PHASE 5: PROJECT DOCUMENTATION & SUBMISSION

PHASE 5 INSTRUCTIONS:

- ***In this section you will document the complete project and prepare it for submission.***
- ***After completion go to the Project Submission Part 5 section and upload your file to your Github account and add the link of your GitHub in the space provided and click on submit.***

DEVELOPMENT:

1. Define Your Problem:

Start by clearly defining your project's objectives and the problem you want to solve. Understanding the problem is crucial for selecting the right dataset and modeling approach.

2. Data Collection:

Gather the data relevant to your problem. This might involve web scraping, data acquisition from databases, or using publicly available datasets.

3. Data Preprocessing:

Clean the data by handling missing values, removing duplicates, and dealing with outliers. Preprocessing might also include data transformation, such as normalization or scaling.

4. Exploratory Data Analysis (EDA):

Perform EDA to gain insights into the dataset. This involves visualizing the data, identifying patterns, and understanding the relationships between different variables.

5. Feature Engineering:

Create or modify features to improve the performance of your model. Feature engineering can involve techniques like one-hot encoding, feature scaling, and creating new features from existing ones.

6. Data Splitting:

Split your dataset into training, validation, and testing sets. The training set is used to train the model, the validation set for hyperparameter tuning, and the testing set for evaluating the final model's performance.

7. Model Selection:

Choose a machine learning or deep learning model that's suitable for your problem. The selection depends on your data and objectives. Common choices include linear regression, decision trees, random forests, neural networks, etc.

8. Model Training:

Train your selected model on the training dataset. This involves optimizing the model's parameters to fit the data. The process may include cross-validation for hyperparameter tuning.

9. Model Evaluation:

Evaluate the model's performance using appropriate metrics (e.g., accuracy, F1-score, RMSE). For classification problems, you can use a confusion matrix. For regression problems, you can plot the predicted vs. actual values.

10. Model Validation:

Validate your model's performance on the validation dataset. Make necessary adjustments to improve its performance.

11. Hyperparameter Tuning:

Fine-tune the model's hyperparameters to optimize its performance. This may involve grid search, random search, or Bayesian optimization.

12. Testing and Final Evaluation:

Finally, evaluate your model on the testing dataset, which it has never seen before. This will provide an unbiased assessment of its performance.

13. Deployment:

If your model performs well, consider deploying it to make predictions in a real-world application.

14. Documentation:

Document the entire process, including data sources, preprocessing steps, model selection, hyperparameters, and evaluation results.

Remember that this is a high-level overview, and the specific steps and tools you use will depend on the project, data, and your objectives. Additionally, it's important to iterate and refine your approach as you learn more about the problem and the data during the project.

PROGRAM PART:

1.Create a ChatBot using Python:To create a chatbot, you can use the following Python code. This code will prompt the user for a question, process the question using the 'Chat' function, and display the answer.

```
class Chat:
    def __init__(self):
        self.name = None
        self.greetings = ['Hello', 'Hi there', 'Hey', 'Heya']
        self.farewells = ['Goodbye', 'Good bye', 'Bye', 'Catch you later']

    def respond(self, user_input):
        user_input = user_input.lower()

        if 'hi' in user_input or 'hello' in user_input:
```

```
            return random.choice(self.greetings)
        elif 'bye' in user_input or 'goodbye' in user_input:
            return random.choice(self.farewells)
        else:
            return 'Sorry, I did not understand your question. Can you please provide more information?'


def chat_bot():
    chat = Chat()

    while True:
        user_input = input('You: ')

        if 'name' in user_input:
            chat.name = user_input.split(" ", 1)[1]
            response = f'Hello, {chat.name}'
        else:
            response = chat.respond(user_input)
```

```
        print('ChatBot: ' + response)



if __name__ == '__main__':
    chat_bot()
```

This chatbot will greet the user if they say "hi" or "hello", and farewell the user if they say "bye" or "goodbye". If the user asks anything else, the chatbot will respond with a generic message.

2. Training a ChatBot using Machine Learning:To create a more intelligent chatbot, you can train it using machine learning techniques. You can use the following Python code as a starting point. This code uses a simple dataset to train a machine learning model to predict the answer based on the question:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import LabelEncoder
import pandas as pd
import numpy as np
```

```
# Load the dataset
url = 'https://raw.githubusercontent.com/songkaisteele/Quora-Question-Pair-Dataset/master/Quora%20Question%20Pairs.csv'
data = pd.read_csv(url)

# Split the dataset into training and testing sets
np.random.seed(0)
split = np.random.rand(len(data)) < 0.9
train = data[split]
test = data[~split]

# Train the machine learning model
text_clf = make_pipeline(TfidfVectorizer(), LabelEncoder(), MultinomialNB())
text_clf.fit(train['question1'], train['is_duplicate'])

# Define a function to get a prediction from the chatbot
def predict_answer(question):
    prediction = text_clf.predict([question])
    return 'Sure, here is an answer.' if prediction[0] == 1 else 'Sorry, I do not have an answer.'
```

```
# Create a function to interact with the chatbot
def chat_bot():
    while True:
        user_input = input('You: ')

        if 'bye' in user_input.lower():
            print('ChatBot: Goodbye!')
            break
        else:
            print('ChatBot: ' + predict_answer(user_input))

if __name__ == '__main__':
    chat_bot()
```

Please note that this code is just a basic example and the chatbot's accuracy will be quite low. To improve the chatbot's performance, you would need to use a more advanced machine learning model and a larger, higher-quality dataset.

3.Creating a ChatBot with Deep Learning:You can create a more advanced chatbot using deep learning techniques. Here's an example using a Sequence-to-Sequence model with LSTM units:

```
import numpy as np

from keras.models import Sequential

from keras.layers import Dense, LSTM, Embedding

from keras.preprocessing.models import Sequential

textfrom import Tokenizer

from keras keras.layers import Dense, L.preprocessing.STM,sequence import pad Embedding_sequences

from sklearn.models import Sequential

from

from keras.preprocessing.text import Tokenizermodel_selection import train_test_split

import pandas keras.layers import Dense,

from keras.preprocessing. as pd

import nltk

from n LSTM, Embedding

fromsequence import pad_sequences

ltk.corpus import stop keras.preprocessing.from sklearn.model_selection importwords
```

```
import string


ntext import Tokenizer

from keras. train_test_split

ltk.download('punktimport pandas as pd

preprocessing.sequence import pad')

nltk.download('import_sequstopwords')


# Load theences

from sklearn. nltk

from dataset

url = 'httpsmodel_ nltk.corpus import stopwords

import://raw.githubusercontent.com/songselection import
train_test_split

import pandas as pd

importkaisteele string


/Quora-Question-Pair- nltk

from nltk.corpus importnltk.download('Dataset/master stopwords

import string
```

```
punkt')

nltk.download('stop/Quora%20Question%20Pairs.csvnltk.download('pu
nkt')

words')

'

data = pd.read_csv(nltk.download('stopwords')url)


# Define a

# Load the dataset


# Load the function to preprocess the data


url = 'https://raw. dataset

url = 'https://def preprocess_text(text):

   textgithubusercontent.com/raw.githubusercontent.com/ =
text.lowersongkaisteele/Qu()

   text = nltk.word_ora-Question-Pairsongkaisteele/Quora-Question-
Pair-Dattokenize(text)-Datasetaset/master/Quora%20Question%2
```

```
    text = [word for word in text if word not in
stop/master/Quora%20Question%0Pairs.words.words('english')20Pairs.
csv'

data = pdcsv'

data = pd.read_csv(url) and word not in string.punctuation.read_]

    text =


# Split the dataset into training and testingcsv(url)


# Clean the text data ' '.join(text)

    return text


data sets

np

stop_words['question1'] = data['question.random.seed(0)

split = np. = set(stopwords.words('english'))

1'].apply(preprocess_text)

datatranslatorrandom.rand(len(data['question2'] =
data['question2'].apply(preprocess_ = str.maketrans('', '',text))) < 0.9

train = data[split string.punctuation)

data['question1']
```

```
# Create training and testing datasets

X = data['question1'].values]

test = data[~ = data['

split]

y = data['question2'].valuesquestion1'].apply(lambda x: ' '.join([

# Preprocess the data

stop_words = set(stop


X_train, X_test, y_train,word for word in x.split() if word not in
stopwords.words('english'))

tokenizer = y_test = train_test_split(X, Token y, test_izer()

tokenizer.fit_on_textssize_words]))

data['question2'] = data['question2'].apply(lambda
x(train['question1'].values)

vocab=0.2, random_state=0)


# Define a function to encode the sequences

def encode_sequences(X_size = len(tokenizer.: ' '.join([word for word in
x.split() if word not in stop_words]))
```

```
# Tokenize the text, y, length=50):

   X_enc = tokenizer.textword_index) + 1

maxlen = 100 data

tokenizer = Tokenizer()

tokenizers_to_sequences(X)

   X_enc =

embedding_dim = 100.fit_on_texts(data pad_sequences(X_enc,
maxlen=length, padding='

trX = tokenizer.texts_to['question1'].tolist() +
data['question2']._sequences(train['question1'].values)

trXpost')

   y_enc = tokenizer.texts_to_sequences(tolist())

vocab_size = = pad_sequences(trX, maxlen=maxy)

   y_enc = pad_sequences(y_ len(tokenizer.word_index) + 1len)

trY = train['enc, maxlen=length


# Prepare the input and output sequences

, padding='post')

   return X_enc,is_duplicate'].values

teX = tokenX = tokenizer. y_enc
```

```
tokenizer = Tokenizer()

tokenizer.fitizer.texts_to_sequencestexts_to_sequences(data['question
1']._on_texts((test['tolistXquestion1'].values)

teX =_train)

())

X = pad_sequences(X, maxlen=100)

y pad_sequences(teX, maxlen=maxlen)

teY = test['is_duplicate']. =
tokenizer.texts_to_sequences(data['question2'].tolist())values


# Create the Sequence

y = pad_sequences(y, maxlen=10-to-Sequence model

model = Sequential()

model.add(Embedding(0)


# Split the dataset into training and testing setsvocab_size,
embedding_dim, input_length=

X_train, X_test, y_train, y_maxlen))

model.add(LSTM(50test = train_test_split(X, y, test_size=0))

model.add(Dense(1, activation='sig.2, random_state=0)moid'))
```

```
model.compile(loss='binary_crossentropy', optimizer


# Create the Sequence-to-Sequence model

model ==='adam Sequential()

model.', metrics=['accuracy'])


# Create a layer that learns to map the input sequence to a probability of the next character

decoder_inputs = Input(shape=(maxlen,))

decoder_embedding = Embedding(vocab_size, Define a function to evaluate the accuracy of the model

def evaluate_accuracy Create training and testing sets

X = pad_sequences(X, maxlen=100)

y = pad_sequences((X, y):

    _, accuracy = model.evaluate(X, y) embedding_dim)(decoder_inputs)

decoder_lstm = LSTM(50, return_

    return accuracy


# Train the Sequence-toy, maxlen=maxlen)
```

```
X_train, X_test, y_train, y_test = train_test_split(X-Sequence model

model.fit(X_trainsequences=True, dropout=0.5)(decoder_embedding)

decoder_outputs = D, y_train, epochs=, y, test_size=0.2,
random_state=0)


# Train the model

history = model.fit(X_train,50, batch_size=32, validation_data=(X_test,
yense(vocab_size, activation='softmax')(decoder_lstm)


# Create the final Sequence-to-Sequence_test))


# Evaluate the accuracy of the y_train, validation_data=(X_test, y_test),
epochs=10, batch_size=64 Sequence-to-Sequence model

ac model

model = Model([encoder_inputs, decoder_inputs], decodercuracy)


# Test the model

loss, accuracy = model.evaluate(X_test, y_test)

print(' = evaluate_accuracy(X_test, y_test)

print_outputs)

model.compile(optimizer='rmsprop', loss='categoricalTest loss:', loss)
```

```
print('Test accuracy:', accuracy)


# Make predictions
predictions = model.predict(('Test set accuracy: {:.4f}'.format(accuracy))


tokenizer_crossentropy', metrics=['accuracy'])


# Training the Sequence-to-Sequence modelX_test)


# Create a submission file
submission = pd..index_word = dict(enumerate(tokenizer.word_index)))
print(len(tokenizer.word_index)))
printDataFrame({'Id': data['Id'], 'is_duplicate': predictions})
submission.to_csv('submission.csv', index=False)
\end{code}
```

Answer: Here's an example of how you can solve this problem:

```
\begin{
```

```
model.fit([X_train, y_train], y_train, batch_size=128, epochs=100,
validation_data=([X_test, y_test], y_test))


# Training the Sequence-to-Sequence model

code}

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise(maxylen)

print(len(set(train['question1'].apply(lambda x: ' '.join([word for word in
x.split() if word not in stop_words]))).values)))

print(len(set(train['question2'].apply(lambda x:model.fit([X_train,
y_train], y_train, batch_size=128, epochs=100, validation_data=([X_test
import cosine_similarity

from sklearn.model_selection import train_test_split


# Initialize the TfidfVectorizer

vectorizer = TfidfVectorizer(stop_words='english ' '.join([word for word
in x.split() if word not in stop_words]))).values)))


train_reverse = train[['question2', 'question1',, y_test], y_test))


# Define a function to predict the next character in a sequence
```

```
def predict', ngram_range=(1, 2), max_features=5000)


# Fit the vectorizer to the dataset

vector 'is_duplicate']]


test_reverse = test[['question2', 'question1', 'is_duplicate']]

izer.fit(data['question1'].tolist() + data['question2'].tolist())


#_next_char(text, char):

   predicted = model.predict([text, char])

   return predicted. Transform the dataset into a tf-idf representation

X =

data = data.append(train_reverse)


X_train vectorizer.transform(data['question1'].tolist() +
dataargmax(axis=-1)


# Testing the model

print('Actual: ', text)

print('Predicted: ', predict_next['question2'].tolist())
```

```
# Create the train and test datasets

, X_test, y_train, y_maxlen))


maxylen = max([len(set(word for word in x.split() if word not in
stop_words)) for x in train['question1']])


X =X_train, X_test, y_train, y_test = train_test_split(X,
data['is_dup_char(text))

print('Predicted Next Character: ',
chr(predict_next_char(text).flatten()[0] + 32))


print('Accuracy Score pad_sequences(X, maxlen=100)


y pad_sequences(teX, maxlen=maxlicate'].tolist(), test_size=0.2,
random_state=0)


# Calculate the cosine similarity between each pair of sentences in the
dataset

similarity = cos: ', model.evaluate([X_test, y_test], y_test))

print('Test Loss: ', modellen)
```

teY = test['is_duplicate']. =
tokenizer.texts_to_sequences(data['question2']..test_on_batch([Xtolist
())values

X_train, X__test, y_test], y_test))

print('Test Accuracy: ', model.test_on_batch([X_test, ytest, y_train,
y_maxlen))

model.fit(X_train_test], y_test))

\end{code}, y_train, epochs=50, batch_size=32,
validation_data=(X_test, y_test))

accuracy = evaluate(model, X_test, y_test, 50, 32, 'Test')

print('Test Accuracy: ', accuracy)

Comment: @abcdj Thanks for your reply. This approach is more comprehensive. I need to check it more thoroughly to see if it solves the problem. However, the initial question is asking about a simpler solution, so I will mark this answer as helpful, but not as the solution. If I don't find any simpler solution, I will consider your approach. Thanks again.

Comment: No problem, glad to help! I understand you are looking for a simpler solution, and my solution might not be ideal in that case. I will try to find a more concise approach. Good luck!

Comment: Thanks. If you have any ideas or solutions, please let me know.

Comment: Sure, I'll be happy to help! One idea that comes to mind is to preprocess the data and extract relevant features. For example, you can calculate the similarity between each pair of questions using a different similarity measure (e.g., cosine similarity, Jaccard similarity). Another idea is to train a simple classification model using features like question length, number of unique words, etc. These are just general ideas, and I hope they help you find a simpler solution!!</s><s>Tags: ios, objective-c, ios7, core-bluetooth

Question: CBCentralManager's delegate method "didDiscoverPeripheral" not<s>Tags: git, jenkins, gerrit

Question: How to push code from Jenkins to Gerrit without checking out

I am trying to push some changes to a Gerrit repository from Jenkins without checking out the repository first. The use getting called in iOS7

I have been trying to write a Core Bluetooth program for the iPhone (iOS 7). The `CBCentralManager`'s delegate method "didDiscoverPeripheral" is case is that the Jenkins job has modified a file and needs to push these changes back to the Gerrit repository.

Here is a snippet not getting called. I have the necessary Core Bluetooth framework and delegates. I am also running this on of my Jenkinsfile that tries to accomplish this:

\begin{code}

pipeline {

an iPhone 5 with iOS 7.1.1. Here is my code:

\begin{code}   agent any

    stages {

        stage('Update') {

            steps {

                echo 'Up

// AppDelegate.h

```
#import <UIKit/UIKit.h>

#importdating...'

            sh '''

                git config --global user.email "email@example.com
<CoreBluetooth/CoreBluetooth.h>


@interface AppDelegate : UIRes"

                git config --global user.name "User Name"

            ponder <UIApplicationDelegate, CBCentralManagerDelegate>


@property (    git remote add origin ssh://username@gerrit-
repo.com:29strong, nonatomic) UIWindow *window;

@property (strong, non418/project-name.git

                git add modified-file.txt

                git commit -m "atomic) CBCentralManager
*centralManager;


@end


// AppDelegate.m

#importUpdated file"
```

```
            git push origin HEAD:refs/for/branch-name
        '''
    "AppDelegate.h"

@implementation AppDelegate

- (BOOL)application:(UIApplication *) }
    }
  }
}
\end{code}

When I run thisapplication didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
  self.centralManager = [[CBCentral job, I get the following error:

\begin{code}
fatal: No configured push destination.
```

Either specify the URL from theManager alloc] initWithDelegate:self queue:nil];

return YES;

}

command-line or configure a remote repository using

git remote add <name> <url>-(void)centralManagerDidUpdateState:(CBCentralManager *)central

{

if (central.state == CBCentralManagerStatePoweredOn) {

and then push using the remote name

git push <name>

\end{code}

However, as you can see in my script NSLog(@"Powered On");

[self.centralManager scanForPeripheralsWithServices:nil options:nil];

} else {

, I have already added the remote repository and attempted to push using the remote name. So I'm not sure NSLog(@"Powered Off");

}

}

- (void)centralManager:(CBCentralManager *)central why this error is occurring.

If I need to provide any more information or didDiscoverPeripheral:(CBPeripheral *)peripheral advertisementData context, please let me know. Any help is greatly appreciated!

Comment: Can you show the full console output of the Jenkins build that runs this Jenkinsfile:(NSDictionary *)advertisementData RSSI:(NSNumber *)RSSI

{

NSLog? The problem may lie in some previous steps that were executed.

Answer: It looks like your issue might(@"Discovered: %@", peripheral.name);

}

@end

\end{code}

In be due to the use of the 'sh' command within your pipeline.

In the ' this code, I get the "Powered On" log, which means that the central manager's state is powered onsh' command, all of the git commands are executed within their own shell. When this shell is closed (after the 'sh' command is completed), the git repository context is lost. This means that any git commands you execute after the 'sh' command.

This code creates a basic chatbot that uses TF-IDF (Term Frequency-Inverse Document Frequency) to calculate similarity between user input and predefined responses. You can expand upon this example by using more extensive datasets, more advanced NLP techniques, and incorporating a broader range of responses for a more sophisticated chatbot.

Conclusion:

This is a simple example of how to create a chatbot using Python with pre-trained language models to enhance the quality of responses with introduction and conclusion. You can customize the chatbot to meet your specific needs by adding more features and functionality.