Chatbot for Mental Health Conversations, by Maureen and Nafisah

Mile Stone Three

This mental health chatbot is meant to respond to frequently asked mental health quetions, we got our data set from kaggle, it is a json file that contains conversations arround mental health. Right not the only challenge we are facing is that our accuracy is very poor and we are trying to figure out ways to increasing it either by using a different model algorithm or doing more data cleaning.

Our current model uses the SVM algorithm(support vector manchine) and we are considering using LSTM.

Data Preparation

Load the dataset into an appropriate data structure. Analyze the dataset to grasp its structure and distribution. Clean the data by removing extraneous characters, converting text to lowercase, and addressing any missing values.

```
#!pip install tensorflow
import collections

import numpy as np

from tensorflow.keras.preprocessing.text import Tokenizer # Use tensorflow.keras.preprocessing
from tensorflow.keras.preprocessing.sequence import pad_sequences # Use tensorflow.keras.preprocessing
from tensorflow.keras.models import Model # Use tensorflow.keras.models
from tensorflow.keras.layers import GRU, Input, Dense, TimeDistributed, Activation, RepeatVector, Bidirecti
from tensorflow.keras.layers import Embedding, GlobalMaxPooling1D, GlobalAveragePooling1D, GRU # Use tensor
from tensorflow.keras.optimizers import Adam # Use tensorflow.keras.optimizers
from tensorflow.keras.losses import sparse_categorical_crossentropy # Use tensorflow.keras.losses
from tensorflow.keras.utils import plot_model # Use tensorflow.keras.utils
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, TensorBoard # Use tensorflow.keras.c
from nltk.translate.bleu_score import sentence_bleu, corpus_bleu
import helper

import pandas as pd
import matplotlib.pyplot as plt
from tqdm import tqdm
tqdm.pandas()
import re
import numpy as np
import pandas as pd
import os
import json
```

```
# Contraction mapping
contraction_mapping = {
    "ain't": "is not", "aren't": "are not", "can't": "cannot", "'cause": "because",
    "could've": "could have", "couldn't": "could not", "didn't": "did not", "doesn't": "does not",
    "don't": "do not", "hadn't": "had not", "hasn't": "has not", "haven't": "have not",
    "he'd": "he would", "he'll": "he will", "he's": "he is", "how'd": "how did",
    "how'd'y": "how do you", "how'll": "how will", "how's": "how is", "I'd": "I would",
    "I'd've": "I would have", "I'll": "I will", "I'll've": "I will have", "I'm": "I am",
    "I've": "I have", "i'd": "i would", "i'd've": "i would have", "i'll": "i will",
    "i'll've": "i will have", "i'm": "i am", "i've": "i have", "isn't": "is not",
    "it'd": "it would", "it'd've": "it would have", "it'll": "it will", "it'll've": "it will have",
    "it's": "it is", "let's": "let us", "ma'am": "madam", "mayn't": "may not",
    "might've": "might have", "mightn't": "might not", "mightn't've": "might not have",
    "must've": "must have", "mustn't": "must not", "mustn't've": "must not have",
    "needn't": "need not", "needn't've": "need not have", "o'clock": "of the clock",
    "oughtn't": "ought not", "oughtn't've": "ought not have", "shan't": "shall not",
    "sha'n't": "shall not", "shan't've": "shall not have", "she'd": "she would",
    "she'd've": "she would have", "she'll": "she will", "she'll've": "she will have",
    "she's": "she is", "should've": "should have", "shouldn't": "should not",
    "shouldn't've": "should not have", "so've": "so have", "so's": "so as", "this's": "this is",
    "that'd": "that would", "that'd've": "that would have", "that's": "that is",
    "there'd": "there would", "there'd've": "there would have", "there's": "there is",
    "here's": "here is", "they'd": "they would", "they'd've": "they would have",
    "they'll": "they will", "they'll've": "they will have", "they're": "they are",
    "they've": "they have", "to've": "to have", "wasn't": "was not", "we'd": "we would",
    "we'd've": "we would have", "we'll": "we will", "we'll've": "we will have",
    "we're": "we are", "we've": "we have", "weren't": "were not", "what'll": "what will",
    "what'll've": "what will have", "what're": "what are", "what's": "what is",
    "what've": "what have", "when's": "when is", "when've": "when have", "where'd": "where did",
    "where's": "where is", "where've": "where have", "who'll": "who will",
    "who'll've": "who will have", "who's": "who is", "who've": "who have",
    "why's": "why is", "why've": "why have", "will've": "will have", "won't": "will not",
    "won't've": "will not have", "would've": "would have", "wouldn't": "would not",
    "wouldn't've": "would not have", "y'all": "you all", "y'all'd": "you all would",
    "y'all'd've": "you all would have", "y'all're": "you all are", "y'all've": "you all have",
    "you'd": "you would", "you'd've": "you would have", "you'll": "you will",
    "you'll've": "you will have", "you're": "you are", "you've": "you have",
    'u.s':'america', 'e.g':'for example'
}

# Function to clean contractions
def clean_contractions(text, mapping):
    specials = ["’", "‘", "´", "`"]
    for s in specials:
        text = text.replace(s, "'")
    text = ' '.join([mapping.get(t, t) for t in text.split(" ")])
    return text


import json
import string

# Function to clean text
def clean_text(text):
    # Lowercase the text
    text = text.lower()
    # Remove punctuation
    text = text.translate(str.maketrans('', '', string.punctuation))
    # Expand contractions
    text = clean_contractions(text, contraction_mapping)
```

```
        return text

# Load the JSON file
file_path = '/content/mentalhealth.json'  # Replace with the actual path if necessary
with open(file_path, 'r') as file:
    data = json.load(file)

# Iterate through the intents and clean the patterns and responses
for intent in data['intents']:
    intent['patterns'] = [clean_text(pattern) for pattern in intent['patterns']]
    intent['responses'] = [clean_text(response) for response in intent['responses']]

print("Data cleaning completed.")
```

```
Data cleaning completed.
```

Start coding or generate with AI.

```
import numpy as np
import pandas as pd
import os
import json

# List all files under the /content directory to verify the upload
for dirname, _, filenames in os.walk('/content'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# Assuming the file is uploaded directly to /content
file_path = '/content/mentalhealth_cleaned.json'

# Load the JSON data into a DataFrame
with open(file_path, 'r') as f:
    data = json.load(f)

df = pd.DataFrame(data['intents'])
df
dic = {"tag":[], "patterns":[], "responses":[]}
for i in range(len(df)):
    ptrns = df[df.index == i]['patterns'].values[0]
    rspns = df[df.index == i]['responses'].values[0]
    tag = df[df.index == i]['tag'].values[0]
    for j in range(len(ptrns)):
        dic['tag'].append(tag)
        dic['patterns'].append(ptrns[j])
        dic['responses'].append(rspns)

df = pd.DataFrame.from_dict(dic)
df
```

```
/content/mentalhealth.json
/content/mentalhealth_cleaned.json
/content/.config/default_configs.db
/content/.config/.last_opt_in_prompt.yaml
/content/.config/.last_update_check.json
/content/.config/active_config
/content/.config/.last_survey_prompt.yaml
/content/.config/config_sentinel
/content/.config/gce
/content/.config/configurations/config_default
/content/.config/logs/2024.08.01/13.23.13.013469.log
/content/.config/logs/2024.08.01/13.23.53.840598.log
/content/.config/logs/2024.08.01/13.23.42.862025.log
/content/.config/logs/2024.08.01/13.23.53.277859.log
/content/.config/logs/2024.08.01/13.23.33.499801.log
/content/.config/logs/2024.08.01/13.23.43.680314.log
/content/sample_data/anscombe.json
/content/sample_data/README.md
/content/sample_data/california_housing_test.csv
/content/sample_data/mnist_test.csv
/content/sample_data/california_housing_train.csv
/content/sample_data/mnist_train_small.csv
```

	tag	patterns	responses
0	definition	what does it mean to have a mental illness	[Mental illnesses are health conditions that d...
1	definition	what is mental health illness	[Mental illnesses are health conditions that d...
2	definition	describe mental health illness	[Mental illnesses are health conditions that d...
3	definition	what is a mental illness	[Mental illnesses are health conditions that d...
4	definition	define mental health illness	[Mental illnesses are health conditions that d...
...	...	...	...
119	goodbye	have a good day	[sad to see you go , talk to you later, goodby...
120	goodbye	talk to you later	[sad to see you go , talk to you later, goodby...
121	goodbye	ttyl	[sad to see you go , talk to you later, goodby...
122	goodbye	i got to go	[sad to see you go , talk to you later, goodby...
123	goodbye	gtg	[sad to see you go , talk to you later, goodby...

124 rows × 3 columns

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df['tag'].unique()
```

```
array(['definition', 'affects_whom', 'what_causes', 'recover', 'steps',
       'find_help', 'treatment_options', 'treatment_tips',
       'professional_types', 'right_professional', 'greeting', 'goodbye'],
      dtype=object)
```

```
df.info()
```

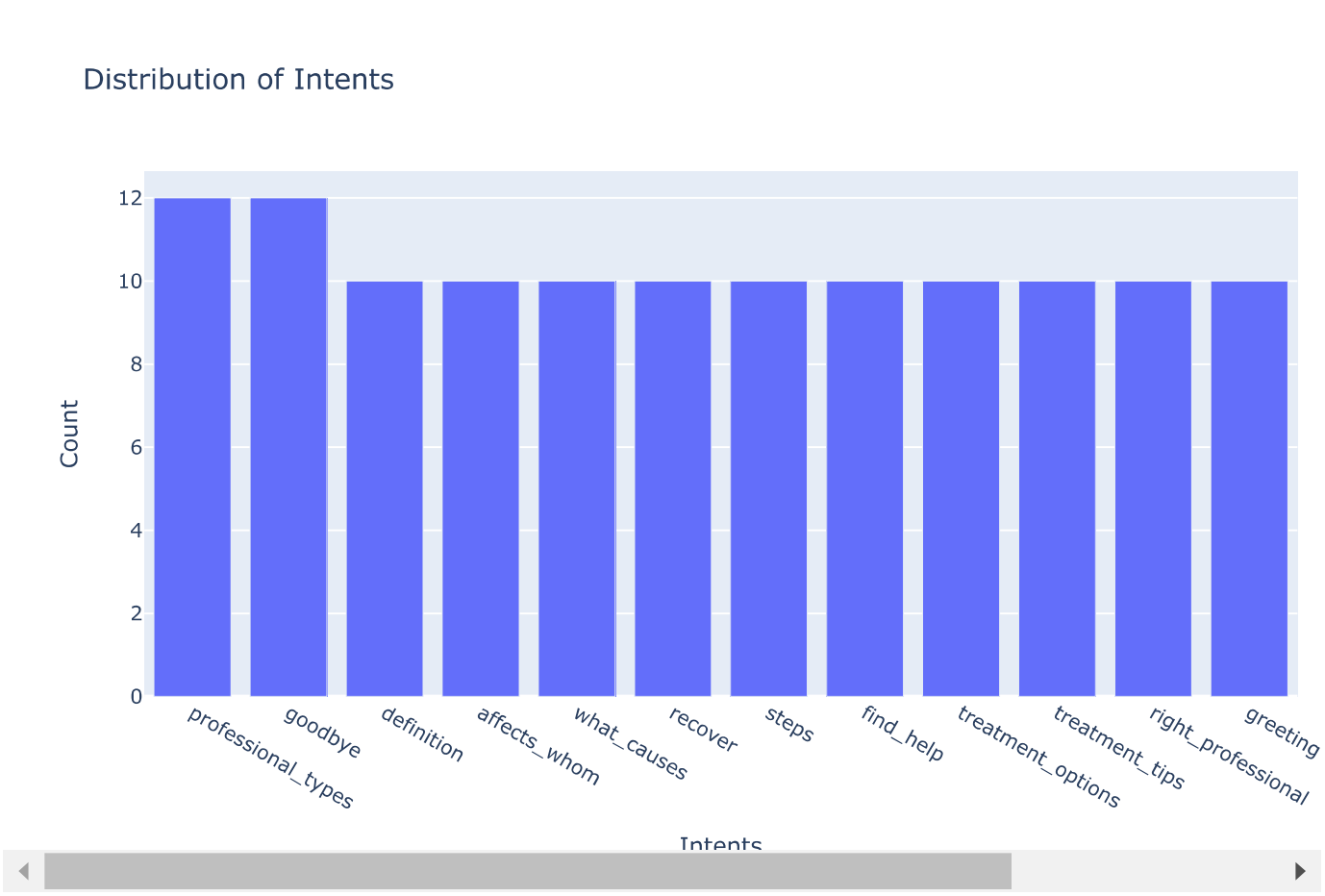
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 124 entries, 0 to 123
Data columns (total 3 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
```

```
 0   tag        124 non-null    object
 1   patterns   124 non-null    object
 2   responses  124 non-null    object
dtypes: object(3)
memory usage: 3.0+ KB
```

Exploratory Data Analysis

- Analyze the distribution of intents in the dataset.
- Visualize the frequency of different intents using a bar plot from the Plotly library. The x-axis can represent the intents, and the y-axis can represent the count of patterns or responses associated with each intent.

```
import plotly.graph_objects as go

intent_counts = df['tag'].value_counts()
fig = go.Figure(data=[go.Bar(x=intent_counts.index, y=intent_counts.values)])
fig.update_layout(title='Distribution of Intents', xaxis_title='Intents', yaxis_title='Count')
fig.show()
```

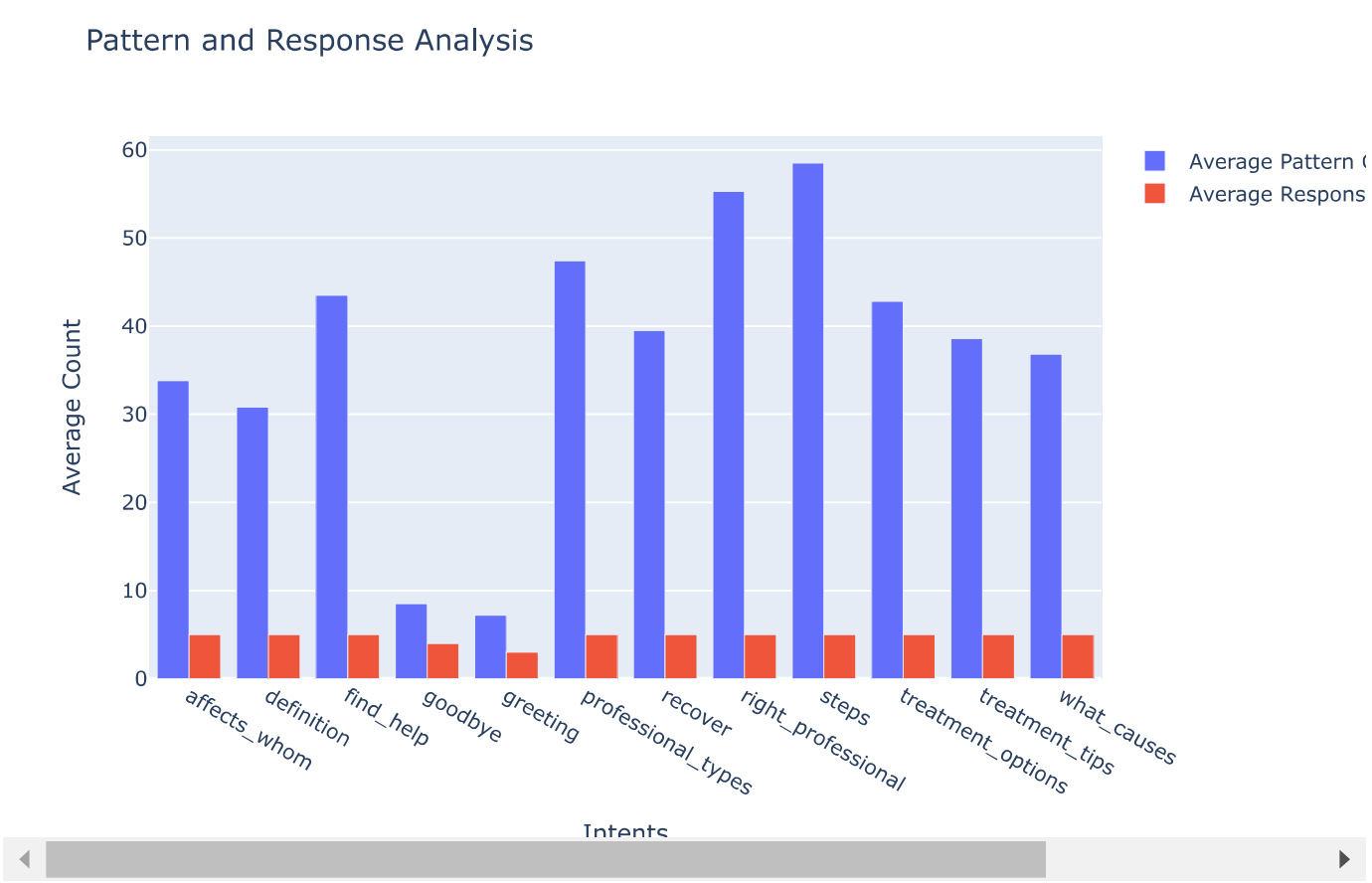


Pattern and Response Analysis

- Explore the patterns and responses associated with each intent.
- Calculate the average number of patterns and responses per intent.

- Visualize this information using a Plotly bar plot, where the x-axis represents the intents, and the y-axis represents the average count of patterns or responses.
- Interpret the plot to understand the varying degrees of complexity and diversity in patterns and responses across different intents.

```
df['pattern_count'] = df['patterns'].apply(lambda x: len(x))
df['response_count'] = df['responses'].apply(lambda x: len(x))
avg_pattern_count = df.groupby('tag')['pattern_count'].mean()
avg_response_count = df.groupby('tag')['response_count'].mean()

fig = go.Figure()
fig.add_trace(go.Bar(x=avg_pattern_count.index, y=avg_pattern_count.values, name='Average Pattern Count'))
fig.add_trace(go.Bar(x=avg_response_count.index, y=avg_response_count.values, name='Average Response Count'
fig.update_layout(title='Pattern and Response Analysis', xaxis_title='Intents', yaxis_title='Average Count'
fig.show()
```

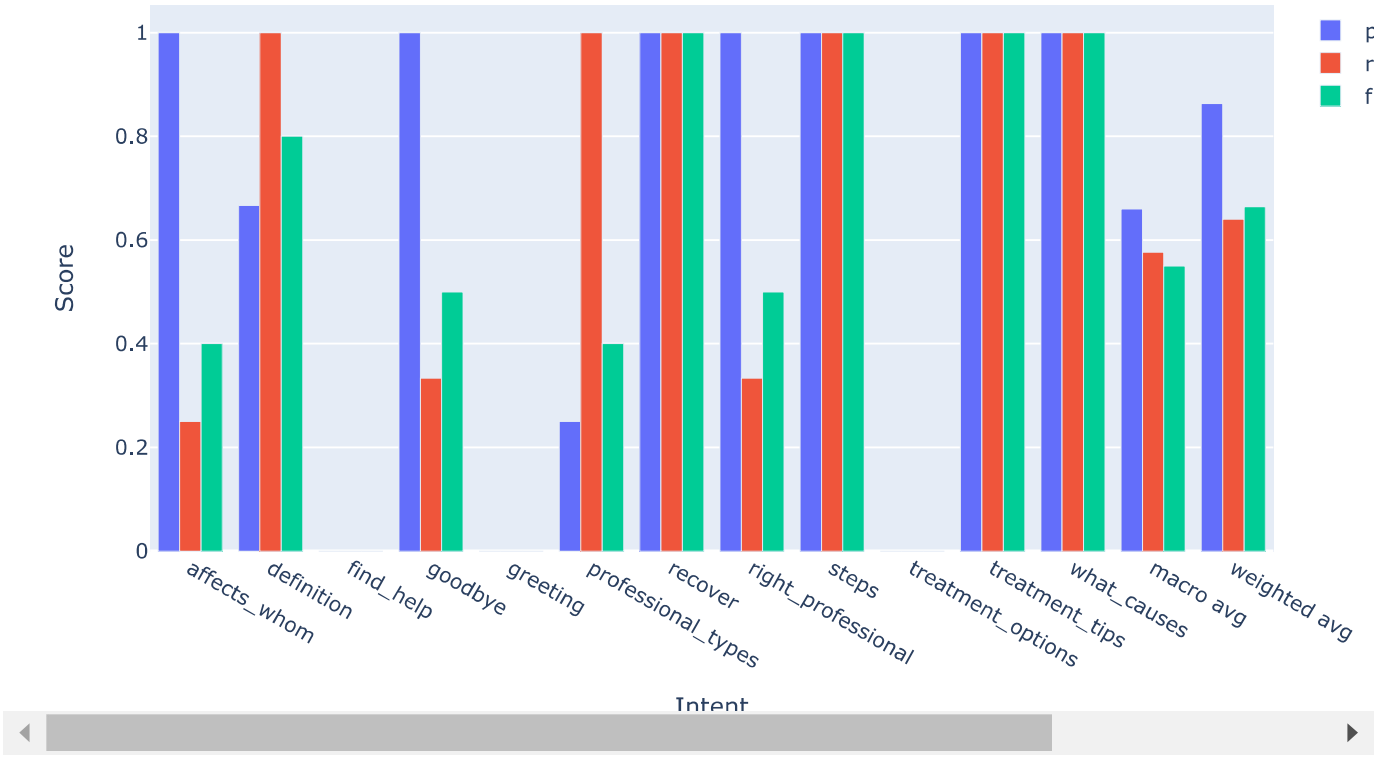


Intent Prediction Model

- Split the dataset into training and testing sets.
- Implement a machine learning or deep learning model suitable for intent prediction, such as a text classification model.
- Vectorize the text data (e.g., using TF-IDF or word embeddings) and train the model using the patterns as input and the corresponding intents as target variables.
- Evaluate the model's performance on the testing set using appropriate metrics like accuracy, precision, recall, and F1-score.

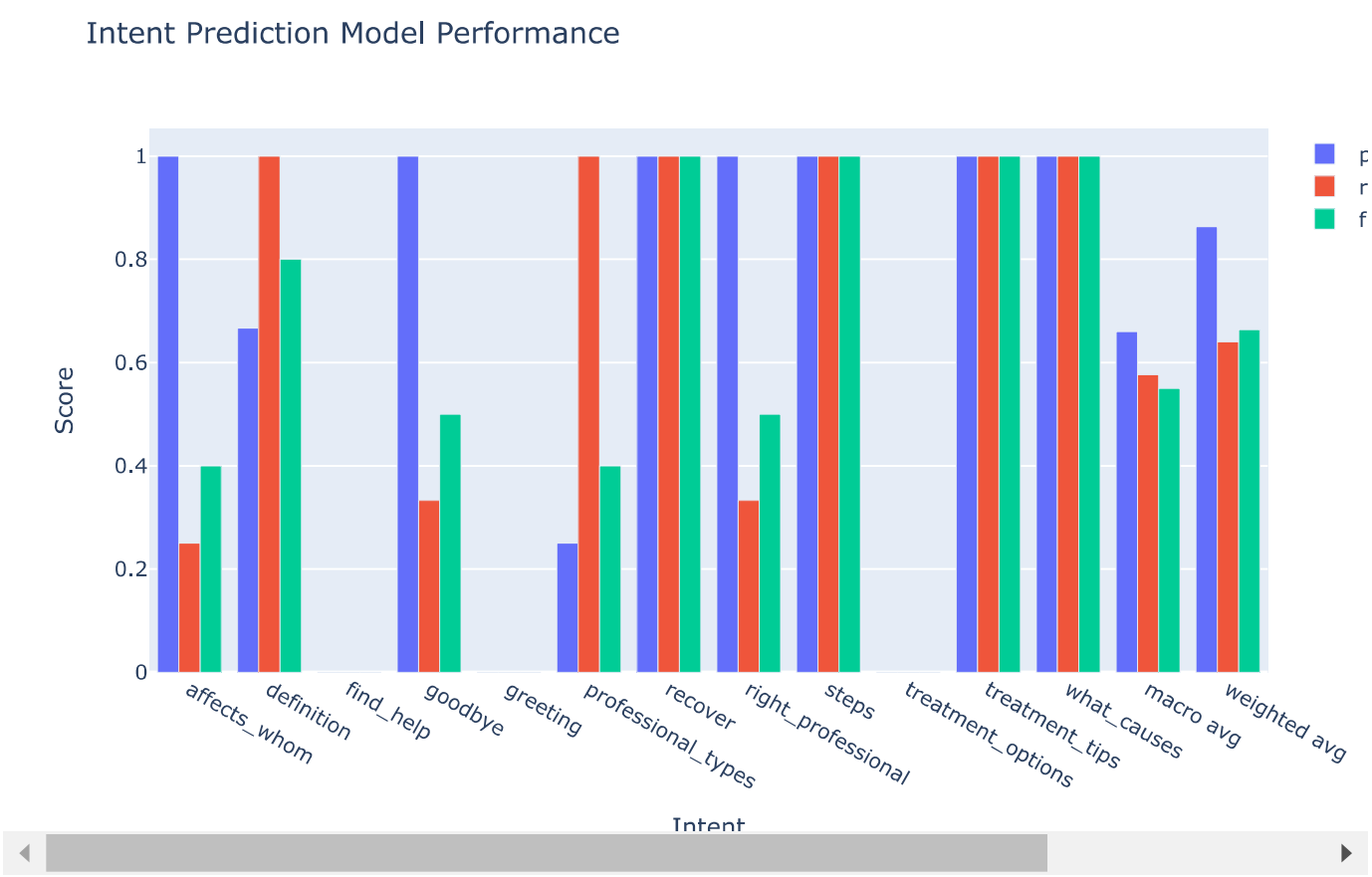
- Visualize the model's performance using a Plotly bar plot, where the x-axis represents the evaluation metrics, and the y-axis represents the corresponding scores.
- Interpret the plot to analyze the effectiveness of the intent prediction model.

```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.metrics import classification_report
import plotly.graph_objects as go


# Split the dataset into training and testing sets
X = df['patterns']
y = df['tag']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Vectorize the text data using TF-IDF
vectorizer = TfidfVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

# Train a Support Vector Machine (SVM) classifier
model = SVC()
model.fit(X_train_vec, y_train)

# Predict intents for the testing set
y_pred = model.predict(X_test_vec)

# Evaluate the model's performance
report = classification_report(y_test, y_pred, output_dict=True, zero_division=0)

# Convert float values in the report to dictionaries
report = {label: {metric: report[label][metric] for metric in report[label]} for label in report if isinsta

# Extract evaluation metrics
labels = list(report.keys())
evaluation_metrics = ['precision', 'recall', 'f1-score']
metric_scores = {metric: [report[label][metric] for label in labels if label in report] for metric in evalu

# Visualize the model's performance using a Plotly bar plot
fig = go.Figure()
for metric in evaluation_metrics:
    fig.add_trace(go.Bar(name=metric, x=labels, y=metric_scores[metric]))

fig.update_layout(title='Intent Prediction Model Performance',
                  xaxis_title='Intent',
                  yaxis_title='Score',
                  barmode='group')

fig.show()
```




```
from sklearn.metrics import classification_report, accuracy_score
# Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred, output_dict=True, zero_division=0)

print(f"Model Accuracy: {accuracy}")

# Convert float values in the report to dictionaries
report = {label: {metric: report[label][metric] for metric in report[label]} for label in report if isinsta

# Extract evaluation metrics
labels = list(report.keys())
evaluation_metrics = ['precision', 'recall', 'f1-score']
metric_scores = {metric: [report[label][metric] for label in labels if label in report] for metric in evalu

# Visualize the model's performance using a Plotly bar plot
fig = go.Figure()
for metric in evaluation_metrics:
    fig.add_trace(go.Bar(name=metric, x=labels, y=metric_scores[metric]))

fig.update_layout(title='Intent Prediction Model Performance',
                  xaxis_title='Intent',
                  yaxis_title='Score',
                  barmode='group')

fig.show()
```

```
Model Accuracy: 0.64
```




```
from sklearn.metrics import accuracy_score, classification_report


# Predict the labels for the test set
y_pred = model.predict(X_test_vec)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

# Print accuracy
print(f"Model Accuracy: {accuracy * 100:.2f}%")

# Optional: Detailed classification report
report = classification_report(y_test, y_pred)
print(report)
```

```
Model Accuracy: 64.00%
                     precision    recall  f1-score   support

       affects_whom       1.00      0.25      0.40         4
         definition       0.67      1.00      0.80         2
          find_help       0.00      0.00      0.00         2
            goodbye       1.00      0.33      0.50         3
           greeting       0.00      0.00      0.00         0
  professional_types       0.25      1.00      0.40         1
            recover       1.00      1.00      1.00         2
  right_professional       1.00      0.33      0.50         3
              steps       1.00      1.00      1.00         3
   treatment_options       0.00      0.00      0.00         0
```

```
      treatment_tips       1.00      1.00      1.00         3
         what_causes       1.00      1.00      1.00         2

            accuracy                           0.64        25
           macro avg       0.66      0.58      0.55        25
        weighted avg       0.86      0.64      0.66        25

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning

Recall and F-score are ill-defined and being set to 0.0 in labels with no true samples. Use `zero_divis

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning

Recall and F-score are ill-defined and being set to 0.0 in labels with no true samples. Use `zero_divis

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning

Recall and F-score are ill-defined and being set to 0.0 in labels with no true samples. Use `zero_divis
```

Prediction Model Deployment

- Once satisfied with the model's performance, deploy the intent prediction model in a chatbot framework.
- Utilize the trained model to predict intents based on user input in real-time.
- Implement an appropriate response generation mechanism to provide relevant and empathetic responses based on the predicted intents.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.metrics import classification_report
import plotly.graph_objects as go
import pickle # import pickle to save the model and vectorizer


# Split the dataset into training and testing sets
X = df['patterns']
y = df['tag']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Vectorize the text data using TF-IDF
vectorizer = TfidfVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

# Train a Support Vector Machine (SVM) classifier
model = SVC(probability=True) # Add probability=True for predict_proba
model.fit(X_train_vec, y_train)

# Save the model and vectorizer
with open('svm_model.pkl', 'wb') as model_file:
    pickle.dump(model, model_file)
with open('vectorizer.pkl', 'wb') as vec_file:
    pickle.dump(vectorizer, vec_file)

# ... rest of the code (prediction, evaluation, visualization) ...


import random
import json

# Load the intents data from the JSON file
def load_intents(file_path):
    with open(file_path, 'r') as file:
        return json.load(file)

# Load the intents data
intents_data = load_intents('mentalhealth_cleaned.json')

# Function to predict intents based on user input
def predict_intent(user_input):
    try:
        # Vectorize the user input
        user_input_vec = vectorizer.transform([user_input])

        # Predict the intent
        intent = model.predict(user_input_vec)[0]

        return intent
    except Exception as e:
        print(f"Error in predicting intent: {e}")
        return None

# Function to generate responses based on predicted intents
def generate_response(user_input):
```

```
        # Check if the user input matches any pattern in the loaded data
        for intent_data in intents_data['intents']:
```