Data

Going to start by importing two data sets, the first is approximately two years worth of transactions from a recently married couple and the second dataset is approximately five years worth of sales from a small business. These sets will be cleaned and examined and then used to create machine learning models with the goal of the end user being able to use the results to help manage their finances in some innovative ways.

In [30]:

```
#importing libraries to read the files
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')

#visualization libraries
import matplotlib.pyplot as plt
import seaborn as sns
from time import gmtime, strftime
from pylab import rcParams

#importing libraries to be used in model building
import statsmodels.api as sm
import itertools
from statsmodels.tsa.statespace.sarimax import SARIMAX
from itertools import product
from statsmodels.tsa.seasonal import seasonal_decompose

from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.graphics.tsaplots import plot_acf

from statsmodels.tsa.holtwinters import ExponentialSmoothing
from statsmodels.tsa.stattools import adfuller

from tqdm import tqdm_notebook
from itertools import product

%matplotlib inline
```

In [31]:

```
#Importing the personal transactions data...
import pandas as pd
df_transactions = pd.read_csv('/Users/natashawyatt/Documents/personal_transac
```

In the next few cells we will be taking a closer look at the details of this dataframe to see what needs to be cleaned and what it contains.

In [32]:
```
df_transactions
```

Out[32]:

	Date	Description	Amount	Transaction Type	Category	Account_Name
0	01/01/2018	Amazon	11.11	debit	Shopping	Platinum Card
1	01/02/2018	Mortgage Payment	1247.44	debit	Mortgage & Rent	Checking
2	01/02/2018	Thai Restaurant	24.22	debit	Restaurants	Silver Card
3	01/03/2018	Credit Card Payment	2298.09	credit	Credit Card Payment	Platinum Card
4	01/04/2018	Netflix	11.76	debit	Movies & DVDs	Platinum Card
...	...	...	...	...	...	...
801	09/27/2019	Biweekly Paycheck	2250.00	credit	Paycheck	Checking
802	09/28/2019	BP	33.46	debit	Gas & Fuel	Platinum Card
803	09/28/2019	Sheetz	4.27	debit	Gas & Fuel	Platinum Card
804	09/30/2019	Starbucks	1.75	debit	Coffee Shops	Platinum Card
805	09/30/2019	Internet Service Provider	75.00	debit	Internet	Checking

806 rows × 6 columns

In [33]:
```
df_transactions.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 806 entries, 0 to 805
Data columns (total 6 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Date              806 non-null    object
 1   Description       806 non-null    object
 2   Amount            806 non-null    float64
 3   Transaction Type  806 non-null    object
 4   Category          806 non-null    object
 5   Account_Name      806 non-null    object
dtypes: float64(1), object(5)
memory usage: 37.9+ KB
```

In [34]:

```python
print(df_transactions.isnull().sum())
```

```
Date                0
Description         0
Amount              0
Transaction Type    0
Category            0
Account_Name        0
dtype: int64
```

In [35]:

```python
df_transactions = df_transactions.drop_duplicates()
```

In [36]:

```python
df_transactions.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 806 entries, 0 to 805
Data columns (total 6 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Date              806 non-null    object
 1   Description       806 non-null    object
 2   Amount            806 non-null    float64
 3   Transaction Type  806 non-null    object
 4   Category          806 non-null    object
 5   Account_Name      806 non-null    object
dtypes: float64(1), object(5)
memory usage: 44.1+ KB
```

After a quick look there are no null values and no duplicates. Lets take a closer look at what is is the columns and how they can be used.

In [37]:

```
# Looking at the ends of the date range, just under 2 years worth of transact
print(df_transactions['Date'])
```

```
0      01/01/2018
1      01/02/2018
2      01/02/2018
3      01/03/2018
4      01/04/2018
          ...
801    09/27/2019
802    09/28/2019
803    09/28/2019
804    09/30/2019
805    09/30/2019
Name: Date, Length: 806, dtype: object
```

In [38]:

```
# Lets see what the descriptions of the transactions include...
print('************************************')
print(df_transactions['Description'].unique())
print('************************************')
print(df_transactions['Description'].nunique())
print('************************************')
```

```
***********************************
['Amazon' 'Mortgage Payment' 'Thai Restaurant' 'Credit Card Payment'
 'Netflix' 'American Tavern' 'Hardware Store' 'Gas Company' 'Spotify'
 'Phone Company' 'Shell' 'Grocery Store' 'Biweekly Paycheck' 'Pizza Place'
 'City Water Charges' 'Power Company' 'Starbucks'
 'Internet Service Provider' 'Brunch Restaurant' 'Japanese Restaurant'
 'Barbershop' 'Bojangles' 'Fancy Restaurant' 'Brewing Company'
 'Mexican Restaurant' 'Gas Station' 'BBQ Restaurant' 'BP'
 'Mediterranean Restaurant' 'Steakhouse' 'Belgian Restaurant' "Chili's"
 'Greek Restaurant' 'Amazon Video' 'Chevron' 'Tiny Deli' 'Irish Pub'
 'Blue Sky Market' 'State Farm' 'QuikTrip' "Mike's Construction Co."
 'Liquor Store' 'Movie Theater' 'Italian Restaurant' 'Chick-Fil-A'
 'Go Mart' 'Circle K' "Wendy's" 'Irish Restaurant' 'Conoco' 'Valero'
 'Sushi Restaurant' 'Exxon' 'German Restaurant' 'Seafood Restaurant'
 'Food Truck' 'Latin Restaurant' 'New York Deli' 'Roadside Diner'
 'Bakery Place' 'Best Buy' 'Vietnamese Restaurant' 'Target'
 'Hawaiian Grill' 'Sheetz']
***********************************
65
***********************************
```

In [39]:

```python
# Making sure there are no other types than debit/credit....
print('***********************************')
print(df_transactions['Transaction Type'].unique())
print('***********************************')
print(df_transactions['Transaction Type'].nunique())
print('***********************************')
```

```
***********************************
['debit' 'credit']
***********************************
2
***********************************
```

In [40]:

```python
# Number of different amounts
print('***********************************')
print(df_transactions['Amount'].nunique())
print('***********************************')
```

```
***********************************
454
***********************************
```

In [41]:

```python
# Seeing the different categories of transactions...
print('***********************************')
print(df_transactions['Category'].unique())
print('***********************************')
print(df_transactions['Category'].nunique())
print('***********************************')
```

```
**********************************
['Shopping' 'Mortgage & Rent' 'Restaurants' 'Credit Card Payment'
 'Movies & DVDs' 'Home Improvement' 'Utilities' 'Music' 'Mobile Phone'
 'Gas & Fuel' 'Groceries' 'Paycheck' 'Fast Food' 'Coffee Shops' 'Internet'
 'Haircut' 'Alcohol & Bars' 'Auto Insurance' 'Entertainment'
 'Food & Dining' 'Television' 'Electronics & Software']
**********************************
22
**********************************
```

In [42]:

```
# Looking at account names it looks like money is spent through either a chec
print('**********************************')
print(df_transactions['Account_Name'].unique())
print('**********************************')
print(df_transactions['Account_Name'].nunique())
print('**********************************')
```

```
**********************************
['Platinum Card' 'Checking' 'Silver Card']
**********************************
3
**********************************
```

In [43]:

```
# Lets make sure all debits and credits are what we want them to be....

# Ensure 'Amount' is numeric
df_transactions['Amount'] = pd.to_numeric(df_transactions['Amount'], errors='

# Aggregate transactions by 'Transaction Type' and 'Category'
aggregated_transactions = df_transactions.groupby(['Transaction Type', 'Categ

# Display aggregated transactions
print(aggregated_transactions)
print('**********************')
# List unique categories for 'Credit'
credit_categories = aggregated_transactions[aggregated_transactions['Transact

print('**********************')
print("Unique Categories under 'Credit':", credit_categories)
print('**********************')
```

```
   Transaction Type                 Category       sum  count
0            credit     Credit Card Payment  30519.76     72
1            credit                Paycheck  93750.00     46
2             debit          Alcohol & Bars    539.13     25
3             debit          Auto Insurance   1350.00     18
4             debit            Coffee Shops    115.54     31
5             debit     Credit Card Payment  33041.36     71
6             debit  Electronics & Software    719.00      4
7             debit           Entertainment      9.62      1
8             debit               Fast Food    330.63     16
9             debit           Food & Dining     77.75      2
10            debit              Gas & Fuel   1715.17     52
11            debit               Groceries   2795.21    105
12            debit                 Haircut    378.00     13
13            debit        Home Improvement  19092.87     36
14            debit                Internet   1570.88     21
15            debit            Mobile Phone   1680.40     21
16            debit          Mortgage & Rent  24754.50     21
17            debit          Movies & DVDs    222.19     18
18            debit                   Music    224.49     21
19            debit             Restaurants   2613.02     81
20            debit                Shopping   1973.24     60
21            debit              Television    104.78      8
22            debit               Utilities   2776.00     63
***********************
***********************
Unique Categories under 'Credit': ['Credit Card Payment' 'Paycheck']
***********************
```

In [44]:

```
# A more detailed inspection, to filter transactions that are 'credit' but no
non_income_credits = df_transactions[(df_transactions['Transaction Type'] ==
print(non_income_credits)
```

```
           Date          Description   Amount Transaction Type  \
3    01/03/2018  Credit Card Payment  2298.09           credit
13   01/12/2018     Biweekly Paycheck  2000.00           credit
20   01/19/2018     Biweekly Paycheck  2000.00           credit
22   01/22/2018  Credit Card Payment   554.99           credit
23   01/22/2018  Credit Card Payment   309.81           credit
..          ...                  ...      ...              ...
784  09/13/2019     Biweekly Paycheck  2250.00           credit
788  09/16/2019  Credit Card Payment    90.57           credit
790  09/17/2019  Credit Card Payment   186.13           credit
796  09/20/2019  Credit Card Payment     9.43           credit
801  09/27/2019     Biweekly Paycheck  2250.00           credit

                Category   Account_Name
3    Credit Card Payment  Platinum Card
13              Paycheck       Checking
20              Paycheck       Checking
22   Credit Card Payment  Platinum Card
23   Credit Card Payment    Silver Card
..                   ...            ...
784             Paycheck       Checking
788  Credit Card Payment    Silver Card
790  Credit Card Payment  Platinum Card
796  Credit Card Payment    Silver Card
801             Paycheck       Checking

[118 rows x 6 columns]
```

A Few changes

After looking at the results of these last few cells we see there is something that needs to be addressed to make sure optimal accuracy moving forward. Under 'Description' we see two different types of credits, paychecks and credit card payments. Obvioulsy paying cash towards a credit card is not income, but sometimes this is listed as such since paying the card down increases the balance you can spend with. In our case we want this to be listed as a debit and only income to be listed as a credit. This will change in the next cell.

In [68]:

```
# Making a copy to work with for our changes....
df_transactions_copy = df_transactions.copy()

# Identifying and correcting the misclassification
df_transactions.loc[df_transactions['Category'] == "Credit Card Payment", 'Tr

# Verify the change by checking if there are any 'credit' transactions left w
credit_card_payments_as_credits = df_transactions[(df_transactions['Category'
print("Credit card payments still classified as 'credit':", len(credit_card_p
```

```
Credit card payments still classified as 'credit': 0
```

In [69]:

```
# Updating this cell from early to double check our work on removing the CC p
# Aggregate transactions by 'Transaction Type' and 'Category'
aggregated_transactions_copy = df_transactions_copy.groupby(['Transaction Typ


# List unique categories for 'Credit'
credit_categories_copy = aggregated_transactions_copy[aggregated_transactions_

print('**********************')
print("Unique Categories under 'Credit':", credit_categories_copy)
print('**********************')
```

```
**********************
Unique Categories under 'Credit': ['Paycheck']
**********************
```

Ok that worked, so this dataset should be good to work with. Begining some EDA, visuals and then models.

In [70]:

```
def calculate_debit_spending_percentage_by_category(df):

    # Filter for debit transactions only
    debit_transactions = df[df['Transaction Type'] == 'debit']

    # Ensure 'Amount' is numeric and calculate the total debit spending
    debit_transactions['Amount'] = pd.to_numeric(debit_transactions['Amount']
    total_debit_spending = debit_transactions['Amount'].sum()

    # Aggregate debit spending by category
    spending_by_category = debit_transactions.groupby('Category')['Amount'].s

    # Calculate the percentage of total debit spending for each category
    spending_by_category['Debit Spending Percentage'] = (spending_by_category

    # Sort categories by debit spending percentage for better readability
    spending_by_category = spending_by_category.sort_values(by='Debit Spendin

    return spending_by_category

# Assuming df_transactions_copy is a copy of your transactions data where cor
category_debit_spending_percentage = calculate_debit_spending_percentage_by_c
category_debit_spending_percentage
```

Out[70]:

	Category	Amount	Debit Spending Percentage
3	Credit Card Payment	63561.12	50.204852
14	Mortgage & Rent	24754.50	19.552771
11	Home Improvement	19092.87	15.080834
9	Groceries	2795.21	2.207845
20	Utilities	2776.00	2.192672
17	Restaurants	2613.02	2.063939
18	Shopping	1973.24	1.558598
8	Gas & Fuel	1715.17	1.354757
13	Mobile Phone	1680.40	1.327293
12	Internet	1570.88	1.240787
1	Auto Insurance	1350.00	1.066321
4	Electronics & Software	719.00	0.567915
0	Alcohol & Bars	539.13	0.425841
10	Haircut	378.00	0.298570
6	Fast Food	330.63	0.261154
16	Music	224.49	0.177317
15	Movies & DVDs	222.19	0.175501
2	Coffee Shops	115.54	0.091261
19	Television	104.78	0.082762
7	Food & Dining	77.75	0.061412
5	Entertainment	9.62	0.007599

In [71]:

```
# We've got the breakdown of where money is spent, lets just see the total of
category_debit_spending_percentage['Amount'].sum()
```

Out[71]: 126603.53999999996

In [72]:

```
# Assuming df_transactions_copy is your DataFrame with transaction data
total_credits = df_transactions_copy[df_transactions_copy['Transaction Type']
print(f"Total Credits: {total_credits}")
```

```
Total Credits: 93750.0
```

In [73]:
```
# Creating a table of credit transactions summarized by category
credits_by_category = df_transactions_copy[df_transactions_copy['Transaction '

# Sorting the table by Total_Amount for better readability
credits_by_category = credits_by_category.sort_values(by='Total_Amount', asce

print(credits_by_category)
```

```
   Category  Total_Amount  Transaction_Count
0  Paycheck       93750.0                 46
```

In [74]:
```
print('--------------------------------')
print('************   Debits Break Down   *******************')
print('--------------------------------')
print(category_debit_spending_percentage)
print('--------------------------------')
print('************   Total Expenses Sum   *******************')
print('--------------------------------')
print(category_debit_spending_percentage['Amount'].sum())


print('--------------------------------')
print('************   Total Income Sum   *******************')
print('--------------------------------')
print(credits_by_category)
```

```
-------------------------------
************   Debits Break Down   ********************
-------------------------------
                   Category    Amount  Debit Spending Percentage
3         Credit Card Payment  63561.12                 50.204852
14            Mortgage & Rent  24754.50                 19.552771
11           Home Improvement  19092.87                 15.080834
9                   Groceries   2795.21                  2.207845
20                  Utilities   2776.00                  2.192672
17                Restaurants   2613.02                  2.063939
18                   Shopping   1973.24                  1.558598
8                  Gas & Fuel   1715.17                  1.354757
13               Mobile Phone   1680.40                  1.327293
12                   Internet   1570.88                  1.240787
1              Auto Insurance   1350.00                  1.066321
4       Electronics & Software    719.00                  0.567915
0              Alcohol & Bars    539.13                  0.425841
10                    Haircut    378.00                  0.298570
6                   Fast Food    330.63                  0.261154
16                      Music    224.49                  0.177317
15              Movies & DVDs    222.19                  0.175501
2                Coffee Shops    115.54                  0.091261
19                 Television    104.78                  0.082762
7               Food & Dining     77.75                  0.061412
5               Entertainment      9.62                  0.007599
-------------------------------
************   Total Expenses Sum   ********************
-------------------------------
126603.53999999996
-------------------------------
************   Total Income Sum   ********************
-------------------------------
   Category  Total_Amount  Transaction_Count
0  Paycheck       93750.0                 46
```

In [75]:

```python
# Getting total cash flow..
net_credits = df_transactions_copy[df_transactions_copy['Transaction Type'] ==
net_debits = df_transactions_copy[df_transactions_copy['Transaction Type'] ==

net_income = net_credits - net_debits

print('Total Cash Flow')
print('-------------------')
print(f"Net Credits: {net_credits}")
print('-')
print(f"Net Debits: {net_debits}")
print('=')
print('-------------------')
print(f"Net Income: {net_income}")
```

```
Total Cash Flow
------------------
Net Credits: 93750.0
-
Net Debits: 126603.54000000001
=
------------------
Net Income: -32853.54000000001
```

In [53]:
```
# ***
# A quick EDA step we can take here is using the SweetViz feature to get some

import sweetviz as sv
report = sv.analyze(df_transactions_copy)
report.show_html()
```

```
Report SWEETVIZ_REPORT.html was generated! NOTEBOOK/COLAB USERS: the web brows
er MAY not pop up, regardless, the report IS saved in your notebook/colab file
s.
```

Final Changes and Models

So our data set looks the way we want it to and we have a basic understanding of how much this family makes and how much/where they spend it.They currently have a negative cash flow and seem to be using the credit cards for a lot of expenses. We will see what our machine learning models can offer. I am also going to create a table for expenses that are discretionary and expenses that are more necessary. Hopefully breaking down expenses can help with getting the budget and spending under control. Note since we see transactions for things like the mortgage, utilities, gas, etc, I will be putting the credit card payments under the discretionary table since although we arent sure entirely where that money is being spent we know its not going towards those categories.

In [82]:
```
# categories considered as discretionary expenses
discretionary_categories = [
    'Shopping', 'Restaurants', 'Movies & DVDs', 'Music',
    'Coffee Shops', 'Fast Food', 'Alcohol & Bars', 'Entertainment',
    'Food & Dining', 'Electronics & Software', 'Television', 'Credit Card Pay
]

# Using a mask for discretionary expenses within the copied DataFrame
discretionary_mask = df_transactions_copy['Category'].isin(discretionary_cate

# Filtering the DataFrame based on the mask to separate discretionary and nec
df_discretionary_expenses_copy = df_transactions_copy[discretionary_mask]
df_necessary_expenses_copy = df_transactions_copy[~discretionary_mask]

# You can now work with df_discretionary_expenses_copy and df_necessary_expen
```

In [83]:
```
df_transactions_copy.head()
```

Out[83]:

	Date	Description	Amount	Transaction Type	Category	Account_Name
0	01/01/2018	Amazon	11.11	debit	Shopping	Platinum Card
1	01/02/2018	Mortgage Payment	1247.44	debit	Mortgage & Rent	Checking
2	01/02/2018	Thai Restaurant	24.22	debit	Restaurants	Silver Card
3	01/03/2018	Credit Card Payment	2298.09	debit	Credit Card Payment	Platinum Card
4	01/04/2018	Netflix	11.76	debit	Movies & DVDs	Platinum Card

In [84]:
```
df_discretionary_expenses_copy
```

Out[84]:

	Date	Description	Amount	Transaction Type	Category	Account_Name
0	01/01/2018	Amazon	11.11	debit	Shopping	Platinum Card
2	01/02/2018	Thai Restaurant	24.22	debit	Restaurants	Silver Card
3	01/03/2018	Credit Card Payment	2298.09	debit	Credit Card Payment	Platinum Card
4	01/04/2018	Netflix	11.76	debit	Movies & DVDs	Platinum Card
5	01/05/2018	American Tavern	25.85	debit	Restaurants	Silver Card
...	...	...	...	...	...	...
796	09/20/2019	Credit Card Payment	9.43	debit	Credit Card Payment	Silver Card
797	09/22/2019	Seafood Restaurant	131.10	debit	Restaurants	Platinum Card
798	09/23/2019	Credit Card Payment	9.43	debit	Credit Card Payment	Checking
800	09/23/2019	Amazon	24.63	debit	Shopping	Platinum Card
804	09/30/2019	Starbucks	1.75	debit	Coffee Shops	Platinum Card

410 rows × 6 columns

In [85]:

```
df_necessary_expenses_copy.head()
```

Out[85]:

	Date	Description	Amount	Transaction Type	Category	Account_Name
1	01/02/2018	Mortgage Payment	1247.44	debit	Mortgage & Rent	Checking
6	01/06/2018	Hardware Store	18.45	debit	Home Improvement	Silver Card
7	01/08/2018	Gas Company	45.00	debit	Utilities	Checking
8	01/08/2018	Hardware Store	15.38	debit	Home Improvement	Silver Card
10	01/10/2018	Phone Company	89.46	debit	Mobile Phone	Checking

In [86]:

```
df_necessary_expenses_copy['Category'].unique()
```

Out[86]:

```
array(['Mortgage & Rent', 'Home Improvement', 'Utilities', 'Mobile Phone',
       'Gas & Fuel', 'Groceries', 'Paycheck', 'Internet', 'Haircut',
       'Auto Insurance'], dtype=object)
```

Taking a look at how money is being spent in these two table.

