In [4]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from statsmodels.tsa.seasonal import seasonal_decompose
import csv
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import LabelBinarizer
from pandas import Timestamp
import datetime
import seaborn as sns
from pandas.tseries.holiday import USFederalHolidayCalendar as calendar

import pickle
```

Importing the Dataset

In [5]:
```
dataset = pd.read_csv('data_daily.csv')
dataset.tail()
```

Out[5]:

	# Date	Receipt_Count
360	2021-12-27	10350408
361	2021-12-28	10219445
362	2021-12-29	10313337
363	2021-12-30	10310644
364	2021-12-31	10211187

In [6]:
```
dataset.describe()
```

Out[6]:

	Receipt_Count
count	3.650000e+02
mean	8.826566e+06
std	7.820089e+05
min	7.095414e+06
25%	8.142874e+06
50%	8.799249e+06
75%	9.476970e+06
max	1.073886e+07

Checking for null values

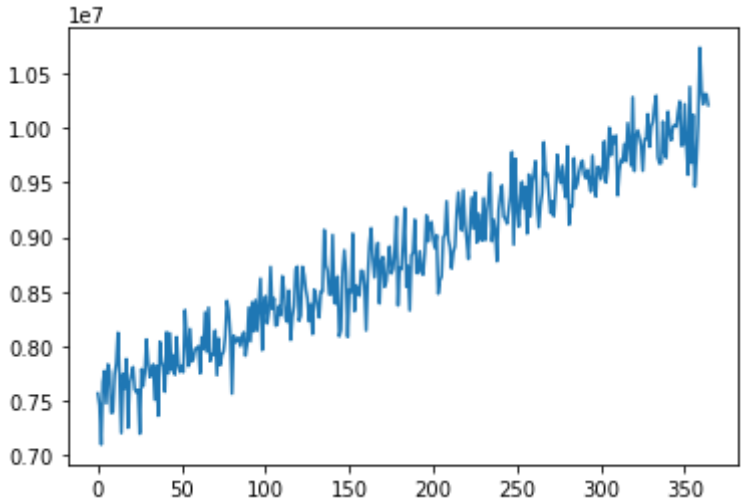
In [7]:
```
dataset.isna().sum().sum()
```

Out[7]: 0

Visualizing the data

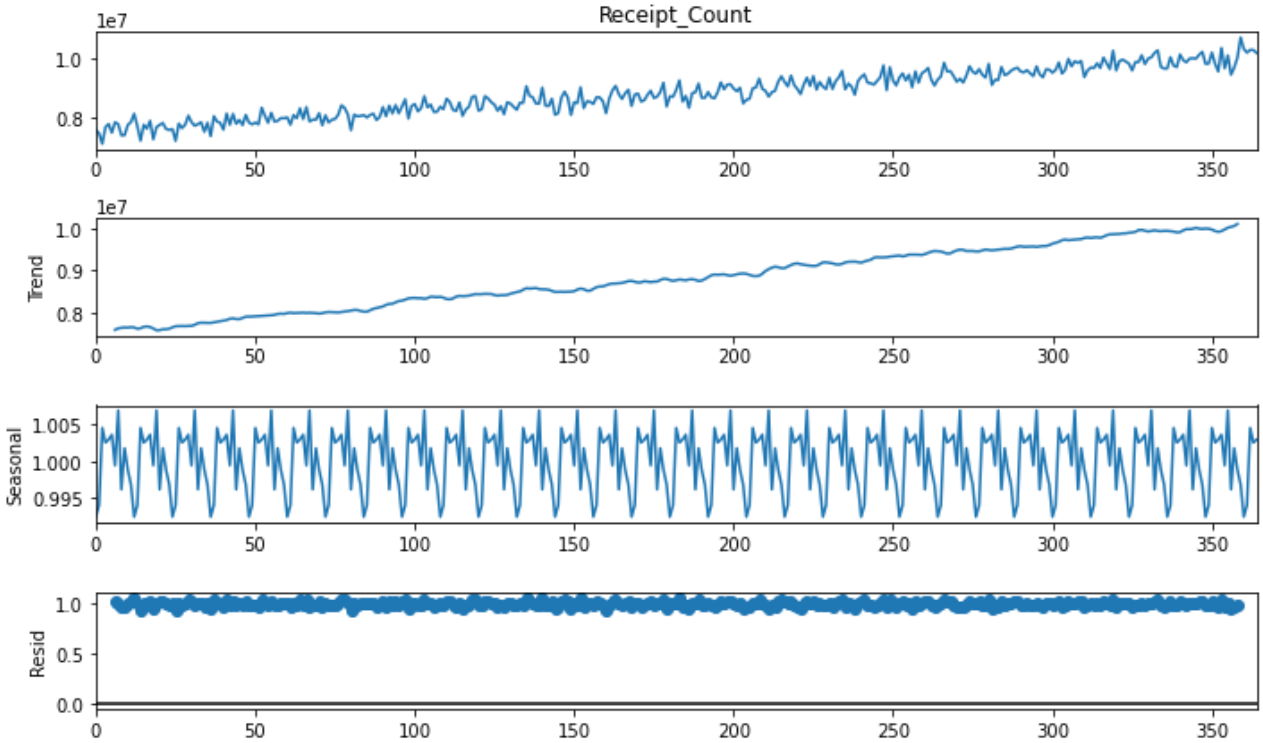
In [8]:
```
plt.plot(dataset.iloc[:, 1])
```

Out[8]:
```
[<matplotlib.lines.Line2D at 0x7f87ea482b50>]
```



Visualizing the data using time-series decomposition

In [9]:
```
# decompose time series
plt.rcParams["figure.figsize"] = (10,6)
result = seasonal_decompose(dataset['Receipt_Count'], model='multiplicative', pe
result.plot()
plt.show()
```



Data preprocessing

In [10]:

```
temp = dataset['# Date']
dataset['# Date'] = pd.to_datetime(dataset['# Date'])
```

Creating weekday and month columns from # Date

In [11]:

```
week = ["monday", "tuesday", "wednesday", "thursday", "friday", "saturday", "sun




weekd = []
for i in range(len(dataset.iloc[:, :])):
    date = dataset.iloc[i, :][0]

    weekd.append(date.weekday())

dataset.insert(2,'weekday', weekd, True)
encoder = LabelBinarizer()

month = []
for i in range(len(dataset.iloc[:, :])):
    date = dataset.iloc[i, :][0]
dataset['month'] = pd.DatetimeIndex(dataset['# Date']).month

dataset.head()
```

Out[11]:

	# Date	Receipt_Count	weekday	month
0	2021-01-01	7564766	4	1
1	2021-01-02	7455524	5	1
2	2021-01-03	7095414	6	1
3	2021-01-04	7666163	0	1
4	2021-01-05	7771289	1	1

Feature engineering to extract more data from the dates column

In [12]:

```
cal = calendar()
holidays = cal.holidays(start=dataset['# Date'].min(), end=dataset['# Date'].max
holidays
```

Out[12]:

```
DatetimeIndex(['2021-01-01', '2021-01-18', '2021-02-15', '2021-05-31',
               '2021-07-05', '2021-09-06', '2021-10-11', '2021-11-11',
               '2021-11-25', '2021-12-24', '2021-12-31'],
              dtype='datetime64[ns]', freq=None)
```

Creating a holidays column

In [13]:
```
dataset['holiday'] = dataset['# Date'].isin(holidays)
```

In [14]:
```
dataset.head()
```

Out[14]:

	# Date	Receipt_Count	weekday	month	holiday
0	2021-01-01	7564766	4	1	True
1	2021-01-02	7455524	5	1	False
2	2021-01-03	7095414	6	1	False
3	2021-01-04	7666163	0	1	False
4	2021-01-05	7771289	1	1	False

Creating columns for days from next holiday and days from prev holiday

In [15]:
```
# funciton code from medium website by author: Naina Chaturvedi
def days_prev_holiday(date, holidays):
    difference=[]
    for item in holidays:
        difference.append(int(((item-date)).days))
    return abs(max([x for x in difference if x<=0]))
def days_next_holiday(date, holidays):
    difference=[]
    for item in holidays:
        difference.append(int(str((item-date).days)))
    return min([x for x in difference if x>=0])
```

In [16]:
```
dataset['days_previous_holiday']= dataset.apply(lambda row: days_prev_holiday((r
dataset['days_next_holiday']= dataset.apply(lambda row: days_next_holiday((row['
```

In [17]:
```
dataset['holiday'] = np.array(encoder.fit_transform(dataset['holiday']))
dataset.head()
```

Out[17]:

	# Date	Receipt_Count	weekday	month	holiday	days_previous_holiday	days_next_holiday
0	2021-01-01	7564766	4	1	1	0	0
1	2021-01-02	7455524	5	1	0	1	16
2	2021-01-03	7095414	6	1	0	2	15
3	2021-01-04	7666163	0	1	0	3	14
4	2021-01-05	7771289	1	1	0	4	13

Encoding the data using OneHotEncoder

In [18]:

```
enc=OneHotEncoder()


enc_data=pd.DataFrame(enc.fit_transform(dataset[['weekday']]).toarray())

df=dataset.join(enc_data)
df = df.rename(columns={0: 'M', 1: 'Tu', 2: 'W', 3: 'Th', 4: 'F', 5: 'Sa', 6: 'S

enc_data=pd.DataFrame(enc.fit_transform(df[['month']]).toarray())

df=df.join(enc_data)
```

In [19]:

```
df['# Date']=df['# Date'].map(datetime.datetime.toordinal)
df.head()
```

Out[19]:

	# Date	Receipt_Count	weekday	month	holiday	days_previous_holiday	days_next_holiday	M
0	737791	7564766	4	1	1	0	0	0.0
1	737792	7455524	5	1	0	1	16	0.0
2	737793	7095414	6	1	0	2	15	0.0
3	737794	7666163	0	1	0	3	14	1.0
4	737795	7771289	1	1	0	4	13	0.0

5 rows × 26 columns

Splitting the data into testing and training sets

In [33]:

```
from sklearn.model_selection import train_test_split
X = df[['# Date', 'holiday','days_previous_holiday', 'days_next_holiday','month'
y = df['Receipt_Count']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
```

In [34]:

```
df.corr()['Receipt_Count']
```

Out[34]:

```
# Date                   0.960658
Receipt_Count            1.000000
weekday                 -0.005646
month                    0.957785
holiday                  0.033517
days_previous_holiday   -0.160776
days_next_holiday       -0.418415
M                        0.010168
Tu                      -0.017177
W                        0.006578
Th                       0.007219
F                        0.000666
Sa                       0.007838
Su                      -0.015298
```

```
0                       -0.464196
1                       -0.357376
2                       -0.314796
3                       -0.180769
4                       -0.131781
5                       -0.052883
6                        0.014503
7                        0.129866
8                        0.208832
9                        0.281154
10                       0.399651
11                       0.457127
Name: Receipt_Count, dtype: float64
```

Scaling the X values for better prediction

In [35]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)
```

Creating the model class

In [36]:

```
class MultipleLinearRegression:
    def __init__(self):
        self.beta = None

    def fit(self, X, y):
        # for bias
        ones = np.ones((X.shape[0], 1))
        X = np.concatenate((ones, X), axis=1)

        # calculate coefficients using normal equations
        X_transpose = np.transpose(X)
        beta = np.linalg.inv(X_transpose @ X) @ X_transpose @ y
        self.beta = beta

    def predict(self, X):
        # to account for bias
        ones = np.ones((X.shape[0], 1))

        X = np.concatenate((ones, X), axis=1)
        y_pred = X @ self.beta

        return y_pred
```

Fit the model

In [37]:

```
regressor = MultipleLinearRegression()
regressor.fit(X_train, y_train)
```

Checking model accuracy using R^2

In [38]:
```
y_pred = regressor.predict(X_test)
y_mean = np.mean(y_test)
ss_tot = np.sum((y_test - y_mean)**2)
ss_res = np.sum((y_test - y_pred)**2)
r2 = 1 - (ss_res / ss_tot)
print("R-squared:", r2)
```

```
R-squared: 0.9245610773495255
```

Storing the model to be used by the web app

In [39]:
```
pickle.dump(regressor, open('model2.pkl','wb'))
```

In [40]:
```
model = pickle.load(open('model2.pkl','rb'))
print(model.predict(np.array([[737791, 1, 0, 0, 4]])))
```

```
[7599193.48061987]
```

Visualizing the predicting against the real values

In [41]:
```
plt.scatter(X_test["# Date"], y_test,color='red')
plt.scatter(X_test["# Date"], y_pred,color='blue')
```

Out[41]:
```
<matplotlib.collections.PathCollection at 0x7f87ef75fac0>
```