```
!pip install -U imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in c:\users\pditi\anaconda3\lib\site-packages (0.11.0)
Collecting imbalanced-learn
  Downloading imbalanced_learn-0.12.0-py3-none-any.whl (257 kB)
     -------------------------------------- 257.7/257.7 kB 1.1 MB/s eta 0:00:00
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\pditi\anaconda3\lib\site-packages (from imbalanced-learn) (2.2.0)
Requirement already satisfied: scipy>=1.5.0 in c:\users\pditi\anaconda3\lib\site-packages (from imbalanced-learn) (1.9.1)
Requirement already satisfied: numpy>=1.17.3 in c:\users\pditi\anaconda3\lib\site-packages (from imbalanced-learn) (1.23.5)
Requirement already satisfied: joblib>=1.1.1 in c:\users\pditi\anaconda3\lib\site-packages (from imbalanced-learn) (1.3.2)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\pditi\anaconda3\lib\site-packages (from imbalanced-learn) (1.3.2)
Installing collected packages: imbalanced-learn
  Attempting uninstall: imbalanced-learn
    Found existing installation: imbalanced-learn 0.11.0
    Uninstalling imbalanced-learn-0.11.0:
      Successfully uninstalled imbalanced-learn-0.11.0
Successfully installed imbalanced-learn-0.12.0
WARNING: Ignoring invalid distribution -rotobuf (c:\users\pditi\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\pditi\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\pditi\anaconda3\lib\site-packages)
    WARNING: Ignoring invalid distribution -rotobuf (c:\users\pditi\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\pditi\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\pditi\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\pditi\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\pditi\anaconda3\lib\site-packages)
```

```
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
```

```
import os
import plotly.express as px
import plotly.graph_objects as go
from contextlib import contextmanager
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from statsmodels.tsa.seasonal import seasonal_decompose
```

```
df=pd.read_csv('thane_ticket_6months_data1.csv')
```

```
df.head()
```

```
   Unnamed: 0           d_name            s_name        date timeslot1  day  \
0           0  Aai Mata Mandir  Chendani Koliwada  2023-01-26    5 to 6    3
1           1  Aai Mata Mandir  Chendani Koliwada  2023-02-19  23 to 24    6
2           2  Aai Mata Mandir  Chendani Koliwada  2023-03-05  23 to 24    6
3           3  Aai Mata Mandir  Chendani Koliwada  2023-03-09  23 to 24    3
4           4  Aai Mata Mandir  Chendani Koliwada  2023-03-20   9 to 10    0

   holiday  month  count
0        1      1      1
1        1      2      2
2        0      3      1
3        0      3      1
4        0      3      6
```

```
total_unique_destinations=df['d_name'].nunique()
total_unique_destinations
destination_names=df['d_name'].unique().tolist()
destination_names
```

```
 'Range Office',
 'Reti Bunder',
 'Reti Bunder Circle',
 'Runwal Garden City',
 'Sahakar Nagar',
 'Sahyadri Society',
 'Saibaba Mandir',
 'Saket Complex',
 'Saket Phata',
 'Sambhaji Nagar',
 'Sasunavghar',
 'Sathe Nagar',
 'Sativali Naka',
 'Sativali Phata',
 'Shashkiy Vishram Gruh Versova',
 'Shastri Nagar',
 'Shil Phata',
 'Shivai Nagar',
 'Shri Ram Hospital',
 'Silver Park',
 'Surai Phata',
 'Teen Hath Naka',
 'Tembhi Naka',
 'Thane Station East Kopri',
 'Thane Station West SATIS',
 'Vandana Cinema',
 'Vanjarpatti Flyover',
 'Vasant Nagari',
 'Vehele Panchayat',
 'Vikhroli Police Station',
 'Virwani Estate /Sarvoday Nagar',
 'Voltas Gate',
 'Vrindavan Society',
 'Waghbil Gaon',
 'Wagle Bus Depot',
 'Wagle Circle',
 'Western Hotel / Lakshmi Baug',
 'Yashodhan Nagar',
 'Yeoor / Patonapada']
```

```
df['s_name'].nunique()
```

```
152
```

out of total days how many of them are holidays and not holidays ?

```
total_days=df['day'].count()
total_days
```

```
2441248
```

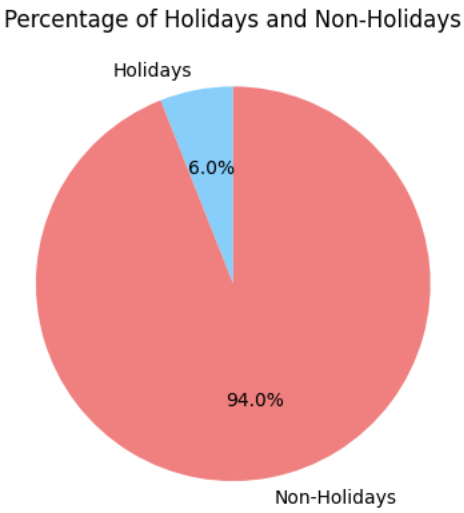
```
total_holidays=df[df['holiday']==1]['day'].count()
total_no_holidays=total_days - total_holidays

print(total_holidays)
print(total_no_holidays)
```

```
156055
2285193
```

```
holidays_percent=((total_holidays/total_days)*100).round()
non_holidays_percent=((total_no_holidays/total_days)*100).round()
print("holidays percent: ", holidays_percent)
print("Non Holidays Percent: " , non_holidays_percent)
```

```
holidays percent:  6.0
Non Holidays Percent:  94.0
```

```
labels=['Holidays' , 'Non-Holidays']
sizes=[holidays_percent , non_holidays_percent]
colors=['lightskyblue', 'lightcoral']

plt.pie(sizes, labels=labels, autopct='%1.1f%%', colors=colors, startangle=90)
plt.title('Percentage of Holidays and Non-Holidays')
plt.show()
```



lowest count of ticket and highest count of ticket generated in a day

```
lowest_ticket_count = df['count'].min()
highest_ticket_count = df['count'].max()
print("Lowest count of ticket:", lowest_ticket_count)
print("Highest count of ticket:", highest_ticket_count)
```

```
Lowest count of ticket: 1
Highest count of ticket: 1103
```

Highest number of tickets generated in which time slot and lowest count of tickets generated in which time slot

```
highest_ticket_timeslot = df.loc[df['count'].idxmax()]['timeslot1']
lowest_ticket_timeslot = df.loc[df['count'].idxmin()]['timeslot1']
print("4) Highest number of tickets generated in time slot:", highest_ticket_timeslot)
print("   Lowest count of tickets generated in time slot:", lowest_ticket_timeslot)
```

```
4) Highest number of tickets generated in time slot: 9 to 10
   Lowest count of tickets generated in time slot: 5 to 6
```

```
class_distribution=df['count'].value_counts()
class_distribution
```

```
1      650456
2      372660
3      245056
4      176023
5      131391
        ...
679         1
996         1
671         1
667         1
641         1
Name: count, Length: 847, dtype: int64
```

```
df.columns.to_list()
```

```
['Unnamed: 0',
 'd_name',
 's_name',
 'date',
 'timeslot1',
 'day',
 'holiday',
 'month',
 'count']
```

```
df['month'] = pd.Series(df['month'])
```

```
df['month'] = pd.to_numeric(df['month'], errors='coerce')
```

```
df.shape
```

```
(2441248, 9)
```

```
df= df.drop('Unnamed: 0' , axis = 1)
```

```
df.shape
```

```
(2441248, 8)
```

```
df.tail()
```

```
                    d_name                     s_name        date timeslot1  \
2441243  Yeoor / Patonapada  Thane Station West SATIS  2023-06-23  23 to 24
2441244  Yeoor / Patonapada  Thane Station West SATIS  2023-06-23    6 to 7
2441245  Yeoor / Patonapada  Thane Station West SATIS  2023-06-23    7 to 8
2441246  Yeoor / Patonapada  Thane Station West SATIS  2023-06-23    8 to 9
2441247  Yeoor / Patonapada  Thane Station West SATIS  2023-06-23   9 to 10

         day  holiday  month  count
2441243    4        0      6      2
2441244    4        0      6      9
2441245    4        0      6      5
2441246    4        0      6     27
2441247    4        0      6     45
```

```
type(df)
```

```
pandas.core.frame.DataFrame
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2441248 entries, 0 to 2441247
Data columns (total 8 columns):
 #   Column     Dtype
---  ------     -----
 0   d_name     object
 1   s_name     object
 2   date       object
 3   timeslot1  object
 4   day        int64
 5   holiday    int64
 6   month      int64
 7   count      int64
dtypes: int64(4), object(4)
memory usage: 149.0+ MB
```

```
df.describe().round()
```

```
             day    holiday      month      count
count  2441248.0  2441248.0  2441248.0  2441248.0
mean         3.0        0.0        3.0       12.0
std          2.0        0.0        2.0       31.0
min          0.0        0.0        1.0        1.0
25%          1.0        0.0        2.0        1.0
50%          3.0        0.0        3.0        3.0
75%          5.0        0.0        5.0        9.0
max          6.0        1.0        6.0     1103.0
```

```
from scipy.stats import skew, kurtosis

skewness = skew(df['count'])
kurt = kurtosis(df['count'])

print(f"Skewness: {skewness}")
print(f"Kurtosis: {kurt}")
```

```
Skewness: 8.819071301973787
Kurtosis: 129.80190695654275
```

The skewness value of 8.819 indicates a highly positively skewed distribution, meaning that the data is concentrated on the left side with a long right tail. The kurtosis value of 129.801 indicates a very high peak, suggesting heavy tails or outliers in the distribution.

our target variable seems to be imbalanced so applying SMOTE (Synthetic Minority Over Sampling Technique) for creating synthetic samples for minority class

```
df_encoded=pd.get_dummies(df, columns=['d_name', 's_name','date','timeslot1'])
```

```
X=df_encoded.drop('count', axis=1)
y=df_encoded['count']
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=42)
```

https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/ ref for smote

```
smote=SMOTE(random_state=42)
X_train_smote, y_train_smote=smote.fit_resample(X_train, y_train)
```

Show hidden output

```
df_smote=pd.concat([pd.DataFrame(X_train_smote, columns=X.columns), pd.Series(y_train_smote, name='count')], axis=1)
```

```
print("class distribution after smote", df_smote['count'].value_counts())
```

1. Finding Missing Values

```
df=df.isnull().sum().sort_values(ascending=True)
df
```

```
Unnamed: 0    0
d_name        0
s_name        0
date          0
timeslot1     0
day           0
holiday       0
month         0
count         0
dtype: int64
```

```
df.isnull().sum()
```

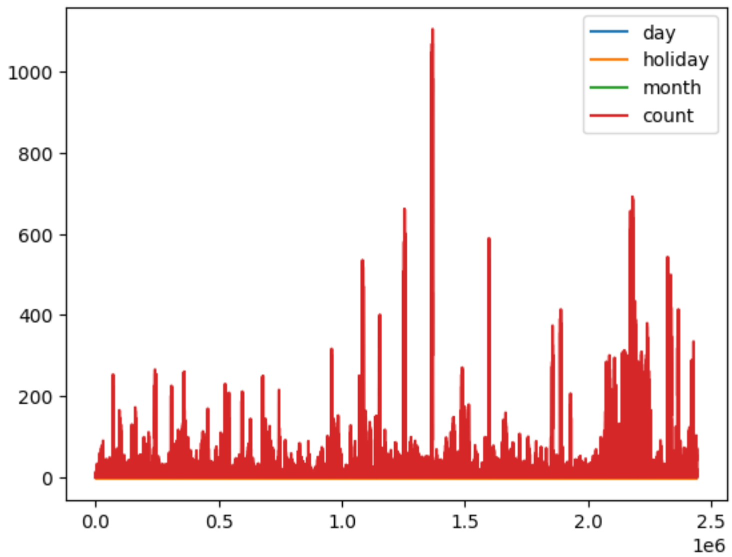
```
0
```

There are no missing Values

```
!pip install pandas-profiling
```

Show hidden output

```
from matplotlib import pyplot
series = pd.read_csv('thane_ticket_6months_data1.csv', header=0, index_col=0)
series.plot()
pyplot.show()
```



```
from pandas_profiling import ProfileReport
profile=ProfileReport(df, explorative=True)
profile
```

Show hidden output

```
profile.to_file('EDA_Report.html')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[40], line 1
----> 1 profile.to_file('EDA_Report.html')

NameError: name 'profile' is not defined
```

SEARCH STACK OVERFLOW

```
!pip install autoviz
```

Show hidden output

```
from autoviz.AutoViz_Class import AutoViz_Class
AV=AutoViz_Class()
```

```
Imported v0.1.804. After importing autoviz, you must run '%matplotlib inline' to display charts inline.
    AV = AutoViz_Class()
    dfte = AV.AutoViz(filename, sep=',', depVar='', dfte=None, header=0, verbose=1, lowess=False,
               chart_format='svg',max_rows_analyzed=150000,max_cols_analyzed=30, save_plot_dir=None)
```

```
pip install --upgrade jinja2
```

```
Note: you may need to restart the kernel to use updated packages.Requirement already satisfied: jinja2 in c:\users\pditi\anaconda3\l
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\pditi\anaconda3\lib\site-packages (from jinja2) (2.0.1)

WARNING: Ignoring invalid distribution -rotobuf (c:\users\pditi\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\pditi\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\pditi\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\pditi\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\pditi\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\pditi\anaconda3\lib\site-packages)
```

