Importing Libraries:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
from keras.models import Sequential
from keras.layers import Dense, Embedding, SimpleRNN, GRU, LSTM, Bidirectional
from keras.layers import Flatten, Dropout
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from tensorflow.keras.preprocessing import sequence
```

Importing Dataset:

```
file=pd.read_csv("googleplaystore.csv")
file.head()
```

App	Category	Rating	Reviews	Size	Installs	Type	Price	Conter Ratir
Photo Editor &								

Columns:

```
file.columns
```

```
Index(['App', 'Category', 'Rating', 'Reviews', 'Size', 'Installs', 'Type',
       'Price', 'Content Rating', 'Genres', 'Last Updated', 'Current Ver',
       'Android Ver'],
      dtype='object')
```

2	FREE Live	ART_AND_DESIGN	4.7	87510	8.7M	5,000,000+	Free	0 Everyor

```
file.shape
```

```
(10841, 13)
```

Checking Nan Values And Handling Them:

```
file.isnull().sum()
```

```
App                0
Category           0
Rating          1474
Reviews            0
Size               0
Installs           0
Type               1
Price              0
Content Rating     1
Genres             0
Last Updated       0
Current Ver        8
Android Ver        3
dtype: int64
```

```
mean_of_rating = file['Rating'].mean()
mode_of_type = file['Type'].mode().iloc[0]
```

```
mode_of_content_rating = file['Content Rating'].mode().iloc[0]
mode_of_android_version = file['Android Ver'].mode().iloc[0]
file['Rating'].fillna(mean_of_rating, inplace=True)
file['Type'].fillna(mode_of_type, inplace=True)
file['Content Rating'].fillna(mode_of_content_rating, inplace=True)
file['Android Ver'].fillna(mode_of_android_version, inplace=True)
```

```
file.isnull().sum()
```

```
App              0
Category         0
Rating           0
Reviews          0
Size             0
Installs         0
Type             0
Price            0
Content Rating   0
Genres           0
Last Updated     0
Current Ver      8
Android Ver      0
dtype: int64
```

```
file.shape
```

```
(10841, 13)
```

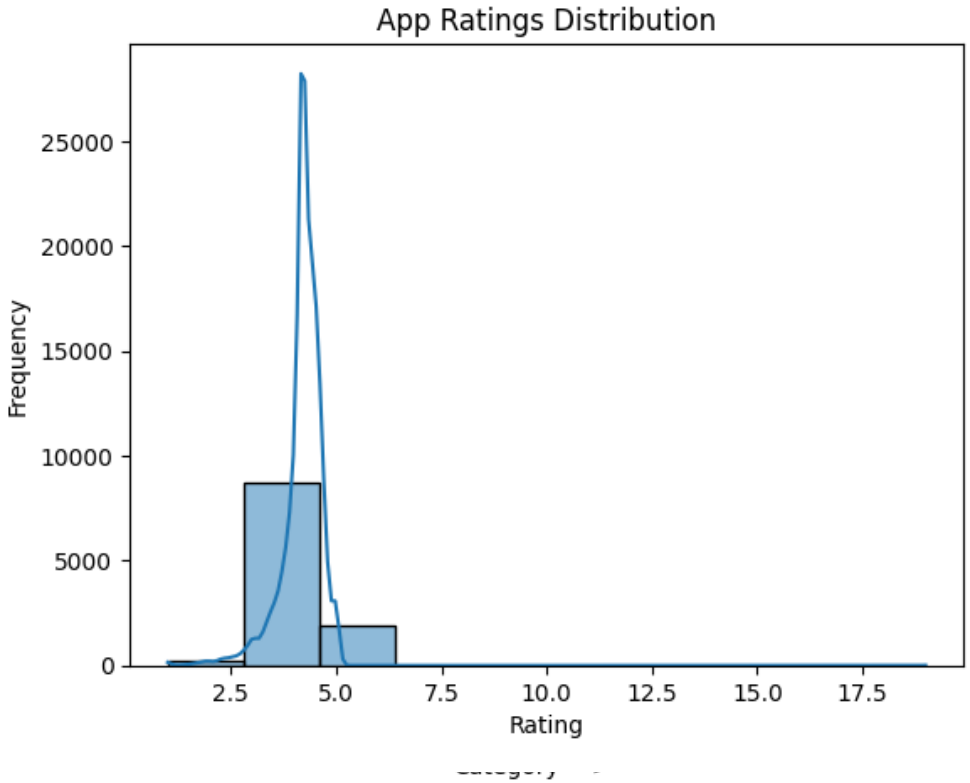
```
file.dtypes
```

```
App               object
Category          object
Rating           float64
Reviews           object
Size              object
Installs          object
Type              object
Price             object
Content Rating    object
Genres            object
Last Updated      object
Current Ver       object
Android Ver       object
dtype: object
```

Exploring the distribution of variables:

```
sns.countplot(x='Category', data=file)
plt.xticks(rotation=90)
plt.title('App Categories Distribution')
plt.xlabel('Category---->')
plt.ylabel('Count---->')
plt.show()
```

```
sns.histplot(file.Rating, bins=10, kde=True)
plt.title('App Ratings Distribution')
plt.xlabel('Rating')
plt.ylabel('Frequency')
plt.show()
```




```
sns.histplot(file.Reviews, bins=50, kde=True)
plt.title('App Reviews Distribution')
plt.xlabel('App Reviews')
plt.ylabel('Frequency')
plt.show()
```


```
for i in file.columns:
  print(file[i].describe())
```

```
count      10841
unique      9660
top       ROBLOX
freq           9
Name: App, dtype: object
count      10841
unique        34
top       FAMILY
freq        1972
Name: Category, dtype: object
count    10841.000000
mean         4.193338
std          0.499557
min          1.000000
25%          4.100000
50%          4.200000
75%          4.500000
max         19.000000
Name: Rating, dtype: float64
count     10841
unique     6002
top           0
freq        596
Name: Reviews, dtype: object
```

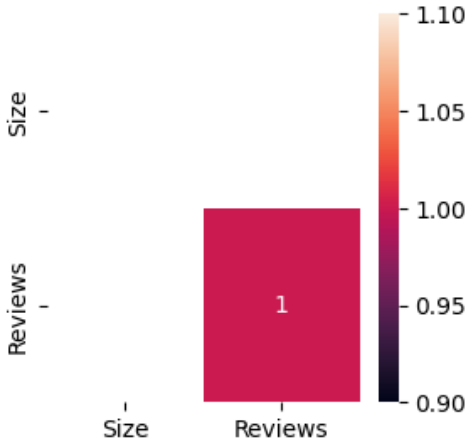
```
count                  10841
unique                   462
top       Varies with device
freq                    1695
Name: Size, dtype: object
count          10841
unique            22
top       1,000,000+
freq            1579
Name: Installs, dtype: object
count     10841
unique        3
top        Free
freq      10040
Name: Type, dtype: object
count     10841
unique       93
top           0
freq      10040
Name: Price, dtype: object
count        10841
unique           6
top       Everyone
freq          8715
Name: Content Rating, dtype: object
count     10841
unique      120
top       Tools
freq        842
Name: Genres, dtype: object
count              10841
unique              1378
top       August 3, 2018
freq                 326
```

```
corr = file[['Installs', 'Rating']].corr()
print(corr)
plt.figure(figsize=(3, 3))
sns.heatmap(corr,annot=True,cmap='coolwarm')
plt.show()
```

```
        Rating
Rating     1.0
<ipython-input-14-3765f7064c9a>:1: FutureWarning: The default value of numeric_only in
  corr = file[['Installs', 'Rating']].corr()
```

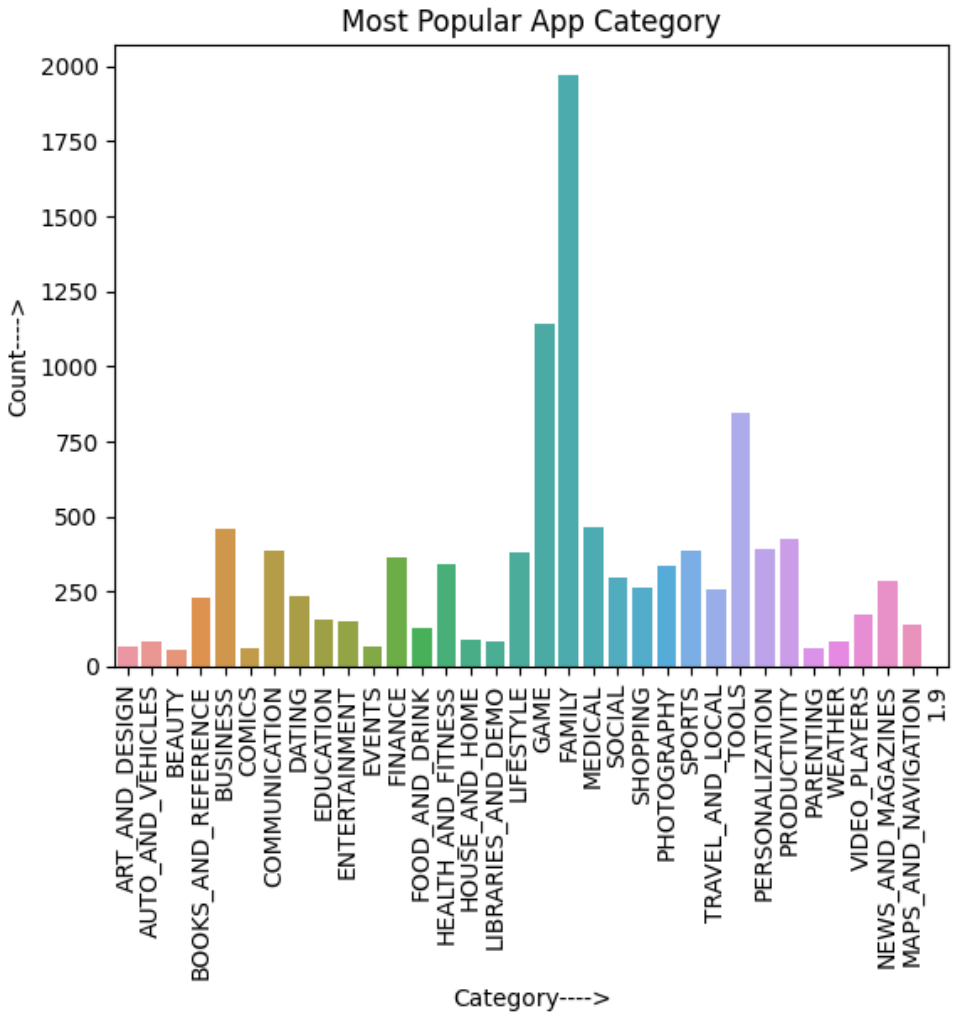

```
file['Size'] = pd.to_numeric(file['Size'], errors='coerce')
file['Reviews'] = pd.to_numeric(file['Reviews'], errors='coerce')
c=file[['Size',"Reviews"]].corr()
print(c)
plt.figure(figsize=(3, 3))
sns.heatmap(c,annot=True)
```

```
           Size  Reviews
Size        NaN      NaN
Reviews     NaN      1.0
<Axes: >
```

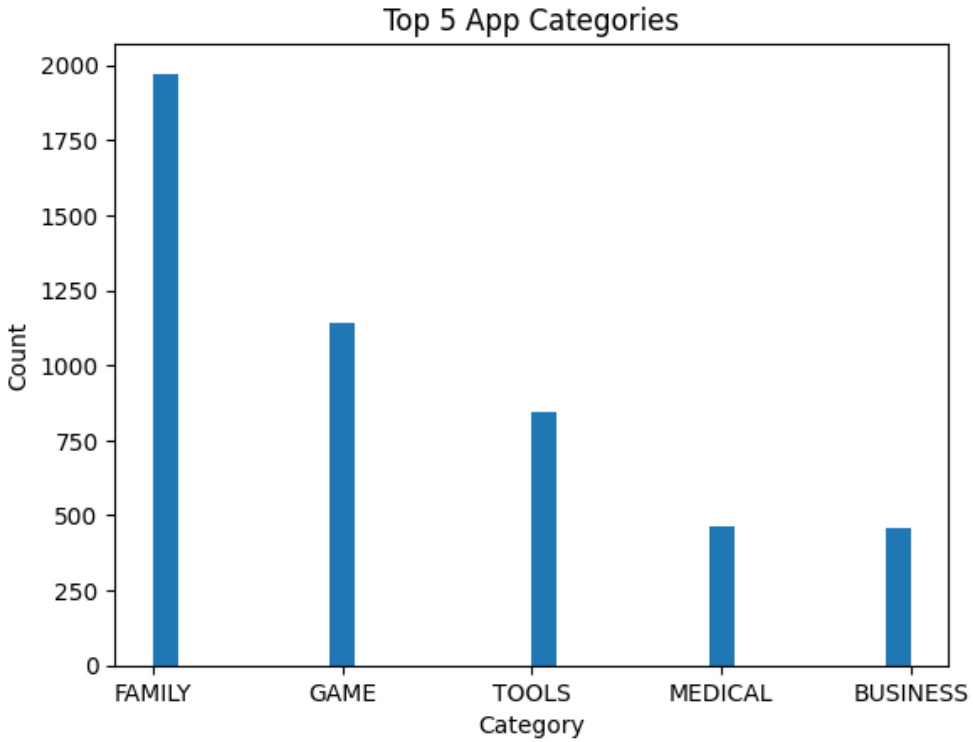



```
sns.countplot(x='Category', data=file)
plt.xticks(rotation=90)
plt.title('Most Popular App Category')
```

```
plt.xlabel('Category---->')
plt.ylabel('Count---->')
plt.show()
```

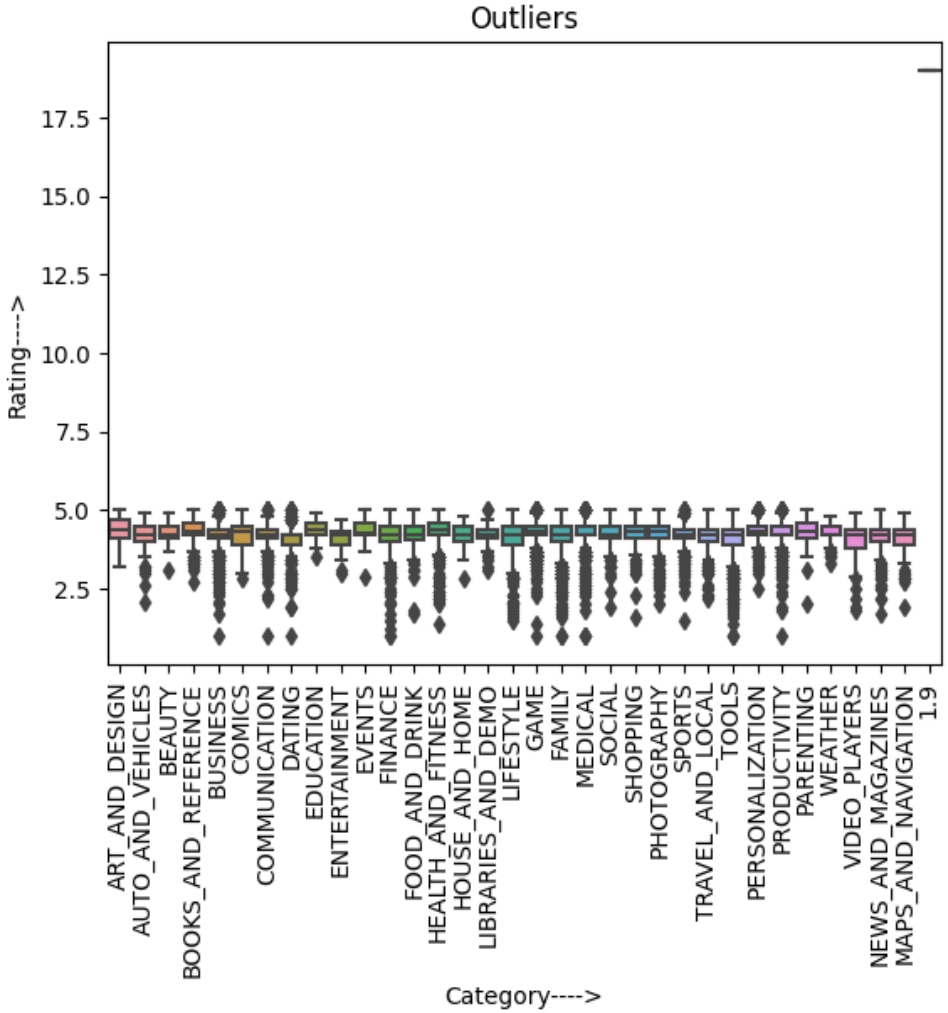



```
category_data = file['Category'].value_counts().head(5)
plt.hist(category_data.index, weights=category_data.values, bins=30)
plt.title('Top 5 App Categories')
plt.xlabel('Category')
plt.ylabel('Count')
plt.show()
```




```
columns_to_include = [i for i in file.columns if i != 'Size']
file[columns_to_include].describe()
```

```
sns.boxplot(x='Category', y="Rating",data=file)
plt.xticks(rotation=90)
plt.title('Outliers')
plt.xlabel('Category---->')
plt.ylabel('Rating---->')
plt.show()
```




```
file=file.drop_duplicates()
file.head()
```

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Conte Rati
0	Photo Editor & Candy Camera & Grid & ScrapBook	ART_AND_DESIGN	4.1	159.0	NaN	10,000+	Free	0	Everyo
1	Coloring book moana	ART_AND_DESIGN	3.9	967.0	NaN	500,000+	Free	0	Everyo
	U Launcher Lite –								

```
file.shape
```

```
(10358, 13)
```

```
def remove_outliers(data, threshold=3):
    z_score = stats.zscore(data)
    outlier_indices = np.abs(z_score) > threshold
    cleaned_data = data[~outlier_indices]
    return cleaned_data
```

```
remove_outliers(file.select_dtypes(include=['int','float']))
```

	Rating	Reviews	Size
0	4.100000	159.0	NaN
1	3.900000	967.0	NaN
2	4.700000	87510.0	NaN
3	4.500000	215644.0	NaN
4	4.300000	967.0	NaN
...	...	...	...
10836	4.500000	38.0	NaN
10837	5.000000	4.0	NaN
10838	4.193338	3.0	NaN
10839	4.500000	114.0	NaN
10840	4.500000	398307.0	NaN

10358 rows × 3 columns

```
def count_outliers(data, threshold):
    z_scores = stats.zscore(data)
    outliers_count = np.sum(np.abs(z_scores) > threshold)
    return outliers_count
```

```
count_outliers(file.select_dtypes(include=['int','float']),threshold=30)
#threshold should greater otherwise  we will have outliers in Rating column.
```

```
Rating     0
Reviews    0
Size       0
dtype: int64
```

```
file.groupby('Category')['Rating'].mean()
```

```
Category
1.9                    19.000000
ART_AND_DESIGN          4.350462
```

```
AUTO_AND_VEHICLES       4.190824
BEAUTY                  4.260882
BOOKS_AND_REFERENCE     4.311943
BUSINESS                4.135958
COMICS                  4.156445
COMMUNICATION           4.158216
DATING                  4.013538
EDUCATION               4.374564
ENTERTAINMENT           4.136036
EVENTS                  4.363647
FAMILY                  4.191406
FINANCE                 4.135315
FOOD_AND_DRINK          4.168388
GAME                    4.277598
HEALTH_AND_FITNESS      4.251656
HOUSE_AND_HOME          4.169001
LIBRARIES_AND_DEMO      4.181962
LIFESTYLE               4.113799
MAPS_AND_NAVIGATION     4.065061
MEDICAL                 4.185279
NEWS_AND_MAGAZINES      4.140784
PARENTING               4.282223
PERSONALIZATION         4.305620
PHOTOGRAPHY             4.183479
PRODUCTIVITY            4.200279
SHOPPING                4.245774
SOCIAL                  4.247001
SPORTS                  4.219279
TOOLS                   4.066280
TRAVEL_AND_LOCAL        4.107539
VIDEO_PLAYERS           4.074858
WEATHER                 4.239675
Name: Rating, dtype: float64
```

TASK#2:

```
df = pd.read_csv('urdu-sentiment-corpus-v1.tsv', delimiter='\t')
Tweets = df['Tweet'].tolist()
Classes  = df['Class'].tolist()
numFreqwords = 5000
maxSeqwords  =  500
tokenizer = Tokenizer(num_words=numFreqwords)
tokenizer.fit_on_texts(Tweets)
sequences = tokenizer.texts_to_sequences(Tweets)
x = pad_sequences(sequences, maxlen=maxSeqwords)
```

```
y = np.array(Classes)
np.random.seed(42)
indices = np.random.permutation(len(x))
x = x[indices]
y = y[indices]
y[y=='N']=0
y[y=='P']=1
y[y=='O']=0
y[y=='nan']=0
y= y.astype(int)
trainSize = int(0.75 * len(x))
x_train, x_test = x[:trainSize], x[trainSize:]
y_train, y_test = y[:trainSize], y[trainSize:]
diffLayers = [2, 3]
dropoutRates = [0.3, 0.7]
modelType = ['RNN', 'GRU', 'LSTM', 'BiLSTM']

def createModel(numOFlayers, dropoutRate, mod):
    model = Sequential()
    model.add(Embedding(5000, 32, input_length=maxSeqwords))
    for i in range(numOFlayers):
        if mod == 'RNN':      # Recurrent Neural Network
            model.add(SimpleRNN(32, return_sequences=True))
        elif mod == 'GRU':    # Gated Recurrent Unit
            model.add(GRU(32, return_sequences=True))
        elif mod == 'LSTM':   # Long short-term memory
            model.add(LSTM(32, return_sequences=True))
        elif mod == 'BiLSTM': # Bidirectional Long Short-Term Memory
            model.add(Bidirectional(LSTM(32, return_sequences=True)))

        model.add(Dropout(dropoutRate))

    model.add(Flatten())                        #Adding flatten layer
    model.add(Dense(1, activation='sigmoid'))   #Adding output layer

    # Compile the model
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

    return model


results = []
for layers in diffLayers:
    for rate in dropoutRates:
        for models in modelType:
```

```
        for models in model_type:

            # Creating and compiling the model by calling the function above called createModel
            model = createModel(layers, rate, models)

            # Training the model
            #Keeping the epochs set to 3
            model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=3, batch_size=64)

            # Making predictions:
            y_pred_probs = model.predict(x_test)
            y_pred = np.round(y_pred_probs).flatten().astype(int)

            # Calculating evaluation metrics according to ones instructed in question:
            accuracy = accuracy_score(y_test, y_pred)
            precision = precision_score(y_test, y_pred)
            recall = recall_score(y_test, y_pred)
            f1 = f1_score(y_test, y_pred)

            # Saving the results by appending dictionary type results for each iteration
            results.append({
                      'Layers': layers,
                      'Dropout Rate': rate,
                      'Model Type': models,
                      'Accuracy': accuracy,
                      'Precision': precision,
                      'Recall': recall,
                      'F1-score': f1
                           })



# Printing the results in table form as instructed in question
print("\nResults after testing different parameters are as follows:")
print("Layers\tDropout Rate\tModel Type\tAccuracy\tPrecision\tRecall\t\tF1-score")
for result in results:
    print(f"{result['Layers']}\t{result['Dropout Rate']}\t\t{result['Model Type']}\t\t{result['Accuracy']:.4f}\t\t{result['Precision']:.4f}\t\t{result['Recall'
```

3	0.7	BiLSTM	0.4240	0.4240	1.0000	0.5955