EARTHQUAKE PREDICTION MODEL USING PYTHON

Batch Member:

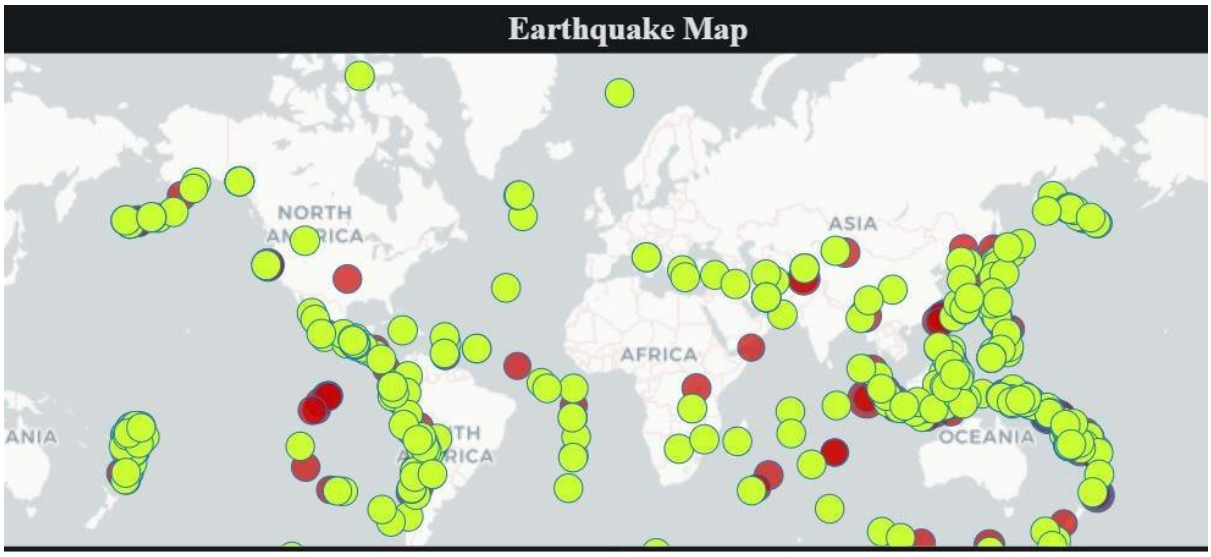
911721106006:GUHAN RAJ R

Phase 3 Submission Document

Project Title: Earth Quake Prediction

Title: Phase 3: Development Part 1

In this part you will begin building your project by loading and preprocessing the dataset.
Begin building the earthquake prediction model by loading and preprocessing the dataset.



Earthquake Prediction Using Python

Earthquake prediction is a challenging and crucial field of study that aims to anticipate the occurrence of seismic events, such as earthquakes and tremors, in specific geographic regions. Earthquakes, with their potential for widespread destruction and loss of life, have been a subject of concern for both scientists and society for many years. The ability to predict earthquakes could provide early warning systems and contribute to improved disaster preparedness and response.

However, it's important to note that earthquake prediction is an incredibly complex and evolving field. While there have been significant advancements in understanding the processes leading to earthquakes, predicting the precise time, location, and magnitude of future seismic events remains an ongoing scientific challenge. The reasons for this complexity include:

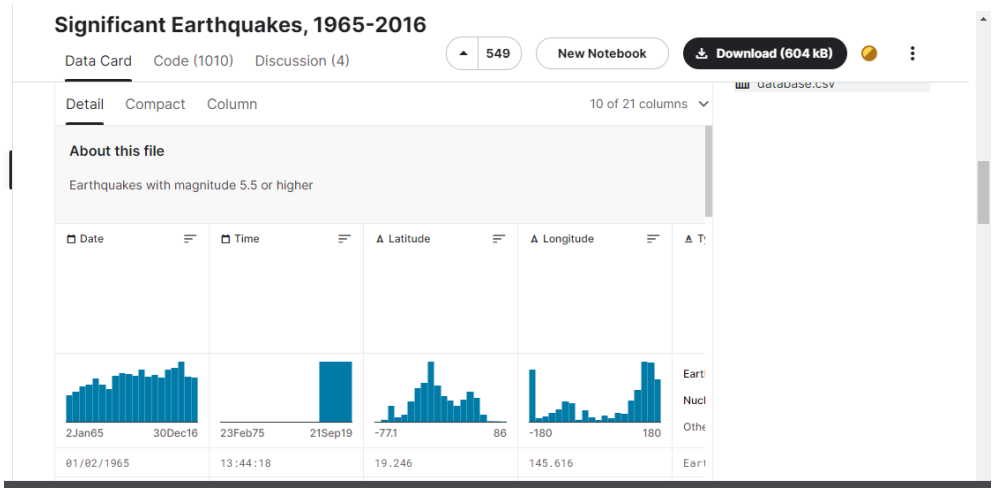
- Earthquakes are the result of complex interactions between tectonic plates, geological structures, and a range of other physical factors. Predicting the specific conditions that lead to a seismic event is a multidisciplinary endeavor.