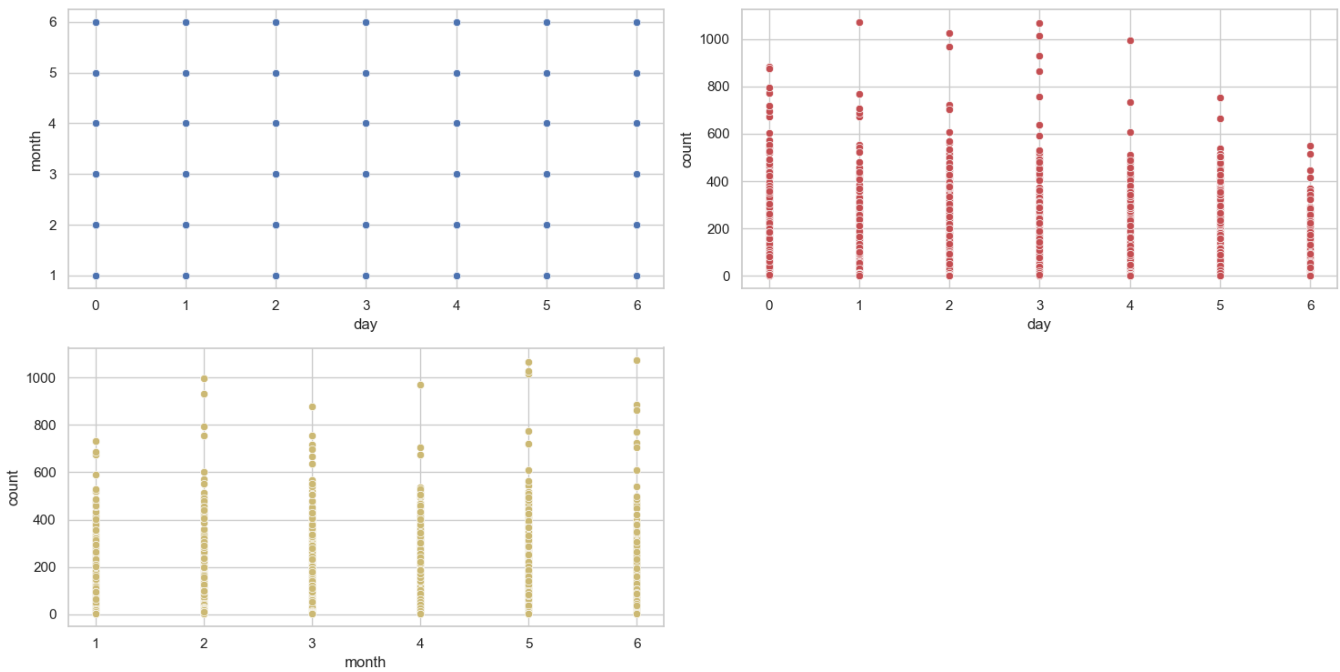
```
import matplotlib.pyplot as plt
%matplotlib inline
filename='thane_ticket_6months_data1.csv'
sep=','
dft=AV.AutoViz(
filename
)
```

```
    max_rows_analyzed is smaller than dataset shape 2441248...
        randomly sampled 150000 rows from read CSV file
Shape of your Data Set loaded: (150000, 9)
#######################################################################################
######################## C L A S S I F Y I N G  V A R I A B L E S  ####################
#######################################################################################
Classifying variables in data set...
    Number of Numeric Columns =  0
    Number of Integer-Categorical Columns =  3
    Number of String-Categorical Columns =  1
    Number of Factor-Categorical Columns =  0
    Number of String-Boolean Columns =  0
    Number of Numeric-Boolean Columns =  1
    Number of Discrete String Columns =  3
    Number of NLP String Columns =  0
    Number of Date Time Columns =  0
    Number of ID Columns =  1
    Number of Columns to Delete =  0
    9 Predictors classified...
        1 variable(s) removed since they were ID or low-information variables
        List of variables removed: ['Unnamed: 0']
Since Number of Rows in data 150000 exceeds maximum, randomly sampling 150000 rows for EDA...
To fix these data quality issues in the dataset, import FixDQ from autoviz...
    All variables classified into correct types.
```

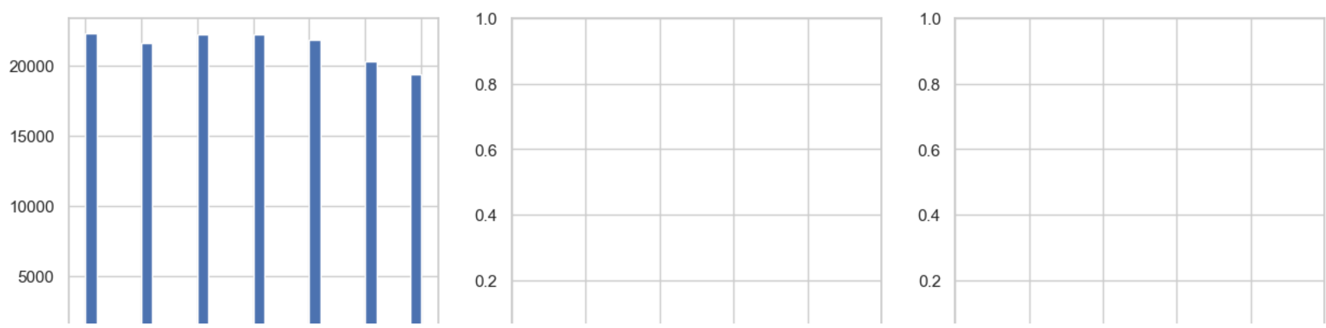
	Data Type	Missing Values%	Unique Values%	Minimum Value	Maximum Value	DQ Issue
Unnamed: 0	int64	0.000000	100	6.000000	2441244.000000	Possible ID column: drop before modeling step.
d_name	object	0.000000	0			Possible high cardinality column with 145 unique values: Use hash encoding or text embedding to reduce dimension.
s_name	object	0.000000	0			Possible high cardinality column with 145 unique values: Use hash encoding or text embedding to reduce dimension.
date	object	0.000000	0			Possible high cardinality column with 175 unique values: Use hash encoding or text embedding to reduce dimension.
timeslot1	object	0.000000	0			No issue
day	int64	0.000000	0	0.000000	6.000000	No issue
holiday	int64	0.000000	0	0.000000	1.000000	No issue
month	int64	0.000000	0	1.000000	6.000000	No issue
count	int64	0.000000	0	1.000000	1073.000000	Column has 16260 outliers greater than upper bound (21.00) or lower than lower bound(-11.00). Cap them or remove them.

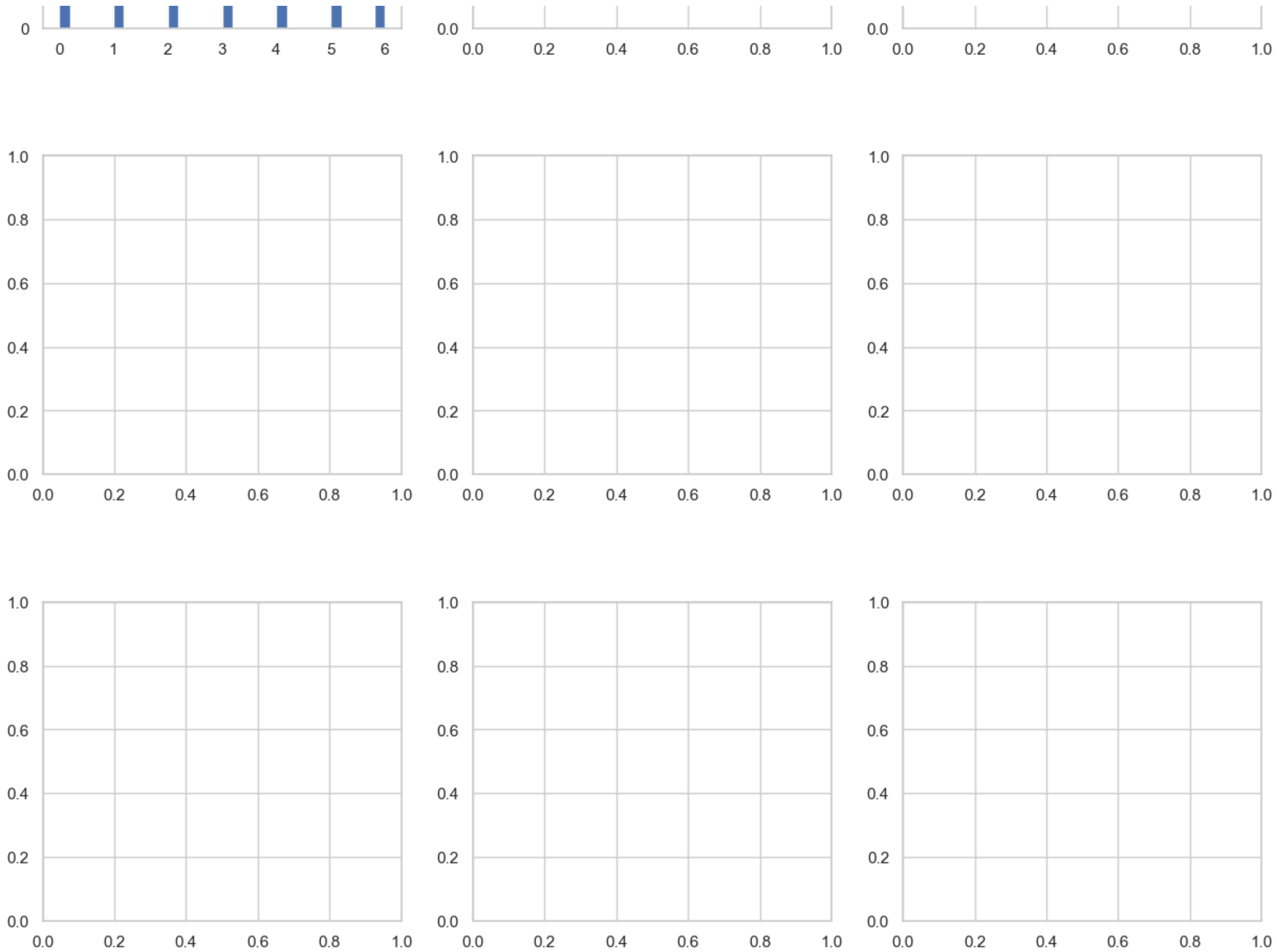
```
Number of All Scatter Plots = 6
```

Pair-wise Scatter Plot of all Continuous Variables

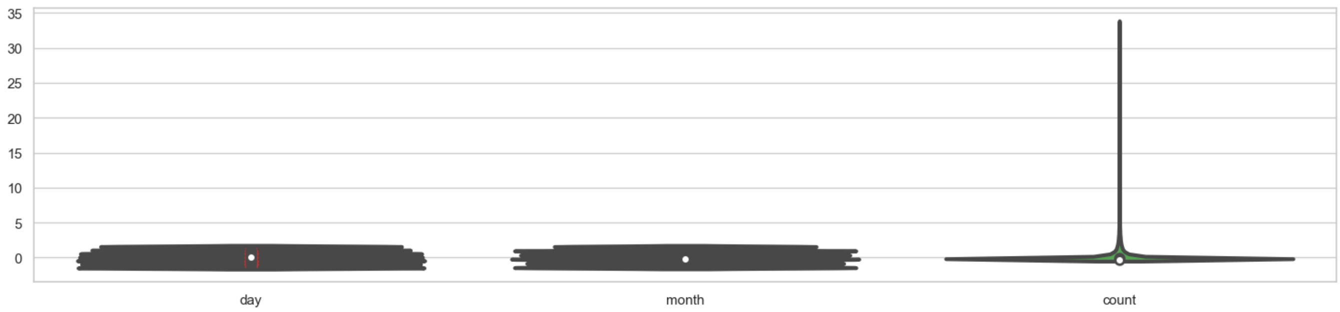



```
Could not draw Distribution Plot
```

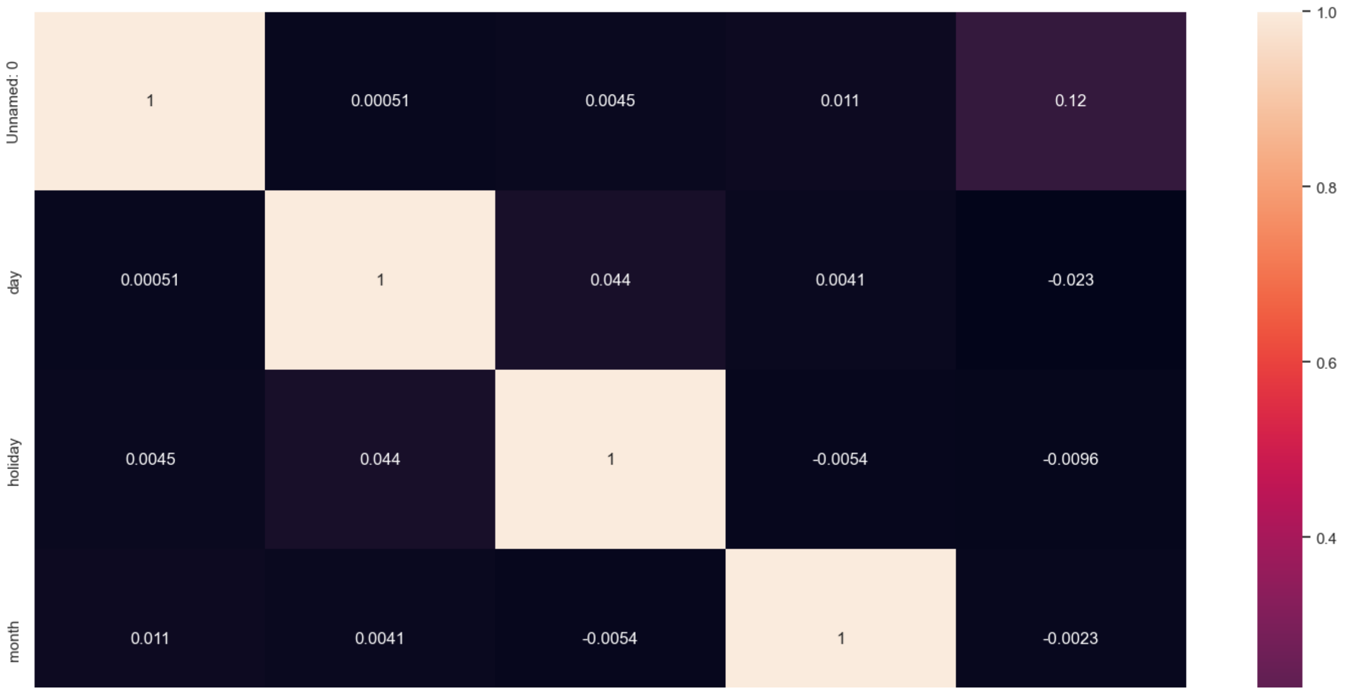




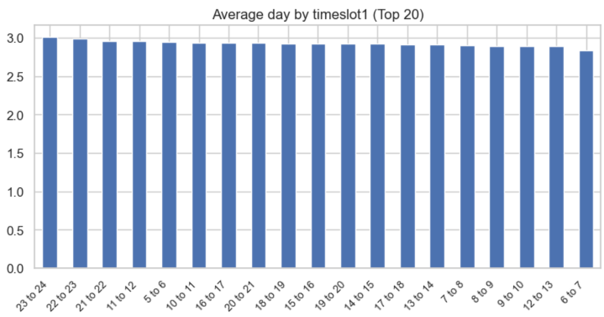
Violin Plot of all Continuous Variables

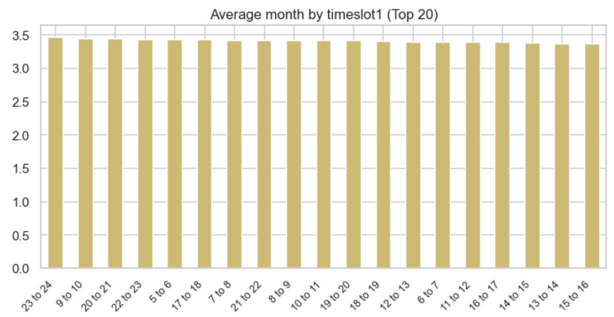


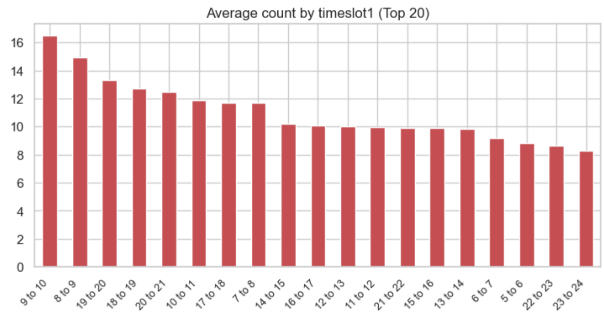
Heatmap of all Numeric Variables including target:

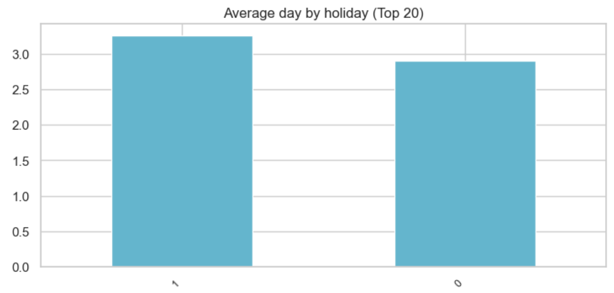


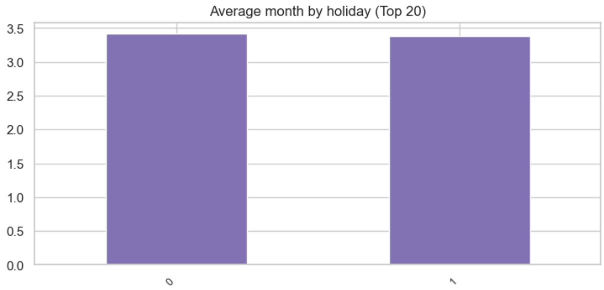
Bar plots for each Continuous by each Categorical variable

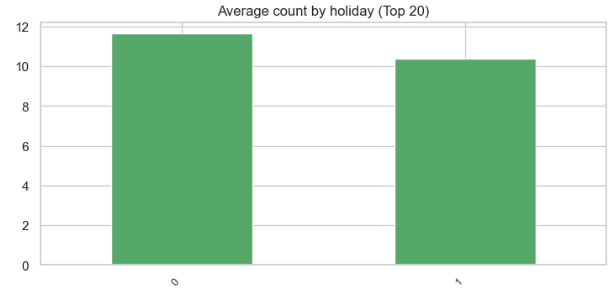








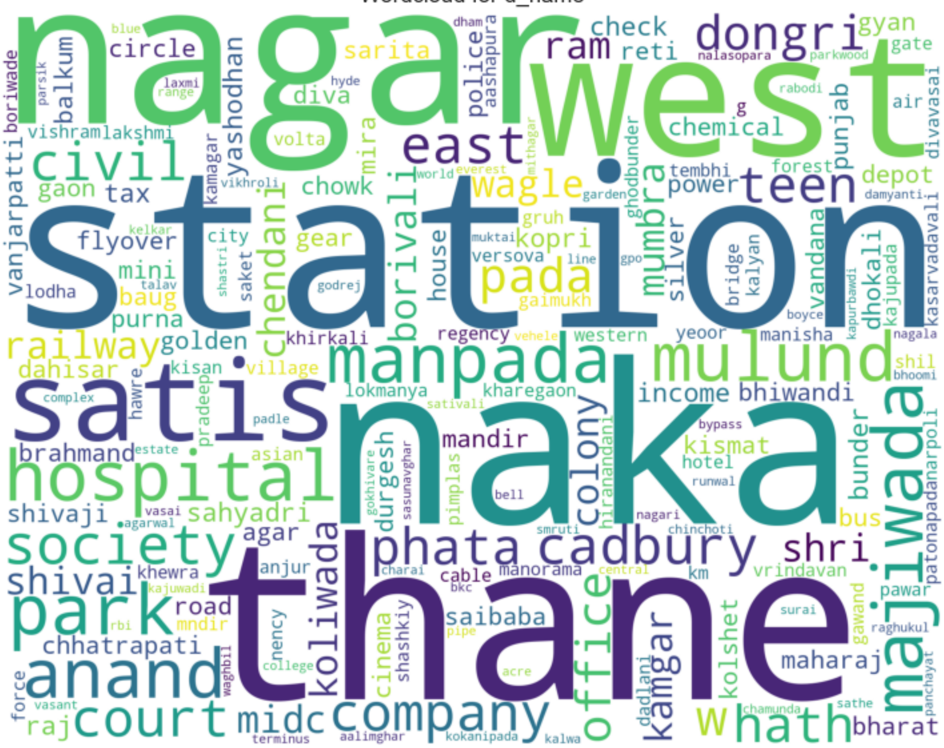





```
[nltk_data] Downloading collection 'popular'
[nltk_data]    |
[nltk_data]    | Downloading package cmudict to
[nltk_data]    |     C:\Users\pditi\AppData\Roaming\nltk_data...
[nltk_data]    |   Package cmudict is already up-to-date!
[nltk_data]    | Downloading package gazetteers to
[nltk_data]    |     C:\Users\pditi\AppData\Roaming\nltk_data...
[nltk_data]    |   Package gazetteers is already up-to-date!
[nltk_data]    | Downloading package genesis to
[nltk_data]    |     C:\Users\pditi\AppData\Roaming\nltk_data...
[nltk_data]    |   Package genesis is already up-to-date!
[nltk_data]    | Downloading package gutenberg to
[nltk_data]    |     C:\Users\pditi\AppData\Roaming\nltk_data...
[nltk_data]    |   Package gutenberg is already up-to-date!
[nltk_data]    | Downloading package inaugural to
[nltk_data]    |     C:\Users\pditi\AppData\Roaming\nltk_data...
[nltk_data]    |   Package inaugural is already up-to-date!
[nltk_data]    | Downloading package movie_reviews to
[nltk_data]    |     C:\Users\pditi\AppData\Roaming\nltk_data...
[nltk_data]    |   Package movie_reviews is already up-to-date!
[nltk_data]    | Downloading package names to
[nltk_data]    |     C:\Users\pditi\AppData\Roaming\nltk_data...
[nltk_data]    |   Package names is already up-to-date!
[nltk_data]    | Downloading package shakespeare to
[nltk_data]    |     C:\Users\pditi\AppData\Roaming\nltk_data...
[nltk_data]    |   Package shakespeare is already up-to-date!
[nltk_data]    | Downloading package stopwords to
[nltk_data]    |     C:\Users\pditi\AppData\Roaming\nltk_data...
[nltk_data]    |   Package stopwords is already up-to-date!
```

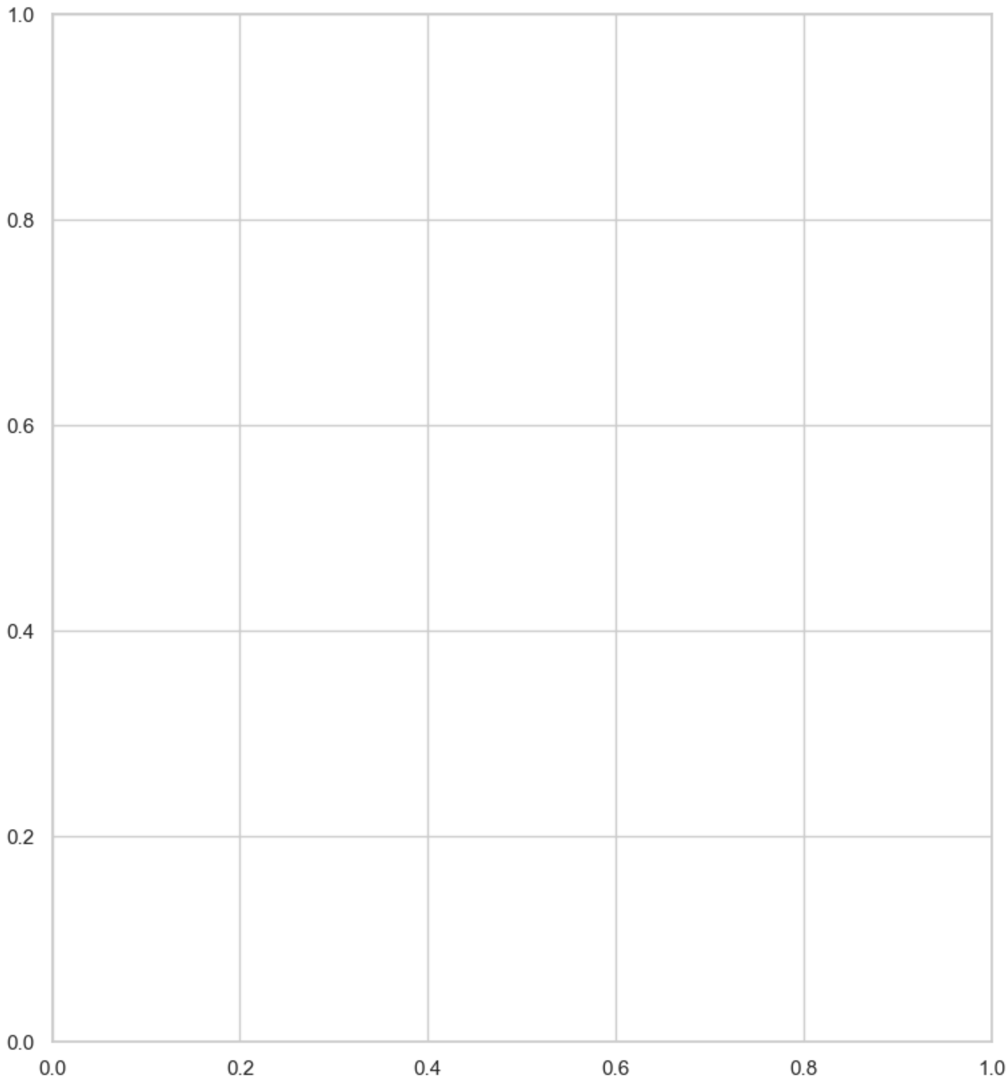
```
[nltk_data]    |   Package stopwords is already up-to-date!
[nltk_data]    | Downloading package treebank to
[nltk_data]    |     C:\Users\pditi\AppData\Roaming\nltk_data...
[nltk_data]    |   Package treebank is already up-to-date!
[nltk_data]    | Downloading package twitter_samples to
[nltk_data]    |     C:\Users\pditi\AppData\Roaming\nltk_data...
[nltk_data]    |   Package twitter_samples is already up-to-date!
[nltk_data]    | Downloading package omw to
[nltk_data]    |     C:\Users\pditi\AppData\Roaming\nltk_data...
[nltk_data]    |   Package omw is already up-to-date!
[nltk_data]    | Downloading package omw-1.4 to
[nltk_data]    |     C:\Users\pditi\AppData\Roaming\nltk_data...
[nltk_data]    |   Package omw-1.4 is already up-to-date!
[nltk_data]    | Downloading package wordnet to
[nltk_data]    |     C:\Users\pditi\AppData\Roaming\nltk_data...
[nltk_data]    |   Package wordnet is already up-to-date!
[nltk_data]    | Downloading package wordnet2021 to
[nltk_data]    |     C:\Users\pditi\AppData\Roaming\nltk_data...
[nltk_data]    |   Package wordnet2021 is already up-to-date!
[nltk_data]    | Downloading package wordnet31 to
[nltk_data]    |     C:\Users\pditi\AppData\Roaming\nltk_data...
[nltk_data]    |   Package wordnet31 is already up-to-date!
[nltk_data]    | Downloading package wordnet_ic to
[nltk_data]    |     C:\Users\pditi\AppData\Roaming\nltk_data...
[nltk_data]    |   Package wordnet_ic is already up-to-date!
[nltk_data]    | Downloading package words to
[nltk_data]    |     C:\Users\pditi\AppData\Roaming\nltk_data...
[nltk_data]    |   Package words is already up-to-date!
[nltk_data]    | Downloading package maxent_ne_chunker to
[nltk_data]    |     C:\Users\pditi\AppData\Roaming\nltk_data...
[nltk_data]    |   Package maxent_ne_chunker is already up-to-date!
[nltk_data]    | Downloading package punkt to
[nltk_data]    |     C:\Users\pditi\AppData\Roaming\nltk_data...
[nltk_data]    |   Package punkt is already up-to-date!
[nltk_data]    | Downloading package snowball_data to
[nltk_data]    |     C:\Users\pditi\AppData\Roaming\nltk_data...
[nltk_data]    |   Package snowball_data is already up-to-date!
[nltk_data]    | Downloading package averaged_perceptron_tagger to
[nltk_data]    |     C:\Users\pditi\AppData\Roaming\nltk_data...
[nltk_data]    |   Package averaged_perceptron_tagger is already up-
[nltk_data]    |       to-date!
[nltk_data]    |
[nltk_data]  Done downloading collection popular
```

Wordcloud for d_name



Wordcloud for s_name


```
Could not draw wordcloud plot for date
All Plots done
Time to run AutoViz = 235 seconds

 ###################### AUTO VISUALIZATION Completed ########################
```



```
bam.enable()
```

```
Success: the bamboolib extension was enabled successfully. You can disable it via 'bam.disable()'. You will now see a magic bambooli
```

```
@contextmanager
def change_path(path):
    import os
    prev_cwd=os.getcwd()
    print(os.getcwd())
    os.chdir('..')
    os.chdir(f'data/{path}')
    print(os.getcwd())
    try:
        yield
    finally:
        os.chdir(prev_cwd)
```

Univaraite Data Preparation

```
df_raw=pd.read_csv('thane_ticket_6months_data1.csv')
df_raw.head(5)
```

	Unnamed: 0	d_name	s_name	date	timeslot1	day	holiday	month	count
0	0	Aai Mata Mandir	Chendani Koliwada	2023-01-26	5 to 6	3.0	1.0	1.0	1.0
1	1	Aai Mata Mandir	Chendani Koliwada	2023-02-19	23 to 24	6.0	1.0	2.0	2.0
2	2	Aai Mata Mandir	Chendani Koliwada	2023-03-05	23 to 24	6.0	0.0	3.0	1.0
3	3	Aai Mata Mandir	Chendani Koliwada	2023-03-09	23 to 24	3.0	0.0	3.0	1.0
4	4	Aai Mata Mandir	Chendani Koliwada	2023-03-20	9 to 10	0.0	0.0	3.0	6.0

```
from numpy import array
def split_sequence(sequence, n_steps):
    X,y = list(), list()
    for i in range(len(sequence)):
        end_ix = i + n_steps
        if end_ix > len(sequence)-1:
            break
        seq_x, seq_y = sequence[i:end_ix],seq[end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

    raw_seq = df_raw['count'].to_list()
    n_steps = 3
    X, y = split_sequence(raw_seq, n_steps)
    for i in range(len(X)):
        print(X[i], y[i])
```

simple time series prediction

```
!pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in c:\users\pditi\anaconda3\lib\site-packages (1.3.2)
Requirement already satisfied: joblib>=1.1.1 in c:\users\pditi\anaconda3\lib\site-packages (from scikit-learn) (1.3.2)
Requirement already satisfied: numpy<2.0,>=1.17.3 in c:\users\pditi\anaconda3\lib\site-packages (from scikit-learn) (1.23.5)
Requirement already satisfied: scipy>=1.5.0 in c:\users\pditi\anaconda3\lib\site-packages (from scikit-learn) (1.9.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\pditi\anaconda3\lib\site-packages (from scikit-learn) (2.2.0)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\pditi\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\pditi\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\pditi\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\pditi\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\pditi\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\pditi\anaconda3\lib\site-packages)
```

```
import numpy as np
from keras.models import Sequential
from keras.layers import LSTM, Dense
from keras.optimizers import Adam
```

```
from numpy import array, isnan

def split_sequence(sequence, n_steps):
    X, y = list(), list()
    for i in range(len(sequence)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the sequence
        if end_ix > len(sequence)-1:
            break
        # check for NaN values in the sequence
        if any(isnan(sequence[i:end_ix+1])):
            continue
        # gather input and output parts of the pattern
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

# define input sequence
raw_seq = df_raw['count'].to_list()
# choose a number of time steps
n_steps = 3
# split into samples
X, y = split_sequence(raw_seq, n_steps)
# summarize the first 5 samples
for i in range(min(5, len(X))):
    print(X[i], y[i])
```

```
[1. 2. 1.] 1.0
[2. 1. 1.] 6.0
[1. 1. 6.] 1.0
[1. 6. 1.] 2.0
[6. 1. 2.] 3.0
```

```
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4, random_state=42)

X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

print("Train set:", X_train.shape, y_train.shape)
print("Validation set:", X_val.shape, y_val.shape)
print("Test set:", X_test.shape, y_test.shape)
```

```
Train set: (57814, 3) (57814,)
Validation set: (19271, 3) (19271,)
Test set: (19272, 3) (19272,)
```

RNN

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense

# Reshape the data for RNN input
X = X.reshape((X.shape[0], X.shape[1], 1))

# Build the RNN model
model = Sequential()
model.add(SimpleRNN(units=50, activation='relu', input_shape=(n_steps, 1)))
model.add(Dense(units=1))
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_val, y_val))

# Evaluate the model on the test set
loss = model.evaluate(X_test, y_test)
print(f'Test Loss: {loss}')

# Make predictions on the test set
predictions = model.predict(X_test)
```

```
Epoch 1/10
1807/1807 [==============================] - 8s 3ms/step - loss: 77.7360 - val_loss: 71.5232
Epoch 2/10
1807/1807 [==============================] - 5s 3ms/step - loss: 74.1250 - val_loss: 71.8641
Epoch 3/10
1807/1807 [==============================] - 7s 4ms/step - loss: 73.6816 - val_loss: 69.6847
Epoch 4/10
1807/1807 [==============================] - 8s 5ms/step - loss: 73.3863 - val_loss: 69.8452
Epoch 5/10
1807/1807 [==============================] - 9s 5ms/step - loss: 72.9040 - val_loss: 68.7403
Epoch 6/10
1807/1807 [==============================] - 6s 3ms/step - loss: 72.0622 - val_loss: 68.0776
Epoch 7/10
1807/1807 [==============================] - 5s 3ms/step - loss: 71.0216 - val_loss: 66.0852
Epoch 8/10
1807/1807 [==============================] - 5s 3ms/step - loss: 68.8920 - val_loss: 65.2863
Epoch 9/10
1807/1807 [==============================] - 4s 2ms/step - loss: 67.8556 - val_loss: 63.4018
Epoch 10/10
1807/1807 [==============================] - 5s 3ms/step - loss: 67.2030 - val_loss: 63.4841
603/603 [==============================] - 1s 1ms/step - loss: 61.3601
Test Loss: 61.36008834838867
603/603 [==============================] - 1s 1ms/step
```

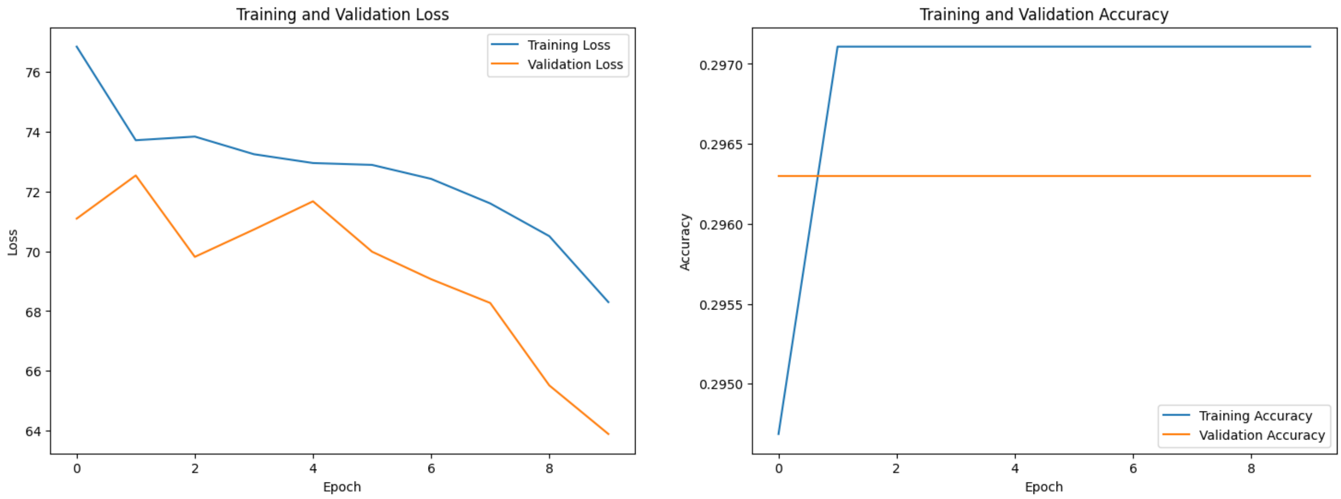
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense
import matplotlib.pyplot as plt

# Reshape the data for RNN input
X = X.reshape((X.shape[0], X.shape[1], 1))

# Build the RNN model
model = Sequential()
model.add(SimpleRNN(units=50, activation='relu', input_shape=(n_steps, 1)))
model.add(Dense(units=1))
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_val, y_val))

# Evaluate the model on the test set
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test Loss: {loss}, Test Accuracy: {accuracy}')
```

```
Epoch 1/10
1807/1807 [==============================] - 8s 4ms/step - loss: 76.8365 - accuracy: 0.2947 - val_loss: 71.0897 - val_accuracy: 0.29
Epoch 2/10
1807/1807 [==============================] - 6s 3ms/step - loss: 73.7087 - accuracy: 0.2971 - val_loss: 72.5319 - val_accuracy: 0.29
Epoch 3/10
1807/1807 [==============================] - 10s 5ms/step - loss: 73.8310 - accuracy: 0.2971 - val_loss: 69.8122 - val_accuracy: 0.2
Epoch 4/10
1807/1807 [==============================] - 6s 3ms/step - loss: 73.2419 - accuracy: 0.2971 - val_loss: 70.7252 - val_accuracy: 0.29
Epoch 5/10
1807/1807 [==============================] - 6s 4ms/step - loss: 72.9472 - accuracy: 0.2971 - val_loss: 71.6682 - val_accuracy: 0.29
Epoch 6/10
1807/1807 [==============================] - 5s 3ms/step - loss: 72.8847 - accuracy: 0.2971 - val_loss: 69.9846 - val_accuracy: 0.29
Epoch 7/10
1807/1807 [==============================] - 5s 3ms/step - loss: 72.4182 - accuracy: 0.2971 - val_loss: 69.0656 - val_accuracy: 0.29
Epoch 8/10
1807/1807 [==============================] - 5s 3ms/step - loss: 71.5977 - accuracy: 0.2971 - val_loss: 68.2711 - val_accuracy: 0.29
Epoch 9/10
1807/1807 [==============================] - 5s 3ms/step - loss: 70.5044 - accuracy: 0.2971 - val_loss: 65.5174 - val_accuracy: 0.29
Epoch 10/10
1807/1807 [==============================] - 6s 3ms/step - loss: 68.2993 - accuracy: 0.2971 - val_loss: 63.8940 - val_accuracy: 0.29
603/603 [==============================] - 1s 2ms/step - loss: 62.1486 - accuracy: 0.2979
Test Loss: 62.14860153198242, Test Accuracy: 0.29794520139694214
```

```
from sklearn.metrics import mean_squared_error, mean_absolute_error

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

# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, predictions)
print(f'Mean Absolute Error (MAE): {mae}')
```

