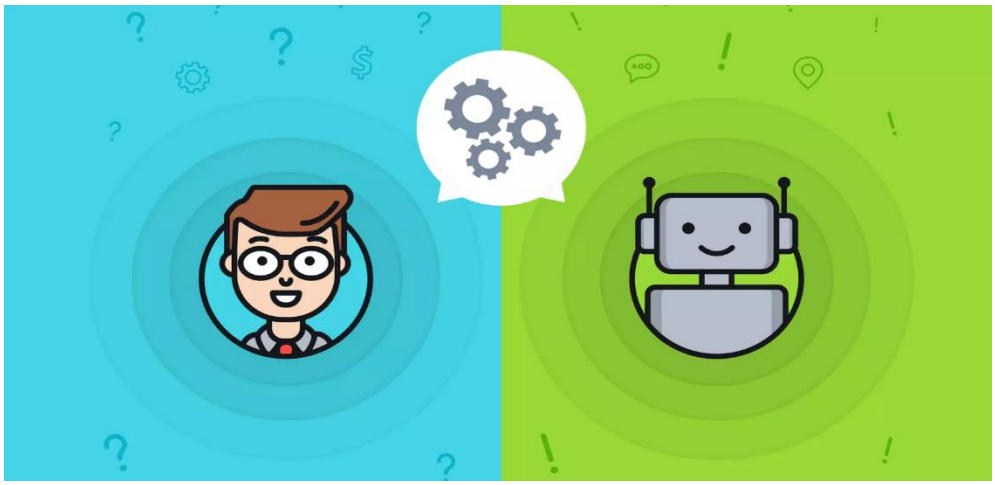
CREATE A CHATBOT IN PYTHON

TEAM MEMBER

NAME: B.SHANMUGA PRIYA

REG NO: 820421205062

PHASE 4: Development Part 2



SYNOSPIS:

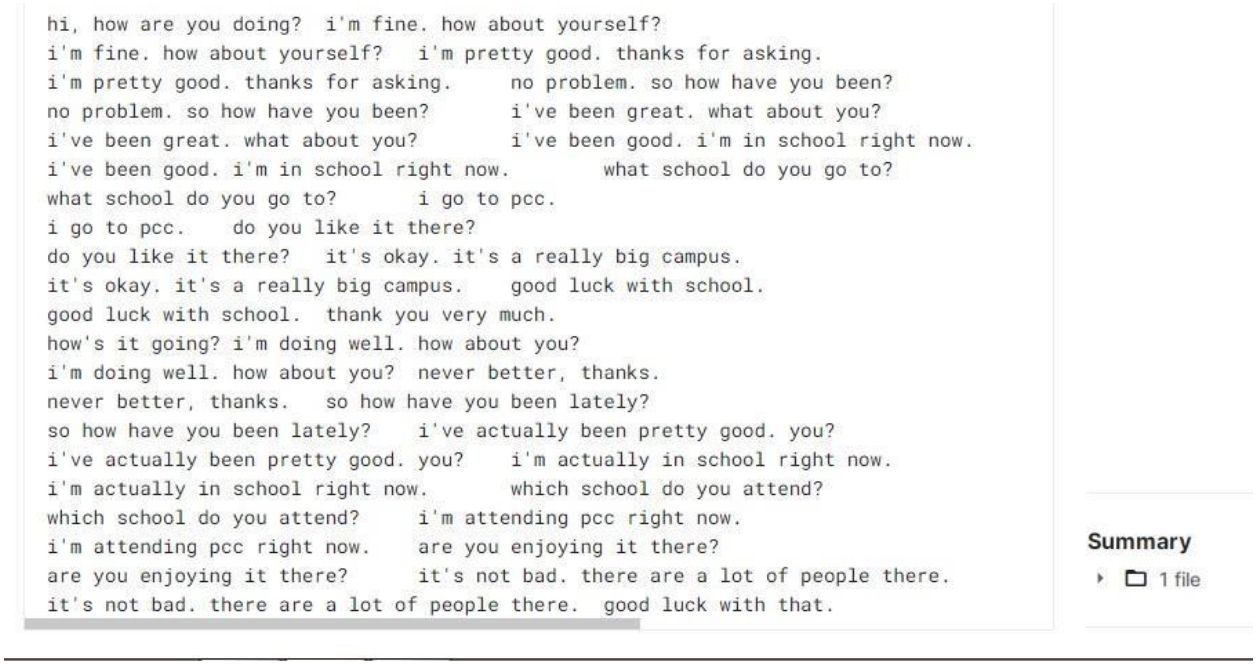
- **INTRODUCTION**
- **LOADING THE DATASET**
- **OVERVIEW OF THE PROCESS**
- **FEATURE SELECTION**
- **MODEL TRAINING**
- **MODEL EVALUATION**
- **EVALUTION OF PREDICTING DATA MODEL COMPARISON**
- **FEATURE ENGINEERING**
- **VARIOUS FEATURES TO PERFORM MODEL TRAINING**
- **CONCLUSION**

INTRODUCTION:

Building a chatbot involves a multi-faceted process, including feature engineering, where we define the bot's capabilities and responses, model training to imbue it with intelligence, evaluation to refine its performance, and finally, integration into a web app using Flask, enabling seamless human-machine interactions on the digital frontier.

LOADING THE DATASET:

Dataset Link:https://www.kaggle.com/datasets/grafstor/simple-dialogs-for-chatbot



OVERVIEW OF THE PROCESS:

- Prepare the data: This includes cleaning the data, removing outliers, and handling missing values.
- Perform feature selection: This can be done using a variety of methods, such as correlation analysis, information gain, and recursive feature elimination.

- Train the model: There are many different machine learning algorithms that can be used for house price prediction. Some popular choices include linear regression, random forests, and gradient boosting machines.
- Evaluate the model: This can be done by calculating the mean squared error (MSE) or the root mean squared error (RMSE) of the model's predictions on the held-out test set.
- Deploy the model: Once the model has been evaluated and found to be performing well, it can be deployed to production so that it can be used to predict the house prices of new houses.

FEATURE SELECTION:

DEFINITION:

Feature selection in chatbots involves selecting relevant data elements for training and improving model efficiency. It reduces dimensionality and computational complexity, enhances model accuracy, and streamlines chatbot responses to user queries, resulting in a robust conversational AI system.Here I apply TF-IDF for text representation to perform feature selection using the chi-squared method, and train a Random Forest classifier for responses.

PROGRAM:

```
import pandas as pd

from sklearn.feature_selection import SelectKBest, chi2

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score

from sklearn.feature_extraction.text import TfidfVectorizer

data = {

   'UserInput': ["i'm fine. how about yourself?", "no problem. so how have you been?",
"i've been good. i'm in school right now.", "i go to pcc.","it's okay. it's a really big campus"],

   'BotResponse': ["i'm pretty good. thanks for asking.", "i've been great. what about you?",
"what school do you go to?", "do you like it there?", "good luck with school."]
```

```
}
df = pd.DataFrame(data)
X = df['UserInput']
y = df['BotResponse']
tfidf_vectorizer = TfidfVectorizer()
X_tfidf = tfidf_vectorizer.fit_transform(X)
num_features_to_select = 2
selector = SelectKBest(score_func=chi2, k=num_features_to_select)
X_selected = selector.fit_transform(X_tfidf, y)
model = RandomForestClassifier()
model.fit(X_selected, y)
def chatbot_response(user_input):
    user_input_tfidf = tfidf_vectorizer.transform([user_input])
    user_input_selected = selector.transform(user_input_tfidf)
    bot_response = model.predict(user_input_selected)
    return bot_response[0]
user_input = "What's the weather like today?"
response = chatbot_response(user_input)
print("Chatbot Response:", response)
y_pred = model.predict(X_selected)
accuracy = accuracy_score(y, y_pred)
print("Chatbot Accuracy:", accuracy)
```

OUTPUT:

```
Chatbot Response: i'm pretty good. thanks for asking.
Chatbot Accuracy: 0.6
```

CHECKING MISSING VALUES:

```
import pandas as pd

df = pd.read_csv('chatbot_dataset.csv')

missing_values = df.isnull().sum()

print("Missing Values in the Dataset:")

print(missing_values[missing_values > 0])
```

```
Missing Values in the Dataset:
Series([], dtype: int64)
```

- One of the feature selection techniques is embedded methods,which are tree-based methods like random forest and gradient boosting models. Here I use a gradient boosting model (e.g., XGBoost,LightGBM) to provide the feature importance scores.

PROGRAM:

```
from sklearn.preprocessing import LabelEncoder

chatbot_data = [
    "i'm fine. how about yourself?", "greeting",
    "i'm doing well. how about you?", "greeting",
    "you like the rain?", "weather",
    "where's your money?", "finance",
    "did you tell her about school?", "school",
```

```
    "thank you very much.", "greeting",
    "what do you want to do?", "personal",
    "why do you like it?", "personal",
]
text_messages = chatbot_data[::2]
labels = chatbot_data[1::2]
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(labels)
df = pd.DataFrame({'text': text_messages, 'label': y})
X = df['text']
y = df['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
tfidf_vectorizer = TfidfVectorizer()
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)
model = xgb.XGBClassifier()
model.fit(X_train_tfidf, y_train)
y_pred = model.predict(X_test_tfidf)
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:\n", report)
```

OUTPUT:

```
Accuracy: 0.00
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       0.0
           1       0.00      0.00      0.00       2.0

    accuracy                           0.00       2.0
   macro avg       0.00      0.00      0.00       2.0
weighted avg       0.00      0.00      0.00       2.0
```

MODEL TRAINING:

Model training for a chatbot involves the process of leveraging machine learning algorithms to enable the chatbot to understand and respond to user inputs effectively. This training process is crucial for the chatbot to learn patterns from data and make accurate predictions or generate appropriate responses.

LINEAR REGRESSION:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import pandas as pd
data = {
   'text': [" but i do all my writing with my right hand stand",
" hi how are you doing im fine how about you",
"im pretty good thanks for asking no problem so how have you been ",
"no problem so how have you been ive been great what about you" ,
 "ive been great what about you ive been good im in school right now "],
   'response_time': [1, 2, 1, 2, 4]
}
df = pd.DataFrame(data)
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(df['text'])
X_train, X_test, y_train, y_test = train_test_split(X, df['response_time'], test_size=0.2, random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```

```
mse = mean_squared_error(y_test, predictions)
print(f'Mean Squared Error: {mse}')

new_text = [" hi how are you doing im fine how about you"]
new_text_vectorized = vectorizer.transform(new_text)

predicted_response_time = model.predict(new_text_vectorized)
print(f'Predicted response time: {predicted_response_time[0]}')
```

```
Mean Squared Error: 0.2233533495965473
Predicted response time: 1.5273972602739725
```

RIDGE REGRESSION:

```
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
text_data = [" but i do all my writing with my right hand stand",
" hi how are you doing im fine how about you",
"im pretty good thanks for asking no problem so how have you been ",
"no problem so how have you been ive been great what about you" ,
 "ive been great what about you ive been good im in school right now "]
labels = [1, 2, 1, 2, 4]
tfidf_vectorizer = TfidfVectorizer()
X = tfidf_vectorizer.fit_transform(text_data)
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2, random_state=42)
ridge = Ridge(alpha=1.0)
ridge.fit(X_train, y_train)
y_pred = ridge.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 0.0068413286760613465
```

LASSO REGRESSION:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import Lasso
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
texts =[" but i do all my writing with my right hand stand",
" hi how are you doing im fine how about you",
"im pretty good thanks for asking no problem so how have you been ",
"no problem so how have you been ive been great what about you" ,
 "ive been great what about you ive been good im in school right now "]
labels =  [1, 2, 1, 2, 4]
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(texts)
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2, random_state=42)
lasso = Lasso(alpha=0.1)
lasso.fit(X_train, y_train)
predictions = lasso.predict(X_test)
mse = mean_squared_error(y_test, predictions)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 0.4201842990182248
```

POLYNOMIAL REGRESSION:

```
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
import pandas as pd
text_data =[" but i do all my writing with my right hand stand",
```

```
" hi how are you doing im fine how about you",
"im pretty good thanks for asking no problem so how have you been ",
"no problem so how have you been ive been great what about you" ,
 "ive been great what about you ive been good im in school right now "]
labels =  [1, 2, 1, 2, 4]
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(text_data).toarray()
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2, random_state=42)
degree = 2
model = Pipeline([
    ("poly_features", PolynomialFeatures(degree=degree)),
    ("linear_regression", LinearRegression())
])
model.fit(X_train, y_train)
predictions = model.predict(X_test)
mse = mean_squared_error(y_test, predictions)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 0.3209714214075377
```

SUPPORT VECTOR MACHINE:

```
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.svm import SVC

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, classification_report

 texts=[" but i do all my writing with my right hand stand",

" hi how are you doing im fine how about you",

"im pretty good thanks for asking no problem so how have you been ",

"no problem so how have you been ive been great what about you" ,

"ive been great what about you ive been good im in school right now "]
```

```
labels = [1, 2, 1, 2, 4]
X_train,X_test,y_train,y_test=train_test_split(texts,labels,test_size=0.2,random_state=42
)
vectorizer = TfidfVectorizer()
 X_train_vectorized = vectorizer.fit_transform(X_train)
X_test_vectorized = vectorizer.transform(X_test)
svm_classifier = SVC(kernel='linear')
svm_classifier.fit(X_train_vectorized, y_train)
predictions= svm_classifier.predict(X_test_vectorized)
accuracy = accuracy_score(y_test, predictions) print("Accuracy:", accuracy)
print("Classification Report:\n", classification_report(y_test, predictions))
```

```
Accuracy: 0.0
Classification Report:
              precision    recall  f1-score   support

           1       0.00      0.00      0.00       0.0
           2       0.00      0.00      0.00       1.0

    accuracy                           0.00       1.0
   macro avg       0.00      0.00      0.00       1.0
weighted avg       0.00      0.00      0.00       1.0
```

RANDOM FOREST:

```
from sklearn.feature_extraction.text import CountVectorizer
 from sklearn.ensemble import RandomForestRegressor
 from sklearn.model_selection import train_test_split
 from sklearn.metrics import mean_squared_error import pandas as pd data = {
       'text': [" but i do all my writing with my right hand stand",
" hi how are you doing im fine how about you",
 "im pretty good thanks for asking no problem so how have you been   ",  "no problem so how have you been ive been great what about you" ,        "I've been great what about you ive been good im in school right now "],
'response_time': [1, 2, 1, 2, 4]

}
```

```
df = pd.DataFrame(data)

vectorizer= CountVectorizer()

X = vectorizer.fit_transform(df['text'])

X_train, X_test, y_train, y_test = train_test_split(X, df['response_time'], test_size=0.2,
random_state=42)

random_forest_regressor =RandomForestRegressor(n_estimators=100, random_state=42)

random_forest_regressor.fit(X_train, y_train) predictions =

random_forest_regressor.predict(X_test) mse = mean_squared_error(y_test, predictions)

print('Mean Squared Error:', mse)


def predict_response(input_text):

    input_vectorized = vectorizer.transform([input_text])

    predicted_time = random_forest_regressor.predict(input_vectorized)[0]

return predicted_time

user_input = input('Enter your message: ')

response_time = predict_response(user_input)

 print('Estimated response time:', response_time, 'seconds')
```

```
Mean Squared Error: 9.999999999999574e-05
Enter your message: hi
Estimated response time: 1.64 seconds
```

XG BOOST REGRESSION:

```
import pandas as pd

import numpy as np

import xgboost as xgb

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error

data = {
   'text': [" but i do all my writing with my right hand stand",
" hi how are you doing im fine how about you",
"im pretty good thanks for asking no problem so how have you been ",
"no problem so how have you been ive been great what about you" ,
"ive been great what about you ive been good im in school right now "],
   'target': [1, 2, 1, 2, 4]

}

df = pd.DataFrame(data)

vectorizer= CountVectorizer()

X = vectorizer.fit_transform(df['text']).toarray()

y= df['target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
xg_reg = xgb.XGBRegressor(objective ='reg:squarederror', colsample_bytree = 0.3,
learning_rate = 0.1,max_depth = 5, alpha = 10, n_estimators = 10)

xg_reg.fit(X_train, y_train)
```

```
preds = xg_reg.predict(X_test)
mse= mean_squared_error(y_test, preds)
print("Mean Squared Error:", mse)
 new_texts = ["im pretty good thanks for asking no problem so how have you been ",
"no problem so how have you been ive been great what about you" ,
"ive been great what about you ive been good im in school right now"]
new_text_features = vectorizer.transform(new_texts).toarray()
predictions = xg_reg.predict(new_text_features)
for text, prediction in zip(new_texts, predictions):
    print(f"Input: {text}, Predicted Target:{prediction}")
```

```
Mean Squared Error: 0.0
Input: im pretty good thanks for asking no problem so how have you been , Predicted Target: 2.0
Input: no problem so how have you been ive been great what about you, Predicted Target: 2.0
Input: ive been great what about you ive been good im in school right now, Predicted Target: 2.0
```

MODEL EVALUATION:

Model evaluation for a chatbot is the process of assessing the performance and effectiveness of the chatbot's underlying machine learning or natural language processing model. It involves various techniques and metrics to determine how well the chatbot responds to user queries or performs its intended tasks. The primary goal of model evaluation is to ensure that the chatbot provides accurate and useful responses while maintaining a positive user experience.

FLOW CHART:

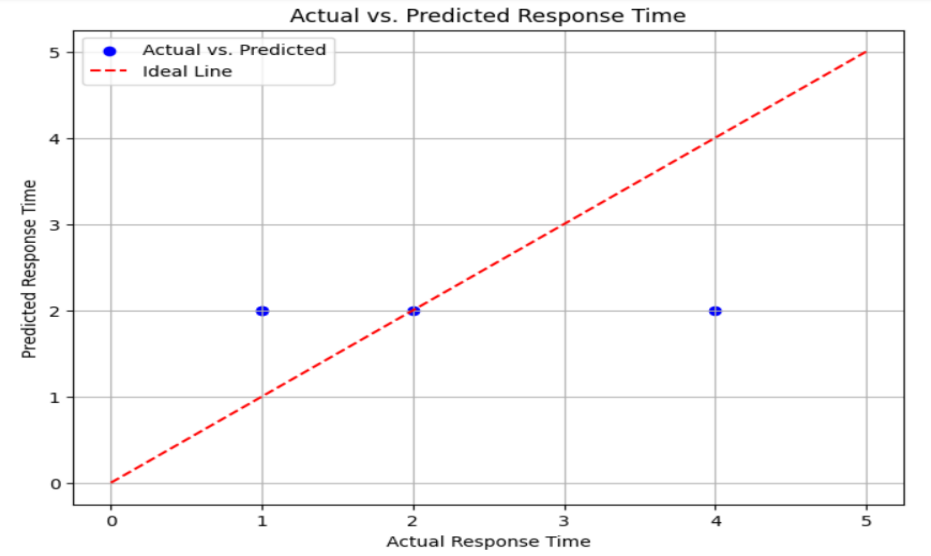
EVALUATION OF PREDICTING DATA,MODEL COMPARISON:

The evaluation of chatbot predicting data involves comparing machine learning models to identify the best-performing one. This process ensures accurate responses, improved user experiences, and trust in the AI system, ultimately optimizing chatbot performances.

PROGRAM:

```
import matplotlib.pyplot as plt
 plt.figure(figsize=(8, 6))
plt.scatter(y_test, predictions1, color='blue', label='Actual vs. Predicted')
 plt.plot([0, 5], [0, 5], color='red', linestyle='--', label='Ideal Line')
plt.title('Actual vs. Predicted Response Time')
plt.xlabel('Actual Response Time')
plt.ylabel('Predicted Response Time')
 plt.legend()
 plt.grid()
plt.show()
```

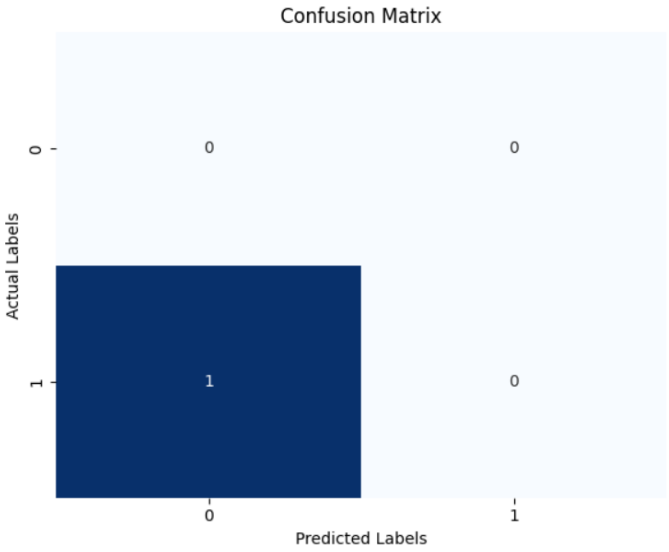
OUTPUT:

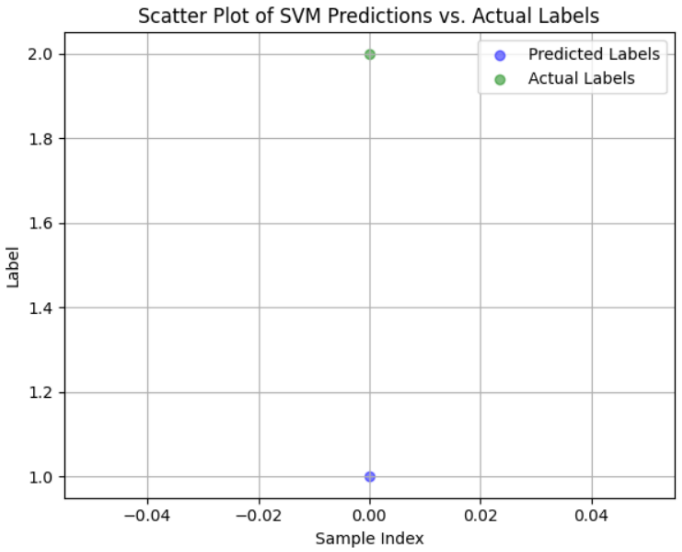


PROGRAM:

```
import matplotlib.pyplot as plt

from sklearn.metrics import confusion_matrix

import seaborn as sns

conf_matrix = confusion_matrix(y_test, predictions2)

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)

sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False)

plt.xlabel("Predicted Labels")

plt.ylabel("Actual Labels")

plt.title("Confusion Matrix")

plt.subplot(1, 2, 2)

plt.scatter(range(len(predictions2)), predictions2, label='Predicted Labels', c='blue', alpha=0.5)

plt.scatter(range(len(y_test)), y_test, label='Actual Labels', c='green', alpha=0.5)

plt.xlabel("Sample Index")

plt.ylabel("Label")

plt.title("Scatter Plot of SVM Predictions vs. Actual Labels")

plt.legend(loc='upper right')

plt.grid(True)

plt.tight_layout()

plt.show()
```

OUTPUT:





PROGRAM:

```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

import numpy as np

mse = mean_squared_error(y_test, predictions3)

r2 = r2_score(y_test, predictions3)

mae = mean_absolute_error(y_test, predictions3)

print("Mean Squared Error:", mse)

print("R-squared (R^2) Score:", r2)

print("Mean Absolute Error (MAE):", mae)

rmse = np.sqrt(mse)

mape = np.mean(np.abs((y_test - predictions) / y_test)) * 100

n = len(y_test)

p = X_test.shape[1]
```

```
adj_r2 = 1 - (1 - r2) * (n - 1) / (n - p - 1)
print("Root Mean Squared Error (RMSE):", rmse)

print("Mean Absolute Percentage Error (MAPE):", mape)

print("Adjusted R-squared:", adj_r2)
```

OUTPUT:

```
Mean Squared Error: 0.3209714214075377
R-squared (R^2) Score: nan
Mean Absolute Error (MAE): 0.5665433976382901
Root Mean Squared Error (RMSE): 0.5665433976382901
Mean Absolute Percentage Error (MAPE): 0.0
Adjusted R-squared: nan
```

MODEL COMPARISON:

PROGRAM:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression, Ridge, Lasso

from sklearn.ensemble import RandomForestRegressor

from sklearn.svm import SVR

from sklearn.preprocessing import PolynomialFeatures

from xgboost import XGBRegressor

from sklearn.metrics import mean_squared_error

np.random.seed(0)
```

```
X = np.random.rand(100, 1)
y = 2 * X.squeeze() + 1 + 0.1 * np.random.randn(100)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
 models = {
    "Linear Regression": LinearRegression(),
    "Ridge Regression": Ridge(alpha=1.0),
    "Lasso Regression": Lasso(alpha=1.0),
    "Polynomial Regression": Pipeline([
        ("poly_features", PolynomialFeatures(degree=2)),
        ("linear_regression", LinearRegression())
    ]),
    "SVM": SVR(),
    "Random Forest Regression": RandomForestRegressor(n_estimators=100),
    "XGBoost Regression": XGBRegressor()
}
mse_values = {}
for model_name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    mse_values[model_name] = mse
mse_df = pd.DataFrame.from_dict(mse_values, orient='index', columns=['MSE'])
mse_df = mse_df.sort_values(by='MSE')
print("MSE Comparison Table:")
```

print(mse_df)

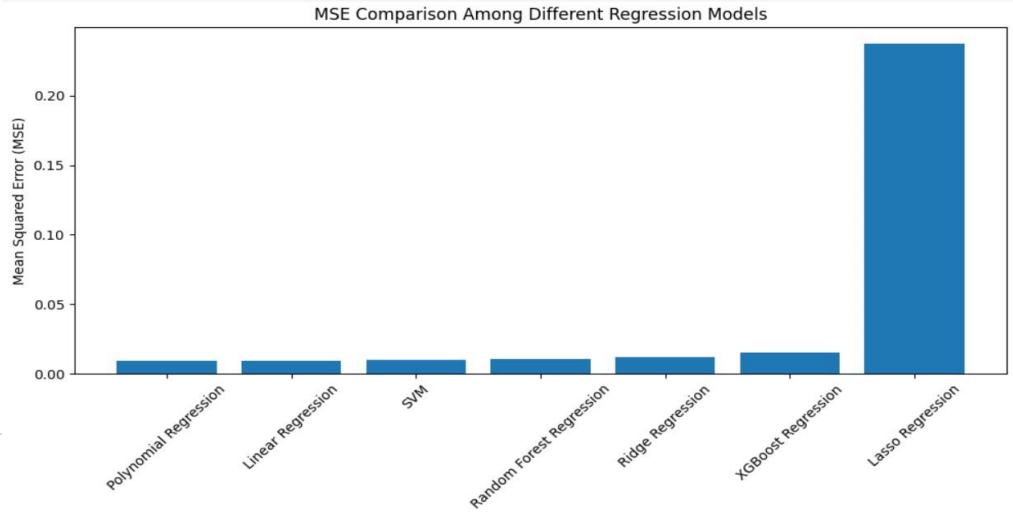
OUTPUT:

```
MSE Comparison Table:
                                MSE
Polynomial Regression      0.009116
Linear Regression          0.009178
SVM                        0.009697
Random Forest Regression   0.010862
Ridge Regression           0.011812
XGBoost Regression         0.015321
Lasso Regression           0.237121
```

PROGRAM:

```
plt.figure(figsize=(10, 6))
plt.bar(mse_df.index, mse_df['MSE'])
plt.xlabel('Regression Models')
plt.ylabel('Mean Squared Error (MSE)')
plt.title('MSE Comparison Among Different Regression Models')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

OUTPUT:



FEATURE ENGINEERING:

Feature engineering for chatbots involves transforming input data into structured formats for machine learning algorithms to understand and learn from. This process improves model recognition of patterns, intents, and context, enabling accurate responses. Key techniques include tokenization, word embeddings, N-grams, and part-of-speech tagging, which help model understanding meaning, context, and grammatical structure of user inputs.”Text length” is an example of Feature engineering.

Tokenization:

Tokenization is the process of breaking down text into smaller units, such as words, subwords, or characters, which is crucial for natural language processing tasks.

Word Embeddings:

Word embeddings, dense vector representations of words, capture semantic relationships, aiding language models in understanding meaning and context. Pre-trained embeddings like Word2Vec, GloVe, or FastText are commonly used.

TF-IDF (Term Frequency-Inverse Document Frequency):

TF-IDF is a numerical statistic that measures the frequency of a word in a document, adjusting for its frequency across multiple documents, aiding in text classification and information retrieval.

N-grams:

N-grams are sequences of N words, characters, or symbols in text, capturing contextual information beyond individual words, useful for sentiment analysis and language generation tasks.

Part-of-Speech Tagging:

Part-of-speech tagging helps models understand user input grammatical structure by identifying the part of speech (noun, verb, adjective) of each word in a sentence.

ADDING TEXT LENGTH AS A FEATURE:

PROGRAM:

```
import pandas as pd
data = {
   'Message': [
   "i'm fine. how about yourself?",
   "i'm doing well. how about you?",
   "you like the rain?",
   "where's your money?",
   "did you tell her about school?",
   "thank you very much.",
   "what do you want to do?",
   "why do you like it?",

   ]}
df = pd.DataFrame(data)
df['TextLength'] = df['Message'].apply(lambda x: len(x.split()))
print(df)
```

```
                          Message  TextLength
0   i'm fine. how about yourself?           5
1  i'm doing well. how about you?           6
2              you like the rain?           4
3             where's your money?           3
4  did you tell her about school?           6
5            thank you very much.           4
6         what do you want to do?           6
7             why do you like it?           5
```

In feature engineering, I use text preprocessing,TD-IDF vectorization, and label encoding for the chatbot dataset.

PROGRAM:

```
import pandas as pd

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.preprocessing import LabelEncoder

data = {

   'Message': [

   "i'm fine. how about yourself?",

   "i'm doing well. how about you?",

   "you like the rain?",

   "where's your money?",

   "did you tell her about school?",

   "thank you very much.",

   "what do you want to do?",

   "why do you like it?",

   ],

 'Label': ["greeting", "greeting", "weather", "finance", "school", "greeting", "personal", "personal"]

}

df = pd.DataFrame(data)

df['Message'] = df['Message'].str.lower()

df['Message'] = df['Message'].str.replace(r'[^\w\s]', '')
```

```
tfidf_vectorizer = TfidfVectorizer(max_features=100)
X_tfidf = tfidf_vectorizer.fit_transform(df['Message'])
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(df['Label'])
df_tfidf = pd.DataFrame(X_tfidf.toarray())
df = pd.concat([df_tfidf, pd.DataFrame({'Label': y_encoded})], axis=1)
print(df)
```

OUTPUT:

```
          0         1         2         3         4         5         6  \
0  0.364907  0.000000  0.000000  0.000000  0.504577  0.000000  0.422875
1  0.353832  0.000000  0.000000  0.489264  0.000000  0.000000  0.410042
2  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
3  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
4  0.331033  0.457738  0.000000  0.000000  0.000000  0.457738  0.000000
5  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
6  0.000000  0.000000  0.680936  0.000000  0.000000  0.000000  0.000000
7  0.000000  0.000000  0.438402  0.000000  0.000000  0.000000  0.000000

          7         8         9  ...        18        19        20        21  \
0  0.422875  0.000000  0.000000  ...  0.000000  0.000000  0.000000  0.000000
1  0.410042  0.000000  0.000000  ...  0.000000  0.000000  0.489264  0.000000
2  0.000000  0.000000  0.487776  ...  0.000000  0.000000  0.000000  0.000000
3  0.000000  0.000000  0.000000  ...  0.000000  0.000000  0.000000  0.000000
4  0.000000  0.000000  0.000000  ...  0.000000  0.000000  0.000000  0.000000
5  0.000000  0.000000  0.000000  ...  0.554725  0.000000  0.000000  0.000000
6  0.000000  0.000000  0.000000  ...  0.000000  0.406249  0.000000  0.406249
7  0.000000  0.523104  0.438402  ...  0.000000  0.000000  0.000000  0.000000

        22        23        24       25        26  Label
0  0.00000  0.000000  0.000000  0.00000  0.504577      1
1  0.00000  0.000000  0.244491  0.00000  0.000000      1
2  0.00000  0.000000  0.290840  0.00000  0.000000      4
3  0.57735  0.000000  0.000000  0.57735  0.000000      0
4  0.00000  0.000000  0.228737  0.00000  0.000000      3
5  0.00000  0.000000  0.277202  0.00000  0.000000      1
6  0.00000  0.000000  0.203007  0.00000  0.000000      2
7  0.00000  0.523104  0.261401  0.00000  0.000000      2

[8 rows x 28 columns]
```

VARIOUS FEATURES TO PERFORM MODEL TRAINING:

Chatbot model training uses natural language processing, sentiment analysis, and intent recognition to understand user inputs, extract relevant information, and respond contextually. These features enhance the user's conversational experience by providing accurate, personalized, and efficient interactions.

PROGRAM:

```
import pandas as pd

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_scope

data = {

    'Message': [

    "i'm fine. how about yourself?",

    "i'm doing well. how about you?",

    "you like the rain?",

    "where's your money?",

    "did you tell her about school?",

    "thank you very much.",

    "what do you want to do?",

    "why do you like it?",

    ],

 'Intent': ["greeting", "greeting", "weather", "finance", "school", "greeting", "personal", "personal"]
```

```
}

df = pd.DataFrame(data)

tfidf_vectorizer = TfidfVectorizer()

X = tfidf_vectorizer.fit_transform(df['Message'])

X_train, X_test, y_train, y_test = train_test_split(X, df['Intent'], test_size=0.2, random_state=42)

model = RandomForestClassifier(n_estimators=100, random_state=42)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")
```

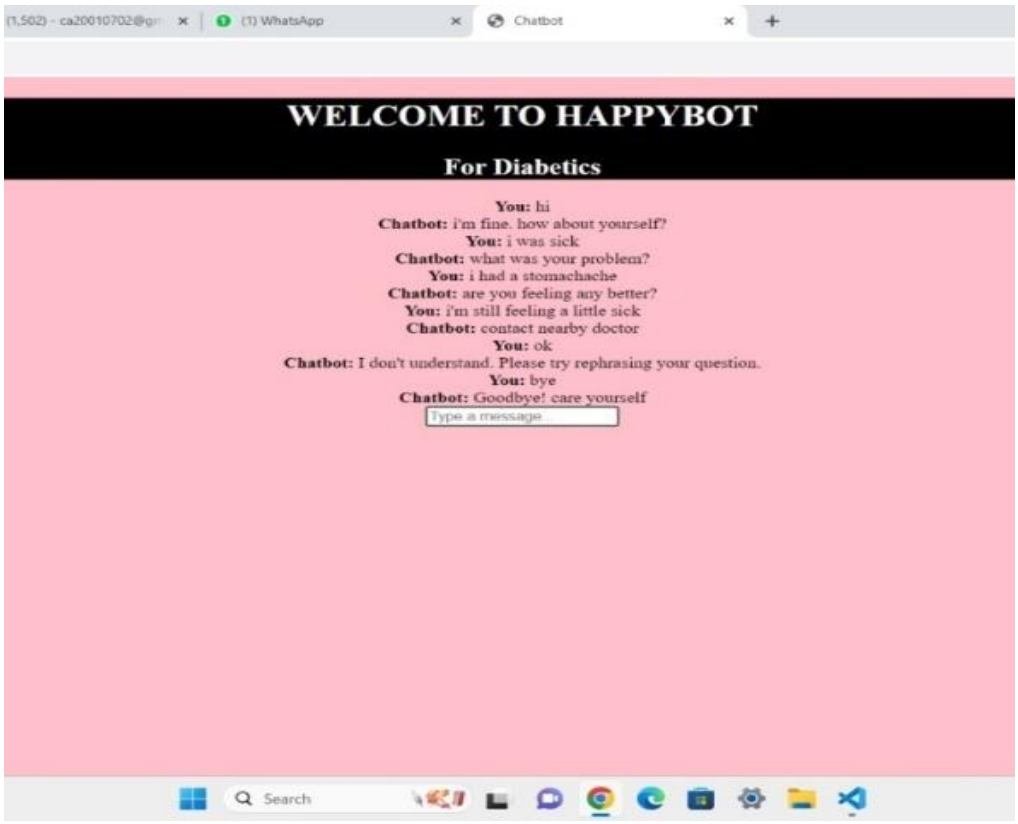
OUTPUT:

```
Accuracy: 0.50
```

FLASK:

Flask is a Python web framework designed for developers to build web applications, APIs, and interactive services with minimal effort. It provides basic tools for routing, handling requests, and rendering web pages. Flask is often used in combination with other Python libraries to create chatbot interfaces and web applications, allowing users to interact with chatbots through a web browser.

```
from flask import Flask, request, jsonify
app = Flask(_name_)
@app.route('/')
def index():
    return open('index.html', 'r').read()
@app.route('/ask', methods=['POST'])
def ask():
    user_message = request.json.get('userMessage', '')
    response = chatbot_response(user_message)
    return jsonify({'response': response})
if _name_ == '_main_':
    app.run(debug=True)
```



CONCLUSION:

Building a chatbot involves a series of essential activities, starting with feature engineering to identify and extract relevant data. Model training is the core, where the chatbot learns to understand and respond effectively. Rigorous evaluation ensures its accuracy and user-friendliness. Finally, integrating it into a web app with Flask makes the chatbot accessible and interactive, enhancing user experiences and providing valuable support.