8.2 Exercise: Time Series Modeling

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# importing warning to surpress the future warnings
import warnings
warnings.simplefilter(action='ignore')
```

This data gives the total monthly retail sales in the US from January 1992 until June 2021. With this dataset, complete the following steps:

1. Plot the data with proper labeling and make some observations on the graph.

In [2]:
```
# reading and saving the csv
df = pd.read_csv('Datasets/us_retail_sales.csv')
```

In [3]:
```
# reviewing the last 5 rows
df.tail(5)
```

Out[3]:

	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC
25	2017	416081	415503	414620	416889	414540	416505	416744.0	417179.0	426501.0	426933.0	431158.0	433282.0
26	2018	432148	434106	433232	435610	439996	438191	440703.0	439278.0	438985.0	444038.0	445242.0	434803.0
27	2019	440751	439996	447167	448709	449552	450927	454012.0	456500.0	452849.0	455486.0	457658.0	458055.0
28	2020	460586	459610	434281	379892	444631	476343	481627.0	483716.0	493327.0	493991.0	488652.0	484782.0
29	2021	520162	504458	559871	562269	548987	550782	NaN	NaN	NaN	NaN	NaN	NaN

In [4]:
```
# reviewing first 5 rows
df.head(5)
```

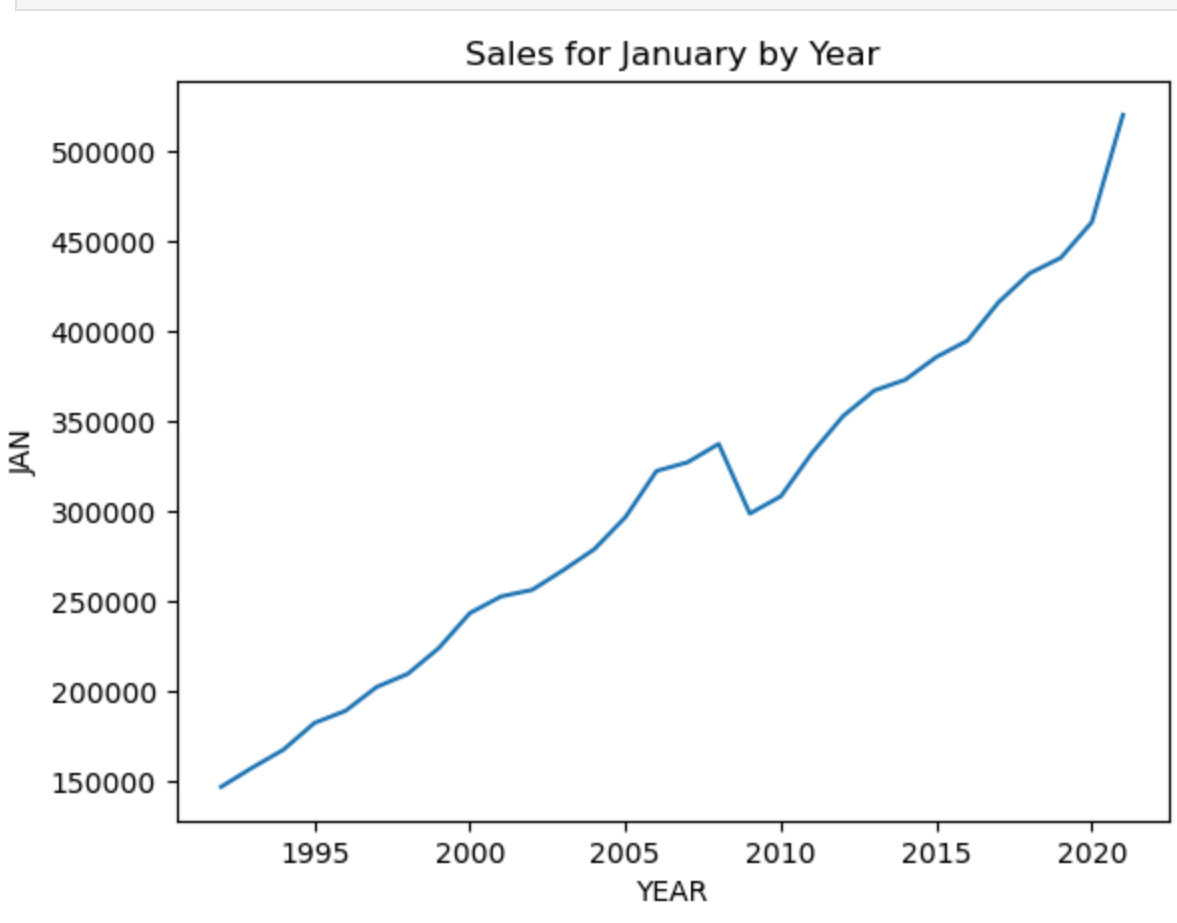
Out[4]:

	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC
0	1992	146925	147223	146805	148032	149010	149800	150761.0	151067.0	152588.0	153521.0	153583.0	155614.0
1	1993	157555	156266	154752	158979	160605	160127	162816.0	162506.0	163258.0	164685.0	166594.0	168161.0
2	1994	167518	169649	172766	173106	172329	174241	174781.0	177295.0	178787.0	180561.0	180703.0	181524.0
3	1995	182413	179488	181013	181686	183536	186081	185431.0	186806.0	187366.0	186565.0	189055.0	190774.0
4	1996	189135	192266	194029	194744	196205	196136	196187.0	196218.0	198859.0	200509.0	200174.0	201284.0

In [5]:
```
# reviewing the shape of the dataframe
df.shape
```

Out[5]:
```
(30, 13)
```

In [6]:
```
# creating a lineplot of January sales by year
sns.lineplot(data=df, x="YEAR", y='JAN')
plt.title('Sales for January by Year')
plt.show()
```



2. Split this data into a training and test set. Use the last year of data (July 2020 – June 2021) of data as your test set and the rest as your training set.

In [7]:
```
# using the melt function in the column year to unpivote the remaining columns, then displaying the first 5 rows
melted_df = pd.melt(df, id_vars=['YEAR'])
melted_df.head(5)
```

Out[7]:

	YEAR	variable	value
0	1992	JAN	146925.0
1	1993	JAN	157555.0
2	1994	JAN	167518.0
3	1995	JAN	182413.0
4	1996	JAN	189135.0

In [8]:
```
# Renaming the un-pivoted columns for visivilty and ease of use, then displaying the first 5 rows
melted_df.rename(columns={'YEAR': 'Year', 'variable': 'Month', 'value': 'Sales'}, inplace=True)
melted_df.head(5)
```

Out[8]:

	Year	Month	Sales
0	1992	JAN	146925.0
1	1993	JAN	157555.0
2	1994	JAN	167518.0
3	1995	JAN	182413.0
4	1996	JAN	189135.0

In [9]:
```
# importing additional libraries
from time import strptime
```

In [10]:
```
# Creating a new column with the month number
melted_df['month_number'] = [strptime(str(x), '%b').tm_mon for x in melted_df['Month']]
```

In [11]:
```
# Dropping the old Month column
melted_df.drop('Month', axis = 1, inplace = True)
```

In [12]:
```
# Re-naming the month number column
melted_df.rename(columns={'month_number': 'Month'}, inplace=True)
```

In [13]:
```
# converting the two columnd to a single datetime column and sorting
melted_df['DATE'] = pd.to_datetime(melted_df[['Year', 'Month']].assign(DAY=1))
melted_df.sort_values('DATE').head(5)
```

Out[13]:

	Year	Sales	Month	DATE
0	1992	146925.0	1	1992-01-01
30	1992	147223.0	2	1992-02-01
60	1992	146805.0	3	1992-03-01
90	1992	148032.0	4	1992-04-01
120	1992	149010.0	5	1992-05-01

In [14]:
```
melted_df
```

Out[14]:

	Year	Sales	Month	DATE
0	1992	146925.0	1	1992-01-01
1	1993	157555.0	1	1993-01-01
2	1994	167518.0	1	1994-01-01
3	1995	182413.0	1	1995-01-01
4	1996	189135.0	1	1996-01-01
...	...	...	...	...
355	2017	433282.0	12	2017-12-01
356	2018	434803.0	12	2018-12-01
357	2019	458055.0	12	2019-12-01
358	2020	484782.0	12	2020-12-01
359	2021	NaN	12	2021-12-01

360 rows × 4 columns

In [15]:
```
# setting the date column as the new index
melted_df.set_index('DATE', inplace=True)
```

In [16]:
```
# melted_df['DATE'] = melted_df['DATE'].asfreq('y')
```

In [17]:
```
# sorting by date
melted_df.sort_values('DATE')
```

Out[17]:

DATE	Year	Sales	Month
1992-01-01	1992	146925.0	1
1992-02-01	1992	147223.0	2
1992-03-01	1992	146805.0	3
1992-04-01	1992	148032.0	4
1992-05-01	1992	149010.0	5
...	...	...	...
2021-08-01	2021	NaN	8
2021-09-01	2021	NaN	9
2021-10-01	2021	NaN	10
2021-11-01	2021	NaN	11
2021-12-01	2021	NaN	12

360 rows × 3 columns

In [18]:
```
# dropping columns no longer needed
melted_df.drop(['Month', 'Year'], axis = 1, inplace = True)
```

In [19]:
```
# creating a test data frame in the specified date range
test_df = melted_df.sort_index().loc['2020-07-01':'2021-06-01']
test_df
```

Out[19]:

DATE	Sales
2020-07-01	481627.0
2020-08-01	483716.0
2020-09-01	493327.0
2020-10-01	493991.0
2020-11-01	488652.0
2020-12-01	484782.0
2021-01-01	520162.0
2021-02-01	504458.0
2021-03-01	559871.0
2021-04-01	562269.0
2021-05-01	548987.0
2021-06-01	550782.0

In [20]:
```
# creating a train data frame with specified date range
train_df = melted_df.sort_index().loc['1992-01-01':'2020-06-01']
train_df
```

Out[20]:

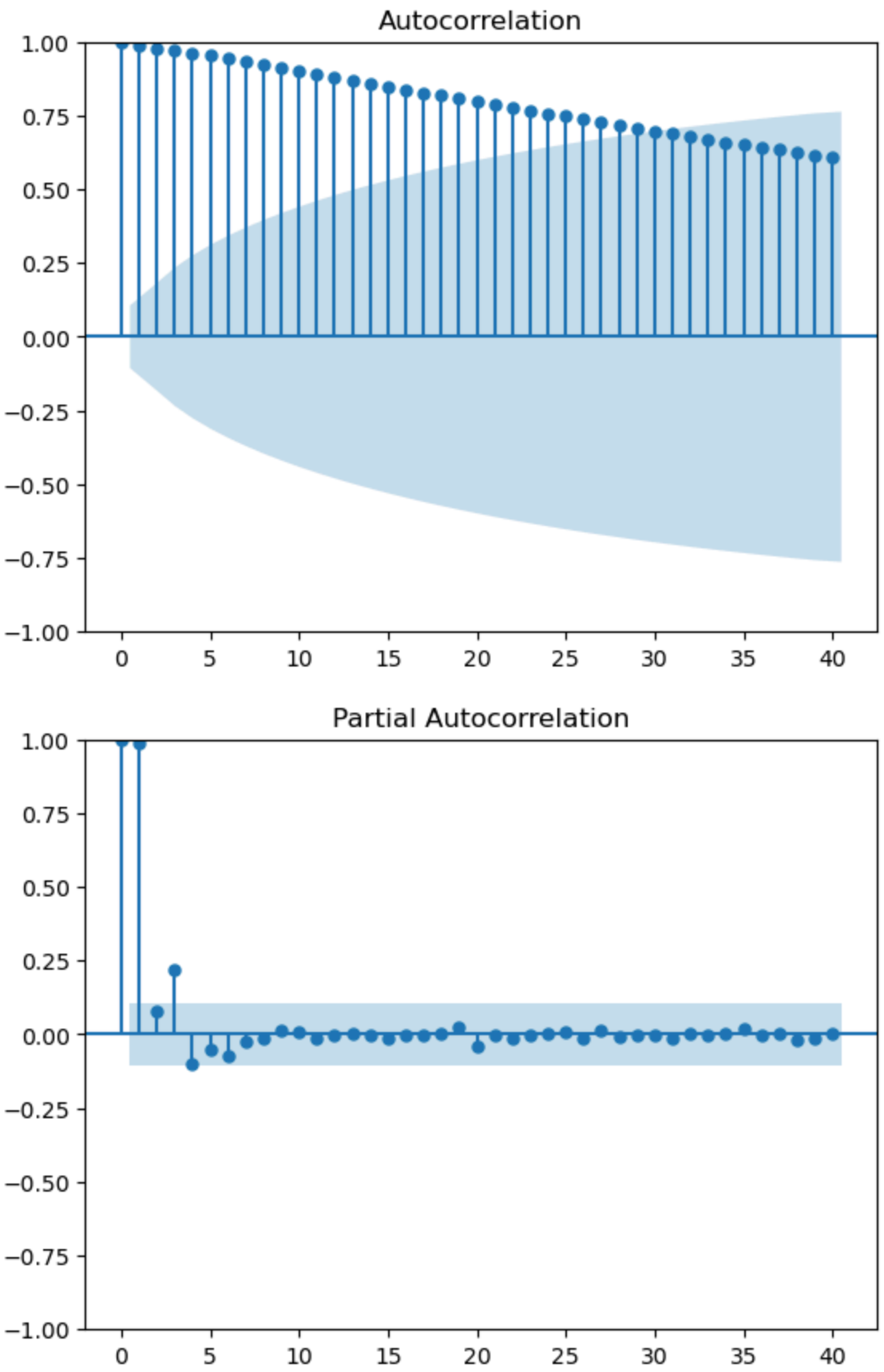
DATE	Sales
1992-01-01	146925.0
1992-02-01	147223.0
1992-03-01	146805.0
1992-04-01	148032.0
1992-05-01	149010.0
...	...
2020-02-01	459610.0
2020-03-01	434281.0
2020-04-01	379892.0
2020-05-01	444631.0
2020-06-01	476343.0

342 rows × 1 columns

3. Use the training set to build a predictive model for the monthly retail sales.

In [21]:
```
# import additional libraries
from statsmodels.tsa.arima.model import ARIMA
```

In [22]:
```
# ploting auto and partial correlation
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
plot_acf(train_df['Sales'], lags=40)
plot_pacf(train_df['Sales'], lags=40)
plt.show()
```



In [23]:
```
# creating the ARIMA model on the training data
model = ARIMA(train_df['Sales'], order=(0, 1, 0))
model_fit = model.fit()
```

```
C:\Users\rbrio\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: No frequency information was provid
ed, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
C:\Users\rbrio\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: No frequency information was provid
ed, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
C:\Users\rbrio\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: No frequency information was provid
ed, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
```

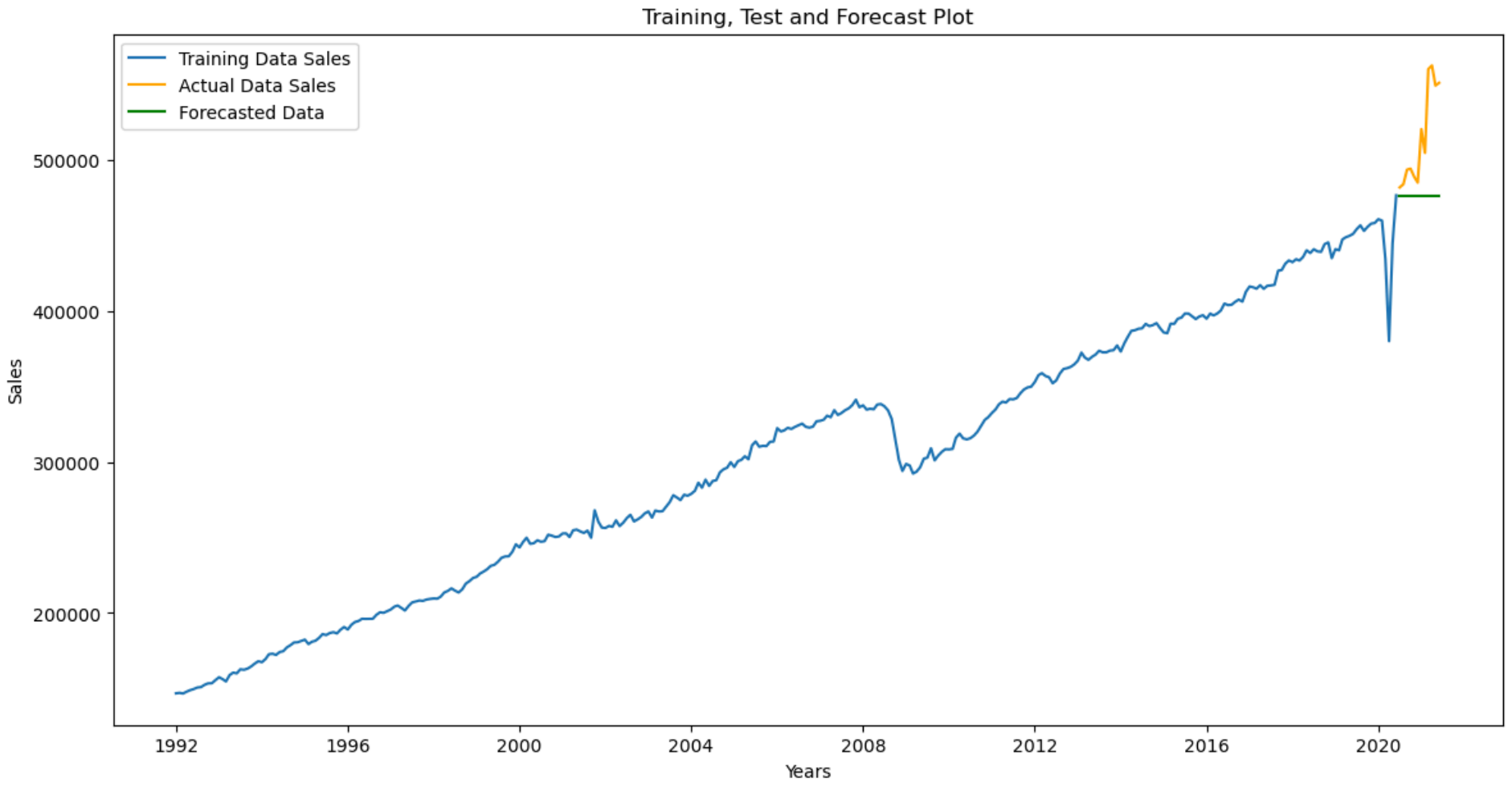
In [24]:
```
# creating the forecast
forecast = model_fit.get_forecast(steps=30)
```

4. Use the model to predict the monthly retail sales on the last year of data.

In [25]:
```
# Forecast on the test dataset
test_forecast = model_fit.get_forecast(steps=len(test_df))
test_forecast_series = pd.Series(test_forecast.predicted_mean, index=test_df.index)
```

In [26]:
```
# plotting the training, testing, and forecast data
plt.figure(figsize=(14,7))
plt.plot(train_df['Sales'], label='Training Data Sales')
plt.plot(test_df['Sales'], label='Actual Data Sales', color='orange')
plt.plot(test_forecast_series, label='Forecasted Data', color='green')
plt.xlabel('Years')
plt.ylabel('Sales')
plt.title('Training, Test and Forecast Plot')
```

```
plt.legend()
plt.show()
```



The forecatse unfortunatley looks flat compared to the actual sales number, I'm not 100% sure where I went wrong so any feedback would be greatly appreciated. I'm wondering if this has to do with the index frequency error?

5. Report the RMSE of the model predictions on the test set.

In [27]:
```
from sklearn.metrics import mean_squared_error
```

In [28]:
```
# Calculate the mean squared error
mse = mean_squared_error(test_df['Sales'], test_forecast_series)
rmse = mse**0.5
```

In [29]:
```
rmse
```

Out[29]: 48984.68795620389