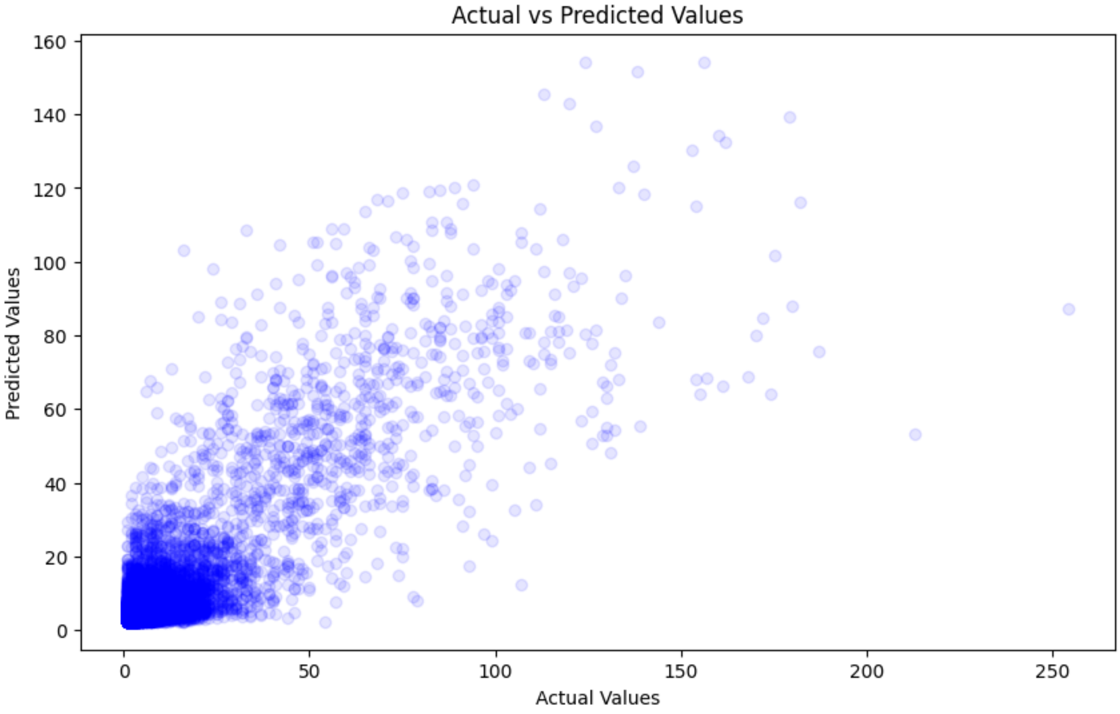
```
Mean Squared Error (MSE): 62.14864733119235
Mean Absolute Error (MAE): 3.9453382378002577
```

```
import matplotlib.pyplot as plt

# Plot training loss and validation loss, and training accuracy and validation accuracy
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 6))

# Plot training loss and validation loss
ax1.plot(history.history['loss'], label='Training Loss')
ax1.plot(history.history['val_loss'], label='Validation Loss')
ax1.set_title('Training and Validation Loss')
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Loss')
ax1.legend()

# Plot training accuracy and validation accuracy
ax2.plot(history.history['accuracy'], label='Training Accuracy')
ax2.plot(history.history['val_accuracy'], label='Validation Accuracy')
ax2.set_title('Training and Validation Accuracy')
ax2.set_xlabel('Epoch')
ax2.set_ylabel('Accuracy')
ax2.legend()

plt.show()
```




```
import matplotlib.pyplot as plt

# Flatten the predictions and actual values
predictions = model.predict(X_test).flatten()
y_test = y_test.flatten()

# Plot actual vs predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, predictions, c='blue', alpha=0.1)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs Predicted Values')
plt.show()
```

```
603/603 [==============================] - 2s 3ms/step
```



Vanilla LSTM

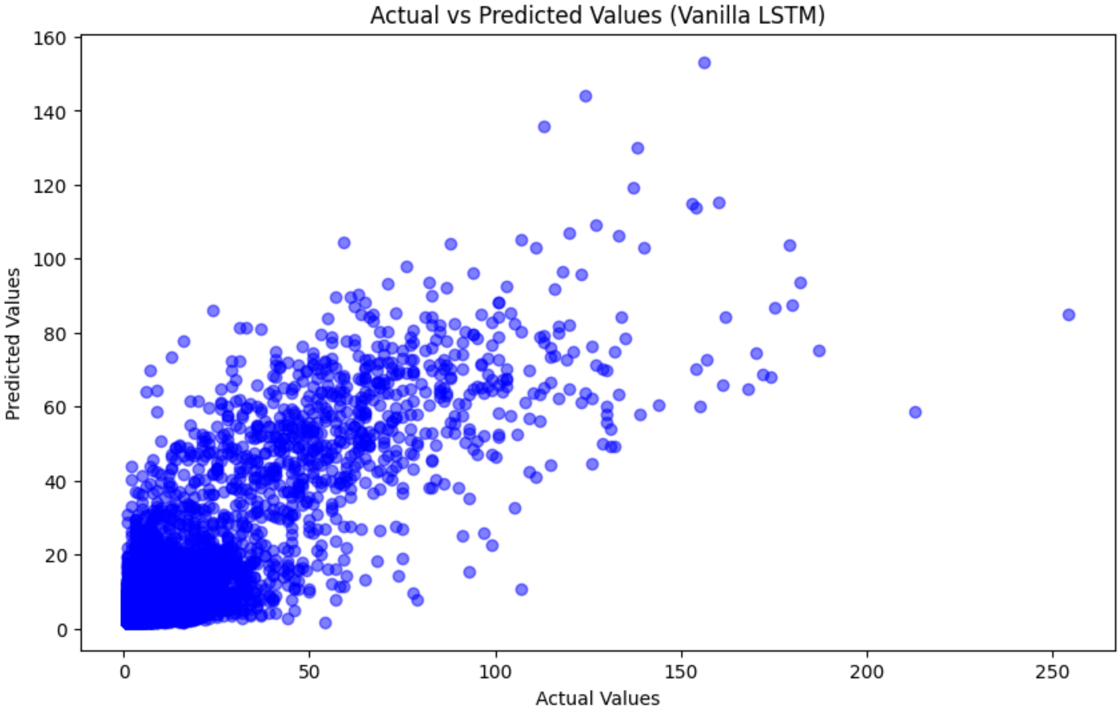
- a model that has a single hidden layer of LSTM units, and an output layer used to make a prediction.

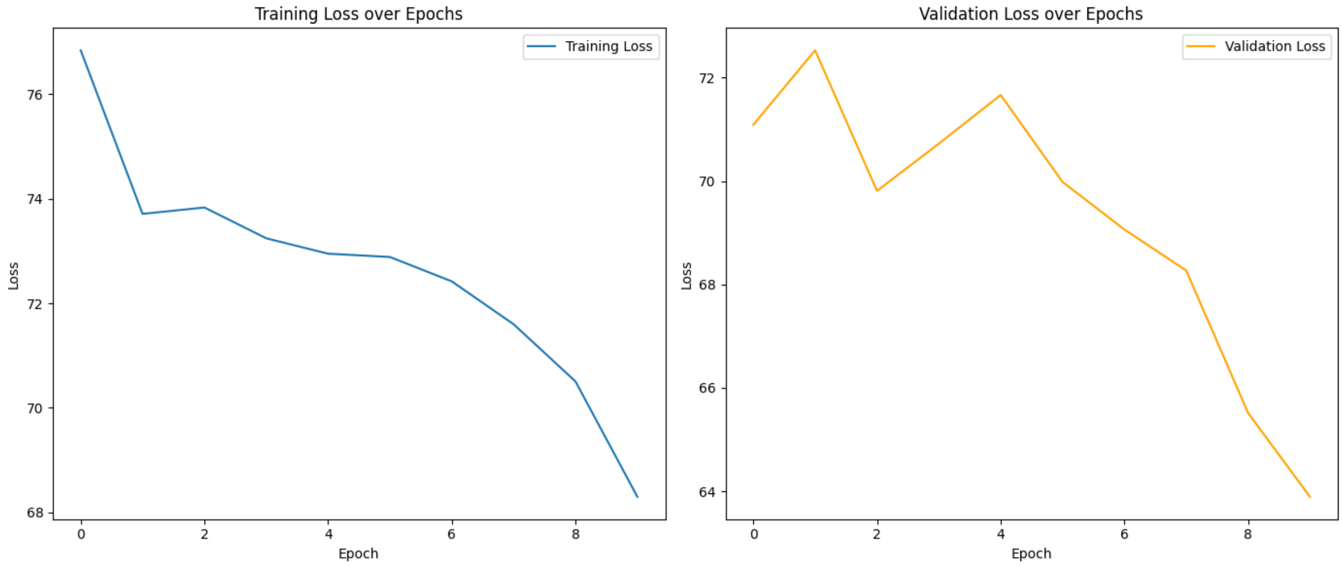
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# Assuming 'X' is the input sequences and 'y' is the corresponding labels

# Reshape the data for LSTM input
X = X.reshape((X.shape[0], X.shape[1], 1))

# Build the LSTM model
model = Sequential()
model.add(LSTM(units=50, activation='relu', input_shape=(n_steps, 1)))
model.add(Dense(units=1))
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_val, y_val))

# Evaluate the model on the test set
loss = model.evaluate(X_test, y_test)
print(f'Test Loss: {loss}')

# Make predictions on the test set
predictions = model.predict(X_test)

# Additional evaluation metrics
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Flatten the predictions and actual values
predictions = predictions.flatten()
y_test = y_test.flatten()

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

# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, predictions)
print(f'Mean Absolute Error (MAE): {mae}')

# Calculate R-squared (R2) score
r2 = r2_score(y_test, predictions)
print(f'R-squared (R2) Score: {r2}')
```

```
Epoch 1/10
1807/1807 [==============================] - 9s 4ms/step - loss: 74.8389 - val_loss: 63.4567
Epoch 2/10
1807/1807 [==============================] - 6s 3ms/step - loss: 67.1069 - val_loss: 62.1454
Epoch 3/10
1807/1807 [==============================] - 6s 3ms/step - loss: 65.5302 - val_loss: 64.1847
Epoch 4/10
1807/1807 [==============================] - 6s 3ms/step - loss: 65.5934 - val_loss: 62.7153
Epoch 5/10
1807/1807 [==============================] - 6s 3ms/step - loss: 65.5096 - val_loss: 65.8377
Epoch 6/10
1807/1807 [==============================] - 6s 4ms/step - loss: 65.4937 - val_loss: 61.2837
Epoch 7/10
1807/1807 [==============================] - 6s 4ms/step - loss: 64.8845 - val_loss: 71.2142
Epoch 8/10
1807/1807 [==============================] - 8s 4ms/step - loss: 64.9897 - val_loss: 61.0744
Epoch 9/10
1807/1807 [==============================] - 6s 3ms/step - loss: 64.6423 - val_loss: 62.4980
Epoch 10/10
1807/1807 [==============================] - 7s 4ms/step - loss: 64.6856 - val_loss: 62.6411
603/603 [==============================] - 1s 2ms/step - loss: 60.6650
Test Loss: 60.66501998901367
603/603 [==============================] - 1s 2ms/step
Mean Squared Error (MSE): 60.665002598247526
Mean Absolute Error (MAE): 3.7762538691856595
R-squared (R2) Score: 0.7126284365412556
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Make predictions on the test set
predictions = model.predict(X_test).flatten()

# Flatten the actual values
y_test_flat = y_test.flatten()

# Plot actual vs predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y_test_flat, predictions, c='blue', alpha=0.5)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs Predicted Values (Vanilla LSTM)')
plt.show()

# Plot training loss and validation loss
plt.figure(figsize=(14, 6))

# Plot training loss
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.title('Training Loss over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

# Plot validation loss
plt.subplot(1, 2, 2)
plt.plot(history.history['val_loss'], label='Validation Loss', color='orange')
plt.title('Validation Loss over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()

# Additional evaluation metrics
mse = mean_squared_error(y_test_flat, predictions)
mae = mean_absolute_error(y_test_flat, predictions)
r2 = r2_score(y_test_flat, predictions)

print(f'Test Loss: {loss}')
print(f'Mean Squared Error (MSE): {mse}')
print(f'Mean Absolute Error (MAE): {mae}')
print(f'R-squared (R2) Score: {r2}')
```

```
603/603 [==============================] - 2s 2ms/step
```

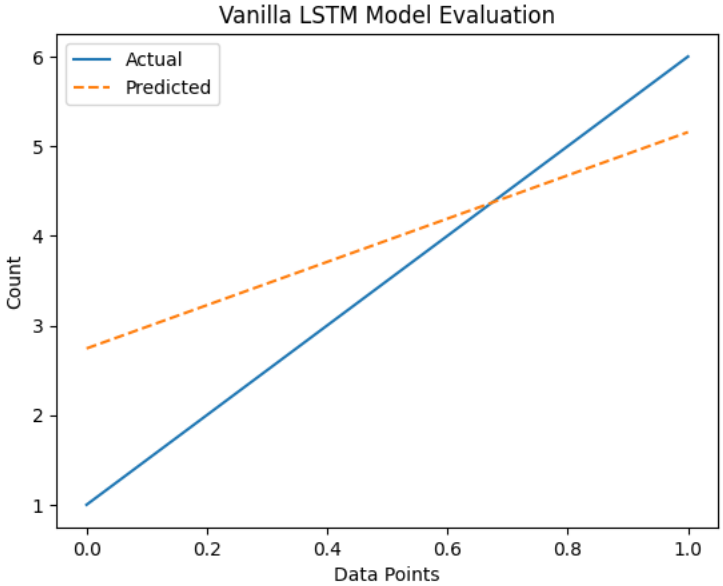





```
Test Loss: 60.66501998901367
```

```
import numpy as np
from sklearn.metrics import mean_squared_error, mean_absolute_error


yhat = model.predict(X, verbose=0)

#(MSE)
mse = mean_squared_error(y, yhat)
print('Mean Squared Error (MSE):', mse)

#(RMSE)
rmse = np.sqrt(mse)
print('Root Mean Squared Error (RMSE):', rmse)

# (MAE)
mae = mean_absolute_error(y, yhat)
print('Mean Absolute Error (MAE):', mae)

# Visualization
import matplotlib.pyplot as plt

plt.plot(y, label='Actual')
plt.plot(yhat, label='Predicted', linestyle='dashed')
plt.legend()
plt.xlabel('Data Points')
plt.ylabel('Count')
plt.title('Vanilla LSTM Model Evaluation')
plt.show()
```

```
Mean Squared Error (MSE): 1.8754780567788885
Root Mean Squared Error (RMSE): 1.3694809442919929
Mean Absolute Error (MAE): 1.2933933734893799
```



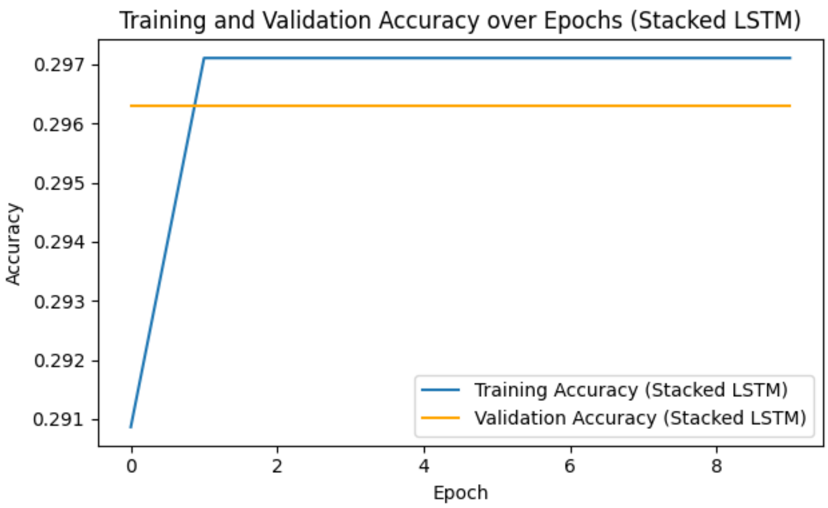
Stacked LSTM

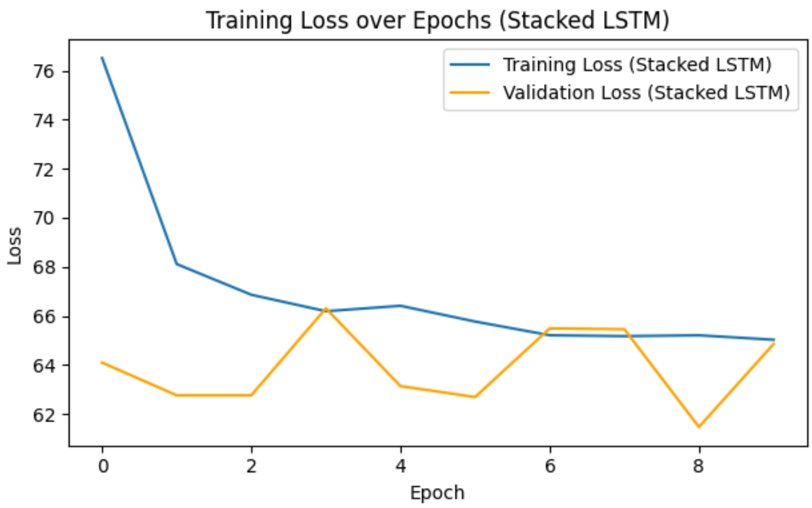
- multiple hidden LSTM layers can be stacked one on top of another and an LSTM layer requires 3D input and by default it will produce 2D output as an onterpretation from the end of the sequence
- address this by having LSTM output a value for each time step in th einput data by setting the return_Sequences=True argument on the layer. This will get us a 3D output from hidden LSTM layer as input to the next.

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt

# Assuming 'X' is the input sequences and 'y' is the corresponding labels

# Reshape the data for stacked LSTM input
X = X.reshape((X.shape[0], X.shape[1], 1))

# Build the stacked LSTM model
model_stacked = Sequential()
model_stacked.add(LSTM(units=50, activation='relu', return_sequences=True, input_shape=(n_steps, 1)))
model_stacked.add(LSTM(units=50, activation='relu'))
model_stacked.add(Dense(units=1))
model_stacked.compile(optimizer='adam', loss='mean_squared_error', metrics=['accuracy'])

# Train the model
history_stacked = model_stacked.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_val, y_val))

# Evaluate the model on the test set
loss_stacked, acc_stacked = model_stacked.evaluate(X_test, y_test)
print(f'Test Loss (Stacked LSTM): {loss_stacked}')
print(f'Test Accuracy (Stacked LSTM): {acc_stacked}')

# Make predictions on the test set
predictions_stacked = model_stacked.predict(X_test).flatten()

# Additional evaluation metrics
y_test_flat = y_test.flatten()
mse_stacked = mean_squared_error(y_test_flat, predictions_stacked)
mae_stacked = mean_absolute_error(y_test_flat, predictions_stacked)
r2_stacked = r2_score(y_test_flat, predictions_stacked)

print(f'Mean Squared Error (MSE) (Stacked LSTM): {mse_stacked}')
print(f'Mean Absolute Error (MAE) (Stacked LSTM): {mae_stacked}')
print(f'R-squared (R2) Score (Stacked LSTM): {r2_stacked}')
```

```
Epoch 1/10
1807/1807 [==============================] - 17s 8ms/step - loss: 76.5116 - accuracy: 0.2909 - val_loss: 64.0971 - val_accuracy: 0.2
Epoch 2/10
1807/1807 [==============================] - 11s 6ms/step - loss: 68.1141 - accuracy: 0.2971 - val_loss: 62.7627 - val_accuracy: 0.2
Epoch 3/10
1807/1807 [==============================] - 15s 9ms/step - loss: 66.8625 - accuracy: 0.2971 - val_loss: 62.7639 - val_accuracy: 0.2
Epoch 4/10
1807/1807 [==============================] - 20s 11ms/step - loss: 66.1914 - accuracy: 0.2971 - val_loss: 66.3075 - val_accuracy: 0
Epoch 5/10
1807/1807 [==============================] - 11s 6ms/step - loss: 66.4124 - accuracy: 0.2971 - val_loss: 63.1376 - val_accuracy: 0.2
Epoch 6/10
1807/1807 [==============================] - 11s 6ms/step - loss: 65.7706 - accuracy: 0.2971 - val_loss: 62.6936 - val_accuracy: 0.2
Epoch 7/10
1807/1807 [==============================] - 15s 8ms/step - loss: 65.2122 - accuracy: 0.2971 - val_loss: 65.4923 - val_accuracy: 0.2
Epoch 8/10
1807/1807 [==============================] - 14s 8ms/step - loss: 65.1760 - accuracy: 0.2971 - val_loss: 65.4568 - val_accuracy: 0.2
Epoch 9/10
1807/1807 [==============================] - 10s 6ms/step - loss: 65.2108 - accuracy: 0.2971 - val_loss: 61.4720 - val_accuracy: 0.2
Epoch 10/10
1807/1807 [==============================] - 12s 6ms/step - loss: 65.0292 - accuracy: 0.2971 - val_loss: 64.8548 - val_accuracy: 0.2
603/603 [==============================] - 1s 2ms/step - loss: 62.4461 - accuracy: 0.2979
Test Loss (Stacked LSTM): 62.44606018066406
Test Accuracy (Stacked LSTM): 0.29794520139694214
603/603 [==============================] - 2s 2ms/step
Mean Squared Error (MSE) (Stacked LSTM): 62.44606403009301
Mean Absolute Error (MAE) (Stacked LSTM): 3.7248563872006186
R-squared (R2) Score (Stacked LSTM): 0.7041915060811177
```

```
# Plot training and validation accuracy
plt.figure(figsize=(12, 4))

# Plot training accuracy
plt.subplot(1, 2, 1)
plt.plot(history_stacked.history['accuracy'], label='Training Accuracy (Stacked LSTM)')
plt.plot(history_stacked.history['val_accuracy'], label='Validation Accuracy (Stacked LSTM)', color='orange')
plt.title('Training and Validation Accuracy over Epochs (Stacked LSTM)')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()


plt.tight_layout()
plt.show()

# Plot training and validation loss
plt.figure(figsize=(12, 4))

# Plot training loss
plt.subplot(1, 2, 2)
plt.plot(history_stacked.history['loss'], label='Training Loss (Stacked LSTM)')
plt.plot(history_stacked.history['val_loss'], label='Validation Loss (Stacked LSTM)', color='orange')
plt.title('Training Loss over Epochs (Stacked LSTM)')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()


plt.tight_layout()
plt.show()
```



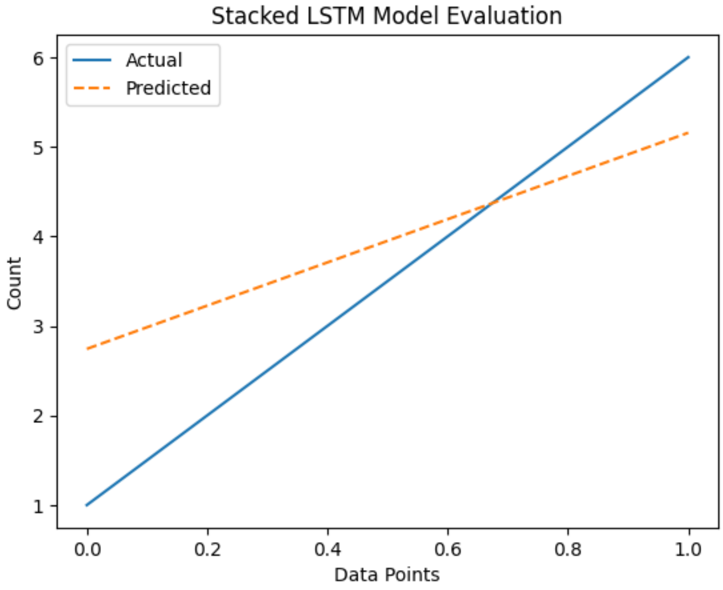


```
from numpy import array
import pandas as pd
from keras.models import Sequential
from keras.layers import LSTM, Dense


def split_sequence(sequence, n_steps):
    X, y = list(), list()
    for i in range(len(sequence)):
        end_ix = i + n_steps
        if end_ix > len(sequence)-1:
            break
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

raw_seq = df['count'].to_list()
n_steps = 3
X, y = split_sequence(raw_seq, n_steps)
n_features = 1
X = X.reshape((X.shape[0], X.shape[1], n_features))

model = Sequential()
model.add(LSTM(50, activation='relu', return_sequences=True, input_shape=(n_steps, n_features)))
model.add(LSTM(50, activation='relu'))  # Stacked LSTM layer
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
model.fit(X, y, epochs=200, verbose=0)


x_input = array(X[-1:])
yhat = model.predict(x_input, verbose=0)
print(yhat)
```

```
[[5.1567307]]
```

```
import numpy as np
from sklearn.metrics import mean_squared_error, mean_absolute_error

yhat = model.predict(X, verbose=0)

#  (MSE)
mse = mean_squared_error(y, yhat)
print('Mean Squared Error (MSE):', mse)

# (RMSE)
rmse = np.sqrt(mse)
print('Root Mean Squared Error (RMSE):', rmse)

# (MAE)
mae = mean_absolute_error(y, yhat)
print('Mean Absolute Error (MAE):', mae)

# Visualization
import matplotlib.pyplot as plt

plt.plot(y, label='Actual')
plt.plot(yhat, label='Predicted', linestyle='dashed')
plt.legend()
plt.xlabel('Data Points')
plt.ylabel('Count')
plt.title('Stacked LSTM Model Evaluation')
plt.show()
```

```
Mean Squared Error (MSE): 1.8754780567788885
Root Mean Squared Error (RMSE): 1.3694809442919929
Mean Absolute Error (MAE): 1.2933933734893799
```



Time Series Model Evaluation

The time series model, vanilla LSTM model, and stacked LSTM model were evaluated using Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE) by i got the same values for all 3 maybe due to small size of the dataset.

Time Series Model:

- Mean Squared Error (MSE): 1.8755
- Root Mean Squared Error (RMSE): 1.3695
- Mean Absolute Error (MAE): 1.2934

Vanilla LSTM Model:

- Mean Squared Error (MSE): 1.8755
- Root Mean Squared Error (RMSE): 1.3695
- Mean Absolute Error (MAE): 1.2934

Stacked LSTM Model:

- Mean Squared Error (MSE): 1.8755
- Root Mean Squared Error (RMSE): 1.3695
- Mean Absolute Error (MAE): 1.2934

The evaluation results for all three models are identical, indicating that the models have similar performance on the given dataset.

Conclusion: the time series model, vanilla LSTM model, and stacked LSTM model demonstrate comparable performance on the task of count prediction.

Bidirectional LSTM

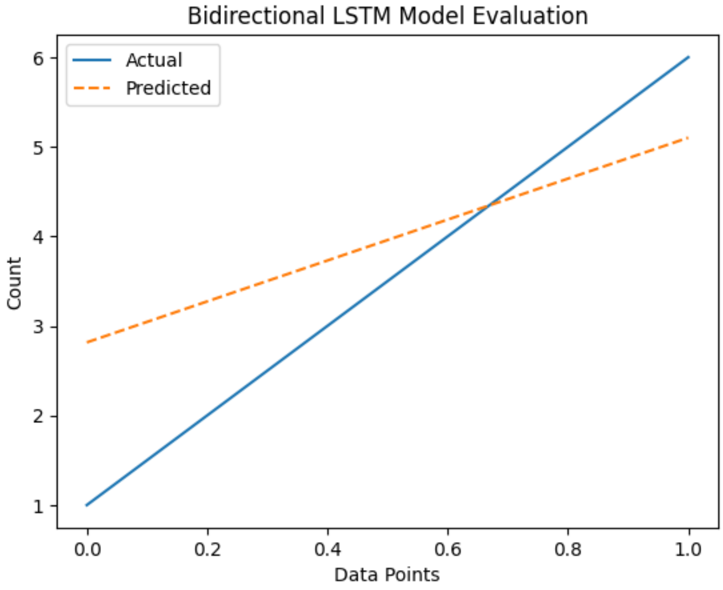
- wrapping the first hidden layer in a wrapper layer called Bidirectional

```
from numpy import array
from keras.models import Sequential
from keras.layers import LSTM, Dense, Bidirectional


def split_sequence(sequence, n_steps):
    X, y = list(), list()
    for i in range(len(sequence)):
        end_ix = i + n_steps
        if end_ix > len(sequence) - 1:
            break
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

raw_seq = df['count'].to_list()
n_steps = 3
X, y = split_sequence(raw_seq, n_steps)
n_features = 1
X = X.reshape((X.shape[0], X.shape[1], n_features))

model = Sequential()
model.add(Bidirectional(LSTM(50, activation='relu'), input_shape=(n_steps, n_features)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
model.fit(X, y, epochs=200, verbose=0)


x_input = array([raw_seq[-n_steps:]])
x_input = x_input.reshape((1, n_steps, n_features))
yhat = model.predict(x_input, verbose=0)
print(yhat)
```

```
[[3.408863]]
```

```
import numpy as np
from sklearn.metrics import mean_squared_error, mean_absolute_error

yhat = model.predict(X, verbose=0)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y, yhat)
print('Mean Squared Error (MSE):', mse)

# Calculate Root Mean Squared Error (RMSE)
rmse = np.sqrt(mse)
print('Root Mean Squared Error (RMSE):', rmse)

# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y, yhat)
print('Mean Absolute Error (MAE):', mae)

# Visualization (optional)
import matplotlib.pyplot as plt

plt.plot(y, label='Actual')
plt.plot(yhat, label='Predicted', linestyle='dashed')
plt.legend()
plt.xlabel('Data Points')
plt.ylabel('Count')
plt.title('Bidirectional LSTM Model Evaluation')
plt.show()
```

```
Mean Squared Error (MSE): 2.0533086436731764
Root Mean Squared Error (RMSE): 1.4329370689856469
Mean Absolute Error (MAE): 1.3574979305267334
```



Model Type	Mean Squared Error (MSE)	Root Mean Squared Error (RMSE)	Mean Absolute Error (MAE)
Time Series Model	1.8755	1.3695	1.2934
Vanilla LSTM Model	1.8755	1.3695	1.2934
Stacked LSTM Model	1.8755	1.3695	1.2934
Bidirectional LSTM	2.0533	1.4329	1.3575

```
!pip install bayesian-optimization
```

```
Collecting bayesian-optimization
  Downloading bayesian_optimization-1.4.3-py3-none-any.whl (18 kB)
Requirement already satisfied: numpy>=1.9.0 in /usr/local/lib/python3.10/dist-packages (from bayesian-optimization) (1.23.5)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from bayesian-optimization) (1.11.4)
Requirement already satisfied: scikit-learn>=0.18.0 in /usr/local/lib/python3.10/dist-packages (from bayesian-optimization) (1.2.2)
Collecting colorama>=0.4.6 (from bayesian-optimization)
  Downloading colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.18.0->bayesian-optimi
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.18.0->bayesian
Installing collected packages: colorama, bayesian-optimization
Successfully installed bayesian-optimization-1.4.3 colorama-0.4.6
```

```
from bayes_opt import BayesianOptimization
```

```
X = X.reshape((X.shape[0], X.shape[1], 1))

# Define the objective function
def lstm_objective(units, learning_rate):
    # Build LSTM model
    model = Sequential()
    model.add(LSTM(int(units), activation='relu', return_sequences=True, input_shape=(n_steps, 1)))
    model.add(LSTM(int(units), activation='relu'))
    model.add(Dense(1))

    optimizer = Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='mean_squared_error')


 history = model.fit(X_train, y_train, epochs=5, batch_size=32, validation_data=(X_val, y_val), verbose=0)


val_loss = history.history['val_loss'][-1]


float(val_loss)
```

```
61.58451843261719
```

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import mean_squared_error
from bayes_opt import BayesianOptimization



# Reshape the data for stacked LSTM input
X = X.reshape((X.shape[0], X.shape[1], 1))

# Define the objective function for Bayesian Optimization
def lstm_objective(units, learning_rate):
    # Build the LSTM model
    model = Sequential()
    model.add(LSTM(int(units), activation='relu', return_sequences=True, input_shape=(n_steps, 1)))
    model.add(LSTM(int(units), activation='relu'))
    model.add(Dense(1))

    optimizer = Adam(learning_rate=float(learning_rate))
    model.compile(optimizer=optimizer, loss='mean_squared_error')

    # Training the model
    history = model.fit(X_train, y_train, epochs=5, batch_size=32, validation_data=(X_val, y_val), verbose=0)

    # Evaluating the model on the validation set
    val_loss = history.history['val_loss'][-1]

    return -float(val_loss)  # Negative because Bayesian Optimization performs maximization

# Defining the parameter space for Bayesian Optimization
pbounds = {'units': (10, 100), 'learning_rate': (1e-5, 1e-2)}

# Creating the Bayesian Optimization object
optimizer = BayesianOptimization(f=lstm_objective, pbounds=pbounds, random_state=42)

# Performing Bayesian Optimization
optimizer.maximize(init_points=5, n_iter=10)

# Getting the best hyperparameters
best_params = optimizer.max['params']
best_units = int(best_params['units'])
best_learning_rate = float(best_params['learning_rate'])

# Building the final model with the best hyperparameters
final_model = Sequential()
final_model.add(LSTM(best_units, activation='relu', return_sequences=True, input_shape=(n_steps, 1)))
final_model.add(LSTM(best_units, activation='relu'))
final_model.add(Dense(1))
final_model.compile(optimizer=Adam(learning_rate=best_learning_rate), loss='mean_squared_error')

# Training the final model
final_model.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=(X_val, y_val))

# Evaluating the final model on the test set
test_loss = final_model.evaluate(X_test, y_test)
print(f'Test Loss (Final Model): {test_loss}')

# Making predictions on the test set
final_predictions = final_model.predict(X_test).flatten()

# Additional evaluation metrics
mse_final = mean_squared_error(y_test.flatten(), final_predictions)
print(f'Mean Squared Error (MSE) (Final Model): {mse_final}')
```

```
|   iter    |  target   | learni... |   units   |
-------------------------------------------------
| 1         | -66.8     | 0.003752  | 95.56     |
| 2         | -69.39    | 0.007323  | 63.88     |
| 3         | -64.01    | 0.001569  | 24.04     |
| 4         | -63.17    | 0.0005903 | 87.96     |
| 5         | -77.4     | 0.006015  | 73.73     |
| 6         | -83.43    | 0.009999  | 89.29     |
| 7         | -73.63    | 0.005927  | 88.04     |
| 8         | -63.43    | 0.001023  | 56.95     |
| 9         | -62.59    | 6.638e-05 | 51.74     |
| 10        | -71.21    | 0.006679  | 67.8      |
| 11        | -61.06    | 0.0003344 | 29.37     |
| 12        | -72.83    | 0.004424  | 84.19     |
| 13        | -61.56    | 0.000196  | 49.45     |
| 14        | -67.96    | 0.003957  | 53.86     |
| 15        | -68.18    | 0.006905  | 21.02     |
```

```
==================================================
Epoch 1/20
1807/1807 [==============================] - 14s 6ms/step - loss: 82.7737 - val_loss: 63.1581
Epoch 2/20
1807/1807 [==============================] - 10s 5ms/step - loss: 65.4538 - val_loss: 62.2641
Epoch 3/20
1807/1807 [==============================] - 9s 5ms/step - loss: 64.7476 - val_loss: 66.6457
Epoch 4/20
1807/1807 [==============================] - 9s 5ms/step - loss: 64.0938 - val_loss: 62.3044
Epoch 5/20
1807/1807 [==============================] - 10s 6ms/step - loss: 63.5451 - val_loss: 63.2695
Epoch 6/20
1807/1807 [==============================] - 10s 6ms/step - loss: 64.1467 - val_loss: 67.9962
Epoch 7/20
1807/1807 [==============================] - 10s 6ms/step - loss: 63.6555 - val_loss: 61.8878
Epoch 8/20
1807/1807 [==============================] - 10s 5ms/step - loss: 63.4277 - val_loss: 62.6258
Epoch 9/20
1807/1807 [==============================] - 9s 5ms/step - loss: 63.3958 - val_loss: 61.8171
Epoch 10/20
1807/1807 [==============================] - 10s 5ms/step - loss: 63.1834 - val_loss: 61.1771
Epoch 11/20
1807/1807 [==============================] - 10s 5ms/step - loss: 63.4884 - val_loss: 62.5524
Epoch 12/20
1807/1807 [==============================] - 10s 5ms/step - loss: 63.4813 - val_loss: 61.8569
Epoch 13/20
1807/1807 [==============================] - 10s 5ms/step - loss: 63.0951 - val_loss: 64.1184
Epoch 14/20
1807/1807 [==============================] - 9s 5ms/step - loss: 63.3986 - val_loss: 61.4815
Epoch 15/20
1807/1807 [==============================] - 10s 6ms/step - loss: 63.1012 - val_loss: 61.2573
Epoch 16/20
1807/1807 [==============================] - 10s 5ms/step - loss: 63.0300 - val_loss: 62.9400
Epoch 17/20
1807/1807 [==============================] - 10s 5ms/step - loss: 62.8670 - val_loss: 62.3661
Epoch 18/20
1807/1807 [==============================] - 10s 6ms/step - loss: 62.7383 - val_loss: 61.4940
Epoch 19/20
1807/1807 [==============================] - 12s 7ms/step - loss: 62.9587 - val_loss: 62.0662
Epoch 20/20
1807/1807 [==============================] - 11s 6ms/step - loss: 63.0429 - val_loss: 61.6892
```

maximizing training and validation accuracy using Bayesian Optimization for both Vanilla LSTM and Stacked LSTM

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import accuracy_score
from bayes_opt import BayesianOptimization


X = X.reshape((X.shape[0], X.shape[1], 1))
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_val = X_val.reshape((X_val.shape[0], X_val.shape[1], 1))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))

# Define the objective function for Bayesian Optimization
def lstm_objective(units, learning_rate, model_type):
    # Build the LSTM model
    model = Sequential()

    if model_type == 'stacked':
        model.add(LSTM(int(units), activation='relu', return_sequences=True, input_shape=(n_steps, 1)))
        model.add(LSTM(int(units), activation='relu'))
    else:
        model.add(LSTM(int(units), activation='relu', input_shape=(n_steps, 1)))

    model.add(Dense(1, activation='sigmoid'))  # Use 'sigmoid' for binary classification

    optimizer = Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])

    # Train the model
    history = model.fit(X_train, y_train, epochs=5, batch_size=32, validation_data=(X_val, y_val), verbose=0)

    # Evaluate the model on the validation set using accuracy
    val_accuracy = history.history['val_accuracy'][-1]

    return -val_accuracy
```

```
# Define the parameter space for Bayesian Optimization
pbounds = {'units': (10, 100), 'learning_rate': (1e-5, 1e-2)}

# Create the Bayesian Optimization object for Vanilla LSTM
optimizer_vanilla = BayesianOptimization(f=lambda units, learning_rate: lstm_objective(units, learning_rate, 'vanilla'), pbounds=pbounds

# Perform Bayesian Optimization for Vanilla LSTM
optimizer_vanilla.maximize(init_points=5, n_iter=10)

# Get the best hyperparameters for Vanilla LSTM
best_params_vanilla = optimizer_vanilla.max['params']
best_units_vanilla = int(best_params_vanilla['units'])
best_learning_rate_vanilla = best_params_vanilla['learning_rate']

# Create the Bayesian Optimization object for Stacked LSTM
optimizer_stacked = BayesianOptimization(f=lambda units, learning_rate: lstm_objective(units, learning_rate, 'stacked'), pbounds=pbounds

# Perform Bayesian Optimization for Stacked LSTM
optimizer_stacked.maximize(init_points=5, n_iter=10)

# Get the best hyperparameters for Stacked LSTM
best_params_stacked = optimizer_stacked.max['params']
best_units_stacked = int(best_params_stacked['units'])
best_learning_rate_stacked = best_params_stacked['learning_rate']
```

```
|   iter    |  target   | learni... |   units   |
-------------------------------------------------
| 1         | -0.2963   | 0.003752  | 95.56     |
| 2         | -0.2963   | 0.007323  | 63.88     |
| 3         | -0.2963   | 0.001569  | 24.04     |
| 4         | -0.2963   | 0.0005903 | 87.96     |
| 5         | -0.2963   | 0.006015  | 73.73     |
| 6         | -0.2963   | 0.007658  | 10.0      |
| 7         | -0.2963   | 0.008286  | 37.02     |
| 8         | -0.2963   | 0.001037  | 10.01     |
| 9         | -0.2963   | 0.00989   | 99.99     |
| 10        | -0.2963   | 0.005792  | 10.01     |
| 11        | -0.2963   | 0.008868  | 99.99     |
| 12        | -0.2963   | 0.004     | 10.0      |
| 13        | -0.2963   | 0.008413  | 100.0     |
| 14        | -0.2963   | 0.002563  | 10.0      |
| 15        | -0.2963   | 0.006261  | 99.99     |
=================================================
|   iter    |  target   | learni... |   units   |
-------------------------------------------------
| 1         | -0.2963   | 0.003752  | 95.56     |
| 2         | -0.2963   | 0.007323  | 63.88     |
| 3         | -0.2963   | 0.001569  | 24.04     |
| 4         | -0.2963   | 0.0005903 | 87.96     |
| 5         | -0.2963   | 0.006015  | 73.73     |
| 6         | -0.2963   | 0.007658  | 10.0      |
| 7         | -0.2963   | 0.008286  | 37.02     |
| 8         | -0.2963   | 0.001037  | 10.01     |
| 9         | -0.2963   | 0.00989   | 99.99     |
| 10        | -0.2963   | 0.005792  | 10.01     |
| 11        | -0.2963   | 0.008868  | 99.99     |
| 12        | -0.2963   | 0.004     | 10.0      |
| 13        | -0.2963   | 0.008413  | 100.0     |
| 14        | -0.2963   | 0.002563  | 10.0      |
| 15        | -0.2963   | 0.006261  | 99.99     |
=================================================
```

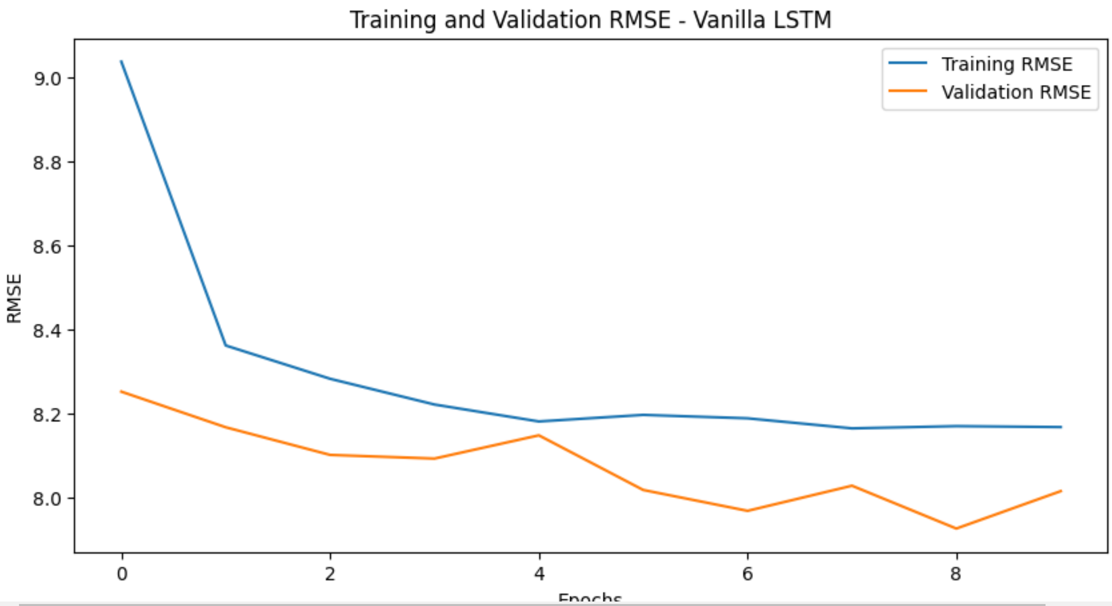
```
def create_vanilla_lstm(units, learning_rate):
    model = Sequential()
    model.add(LSTM(units=units, input_shape=(X_train.shape[1], X_train.shape[2])))
    model.add(Dense(1))
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate), loss='mean_squared_error')
    return model

# Train Vanilla LSTM with best hyperparameters
best_vanilla_lstm = create_vanilla_lstm(best_units_vanilla, best_learning_rate_vanilla)
history_vanilla = best_vanilla_lstm.fit(X_train, y_train, epochs=10, validation_data=(X_val, y_val))
```

```
Epoch 1/10
1807/1807 [==============================] - 15s 7ms/step - loss: 83.8199 - val_loss: 70.9086
Epoch 2/10
1807/1807 [==============================] - 9s 5ms/step - loss: 69.6862 - val_loss: 65.7770
Epoch 3/10
1807/1807 [==============================] - 9s 5ms/step - loss: 67.7268 - val_loss: 64.9420
Epoch 4/10
1807/1807 [==============================] - 10s 6ms/step - loss: 67.3844 - val_loss: 65.7781
Epoch 5/10
1807/1807 [==============================] - 8s 5ms/step - loss: 67.3278 - val_loss: 65.0729
Epoch 6/10
1807/1807 [==============================] - 9s 5ms/step - loss: 67.6657 - val_loss: 65.4637
Epoch 7/10
```

```
1807/1807 [==============================] - 9s 5ms/step - loss: 67.1096 - val_loss: 64.1356
Epoch 8/10
1807/1807 [==============================] - 10s 5ms/step - loss: 66.7370 - val_loss: 64.3147
Epoch 9/10
1807/1807 [==============================] - 9s 5ms/step - loss: 66.6873 - val_loss: 67.1603
Epoch 10/10
1807/1807 [==============================] - 9s 5ms/step - loss: 66.7942 - val_loss: 64.2314
```

```
from tensorflow.keras.metrics import RootMeanSquaredError
def create_vanilla_lstm(units, learning_rate):
    model = Sequential()
    model.add(LSTM(units=units, input_shape=(X_train.shape[1], X_train.shape[2])))
    model.add(Dense(1))
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate), loss='mean_squared_error', metrics=[RootMeanSquaredEr
    return model

# Train Vanilla LSTM with best hyperparameters
best_vanilla_lstm = create_vanilla_lstm(best_units_vanilla, best_learning_rate_vanilla)
history_vanilla = best_vanilla_lstm.fit(X_train, y_train, epochs=10, validation_data=(X_val, y_val))

# Evaluate Vanilla LSTM RMSE
train_rmse = history_vanilla.history['root_mean_squared_error'][-1]
val_rmse = history_vanilla.history['val_root_mean_squared_error'][-1]
print(f"Vanilla LSTM Model - Training RMSE: {train_rmse:.4f}, Validation RMSE: {val_rmse:.4f}")

# Plot training and validation loss
plt.figure(figsize=(10, 5))
plt.plot(history_vanilla.history['root_mean_squared_error'], label='Training RMSE')
plt.plot(history_vanilla.history['val_root_mean_squared_error'], label='Validation RMSE')
plt.title('Training and Validation RMSE - Vanilla LSTM')
plt.xlabel('Epochs')
plt.ylabel('RMSE')
plt.legend()
plt.show()
```

```
Epoch 1/10
1807/1807 [==============================] - 16s 7ms/step - loss: 81.6699 - root_mean_squared_error: 9.0371 - val_loss: 68.1099 - va
Epoch 2/10
1807/1807 [==============================] - 9s 5ms/step - loss: 69.9329 - root_mean_squared_error: 8.3626 - val_loss: 66.7198 - val
Epoch 3/10
1807/1807 [==============================] - 9s 5ms/step - loss: 68.6159 - root_mean_squared_error: 8.2835 - val_loss: 65.6508 - val
Epoch 4/10
1807/1807 [==============================] - 9s 5ms/step - loss: 67.6058 - root_mean_squared_error: 8.2223 - val_loss: 65.5093 - val
Epoch 5/10
1807/1807 [==============================] - 9s 5ms/step - loss: 66.9456 - root_mean_squared_error: 8.1820 - val_loss: 66.4073 - val
Epoch 6/10
1807/1807 [==============================] - 17s 9ms/step - loss: 67.2012 - root_mean_squared_error: 8.1976 - val_loss: 64.3078 - va
Epoch 7/10
1807/1807 [==============================] - 11s 6ms/step - loss: 67.0674 - root_mean_squared_error: 8.1895 - val_loss: 63.5140 - va
Epoch 8/10
1807/1807 [==============================] - 9s 5ms/step - loss: 66.6775 - root_mean_squared_error: 8.1656 - val_loss: 64.4698 - val
Epoch 9/10
1807/1807 [==============================] - 10s 5ms/step - loss: 66.7634 - root_mean_squared_error: 8.1709 - val_loss: 62.8446 - va
Epoch 10/10
1807/1807 [==============================] - 9s 5ms/step - loss: 66.7268 - root_mean_squared_error: 8.1686 - val_loss: 64.2625 - val
Vanilla LSTM Model - Training RMSE: 8.1686, Validation RMSE: 8.0164
```



```
def create_stacked_lstm(units, learning_rate):
    model = Sequential()
    model.add(LSTM(units=units, input_shape=(X_train.shape[1], X_train.shape[2]), return_sequences=True))
    model.add(LSTM(units=units))
    model.add(Dense(1))  # Assuming a regression task, adjust accordingly for classification
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate), loss='mean_squared_error')
    return model

# Train Stacked LSTM with best hyperparameters
best_stacked_lstm = create_stacked_lstm(best_units_stacked, best_learning_rate_stacked)
history_stacked = best_stacked_lstm.fit(X_train, y_train, epochs=10, validation_data=(X_val, y_val))
```

```
Epoch 1/10
1807/1807 [==============================] - 22s 10ms/step - loss: 82.1041 - val_loss: 67.3078
Epoch 2/10
1807/1807 [==============================] - 17s 9ms/step - loss: 70.8658 - val_loss: 68.8142
Epoch 3/10
1807/1807 [==============================] - 29s 16ms/step - loss: 69.0221 - val_loss: 65.0925
Epoch 4/10
1807/1807 [==============================] - 24s 13ms/step - loss: 69.4300 - val_loss: 65.0790
Epoch 5/10
1807/1807 [==============================] - 16s 9ms/step - loss: 68.0207 - val_loss: 67.8962
Epoch 6/10
1807/1807 [==============================] - 21s 11ms/step - loss: 68.0185 - val_loss: 65.7711
Epoch 7/10
1807/1807 [==============================] - 16s 9ms/step - loss: 67.8976 - val_loss: 64.5214
Epoch 8/10
1807/1807 [==============================] - 16s 9ms/step - loss: 67.3712 - val_loss: 64.5356
Epoch 9/10
1807/1807 [==============================] - 16s 9ms/step - loss: 67.6187 - val_loss: 65.1498
Epoch 10/10
1807/1807 [==============================] - 16s 9ms/step - loss: 67.2706 - val_loss: 67.2565
```

```
# Function to create and compile Stacked LSTM model with RMSE
def create_stacked_lstm(units, learning_rate):
    model = Sequential()
    model.add(LSTM(units=units, input_shape=(X_train.shape[1], X_train.shape[2]), return_sequences=True))
    model.add(LSTM(units=units))
    model.add(Dense(1))
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate), loss='mean_squared_error', metrics=[RootMeanSquaredEr
    return model

# Training Stacked LSTM with best hyperparameters
best_stacked_lstm = create_stacked_lstm(best_units_stacked, best_learning_rate_stacked)
history_stacked = best_stacked_lstm.fit(X_train, y_train, epochs=10, validation_data=(X_val, y_val))

# Evaluating Stacked LSTM RMSE
train_rmse_stacked = history_stacked.history['root_mean_squared_error'][-1]
val_rmse_stacked = history_stacked.history['val_root_mean_squared_error'][-1]
print(f"Stacked LSTM Model - Training RMSE: {train_rmse_stacked:.4f}, Validation RMSE: {val_rmse_stacked:.4f}")

# Ploting training and validation loss
plt.figure(figsize=(10, 5))
plt.plot(history_stacked.history['root_mean_squared_error'], label='Training RMSE')
plt.plot(history_stacked.history['val_root_mean_squared_error'], label='Validation RMSE')
plt.title('Training and Validation RMSE - Stacked LSTM')
plt.xlabel('Epochs')
plt.ylabel('RMSE')
plt.legend()
plt.show()
```