Outstanding-Project-1

By:
Name: Jatin Jindal
E-mail Id: jatinjindal1199@gmail.com

Problem Statement 1

Importing the Dataset

```
1 df=pd.read_csv('/content/drive/My Drive/Colab Notebooks/DataSets/train.csv')
2 df
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Cond
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	CollgCr	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	FR2	Gtl	Veenker	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	CollgCr	

1. Data Understanding, Preparation and EDA

Finding the columns having the null values

```
1 # checking the columns wir=th the null values
2 df.isnull().sum()
```

```
Id                0
MSSubClass        0
MSZoning          0
LotFrontage     259
LotArea           0
                 ...
MoSold            0
YrSold            0
SaleType          0
SaleCondition     0
SalePrice         0
Length: 81, dtype: int64
```

a) Removing the non-relevant or Null columns

Finding the columns having more than 40% of NULL values

```
[ ]  1 # Columns with more than 40% of NULL values are removed
     2 miss_col=df.isnull().sum()/df.shape[0]
     3 miss_col=miss_col[miss_col>0.4]
     4 miss_col
```

```
Alley          0.937671
FireplaceQu    0.472603
PoolQC         0.995205
Fence          0.807534
MiscFeature    0.963014
dtype: float64
```

Removing the 'Id' column and other columns having More than 40% Null values as id column has no significance in model training and columns with more than 40% Null values will make model biased even after Imputation on that missing data.

```
[ ]  1 # id is not used in prediction
     2 df=df.drop(columns=['Id'])
```

```
[ ]  1 df=df.drop(columns=['Alley','FireplaceQu','PoolQC','Fence','MiscFeature'])
     2 df
```

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condi
0	60	RL	65.0	8450	Pave	Reg	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	
1	20	RL	80.0	9600	Pave	Reg	Lvl	AllPub	FR2	Gtl	Veenker	Feedr	
2	60	RL	68.0	11250	Pave	IR1	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	

b) Filling the missing values

```
1 # Filling the missing numerical Values with mean strategy
2 from sklearn.impute import SimpleImputer
3 impute_num=SimpleImputer(missing_values=np.nan,strategy='mean')
4 df[num_col]=impute_num.fit_transform(df[num_col])
```

```
[ ]  1 # Filling the missing categorical Values with median strategy
     2 from sklearn.impute import SimpleImputer
     3 impute_obj=SimpleImputer(missing_values=np.nan,strategy='most_frequent')
     4 df[obj_col]=impute_obj.fit_transform(df[obj_col])
```

```
[ ]  1 df.isnull().sum().sum()
```

```
0
```

Filling the Numerical values with the mean strategy and categorical values with the most frequent strategy.

c) Data Encoding

```
[ ]  1 #Encoding Categorical Data for each object type column
     2 from sklearn.preprocessing import LabelEncoder
     3 le=LabelEncoder()
     4 for i in obj_col:
     5     df[i]=le.fit_transform(df[i])
     6 print(df.select_dtypes(include=['object']).columns)
     7 df
```

```
Index([], dtype='object')
```

Encoding the object data types to numerical values using the Label Encoder as the number of Columns and Unique values are too much so Dummy encoding will lead to huge increase in data.

d) Separating the dependent and independent data

```
1 # 'x' is dependent Variable and 'y' is independent variable
2 y=df.iloc[:,-1].values
3 x=df.iloc[:,1:-1].values
```

e) Splitting od data to train and test data

```
1 # split the data to train and test
2 # x_train and y_train are used for training
3 # y_train and y_test are ysed for testing
4 from sklearn.model_selection import train_test_split
5 x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=0)
6 print(x_train.shape)
7 print(x_test.shape)
8 print(y_train.shape)
9 print(y_test.shape)
```

```
(1095, 73)
(365, 73)
(1095, 1)
(365, 1)
```

f) Standardization of data to bring them to a single scale

```
#Standardisation of data
from sklearn.preprocessing import StandardScaler
x_sc=StandardScaler()
y_sc=StandardScaler()
x_train_sc=x_sc.fit_transform(x_train)  # x_train_sc and x_test_sc are Standardised Data
x_test_sc=x_sc.transform(x_test)
y_train_sc=y_sc.fit_transform(y_train)
y_test_sc=y_sc.transform(y_test)
```

2. Model training

a) Linear Regression

```
[23]  # Using the Linear Regression on Standardised data
      from sklearn.linear_model import LinearRegression
      lr_sc=LinearRegression()
      lr_sc.fit(x_train_sc,y_train_sc)
      lr_sc.score(x_test_sc,y_test_sc)

      0.6268951603734914
```

Applying Linear regression on the scaled data and we can see that the accuracy of the data is just 60% which is not so good accuracy.

b) Decision Tree Regression

```
[24]  from sklearn.tree import DecisionTreeRegressor
      tree_reg=DecisionTreeRegressor(random_state=10)
      tree_reg.fit(x_train,y_train)
      tree_reg.score(x_test,y_test)

      0.7787060591355801
```

```
from sklearn.tree import DecisionTreeRegressor
tree_reg_sc=DecisionTreeRegressor(random_state=10)
tree_reg_sc.fit(x_train_sc,y_train_sc)
tree_reg_sc.score(x_test_sc,y_test_sc)

0.5677359950152605
```

As we can see that the accuracy of non-standardized data id less than that of the Standardized data as it can be due to underfitting of the model.
so we will Cross validation to check the actual accuracy of the model.

Cross Validation on Decision Tree Regression

```
[39] 1 from sklearn.model_selection import cross_val_score
     2 accuracies_tree= cross_val_score(estimator = tree_reg_sc, X = x_train_sc, y = y_train_sc, cv = 10)
     3 print('Accuracy={:f} and Standard Deviation={:f}'.format(accuracies_tree.mean()*100,accuracies_tree.std()*100))
     4 accuracies_tree

Accuracy=74.108599 and Standard Deviation=3.787729
array([0.72195466, 0.70866687, 0.66557745, 0.7885354 , 0.76954738,
       0.73474046, 0.73840736, 0.79224192, 0.71855878, 0.77262961])
```

As we can now see that the accuracy of the data is coming out to be 74% . So we can say that our data's accuracy is more than that of Linear Regression.

c) Random Forest Regression

```
[27] 1 #Checking random Forest for different values of n_estimators
     2 from sklearn.ensemble import RandomForestRegressor
     3 for i in range(10,30):
     4     forest_reg=RandomForestRegressor(n_estimators=i,random_state=10)
     5     forest_reg.fit(x_train,y_train)
     6     print(forest_reg.score(x_test,y_test))

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5: DataConversio
  """
0.8312147509362738
0.8345527243408589
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5: DataConversio
  """
0.8358626980662235
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5: DataConversio
  """
0.8423191976651944
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5: DataConversio
  """
0.847734125251339
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5: DataConversio
  """
0.8499431377627367
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5: DataConversio
  """
0.8531192891965078
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5: DataConversio
  """
0.8503359651136666
```

We can see that clearly that the Accuracy of Random Forest Regression has raised upto 84%.
That is this is the best accuracy so far we have achieved.

Hyperparameter Tuning
So now applying Parameter tuning of the Random Forest Regression to find the best 'n_estimators' value for our model.

```
from sklearn.model_selection import GridSearchCV
forest_reg_sc=RandomForestRegressor(random_state=10)
para={'n_estimators':[1,5,10,20,30,35,40,45,50,55,100]}
print(para)
grid_rig=GridSearchCV(estimator=forest_reg_sc,
                      param_grid=para,scoring='neg_mean_squared_error',
                      cv=10,n_jobs=-1)
grid_rig=grid_rig.fit(x_train_sc,y_train_sc)
print('Best Score= ',grid_rig.best_score_)
print('Best parameter= ',grid_rig.best_params_)
```

```
{'n_estimators': [1, 5, 10, 20, 30, 35, 40, 45, 50, 55, 100]}
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_search.py:7
  self.best_estimator_.fit(X, y, **fit_params)
Best Score=  -0.14225156741252742
Best parameter=  {'n_estimators': 45}
```

Now training the best model obtained so far with the best value of the its paramters funded so far. As Our Best model was Random Forest Regression and the its best parameter is 45 for n_estimators.

```
[30]
forest_reg_sc=RandomForestRegressor(n_estimators=45,random_state=10)
forest_reg_sc.fit(x_train_sc,y_train_sc)
print(forest_reg_sc.score(x_test_sc,y_test_sc))
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2: DataConvers

0.8503088141303556
```

Cross Validation of Random Forest Regression Model to find the General Accuracy of the Model and Checking whether the model is underfitted or overfitted.

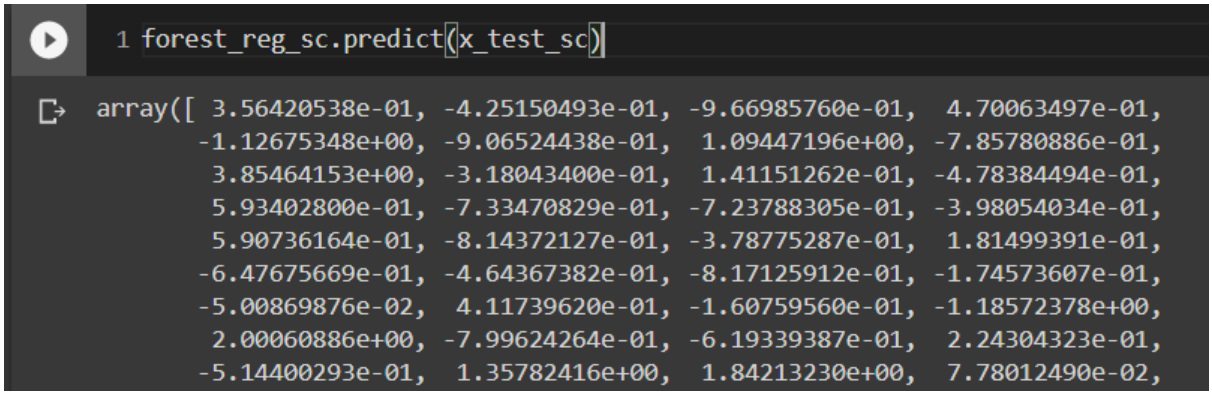
```
# Cross validation of the Random Forest Regression model to check for the General Accuracy of the Model
from sklearn.model_selection import cross_val_score
accuracies_forest= cross_val_score(estimator = forest_reg_sc, X = x_train_sc, y = y_train_sc, cv = 10)
print('Accuracy={:f} and Standard Deviation={:f}'.format(accuracies_forest.mean()*100,accuracies_forest.std()*100))
accuracies_forest
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_validation.py:515: DataConversionWarning: A column-vector
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_validation.py:515: DataConversionWarning: A column-vector
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_validation.py:515: DataConversionWarning: A column-vector
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_validation.py:515: DataConversionWarning: A column-vector
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_validation.py:515: DataConversionWarning: A column-vector
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_validation.py:515: DataConversionWarning: A column-vector
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_validation.py:515: DataConversionWarning: A column-vector
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_validation.py:515: DataConversionWarning: A column-vector
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_validation.py:515: DataConversionWarning: A column-vector
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_validation.py:515: DataConversionWarning: A column-vector
  estimator.fit(X_train, y_train, **fit_params)
Accuracy=85.271608 and Standard Deviation=8.008481
array([0.86210824, 0.89326848, 0.61663332, 0.89964736, 0.87692039,
       0.87801207, 0.84731309, 0.8840936 , 0.87470616, 0.89445812])
```

As we can see that the accuracy of the model and the through the cross validation is same so we can same that our model is neither overfitted nor underfitted.

Prediction

Predicting the values corresponding to the test data



```
1 forest_reg_sc.predict(x_test_sc)
```

```
array([ 3.56420538e-01, -4.25150493e-01, -9.66985760e-01,  4.70063497e-01,
       -1.12675348e+00, -9.06524438e-01,  1.09447196e+00, -7.85780886e-01,
        3.85464153e+00, -3.18043400e-01,  1.41151262e-01, -4.78384494e-01,
        5.93402800e-01, -7.33470829e-01, -7.23788305e-01, -3.98054034e-01,
        5.90736164e-01, -8.14372127e-01, -3.78775287e-01,  1.81499391e-01,
       -6.47675669e-01, -4.64367382e-01, -8.17125912e-01, -1.74573607e-01,
       -5.00869876e-02,  4.11739620e-01, -1.60759560e-01, -1.18572378e+00,
        2.00060886e+00, -7.99624264e-01, -6.19339387e-01,  2.24304323e-01,
       -5.14400293e-01,  1.35782416e+00,  1.84213230e+00,  7.78012490e-02,
```