Forecasting

July 23, 2025

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[3]:
```
df = pd.read_csv(r"C:\Users\Raushan\Desktop\Sales Forecasting\uncleaned_stores_sales_forecasting.csv")
```

[4]:
```
df.head()
```

[4]:

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	…	Postal Code	Region	Product ID	Category	Sub-Category	Product Name	Sales	Quantity
0	1.0	CA-2016-152156	11/8/2016	11/11/2016	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	…	42420.0	South	FUR-BO-10001798	Furniture	Bookcases	Bush Somerset Collection Bookcase	261.9600	2.0
1	2.0	CA-2016-152156	11/8/2016	11/11/2016	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	…	42420.0	South	FUR-CH-10000454	Furniture	Chairs	Hon Deluxe Fabric Upholstered Stacking Chairs,…	731.9400	3.0
2	4.0	US-2015-108966	10/11/2015	10/18/2015	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	…	33311.0	South	FUR-TA-10000577	Furniture	Tables	Bretford CR4500 Series Slim Rectangular Table	957.5775	5.0
3	6.0	CA-2014-115812	6/9/2014	6/14/2014	Standard Class	BH-11710	Brosina Hoffman	Consumer	United States	Los Angeles	…	90032.0	West	FUR-FU-10001487	Furniture	Furnishings	Eldon Expressions Wood and Plastic Desk Access…	NaN	7.0
4	11.0	CA-2014-115812	6/9/2014	6/14/2014	Standard Class	BH-11710	Brosina Hoffman	Consumer	United States	Los Angeles	…	90032.0	West	FUR-TA-10001539	Furniture	Tables	Chromcraft Rectangular Conference Tables	1706.1840	9.0

```
   Discount    Profit
0      0.00   41.9136
1      0.00  219.5820
2      0.45 -383.0310
3      0.00   14.1694
4      0.20   85.3092

[5 rows x 21 columns]
```

[5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2227 entries, 0 to 2226
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Row ID         2114 non-null   float64
 1   Order ID       2120 non-null   object
 2   Order Date     2114 non-null   object
 3   Ship Date      2115 non-null   object
 4   Ship Mode      2119 non-null   object
 5   Customer ID    2115 non-null   object
 6   Customer Name  2116 non-null   object
 7   Segment        2119 non-null   object
 8   Country        2115 non-null   object
 9   City           2117 non-null   object
 10  State          2118 non-null   object
 11  Postal Code    2117 non-null   float64
 12  Region         2114 non-null   object
 13  Product ID     2117 non-null   object
 14  Category       2116 non-null   object
 15  Sub-Category   2116 non-null   object
 16  Product Name   2117 non-null   object
 17  Sales          2113 non-null   float64
 18  Quantity       2116 non-null   float64
 19  Discount       2117 non-null   float64
 20  Profit         2120 non-null   float64
dtypes: float64(6), object(15)
memory usage: 365.5+ KB
```

[6]:
```
df.shape
```

[6]:
```
(2227, 21)
```

[7]:
```
df.isnull().sum()/df.shape[0]*100
```

[7]:
```
Row ID          5.074091
Order ID        4.804670
```