In [87]:

```
#net_debits = df_transactions_copy[df_transactions_copy['Transaction Type'] =
disc_debits= df_discretionary_expenses_copy[df_discretionary_expenses_copy['T
nesc_debits= df_necessary_expenses_copy[df_necessary_expenses_copy['Transacti
print('Total discretionary dollars spent:   $',disc_debits)
print('Total necessary dollars spent:   $',nesc_debits)
```

```
Total discretionary dollars spent:   $ 70490.51
Total necessary dollars spent:   $ 56113.03
```

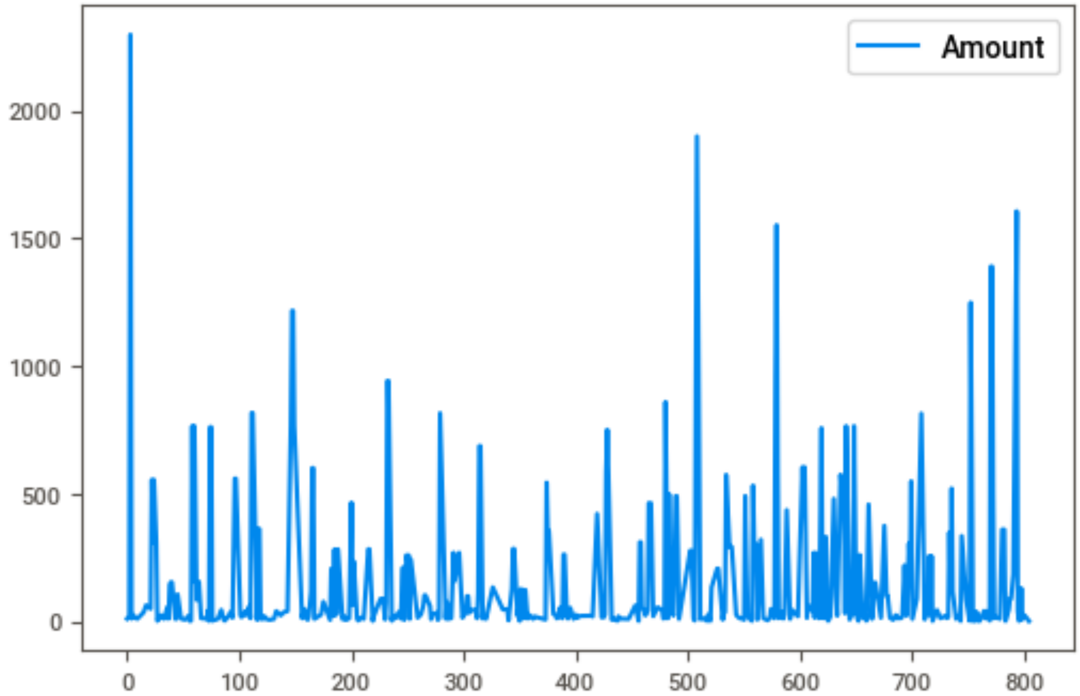
Again we see something that can lead to budget and cash flow issues, the discretionary spending is out pacing the necessary spending. Since we saw there is a negative cash flow this should be addressed.

Initial Models

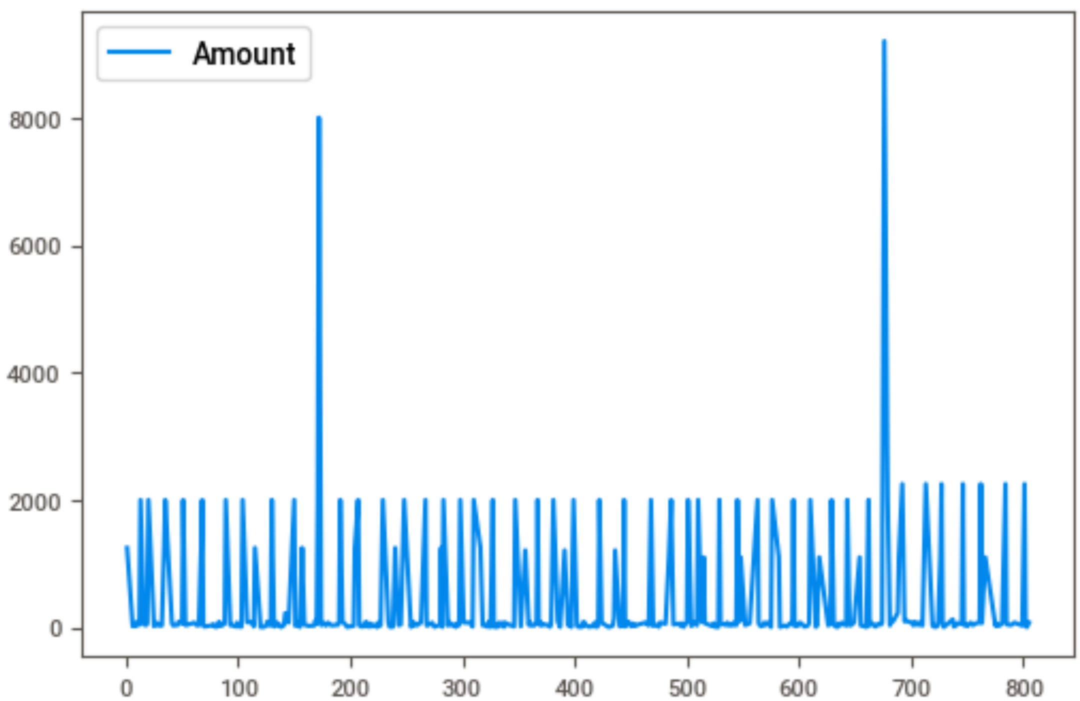
Im going to create a few different forecasting models with the discretionary and necessary tables. In the next few cells some steps will be taken needed to make the models run effectively. This includes setting the index to Date/Time and checking for stationarity, then fine tuning the parameters of the model.

In [92]:

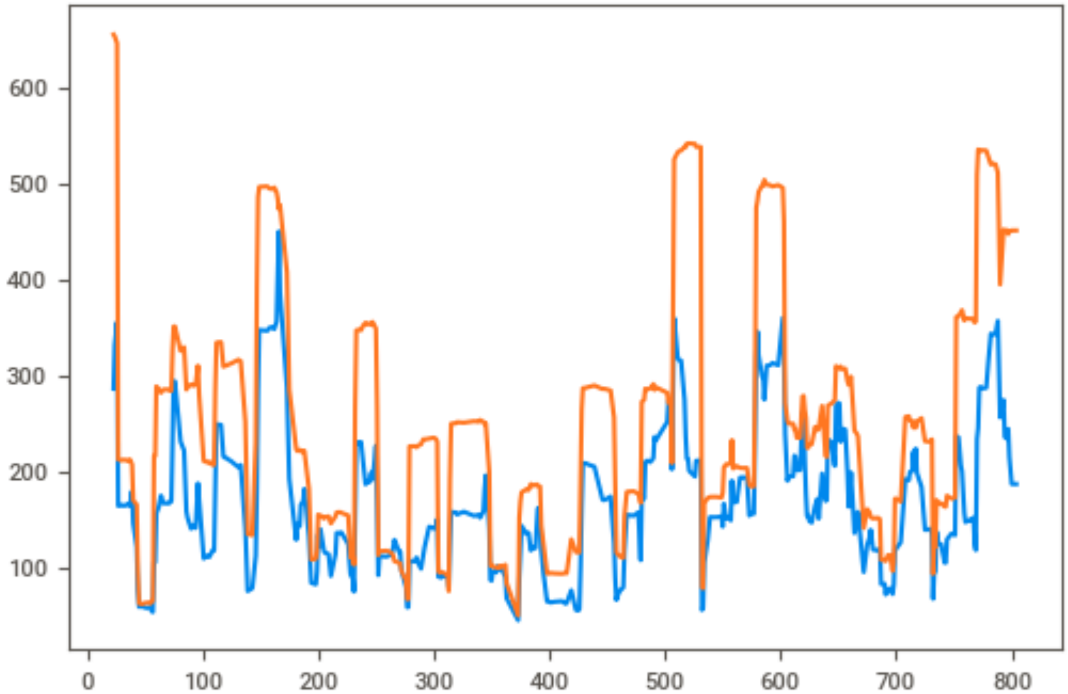
```
#Plotting the discretionary time series...
df_discretionary_expenses_copy.plot(y='Amount')
plt.show()
```



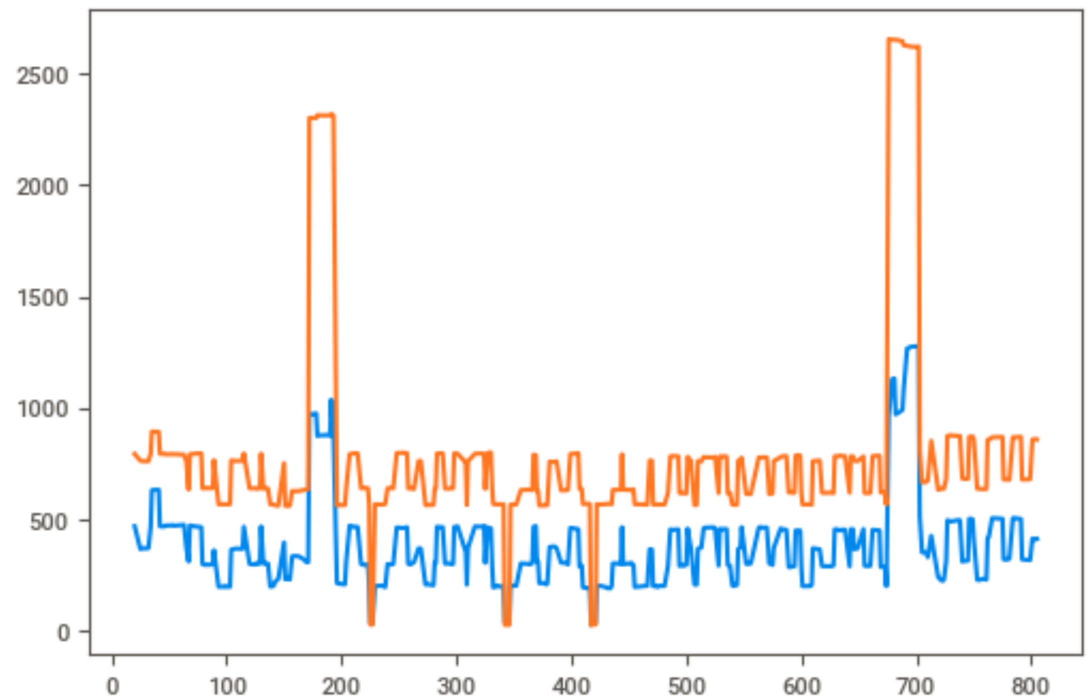
In [96]:
```
#Plotting the table of necessary expenses..
df_necessary_expenses_copy.plot(y='Amount')
plt.show()
```



In [97]:
```
# Plot the rolling mean and rolling standard deviation of the 'discretionary'
df_discretionary_expenses_copy['Amount'].rolling(window=12).mean().plot()
df_discretionary_expenses_copy['Amount'].rolling(window=12).std().plot()
plt.show()
```



In [98]:
```
# Plot the rolling mean and rolling standard deviation of the 'discretionary'
df_necessary_expenses_copy['Amount'].rolling(window=12).mean().plot()
df_necessary_expenses_copy['Amount'].rolling(window=12).std().plot()
plt.show()
```



Ok the plots of the time series and means looks solid but we may have to address outliers in our model reiterations. Now checking for stationarity.

In [99]:

```
from statsmodels.tsa.stattools import adfuller

result = adfuller(df_necessary_expenses_copy['Amount'])
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
#https://machinelearningmastery.com/time-series-data-stationary-python/
```

```
ADF Statistic: -11.533022
p-value: 0.000000
```

In [100…

```
from statsmodels.tsa.stattools import adfuller

result = adfuller(df_discretionary_expenses_copy['Amount'])
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
#https://machinelearningmastery.com/time-series-data-stationary-python/
```

```
ADF Statistic: -11.000442
p-value: 0.000000
```

A negative ADF statistic value, in this case both are approximately -11, indicates that the time series is very likely to be stationary. This is because, in the ADF test, the null hypothesis is that there is a unit root (non-stationarity) in the time series, and a low p-value (typically less than 0.05) is used to reject the null hypothesis and conclude the time series is stationary.

P,D, Q Another important aspect we will have to address soon is the parameter for the SARIMA time-series, which are denoted with 'P', 'D', and 'Q'. With the results of this ADFuller test we can assume our D paramter will be set to 0. The parameters are represented as follows:

p: is the order of the autoregressive term (AR), which is the number of lags used in the model. It describes the number of past values used to predict the next value. d: is the order of the differencing term (I), which is used to make the time series stationary by removing trends or seasonality. It represents the number of times the data has been differenced. q: is the order of the moving average term (MA), which is the error term that captures the short-term fluctuations in the data. It represents the number of past forecast errors used to predict the next value. The 'S' in SARIMA represents the seasonality aspect of the model, usually the notation is 'SARIMA(p,d,q)(P,D,Q)m' with 'm' being a constant such as 12(months).

In [101…

```
# Data is stationary, changing both tables to date time index.
# Convert the 'Date' column to a datetime object
df_discretionary_expenses_copy['Date'] = pd.to_datetime(df_discretionary_expe

# Set the 'Date' column as the DataFrame index
df_discretionary_expenses_copy.set_index('Date', inplace=True)
```

In [102…

```
# Data is stationary, changing both tables to date time index.
# Convert the 'Date' column to a datetime object
df_necessary_expenses_copy['Date'] = pd.to_datetime(df_necessary_expenses_cop

# Set the 'Date' column as the DataFrame index
df_necessary_expenses_copy.set_index('Date', inplace=True)
```

In [105…

```
# Checking that the date column is index...
(df_necessary_expenses_copy.head())
```

Out[105…

Date	Description	Amount	Transaction Type	Category	Account_Name
2018-01-02	Mortgage Payment	1247.44	debit	Mortgage & Rent	Checking
2018-01-06	Hardware Store	18.45	debit	Home Improvement	Silver Card
2018-01-08	Gas Company	45.00	debit	Utilities	Checking
2018-01-08	Hardware Store	15.38	debit	Home Improvement	Silver Card
2018-01-10	Phone Company	89.46	debit	Mobile Phone	Checking

In [106…

```
df_discretionary_expenses_copy.head()
```

Out[106…

Date	Description	Amount	Transaction Type	Category	Account_Name
2018-01-01	Amazon	11.11	debit	Shopping	Platinum Card
2018-01-02	Thai Restaurant	24.22	debit	Restaurants	Silver Card
2018-01-03	Credit Card Payment	2298.09	debit	Credit Card Payment	Platinum Card
2018-01-04	Netflix	11.76	debit	Movies & DVDs	Platinum Card
2018-01-05	American Tavern	25.85	debit	Restaurants	Silver Card

The index is now changed to 'Date' and it should also be noted that if we leave the frequency of the dates as inidividual days it can create a lot of noise in our models. Therefore it should be beneficial to resample the dates to weeks or months to reduce the noise in the data and make it easier to identify patterns and trends. This will also make it easier to train the model as fewer data points will be used. For the dataset ranges 5 years, formatting it to months, would allow to better identify trends in sales over time. By formatting it to weeks, we can analyze the data by looking at the seasonality of the data. We can identify which months of the year the sales are highest and lowest, or identify any cyclical patterns that occur over time. This can be useful to understand patterns in the data and make predictions on future sales.

I will use the 'bfill' attribute which should fill missing values with the last valid observation and helps maintain integrity of the data when going through the model. We will also resample the tables so that they are formatted to weeks instead of months which I think is better for this sized dataset.

In [124…

```
# The term bfill means that we use the value before filling in missing values
df_discretionary_expenses_copy = df_discretionary_expenses_copy.fillna(df_dis

df_discretionary_expenses_copy
```

Out[124…

Date	Description	Amount	Transaction Type	Category	Account_Name
2018-01-01	Amazon	11.11	debit	Shopping	Platinum Card
2018-01-02	Thai Restaurant	24.22	debit	Restaurants	Silver Card
2018-01-03	Credit Card Payment	2298.09	debit	Credit Card Payment	Platinum Card
2018-01-04	Netflix	11.76	debit	Movies & DVDs	Platinum Card
2018-01-05	American Tavern	25.85	debit	Restaurants	Silver Card
...	...	...	...	...	...
2019-09-20	Credit Card Payment	9.43	debit	Credit Card Payment	Silver Card
2019-09-22	Seafood Restaurant	131.10	debit	Restaurants	Platinum Card
2019-09-23	Credit Card Payment	9.43	debit	Credit Card Payment	Checking
2019-09-23	Amazon	24.63	debit	Shopping	Platinum Card
2019-09-30	Starbucks	1.75	debit	Coffee Shops	Platinum Card

410 rows × 5 columns

In [125…

```
# The term bfill means that we use the value before filling in missing values
df_necessary_expenses_copy = df_necessary_expenses_copy.fillna(df_necessary_e

df_necessary_expenses_copy
```

Out[125…

	Description	Amount	Transaction Type	Category	Account_Name
Date					
2018-01-02	Mortgage Payment	1247.44	debit	Mortgage & Rent	Checking
2018-01-06	Hardware Store	18.45	debit	Home Improvement	Silver Card
2018-01-08	Gas Company	45.00	debit	Utilities	Checking
2018-01-08	Hardware Store	15.38	debit	Home Improvement	Silver Card
2018-01-10	Phone Company	89.46	debit	Mobile Phone	Checking
...	...	...	...	...	...
2019-09-23	Grocery Store	27.71	debit	Groceries	Platinum Card
2019-09-27	Biweekly Paycheck	2250.00	credit	Paycheck	Checking
2019-09-28	BP	33.46	debit	Gas & Fuel	Platinum Card
2019-09-28	Sheetz	4.27	debit	Gas & Fuel	Platinum Card
2019-09-30	Internet Service Provider	75.00	debit	Internet	Checking

396 rows × 5 columns

In [126…

```
# Resampling to the data into groups by weeks starting on Saturday...
df_disc_weekly = df_discretionary_expenses_copy.resample('W-SAT')
weekly_disc_mean = df_disc_weekly.mean()
weekly_disc_mean
```

Out[126…

Date	Amount
2018-01-06	474.206000
2018-01-13	27.550000
2018-01-20	57.160000
2018-01-27	346.520000
2018-02-03	27.687500
...	...
2019-09-07	421.042500
2019-09-14	150.772000
2019-09-21	259.297500
2019-09-28	55.053333
2019-10-05	1.750000

92 rows × 1 columns

In [127…

```
# Resampling to the data into groups by weeks starting on Saturday...
df_nec_weekly = df_necessary_expenses_copy.resample('W-SAT')
weekly_nec_mean = df_nec_weekly.mean()
weekly_nec_mean
```

Out[127…

Date	Amount
2018-01-06	632.945000
2018-01-13	371.375000
2018-01-20	534.797500
2018-01-27	43.685000
2018-02-03	826.412500
...	...
2019-09-07	1100.000000
2019-09-14	358.792857
2019-09-21	56.666667
2019-09-28	578.860000
2019-10-05	75.000000

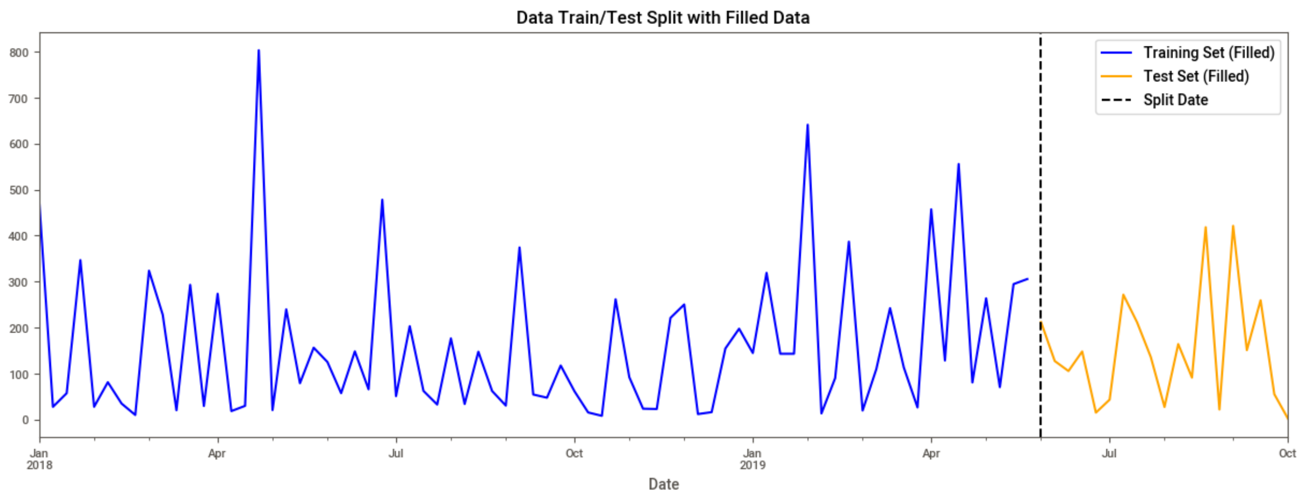
92 rows × 1 columns

First Model For Discretionary Income</center>

We will run 2 forecasting models on this personal data, one for the discretionary and one for necessary expenses. The reason for this is to address the budgeting and expense management issues. Starting with the discretionary table, We will begin by splitting the data into a train and test set and then use a grid search function on the test to get the parameters for the model and then begin fitting and predicting.

In [140…

```
#Ensure data is filled
weekly_disc_filled = weekly_disc_mean['Amount'].bfill()

# Determine the split date(approx 80% of data)
split_date = '2019-05-31'

# Splitting the filled data
train_disc_filled = weekly_disc_filled.loc[weekly_disc_filled.index <= split_
test_disc_filled = weekly_disc_filled.loc[weekly_disc_filled.index > split_da

# Visualization
fig, ax = plt.subplots(figsize=(15, 5))
train_disc_filled.plot(ax=ax, label='Training Set (Filled)', color='blue')
test_disc_filled.plot(ax=ax, label='Test Set (Filled)', color='orange')
ax.axvline(x=split_date, color='black', ls='--', label='Split Date')
ax.legend()
plt.title('Data Train/Test Split with Filled Data')
plt.show()
```



In [141…

```
# Define the p, d and q parameters to take any value between 0 and 3 (exclusi
p = d = q = range(0, 2)

# Generate all different combinations of p, q and q triplets
pdq = list(itertools.product(p, d, q))

# Generate all different combinations of seasonal p, q and q triplets
# Note: here we have 52 in the 's' position as we have weekly data
pdqs = [(x[0], x[1], x[2], 51) for x in list(itertools.product(p, d, q))]
```

In [145…

```
def sarimax_gridsearch(ts, pdq, pdqs, maxiter=50, freq='W-SAT'):
    '''
    Input:
        ts : your time series data
        pdq : ARIMA combinations from above
        pdqs : seasonal ARIMA combinations from above
        maxiter : number of iterations, increase if your model isn't convergi
        frequency : default='M' for month. Change to suit your time series fr
            e.g. 'D' for day, 'H' for hour, 'Y' for year.

    Return:
        Prints out top 5 parameter combinations
        Returns dataframe of parameter combinations ranked by BIC
    '''

    # Run a grid search with pdq and seasonal pdq parameters and get the best
    ans = []
    for comb in pdq:
        for combs in pdqs:
            try:
                mod = sm.tsa.statespace.SARIMAX(train_disc,
                                                order=comb,
                                                seasonal_order=combs,
                                                enforce_stationarity=False,
                                                enforce_invertibility=False,
                                                )

                output_disc = mod.fit(maxiter=maxiter)
                ans.append([comb, combs, output.bic])
                print('SARIMAX {} x {}51 : AIC Calculated ={}'.format(comb, c
            except:
                continue

    # Find the parameters with minimal BIC value

    # Convert into dataframe
    ans_df = pd.DataFrame(ans, columns=['pdq', 'pdqs', 'aic'])

    # Sort and return top 5 combinations
    ans_df = ans_df.sort_values(by=['aic'],ascending=True)[0:5]

    return ans_df
```

In [164…

```
sarimax_gridsearch(train_disc, pdq, pdqs, freq='W-SAT')
```

```
SARIMAX (0, 0, 0) x (0, 0, 0, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (0, 0, 0) x (0, 0, 1, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (0, 0, 0) x (0, 1, 0, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (0, 0, 0) x (0, 1, 1, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (0, 0, 0) x (1, 0, 0, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (0, 0, 0) x (1, 0, 1, 51)51 : AIC Calculated =278.2090400770546
```

```
SARIMAX (0, 0, 0) x (1, 1, 0, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (0, 0, 0) x (1, 1, 1, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (0, 0, 1) x (0, 0, 0, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (0, 0, 1) x (0, 0, 1, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (0, 0, 1) x (0, 1, 0, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (0, 0, 1) x (0, 1, 1, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (0, 0, 1) x (1, 0, 0, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (0, 0, 1) x (1, 0, 1, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (0, 0, 1) x (1, 1, 0, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (0, 0, 1) x (1, 1, 1, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (0, 1, 0) x (0, 0, 0, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (0, 1, 0) x (0, 0, 1, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (0, 1, 0) x (0, 1, 0, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (0, 1, 0) x (0, 1, 1, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (0, 1, 0) x (1, 0, 0, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (0, 1, 0) x (1, 0, 1, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (0, 1, 0) x (1, 1, 0, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (0, 1, 0) x (1, 1, 1, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (0, 1, 1) x (0, 0, 0, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (0, 1, 1) x (0, 0, 1, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (0, 1, 1) x (0, 1, 0, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (0, 1, 1) x (0, 1, 1, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (0, 1, 1) x (1, 0, 0, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (0, 1, 1) x (1, 0, 1, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (0, 1, 1) x (1, 1, 0, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (0, 1, 1) x (1, 1, 1, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (1, 0, 0) x (0, 0, 0, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (1, 0, 0) x (0, 0, 1, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (1, 0, 0) x (0, 1, 0, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (1, 0, 0) x (0, 1, 1, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (1, 0, 0) x (1, 0, 0, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (1, 0, 0) x (1, 0, 1, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (1, 0, 0) x (1, 1, 0, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (1, 0, 0) x (1, 1, 1, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (1, 0, 1) x (0, 0, 0, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (1, 0, 1) x (0, 0, 1, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (1, 0, 1) x (0, 1, 0, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (1, 0, 1) x (0, 1, 1, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (1, 0, 1) x (1, 0, 0, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (1, 0, 1) x (1, 0, 1, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (1, 0, 1) x (1, 1, 0, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (1, 0, 1) x (1, 1, 1, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (1, 1, 0) x (0, 0, 0, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (1, 1, 0) x (0, 0, 1, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (1, 1, 0) x (0, 1, 0, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (1, 1, 0) x (0, 1, 1, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (1, 1, 0) x (1, 0, 0, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (1, 1, 0) x (1, 0, 1, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (1, 1, 0) x (1, 1, 0, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (1, 1, 0) x (1, 1, 1, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (1, 1, 1) x (0, 0, 0, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (1, 1, 1) x (0, 0, 1, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (1, 1, 1) x (0, 1, 0, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (1, 1, 1) x (0, 1, 1, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (1, 1, 1) x (1, 0, 0, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (1, 1, 1) x (1, 0, 1, 51)51 : AIC Calculated =278.2090400770546
SARIMAX (1, 1, 1) x (1, 1, 0, 51)51 : AIC Calculated =278.2090400770546
```

```
SARIMAX (1, 1, 1) x (1, 1, 1, 51)51 : AIC Calculated =278.2090400770546
```

Out[164…

	pdq	pdqs	aic
0	(0, 0, 0)	(0, 0, 0, 51)	281.342607
34	(1, 0, 0)	(0, 1, 0, 51)	281.342607
35	(1, 0, 0)	(0, 1, 1, 51)	281.342607
36	(1, 0, 0)	(1, 0, 0, 51)	281.342607
37	(1, 0, 0)	(1, 0, 1, 51)	281.342607

Discretionary Parameter Results: After some tinkering around I set the parameter boundaries for p,d,q to (0,2) after initially using a larger range due to it being very computationally expensive. The best results here with a BIC score of 248.45 is :

(1, 1, 1) (0,0, 1, 51) We will now fit the model.

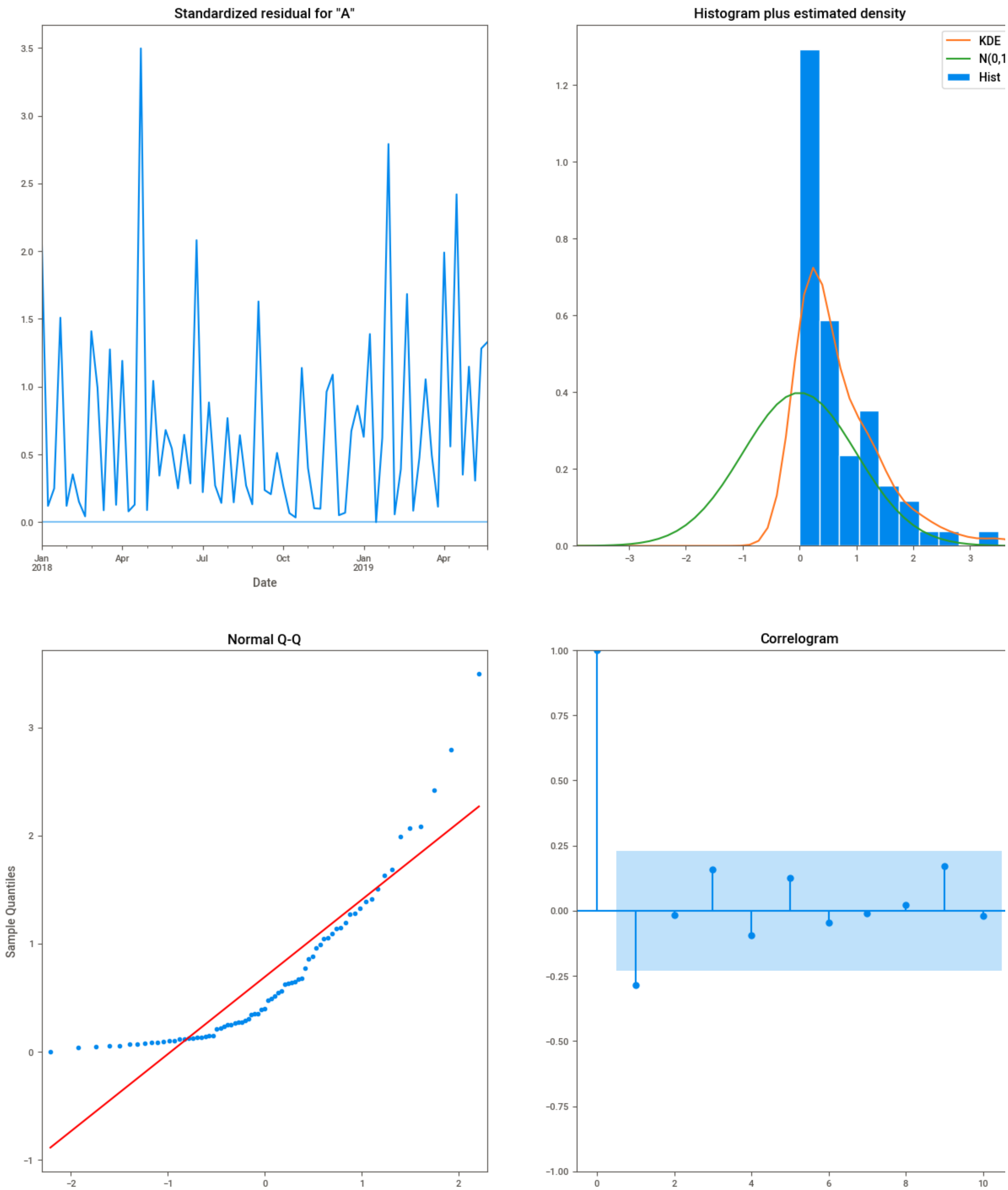
In [251…

```python
# Plug the optimal parameter values into a new SARIMAX model
ARIMA_MODEL_disc = sm.tsa.statespace.SARIMAX(train_disc,
                                        order=(1,0,0),
                                        seasonal_order=(1,0,1, 51),
                                        enforce_invertibility=False)

# Fit the model and print results
output_disc = ARIMA_MODEL_disc.fit()

print(output_disc.summary().tables[1])
```

```
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.2678      0.162      1.651      0.099      -0.050      0.586
ar.S.L51       0.1135      0.392      0.290      0.772      -0.654      0.882
ma.S.L51       0.6599      0.400      1.651      0.099      -0.123      1.443
sigma2        2.8e+04   3.51e-05   7.97e+08      0.000     2.8e+04     2.8e+04
==============================================================================
```

In [249…

```python
# Call plot_diagnostics() on the results calculated above
output_disc.plot_diagnostics(figsize=(15, 18))
plt.show()
```



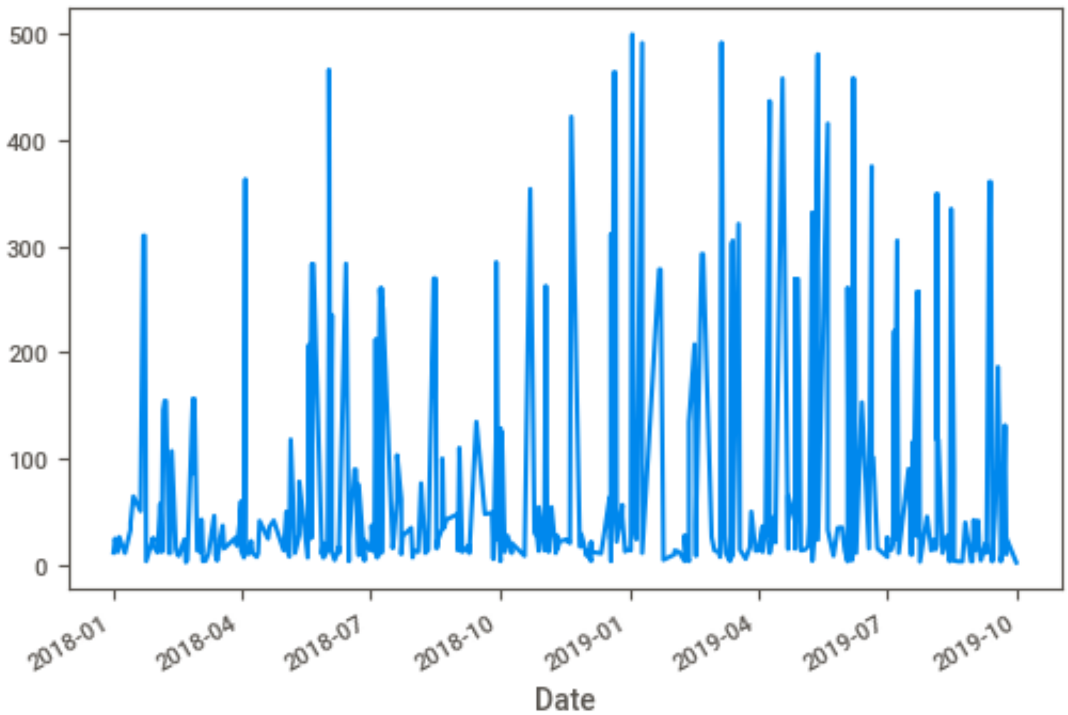
Assumptions:</center> So our assumptions do not look great, we saw earlier that there was an outlier in terms of a large transactions. Initially I wanted to keep it for the integrity of the dataset, but it may need to be removed.

Model with Outliers Removed:

In [254…

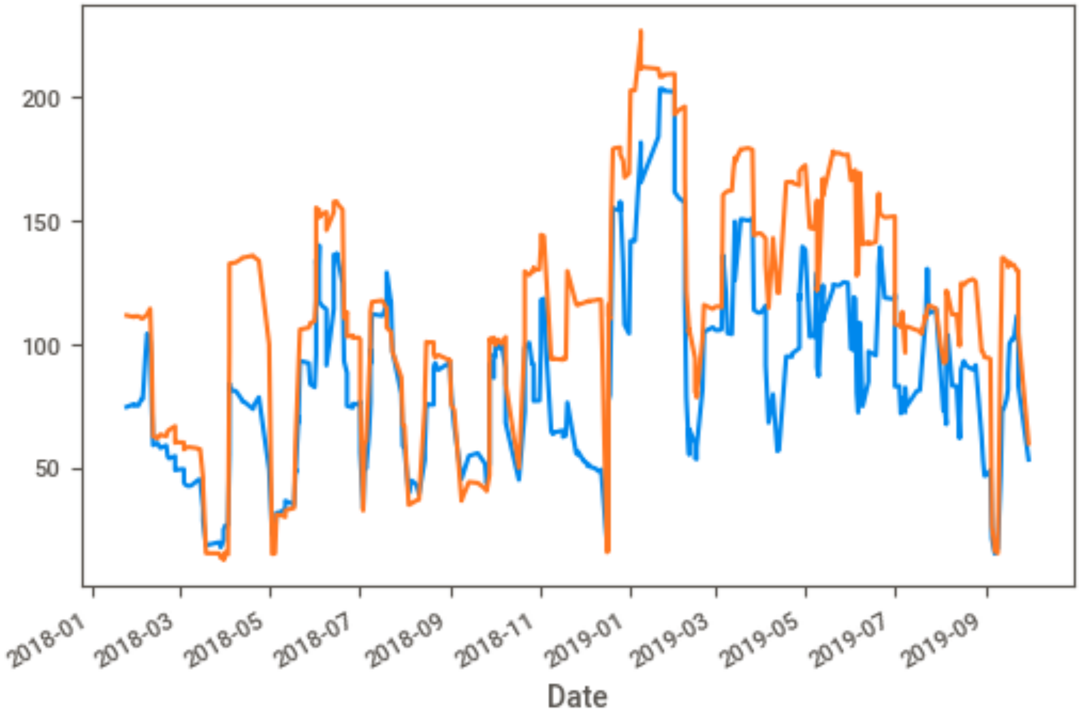
```
df_disc_outlier = df_discretionary_expenses_copy[df_discretionary_expenses_co
df_disc_outlier['Amount'].plot()
```

Out[254… <AxesSubplot:xlabel='Date'>



In [255…

```
# Plot the rolling mean and rolling standard deviation of the 'discretionary'
df_disc_outlier['Amount'].rolling(window=12).mean().plot()
df_disc_outlier['Amount'].rolling(window=12).std().plot()
plt.show()
```



In [256…

```
from statsmodels.tsa.stattools import adfuller

result = adfuller(df_disc_outlier['Amount'])
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
#https://machinelearningmastery.com/time-series-data-stationary-python/
```

```
ADF Statistic: -16.583393
p-value: 0.000000
```

Outliers removed and data is still stationary, repeating the process with this data.

In [257…

```
# Resampling to the data into groups by weeks starting on Saturday...
df_disc_weekly_outlier = df_disc_outlier.resample('W-SAT')
weekly_disc_mean_outlier = df_disc_weekly_outlier.mean()
weekly_disc_mean_outlier
```

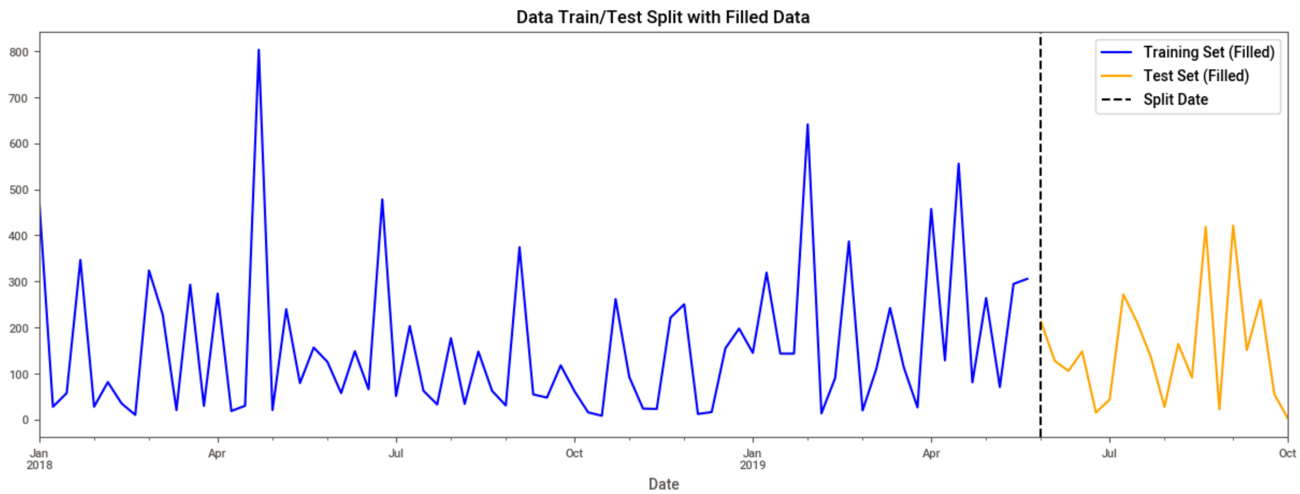
Out[257...

Date	Amount
2018-01-06	18.235000
2018-01-13	27.550000
2018-01-20	57.160000
2018-01-27	207.540000
2018-02-03	27.687500
...	...
2019-09-07	16.970000
2019-09-14	150.772000
2019-09-21	66.845714
2019-09-28	55.053333
2019-10-05	1.750000

92 rows × 1 columns

In [258...

```
#Ensure data is filled
weekly_disc_filled_outlier = weekly_disc_mean_outlier['Amount'].bfill()

# Determine the split date(approx 80% of data)
split_date = '2019-05-31'

# Splitting the filled data
train_disc_filled_outlier = weekly_disc_filled_outlier.loc[weekly_disc_filled_
test_disc_filled_outlier = weekly_disc_filled_outlier.loc[weekly_disc_filled_

# Visualization
fig, ax = plt.subplots(figsize=(15, 5))
train_disc_filled.plot(ax=ax, label='Training Set (Filled)', color='blue')
test_disc_filled.plot(ax=ax, label='Test Set (Filled)', color='orange')
ax.axvline(x=split_date, color='black', ls='--', label='Split Date')
ax.legend()
plt.title('Data Train/Test Split with Filled Data')
plt.show()
```



In [263…

```
# Define the p, d and q parameters to take any value between 0 and 3 (exclusi
p = d = q = range(0, 2)

# Generate all different combinations of p, q and q triplets
pdq = list(itertools.product(p, d, q))

# Generate all different combinations of seasonal p, q and q triplets
# Note: here we have 52 in the 's' position as we have weekly data
pdqs = [(x[2], x[2], x[2], 51) for x in list(itertools.product(p, d, q))]
```

In [264…

```
def sarimax_gridsearch(ts, pdq, pdqs, maxiter=50, freq='W-SAT'):
    '''
    Input:
        ts : your time series data
        pdq : ARIMA combinations from above
        pdqs : seasonal ARIMA combinations from above
        maxiter : number of iterations, increase if your model isn't convergi
        frequency : default='M' for month. Change to suit your time series fr
            e.g. 'D' for day, 'H' for hour, 'Y' for year.

    Return:
        Prints out top 5 parameter combinations
        Returns dataframe of parameter combinations ranked by BIC
    '''

    # Run a grid search with pdq and seasonal pdq parameters and get the best
    ans = []
    for comb in pdq:
        for combs in pdqs:
            try:
                mod = sm.tsa.statespace.SARIMAX(train_disc_filled_outlier,
                                                order=comb,
                                                seasonal_order=combs,
                                                enforce_stationarity=False,
                                                enforce_invertibility=False)


                output_disc_outlier = mod.fit(maxiter=maxiter)
                ans.append([comb, combs, output.bic])
                print('SARIMAX {} x {}51 : AIC Calculated ={}'.format(comb, c
            except:
                continue

    # Find the parameters with minimal BIC value

    # Convert into dataframe
    ans_df = pd.DataFrame(ans, columns=['pdq', 'pdqs', 'aic'])

    # Sort and return top 5 combinations
    ans_df = ans_df.sort_values(by=['aic'],ascending=True)[0:5]

    return ans_df
```

In [265…

```
sarimax_gridsearch(train_disc_filled_outlier, pdq, pdqs, freq='W-SAT')
```

```
SARIMAX (0, 0, 0) x (0, 0, 0, 51)51 : AIC Calculated =894.3594495161997
SARIMAX (0, 0, 0) x (1, 1, 1, 51)51 : AIC Calculated =6.0
SARIMAX (0, 0, 0) x (0, 0, 0, 51)51 : AIC Calculated =894.3594495161997
SARIMAX (0, 0, 0) x (1, 1, 1, 51)51 : AIC Calculated =6.0
SARIMAX (0, 0, 0) x (0, 0, 0, 51)51 : AIC Calculated =894.3594495161997
SARIMAX (0, 0, 0) x (1, 1, 1, 51)51 : AIC Calculated =6.0
```

```
SARIMAX (0, 0, 0) x (0, 0, 0, 51)51 : AIC Calculated =894.3594495161997
SARIMAX (0, 0, 0) x (1, 1, 1, 51)51 : AIC Calculated =6.0
SARIMAX (0, 0, 1) x (0, 0, 0, 51)51 : AIC Calculated =867.6395278095861
SARIMAX (0, 0, 1) x (1, 1, 1, 51)51 : AIC Calculated =8.0
SARIMAX (0, 0, 1) x (0, 0, 0, 51)51 : AIC Calculated =867.6395278095861
SARIMAX (0, 0, 1) x (1, 1, 1, 51)51 : AIC Calculated =8.0
SARIMAX (0, 0, 1) x (0, 0, 0, 51)51 : AIC Calculated =867.6395278095861
SARIMAX (0, 0, 1) x (1, 1, 1, 51)51 : AIC Calculated =8.0
SARIMAX (0, 0, 1) x (0, 0, 0, 51)51 : AIC Calculated =867.6395278095861
SARIMAX (0, 0, 1) x (1, 1, 1, 51)51 : AIC Calculated =8.0
SARIMAX (0, 1, 0) x (0, 0, 0, 51)51 : AIC Calculated =873.5800991153773
SARIMAX (0, 1, 0) x (1, 1, 1, 51)51 : AIC Calculated =6.0
SARIMAX (0, 1, 0) x (0, 0, 0, 51)51 : AIC Calculated =873.5800991153773
SARIMAX (0, 1, 0) x (1, 1, 1, 51)51 : AIC Calculated =6.0
SARIMAX (0, 1, 0) x (0, 0, 0, 51)51 : AIC Calculated =873.5800991153773
SARIMAX (0, 1, 0) x (1, 1, 1, 51)51 : AIC Calculated =6.0
SARIMAX (0, 1, 0) x (0, 0, 0, 51)51 : AIC Calculated =873.5800991153773
SARIMAX (0, 1, 0) x (1, 1, 1, 51)51 : AIC Calculated =6.0
SARIMAX (0, 1, 1) x (0, 0, 0, 51)51 : AIC Calculated =822.1851653167263
SARIMAX (0, 1, 1) x (1, 1, 1, 51)51 : AIC Calculated =8.0
SARIMAX (0, 1, 1) x (0, 0, 0, 51)51 : AIC Calculated =822.1851653167263
SARIMAX (0, 1, 1) x (1, 1, 1, 51)51 : AIC Calculated =8.0
SARIMAX (0, 1, 1) x (0, 0, 0, 51)51 : AIC Calculated =822.1851653167263
SARIMAX (0, 1, 1) x (1, 1, 1, 51)51 : AIC Calculated =8.0
SARIMAX (0, 1, 1) x (0, 0, 0, 51)51 : AIC Calculated =822.1851653167263
SARIMAX (0, 1, 1) x (1, 1, 1, 51)51 : AIC Calculated =8.0
SARIMAX (1, 0, 0) x (0, 0, 0, 51)51 : AIC Calculated =869.6281474771138
SARIMAX (1, 0, 0) x (1, 1, 1, 51)51 : AIC Calculated =8.0
SARIMAX (1, 0, 0) x (0, 0, 0, 51)51 : AIC Calculated =869.6281474771138
SARIMAX (1, 0, 0) x (1, 1, 1, 51)51 : AIC Calculated =8.0
SARIMAX (1, 0, 0) x (0, 0, 0, 51)51 : AIC Calculated =869.6281474771138
SARIMAX (1, 0, 0) x (1, 1, 1, 51)51 : AIC Calculated =8.0
SARIMAX (1, 0, 0) x (0, 0, 0, 51)51 : AIC Calculated =869.6281474771138
SARIMAX (1, 0, 0) x (1, 1, 1, 51)51 : AIC Calculated =8.0
SARIMAX (1, 0, 1) x (0, 0, 0, 51)51 : AIC Calculated =831.8041627073536
SARIMAX (1, 0, 1) x (1, 1, 1, 51)51 : AIC Calculated =10.0
SARIMAX (1, 0, 1) x (0, 0, 0, 51)51 : AIC Calculated =831.8041627073536
SARIMAX (1, 0, 1) x (1, 1, 1, 51)51 : AIC Calculated =10.0
SARIMAX (1, 0, 1) x (0, 0, 0, 51)51 : AIC Calculated =831.8041627073536
SARIMAX (1, 0, 1) x (1, 1, 1, 51)51 : AIC Calculated =10.0
SARIMAX (1, 0, 1) x (0, 0, 0, 51)51 : AIC Calculated =831.8041627073536
SARIMAX (1, 0, 1) x (1, 1, 1, 51)51 : AIC Calculated =10.0
SARIMAX (1, 1, 0) x (0, 0, 0, 51)51 : AIC Calculated =864.35975729623
SARIMAX (1, 1, 0) x (1, 1, 1, 51)51 : AIC Calculated =8.0
SARIMAX (1, 1, 0) x (0, 0, 0, 51)51 : AIC Calculated =864.35975729623
SARIMAX (1, 1, 0) x (1, 1, 1, 51)51 : AIC Calculated =8.0
SARIMAX (1, 1, 0) x (0, 0, 0, 51)51 : AIC Calculated =864.35975729623
SARIMAX (1, 1, 0) x (1, 1, 1, 51)51 : AIC Calculated =8.0
SARIMAX (1, 1, 0) x (0, 0, 0, 51)51 : AIC Calculated =864.35975729623
SARIMAX (1, 1, 0) x (1, 1, 1, 51)51 : AIC Calculated =8.0
SARIMAX (1, 1, 1) x (0, 0, 0, 51)51 : AIC Calculated =824.1851644330973
SARIMAX (1, 1, 1) x (1, 1, 1, 51)51 : AIC Calculated =10.0
SARIMAX (1, 1, 1) x (0, 0, 0, 51)51 : AIC Calculated =824.1851644330973
SARIMAX (1, 1, 1) x (1, 1, 1, 51)51 : AIC Calculated =10.0
SARIMAX (1, 1, 1) x (0, 0, 0, 51)51 : AIC Calculated =824.1851644330973
SARIMAX (1, 1, 1) x (1, 1, 1, 51)51 : AIC Calculated =10.0
SARIMAX (1, 1, 1) x (0, 0, 0, 51)51 : AIC Calculated =824.1851644330973
```

```
SARIMAX (1, 1, 1) x (1, 1, 1, 51)51 : AIC Calculated =10.0
```

Out[265…

	pdq	pdqs	aic
0	(0, 0, 0)	(0, 0, 0, 51)	281.342607
34	(1, 0, 0)	(0, 0, 0, 51)	281.342607
35	(1, 0, 0)	(1, 1, 1, 51)	281.342607
36	(1, 0, 0)	(0, 0, 0, 51)	281.342607
37	(1, 0, 0)	(1, 1, 1, 51)	281.342607

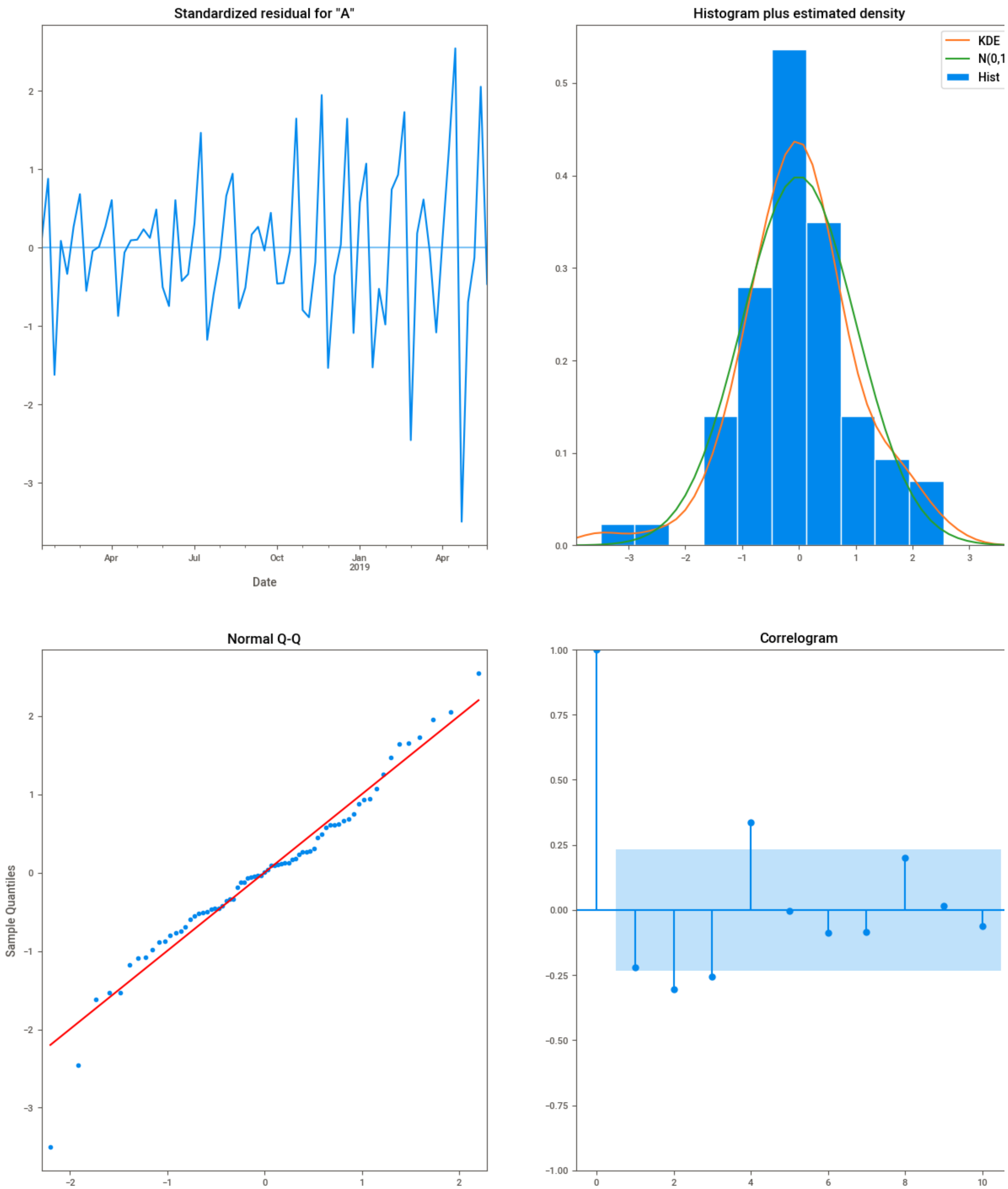
In [266…

```python
# Plug the optimal parameter values into a new SARIMAX model
ARIMA_MODEL_disc_outlier = sm.tsa.statespace.SARIMAX(train_disc_filled_outlie
                                        order=(2,2,0),
                                        seasonal_order=(1,0,0, 51),
                                        enforce_invertibility=False)

# Fit the model and print results
output_disc_outlier = ARIMA_MODEL_disc_outlier.fit()

print(output_disc_outlier.summary().tables[1])
```

```
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.8224      0.118     -6.956      0.000      -1.054      -0.591
ar.L2         -0.4174      0.117     -3.581      0.000      -0.646      -0.189
ar.S.L51       0.0662      0.405      0.164      0.870      -0.727       0.859
sigma2      1.871e+04   2779.136      6.734      0.000    1.33e+04    2.42e+04
==============================================================================
```

In [267…

```python
# Call plot_diagnostics() on the results calculated above
output_disc_outlier.plot_diagnostics(figsize=(15, 18))
plt.show()
```

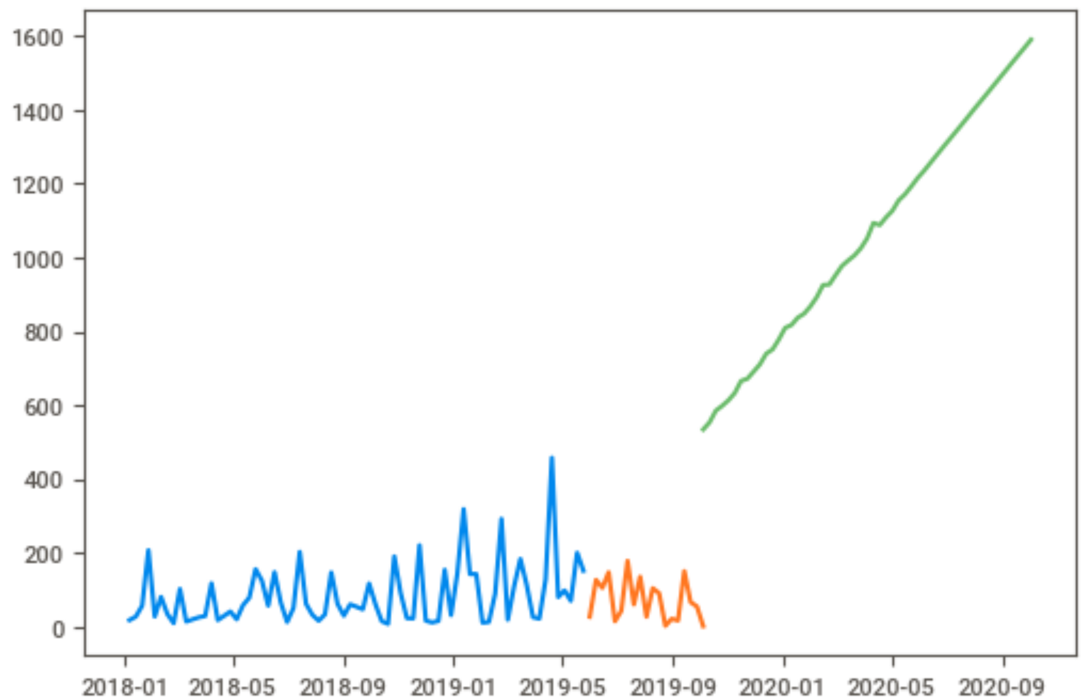


I had to do some tinkering around with the parameters but at least the assumptions can be worked with.

In [268…

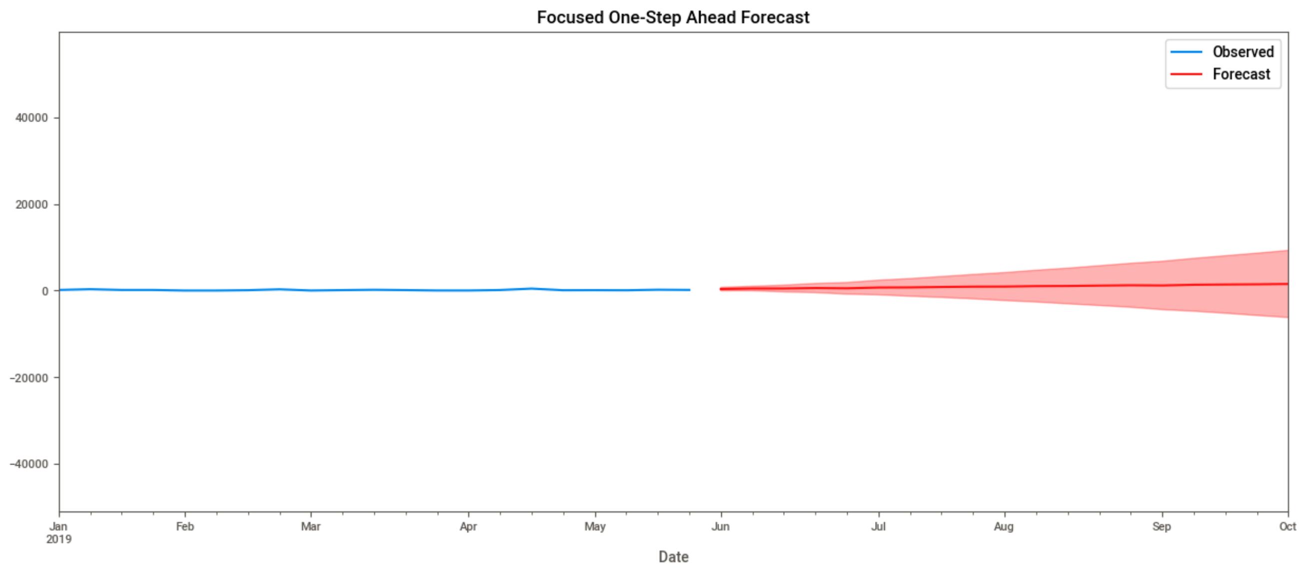
```
# Get the predicted values
pred_disc = output_disc_outlier.get_prediction(start=pd.to_datetime('09-30-20
pred_conf_disc = pred_disc.conf_int()

# Plot the actual values and predicted values
plt.plot(train_disc_filled_outlier, label='Train')
plt.plot(test_disc_filled_outlier, label='Test')
plt.plot(pred_disc.predicted_mean, label='One-step Ahead Forecast', alpha=.7)

# Shade the area between the confidence intervals
plt.fill_between(pred_disc.index, pred_conf_disc.iloc[:, 0], pred_conf_disc.i

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

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-268-c6b3ba0b74dc> in <module>
      9 
     10 # Shade the area between the confidence intervals
---> 11 plt.fill_between(pred_disc.index, pred_conf_disc.iloc[:, 0],
pred_conf_disc.iloc[:, 1], color='k', alpha=.2)
     12 
     13 plt.legend()

/opt/anaconda3/lib/python3.8/site-packages/statsmodels/base/wrapper.py in __ge
tattribute__(self, attr)
     32             pass
     33 
---> 34         obj = getattr(results, attr)
     35         data = results.model.data
     36         how = self._wrap_attrs.get(attr)

AttributeError: 'PredictionResults' object has no attribute 'index'
```



In [270…

```
# Plotting with a focus on the forecast period
plt.figure(figsize=(15, 6))

# Plotting the observed values up until the end of the dataset
train_disc_filled_outlier.loc['2019-01-01':].plot(label='Observed')

# Plotting the forecasted values
pred.predicted_mean.plot(label='Forecast', alpha=0.9, color='red')

# Shading the confidence interval
plt.fill_between(pred_conf.index,
                 pred_conf.iloc[:, 0],
                 pred_conf.iloc[:, 1], color='red', alpha=0.3)

plt.legend()
plt.title('Focused One-Step Ahead Forecast')
plt.xlim(['2019-01-01', '2019-09-30'])  # Focusing on 2019 onwards including
plt.show()
```



Budget Overrun </center>

Future Spending Forecast</center>

Business Transactions

Moving on we also want to offer some machine learning insights for small business owners. I will import about 5 years worth of data from a company that makes tools and grip wraps for those tools. The table will show strictly the grip wrap sales, they are broken down by different color options but the products are essentially all the same. Also included are things like the date of sale, quantity and sales price.

In [271…

```
df_biz = pd.read_csv('/Users/natashawyatt/Documents/Flatiron_school/capstone/\
```

In [272…

```
df_biz.head()
```

Out[272…

	Product_ID	Date	Transaction Type	Qty	Sales_Price	Amount
0	FireWrap Grip Kit - Light Blue	03/23/2018	Sales Receipt	1.0	24.95	24.95
1	FireWrap Grip Kit - Light Blue	04/26/2018	Sales Receipt	1.0	24.95	24.95
2	FireWrap Grip Kit - Light Blue	04/27/2018	Sales Receipt	1.0	24.95	24.95
3	FireWrap Grip Kit - Light Blue	05/14/2018	Sales Receipt	1.0	34.95	34.95
4	FireWrap Grip Kit - Light Blue	06/22/2018	Invoice	4.0	26.00	104.00

In [273…

```
df_biz.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5740 entries, 0 to 5739
Data columns (total 6 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Product_ID        5740 non-null   object 
 1   Date              5740 non-null   object 
 2   Transaction Type  5740 non-null   object 
 3   Qty               5740 non-null   float64
 4   Sales_Price       5740 non-null   float64
 5   Amount            5740 non-null   object 
dtypes: float64(2), object(4)
memory usage: 269.2+ KB
```

In [274…
```
# Just taking a look at the products...
df_biz['Product_ID'].unique()
```

Out[274…
```
array(['FireWrap Grip Kit - Light Blue', 'FireWrap Grip Kit - Pink',
       'FireWrap® Grip Kit Black', 'FireWrap® Grip Kit Blue',
       'FireWrap® Grip Kit GLOW - Aqua',
       'FireWrap® Grip Kit GLOW - Green ( 927 )',
       'FireWrap® Grip Kit Orange', 'FireWrap® Grip Kit Red',
       'FireWrap® Grip Kit Yellow', 'FireWrap® Grip Kit Green',
       'FireWrap® Grip Kit White'], dtype=object)
```

In [275…
```
# Breakdown of the sales of each product...
df_biz['Product_ID'].value_counts()
```

Out[275…
```
FireWrap® Grip Kit Black                    1614
FireWrap® Grip Kit Red                      1265
FireWrap® Grip Kit Blue                      627
FireWrap® Grip Kit Orange                    513
FireWrap® Grip Kit GLOW - Green ( 927 )      474
FireWrap® Grip Kit GLOW - Aqua               428
FireWrap® Grip Kit Yellow                    329
FireWrap® Grip Kit Green                     322
FireWrap® Grip Kit White                     144
FireWrap Grip Kit - Light Blue                15
FireWrap Grip Kit - Pink                       9
Name: Product_ID, dtype: int64
```

In [276…
```
print('Total Units Sold =',df_biz['Qty'].sum())
print('***********')
print('***********')


print(df_biz['Qty'].describe())
```

```
Total Units Sold = 13988.0
***********
***********
count    5740.000000
mean        2.436934
std         7.778742
min         0.000000
25%         1.000000
50%         1.000000
75%         2.000000
max       400.000000
Name: Qty, dtype: float64
```

In [277…
```
# Glance at different value counts per each date...
df_biz['Date'].value_counts()
```

Out[277…
```
03/27/2021    48
04/27/2018    38
05/03/2022    35
04/26/2018    33
01/19/2021    32
              ..
09/30/2019     1
10/13/2022     1
08/13/2020     1
07/19/2021     1
08/06/2022     1
Name: Date, Length: 1467, dtype: int64
```

In [278…
```
# Table to find the beginning and end of dates...
date_range = df_biz.groupby('Date').sum().reset_index()
date_range = date_range.sort_values(by= 'Date', ascending = False)
date_range.head()
```

Out[278…

	Date	Qty	Sales_Price
1466	12/31/2022	7.0	115.80
1465	12/31/2021	1.0	24.95
1464	12/31/2020	3.0	87.85
1463	12/31/2019	1.0	34.95
1462	12/31/2018	3.0	117.85

In [279…
```
date_range.tail()
```

Out[279…

	Date	Qty	Sales_Price
4	01/02/2019	1.0	34.95
3	01/01/2023	8.0	195.70
2	01/01/2022	3.0	37.95
1	01/01/2021	3.0	87.85
0	01/01/2020	2.0	69.90

In [280…
```
# All the different types of Transactions, these 2 look interchangeable
df_biz['Transaction Type'].unique()
```

Out[280…
```
array(['Sales Receipt', 'Invoice'], dtype=object)
```

In [281…

```
# A glance at the quantity of units sold
df_biz['Qty'].describe()
```

Out[281…

```
count    5740.000000
mean        2.436934
std         7.778742
min         0.000000
25%         1.000000
50%         1.000000
75%         2.000000
max       400.000000
Name: Qty, dtype: float64
```

In [282…

```
df_biz['Qty'].value_counts()
```

Out[282…

```
1.0      3735
2.0       852
4.0       259
3.0       247
5.0       163
6.0       141
10.0       86
8.0        71
7.0        43
20.0       22
12.0       18
15.0       15
9.0        12
11.0       11
14.0       10
16.0        8
17.0        7
13.0        6
25.0        4
18.0        4
40.0        3
22.0        3
21.0        3
32.0        2
36.0        2
150.0       2
24.0        2
50.0        1
27.0        1
34.0        1
33.0        1
28.0        1
300.0       1
400.0       1
30.0        1
0.0         1
Name: Qty, dtype: int64
```

In [283…

```
df_biz['Sales_Price'].describe()
```

Out[283…

```
count    5740.000000
mean       28.602956
std         6.455606
min         0.000000
25%        24.950000
50%        27.950000
75%        34.950000
max        47.950000
Name: Sales_Price, dtype: float64
```

In [284…

```
df_biz['Sales_Price'].value_counts()
```

Out[284…

```
24.95    1246
27.95    1101
34.95     758
24.99     632
37.95     399
21.75     313
18.75     313
40.95     158
47.95     146
28.95     127
30.00     115
19.99      89
28.50      69
22.95      47
31.50      43
26.00      32
41.95      27
28.00      16
41.00      14
32.99      14
16.50      13
36.00      12
25.00      12
20.00      11
29.00       8
35.99       8
15.00       4
24.00       4
0.00        4
32.00       2
34.00       1
12.95       1
37.90       1
Name: Sales_Price, dtype: int64
```

In [285…

```
# Seeing how many missing values are in each column
df_biz.isna().sum()
```

Out[285…
```
Product_ID          0
Date                0
Transaction Type    0
Qty                 0
Sales_Price         0
Amount              0
dtype: int64
```

In [286…
```
# filter the rows where Quantity is greater than 100
df_qty = df_biz[df_biz['Qty'] > 100]

# print the Quantity column of the filtered DataFrame
(df_qty['Qty'])
```

Out[286…
```
269     400.0
2197    150.0
3519    300.0
4543    150.0
Name: Qty, dtype: float64
```

In [287…
```
import sweetviz as sv
report_biz = sv.analyze(df_biz)
report_biz.show_html()
```

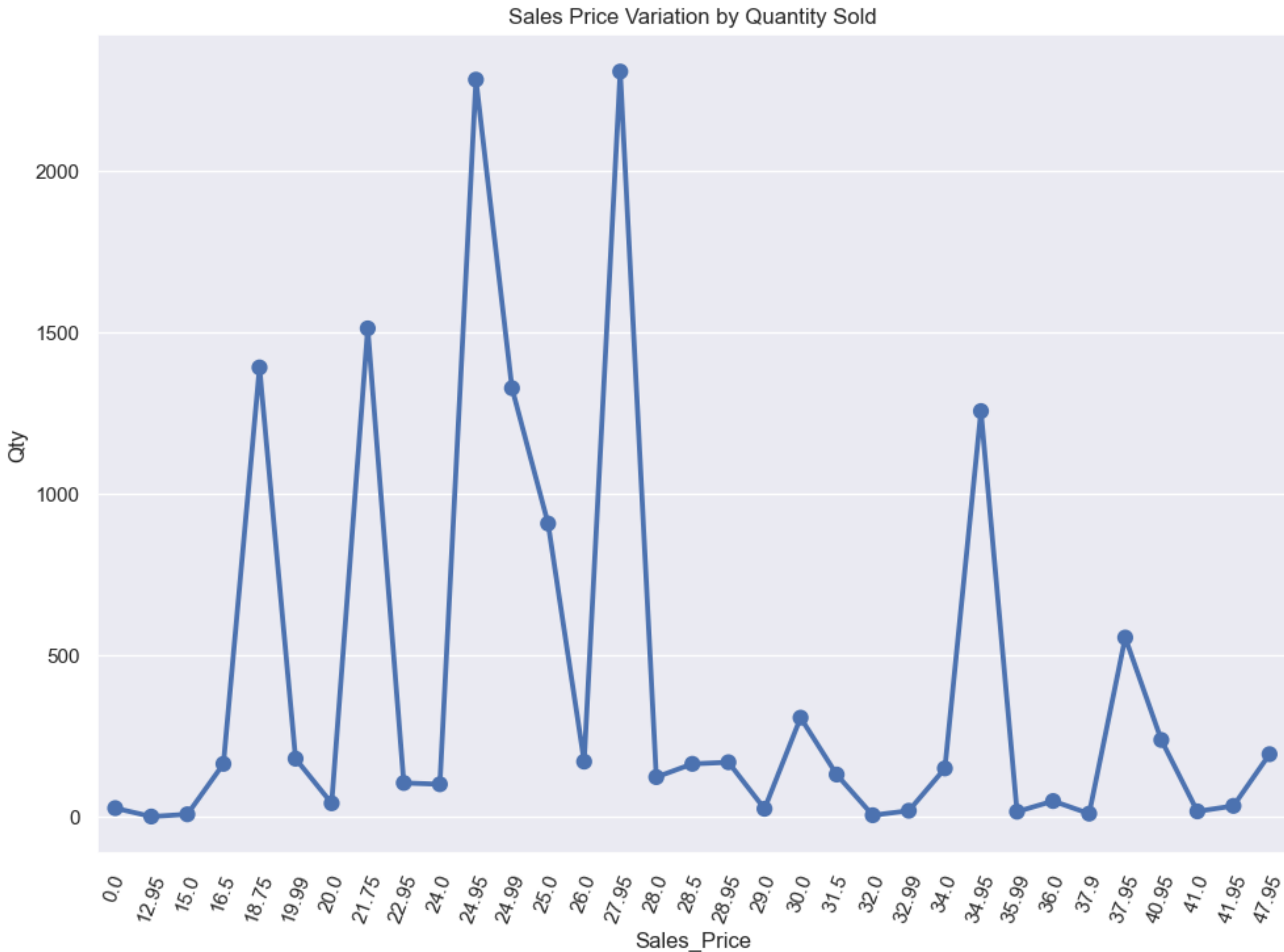
```
Report SWEETVIZ_REPORT.html was generated! NOTEBOOK/COLAB USERS: the web brows
er MAY not pop up, regardless, the report IS saved in your notebook/colab file
s.
```

In [288…
```
# Creating a table to show the amount of sales for a specific price point...

#This table will be used for the coming visual...

sales = df_biz.groupby('Sales_Price').sum().reset_index()
sales = sales.sort_values(by= 'Sales_Price', ascending = False)
print('******  Highest Prices *********')
print(sales.head())
print('*****  Lowest Prices  **********')
print(sales.tail())
```

```
******  Highest Prices *********
    Sales_Price    Qty
32        47.95  194.0
31        41.95   34.0
30        41.00   17.0
29        40.95  238.0
28        37.95  555.0
*****  Lowest Prices  **********
   Sales_Price     Qty
4        18.75  1391.0
3        16.50   164.0
2        15.00     8.0
1        12.95     1.0
0         0.00    27.0
```

