Daily Website Visitors RNN Project

Import Packages and Data

https://www.kaggle.com/datasets/bobnau/daily-website-visitors

In [1]:
```
import os
os.getcwd()

%cd "C:\Users\Angela\OneDrive\Desktop\ANA500"
```

```
C:\Users\Angela\OneDrive\Desktop\ANA500
```

In [91]:
```
#import packages
import numpy as np
import pandas as pd

import seaborn as sns
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, SimpleRNN, Bidirectional, Dropout
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
```

In [148…
```
#load dataset
df = pd.read_csv('daily-website-visitors.csv')
```

Preprocessing Data

In [149…
```
#Review data type of variables
df.dtypes
```

Out[149]:
```
Row                 int64
Day                object
Day.Of.Week         int64
Date               object
Page.Loads         object
Unique.Visits      object
First.Time.Visits  object
Returning.Visits   object
dtype: object
```

The target variable will be Unique.Visits. This variable has a type of object.

In [150…
```
#Review size of data
df.shape
```

Out[150]:
```
(2167, 8)
```

The dataset has 2,167 rows and 8 columns.

In [151…
```
# check if there are null values
df.isnull().sum()
```

Out[151]:
```
Row                0
Day                0
Day.Of.Week        0
Date               0
Page.Loads         0
Unique.Visits      0
First.Time.Visits  0
Returning.Visits   0
dtype: int64
```

There are no missing values in this dataset.

In [152…
```
# Review the values of columns
df.head()
```

Out[152]:

	Row	Day	Day.Of.Week	Date	Page.Loads	Unique.Visits	First.Time.Visits	Returning.Visits
0	1	Sunday	1	9/14/2014	2,146	1,582	1,430	152
1	2	Monday	2	9/15/2014	3,621	2,528	2,297	231
2	3	Tuesday	3	9/16/2014	3,698	2,630	2,352	278
3	4	Wednesday	4	9/17/2014	3,667	2,614	2,327	287
4	5	Thursday	5	9/18/2014	3,316	2,366	2,130	236

Page.Loads, Unique.Visits and First.Time.Visits all have string values with commas.

In [153…
```
#Remove commas
df['Page.Loads'] = df['Page.Loads'].str.replace(',', '')
df['Unique.Visits'] = df['Unique.Visits'].str.replace(',', '')
df['First.Time.Visits'] = df['First.Time.Visits'].str.replace(',', '')
df['Returning.Visits'] = df['Returning.Visits'].str.replace(',', '')
```

In [154…
```
# Check that commas are removed
df.head()
```

Out[154]:

	Row	Day	Day.Of.Week	Date	Page.Loads	Unique.Visits	First.Time.Visits	Returning.Visits
0	1	Sunday	1	9/14/2014	2146	1582	1430	152
1	2	Monday	2	9/15/2014	3621	2528	2297	231
2	3	Tuesday	3	9/16/2014	3698	2630	2352	278
3	4	Wednesday	4	9/17/2014	3667	2614	2327	287
4	5	Thursday	5	9/18/2014	3316	2366	2130	236

In [155…
```
df['Page.Loads'] = df['Page.Loads'].astype('float32')
df['Unique.Visits'] = df['Unique.Visits'].astype('float32')
df['First.Time.Visits'] = df['First.Time.Visits'].astype('float32')
df['Returning.Visits'] = df['Returning.Visits'].astype('float32')
```

In [156…
```
# Check that data types are changed
df.dtypes
```

Out[156]:
```
Row                   int64
Day                  object
Day.Of.Week           int64
Date                 object
Page.Loads          float32
Unique.Visits       float32
First.Time.Visits   float32
Returning.Visits    float32
dtype: object
```

The commas were removed from Page.Loads, Unique.Visits, and First.Tiime.Visits and Returning.Visits and the datatype was changed to Float32 so we can use as numeric values.

