```
from google.colab import files
uploaded=files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving train (1).csv to train (1) (2).csv
```

```
from google.colab import files
uploaded1=files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving test (1).csv to test (1) (2).csv
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
import pandas as pd
import io
train_df=pd.read_csv(io.BytesIO(uploaded['train (1).csv']))
```

```
import pandas as pd
import io
test_df=pd.read_csv(io.BytesIO(uploaded1['test (1).csv']))
```

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
train_df.head()
```

```
test_df.head()
```

```
test_df.isnull().sum()
```

```
sns.heatmap(train_df.isnull(),cbar=False)
```

```
train_df['LandContour'].value_counts()
```

```
test_df['LandContour'].value_counts()
```

```
plt.figure(figsize=(26,26))
x=sns.heatmap(train_df.corr(),annot=True)
plt.show()
```

```
train_df.shape , test_df.shape
```

```
list_col_obj=list(train_df.select_dtypes(include='object').columns)
```

```
list_col_obj
```

```
len(list_col_obj)
```

```
list_num_obj=list(train_df.select_dtypes(exclude='object').columns)


def fillna_all(df):
    for col in list_col_obj:
        df[col].fillna(value=df[col].mode()[0],inplace=True)
    for col in list_num_obj:
        df[col].fillna(value=df[col].mean(),inplace=True)


list_num_obj.remove('SalePrice')


list_col_obj+list_num_obj
```

```
train_test_df=pd.concat([train_df.drop('SalePrice',axis=1),test_df],axis=0)
```

```
train_test_df.shape
```

```
fillna_all(train_test_df)
```

```
train_test_df.drop(['Alley','FireplaceQu','PoolQC','Fence','MiscFeature'],axis=1,inplace=True
```

```
sns.heatmap(train_test_df.isnull())
```

```
train_test_df.info()
```

```
list_col_obj.remove('Alley')


list_col_obj.remove('FireplaceQu')
list_col_obj.remove('PoolQC')
list_col_obj.remove('Fence')
list_col_obj.remove('MiscFeature')


onehot_encode=pd.get_dummies(train_test_df[list_col_obj],prefix=list_col_obj)


onehot_encode.shape
```

```
onehot_encode.head()
```

```
train_test_df.drop(list_col_obj,axis=1,inplace=True)
```

```
train_test_df_final=pd.concat([train_test_df,onehot_encode],axis=1)
```

```
train_test_df_final.shape
```

```
X_train=train_test_df_final.iloc[0:1460]
X_test=train_test_df_final.iloc[1460:]
```

```
X_train.shape , X_test.shape
```

```
y_train=train_df['SalePrice']
```

```
from sklearn.ensemble import RandomForestRegressor
model=RandomForestRegressor(random_state=23)
```

```
model.fit(X_train,y_train)
```

```
y_pred=model.predict(X_test)
```

```
y_pred
```

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

#splitting the dataset as training and testing dataset
X_train1, X_test1, y_train1, y_test1 = train_test_split(X_train, y_train)

#building the model
```

```
linreg = LinearRegression()
linreg.fit(X_train1, y_train1)

#Accuracy
print("R-Squared Value for Training Set: {:.3f}".format(linreg.score(X_train1, y_train1)))
print("R-Squared Value for Test Set: {:.3f}".format(linreg.score(X_test1, y_test1)))
```

```
y_pred1=linreg.predict(X_test)
```

```
y_pred1
```

```
from sklearn.neighbors import KNeighborsRegressor

knnreg = KNeighborsRegressor(n_neighbors = 2)
knnreg.fit(X_train1, y_train1)

print('R-squared train score: {:.3f}'.format(knnreg.score(X_train1, y_train1)))
print('R-squared test score: {:.3f}'.format(knnreg.score(X_test1, y_test1)))
```

```
import xgboost
regressor=xgboost.XGBRegressor()


n_estimators = [100, 500, 900, 1100, 1500]
max_depth = [2, 3, 5, 10, 15]
booster=['gbtree','gblinear']
learning_rate=[0.05,0.1,0.15,0.20]
min_child_weight=[1,2,3,4]
base_score=[0.25,0.5,0.75,1]

# Define the grid of hyperparameters to search
hyperparameter_grid = {
    'n_estimators': n_estimators,
    'max_depth':max_depth,
    'learning_rate':learning_rate,
    'min_child_weight':min_child_weight,
    'booster':booster,
    'base_score':base_score
    }
random_cv = RandomizedSearchCV(estimator=regressor,
```

```
                param_distributions=hyperparameter_grid,
                cv=5, n_iter=50,
                scoring = 'neg_mean_absolute_error',n_jobs = 4,
                verbose = 5,
                return_train_score = True,
                random_state=42)


random_cv.fit(X_train,y_train)
```

```
random_cv.best_estimator_
```

```
reg=xgboost.XGBRegressor(base_score=0.25, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.1, max_delta_step=0,
```

```
                max_depth=2, min_child_weight=1, missing=None, n_estimators=900,
                n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
                reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
                silent=None, subsample=1, verbosity=1)
```

```
reg.fit(X_train,y_train)
```

```
y_pred2=reg.predict(X_test)
```

```
y_pred2
```

```
#submission_file
```

```
submission1_df=pd.DataFrame({'Id':test_df['Id'],'SalePrice':y_pred1})
```

```
submission2_df=pd.DataFrame({'Id':test_df['Id'],'SalePrice':y_pred2})
```

```
submission1_df
```

```
submission2_df
```

```
submission1_df.to_csv('submission1.csv',index=False)
```

```
submission2_df.to_csv('submission2.csv',index=False)
```

```
submission1=pd.read_csv('submission1.csv')
```

```
submission2=pd.read_csv('submission2.csv')
```

```
submission1
```

```
submission2
```

```
submission1.to_csv('submission1.csv',index=False)
```

```
submission2.to_csv('submission2.csv',index=False)
```

```
! ls
```

```
! cat submission1.csv
```

```
! cat submission2.csv
```