Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

PythonProject_1.ipynb

C: > Users > USER > Downloads > PythonProject_1.ipynb > M↓#Earthquake Prediction

+ Code + Markdown ··· Select Kernel

#Earthquake Prediction Gayathry M Wariyar
AM.EN.U4AIE21130

```python
import numpy as np
import pandas as pd
import requests
from sklearn import preprocessing
import matplotlib.pyplot as plt
import seaborn as sns
from pandas.plotting import scatter_matrix
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
import time
```

[398] Python

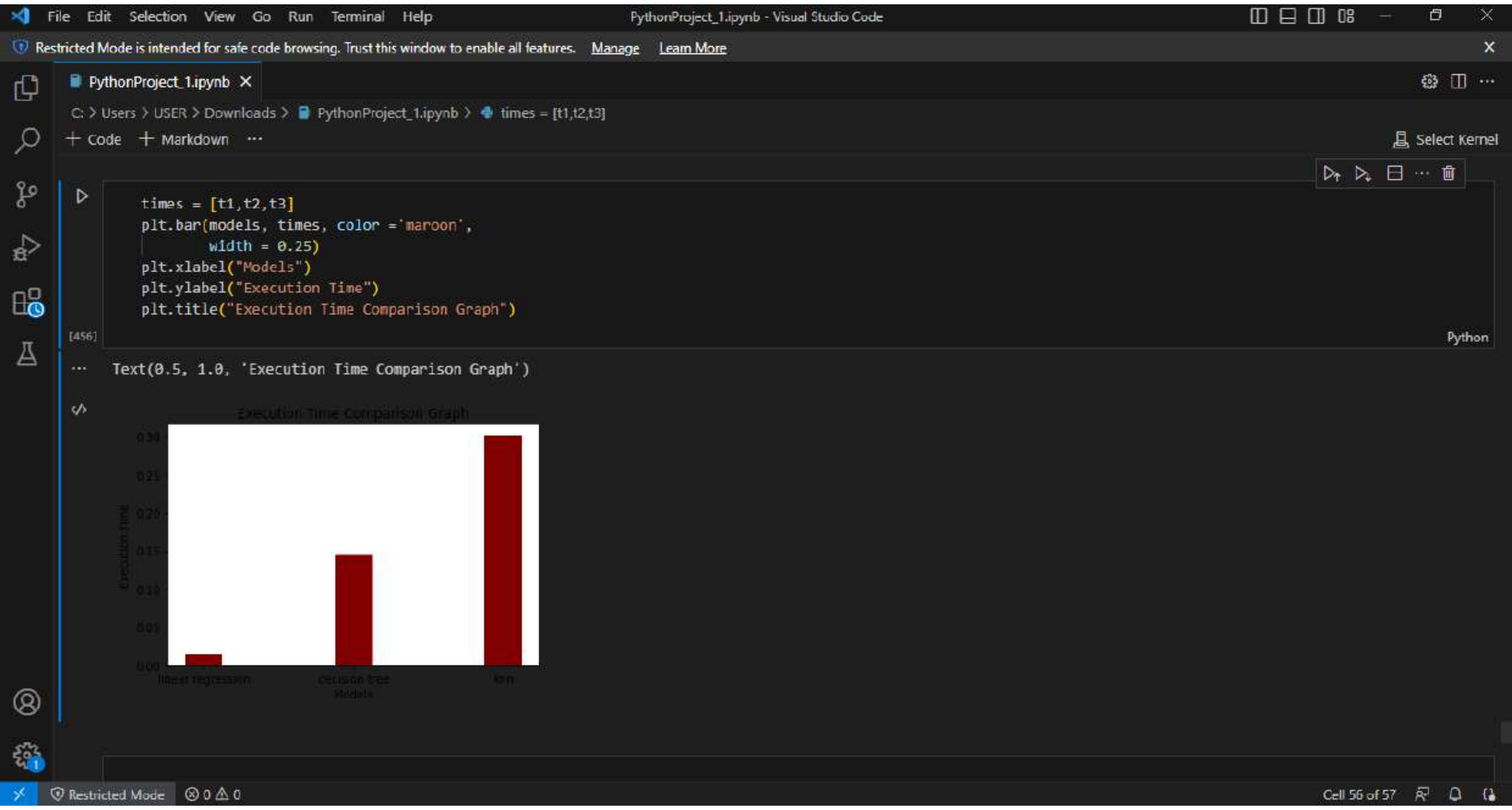
```python
from google.colab import drive
drive.mount('/content/drive')
df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/earthquake_prediction/earthquake1.csv")
```

[399] Python

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```python
df.info()
```

Restricted Mode ⊗ 0 ⚠ 0 Cell 1 of 57


File Edit Selection View Go Run Terminal Help
PythonProject_1.ipynb - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
PythonProject_1.ipynb
C: > Users > USER > Downloads > PythonProject_1.ipynb > times = [t1,t2,t3]
+ Code + Markdown
Select Kernel
times = [t1,t2,t3]
plt.bar(models, times, color ='maroon',
width = 0.25)
plt.xlabel("Models")
plt.ylabel("Execution Time")
plt.title("Execution Time Comparison Graph")
[456]
Python
Text(0.5, 1.0, 'Execution Time Comparison Graph')
Restricted Mode
Cell 56 of 57

PythonProject_1.ipynb

C: > Users > USER > Downloads > PythonProject_1.ipynb > plt.plot(y_test, ans1, 'o')

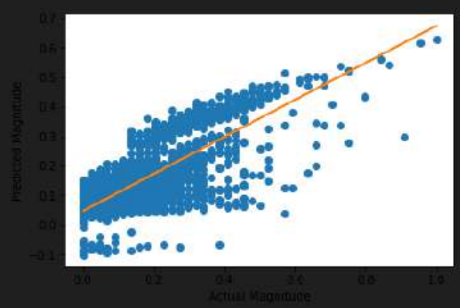
+ Code + Markdown ··· Select Kernel

```
Mean Absolute Error: 0.05878246463205686
Mean Squared Error: 0.00625827169726636
Root Mean Squared Error: 0.07910923901331854
```

```python
plt.plot(y_test, ans1, 'o')
m, b = np.polyfit(y_test,ans1, 1)
plt.plot(y_test, m*y_test + b)
plt.xlabel("Actual Magnitude")
plt.ylabel("Predicted Magnitude")
```

[428] Python

```
Text(0, 0.5, 'Predicted Magnitude')
```



Restricted Mode ⊗ 0 ⚠ 0 Cell 39 of 57

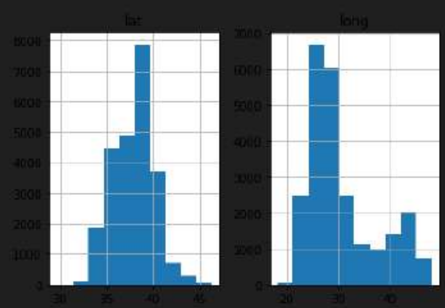
PythonProject_1.ipynb

C: > Users > USER > Downloads > PythonProject_1.ipynb > from sklearn import metrics

+ Code + Markdown ··· Select Kernel

[413]
```python
plt.figure(figsize=(7,7))
df[['lat','long']].hist()
plt.show()
```
Python

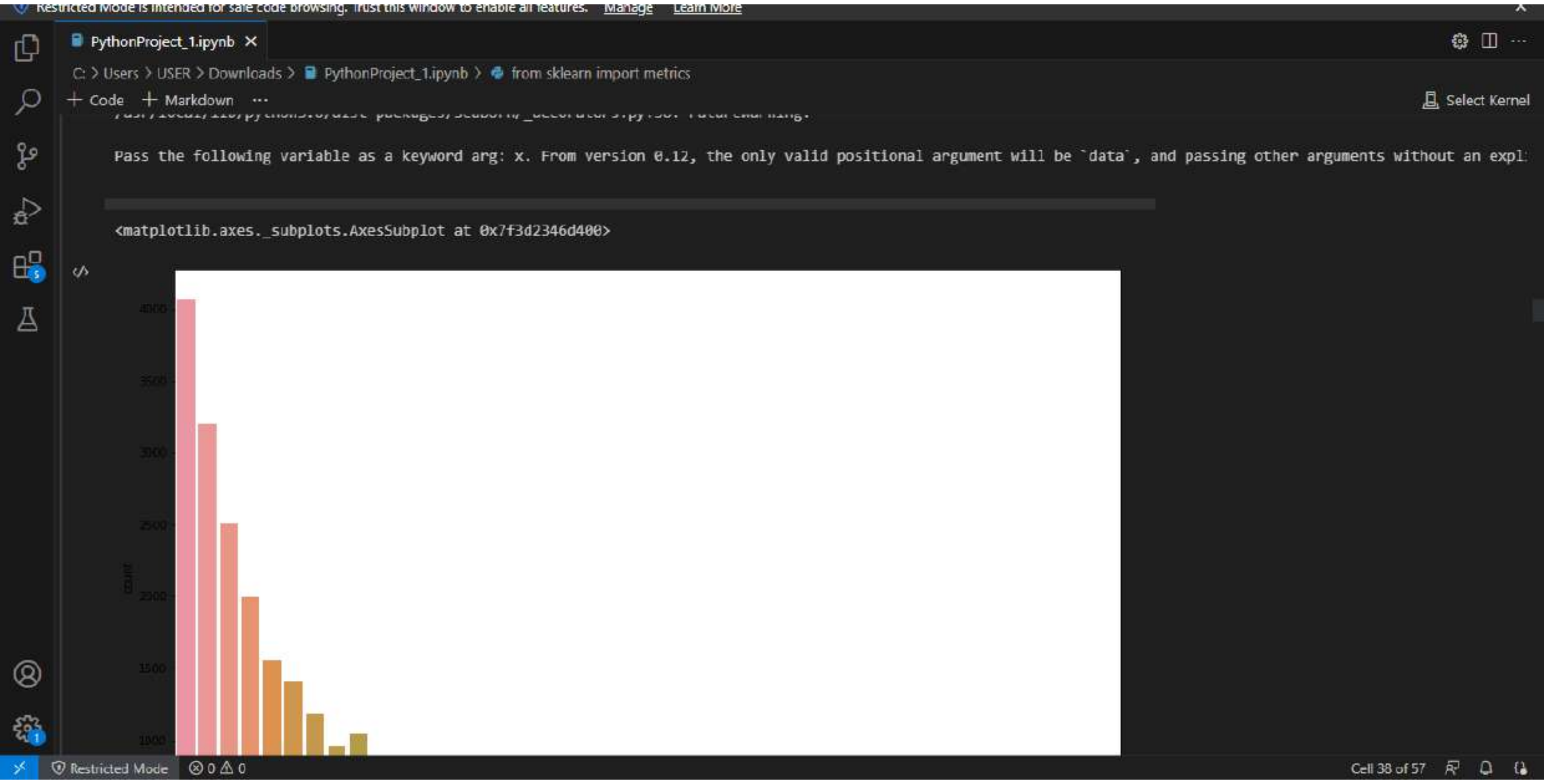
```
<Figure size 504x504 with 0 Axes>
```

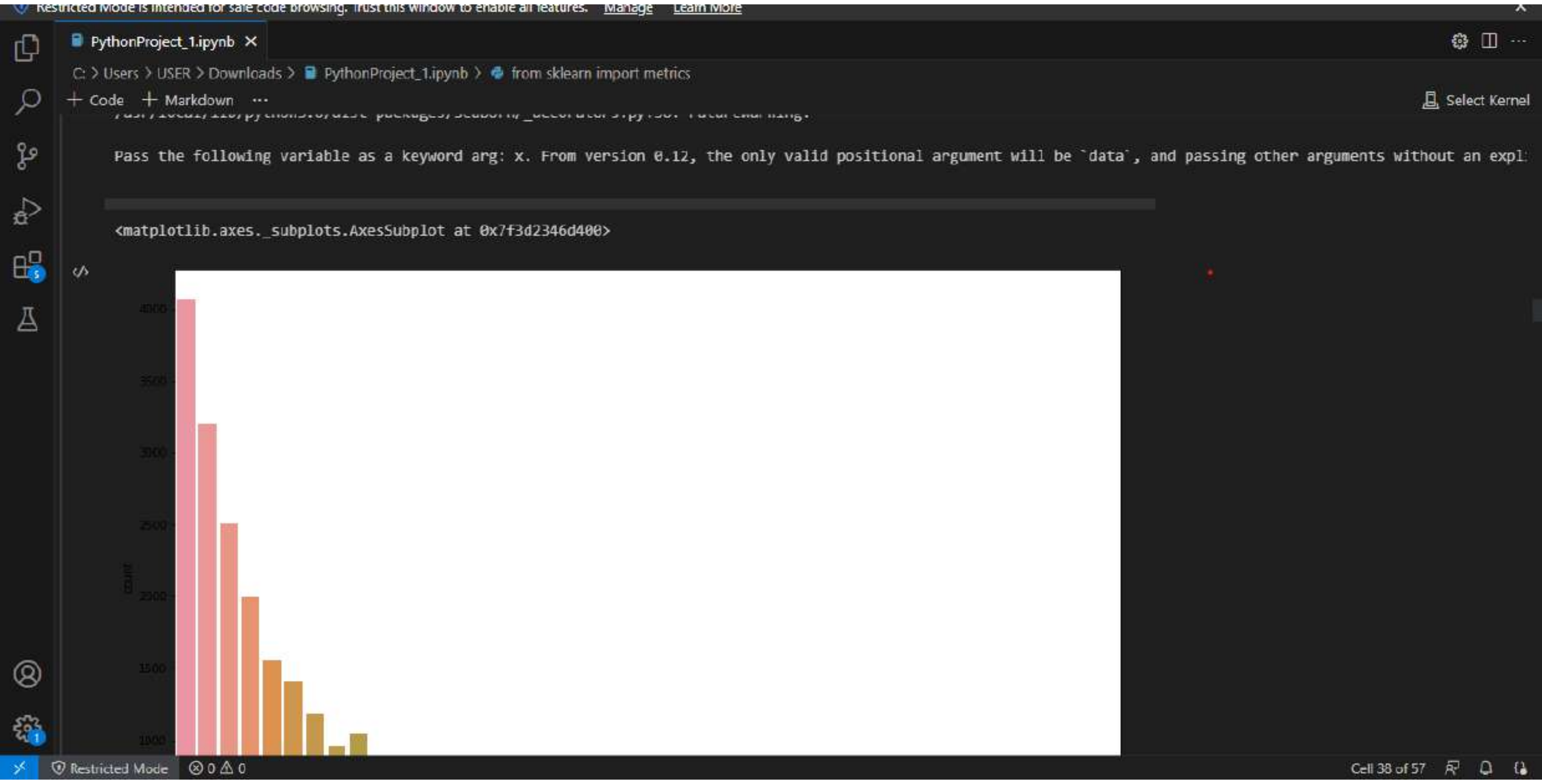


[414]
```python
plt.figure(figsize=(15,10))
sns.countplot(df.xm)
```
Python

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWarning:
```

Restricted Mode ⊗ 0 ⚠ 0 Cell 38 of 57

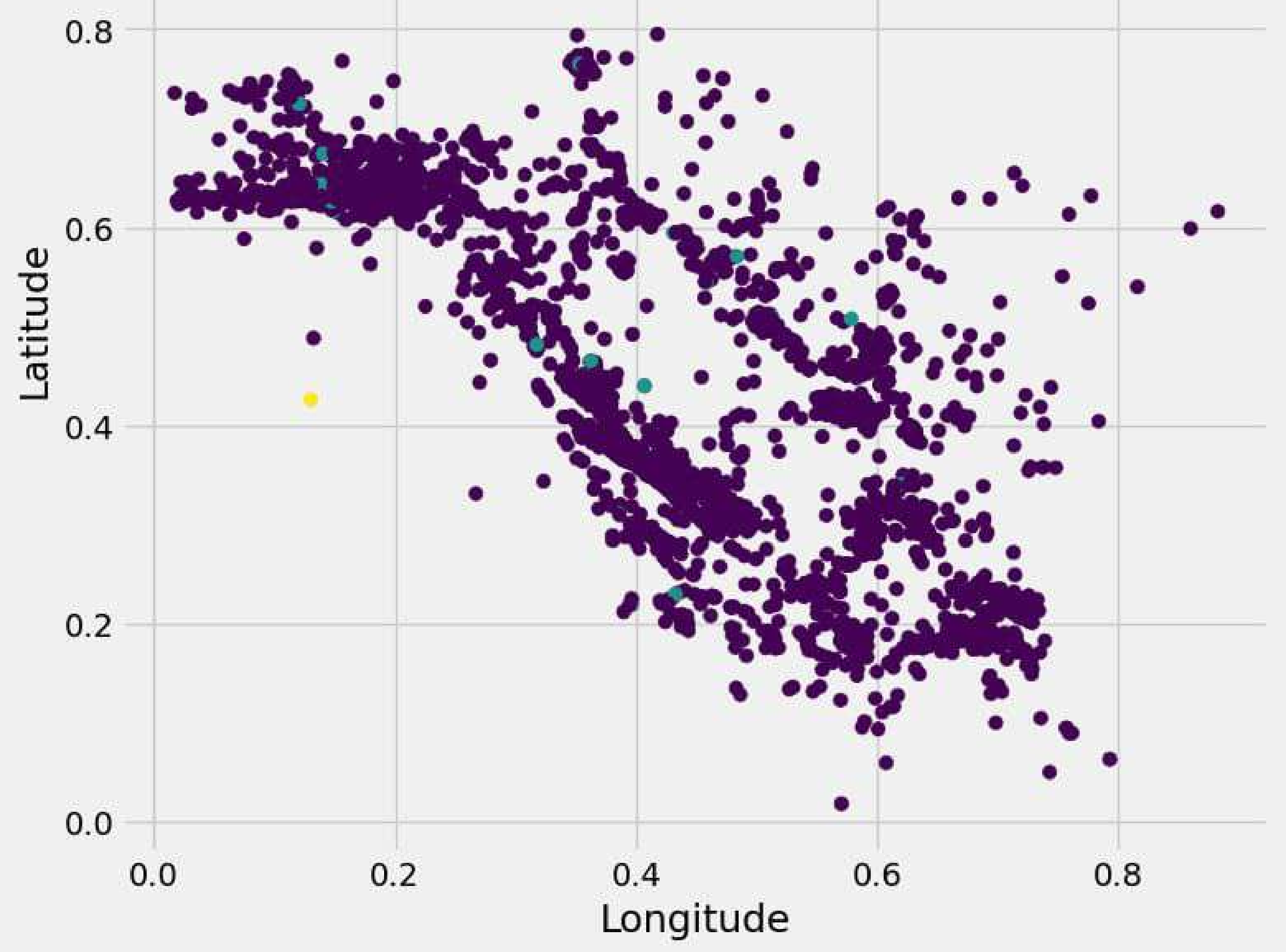

Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
PythonProject_1.ipynb
C: > Users > USER > Downloads > PythonProject_1.ipynb > from sklearn import metrics
Code
Markdown
Select Kernel
Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an expl
<matplotlib.axes._subplots.AxesSubplot at 0x7f3d2346d400>
4000
3500
3000
2500
2000
1500
1000
count
Restricted Mode
0
0
Cell 38 of 57

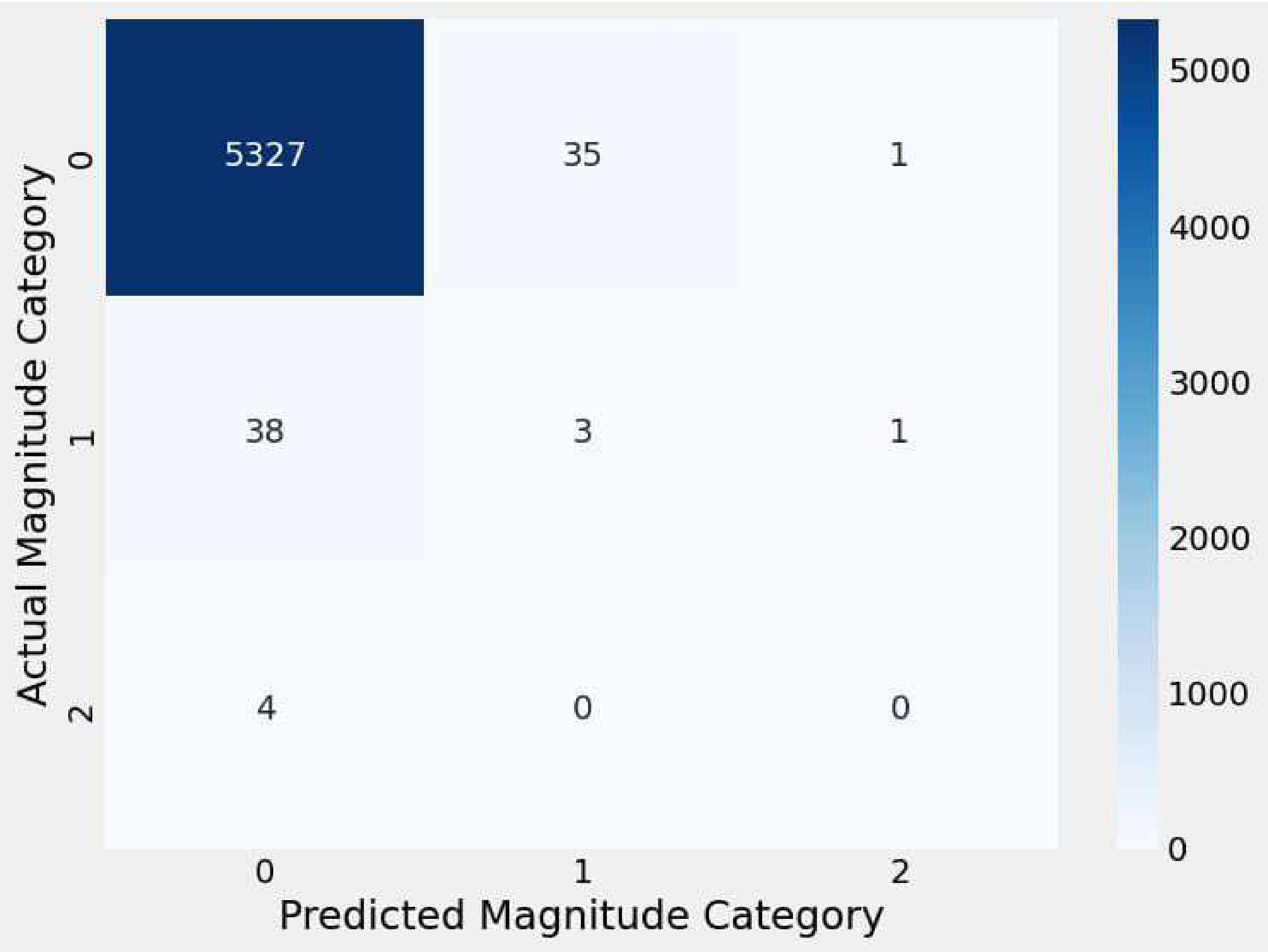


PythonProject_1.ipynb
C: > Users > USER > Downloads > PythonProject_1.ipynb > from sklearn import metrics
+ Code + Markdown ···
Select Kernel
Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an expl
<matplotlib.axes._subplots.AxesSubplot at 0x7f3d2346d400>
Restricted Mode
Cell 38 of 57

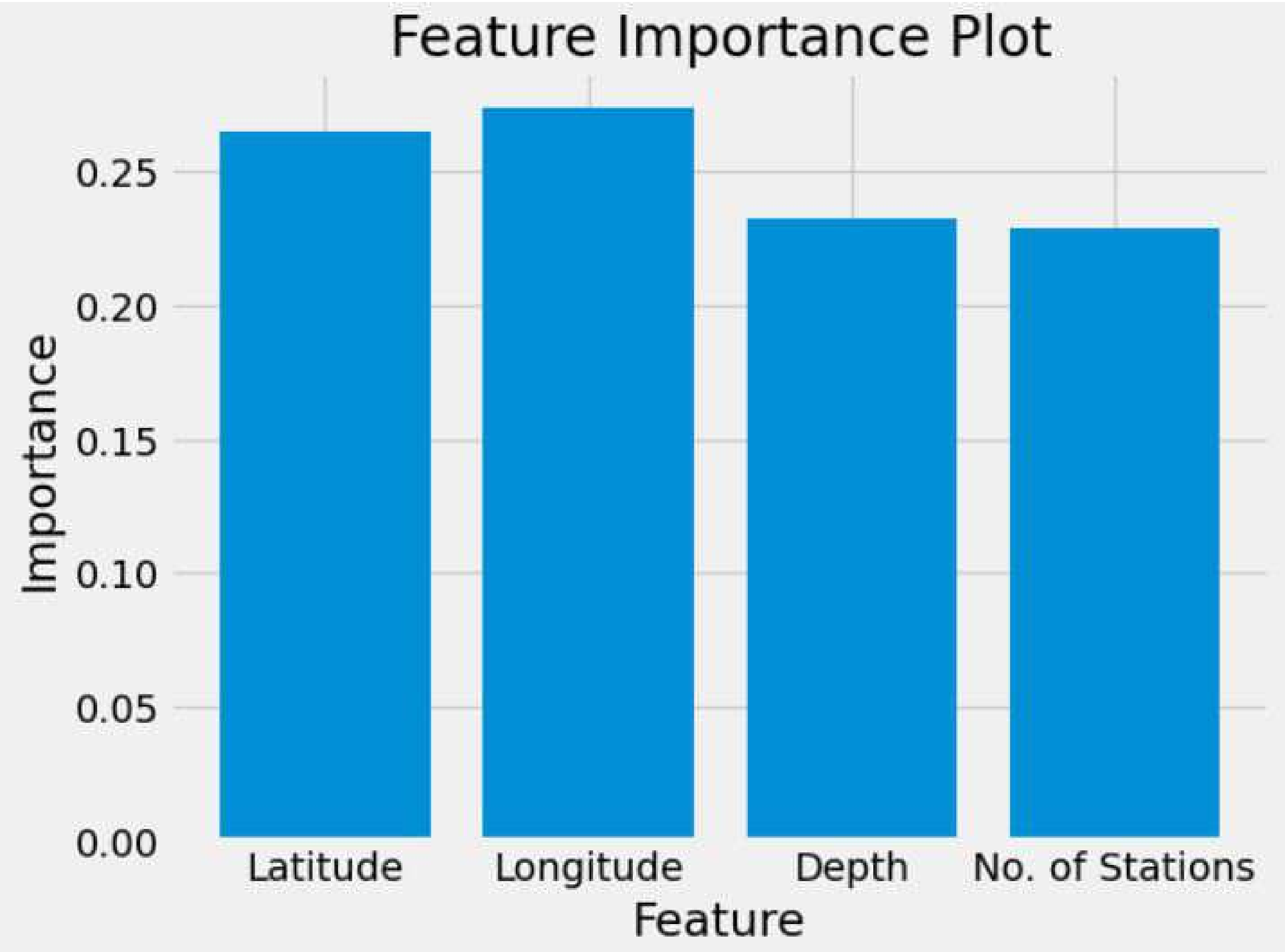
10:09 PM 10.6KB/s 13%

229195883-8aec...
es.githubusercontent.com


Predicted Magnitude Category
1.0
0.8
0.6
0.4
0.2
0.0
Latitude
0.0
0.2
0.4
0.6
0.8
Longitude



5327
35
1
38
3
1
4
0
0
Actual Magnitude Category
Predicted Magnitude Category
0
1
2
0
1
2
5000
4000
3000
2000
1000
0


Feature Importance Plot
Importance
0.25
0.20
0.15
0.10
0.05
0.00
Latitude
Longitude
Depth
No. of Stations
Feature

V. EVALUATION AND DISCUSSIONS

DATA PRE-PROCESSING

For both the datasets the following pre-processing steps were taken:

- First, we dropped the 'id' column since it is not useful for prediction.

```
df = df.drop('id',axis=1)
```

- Next we converted categorical variables into numerical values so that it could be easily fitted to a machine learning model using a label encoder.

```
# Data Encoding
label_encoder = preprocessing.LabelEncoder()
for col in df.columns:
    if df[col].dtype == 'object':
      label_encoder.fit(df[col])
      df[col] = label_encoder.transform(df[col])
df.dtypes
```

- Imputation with SimpleImputer which replaced the missing data with mean was done to handle null values.

```
# Imputing Missing Values with Mean
si=SimpleImputer(missing_values = np.nan, strategy="mean")
si.fit(df[["dist","mw"]])
df[["dist","mw"]] = si.transform(df[["dist","mw"]])
df.isnull().sum()
```

- Normalization of the data was done using MinMax

```
import datetime
import time

timestamp = []
for d, t in zip(df['date'], df['time']):
  ts = datetime.datetime.strptime(d+' '+t, '%Y.%m.%d %I:%M:%S %p')
  timestamp.append(time.mktime(ts.timetuple()))
timeStamp = pd.Series(timestamp)
df['Timestamp'] = timeStamp.values
final_data = df.drop(['date', 'time'], axis=1)
final_data = final_data[final_data.Timestamp != 'ValueError']
df = final_data
df.head()
```

SPLITTING THE DATASET

The datasets are split into training and testing datasets.

The dataset 1 is split into 80% training and 20% testing, with a random state of 2, meaning that the same random sample will be used each time the code is run.

```
y=np.array(df['xm'])
X=np.array(df.drop('xm',axis=1))
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=2)
```

The dataset 2 is split into 75% and 25% for training and testing respectively with a random state of 2.

```
y=np.array(df['xm'])
X=np.array(df.drop('xm',axis=1))
```

B. Decision Tree

```
from sklearn.tree import DecisionTreeRegressor
start2 = time.time()
regressor = DecisionTreeRegressor(random_state = 40)
regressor.fit(X_train,y_train)
ans2 = regressor.predict(X_test)
end2 = time.time()
t2 = end2-start2
```

C. KNN

```
from sklearn.neighbors import KNeighborsRegressor
start3 = time.time()
knn = KNeighborsRegressor(n_neighbors=6)
knn.fit(X_train, y_train)
ans3 = knn.predict(X_test)
end3 = time.time()
t3 = end3-start3
```

RESULTS

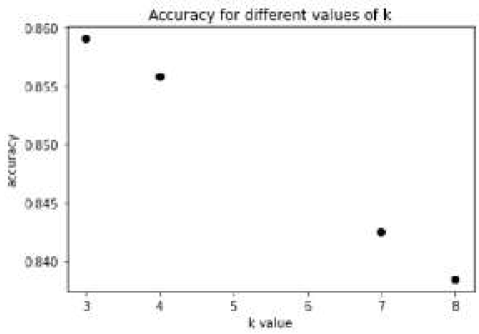
DATASET 1: earthquake1.csv

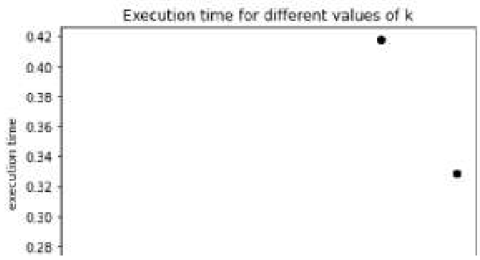
Linear Regression Model

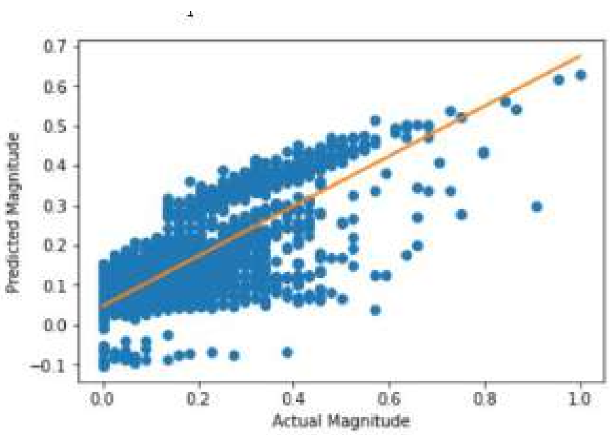
- Accuracy of Linear Regression model is: 0.63134131503029
- Mean Absolute Error: 0.05878246463205686
- Mean Squared Error: 0.00625827169726636
- Root Mean Squared Error: 0.07910923901331854

KNN Model

- Accuracy of KNN model is: 0.8457466919393031
- Mean Absolute Error: 0.03305598677318794
- Mean Squared Error: 0.002618571462992348
- Root Mean Squared Error: 0.051171979275696854

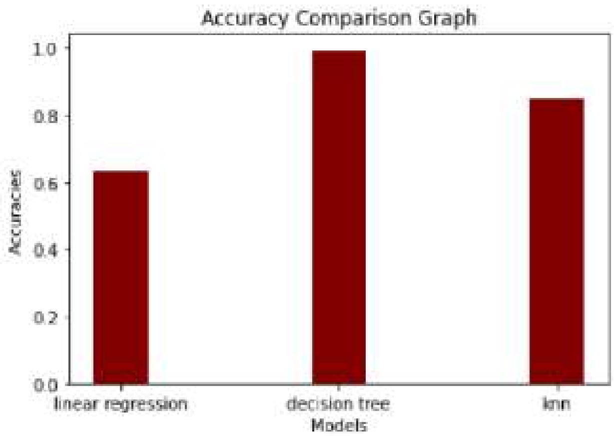


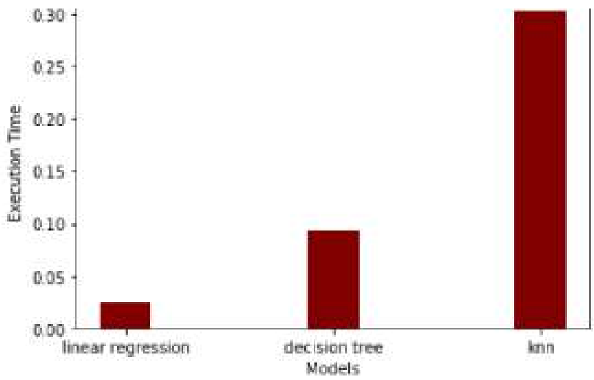


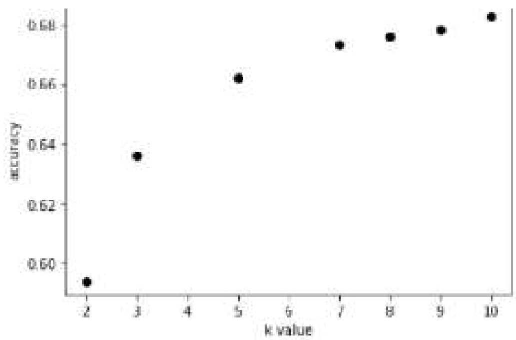


Decision Tree Model

- Accuracy of Decision Tree model is: 0.9932960893884235.
- Mean Absolute Error: 0.0006909999621372331
- Mean Squared Error: 0.00011380416561969702
- Root Mean Squared Error: 0.010667903525046383







DATASET 2: earthquake2.csv

Linear Regression Model

- Accuracy of Linear Regression model is: 0.6520026760336074
- Mean Absolute Error: 0.0447298826759214
- Mean Squared Error: 0.0034475163509273773
- Root Mean Squared Error: 0.058715554590988726