# Earthquake Prediction Model using Python AI_Phase5 Project Documentation & Submission

# TABLE OF CONTENT:

## Introduction:

Title: "Earthquake Prediction Model using Python"

In this project, we will develop an earthquake prediction model using Python. Earthquakes can have devastating consequences, and predicting them is a complex and crucial task. This project aims to leverage data analysis and machine learning techniques to build a model that can forecast earthquake occurrences based on historical seismic data and relevant environmental factors.

The project will involve various stages, including data collection, preprocessing, feature engineering, model selection, training, and evaluation. We will explore different machine learning algorithms and fine-tune the model to achieve the best possible predictions.

Our ultimate goal is to create a reliable and accurate earthquake prediction model that can contribute to seismic research and disaster preparedness efforts. By using Python, we can take advantage of its rich ecosystem of libraries for data analysis and machine learning, making this project both powerful and accessible.

Throughout this project, we will gain valuable insights into the field of earthquake prediction and further our understanding of the factors that influence seismic activity. Whether you are a data science enthusiast, a geoscientist, or simply curious about earthquake prediction, this project provides an opportunity to explore a fascinating and important area of research.

## Neural Network model:

we need to 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.

## Input:

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

def create_model(neurons, activation, opt
imizer, loss):
    model = Sequential()
    model.add(Dense(neurons, activation=a
ctivation, input_shape=(3,)))
    model.add(Dense(neurons, activation=a
ctivation))
    model.add(Dense(2, activation='softma
x'))
```

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

    return model
```

Using TensorFlow backend.

In this, we define the hyperparameters with two or more options to find the best fit.

## Input:

```
from keras.wrappers.scikit_learn impo
rt KerasClassifier

model = KerasClassifier(build_fn=crea
te_model, verbose=0)

# neurons = [16, 64, 128, 256]
neurons = [16]
# batch_size = [10, 20, 50, 100]
batch_size = [10]
epochs = [10]
# activation = ['relu', 'tanh', 'sigmo
id', 'hard_sigmoid', 'linear', 'expone
ntial']
activation = ['sigmoid', 'relu']
```

```
# optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nadam']
optimizer = ['SGD', 'Adadelta']
loss = ['squared_hinge']

param_grid = dict(neurons=neurons, batch_size=batch_size, epochs=epochs, activation=activation, optimizer=optimizer, loss=loss)
```

Here, we find the best fit of the above model and get the mean test score and standard deviation of the best fit model.

## Input:

```
grid = GridSearchCV(estimator=model,
param_grid=param_grid, n_jobs=-1)
grid_result = grid.fit(X_train, y_train)

print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
means = grid_result.cv_results_['mean
_test_score']
stds = grid_result.cv_results_['std_t
est_score']
params = grid_result.cv_results_['par
ams']
for mean, stdev, param in zip(means,
stds, params):
    print("%f (%f) with: %r" % (mean,
stdev, param))
```

```
Best: 0.957655 using {'activation': 'relu', 'batch_si
ze': 10, 'epochs': 10, 'loss': 'squared_hinge', 'neur
ons': 16, 'optimizer': 'SGD'}
0.333316 (0.471398) with: {'activation': 'sigmoid', '
batch_size': 10, 'epochs': 10, 'loss': 'squared_hinge
', 'neurons': 16, 'optimizer': 'SGD'}
0.000000 (0.000000) with: {'activation': 'sigmoid', '
batch_size': 10, 'epochs': 10, 'loss': 'squared_hinge
', 'neurons': 16, 'optimizer': 'Adadelta'}
0.957655 (0.029957) with: {'activation': 'relu', 'bat
ch_size': 10, 'epochs': 10, 'loss': 'squared_hinge',
'neurons': 16, 'optimizer': 'SGD'}
0.645111 (0.456960) with: {'activation': 'relu', 'bat
ch_size': 10, 'epochs': 10, 'loss': 'squared_hinge',
'neurons': 16, 'optimizer': 'Adadelta'}
```

The best fit parameters are used for same model to compute the score with training data and testing data.

## Input:

```
model = Sequential()
model.add(Dense(16, activation='relu', input_shape=(3,)))
model.add(Dense(16, activation='relu'))
model.add(Dense(2, activation='softmax'))

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

## Input:

```
model.fit(X_train, y_train, batch_size=10, epochs=20, verbose=1, validation_data=(X_test, y_test))
```

```
Train on 18727 samples, validate on 4682 samples
Epoch 1/20
18727/18727 [==============================] - 6s 330us/step -
loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.924
2
Epoch 2/20
18727/18727 [==============================] - 6s 320us/step -
loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.924
2
Epoch 3/20
18727/18727 [==============================] - 6s 320us/step -
loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.924
2
Epoch 4/20
18727/18727 [==============================] - 6s 322us/step -
loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.924
2
Epoch 5/20
18727/18727 [==============================] - 6s 321us/step -
loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.924
2
Epoch 6/20
18727/18727 [==============================] - 6s 323us/step -
loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.924
2
Epoch 7/20
18727/18727 [==============================] - 6s 322us/step -
loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.924
2
Epoch 8/20
18727/18727 [==============================] - 6s 321us/step -
loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.924
2
Epoch 9/20
18727/18727 [==============================] - 6s 322us/step -
loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.924
2
Epoch 10/20
18727/18727 [==============================] - 6s 322us/step -
loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.924
2
Epoch 11/20
```

```
18727/18727 [==============================] - 6s 322us/step -
loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.924
2
Epoch 12/20
18727/18727 [==============================] - 6s 322us/step -
loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.924
2
Epoch 13/20
18727/18727 [==============================] - 6s 321us/step -
loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.924
2
Epoch 14/20
18727/18727 [==============================] - 6s 322us/step -
loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.924
2
Epoch 15/20
18727/18727 [==============================] - 6s 322us/step -
loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.924
2
Epoch 16/20
18727/18727 [==============================] - 6s 323us/step -
loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.924
2
Epoch 17/20
18727/18727 [==============================] - 6s 322us/step -
loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.924
2
Epoch 18/20
18727/18727 [==============================] - 6s 321us/step -
loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.924
2
Epoch 19/20
18727/18727 [==============================] - 6s 321us/step -
loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.924
2
Epoch 20/20
18727/18727 [==============================] - 6s 322us/step -
loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.924
2
```

## Output:

```
<keras.callbacks.History at 0x7ff0a8d
b8cc0>
```

## Input:

```
[test_loss, test_acc] = model.evaluat
e(X_test, y_test)
print("Evaluation result on Test Data
: Loss = {}, accuracy = {}".format(te
st_loss, test_acc))
```

```
4682/4682 [==========================
====] - 0s 39us/step
Evaluation result on Test Data : Loss
= 0.503845579040605 6, accuracy = 0.92
41777017858995
```

We see that the above model performs better but it also has lot of noise (loss) which can be neglected for prediction and use it for furthur prediction.

The above model is saved for furthur prediction.

## Input:

```
model.save('earthquake.h5')
```

## Conclusion:

With the successful completion of our earthquake prediction model using Python, we have taken a significant stride in the realm of seismic forecasting. This endeavor has showcased the power of data-driven approaches and machine learning in addressing one of the world's most pressing challenges.

We have meticulously collected and processed seismic data, explored various machine learning algorithms, and fine-tuned our model to provide valuable insights into earthquake prediction. While the journey doesn't end here, this accomplishment serves as a testament to our dedication and the potential of technology to contribute to disaster risk reduction.

As we move forward, we will continue to refine and expand this model, and we are committed to sharing our findings with the scientific community. This project is a reminder that, by embracing innovation and collaboration, we can enhance our understanding of seismic activity and work towards a safer, more resilient future for all.