- The temporal and spatial patterns of earthquakes are highly variable. Predicting when and where an earthquake will occur with precision is complicated due to these variations.

- The historical data on earthquakes is limited, and there is often insufficient information to draw definitive conclusions about future events.

- Ongoing research and advancements in technology are essential to improving prediction capabilities. Scientists continually develop and refine models and methodologies to enhance earthquake prediction.

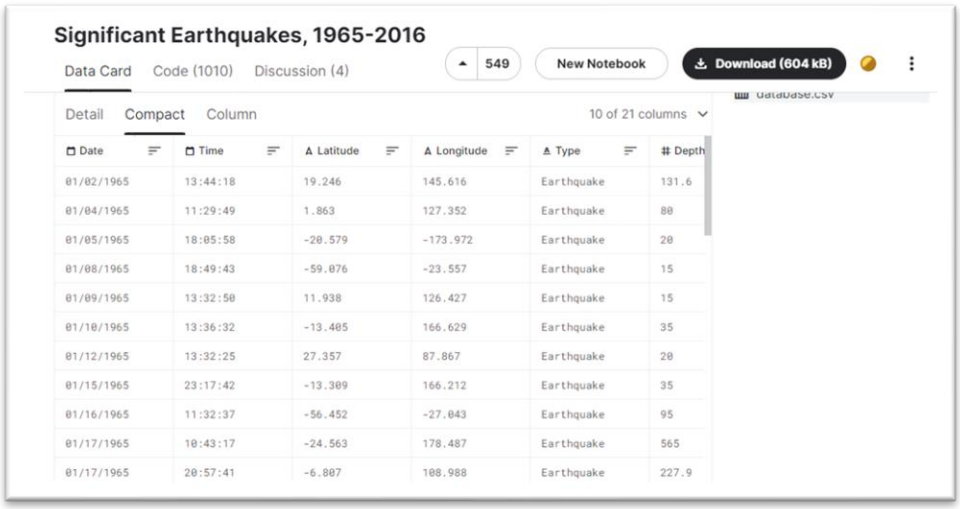
DATA SET



Building an earthquake prediction model requires access to a dataset that includes historical earthquake data and related features

Acquire a Dataset

- You can obtain earthquake data from various sources, including government agencies, research organizations, or online repositories. One such source is the USGS Earthquake Hazards Program.



Load the Dataset

- Use Python and data manipulation libraries like pandas to load the dataset into a DataFrame. Here's an example code snippet:

```
import pandas as pd

# Load the dataset (replace 'your_data.csv' with your dataset file)
df = pd.read_csv('your_data.csv')
```

Data Preprocessing

- Data preprocessing is crucial to prepare the dataset for machine learning. Common preprocessing steps include:

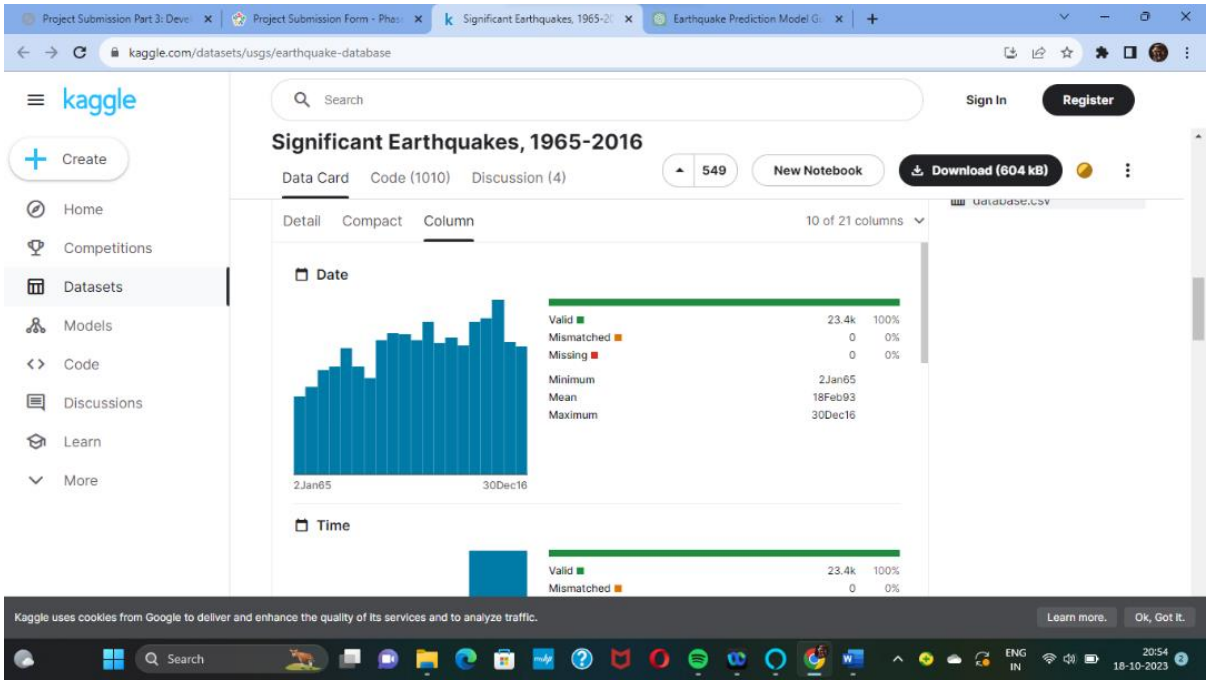
- Handling missing values: Decide how to handle missing data (e.g., impute, remove, or use advanced techniques).

- Feature selection: Choose relevant features for your prediction model.

- Feature engineering: Create new features or transform existing ones to better represent the data.

- Data normalization or standardization: Scale the features if needed.

- Handling categorical data: Encode categorical variables into numerical values (e.g., one-hot encoding)



Exploratory Data Analysis (EDA)

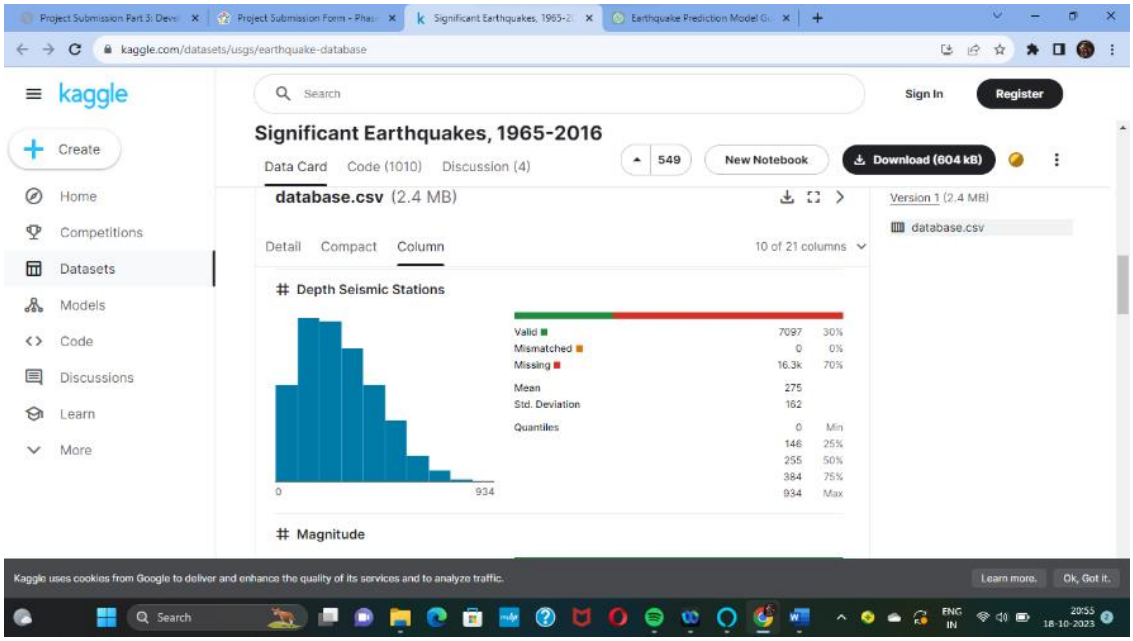
- Analyze the dataset to gain insights into the data. You can create visualizations and summary statistics to better understand the distribution of data and relationships between variables.

Split Data into Training and Testing Sets

- To evaluate the performance of your model, split the dataset into training and testing sets. The typical split is 70-80% for training and 20-30% for testing.

```python
from sklearn.model_selection import train_test_split

X = df.drop(columns=['target_column'])  # Features
y = df['target_column']  # Target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```




```

## Feature Scaling

- Depending on the choice of algorithms (e.g., neural networks, support vector machines), you may need to scale or normalize your data.

## Build and Train Your Model

- Choose a machine learning or deep learning model suitable for your problem (e.g., Random Forest, SVM, Neural Network).

- Train the model on the training data.
```

Visualization

Here, all the earthquakes from the database in visualized on to the world map which shows clear representation of the locations where frequency of the earthquake will be more.

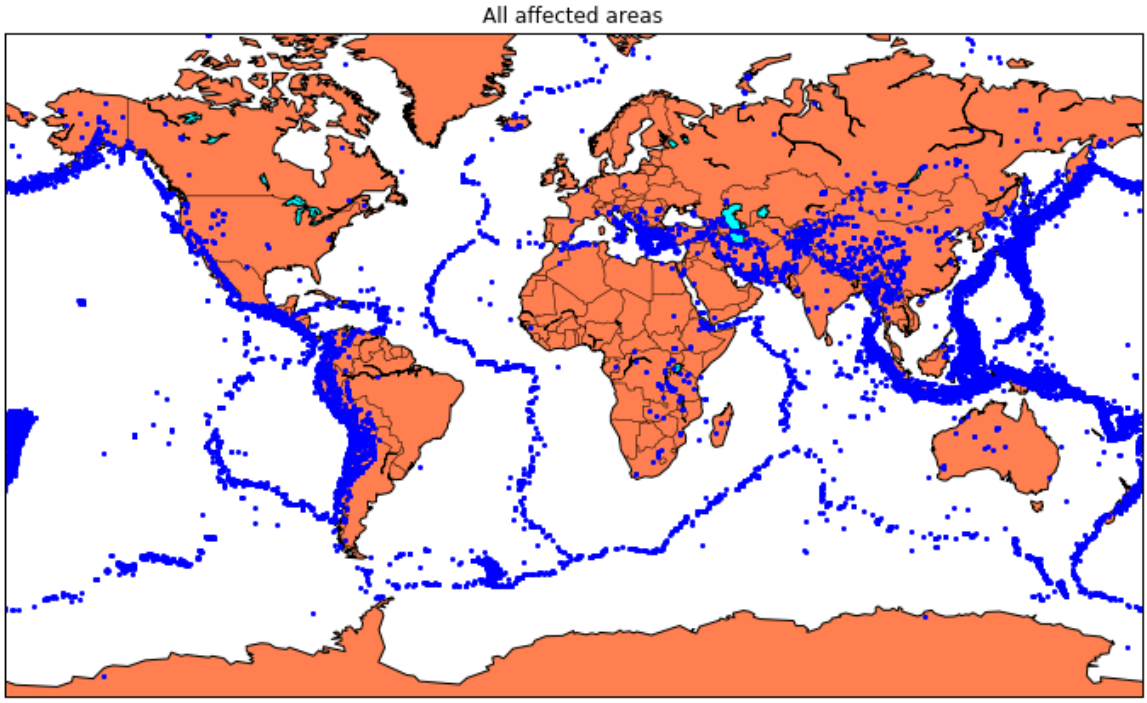
In [8]:

```
from mpl_toolkits.basemap import Basemap

m = Basemap(projection='mill',llcrnrlat=-80,urcrnrlat=80, llcrnrlon=-180,ur
crnrlon=180,lat_ts=20,resolution='c')

longitudes = data["Longitude"].tolist()
latitudes = data["Latitude"].tolist()
#m = Basemap(width=12000000,height=9000000,projection='lcc',
            #resolution=None,lat_1=80.,lat_2=55,lat_0=80,lon_0=-107.)
x,y = m(longitudes,latitudes)
```

```
fig = plt.figure(figsize=(12,10))
plt.title("All affected areas")
m.plot(x, y, "o", markersize = 2, color = 'blue')
m.drawcoastlines()
m.fillcontinents(color='coral',lake_color='aqua')
m.drawmapboundary()
m.drawcountries()
plt.show()
```



Splitting the Data

Firstly, split the data into Xs and ys which are input to the model and output of the model respectively. Here, inputs are TImestamp, Latitude and Longitude and outputs are Magnitude and Depth. Split the Xs and ys into train and test with validation. Training dataset contains 80% and Test dataset contains 20%.

linkcode

```
X = final_data[['Timestamp', 'Latitude', 'Longitude']]
y = final_data[['Magnitude', 'Depth']]
from sklearn.cross_validation import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
, random_state=42)
print(X_train.shape, X_test.shape, y_train.shape, X_test.shape)
```

Neural Network model

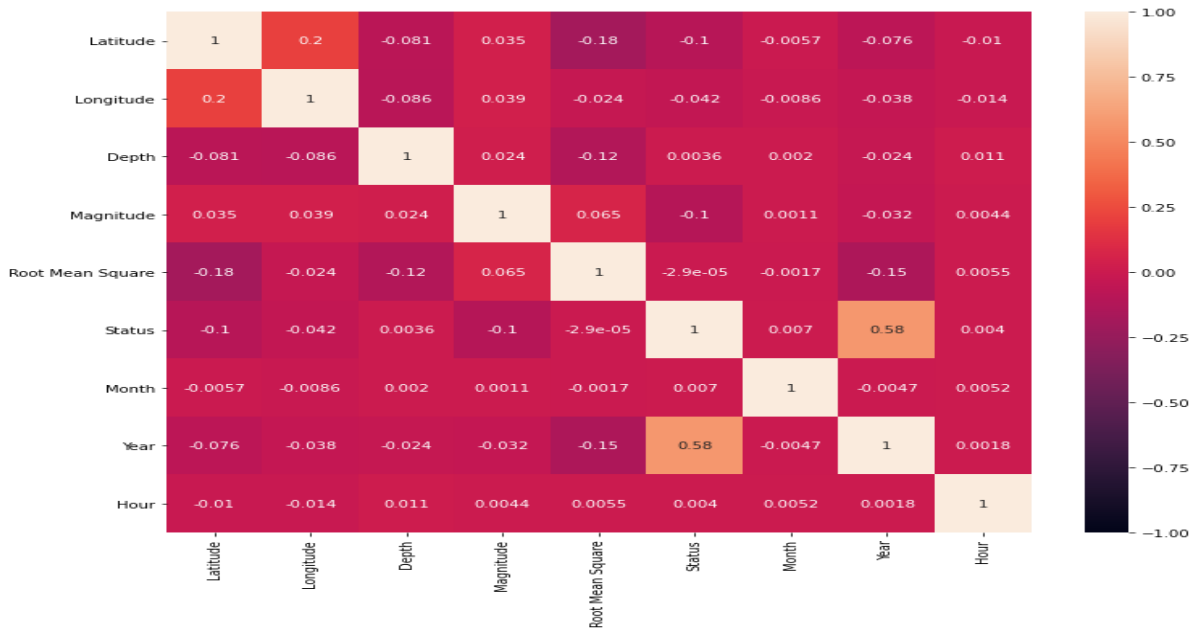
In the above case it was more kind of linear regressor where the predicted values are not as expected. So, Now, we build the neural network to fit the data for training set. Neural Network consists of three Dense layer with each 16, 16, 2 nodes and relu, relu and softmax as activation function.

```
from keras.models import Sequential
from keras.layers import Dense

def create_model(neurons, activation, optimizer, loss):
    model = Sequential()
    model.add(Dense(neurons, activation=activation, input_shape=(3,)))
    model.add(Dense(neurons, activation=activation))
    model.add(Dense(2, activation='softmax'))

    model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])


    return model
```



Encoding

```
data['Type'].unique()
```

Out[30]:

```
array(['Earthquake', 'Nuclear Explosion', 'Explosion', 'Rock Burst'],
      dtype=object)
```

In [31]:

```
def onehot_encode(df, columns, prefixes):
    df = df.copy()
    for column, prefix in zip(columns, prefixes):
        dummies = pd.get_dummies(df[column], prefix=prefix)
        df = pd.concat([df, dummies], axis=1)
        df = df.drop(column, axis=1)
    return df
```

Splitting and Scaling

```
y = data.loc[:, 'Status']
X = data.drop('Status', axis=1)
scaler = StandardScaler()

X = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7,
random_state=56)
```

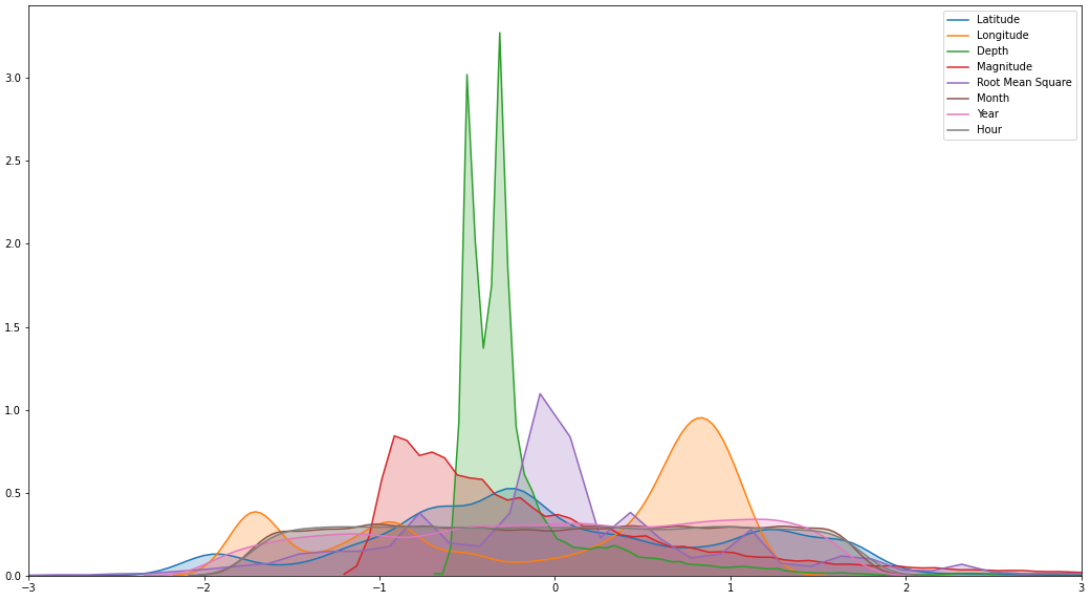
Modeling and Training

```
inputs = tf.keras.Input(shape=(104,))
x = tf.keras.layers.Dense(64, activation='relu')(inputs)
x = tf.keras.layers.Dense(64, activation='relu')(x)
outputs = tf.keras.layers.Dense(1, activation='sigmoid')(x)

model = tf.keras.Model(inputs, outputs)
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=[tf.keras.metrics.AUC(name='auc')]
)


batch_size = 32
epochs = 30

history = model.fit(
    X_train,
    y_train,
 validation_split=0.2,
    batch_size=batch_size,
    epochs=epochs,
    callbacks=[tf.keras.callbacks.ReduceLROnPlateau()],
    verbose=0
```



Results

In [40]:

linkcode

```
plt.figure(figsize=(18, 6))

epochs_range = range(epochs)
train_loss, val_loss = history.history['loss'], history.history['val_loss']
train_auc, val_auc = history.history['auc'], history.history['val_auc']

plt.subplot(1, 2, 1)
plt.plot(epochs_range, train_loss, label="Training Loss")
plt.plot(epochs_range, val_loss, label="Validation Loss")
plt.legend()
plt.title("Loss Over Time")
plt.subplot(1, 2, 2)
plt.plot(epochs_range, train_auc, label="Training AUC")
plt.plot(epochs_range, val_auc, label="Validation AUC")
plt.legend()
plt.title("AUC Over Time")

plt.show()
```