In [87]:
```
#Change date datatype to date
df['Date'] = pd.to_datetime(df['Date'])
```

In [88]:
```
# Check that data type changed
df.dtypes
```

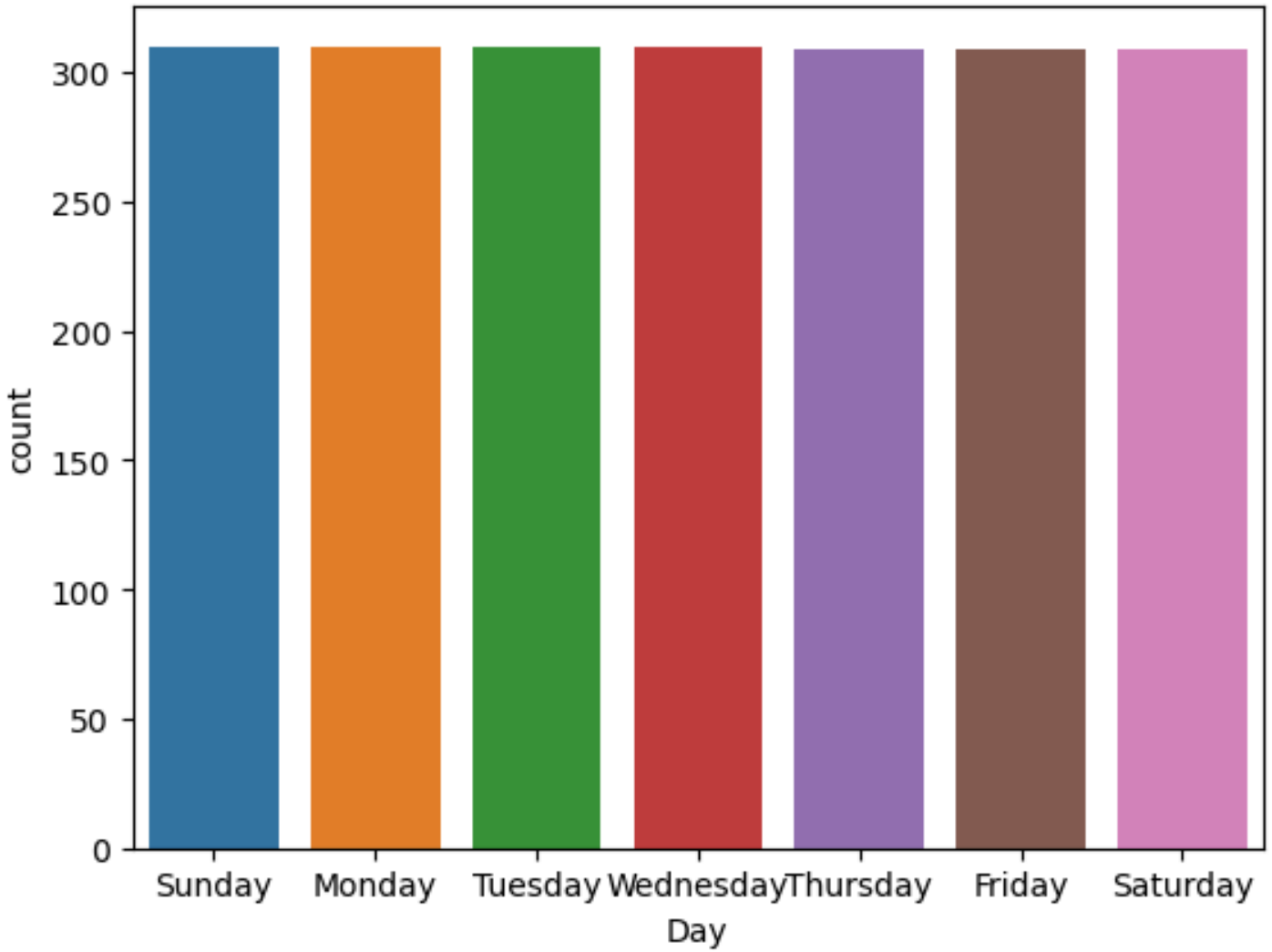
Out[88]:
```
Row                          int64
Day                         object
Day.Of.Week                  int64
Date                datetime64[ns]
Page.Loads                 float32
Unique.Visits              float32
First.Time.Visits          float32
Returning.Visits           float32
dtype: object
```

Date was changed from object to datetime for easier processing of the date values.

Data Visualizations

In [14]:
```
# Count Bar Chart of Day
sns.countplot(df, x = "Day")
```

Out[14]:
```
<Axes: xlabel='Day', ylabel='count'>
```



Day is a categorical variable. Day holds values for all 7 days of the week; Sunday, Monday, Tuesday, Wednesday, Thursday, Friday and Saturday. The count for each of the days is a little over 300 visitors each. The counts for each day are evenly distributed.

In [16]:
```
#Day of the week Histogram

sns.histplot(df['Day.Of.Week'], bins=30, color='pink', edgecolor='purple')
plt.title('Day of Week')
```

Out[16]:
```
Text(0.5, 1.0, 'Day of Week')
```

Day.Of.Week is an integer variable. Day.Of.Week holds values for each numerical representation of the day of the week. The week starts with Sunday which is represented as 1. The histogram is an even distribution.

In [89]:
```
# Statistics on Date
df['Date'].describe()
```

Out[89]:
```
count                    2167
mean      2017-09-01 00:00:00
min       2014-09-14 00:00:00
25%       2016-03-08 12:00:00
50%       2017-09-01 00:00:00
75%       2019-02-24 12:00:00
max       2020-08-19 00:00:00
Name: Date, dtype: object
```

In [90]:
```
# Histogram of Unique.Visits
sns.histplot(df['Date'], bins=30, color='pink', edgecolor='purple')
plt.title('Date')
```

Out[90]:
```
Text(0.5, 1.0, 'Date')
```

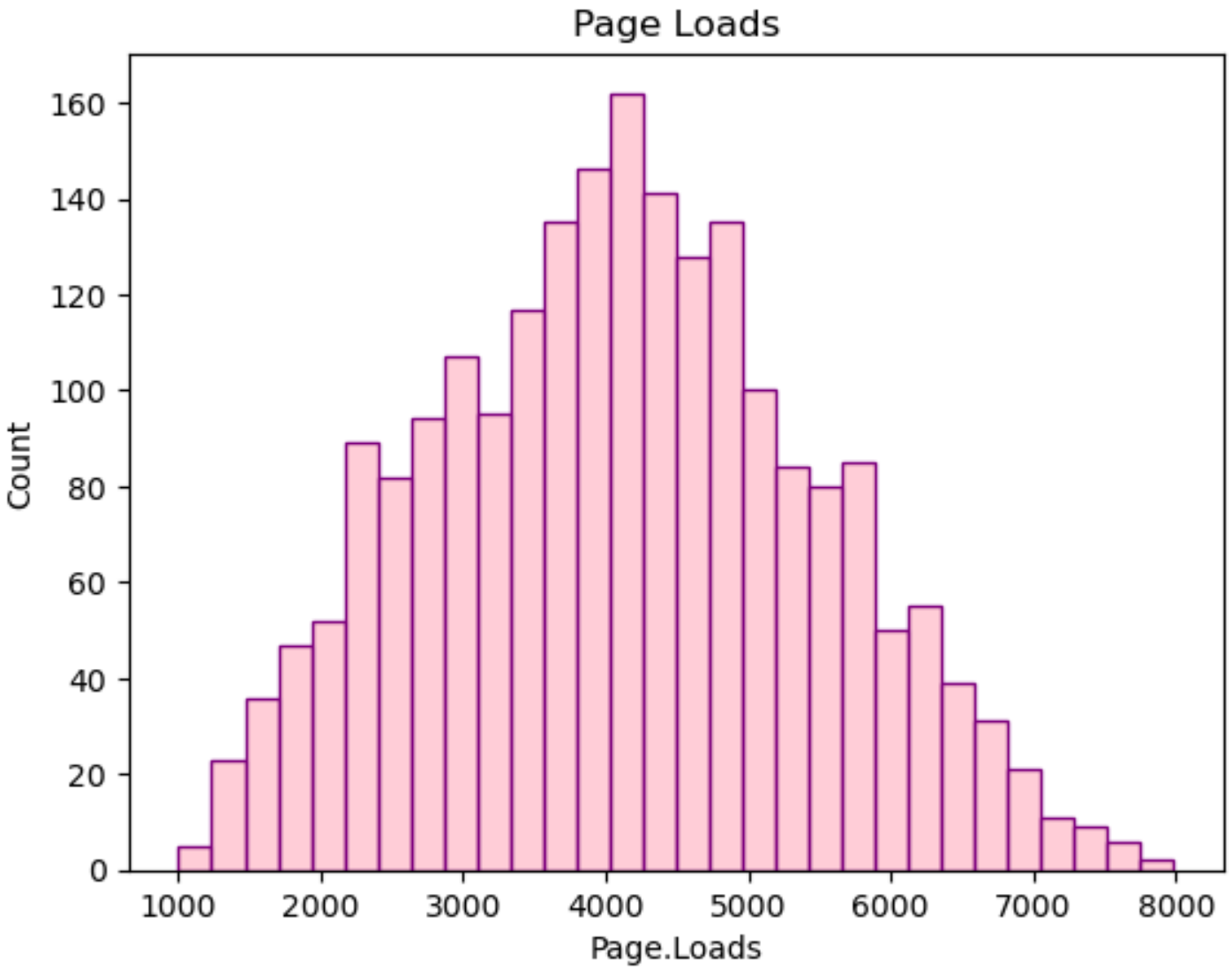
Date is a datetime variable. The dates range from September 14, 2014 to August 19, 2020. Date has an even distribution.

In [57]:
```
# Statistics of Page Loads
df['Page.Loads'].describe()
```

Out[57]:
```
count    2167.000000
mean     4116.989258
std      1350.977539
min      1002.000000
25%      3114.500000
50%      4106.000000
75%      5020.500000
max      7984.000000
Name: Page.Loads, dtype: float64
```

In [56]:
```
#Histogram of Page Loads
sns.histplot(df['Page.Loads'], bins=30, color='pink', edgecolor='purple')
plt.title('Page Loads')
```

Out[56]:
```
Text(0.5, 1.0, 'Page Loads')
```



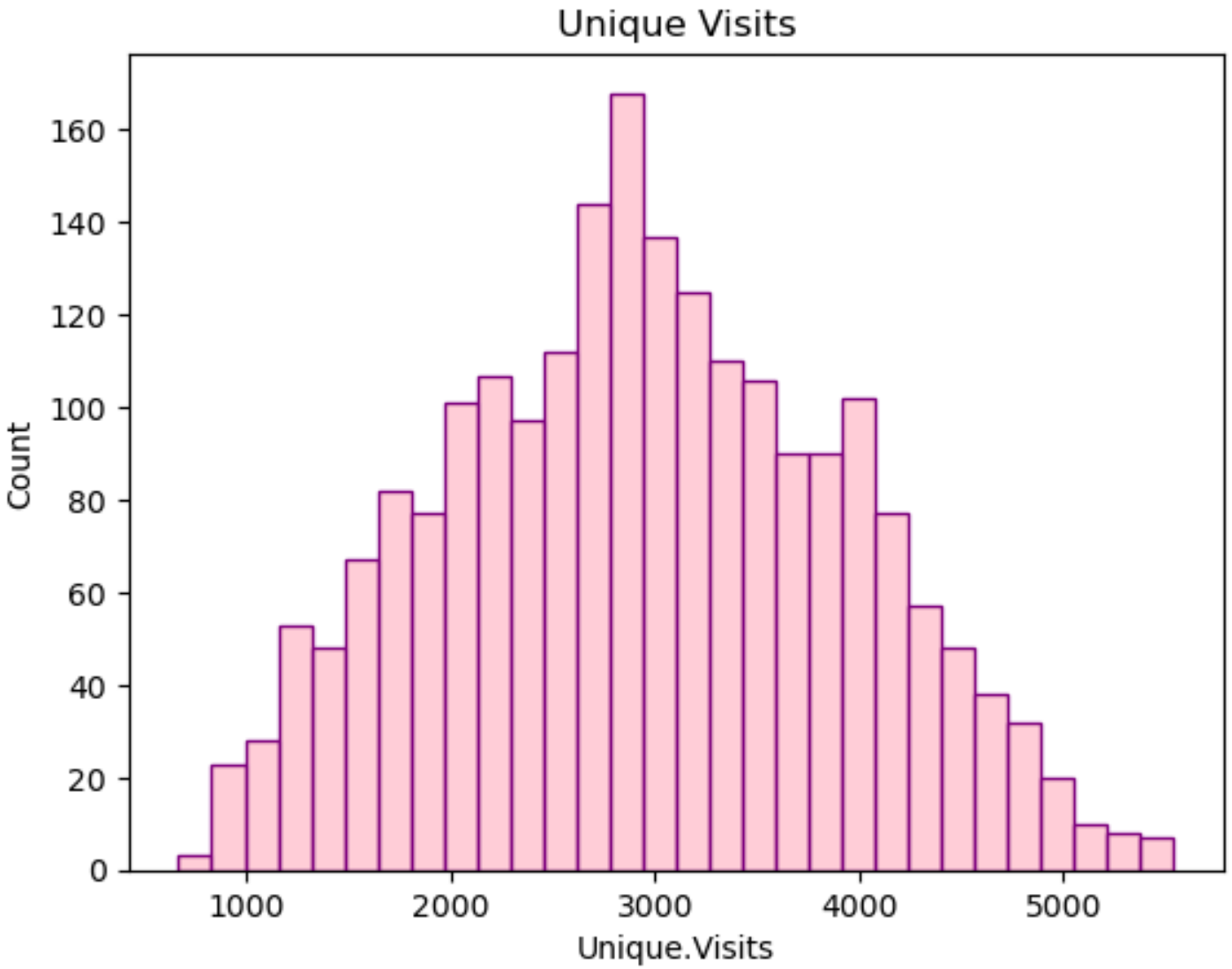
Page.Loads has a Normal Distribution. The Values range from 1,002 to 7.984. The highest count of visitors is 160 which correlates to about 4,000 page loads.