In [289…

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(rc={'figure.figsize':(11.7,8.27)})
# Create the plot
sns.pointplot(data = sales, x ='Sales_Price', y ='Qty')
# Add a title, ticks
plt.title('Sales Price Variation by Quantity Sold')
plt.xticks(rotation = 70)
# Show the plot
plt.show()
#https://seaborn.pydata.org/generated/seaborn.pointplot.html
```



In [290…

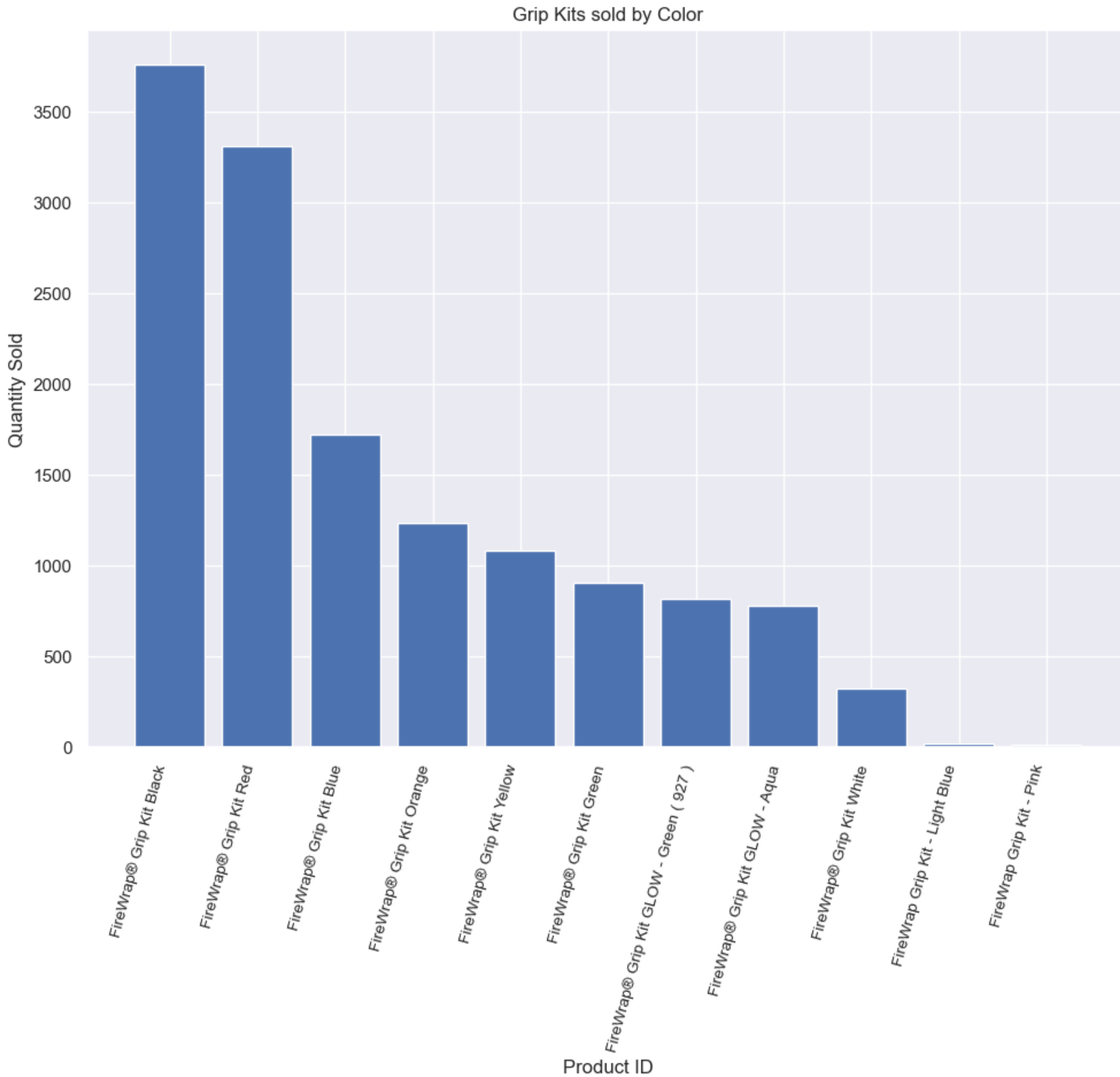
```
# Setting the kits in order of most sold
products = df_biz.groupby('Product_ID')['Qty'].sum().reset_index()
products = products.sort_values(by= 'Qty', ascending = False)
products
```

Out[290…

	Product_ID	Qty
2	FireWrap® Grip Kit Black	3763.0
8	FireWrap® Grip Kit Red	3313.0
3	FireWrap® Grip Kit Blue	1726.0
7	FireWrap® Grip Kit Orange	1235.0
10	FireWrap® Grip Kit Yellow	1081.0
6	FireWrap® Grip Kit Green	907.0
5	FireWrap® Grip Kit GLOW - Green (927)	821.0
4	FireWrap® Grip Kit GLOW - Aqua	781.0
9	FireWrap® Grip Kit White	323.0
0	FireWrap Grip Kit - Light Blue	21.0
1	FireWrap Grip Kit - Pink	17.0

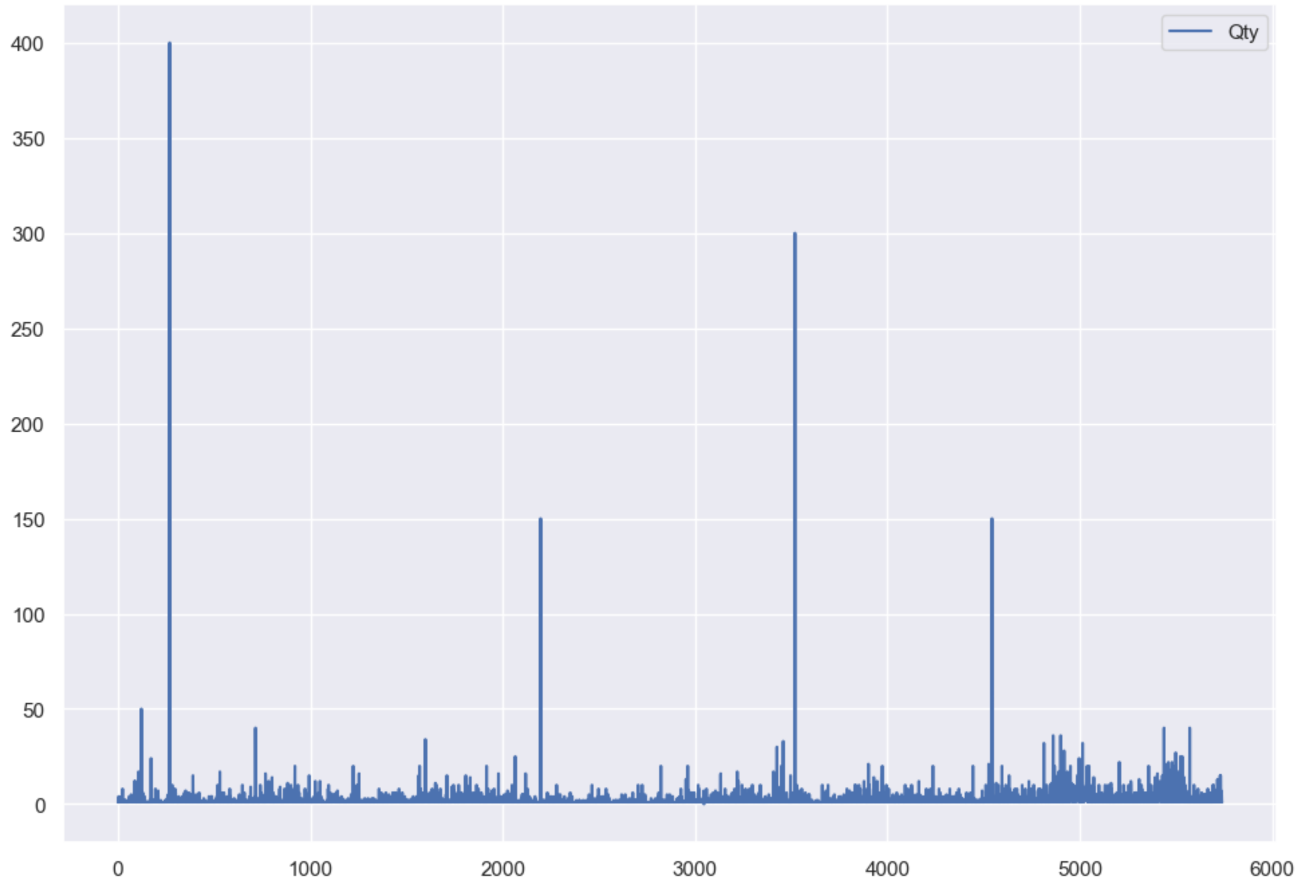
In [291…

```
# Visual for sales... IN ORDER!
import matplotlib.pyplot as plt

# Create a bar chart that shows sales IN ORDER!
plt.bar(products['Product_ID'], products['Qty'])
fig = plt.figsize=(20,10)
plt.xlabel('Product ID')
plt.xticks(rotation = 75, fontsize = 10, ha= 'right')
plt.ylabel('Quantity Sold')
plt.title('Grip Kits sold by Color')

# Show the chart
plt.show()
```



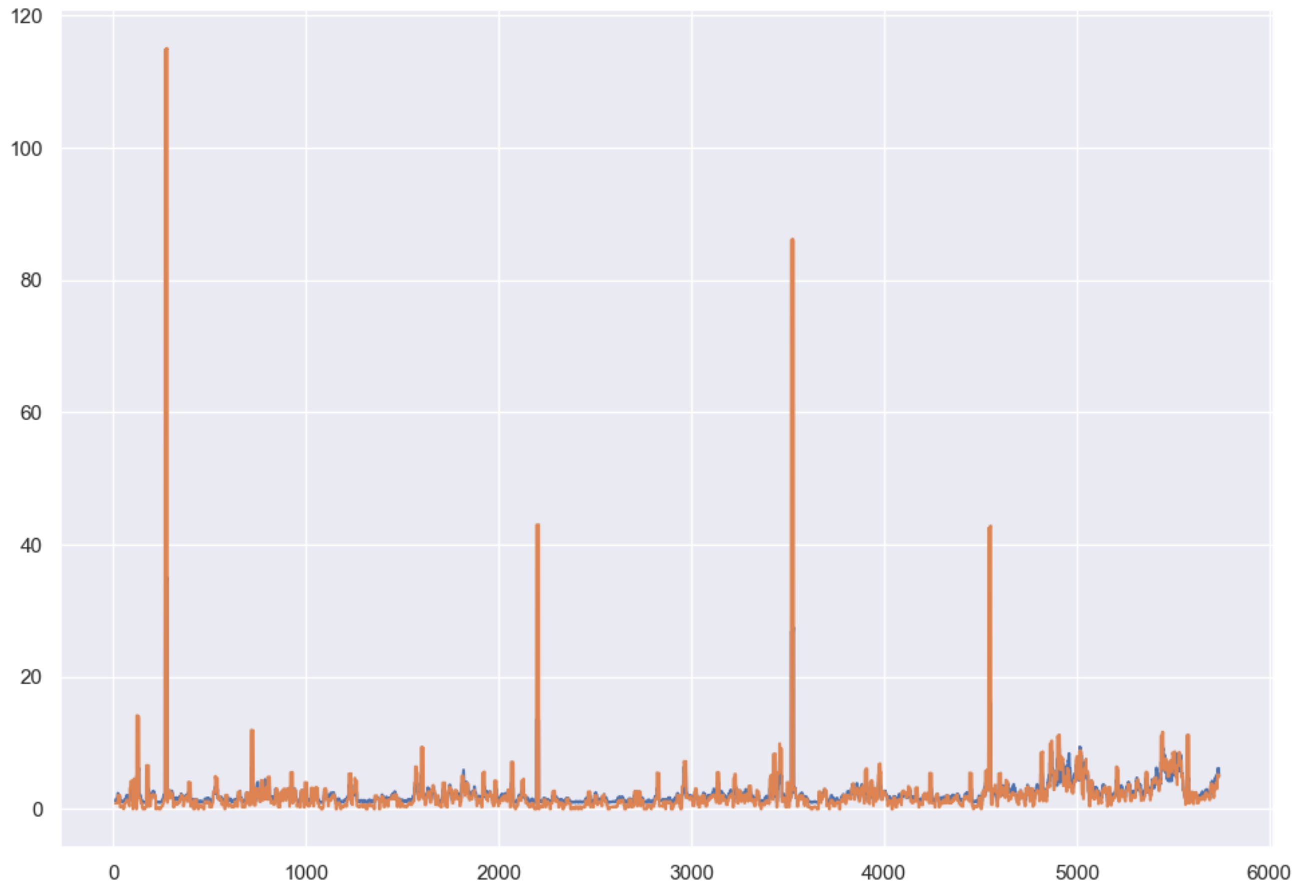
In [292…

```
# Plot the time series of Qty
df_biz.plot(y = 'Qty')
plt.show()
```



In [293…

```
# Plot the rolling mean and rolling standard deviation of the 'Qty' column
df_biz['Qty'].rolling(window=12).mean().plot()
df_biz['Qty'].rolling(window=12).std().plot()
plt.show()
```



In [294…

```
from statsmodels.tsa.stattools import adfuller

result = adfuller(df_biz['Qty'])
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
#https://machinelearningmastery.com/time-series-data-stationary-python/
```

```
ADF Statistic: -73.642596
p-value: 0.000000
```

A negative ADF statistic value, in this case -76, indicates that the time series is very likely to be stationary. This is because, in the ADF test, the null hypothesis is that there is a unit root (non-stationarity) in the time series, and a low p-value (typically less than 0.05) is used to reject the null hypothesis and conclude the time series is stationary.

There is the occasional large that represents the single large orders, other than that the data looks sationary which is backed up by our two different AdFuller test and visuals.

P,D, Q Another important aspect we will have to address soon is the parameter for the SARIMA time-series, which are denoted with 'P', 'D', and 'Q'. With the results of this ADFuller test we can assume our D paramter will be set to 0. The parameters are represented as follows:

p: is the order of the autoregressive term (AR), which is the number of lags used in the model. It describes the number of past values used to predict the next value. d: is the order of the differencing term (I), which is used to make the time series stationary by removing trends or seasonality. It represents the number of times the data has been differenced. q: is the order of the moving average term (MA), which is the error term that captures the short-term fluctuations in the data. It represents the number of past forecast errors used to predict the next value. The 'S' in SARIMA represents the seasonality aspect of the model, usually the notation is 'SARIMA(p,d,q)(P,D,Q)m' with 'm' being a constant such as 12(months).

To find these values I will perform a GridSearch, but first a few last things with our dataframe. A key component of a time-series model is converting the table to 'DateTimeIndex' which makes the 'Date' column the index and lets us use the date's frequency information in our models.

In [295…

```
# Data is stationary, changing to date time index.
# Convert the 'Date' column to a datetime object
df_biz['Date'] = pd.to_datetime(df_biz['Date'])

# Set the 'Date' column as the DataFrame index
df_biz.set_index('Date', inplace=True)
```

In [296…

```
df_biz.head()
```

Out[296...

Date	Product_ID	Transaction Type	Qty	Sales_Price	Amount
2018-03-23	FireWrap Grip Kit - Light Blue	Sales Receipt	1.0	24.95	24.95
2018-04-26	FireWrap Grip Kit - Light Blue	Sales Receipt	1.0	24.95	24.95
2018-04-27	FireWrap Grip Kit - Light Blue	Sales Receipt	1.0	24.95	24.95
2018-05-14	FireWrap Grip Kit - Light Blue	Sales Receipt	1.0	34.95	34.95
2018-06-22	FireWrap Grip Kit - Light Blue	Invoice	4.0	26.00	104.00

I will use the 'bfill' attribute which should fill missing values with the last valid observation and helps maintain integrity of the data when going through the model. We will also resample the table so that it is formatted to weeks instead of months which I think is better for this sized dataset.

In [297...

```
# The term bfill means that we use the value before filling in missing values
df_biz_model= df_biz.fillna(df_biz.bfill())

df_biz_model
```

Out[297...

Date	Product_ID	Transaction Type	Qty	Sales_Price	Amount
2018-03-23	FireWrap Grip Kit - Light Blue	Sales Receipt	1.0	24.95	24.95
2018-04-26	FireWrap Grip Kit - Light Blue	Sales Receipt	1.0	24.95	24.95
2018-04-27	FireWrap Grip Kit - Light Blue	Sales Receipt	1.0	24.95	24.95
2018-05-14	FireWrap Grip Kit - Light Blue	Sales Receipt	1.0	34.95	34.95
2018-06-22	FireWrap Grip Kit - Light Blue	Invoice	4.0	26.00	104.00
...	...	...	...	...	...
2022-11-28	FireWrap® Grip Kit Yellow	Invoice	7.0	21.75	152.25
2022-12-06	FireWrap® Grip Kit Yellow	Invoice	6.0	27.95	167.70
2022-12-12	FireWrap® Grip Kit Yellow	Invoice	5.0	21.75	108.75
2022-12-13	FireWrap® Grip Kit Yellow	Invoice	1.0	21.75	21.75
2022-12-27	FireWrap® Grip Kit Yellow	Invoice	1.0	21.75	21.75

5740 rows × 5 columns

In [298…

```
# Resampling to the data into groups by weeks starting on Saturday...
df_biz_weekly = df_biz_model.resample('W-SAT')
weekly_mean = df_biz_weekly.mean()
weekly_mean
```

Out[298…

Date	Qty	Sales_Price
2018-03-24	1.687500	26.575000
2018-03-31	1.000000	27.550000
2018-04-07	1.181818	27.313636
2018-04-14	1.235294	28.773529
2018-04-21	1.300000	26.355000
...	...	...
2022-12-17	2.723077	28.057692
2022-12-24	2.586207	30.743103
2022-12-31	1.543860	27.897368
2023-01-07	2.090909	29.967045
2023-01-14	2.909091	30.690909

252 rows × 2 columns

Models: We now have our data set up to where we can work with it, finally. This brings us to the meat and potatoes portion of the project, the modelling. To begin we need to identify our parameters which will be done via grid search, after that we will fit them to the model which will allow us to make predictions and evaluate. To give us a better idea of how the SARIMA model works, here is a brief summary:

SARIMA (Seasonal AutoRegressive Integrated Moving Average) models are a type of time series forecasting models that are used to model and predict future values based on past observations. They are an extension of the standard ARIMA (AutoRegressive Integrated Moving Average) models that include a seasonal component.

The basic structure of a SARIMA model is composed of three components:

AutoRegressive (AR) component: This component models the relationship between an observation and a number of lagged observations. It's represented by the parameter "p" in the SARIMA model.

Integrated (I) component: This component models the relationship between the observations and the differences between consecutive observations. It's represented by the parameter "d" in the SARIMA model.

Moving Average (MA) component: This component models the relationship between the observations and the error term (i.e. the difference between the actual observation and the prediction). It's represented by the parameter "q" in the SARIMA model.

Seasonal component: This component models the relationship between the observation and the lagged observations at the same time of the year. It's represented by the parameter "P", "D", and "Q" in the SARIMA model. These parameters of the model we will try to find by performing a grid search over different combinations of parameters.

Once the parameters are chosen, the model is trained on a set of historical data, and used to make predictions about future values. The model takes into account both the trend and the seasonality of the data.

https://neptune.ai/blog/arima-sarima-real-world-time-series-forecasting-guide
Regularization Measure The Bayesian Information Criterion (BIC) is a measure of the relative quality of statistical models. It is commonly used in the field of time series analysis to compare the quality of different models. BIC is a trade-off between the goodness of fit of the model and the complexity of the model. The lower the BIC score, the better the model fit is, and the simpler the model is.

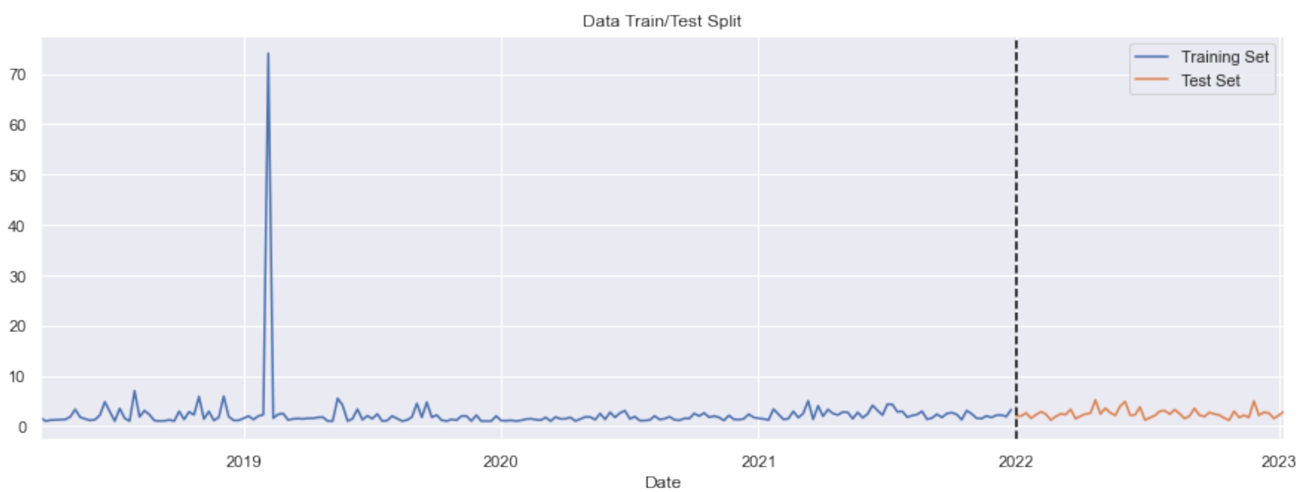
The Models As mentioned we will run a time-series models. The reason for doing this goes back to the original business problem. The owner does not want to be short on supplies but also does not want to take up unneeded shop space. We will begin by splitting the data into a train and test set and then use a grid search function on the test to get the parameters for the model and then begin fitting and predicting.

In [299…

```
#importing libraries to be used in model building
import statsmodels.api as sm
import itertools
from statsmodels.tsa.statespace.sarimax import SARIMAX
from itertools import product
from statsmodels.tsa.seasonal import seasonal_decompose

from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.graphics.tsaplots import plot_acf

from statsmodels.tsa.holtwinters import ExponentialSmoothing
from statsmodels.tsa.stattools import adfuller

from tqdm import tqdm_notebook
from itertools import product

%matplotlib inline
```

In [300…

```
# split data into 80/20 test by dates..
train_biz = weekly_mean['Qty'].loc[weekly_mean.index < '01-01-2022']
test_biz = weekly_mean['Qty'].loc[weekly_mean.index >= '01-01-2022']

fig, ax = plt.subplots(figsize=(15, 5))
train_biz.plot(ax=ax, label='Training Set', title='Data Train/Test Split')
test_biz.plot(ax=ax, label='Test Set')
ax.axvline('01-01-2022', color='black', ls='--')
ax.legend(['Training Set', 'Test Set'])
plt.show()
```



In [301…

```
# Define the p, d and q parameters to take any value between 0 and 3 (exclusi
p = d = q = range(0, 2)

# Generate all different combinations of p, q and q triplets
pdq = list(itertools.product(p, d, q))

# Generate all different combinations of seasonal p, q and q triplets
# Note: here we have 52 in the 's' position as we have weekly data
pdqs = [(x[0], x[1], x[2], 51) for x in list(itertools.product(p, d, q))]
```

In [302…

```
sarimax_gridsearch(train_biz, pdq, pdqs, freq='W-SAT')
```

```
SARIMAX (0, 0, 0) x (0, 0, 0, 51)51 : AIC Calculated =894.3594495161997
SARIMAX (0, 0, 0) x (0, 0, 1, 51)51 : AIC Calculated =279.87073966324687
SARIMAX (0, 0, 0) x (0, 1, 0, 51)51 : AIC Calculated =267.87448925034226
SARIMAX (0, 0, 0) x (0, 1, 1, 51)51 : AIC Calculated =4.0
SARIMAX (0, 0, 0) x (1, 0, 0, 51)51 : AIC Calculated =280.30966032060445
SARIMAX (0, 0, 0) x (1, 0, 1, 51)51 : AIC Calculated =270.72470389320324
SARIMAX (0, 0, 0) x (1, 1, 0, 51)51 : AIC Calculated =4.0
SARIMAX (0, 0, 0) x (1, 1, 1, 51)51 : AIC Calculated =6.0
SARIMAX (0, 0, 1) x (0, 0, 0, 51)51 : AIC Calculated =867.6395278095861
SARIMAX (0, 0, 1) x (0, 0, 1, 51)51 : AIC Calculated =264.50047931225026
SARIMAX (0, 0, 1) x (0, 1, 0, 51)51 : AIC Calculated =256.200873405359
SARIMAX (0, 0, 1) x (0, 1, 1, 51)51 : AIC Calculated =6.0
SARIMAX (0, 0, 1) x (1, 0, 0, 51)51 : AIC Calculated =281.85822528450916
SARIMAX (0, 0, 1) x (1, 0, 1, 51)51 : AIC Calculated =260.3342806939834
SARIMAX (0, 0, 1) x (1, 1, 0, 51)51 : AIC Calculated =6.0
SARIMAX (0, 0, 1) x (1, 1, 1, 51)51 : AIC Calculated =8.0
SARIMAX (0, 1, 0) x (0, 0, 0, 51)51 : AIC Calculated =873.5800991153773
SARIMAX (0, 1, 0) x (0, 0, 1, 51)51 : AIC Calculated =263.6388305552192
SARIMAX (0, 1, 0) x (0, 1, 0, 51)51 : AIC Calculated =263.3265419772717
SARIMAX (0, 1, 0) x (0, 1, 1, 51)51 : AIC Calculated =4.0
SARIMAX (0, 1, 0) x (1, 0, 0, 51)51 : AIC Calculated =275.7585846962404 6
SARIMAX (0, 1, 0) x (1, 0, 1, 51)51 : AIC Calculated =265.3154396947187
SARIMAX (0, 1, 0) x (1, 1, 0, 51)51 : AIC Calculated =4.0
SARIMAX (0, 1, 0) x (1, 1, 1, 51)51 : AIC Calculated =6.0
SARIMAX (0, 1, 1) x (0, 0, 0, 51)51 : AIC Calculated =822.1851653167263
SARIMAX (0, 1, 1) x (0, 0, 1, 51)51 : AIC Calculated =241.2128201478381
SARIMAX (0, 1, 1) x (0, 1, 0, 51)51 : AIC Calculated =239.93366794586314
SARIMAX (0, 1, 1) x (0, 1, 1, 51)51 : AIC Calculated =6.0
SARIMAX (0, 1, 1) x (1, 0, 0, 51)51 : AIC Calculated =265.2947311560555
SARIMAX (0, 1, 1) x (1, 0, 1, 51)51 : AIC Calculated =242.9839626889524
SARIMAX (0, 1, 1) x (1, 1, 0, 51)51 : AIC Calculated =6.0
SARIMAX (0, 1, 1) x (1, 1, 1, 51)51 : AIC Calculated =8.0
SARIMAX (1, 0, 0) x (0, 0, 0, 51)51 : AIC Calculated =869.6281474771138
SARIMAX (1, 0, 0) x (0, 0, 1, 51)51 : AIC Calculated =273.4715705740043
SARIMAX (1, 0, 0) x (0, 1, 0, 51)51 : AIC Calculated =268.26931477050005
SARIMAX (1, 0, 0) x (0, 1, 1, 51)51 : AIC Calculated =6.0
SARIMAX (1, 0, 0) x (1, 0, 0, 51)51 : AIC Calculated =270.18032308222695
SARIMAX (1, 0, 0) x (1, 0, 1, 51)51 : AIC Calculated =272.18032309596805
SARIMAX (1, 0, 0) x (1, 1, 0, 51)51 : AIC Calculated =6.0
SARIMAX (1, 0, 0) x (1, 1, 1, 51)51 : AIC Calculated =8.0
SARIMAX (1, 0, 1) x (0, 0, 0, 51)51 : AIC Calculated =831.8041627073536
```

```
SARIMAX (1, 0, 1) x (0, 0, 1, 51)51 : AIC Calculated =256.3474012798995
SARIMAX (1, 0, 1) x (0, 1, 0, 51)51 : AIC Calculated =254.9849127710569
SARIMAX (1, 0, 1) x (0, 1, 1, 51)51 : AIC Calculated =8.0
SARIMAX (1, 0, 1) x (1, 0, 0, 51)51 : AIC Calculated =267.98779696437487
SARIMAX (1, 0, 1) x (1, 0, 1, 51)51 : AIC Calculated =257.939581319815
SARIMAX (1, 0, 1) x (1, 1, 0, 51)51 : AIC Calculated =8.0
SARIMAX (1, 0, 1) x (1, 1, 1, 51)51 : AIC Calculated =10.0
SARIMAX (1, 1, 0) x (0, 0, 0, 51)51 : AIC Calculated =864.35975729623
SARIMAX (1, 1, 0) x (0, 0, 1, 51)51 : AIC Calculated =263.18301648595576
SARIMAX (1, 1, 0) x (0, 1, 0, 51)51 : AIC Calculated =261.7636706702853
SARIMAX (1, 1, 0) x (0, 1, 1, 51)51 : AIC Calculated =6.0
SARIMAX (1, 1, 0) x (1, 0, 0, 51)51 : AIC Calculated =262.60572147427484
SARIMAX (1, 1, 0) x (1, 0, 1, 51)51 : AIC Calculated =264.6057214784193
SARIMAX (1, 1, 0) x (1, 1, 0, 51)51 : AIC Calculated =6.0
SARIMAX (1, 1, 0) x (1, 1, 1, 51)51 : AIC Calculated =8.0
SARIMAX (1, 1, 1) x (0, 0, 0, 51)51 : AIC Calculated =824.1851644330973
SARIMAX (1, 1, 1) x (0, 0, 1, 51)51 : AIC Calculated =243.12568607967324
SARIMAX (1, 1, 1) x (0, 1, 0, 51)51 : AIC Calculated =241.93229782629638
SARIMAX (1, 1, 1) x (0, 1, 1, 51)51 : AIC Calculated =8.0
SARIMAX (1, 1, 1) x (1, 0, 0, 51)51 : AIC Calculated =255.58343451089814
SARIMAX (1, 1, 1) x (1, 0, 1, 51)51 : AIC Calculated =244.8930733815562
SARIMAX (1, 1, 1) x (1, 1, 0, 51)51 : AIC Calculated =8.0
SARIMAX (1, 1, 1) x (1, 1, 1, 51)51 : AIC Calculated =10.0
```

Out[302...

	pdq	pdqs	aic
0	(0, 0, 0)	(0, 0, 0, 51)	281.342607
34	(1, 0, 0)	(0, 1, 0, 51)	281.342607
35	(1, 0, 0)	(0, 1, 1, 51)	281.342607
36	(1, 0, 0)	(1, 0, 0, 51)	281.342607
37	(1, 0, 0)	(1, 0, 1, 51)	281.342607

In [305…

```
import pandas as pd
import statsmodels.api as sm

def sarimax_gridsearch(ts, pdq, pdqs, maxiter=50, freq='W-SAT'):
    ans = []
    for comb in pdq:
        for combs in pdqs:
            try:
                mod = sm.tsa.statespace.SARIMAX(train_biz,
                                                order=comb,
                                                seasonal_order=combs,
                                                enforce_stationarity=False,
                                                enforce_invertibility=False,
                                                freq=freq
                                                )
                output = mod.fit(maxiter=maxiter)
                ans.append([comb, combs, output.bic])
                print('SARIMAX {} x {} : BIC Calculated ={}'.format(comb, com
            except Exception as e:
                print('Error with combination:', comb, combs, 'Error:', e)
                continue

    ans_df = pd.DataFrame(ans, columns=['pdq', 'pdqs', 'bic'])
    ans_df = ans_df.sort_values(by=['bic'], ascending=True)[0:5]

    return ans_df
```

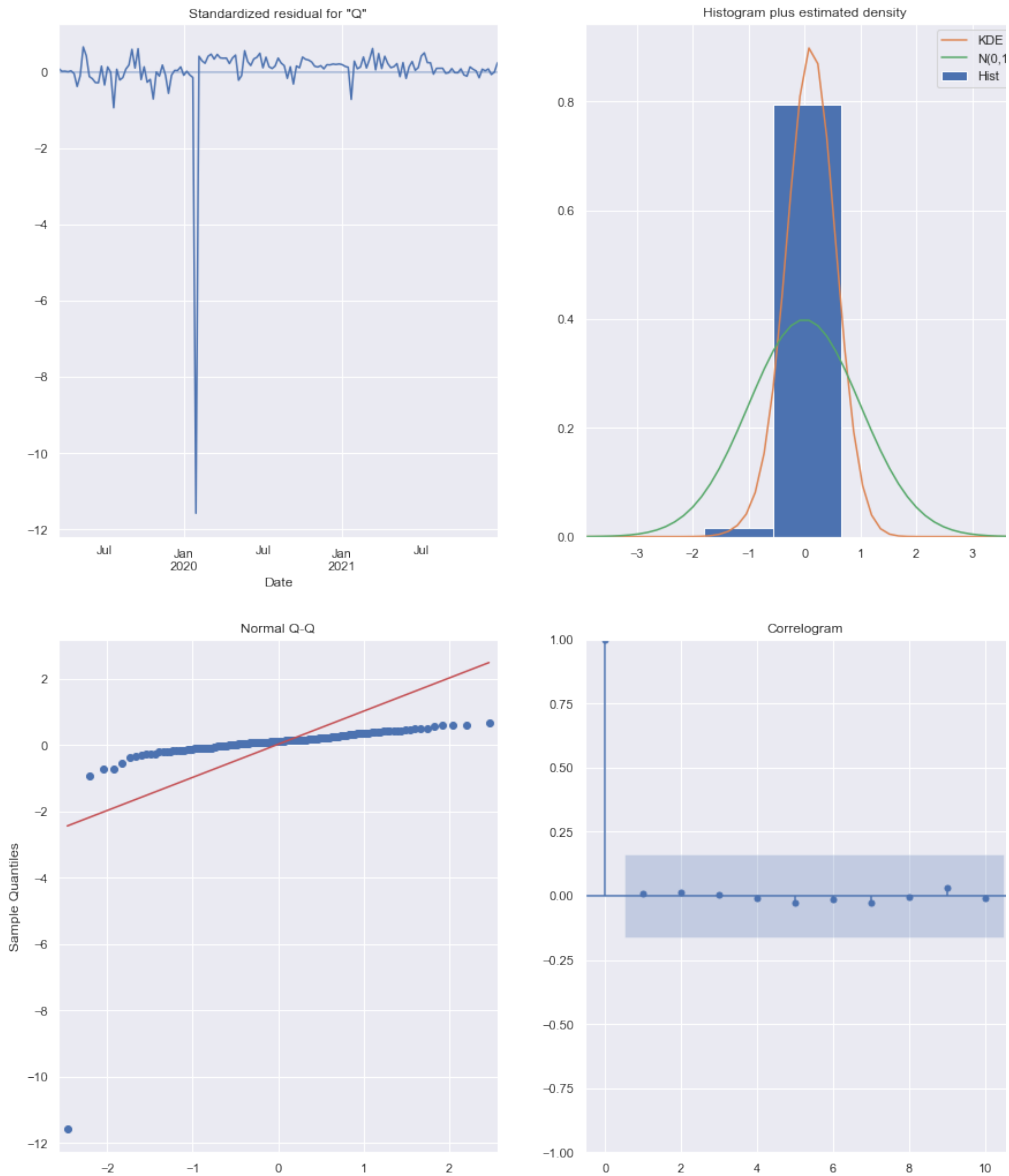
In [306…

```
# Plug the optimal parameter values into a new SARIMAX model
ARIMA_MODEL = sm.tsa.statespace.SARIMAX(train_biz,
                                        order=(0,1,1),
                                        seasonal_order=(0,1,1, 51),
                                        enforce_invertibility=False)

# Fit the model and print results
output = ARIMA_MODEL.fit()

print(output.summary().tables[1])
```

```
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1         -0.9709      0.101     -9.588      0.000      -1.169      -0.772
ma.S.L51      -0.1024      0.125     -0.822      0.411      -0.347       0.142
sigma2        38.6781      1.387     27.889      0.000      35.960      41.396
==============================================================================
```

In [308…

```
# Call plot_diagnostics() on the results calculated above
output.plot_diagnostics(figsize=(15, 18))
plt.show()
```



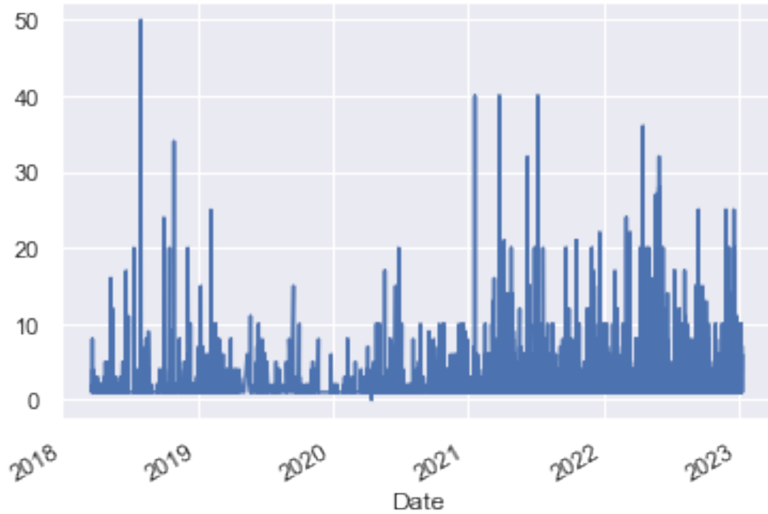
Once again assumptions arent too hot, removing outliers then will run it again.

In [309…

```
df_biz_outlier = df_biz[df_biz['Qty']<=100]
df_biz_outlier['Qty'].plot()
```

Out[309…

```
<AxesSubplot:xlabel='Date'>
```



In [310…

```
# Resampling to the data into groups by weeks starting on Saturday...
df_weekly_outlier = df_biz_outlier.resample('W-SAT')
weekly_mean_outlier = df_weekly_outlier.mean()
weekly_mean_outlier
```

Out[310…

Date	Qty	Sales_Price
2018-03-24	1.687500	26.575000
2018-03-31	1.000000	27.550000
2018-04-07	1.181818	27.313636
2018-04-14	1.235294	28.773529
2018-04-21	1.300000	26.355000
...	...	...
2022-12-17	2.723077	28.057692
2022-12-24	2.586207	30.743103
2022-12-31	1.543860	27.897368
2023-01-07	2.090909	29.967045
2023-01-14	2.909091	30.690909

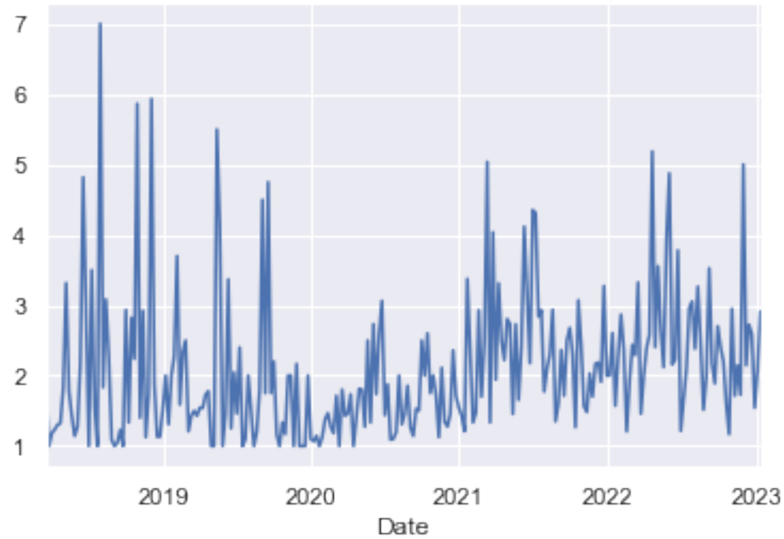
252 rows × 2 columns

In [311…

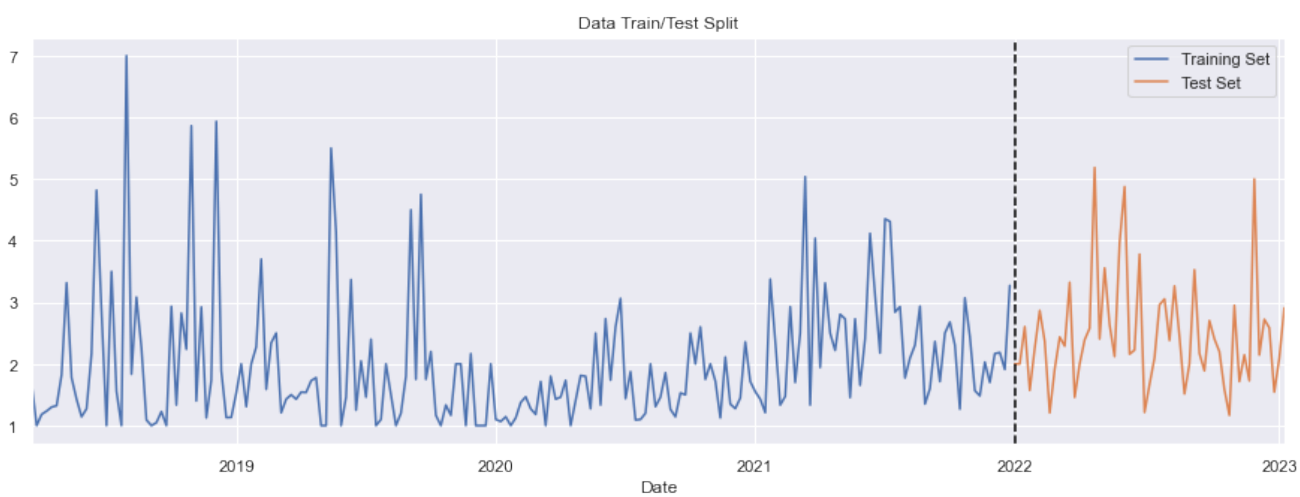
```
weekly_mean_outlier = weekly_mean_outlier.fillna(weekly_mean_outlier.bfill())
```

In [312…
```
weekly_mean_outlier['Qty'].plot()
```

Out[312… <AxesSubplot:xlabel='Date'>



In [313…
```
# split data into 80/20 test by dates..
train_outlier= weekly_mean_outlier['Qty'].loc[weekly_mean_outlier.index < '01
test_outlier = weekly_mean_outlier['Qty'].loc[weekly_mean_outlier.index >= '0

fig, ax = plt.subplots(figsize=(15, 5))
train_outlier.plot(ax=ax, label='Training Set', title='Data Train/Test Split'
test_outlier.plot(ax=ax, label='Test Set')
ax.axvline('01-01-2022', color='black', ls='--')
ax.legend(['Training Set', 'Test Set'])
plt.show()
```



In [318…

