Engr. Babu Mal

import the libraries

```
import numpy as py

import matplotlib.pyplot as plt

import pandas as pd
```

Import the dataset

```
df = pd.read_csv('data.csv')
```

Exploring Data

```
df
```

	date	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement	yr_built	yr_renovated
0	02/05/2014 0:00	3	1.50	1340	7912	1.5	0	0	3	1340	0	1955	2005
1	02/05/2014 0:00	5	2.50	3650	9050	2.0	0	4	5	3370	280	1921	0
2	02/05/2014 0:00	3	2.00	1930	11947	1.0	0	0	4	1930	0	1966	0
3	02/05/2014 0:00	3	2.25	2000	8030	1.0	0	0	4	1000	1000	1963	0
4	02/05/2014 0:00	4	2.50	1940	10500	1.0	0	0	4	1140	800	1976	1992
...	...	...	...	...	...	...	...	...	...	...	...	...	...
4596	09/07/2014 0:00	3	2.50	1460	7573	2.0	0	0	3	1460	0	1983	2009
4597	[illegible] 0:00	3	2.50	3010	7014	2.0	0	0	3	3010	0	2009	0
4598	10/07/2014 0:00	4	2.00	2090	6630	1.0	0	0	3	1070	1020	1974	0

```
df.shape
```

```
(4600, 18)
```

```
df.columns
```

```
Index(['date', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'waterfront', 'view', 'condition', 'sqft_above', 'sqft_basement',
       'yr_built', 'yr_renovated', 'street', 'city', 'statezip', 'country',
       'price'],
      dtype='object')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4600 entries, 0 to 4599
Data columns (total 18 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   date           4600 non-null   object
 1   bedrooms       4600 non-null   int64
 2   bathrooms      4600 non-null   float64
 3   sqft_living    4600 non-null   int64
 4   sqft_lot       4600 non-null   int64
 5   floors         4600 non-null   float64
 6   waterfront     4600 non-null   int64
 7   view           4600 non-null   int64
 8   condition      4600 non-null   int64
```

```
 9   sqft_above     4600 non-null   int64
 10  sqft_basement  4600 non-null   int64
 11  yr_built       4600 non-null   int64
 12  yr_renovated   4600 non-null   int64
 13  street         4600 non-null   object
 14  city           4600 non-null   object
 15  statezip       4600 non-null   object
 16  country        4600 non-null   object
 17  price          4600 non-null   float64
dtypes: float64(3), int64(10), object(5)
memory usage: 647.0+ KB
```

Data Cleaning

Checking Null Values

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

```
date             False
bedrooms         False
bathrooms        False
sqft_living      False
sqft_lot         False
floors           False
waterfront       False
view             False
condition        False
sqft_above       False
sqft_basement    False
yr_built         False
yr_renovated     False
street           False
city             False
statezip         False
country          False
price            False
dtype: bool
```

```
df.isnull().sum()
```

```
date             0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
```

```
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
street           0
city             0
statezip         0
country          0
price            0
dtype: int64
```

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

```
0
```

```
# Remove irrelevant columns
```

```
df = df.drop(columns=['date','country'])
```

```
df.columns
```

```
Index(['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'waterfront', 'view', 'condition', 'sqft_above', 'sqft_basement',
       'yr_built', 'yr_renovated', 'street', 'city', 'statezip', 'price'],
      dtype='object')
```

Separate dependent and independent feature

```
x = df[['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
        'waterfront', 'view', 'condition', 'sqft_above', 'sqft_basement',
        'yr_built', 'yr_renovated', 'street', 'city', 'statezip']]

y = df['price']
```

```
x.shape
```

```
(4600, 15)
```

```
y.shape
```

```
(4600,)
```

Separating categorical & Numerical columns

```
df_cat=x.select_dtypes("object")
df_num=x.select_dtypes("number")
```

Encoding of categorical data

Label Encoding

```
from sklearn.preprocessing import LabelEncoder

#labelencoder = LabelEncoder()

#for col in df_cat.columns:
 #   df_cat[col] = labelencoder.fit_transform(df_cat[col])
```

```
#df_cat
```

Dummies Method

```
df_cat_encoded=pd.get_dummies(df_cat,columns=df_cat.columns.to_list(), dtype=int)
df_cat_encoded.head()
```

	street_1 View Ln NE	street_10 W Etruria St	street_100 20th Ave E	street_100 24th Ave E	street_100 Mt Si Pl NW	street_1000 Mountain View Blvd SE	street_10000- 10026 S 100th St	street_10005 16th Ave S	street_10009 SE 247th Pl	street_1001 SW 102nd St	...	statezip_WA 98155	s
0	0	0	0	0	0	0	0	0	0	0	...	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	

5 rows × 4646 columns

```
# Concatenating encoded categorical data with numerical data
x = pd.concat([df_cat_encoded, df_num], axis=1, join='outer')
x.head()
```

	street_1 View Ln NE	street_10 W Etruria St	street_100 20th Ave E	street_100 24th Ave E	street_100 Mt Si Pl NW	street_1000 Mountain View Blvd SE	street_10000-10026 S 100th St	street_10005 16th Ave S	street_10009 SE 247th Pl	street_1001 SW 102nd St	...	sqft_living	s
0	0	0	0	0	0	0	0	0	0	0	...	1340	
1	0	0	0	0	0	0	0	0	0	0	...	3650	
2	0	0	0	0	0	0	0	0	0	0	...	1930	
3	0	0	0	0	0	0	0	0	0	0	...	2000	
4	0	0	0	0	0	0	0	0	0	0	...	1940	

5 rows × 4658 columns

```
x.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4600 entries, 0 to 4599
Columns: 4658 entries, street_1 View Ln NE to yr_renovated
dtypes: float64(2), int32(4646), int64(10)
memory usage: 81.9 MB
```

```
x.shape
```

```
(4600, 4658)
```

Training and Testing Data (divide the data into two part)

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test =train_test_split(x,y,test_size=0.2, random_state=0)
```

```
x_train.shape
```

```
(3680, 4658)
```

```
y_train.shape
```

```
(3680,)
```

```
x_test.shape
```

```
(920, 4658)
```

```
y_test.shape
```

```
(920,)
```

Linear Regression

```
from sklearn.linear_model import LinearRegression

reg = LinearRegression()
#Fit the model
reg.fit(x_train, y_train)
```

Prediction

```
y_predict1=reg.predict(x_test)
```

```
test=pd.DataFrame({
    'Actual':y_test,
    'Y test predicted':y_predict1
})
```

```
pred_df=pd.DataFrame({'Actual Value':y_test,'Predicted Value':y_predict1,'Difference':y_test-y_predict1})
pred_df
```

Accuracy

```
print("Accuracy --> ", reg.score(x_test, y_test)*100)
```

```
Accuracy -->  60.48134100244235
```

R-Square

```
from sklearn.metrics import mean_squared_error, r2_score
r2_square = r2_score(y_test,y_predict1)
print(f" R-squared: {r2_square}")
```

```
 R-squared: 0.6048134100244235
```

Mean Squared Error

```
mse = mean_squared_error(y_test, y_predict1)
print(f'Mean Squared Error: {mse}')
```

```
Mean Squared Error: 58467232765.23475
```

Multiple Regression

Double-click (or enter) to edit

```
from sklearn.linear_model import LinearRegression
mreg = LinearRegression()
mreg.fit(x_train,y_train)
```

Prediction

```
y_predict2=mreg.predict(x_test)
```

```
test=pd.DataFrame({
    'Actual':y_test,
    'Y test predicted':y_predict2
})
```

```
pred_df=pd.DataFrame({'Actual Value':y_test,'Predicted Value':y_predict2,'Difference':y_test-y_predict2})
pred_df
```

Accuracy

```
print("Accuracy --> ", mreg.score(x_test, y_test)*100)
```

R-Squared

```
from sklearn.metrics import mean_squared_error, r2_score
r2_square = r2_score(y_test,y_predict2)
print(f" R-squared: {r2_square}")
```

Mean Squraed Error

```
mse = mean_squared_error(y_test, y_predict2)
print(f'Mean Squared Error: {mse}')
```

Support Vector Regression

```
from sklearn.svm import SVR
SV_reg = SVR(kernel='rbf')
SV_reg.fit(x, y)
```

Prediction

```
y_predict3=mreg.predict(x_test)
```

```
test=pd.DataFrame({
    'Actual':y_test,
    'Y test predicted':y_predict3
})
```

```
pred_df=pd.DataFrame({'Actual Value':y_test,'Predicted Value':y_predict3,'Difference':y_test-y_predict3})
pred_df
```

Accuracy

```
print("Accuracy --> ", mreg.score(x_test, y_test)*100)
```

R-Squared

```
from sklearn.metrics import mean_squared_error, r2_score
r2_square = r2_score(y_test,y_predict3)
print(f" R-squared: {r2_square}")
```

Mean Squared Error

```
mse = mean_squared_error(y_test, y_predict3)
print(f'Mean Squared Error: {mse}')
```

Random Forest Regression

```
from sklearn.ensemble import RandomForestRegressor
RF_reg = RandomForestRegressor(n_estimators=10,random_state =0)
RF_reg.fit(x, y)
```

Prediction

```
y_predict4=mreg.predict(x_test)
```

```
test=pd.DataFrame({
    'Actual':y_test,
    'Y test predicted':y_predict4
})
```

```
pred_df=pd.DataFrame({'Actual Value':y_test,'Predicted Value':y_predict4,'Difference':y_test-y_predict4})
pred_df
```

Accuracy

```
print("Accuracy --> ", mreg.score(x_test, y_test)*100)
```

R-Squared

```
from sklearn.metrics import mean_squared_error, r2_score
r2_square = r2_score(y_test,y_predict4)
print(f" R-squared: {r2_square}")
```

Mean Squared Error

```
mse = mean_squared_error(y_test, y_predict4)
print(f'Mean Squared Error: {mse}')
```

Decision Tree Regression

```
from sklearn.tree import DecisionTreeRegressor
DT_reg = DecisionTreeRegressor(random_state =0)
DT_reg.fit(x, y)
```

Prediction

```
y_predict5=mreg.predict(x_test)
```

```
test=pd.DataFrame({
    'Actual':y_test,
    'Y test predicted':y_predict5
})
```

```
pred_df=pd.DataFrame({'Actual Value':y_test,'Predicted Value':y_predict5,'Difference':y_test-y_predict5})
pred_df
```

Accuracy

```
print("Accuracy --> ", mreg.score(x_test, y_test)*100)
```

R-Squared

```
from sklearn.metrics import mean_squared_error, r2_score
r2_square = r2_score(y_test,y_predict5)
print(f" R-squared: {r2_square}")
```

Mean Squared Error

```
mse = mean_squared_error(y_test, y_predict5)
print(f'Mean Squared Error: {mse}')
```