House Price Accessment

In [ ]:

```
# Importing the libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelBinarizer # one hot encoding
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score
import xgboost as xgb
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import *
from sklearn.ensemble import GradientBoostingClassifier
```

In [ ]:

```
#Read the dataset
dataset=pd.read_csv('House_Pricing.csv')
```

In [ ]:

```
dataset.head(3)
```

In [ ]:

```
dataset.dtypes
```

In [ ]:

```
dataset.shape
```

In [ ]:

```
list(dataset.columns)
```

Remove (House number,Unit number,Street Name,Zip Code) columns

In [ ]:

```
dataset=dataset.drop(['house_number', 'unit_number','street_name','zip_code'], axis = 1)
```

After removing four columns, dataset has 16 columns only.

In [ ]:

```
dataset.shape
```

In [ ]:

```
price_category=dataset['sale_price']
print('Maximum price : ',price_category.max() )
print('Minimum price : ',price_category.min())
print('Mean price :',price_category.mean())
```

In [ ]:

```
# Convert Y variable (Price) into 2 categories
#set up bins
bin=[664.0,441986.20551249327,22935778.0]
#use pd.cut function can attribute the values into its specific bins
# 'o' means the range between minimum and mean values, '1' means the range between mean and
category = pd.cut(price_category,bin,labels=['0','1'])
category = category.to_frame()
#Replace an entire  sale_price and its bin
new_dataset = dataset.assign(sale_price=category)
```

In [ ]:

```
#After converting categorical varibales
new_dataset.head(3)
```

In [ ]:

```
# Replace categorical data with one-hot encoded data(Garage type,city)
new_dataset['garage_type']
new_dataset.groupby('garage_type').size()
# convert the data type to category
new_dataset['garage_type'] = new_dataset['garage_type'].astype('category')
```

In [ ]:

```
#label encoding
new_dataset['garage_type_cat']= new_dataset['garage_type'].cat.codes
```

In [ ]:

```
new_dataset.groupby(['garage_type', 'garage_type_cat']).size()
```

In [ ]:

```
df_one_hot = new_dataset.copy()
lb = LabelBinarizer()
lb_results = lb.fit_transform(df_one_hot['garage_type'])
lb_results_df = pd.DataFrame(lb_results, columns=lb.classes_)
```

In [ ]:

```
## concatenate this data to our data set
final_df = pd.concat([df_one_hot, lb_results_df], axis=1)
print('original df dimensions:', new_dataset.shape)
print('one hot encoded df dimensions:', final_df.shape)
```

In [ ]:

```
# Replace categorical data with one-hot encoded data(city)
final_df['city']
final_df.groupby('city').size()
# convert the data type to category
final_df['city'] = final_df['city'].astype('category')
```

In [ ]:

```
#label encoding
final_df['city_cat']= final_df['city'].cat.codes
```

In [ ]:

```
final_df.groupby(['city', 'city_cat']).size()
```

In [ ]:

```
# one hot encoding
df_one_hot = final_df.copy()
lb = LabelBinarizer()
lb_results = lb.fit_transform(df_one_hot['city'])
lb_results_df = pd.DataFrame(lb_results, columns=lb.classes_)
lb_results_df.head()
# concatenate this data to our data set
final_dataset = pd.concat([final_df, lb_results_df], axis=1)
```

In [ ]:

```
print('original df dimensions:', dataset.shape)
print('one hot encoded df dimensions:', final_dataset.shape)
```

In [ ]:

```
df=final_dataset.drop(['garage_type', 'city'], axis=1)  # dropping city and garage_type col
```

In [ ]:

```
# Create the X and y arrays
df_Train=df.iloc[1000:,:] #copying all columns inside df_train
X=df_Train.drop(['sale_price'],axis=1) # dropping sale_price column
y = df.loc[1000:, 'sale_price'].values
```

In [ ]:

```
# Split the data set in a training set (70%) and a test set (30%)
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

Feature Scaling

In [ ]:

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Define a model, using Random Forest and Gradient Boosting Classifier

Random Forest Classifier

In [ ]:

```
random_classifier=RandomForestClassifier(
            max_depth=6, max_features=1.0,
            min_samples_leaf=17,
            n_estimators=500, n_jobs=-1, random_state=0,
            )
```

In [ ]:

```
random_classifier.fit(X_train,Y_train)
```

In [ ]:

```
#Predicting the test set results
y_pred = random_classifier.predict(X_test)
```

In [ ]:

```
y_pred
```

In [ ]:

```
#Accuray Score
accuaracy=accuracy_score(Y_test, y_pred)
print("Random Forest Classifier : %.2f%% "%(accuaracy*100.0))
```

In [ ]:

```
print("Confusion Matrix:")
print(confusion_matrix(Y_test, y_pred))

print("Classification Report")
print(classification_report(Y_test, y_pred))
```

Random Forest Classifier Using GridSearch

In [ ]:

```
param_grid = {
    'bootstrap': [True],
    'max_depth': [6],
    'max_features': [1.0],
    'min_samples_leaf': [17],
    'criterion': ['gini'],
    'n_estimators': [500],
    'random_state':[0],
}
```

In [ ]:

```
random_classifier_ = RandomForestClassifier()
```

In [ ]:

```
random_gridsearch = GridSearchCV(random_classifier_, param_grid=param_grid, n_jobs=-1, cv=
                    scoring='accuracy')
```

In [ ]:

```
random_gridsearch.fit(X_train,Y_train)
```

In [ ]:

```
best_parameters = random_gridsearch.best_params_
print(best_parameters)
```

In [ ]:

```
best_result = random_gridsearch.best_score_
print("Score of Random Forest Classifier Using GridSearchCV : %.2f%% " %(best_result*100.0)
```

In [ ]:

```
#Predicting the test set results
y_pred_gridsearch = random_gridsearch.predict(X_test)
```

In [ ]:

```
y_pred_gridsearch
```

In [ ]:

```
#Accuray Score
accuaracy=accuracy_score(Y_test, y_pred_gridsearch)
print("Accuracy of Random Forest Classifier Using GridSearchCV: %.2f%% "%(accuaracy*100.0))
```

In [ ]:

```
print("Confusion Matrix:")
print(confusion_matrix(Y_test, y_pred_gridsearch))

print("Classification Report")
print(classification_report(Y_test, y_pred_gridsearch))
```

Gradient Boosting Classifier Model

In [ ]:

```
gb_clf = GradientBoostingClassifier(n_estimators=500, learning_rate=0.2, max_features=1.0,
```

In [ ]:

```
gb_clf.fit(X_train,Y_train)
```

In [ ]:

```
gb_y_pred = gb_clf.predict(X_test)
```

In [ ]:

```
print("Confusion Matrix:")
print(confusion_matrix(Y_test, gb_y_pred))

print("Classification Report")
print(classification_report(Y_test, gb_y_pred))
```

In [ ]:

```
# Accuray Score
accuaracy=accuracy_score(Y_test, gb_y_pred)
print("Accuracy Score of Gradient Boosting Classifier : %.2f%% "%(accuaracy*100.0))
```

Gradient Boosting Classifier Model Using GridSearch

In [ ]:

```
parameters={
 'learning_rate':[0.1],
 'max_depth': [6],
 'max_features': [1.0],
 'min_samples_leaf': [5],
 'min_samples_split': [12],
 'n_estimators': [500],
  'random_state':[0]
}
```

In [ ]:

```
gb_clf = GradientBoostingClassifier()
```

In [ ]:

```
gb_gridsearch = GridSearchCV(gb_clf,parameters, n_jobs=-1,
                   cv=2,
                   scoring='accuracy',
                   verbose=2, refit=True)
```

In [ ]:

```
gb_gridsearch.fit(X_train, Y_train)
```

In [ ]:

```
best_parameters = gb_gridsearch.best_params_
print(best_parameters)
```

In [ ]:

```
best_result = gb_gridsearch.best_score_
print("Score of Gradient Boosting Classifier Using GridSearchCV : %.2f%% " %(best_result*10
```

In [ ]:

```
gb_gridsearch_y_pred = gb_gridsearch.predict(X_test)
```

In [ ]:

```
gb_gridsearch_y_pred
```

In [ ]:

```
print("Confusion Matrix:")
print(confusion_matrix(Y_test, gb_gridsearch_y_pred))

print("Classification Report")
print(classification_report(Y_test, gb_gridsearch_y_pred))
```

In [ ]:

```
# Accuray Score
accuaracy=accuracy_score(Y_test, gb_gridsearch_y_pred)
print("Accuracy of Gradient Boosting Classifier Using GridSearchCV : %.2f%% "%(accuaracy*10
```

XGBoost Classifier

In [ ]:

```
xgb_model = xgb.XGBClassifier(n_estimators=500, learning_rate=0.2, max_features=1.0, max_de
```

In [ ]:

```
xgb_model.fit(X_train, Y_train)
```

In [ ]:

```
xgb_y_pred=xgb_model.predict(X_test)
```

In [ ]:

```
print("Confusion Matrix:")
print(confusion_matrix(Y_test, xgb_y_pred))

print("Classification Report")
print(classification_report(Y_test, xgb_y_pred))
```

In [ ]:

```
# Accuray Score
accuaracy=accuracy_score(Y_test, xgb_y_pred)
print("Accuracy of XGBoost Classifier : %.2f%% "%(accuaracy*100.0))
```

XGboost Classifier Using GridSearch

In [ ]:

```
xgb_model = xgb.XGBClassifier()

param = {
    "learning_rate": [0.2],
    "max_depth":[6],
    "min_samples_leaf":[3],
    "max_features":[1.0],
    "n_estimators":[500],
    "random_state":[0]
    }

xg_clf_gridsearch = GridSearchCV(xgb_model, param_grid=param, n_jobs=-1,
                   cv=2,
                   scoring='accuracy',
verbose=2, refit=True)
```

In [ ]:

```
xg_clf_gridsearch.fit(X_train, Y_train)
```

In [ ]:

```
best_parameters = xg_clf_gridsearch.best_params_
print(best_parameters)
```

In [ ]:

```
best_result = xg_clf_gridsearch.best_score_
print("Score of XGBoost Classifier Using GridSearchCV : %.2f%% " %(best_result*100.0))
```

In [ ]:

```
xgb_gridsearch_y_pred=xg_clf_gridsearch.predict(X_test)
```

In [ ]:

```
print("Confusion Matrix:")
print(confusion_matrix(Y_test, xgb_gridsearch_y_pred))

print("Classification Report")
print(classification_report(Y_test, xgb_gridsearch_y_pred))
```

In [ ]:

```
# Accuray Score
xgcv_accuracy=accuracy_score(Y_test, xgb_gridsearch_y_pred)
print("Accuracy of XGBoost Classifier Using GridsearchCV : %.2f%% "%(accuaracy*100.0))
```

Save the trained model to a file so we can use it in other programs using joblib.dump

In [ ]:

```
from sklearn.externals import joblib
```

In [ ]:

```
joblib.dump(xg_clf_gridsearch,'train_model')
```

In [ ]:

```
load_model=joblib.load('train_model')
```

In [ ]:

```
load_model
```

In [ ]:

```
# Find the error rate on the training set

training_error=1-(xg_clf_gridsearch.score(X_train, Y_train))
```

In [ ]:

```
training_error
```

In [ ]:

```
# Find the error rate on the training set

test_errors=1-xgcv_accuracy
```

In [ ]:

```
test_errors
```

In [ ]:

```
# Try to give some real values in a list and predict them by loading the file you have save
future_pred=xg_clf_gridsearch.predict([0.0,100.0,200.0,300.0,400.0,500.0,600.0,700.0,800.0,


# future_pred=xg_clf_gridsearch.predict([0.0,12000.0,1100.0,0.0,1400.0,0.0,0.0,1200.0,1800.
```

In [ ]:

```
future_pred
```

In [ ]:

```

```