```
import pandas as pd
import statsmodels.api as sm

def sarimax_gridsearch(ts, pdq, pdqs, maxiter=50, freq='W-SAT'):
    ans = []
    for comb in pdq:
        for combs in pdqs:
            try:
                mod = sm.tsa.statespace.SARIMAX(train_outlier,
                                                order=comb,
                                                seasonal_order=combs,
                                                enforce_stationarity=False,
                                                enforce_invertibility=False,
                                                freq=freq
                                                )
                output_2 = mod.fit(maxiter=maxiter)
                ans.append([comb, combs, output.bic])
                print('SARIMAX {} x {} : BIC Calculated ={}'.format(comb, com
            except Exception as e:
                print('Error with combination:', comb, combs, 'Error:', e)
                continue

    ans_df = pd.DataFrame(ans, columns=['pdq', 'pdqs', 'bic'])
    ans_df = ans_df.sort_values(by=['bic'], ascending=True)[0:5]

    return ans_df
```

In [315…

```
sarimax_gridsearch(train_outlier, pdq, pdqs, freq='W-SAT')
```

```
SARIMAX (0, 0, 0) x (0, 0, 0, 51) : BIC Calculated =879.2275325513729
SARIMAX (0, 0, 0) x (0, 0, 1, 51) : BIC Calculated =6227.9170561241035
SARIMAX (0, 0, 0) x (0, 1, 0, 51) : BIC Calculated =524.3891124213662
SARIMAX (0, 0, 0) x (0, 1, 1, 51) : BIC Calculated =284.75351225513134
SARIMAX (0, 0, 0) x (1, 0, 0, 51) : BIC Calculated =516.1168772853546
SARIMAX (0, 0, 0) x (1, 0, 1, 51) : BIC Calculated =5953.865598793898
SARIMAX (0, 0, 0) x (1, 1, 0, 51) : BIC Calculated =291.3246020707876
SARIMAX (0, 0, 0) x (1, 1, 1, 51) : BIC Calculated =293.8059663113724
SARIMAX (0, 0, 1) x (0, 0, 0, 51) : BIC Calculated =785.0382288494981
SARIMAX (0, 0, 1) x (0, 0, 1, 51) : BIC Calculated =5040.563903579988
SARIMAX (0, 0, 1) x (0, 1, 0, 51) : BIC Calculated =521.2194440943506
SARIMAX (0, 0, 1) x (0, 1, 1, 51) : BIC Calculated =283.60167715876366
SARIMAX (0, 0, 1) x (1, 0, 0, 51) : BIC Calculated =506.1543865986851
SARIMAX (0, 0, 1) x (1, 0, 1, 51) : BIC Calculated =4680.504979974978
SARIMAX (0, 0, 1) x (1, 1, 0, 51) : BIC Calculated =290.82073694108624
SARIMAX (0, 0, 1) x (1, 1, 1, 51) : BIC Calculated =291.24663724687014
SARIMAX (0, 1, 0) x (0, 0, 0, 51) : BIC Calculated =685.7990695232527
SARIMAX (0, 1, 0) x (0, 0, 1, 51) : BIC Calculated =5764.212695525362
SARIMAX (0, 1, 0) x (0, 1, 0, 51) : BIC Calculated =584.3122776057868
SARIMAX (0, 1, 0) x (0, 1, 1, 51) : BIC Calculated =319.85088119534794
SARIMAX (0, 1, 0) x (1, 0, 0, 51) : BIC Calculated =456.44152376796853
SARIMAX (0, 1, 0) x (1, 0, 1, 51) : BIC Calculated =3519.4719117839477
SARIMAX (0, 1, 0) x (1, 1, 0, 51) : BIC Calculated =322.30737391488833
```

```
SARIMAX (0, 1, 0) x (1, 1, 1, 51) : BIC Calculated =323.9178719553233
SARIMAX (0, 1, 1) x (0, 0, 0, 51) : BIC Calculated =572.1006440857694
SARIMAX (0, 1, 1) x (0, 0, 1, 51) : BIC Calculated =4546.877489127882
SARIMAX (0, 1, 1) x (0, 1, 0, 51) : BIC Calculated =514.1198006127149
SARIMAX (0, 1, 1) x (0, 1, 1, 51) : BIC Calculated =268.91973319169756
SARIMAX (0, 1, 1) x (1, 0, 0, 51) : BIC Calculated =383.2746225594534
SARIMAX (0, 1, 1) x (1, 0, 1, 51) : BIC Calculated =4558.5370805127495
SARIMAX (0, 1, 1) x (1, 1, 0, 51) : BIC Calculated =276.95453000130067
SARIMAX (0, 1, 1) x (1, 1, 1, 51) : BIC Calculated =275.69016589095276
SARIMAX (1, 0, 0) x (0, 0, 0, 51) : BIC Calculated =675.0526635810741
SARIMAX (1, 0, 0) x (0, 0, 1, 51) : BIC Calculated =6180.289999926278
SARIMAX (1, 0, 0) x (0, 1, 0, 51) : BIC Calculated =521.9319002944123
SARIMAX (1, 0, 0) x (0, 1, 1, 51) : BIC Calculated =284.10271481777625
SARIMAX (1, 0, 0) x (1, 0, 0, 51) : BIC Calculated =451.3669630402569
SARIMAX (1, 0, 0) x (1, 0, 1, 51) : BIC Calculated =5817.528587987903
SARIMAX (1, 0, 0) x (1, 1, 0, 51) : BIC Calculated =286.3383891345685
SARIMAX (1, 0, 0) x (1, 1, 1, 51) : BIC Calculated =290.8393715129793
SARIMAX (1, 0, 1) x (0, 0, 0, 51) : BIC Calculated =579.8946229447386
SARIMAX (1, 0, 1) x (0, 0, 1, 51) : BIC Calculated =6252.99435398024
SARIMAX (1, 0, 1) x (0, 1, 0, 51) : BIC Calculated =519.9910793684221
SARIMAX (1, 0, 1) x (0, 1, 1, 51) : BIC Calculated =274.91163457334955
SARIMAX (1, 0, 1) x (1, 0, 0, 51) : BIC Calculated =388.248122381632
SARIMAX (1, 0, 1) x (1, 0, 1, 51) : BIC Calculated =5892.761701825491
SARIMAX (1, 0, 1) x (1, 1, 0, 51) : BIC Calculated =280.63701342652763
SARIMAX (1, 0, 1) x (1, 1, 1, 51) : BIC Calculated =281.64546284277907
SARIMAX (1, 1, 0) x (0, 0, 0, 51) : BIC Calculated =628.9342268634996
SARIMAX (1, 1, 0) x (0, 0, 1, 51) : BIC Calculated =5281.162119477299
SARIMAX (1, 1, 0) x (0, 1, 0, 51) : BIC Calculated =549.3630698996127
SARIMAX (1, 1, 0) x (0, 1, 1, 51) : BIC Calculated =295.9535442607899
SARIMAX (1, 1, 0) x (1, 0, 0, 51) : BIC Calculated =415.9666602597574
SARIMAX (1, 1, 0) x (1, 0, 1, 51) : BIC Calculated =5292.89457190748
SARIMAX (1, 1, 0) x (1, 1, 0, 51) : BIC Calculated =295.7060648375379
SARIMAX (1, 1, 0) x (1, 1, 1, 51) : BIC Calculated =299.0561874623089
SARIMAX (1, 1, 1) x (0, 0, 0, 51) : BIC Calculated =577.3525372300113
SARIMAX (1, 1, 1) x (0, 0, 1, 51) : BIC Calculated =3652.249754580993
SARIMAX (1, 1, 1) x (0, 1, 0, 51) : BIC Calculated =517.4267753455131
SARIMAX (1, 1, 1) x (0, 1, 1, 51) : BIC Calculated =273.42169418801666
SARIMAX (1, 1, 1) x (1, 0, 0, 51) : BIC Calculated =386.1126713251233
SARIMAX (1, 1, 1) x (1, 0, 1, 51) : BIC Calculated =3664.1728462212286
SARIMAX (1, 1, 1) x (1, 1, 0, 51) : BIC Calculated =278.90625628606267
SARIMAX (1, 1, 1) x (1, 1, 1, 51) : BIC Calculated =280.2021596017467
```

Out[315…

	pdq	pdqs	bic
27	(0, 1, 1)	(0, 1, 1, 51)	268.919733
59	(1, 1, 1)	(0, 1, 1, 51)	273.421694
43	(1, 0, 1)	(0, 1, 1, 51)	274.911635
31	(0, 1, 1)	(1, 1, 1, 51)	275.690166
30	(0, 1, 1)	(1, 1, 0, 51)	276.954530

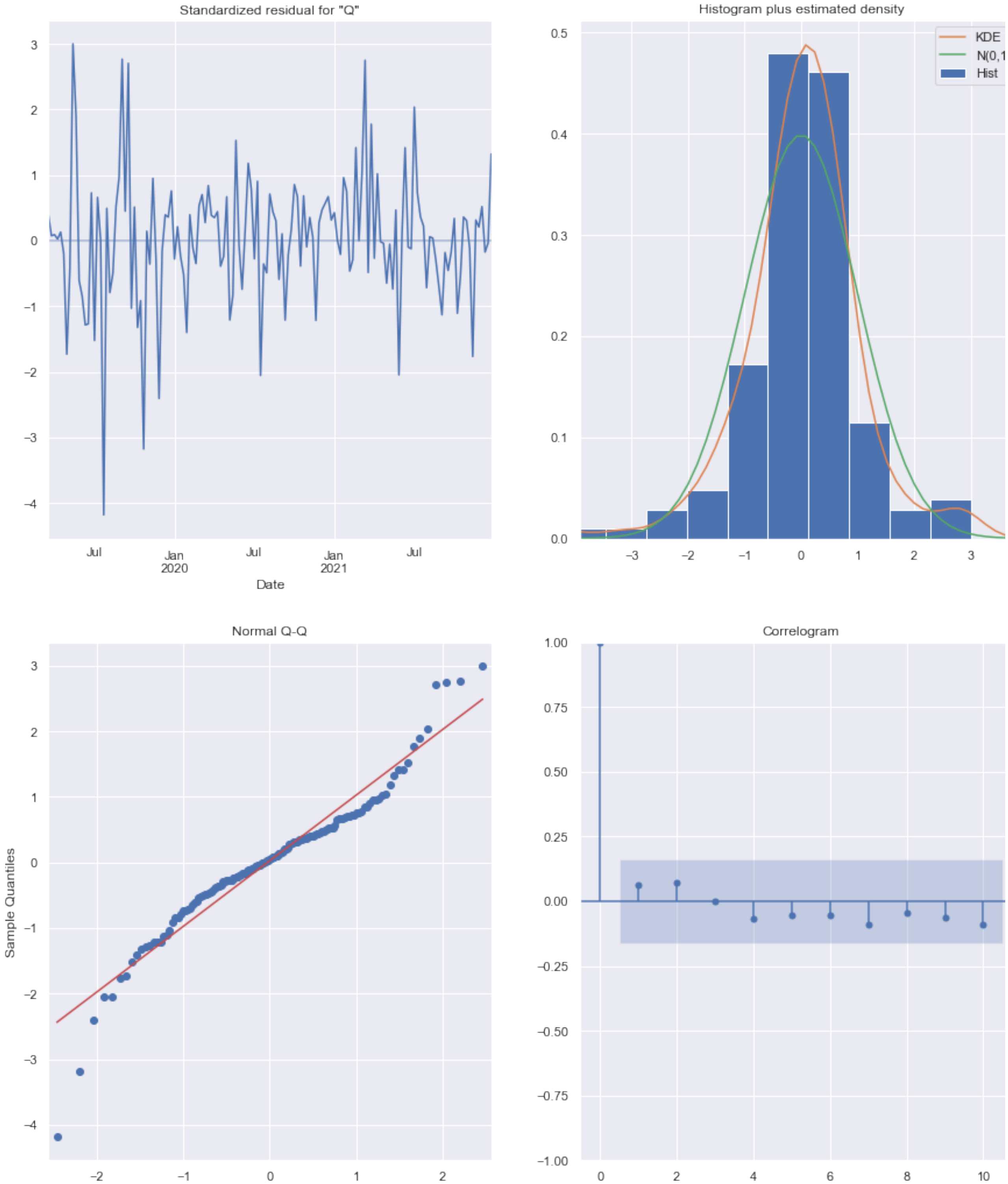
In [319…

```
# Plug the optimal parameter values into a new SARIMAX model
ARIMA_MODEL = sm.tsa.statespace.SARIMAX(train_outlier,
                                        order=(0,1,1),
                                        seasonal_order=(0,1,1, 51),
                                        enforce_invertibility=False)

# Fit the model and print results
output_2 = ARIMA_MODEL.fit()

print(output.summary().tables[1])
```

```
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1         -0.9709      0.101     -9.588      0.000      -1.169      -0.772
ma.S.L51      -0.1024      0.125     -0.822      0.411      -0.347       0.142
sigma2        38.6781      1.387     27.889      0.000      35.960      41.396
==============================================================================
```

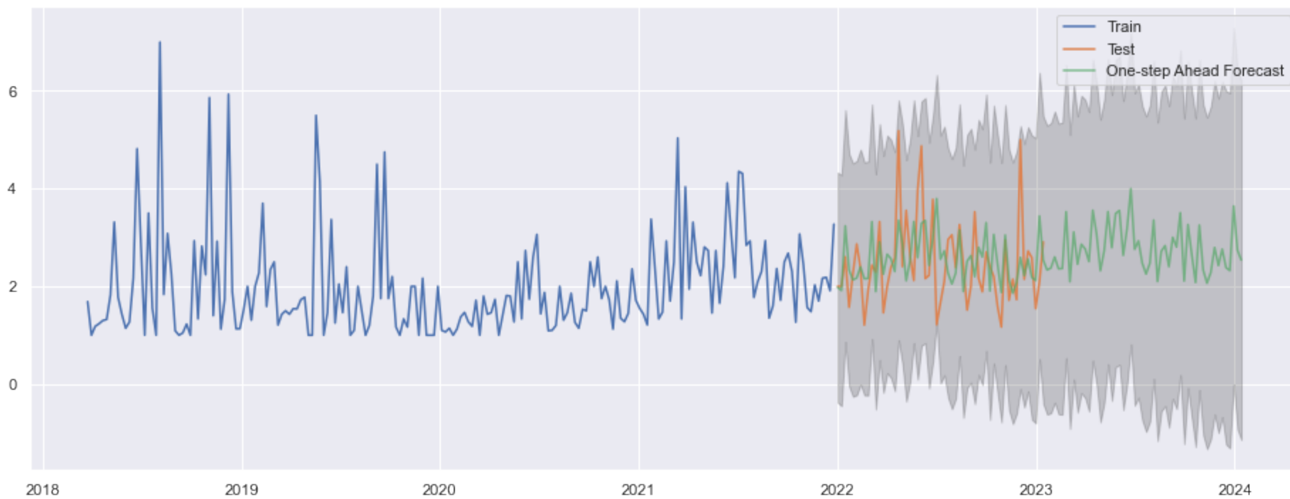
In [320…

```
# Call plot_diagnostics() on the results calculated above
output_2.plot_diagnostics(figsize=(15, 18))
plt.show()
```


Standardized residual for "Q"
Date
Jul
Jan 2020
Jan 2021
Histogram plus estimated density
KDE
N(0,1
Hist
Normal Q-Q
Sample Quantiles
Correlogram

In [321…

```
# Get the predicted values
pred = output_2.get_prediction(start=pd.to_datetime('01-01-2022'), end=pd.to_
pred_conf = pred.conf_int()

# Plot the actual values and predicted values
plt.figure(figsize=(16, 6))
plt.plot(train_outlier, label='Train')
plt.plot(test_outlier, label='Test')
plt.plot(pred.predicted_mean, label='One-step Ahead Forecast', alpha=.7)

# Shade the area between the confidence intervals
plt.fill_between(pred_conf.index, pred_conf.iloc[:, 0], pred_conf.iloc[:, 1],

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



In [324…

```
pred2 = output_2.get_prediction(start=pd.to_datetime('01-01-2022'), end=pd.to_
pred_mean2 = pred.predicted_mean
print('One Step Ahead')
print('Predicted Weekly Mean of Quantity Sold')
print(pred_mean2.tail())
print('*****')
pred_conf2 = pred2.conf_int()
print('Confidence Interval:')
print(pred_conf2.tail())
print('*****')
print(pred_mean2.describe())
```

```
One Step Ahead
Predicted Weekly Mean of Quantity Sold
2023-12-16    2.384234
2023-12-23    2.323122
2023-12-30    3.644494
2024-01-06    2.738996
2024-01-13    2.539155
Freq: W-SAT, Name: predicted_mean, dtype: float64
*****
Confidence Interval:
            lower Qty  upper Qty
2023-12-16  -1.234674   6.003142
2023-12-23  -1.311411   5.957654
2023-12-30  -0.005596   7.294584
2024-01-06  -0.926585   6.404578
2024-01-13  -1.141853   6.220164
*****
count    107.000000
mean       2.649784
std        0.498495
min        1.862035
25%        2.270758
50%        2.557417
75%        2.988900
max        3.999040
Name: predicted_mean, dtype: float64
```

In [325…

```
# Get the real and predicted values
Qty_forecasted = pred2.predicted_mean
Qty_truth = test_outlier['2022-01-01':]

# Compute the mean square error
mse = ((Qty_forecasted - Qty_truth) ** 2).mean()
print('The Mean Squared Error of our forecasts is {}'.format(round(mse, 2)))
```

```
The Mean Squared Error of our forecasts is 0.79
```

In [ ]: