
                18150, 12764, 22018, 32528, 34028, 37028,  9295,  9895, 11850,
                12170, 15040, 15510, 18620,  5118,  7053,  7603,  7126,  7775,
                 9960,  9233, 11259,  7463, 10198,  8013, 11694,  5348,  6338,
                 6488,  6918,  7898,  8778,  6938,  7198,  7788,  7738,  8358,
                 9258,  8058,  8238,  9298,  9538,  8449,  9639,  9989, 11199,
                11549, 17669,  8948, 10698,  9988, 10898, 11248, 16558, 15998,
                15690, 15750,  7975,  7995,  8195,  9495,  9995, 11595,  9980,
                13295, 13845, 12290, 12940, 13415, 15985, 16515, 18420, 18950,
                16845, 19045, 21485, 22470, 22625], dtype=int64)
```

```
In [71]: # Here it contains '?', so we Drop it
         data = data[data.price!= '?']

         data['price'] = data['price'].astype(int)
         #checking it again
         data.dtypes

Out[71]: symboling              int64
         normalized-losses     object
         make                  object
         fuel-type             object
         aspiration            object
         num-of-doors          object
         body-style            object
         drive-wheels          object
         engine-location       object
         wheel-base           float64
         length               float64
         width                float64
         height               float64
         curb-weight            int64
         engine-type           object
         num-of-cylinders      object
         engine-size            int64
         fuel-system           object
         bore                  object
         stroke                object
         compression-ratio    float64
         horsepower            object
         peak-rpm              object
         city-mpg             float64
         highway-mpg            int64
         price                  int32
         price-binned        category
         dtype: object
```

Step 7: Normalizing values by using simple feature scaling method examples(do for the rest) and binning- grouping values

```
data['length'] = data['length']/data['length'].max()
data['width'] = data['width']/data['width'].max()
data['height'] = data['height']/data['height'].max()
```

```
# binning- grouping values
bins = np.linspace(min(data['price']), max(data['price']), 4)
group_names = ['Low', 'Medium', 'High']
data['price-binned'] = pd.cut(data['price'], bins,
                              labels = group_names,
                              include_lowest = True)
```

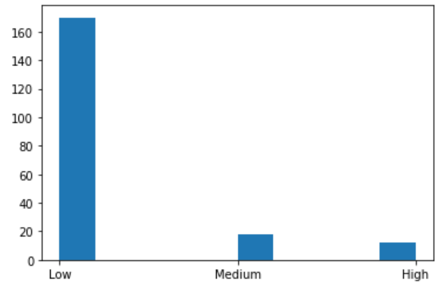
```
print(data['price-binned'])
plt.hist(data['price-binned'])
```

```
plt.show()
```

Output:

```
In [69]: #Normalizing values by using simple feature scaling method examples(do for the rest)
         data['length'] = data['length']/data['length'].max()
         data['width'] = data['width']/data['width'].max()
         data['height'] = data['height']/data['height'].max()

         #binning- grouping values
         bins = np.linspace(min(data['price']), max(data['price']), 4)
         group_names = ['Low','Medium', 'High']
         data['price-binned'] = pd.cut(data['price'], bins, labels=group_names, include_lowest=True)
         print(data['price-binned'])
         plt.hist(data['price-binned'])
         plt.show()
```

```
0         Low
1         Low
2         Low
3         Low
4         Low
        ...
199       Low
200    Medium
201    Medium
202    Medium
203    Medium
Name: price-binned, Length: 200, dtype: category
Categories (3, object): [Low < Medium < High]
```



Step 8: Doing descriptive analysis of data categorical to numerical values.

```
# categorical to numerical variables
pd.get_dummies(data['fuel-type']).head()

# descriptive analysis
# NaN are skipped
data.describe()
```

Output:

```
In [46]: #categorical to numerical variables
         pd.get_dummies(data['fuel-type']).head()

         #descriptive analysis
         #NaN are skipped
         data.describe()
```

Out[46]:

	symboling	wheel-base	length	width	height	curb-weight	engine-size	compression-ratio	city-mpg	highway-mpg	price
count	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000
mean	0.830000	98.848000	0.837232	0.915250	0.899523	2555.705000	126.860000	10.170100	9.937914	30.705000	13205.690000
std	1.248557	6.038261	0.059333	0.029207	0.040610	518.594552	41.650501	4.014163	2.539415	6.827227	7966.982558
min	-2.000000	86.600000	0.678039	0.837500	0.799331	1488.000000	61.000000	7.000000	4.795918	16.000000	5118.000000
25%	0.000000	94.500000	0.800937	0.891319	0.869565	2163.000000	97.750000	8.575000	7.833333	25.000000	7775.000000
50%	1.000000	97.000000	0.832292	0.909722	0.904682	2414.000000	119.500000	9.000000	9.791667	30.000000	10270.000000
75%	2.000000	102.400000	0.881788	0.926042	0.928512	2928.250000	142.000000	9.400000	12.368421	34.000000	16500.750000
max	3.000000	120.900000	1.000000	1.000000	1.000000	4066.000000	326.000000	23.000000	18.076923	54.000000	45400.000000

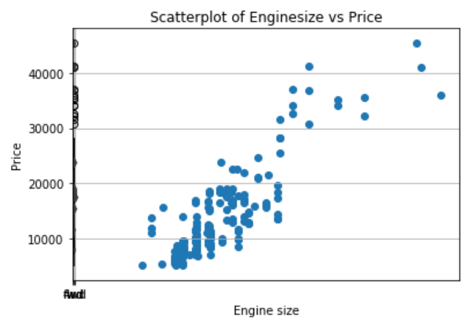
Step 9: Plotting the data according to the price based on engine size.

```
# examples of box plot
```

```
plt.boxplot(data['price'])

# by using seaborn
sns.boxplot(x ='drive-wheels', y ='price', data = data)

# Predicting price based on engine size
# Known on x and predictable on y
plt.scatter(data['engine-size'], data['price'])
plt.title('Scatterplot of Enginesize vs Price')
plt.xlabel('Engine size')
plt.ylabel('Price')
plt.grid()
plt.show()
```

Output:

```
In [48]: #examples of box plot
plt.boxplot(data['price'])

#by using seaborn
sns.boxplot(x='drive-wheels', y='price', data=data)

# Predicting price based on engine size
# Known on x and predictable on y
plt.scatter(data['engine-size'], data['price'])
plt.title('Scatterplot of Enginesize vs Price')
plt.xlabel('Engine size')
plt.ylabel('Price')
plt.grid()
plt.show()
```



Step 10: Grouping the data according to wheel, body-style and price.

```
# Grouping Data
test = data[['drive-wheels', 'body-style', 'price']]
data_grp = test.groupby(['drive-wheels', 'body-style'],
                        as_index = False).mean()

data_grp
```

Output:

```
In [49]: #Grouping Data
         test=data[['drive-wheels','body-style','price']]
         data_grp=test.groupby(['drive-wheels','body-style'], as_index=False).mean()
         data_grp
```

Out[49]:

	drive-wheels	body-style	price
0	4wd	hatchback	7603.000000
1	4wd	sedan	12647.333333
2	4wd	wagon	9095.750000
3	fwd	convertible	11595.000000
4	fwd	hardtop	8249.000000
5	fwd	hatchback	8396.387755
6	fwd	sedan	9811.800000
7	fwd	wagon	9997.333333
8	rwd	convertible	26563.250000
9	rwd	hardtop	24202.714286
10	rwd	hatchback	14337.777778
11	rwd	sedan	21711.833333
12	rwd	wagon	16994.222222

Step 11: Using the pivot method and plotting the heatmap according to the data obtained by pivot method

```
# pivot method
data_pivot = data_grp.pivot(index = 'drive-wheels',
columns = 'body-style')
data_pivot
# heatmap for visualizing data
plt.pcolor(data_pivot, cmap ='RdBu')
plt.colorbar()
plt.show()
```

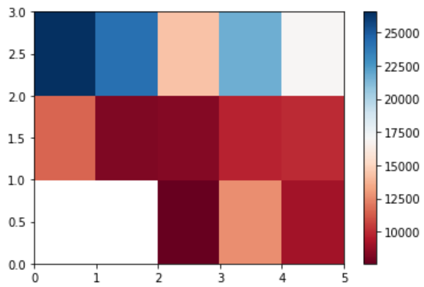
Output:

```
In [50]: #pivot method
         data_pivot = data_grp.pivot(index = 'drive-wheels', columns= 'body-style')
         data_pivot
```

Out[50]:

	price				
body-style	convertible	hardtop	hatchback	sedan	wagon
drive-wheels					
4wd	NaN	NaN	7603.000000	12647.333333	9095.750000
fwd	11595.00	8249.000000	8396.387755	9811.800000	9997.333333
rwd	26563.25	24202.714286	14337.777778	21711.833333	16994.222222

```
In [51]: #heatmap for visualizing data
         plt.pcolor(data_pivot, cmap='RdBu')
         plt.colorbar()
         plt.show()
```

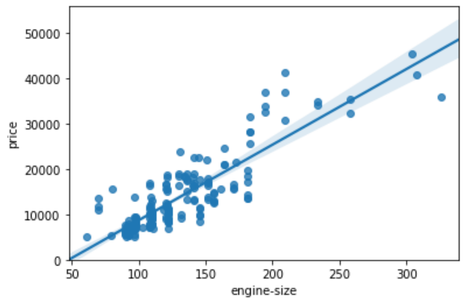


Step 12: Obtaining the final result and showing it in the form of a graph. As the slope is increasing in a positive direction, it is a positive linear relationship.

```
# Analysis of Variance- ANOVA
# returns f-test and p-value
# f-test = variance between sample group means divided by
# variation within sample group
# p-value = confidence degree
data_annova = data[['make', 'price']]
grouped_annova = data_annova.groupby(['make'])
annova_results_l = sp.stats.f_oneway(
grouped_annova.get_group('honda')['price'],
grouped_annova.get_group('subaru')['price'] )
print(annova_results_l)
# strong corealtion between a categorical variable
# if annova test gives large f-test and small p-value
# Correlation- measures dependency, not causation
sns.regplot(x ='engine-size', y ='price', data = data)
plt.ylim(0, )
```

Output:

```
In [52]: #Analysis of Variance- ANOVA
         # returns f-test and p-value
         #f-test= variance between sample group means divided by variation within sample group
         #p-value= confidence degree
         data_annova= data[['make','price']]
         grouped_annova=data_annova.groupby(['make'])
         annova_results_l=sp.stats.f_oneway(grouped_annova.get_group('honda')['price'], grouped_annova.get_group('subaru')['price'])
         print(annova_results_l)

         F_onewayResult(statistic=0.19744030127462606, pvalue=0.6609478240622193)

In [53]: #strong corealtion between a categorical variable if annova test gives large f-test and small p-value

         #Correlation- measures dependency, not causation
         sns.regplot(x='engine-size', y='price', data=data)
         plt.ylim(0,)

Out[53]: (0, 55927.89182129007)
```



Here similar approach will be used for Motor Bikes as well

The Dataset for Motorbikes is something like this

name,selling_price,year,seller_type,owner,km_driven,ex_showroom_price

Royal Enfield Classic 350,175000,2019,Individual,1st owner,350,

Honda Dio,45000,2017,Individual,1st owner,5650,

Royal Enfield Classic Gunmetal Grey,150000,2018,Individual,1st owner,12000,148114

........and many more is stored in a csv file

We take the similar approach as we have taken for Cars and get the desired results

	id	price	year	odometer	county	lat	lon
count	509577.000000	509577.000000	508050.000000	417253.000000	0.000000	499285.000000	49
mean	7044175513.514953	54796.838519	2009.662236	101729.961515	nan	38.453818	-94
std	4937218.519498	9575025.122822	8.567953	107378.985422	nan	5.902152	17.
min	6995212189.000000	0.000000	1900.000000	0.000000	nan	-82.688100	-16
25%	7040801843.000000	3995.000000	2007.000000	48488.000000	nan	34.557400	-10
50%	7045324884.000000	9377.000000	2011.000000	94894.000000	nan	39.145300	-88
75%	7048556309.000000	17955.000000	2015.000000	138778.000000	nan	42.449000	-81
max	7050103253.000000	3600028900.000000	2021.000000	10000000.000000	nan	81.569300	94.