In [58]:
```
# Statistics of Unique Visits
df['Unique.Visits'].describe()
```

Out[58]:
```
count    2167.000000
mean     2943.646484
std       977.886597
min       667.000000
25%      2226.000000
50%      2914.000000
75%      3667.500000
max      5541.000000
Name: Unique.Visits, dtype: float64
```

In [59]:
```
# Histogram of Unique.Visits
sns.histplot(df['Unique.Visits'], bins=30, color='pink', edgecolor='purple')
plt.title('Unique Visits')
```

Out[59]:
```
Text(0.5, 1.0, 'Unique Visits')
```



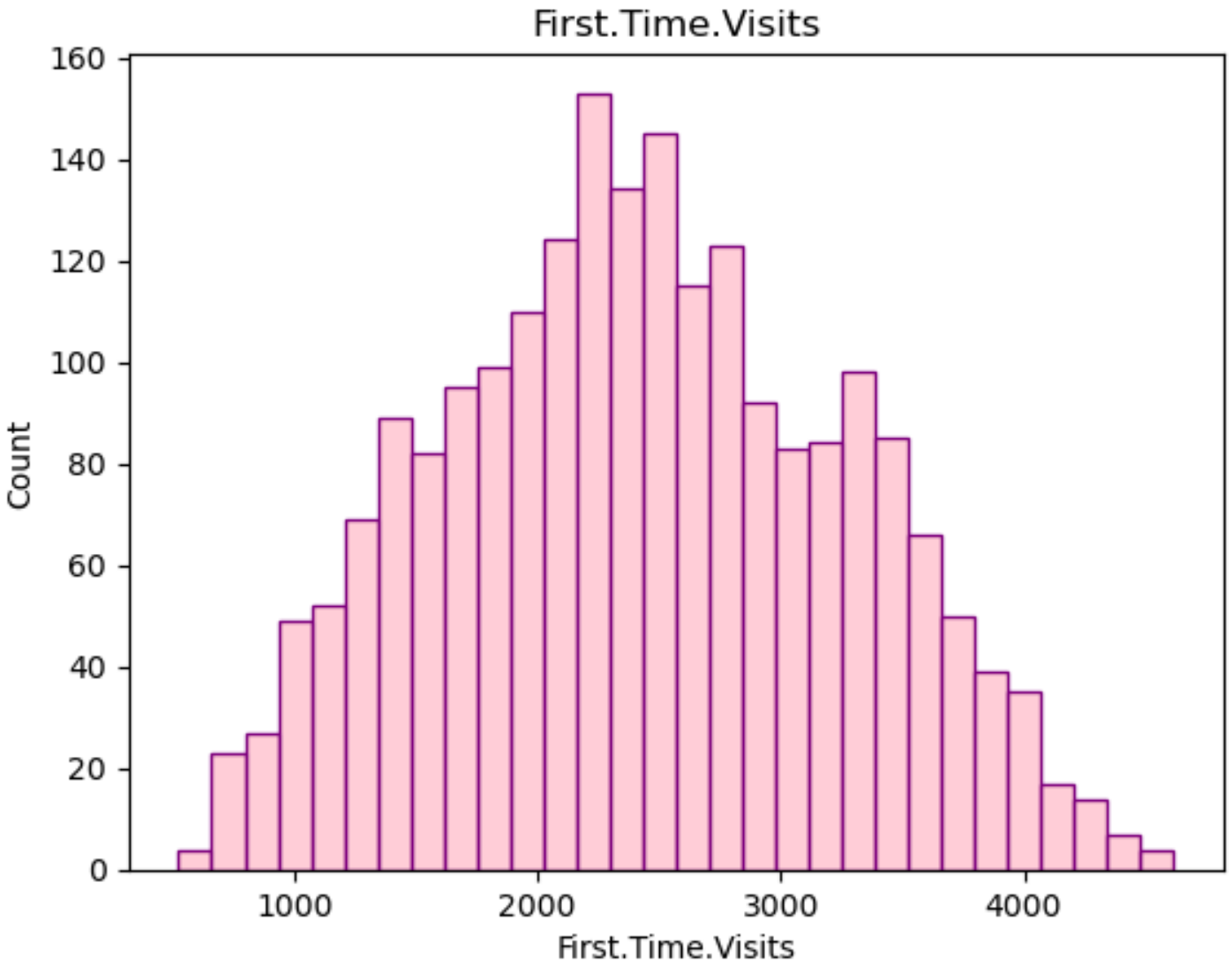
Unique.Visits has a normal distribution. The values range from 667 to 5,541. The highest count of visitors is 160 which accounts for about 2,900 unique visits.

