Task 1:

```
TASK1
"""

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
import numpy as np

data = pd.read_csv("/content/googleplaystore.csv")

data.info()

#analysing category
data['Category'].unique()

data[data['Category'] == '1.9']

#removing 1.9 value and replacing it with NA(Not Applicable) as 1.9 is
incorrect value
data['Category'] = data['Category'].str.replace("1.9","NA")

#Number of apps in each category
plt.figure(figsize=(10, 6))
sns.countplot(data=data, y='Category')
plt.xlabel('Count')
plt.ylabel('App Category')
plt.title('Distribution of App Categories')
plt.show()

#analyzing rating
data['Rating'].unique()

# changing the data type of reviews from object to numeric
data['Rating'] = pd.to_numeric(data['Rating'], errors='coerce')
data['Rating'].dtype

#replacing nan with mean
```

```
data['Rating'] = data['Rating'].replace(np.nan, np.mean(data['Rating']))

#visualizing rating
plt.hist(data['Rating'], edgecolor='black')
plt.xlabel('Ratings')
plt.ylabel('Count')
plt.title('Distribution of Ratings')
plt.show()

#analyzing reviews
data['Reviews'].unique()

# replacing the value
data['Reviews'] = data.Reviews.replace("3.0M",3000000.0)
# changing data type
data['Reviews'] = data.Reviews.astype(float)

#descriptive statistics
category = data["Category"].describe()
rating = data["Rating"].describe()
reviews = data["Reviews"].describe()
print(category , "\n\n" , rating , "\n\n" , reviews)

data['Installs'].unique()

# removing , + & Free
data['Installs'] = data['Installs'].str.replace(",","")

data['Installs'] = data['Installs'].str.replace("+","")

data['Installs'] = data['Installs'].str.replace("Free","NaN")

# changing the data type of the column
data['Installs'] = data['Installs'].astype(float)
data['Installs'].dtype

#correlation between number of installs and rating
rating = data['Rating']
data['Installs'] = pd.to_numeric(data['Installs'], errors='coerce')
installs = data['Installs']
```

```
correlation = rating.corr(installs)
print("Correlation coefficient:", correlation)

#scatter plot between rating and install to see their relationship
plt.scatter(x=data['Installs'] , y=data["Rating"])
plt.xlabel("Installs")
plt.ylabel("Rating")
plt.show()

#scatter plot between size and reviews to see their relationship
plt.scatter(x=data['Size'] , y=data["Reviews"])
plt.xlabel("Size")
plt.ylabel("Reviews")
plt.show()

#Identify the most popular app categories and visualize their distribution
using bar charts or pie charts.
counts = data['Category'].value_counts()
popular = counts.head(5)
print(popular)

#barchart
plt.bar(x=popular.index, height=popular)
plt.xlabel('Count')
plt.ylabel('App Category')
plt.title('5 Most Popular App Categories')
plt.show()
print("\n\n")
#piechart
plt.pie(popular, labels=popular.index, autopct='%1.1f%%')
plt.title('5 Most Popular App Categories')
plt.show()

#Analyze the distribution of app ratings and reviews to understand user
sentiments and identify any potential outliers or suspicious patterns.
#plotting histogram to see distribution
plt.hist(data=data, x='Rating' , edgecolor = 'black')
plt.xlabel('Rating')
plt.ylabel('Count')
plt.title('Distribution of App Ratings')
```

```
plt.show()
print("\n\n")

plt.hist(data=data, x='Reviews' , edgecolor = 'black')
plt.xlabel('Review')
plt.ylabel('Count')
plt.title('Distribution of App Review')
plt.show()
print("\n\n")
#plotting boxplot to look for outliers
plt.figure(figsize=(8, 6))
sns.boxplot(data=data, y='Rating')
plt.ylabel('Rating')
plt.title('Box Plot of App Ratings')
plt.show()
print("\n\n")

plt.figure(figsize=(8, 6))
sns.boxplot(data=data, y='Reviews')
plt.ylabel('Number of Reviews')
plt.title('Box Plot of App Reviews')
plt.show()

#Perform data cleaning and preprocessing if necessary. Handle missing
values, duplicate entries, or inconsistent data to ensure the quality of
your analysis.
data.dropna()
data.drop_duplicates()
data.describe()

#Extract insights from the dataset. For instance, you can determine which
app categories tend to have higher ratings, identify the most significant
factors influencing app popularity, or explore any interesting trends
within the data.
#Determine app categories with higher ratings
category_ratings =
data.groupby('Category')['Rating'].mean().sort_values(ascending=False)
print(category_ratings)

#counting free and paid apps
```

```
df = data.groupby("Type")
free_apps_count = data["Type"].value_counts()["Free"]
print("Number of Free Apps:", free_apps_count)

paid_apps_count = data["Type"].value_counts()["Paid"]
print("Number of Paid Apps:", paid_apps_count)

#Top 5 Apps
app_count = data['App'].value_counts()
app_count = app_count.sort_values(ascending = False)
app_count.head(5)

#Identify factors influencing app popularity
corelation = data.corr()
plt.figure(figsize=(4,4))
sns.heatmap(corelation , annot=True)
plt.show()

"""#summarising
1)App Categories with Top 3 Higher Ratings:

The average ratings for each app category were calculated, and the
categories with higher ratings were identified.
The top-rated app categories are as follows:
EDUCATION               4.387778
EVENTS                  4.363647
ART_AND_DESIGN          4.350462

2)Top 5 Catogeries
FAMILY      1972
GAME        1144
TOOLS        843
MEDICAL      463
BUSINESS     460

3)Type:
there are two type of apps paid and free
Number of Free Apps: 10039
Number of Paid Apps: 800
```

```
5)Top 5 Apps:
ROBLOX                                                  9
CBS Sports App - Scores, News, Stats & Watch Live       8
ESPN                                                    7
Duolingo: Learn Languages Free                          7
Candy Crush Saga
```

Task 2:

```
Task2
"""

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score
from keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Embedding, Dense, SimpleRNN, GRU, LSTM,
Bidirectional, Dropout
from sklearn.preprocessing import LabelEncoder
from keras.layers import Bidirectional


# Load the dataset
df = pd.read_csv("/content/urdu-sentiment-corpus-v1 (1).tsv",
delimiter='\t')
df.info()

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(df['Tweet'],
df['Class'], test_size=0.25, random_state=42)

# Tokenize the text data
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_train)
vocab_size = len(tokenizer.word_index) + 1

# Convert text data to sequences
```

```
X_train_seq = tokenizer.texts_to_sequences(X_train)
X_test_seq = tokenizer.texts_to_sequences(X_test)

# Pad sequences for equal length
max_length = max(len(seq) for seq in X_train_seq)
X_train_padded = pad_sequences(X_train_seq, maxlen=max_length,
padding='post')
X_test_padded = pad_sequences(X_test_seq, maxlen=max_length,
padding='post')

# Convert labels to binary format
label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(y_train)
y_test = label_encoder.transform(y_test)

# Define the models
models = [
    ('RNN', SimpleRNN),
    ('GRU', GRU),
    ('LSTM', LSTM),
    ('BiLSTM', lambda units: Bidirectional(LSTM(units,
return_sequences=True)))
]

# Define the hyperparameters
num_layers = [2, 3]
dropout_rates = [0.3, 0.7]

# Train and evaluate the models
results = []
for model_name, model_class in models:
    for num_layer in num_layers:
        for dropout_rate in dropout_rates:
            # Build the model
            model = Sequential()
            model.add(Embedding(vocab_size, 100, input_length=max_length))
            for _ in range(num_layer):
                if model_class == LSTM:
                    model.add(Bidirectional(model_class(100,
return_sequences=True)))
```

```
                else:
                    model.add(model_class(100, return_sequences=True))
                model.add(Dropout(dropout_rate))
            if model_class == LSTM:
                model.add(Bidirectional(model_class(100)))
            else:
                model.add(model_class(100))
            model.add(Dropout(dropout_rate))
            model.add(Dense(len(label_encoder.classes_),
activation='softmax'))

            # Compile and train the model
            model.compile(loss='sparse_categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])
            model.fit(X_train_padded, y_train, epochs=10, batch_size=64,
verbose=0)

            # Evaluate the model
            y_pred_probs = model.predict(X_test_padded)
            y_pred = y_pred_probs.argmax(axis=1)
            accuracy = accuracy_score(y_test, y_pred)
            precision = precision_score(y_test, y_pred,
average='weighted', zero_division=1)
            recall = recall_score(y_test, y_pred, average='weighted')
            f1 = f1_score(y_test, y_pred, average='weighted')

            # Store the results
            results.append([model_name, num_layer, dropout_rate, accuracy,
precision, recall, f1])
# Create a DataFrame for the results
columns = ['Model', 'Num Layers', 'Dropout Rate', 'Accuracy', 'Precision',
'Recall', 'F1 Score']
df_final = pd.DataFrame(results, columns=columns)

# Display the results
print(df_final)
```