MACHINE LEARNING ASSIGNMENT

Group – 8

Project Topic

Use the 'Real E-state Valuation’ database:

use linear regression, and logistic regression to justify the outcome using the database.

Code Implementation

**#Importing necessary libraries**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression, LogisticRegression

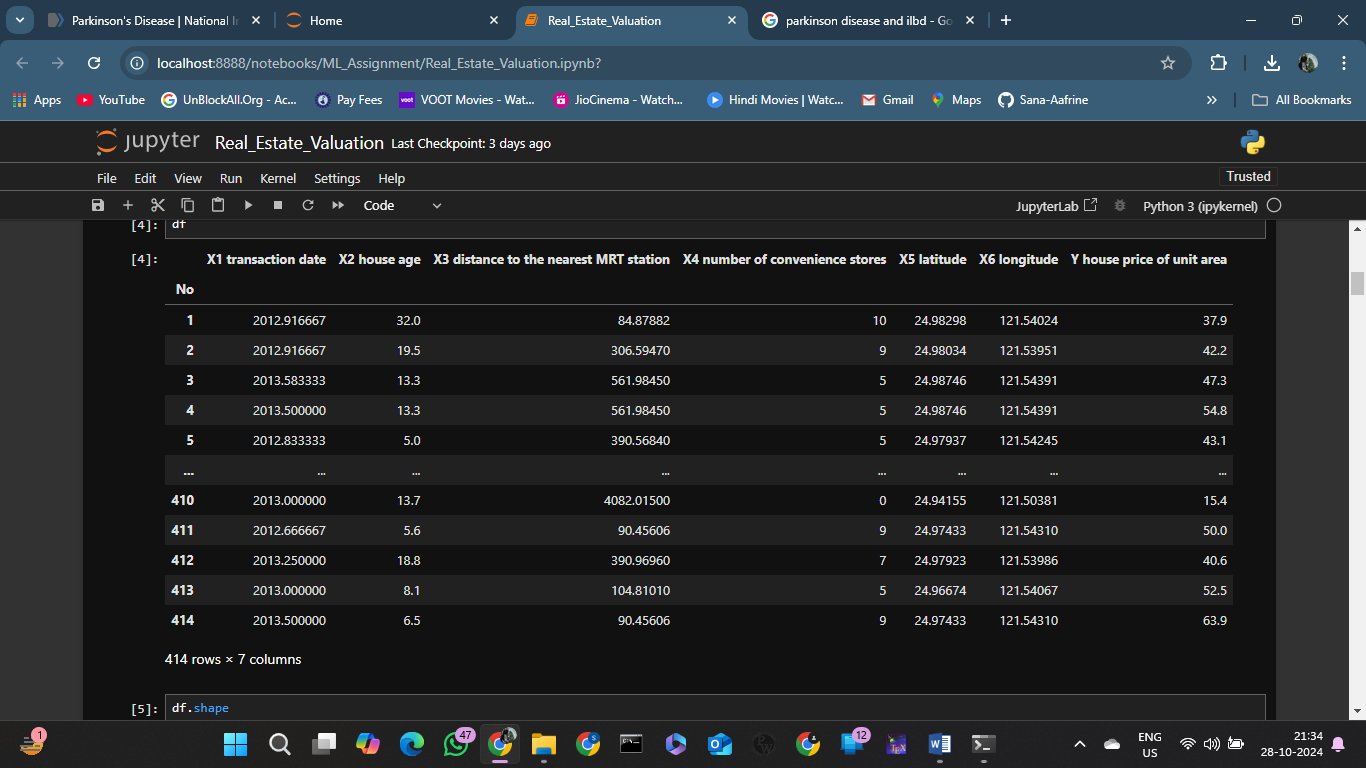
from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import root\_mean\_squared\_error, r2\_score, accuracy\_score, precision\_score, recall\_score, f1\_score, mean\_absolute\_error

**# Importing the dataset**

df = pd.read\_excel('real\_estate\_valuation.xlsx', index\_col=0)

df



**#Analyzing the dataset**

df.shape

df.describe()

df.info()

**#Finding missing values**

df.isna().sum()

**#Extracting transaction year and dropping transaction date**

df['transaction year'] = df['X1 transaction date'].apply(lambda x: int(x))

df = df.drop(columns = 'X1 transaction date')

df.info()

**#Histogram plot to visualize distribution of data**

plt.figure(figsize=(15, 10))

for i, j in enumerate(df.columns, 1):

plt.subplot(4, 2, i)

sns.histplot(df[j], kde=True, bins=30)

plt.title(f'Distribution of {j}')

plt.xlabel(j)

plt.ylabel('Frequency')

plt.tight\_layout()

plt.show()

**#Boxplot for better visualization of outliers in features and their spread**

plt.figure(figsize=(15, 10))

for i, j in enumerate(df.columns):

plt.subplot(3, 3, i+1)

sns.boxplot(x=df[j])

plt.title(f'Boxplot for {j}')

plt.tight\_layout()

plt.show()

**#Logarithmic transformation of ‘distance to nearest MRT station’ to counteract skewness and handle outliers**

df['X3 distance to the nearest MRT station'] = np.log1p(df['X3 distance to the nearest MRT station'])

df['X3 distance to the nearest MRT station'].hist(bins=50)

sns.boxplot(x=df['X3 distance to the nearest MRT station'])

**#Forming a correlation matrix and generating a heatmap**

corr\_matrix = df.corr()

sns.heatmap(corr\_matrix, annot=True, cmap='coolwarm')

plt.show()

**#splitting data into 75-25% training and test set**

X = df.drop(columns = 'Y house price of unit area')

Y = df['Y house price of unit area']

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X,Y,

test\_size = 0.25,

random\_state = 42)

**#Normalization of the features using StrandardScaler**

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

**#Using GridSearch to find the optimal parameters for linear regression**

linear\_regression\_params = {

'fit\_intercept': [True, False],

'copy\_X': [True, False]

}

linear\_regression\_grid = GridSearchCV(estimator=LinearRegression(), param\_grid=linear\_regression\_params,

scoring='r2', cv=5, n\_jobs=-1)

linear\_regression\_grid.fit(X\_train\_scaled, Y\_train)

best\_lr\_model = linear\_regression\_grid.best\_estimator\_

best\_lr\_params = linear\_regression\_grid.best\_params\_

print("Best Linear Regression Parameters:", best\_lr\_params)

**#Linear regression with optimal parameters**

lin\_reg = LinearRegression() #default hyperparameter settings are found to be the best

lin\_reg.fit(X\_train\_scaled, Y\_train)

y\_pred\_linear = lin\_reg.predict(X\_test\_scaled)

**#Evaluation of linear regression model**

linear\_rmse = root\_mean\_squared\_error(Y\_test, y\_pred\_linear)

linear\_r2 = r2\_score(Y\_test, y\_pred\_linear)

linear\_mae = mean\_absolute\_error(Y\_test, y\_pred\_linear)

print(f'Root Mean Squared Error : {linear\_rmse}\nR2-score : {linear\_r2}\nMean Absolute Error : {linear\_mae}')

**#Binarization of target value**

median\_price = Y\_train.median()

y\_train\_binary = (Y\_train >= median\_price).astype(int)

y\_test\_binary = (Y\_test >= median\_price).astype(int)

**#Using GridSearch to find optimal parameters for logistic regression model**

logistic\_regression\_params = {

'max\_iter' : [1000, 5000, 10000]

}

logistic\_regression\_grid = GridSearchCV(estimator=LogisticRegression(random\_state = 42), param\_grid=logistic\_regression\_params,

scoring='accuracy', cv=5, n\_jobs=-1)

logistic\_regression\_grid.fit(X\_train\_scaled, y\_train\_binary)

best\_log\_model = logistic\_regression\_grid.best\_estimator\_

best\_log\_params = logistic\_regression\_grid.best\_params\_

print("Best Linear Regression Parameters:", best\_log\_params)

**#Logistic regression with optimal parameters**

log\_reg = LogisticRegression(max\_iter=1000, class\_weight = 'balanced', random\_state=42)

log\_reg.fit(X\_train\_scaled, y\_train\_binary)

y\_pred\_logistic = log\_reg.predict(X\_test\_scaled)

**#Evaluation of logistic regression model**

logistic\_accuracy = accuracy\_score(y\_test\_binary, y\_pred\_logistic)

logistic\_precision = precision\_score(y\_test\_binary, y\_pred\_logistic)

logistic\_recall = recall\_score(y\_test\_binary, y\_pred\_logistic)

logistic\_f1 = f1\_score(y\_test\_binary, y\_pred\_logistic)

print(f'Accuracy : {logistic\_accuracy}\nPrecision : {logistic\_precision}\nRecall : {logistic\_recall}\nF1-score : {logistic\_f1}')

**#Printing results of both models in table format**

performance\_results = {

'Metric': [

'Root Mean Squared Error',

'Mean Absolute Error',

'R^2 Score',

'Accuracy',

'Precision',

'Recall',

'F1 Score'

],

'Linear Regression': [

linear\_rmse,

linear\_mae,

linear\_r2,

None, # No value for linear regression metrics

None,

None,

None

],

'Logistic Regression': [

None, # No value for logistic regression metrics

None,

None,

logistic\_accuracy,

logistic\_precision,

logistic\_recall,

logistic\_f1

]

}

results\_df = pd.DataFrame(performance\_results)

results\_df