In [83]:
```
# Statistics of First.Time.Visits
df['First.Time.Visits'].describe()
```

Out[83]:
```
count    2167.000000
mean     2431.824219
std       828.704407
min       522.000000
25%      1830.000000
50%      2400.000000
75%      3038.000000
max      4616.000000
Name: First.Time.Visits, dtype: float64
```

In [84]:
```
# Histogram of Unique.Visits
sns.histplot(df['First.Time.Visits'], bins=30, color='pink', edgecolor='purple')
plt.title('First.Time.Visits')
```

Out[84]:
```
Text(0.5, 1.0, 'First.Time.Visits')
```



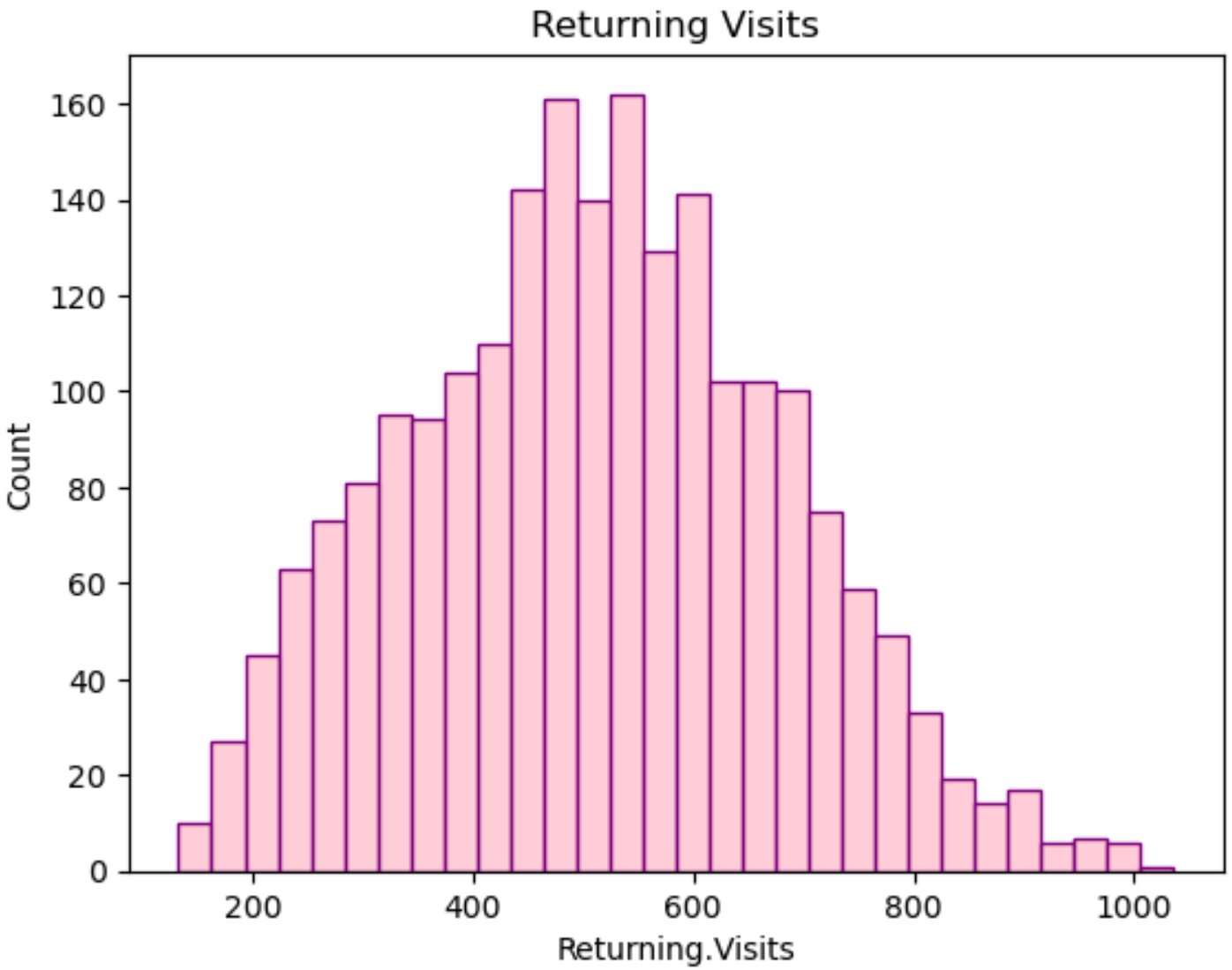
First.Time.Visits has a normal distribution. The values range from 522 to 4,616. The highest count is about 2,400 first time visitors.

In [85]:
```
# Statistics of First.Time.Visits
df['Returning.Visits'].describe()
```

Out[85]:
```
count    2167.000000
mean      511.822327
std       168.736359
min       133.000000
25%       388.500000
50%       509.000000
75%       626.500000
max      1036.000000
Name: Returning.Visits, dtype: float64
```

In [86]:
```
# Histogram of Unique.Visits
sns.histplot(df['Returning.Visits'], bins=30, color='pink', edgecolor='purple')
plt.title('Returning Visits')
```

Out[86]:
```
Text(0.5, 1.0, 'Returning Visits')
```



Returning.Visits has a normal distribution. The values range from 133 to 1,036. The highest count of returning visits is 509.

RNN - LSTM Model

In [192…

```
# Set target variable (X) and features (y)
y = df[['Unique.Visits']]
```

```
X = df.drop(['Unique.Visits', 'Day', 'Date'], axis = 1)
time_step = 10
```

In [193…
```
#Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```

In [194…
```
#scale the data
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [195…
```
#reshape input to be samples timesteps features, which is required for LSTM
X_train = X_train.values.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.values.reshape(X_test.shape[0], X_test.shape[1], 1)
```

In [196…
```
#Create Model
model = Sequential()

model.add(LSTM(64, activation = 'tanh', return_sequences=True, input_shape = (time_step, 1)))
model.add(LSTM(32, activation = 'tanh', return_sequences=True))
model.add(LSTM(16, activation = 'tanh'))
model.add(Dense(1))

#Compile Model
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

In [197…
```
#Summarize Model
model.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_23 (LSTM)              (None, 10, 64)            16896

 lstm_24 (LSTM)              (None, 10, 32)            12416

 lstm_25 (LSTM)              (None, 16)                3136

 dense_7 (Dense)             (None, 1)                 17

=================================================================
Total params: 32465 (126.82 KB)
Trainable params: 32465 (126.82 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [198…

```python
#Train the model

train_data = tf.data.Dataset.from_tensor_slices((X_train, y_train))
valid_data = tf.data.Dataset.from_tensor_slices((X_test, y_test))

history = model.fit(train_data, epochs=10, validation_data=valid_data)
```

```
Epoch 1/10
1733/1733 [==============================] - 18s 8ms/step - loss: 9441983.0000 - val_loss: 9806939.0
000
Epoch 2/10
1733/1733 [==============================] - 10s 6ms/step - loss: 9312234.0000 - val_loss: 9678427.0
000
Epoch 3/10
1733/1733 [==============================] - 13s 7ms/step - loss: 9187613.0000 - val_loss: 9551601.0
000
Epoch 4/10
1733/1733 [==============================] - 12s 7ms/step - loss: 9064316.0000 - val_loss: 9425915.0
000
Epoch 5/10
1733/1733 [==============================] - 14s 8ms/step - loss: 8942115.0000 - val_loss: 9301279.0
000
Epoch 6/10
1733/1733 [==============================] - 11s 6ms/step - loss: 8820950.0000 - val_loss: 9177680.0
000
Epoch 7/10
1733/1733 [==============================] - 12s 7ms/step - loss: 8700804.0000 - val_loss: 9055110.0
000
Epoch 8/10
1733/1733 [==============================] - 11s 6ms/step - loss: 8581689.0000 - val_loss: 8933565.0
000
Epoch 9/10
1733/1733 [==============================] - 13s 8ms/step - loss: 8463612.0000 - val_loss: 8813054.0
000
Epoch 10/10
1733/1733 [==============================] - 12s 7ms/step - loss: 8346553.5000 - val_loss: 8693562.0
000
```

Evaluate Model

In [201…

```
# Make Predictions
train_pred = model.predict(X_train)
test_pred = model.predict(X_test)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[201], line 2
      1 # Make Predictions
----> 2 train_pred = model.predict(X_train)
      3 test_pred = model.predict(X_test)

File ~\anaconda3\Lib\site-packages\keras\src\utils\traceback_utils.py:70, in filter_traceback.<locals>.error_handler(*args, **kwargs)
     67     filtered_tb = _process_traceback_frames(e.__traceback__)
     68     # To get the full stack trace, call:
     69     # `tf.debugging.disable_traceback_filtering()`
---> 70     raise e.with_traceback(filtered_tb) from None
     71 finally:
     72     del filtered_tb

File ~\AppData\Local\Temp\__autograph_generated_filet_n2l0zh.py:15, in outer_factory.<locals>.inner_factory.<locals>.tf__predict_function(iterator)
     13 try:
     14     do_return = True
---> 15     retval_ = ag__.converted_call(ag__.ld(step_function), (ag__.ld(self), ag__.ld(iterator)), None, fscope)
     16 except:
     17     do_return = False

ValueError: in user code:

    File "C:\Users\Angela\anaconda3\Lib\site-packages\keras\src\engine\training.py", line 2341, in predict_function  *
        return step_function(self, iterator)
    File "C:\Users\Angela\anaconda3\Lib\site-packages\keras\src\engine\training.py", line 2327, in step_function  **
        outputs = model.distribute_strategy.run(run_step, args=(data,))
    File "C:\Users\Angela\anaconda3\Lib\site-packages\keras\src\engine\training.py", line 2315, in run_step  **
        outputs = model.predict_step(data)
    File "C:\Users\Angela\anaconda3\Lib\site-packages\keras\src\engine\training.py", line 2283, in predict_step
        return self(x, training=False)
```

```
    File "C:\Users\Angela\anaconda3\Lib\site-packages\keras\src\utils\traceback_utils.py", line 70,
in error_handler
        raise e.with_traceback(filtered_tb) from None
    File "C:\Users\Angela\anaconda3\Lib\site-packages\keras\src\engine\input_spec.py", line 298, in
assert_input_compatibility
        raise ValueError(

    ValueError: Input 0 of layer "sequential_9" is incompatible with the layer: expected shape=(Non
e, 10, 1), found shape=(None, 5, 1)
```

In [ ]: