earthquakemagnitudepredicton

October 4, 2023

0.1 Importing Required Packaged

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import geopandas as gpd
import cufflinks as cf

%matplotlib inline
```

1 1) Data Source

[2]:
```
data = pd.read_csv("database.csv",
                   header=0,
                   index_col=None,
                   sep=","
                  )
```

[3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23412 entries, 0 to 23411
Data columns (total 21 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Date                        23412 non-null  object
 1   Time                        23412 non-null  object
 2   Latitude                    23412 non-null  float64
 3   Longitude                   23412 non-null  float64
 4   Type                        23412 non-null  object
 5   Depth                       23412 non-null  float64
 6   Depth Error                 4461 non-null   float64
 7   Depth Seismic Stations      7097 non-null   float64
 8   Magnitude                   23412 non-null  float64
 9   Magnitude Type              23409 non-null  object
```

```
 10  Magnitude Error             327 non-null    float64
 11  Magnitude Seismic Stations  2564 non-null   float64
 12  Azimuthal Gap               7299 non-null   float64
 13  Horizontal Distance         1604 non-null   float64
 14  Horizontal Error            1156 non-null   float64
 15  Root Mean Square            17352 non-null  float64
 16  ID                          23412 non-null  object
 17  Source                      23412 non-null  object
 18  Location Source             23412 non-null  object
 19  Magnitude Source            23412 non-null  object
 20  Status                      23412 non-null  object
dtypes: float64(12), object(9)
memory usage: 3.8+ MB
```

1.0.1 Required Feautures

- **Latitude**
- **Longitude**
- **Depth**
- **Depth Error**
- **Root Mean Square**

[4]:
```
data = data[["Latitude","Longitude","Root Mean Square","Depth","Depth␣
↪Error","Magnitude"]]
```

[5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23412 entries, 0 to 23411
Data columns (total 6 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Latitude          23412 non-null  float64
 1   Longitude         23412 non-null  float64
 2   Root Mean Square  17352 non-null  float64
 3   Depth             23412 non-null  float64
 4   Depth Error       4461 non-null   float64
 5   Magnitude         23412 non-null  float64
dtypes: float64(6)
memory usage: 1.1 MB
```

[6]:
```
data.describe()
```

[6]:

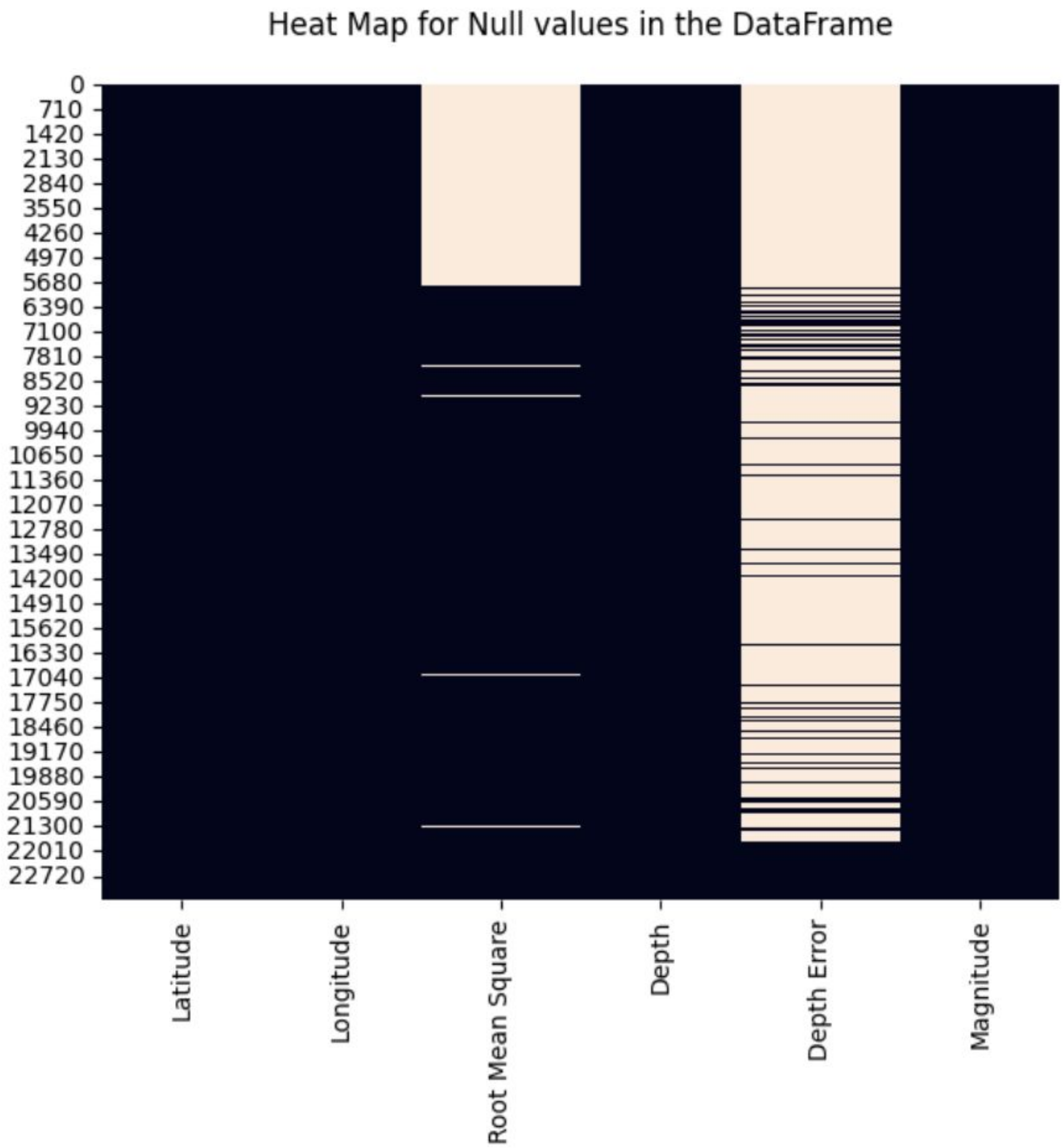
	Latitude	Longitude	Root Mean Square	Depth	\
count	23412.000000	23412.000000	17352.000000	23412.000000	
mean	1.679033	39.639961	1.022784	70.767911	
std	30.113183	125.511959	0.188545	122.651898	
min	-77.080000	-179.997000	0.000000	-1.100000	

```
25%      -18.653000    -76.349750            0.900000     14.522500
50%       -3.568500    103.982000            1.000000     33.000000
75%       26.190750    145.026250            1.130000     54.000000
max       86.005000    179.998000            3.440000    700.000000

       Depth Error     Magnitude
count  4461.000000  23412.000000
mean      4.993115      5.882531
std       4.875184      0.423066
min       0.000000      5.500000
25%       1.800000      5.600000
50%       3.500000      5.700000
75%       6.300000      6.000000
max      91.295000      9.100000
```

2 2) Feauture Exploration

2.1 Exploratory Data Analysis (EDA)

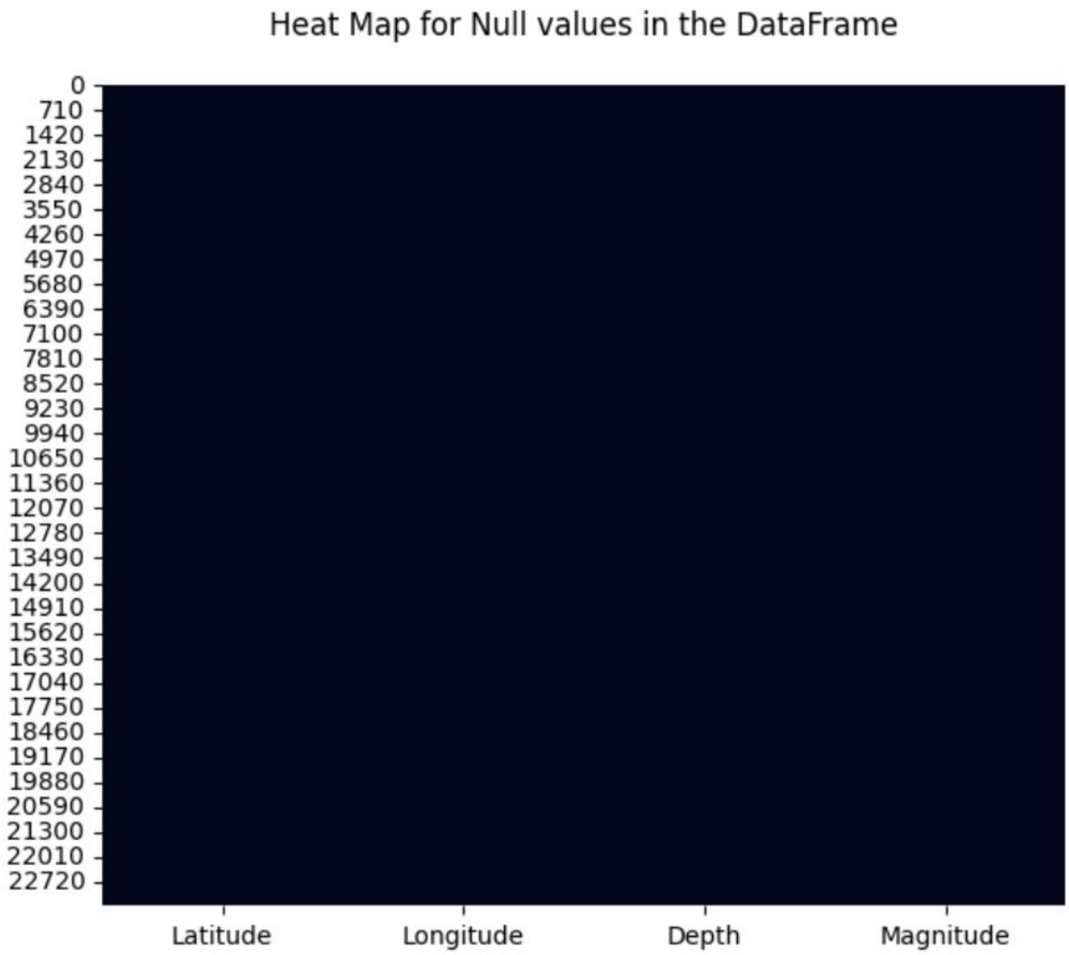
[7]:
```python
plt.figure(figsize=(7,6))
sns.heatmap(data=data.isnull(),
            cbar=False)
txt = plt.title("\nHeat Map for Null values in the DataFrame\n")
```



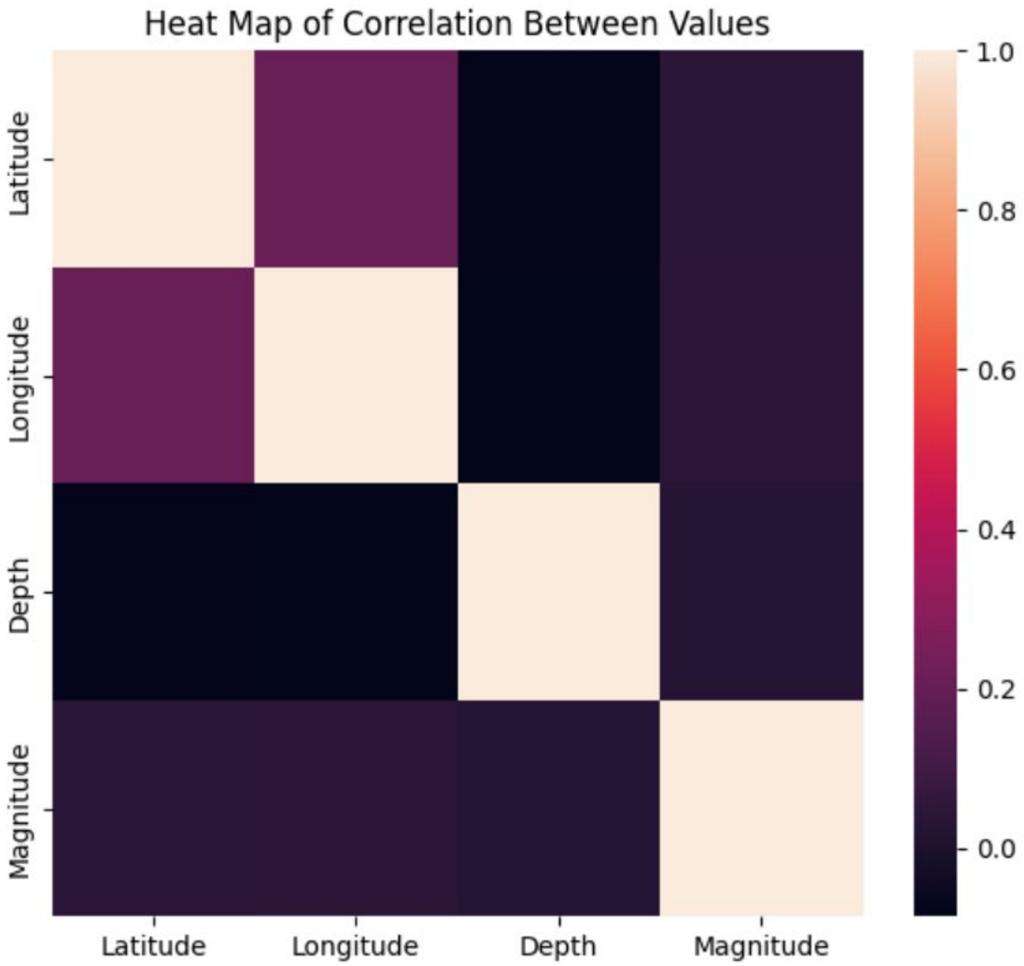

```
Dropping Depth Error And Root Mean Square,  It is having null values and it is not
gonna make much more change in model
```

[8]:
```
data = data.drop(["Depth Error","Root Mean Square"],axis=1)
```

[9]:
```
plt.figure(figsize=(7,6))
sns.heatmap(data=data.isnull(),
            cbar=False)
txt = plt.title("\nHeat Map for Null values in the DataFrame\n")
```



[10]:
```
plt.figure(figsize=(7,6))
sns.heatmap(data=data.corr())
txt = plt.title("Heat Map of Correlation Between Values")
```



[11]:
```
correlation = data['Depth'].corr(data['Magnitude'])
print(f"Correlation Between Depth and Magnitude is {correlation}")
correlation = data['Latitude'].corr(data['Magnitude'])
print(f"Correlation Between Lattitude and Magnitude is {correlation}")
correlation = data['Longitude'].corr(data['Magnitude'])
print(f"Correlation Between Longitude and Magnitude is {correlation}")
```

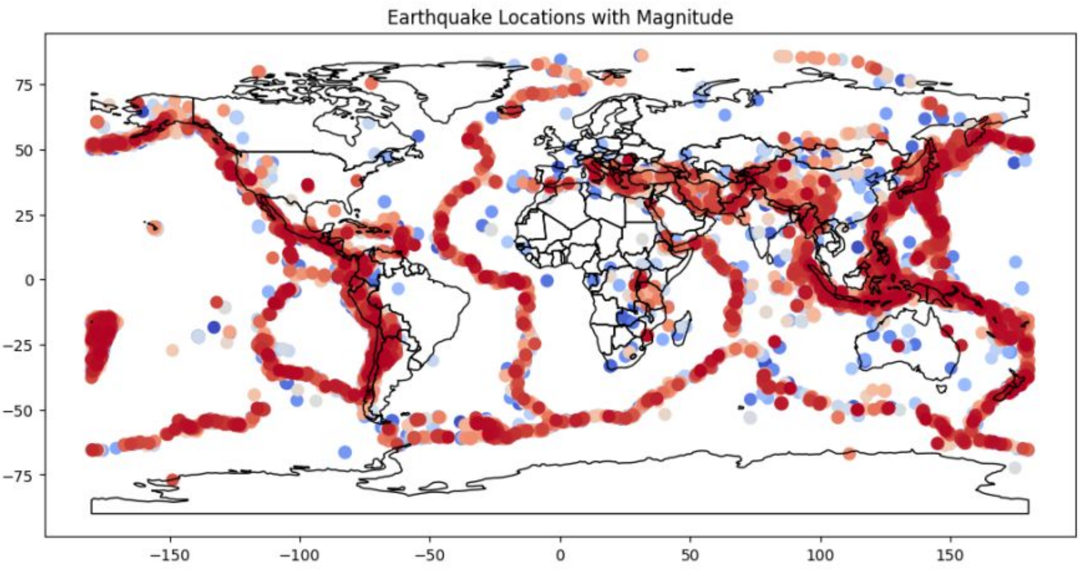
```
Correlation Between Depth and Magnitude is 0.023457312492053895
Correlation Between Lattitude and Magnitude is 0.03498650628261446
Correlation Between Longitude and Magnitude is 0.03857859753074192
```

[ ]:

3 3) Visualization

[12]:
```
df = data
gdf = gpd.GeoDataFrame(df, geometry=gpd.points_from_xy(df.Longitude, df.
 ↪Latitude))

world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))

fig, ax = plt.subplots(figsize=(12, 8))
world.boundary.plot(ax=ax, linewidth=1, color='k')

gdf.plot(ax=ax, markersize=df['Magnitude'] * 10, cmap='coolwarm', legend=True)

ax.set_title('Earthquake Locations with Magnitude')
# plt.legend(title='Magnitude')

plt.show()
```

```
/tmp/ipykernel_33037/249791788.py:4: FutureWarning:

The geopandas.dataset module is deprecated and will be removed in GeoPandas 1.0.
You can get the original 'naturalearth_lowres' data from
https://www.naturalearthdata.com/downloads/110m-cultural-vectors/.
```



[13]:
```
df = pd.DataFrame(data)

fig = df.iplot(
```

```
    kind='scattergeo',
    lon='Longitude',
    lat='Latitude',
    size='Magnitude',
    text='Magnitude',
    colorscale='YlOrRd',
    dimensions=(800, 600),
    title='Earthquake Locations with Magnitude',
    asFigure=True
)

fig.update_geos(
    projection_type="natural earth",
    coastlinecolor="black",
    landcolor="white",
    showland=True,
    showcoastlines=True,
    showocean=True,
    oceancolor="lightblue"
)

# Show the plot
fig.show()
```

[14]:
```
df = pd.DataFrame(data)
plt.figure(figsize=(20,20))
fig = px.scatter_geo(
    df,
    lat='Latitude',
    lon='Longitude',
    color='Magnitude',
    size='Magnitude',
    hover_name='Magnitude',
    projection='natural earth'
)

fig.update_geos(showcoastlines=True, coastlinecolor="Black", showland=True,␣
 ↪landcolor="lightgray")

fig.show()
```

```
<Figure size 2000x2000 with 0 Axes>
```

[ ]:
```

```

4 4) Data Splitting

[15]:
```
from sklearn.model_selection import train_test_split
```

[16]:
```
X = data.drop("Magnitude",axis=1)
y = data["Magnitude"]
```

[17]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,␣
↪random_state=42)
```

[ ]:
```

```

5 5) Model Development

[18]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

```
2023-10-04 18:35:28.257319: I tensorflow/core/util/port.cc:110] oneDNN custom
operations are on. You may see slightly different numerical results due to
floating-point round-off errors from different computation orders. To turn them
off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2023-10-04 18:35:28.284433: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not
find cuda drivers on your machine, GPU will not be used.
2023-10-04 18:35:28.318832: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not
find cuda drivers on your machine, GPU will not be used.
2023-10-04 18:35:28.319665: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other
operations, rebuild TensorFlow with the appropriate compiler flags.
2023-10-04 18:35:29.468449: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not
find TensorRT
```

5.0.1 Scaling the feautures

[19]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[20]:
```
model = keras.Sequential([
    layers.Dense(64, activation='relu', input_shape=(3,)),
    layers.Dense(32, activation='relu'),
```

```
    layers.Dense(1)
])
```

[21]:
```
model.compile(optimizer='adam',
              loss='mean_squared_error',
             )
```

[22]:
```
model.summary()
```

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                256

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 1)                 33

=================================================================
Total params: 2369 (9.25 KB)
Trainable params: 2369 (9.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

6 6) Training and Evaluation

[23]:
```
history = model.fit(X_train,
                    y_train,
                    epochs=25,
                    batch_size=32,
                    validation_split=0.2,
                    validation_data=(X_test,y_test))
```

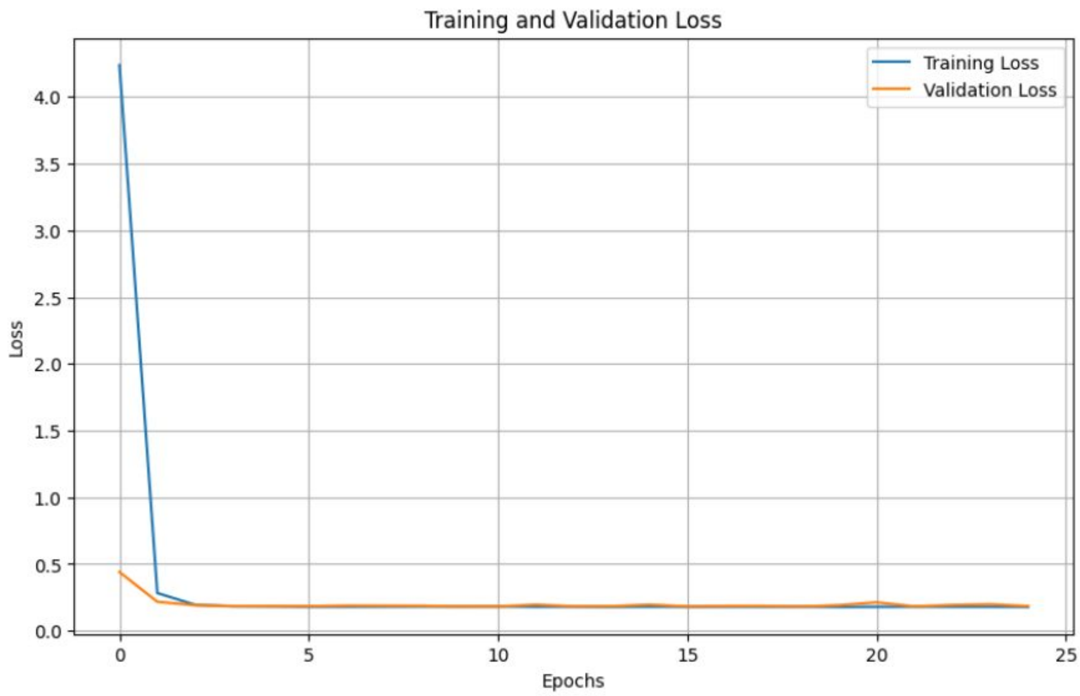
```
Epoch 1/25
491/491 [==============================] - 2s 2ms/step - loss: 4.2382 -
val_loss: 0.4398
Epoch 2/25
491/491 [==============================] - 1s 2ms/step - loss: 0.2826 -
val_loss: 0.2169
Epoch 3/25
491/491 [==============================] - 1s 2ms/step - loss: 0.1940 -
val_loss: 0.1913
Epoch 4/25
491/491 [==============================] - 1s 2ms/step - loss: 0.1831 -
val_loss: 0.1851
Epoch 5/25
```

```
491/491 [==============================] - 1s 2ms/step - loss: 0.1821 -
val_loss: 0.1843
Epoch 6/25
491/491 [==============================] - 1s 2ms/step - loss: 0.1800 -
val_loss: 0.1837
Epoch 7/25
491/491 [==============================] - 1s 2ms/step - loss: 0.1796 -
val_loss: 0.1881
Epoch 8/25
491/491 [==============================] - 1s 2ms/step - loss: 0.1805 -
val_loss: 0.1874
Epoch 9/25
491/491 [==============================] - 1s 2ms/step - loss: 0.1812 -
val_loss: 0.1867
Epoch 10/25
491/491 [==============================] - 1s 2ms/step - loss: 0.1814 -
val_loss: 0.1822
Epoch 11/25
491/491 [==============================] - 1s 2ms/step - loss: 0.1819 -
val_loss: 0.1818
Epoch 12/25
491/491 [==============================] - 1s 2ms/step - loss: 0.1798 -
val_loss: 0.1955
Epoch 13/25
491/491 [==============================] - 1s 2ms/step - loss: 0.1803 -
val_loss: 0.1826
Epoch 14/25
491/491 [==============================] - 1s 1ms/step - loss: 0.1788 -
val_loss: 0.1834
Epoch 15/25
491/491 [==============================] - 1s 2ms/step - loss: 0.1802 -
val_loss: 0.1955
Epoch 16/25
491/491 [==============================] - 1s 2ms/step - loss: 0.1799 -
val_loss: 0.1816
Epoch 17/25
491/491 [==============================] - 1s 1ms/step - loss: 0.1798 -
val_loss: 0.1854
Epoch 18/25
491/491 [==============================] - 1s 2ms/step - loss: 0.1796 -
val_loss: 0.1856
Epoch 19/25
491/491 [==============================] - 1s 1ms/step - loss: 0.1806 -
val_loss: 0.1812
Epoch 20/25
491/491 [==============================] - 1s 2ms/step - loss: 0.1783 -
val_loss: 0.1910
Epoch 21/25
```

```
491/491 [==============================] - 1s 2ms/step - loss: 0.1795 -
val_loss: 0.2128
Epoch 22/25
491/491 [==============================] - 1s 2ms/step - loss: 0.1805 -
val_loss: 0.1824
Epoch 23/25
491/491 [==============================] - 1s 2ms/step - loss: 0.1796 -
val_loss: 0.1927
Epoch 24/25
491/491 [==============================] - 1s 3ms/step - loss: 0.1800 -
val_loss: 0.1982
Epoch 25/25
491/491 [==============================] - 1s 3ms/step - loss: 0.1785 -
val_loss: 0.1841
```

[24]:

```
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')
plt.grid(True)
plt.show()
```



[25]:
```
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error on Test Set: {mse:.2f}")
```

```
242/242 [==============================] - 0s 1ms/step
Mean Squared Error on Test Set: 0.18
```

[26]:
```
model.evaluate(X_test,y_test)
```

```
242/242 [==============================] - 0s 1ms/step - loss: 0.1841
```

[26]:
```
0.18406274914741516
```

6.1 Conclusion:

Mean Squared Error on Test Data : 0.19

[ ]:
```

```