```
Order Date       5.074091
Ship Date        5.029187
Ship Mode        4.849573
Customer ID      5.029187
Customer Name    4.984284
Segment          4.849573
Country          5.029187
City             4.939380
State            4.894477
Postal Code      4.939380
Region           5.074091
Product ID       4.939380
Category         4.984284
Sub-Category     4.984284
Product Name     4.939380
Sales            5.118994
Quantity         4.984284
Discount         4.939380
Profit           4.804670
dtype: float64
```

Data Processing

[8]:
```
df['Order Date'] = pd.to_datetime(df['Order Date'])
df['Ship Date'] = pd.to_datetime(df['Ship Date'])
```

[9]:
```
df= df.drop(columns=['Postal Code','Product ID','Row ID'], axis=1)
```

[10]:
```
df.duplicated().sum()
```

[10]:
```
np.int64(106)
```

[11]:
```
df= df.drop_duplicates()
```

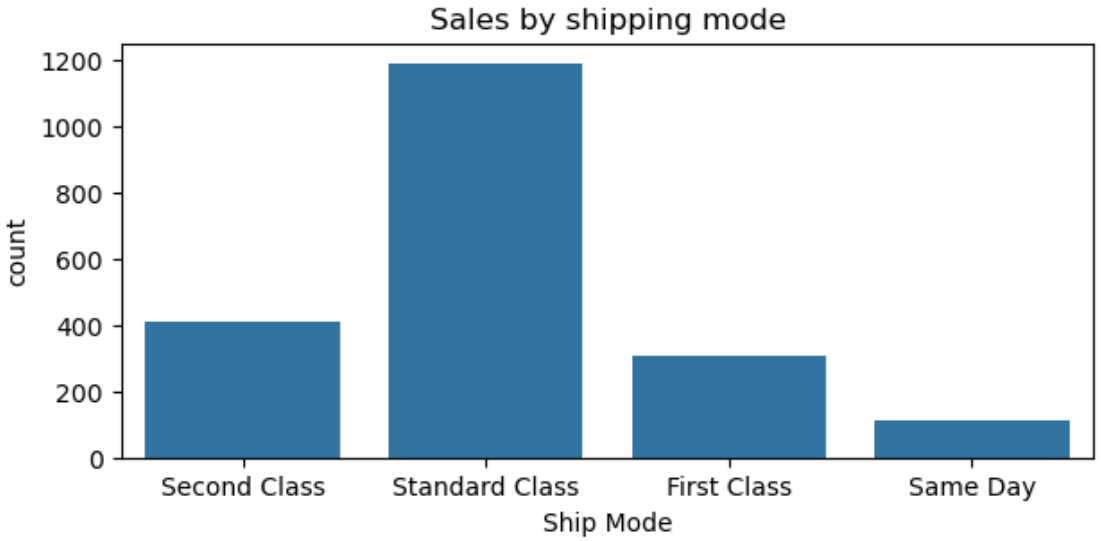
[12]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2121 entries, 0 to 2120
Data columns (total 18 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Order ID       2015 non-null   object
 1   Order Date     2015 non-null   datetime64[ns]
 2   Ship Date      2015 non-null   datetime64[ns]
 3   Ship Mode      2015 non-null   object
 4   Customer ID    2015 non-null   object
 5   Customer Name  2015 non-null   object
 6   Segment        2015 non-null   object
 7   Country        2015 non-null   object
```

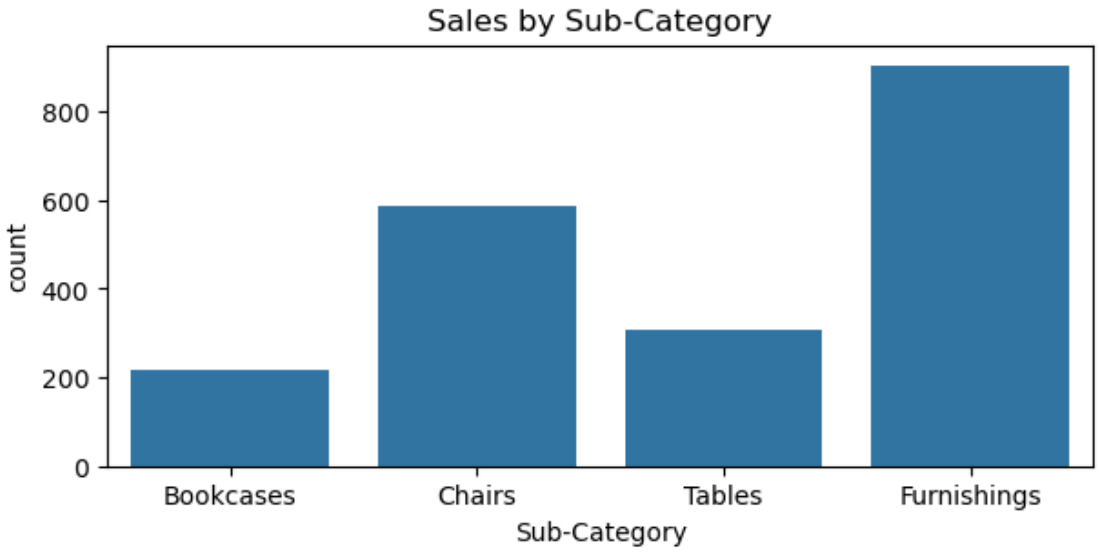
```
 8   City           2015 non-null   object
 9   State          2015 non-null   object
 10  Region         2015 non-null   object
 11  Category       2015 non-null   object
 12  Sub-Category   2015 non-null   object
 13  Product Name   2015 non-null   object
 14  Sales          2015 non-null   float64
 15  Quantity       2015 non-null   float64
 16  Discount       2015 non-null   float64
 17  Profit         2015 non-null   float64
dtypes: datetime64[ns](2), float64(4), object(12)
memory usage: 314.8+ KB
```

Exploratory Data Analysis (EDA)

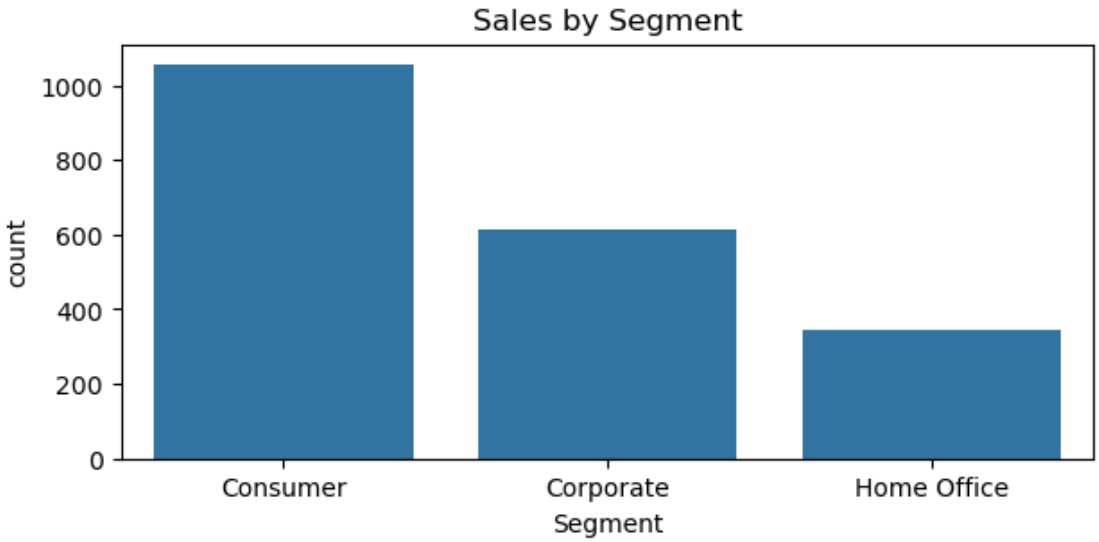
[13]:
```
plt.figure(figsize=(7,3))
sns.countplot(x="Ship Mode" , data= df)
plt.title("Sales by shipping mode")
plt.show()
```



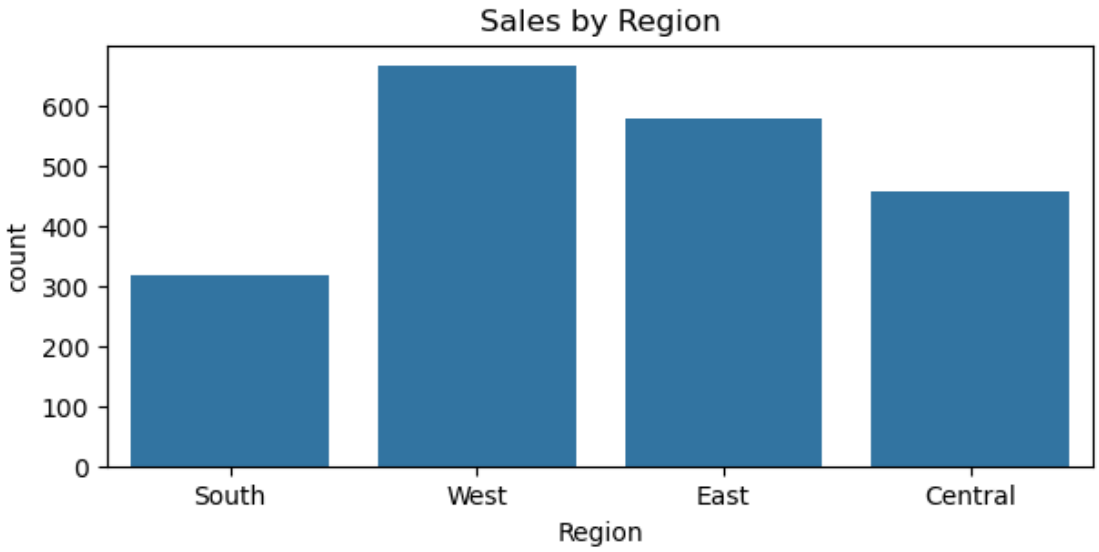
[14]:
```
plt.figure(figsize=(7,3))
sns.countplot(x="Sub-Category" , data= df)
plt.title("Sales by Sub-Category")
plt.show()
```



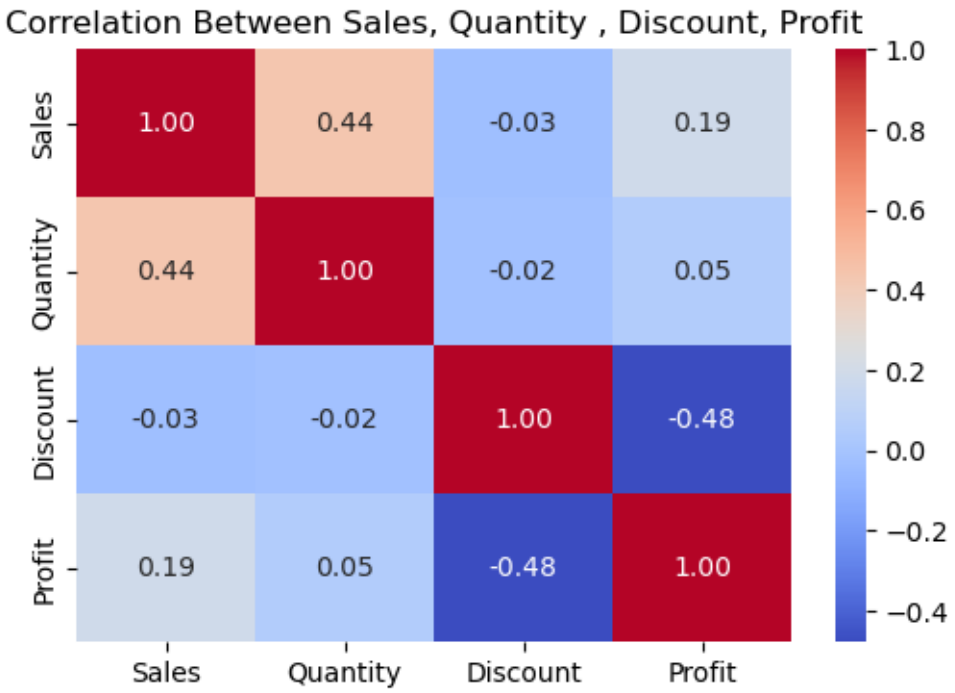
[15]:
```
plt.figure(figsize=(7,3))
sns.countplot(x="Segment" , data= df)
plt.title("Sales by Segment")
plt.show()
```



[16]:
```
plt.figure(figsize=(7,3))
sns.countplot(x="Region" , data= df)
plt.title("Sales by Region")
plt.show()
```



[17]:
```
selected_cols = ['Sales', 'Quantity', 'Discount', 'Profit']
corr_matrix = df[selected_cols].corr()

# Select specific numeric columns
selected_cols = ['Sales', 'Quantity', 'Discount', 'Profit']  # Add more if␣
 ↪needed
corr_matrix = df[selected_cols].corr()

# Plot correlation matrix
plt.figure(figsize=(6, 4))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Between Sales, Quantity , Discount, Profit")
plt.show()
```

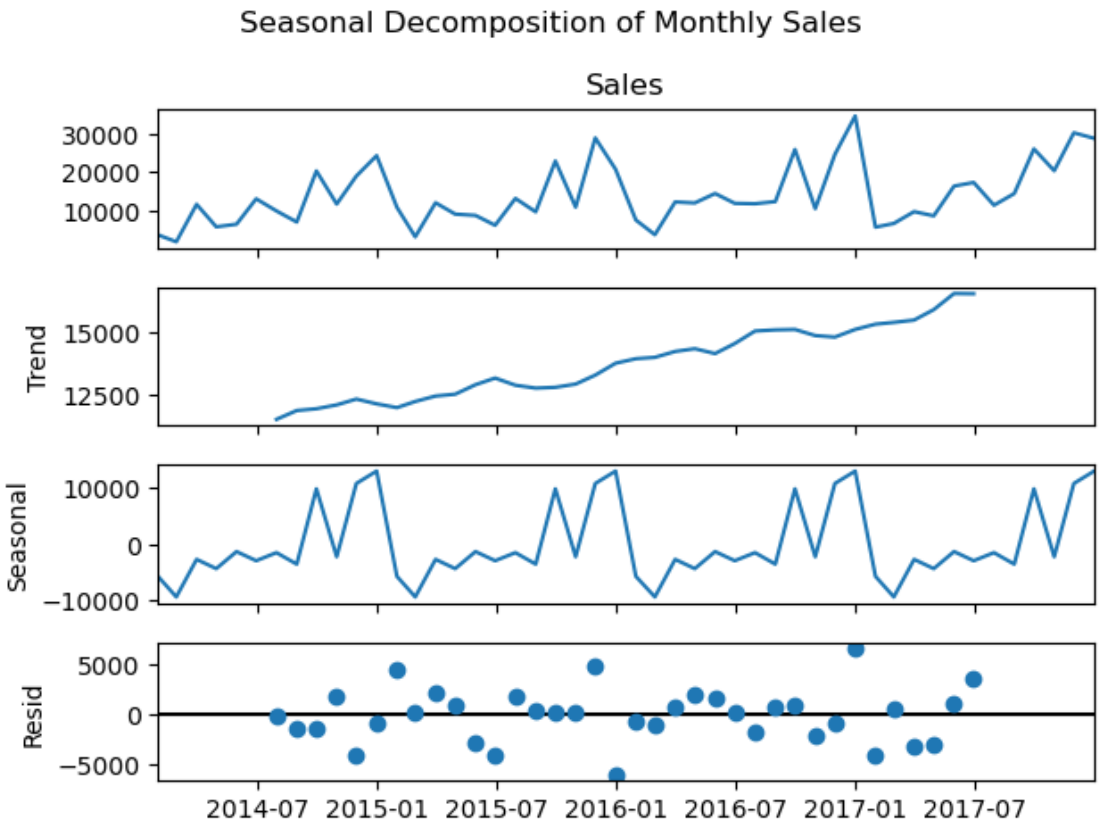


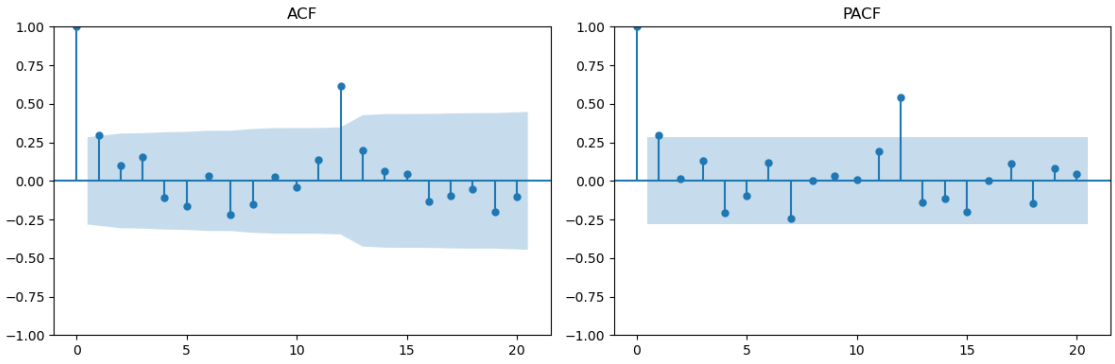
Time Series Analysis

[18]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# Convert the date column to datetime
df['Order Date'] = pd.to_datetime(df['Order Date'])  # Replace 'Date' with your␣
 ↪actual date column name

# Set Date as index
df.set_index('Order Date', inplace=True, drop=False)


# Resample to monthly frequency
monthly_sales = df['Sales'].resample('M').sum()  # Replace 'Sales' with your␣
 ↪actual sales column name

# Seasonal decomposition
decomposition = seasonal_decompose(monthly_sales, model='additive')
decomposition.plot()
plt.suptitle('Seasonal Decomposition of Monthly Sales')
plt.tight_layout()
plt.show()
```

```
# ACF & PACF plots
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plot_acf(monthly_sales.dropna(), lags=20, ax=plt.gca())
plt.title("ACF")

plt.subplot(1, 2, 2)
plot_pacf(monthly_sales.dropna(), lags=20, ax=plt.gca())
plt.title("PACF")

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

```
C:\Users\Raushan\AppData\Local\Temp\ipykernel_2236\3648680849.py:12:
FutureWarning: 'M' is deprecated and will be removed in a future version, please
use 'ME' instead.
  monthly_sales = df['Sales'].resample('M').sum()  # Replace 'Sales' with your
actual sales column name
```





[19]:
```
start_date = df['Order Date'].min()
print("Start date of the forecast:",start_date)
end_date = df['Order Date'].max()
print("end date of the forecast:",end_date)
```

```
Start date of the forecast: 2014-01-06 00:00:00
end date of the forecast: 2017-12-30 00:00:00
```

Forecasting with Prophet

[20]:
```
from prophet import Prophet

# Initialize the Model
model=Prophet()
```

[21]:
```
from prophet import Prophet
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

# Prepare data
df_prophet = df[['Order Date', 'Sales']].rename(columns={'Order Date': 'ds',␣
 ↪'Sales': 'y'})

# Train-test split
train = df_prophet[df_prophet['ds'] < '2025-01-01']
test = df_prophet[df_prophet['ds'] >= '2014-01-06']

# Model training
model = Prophet()
model.fit(train)

### Create future dates of 365 days
future = model.make_future_dataframe(periods=365)
forecast = model.predict(future)
```

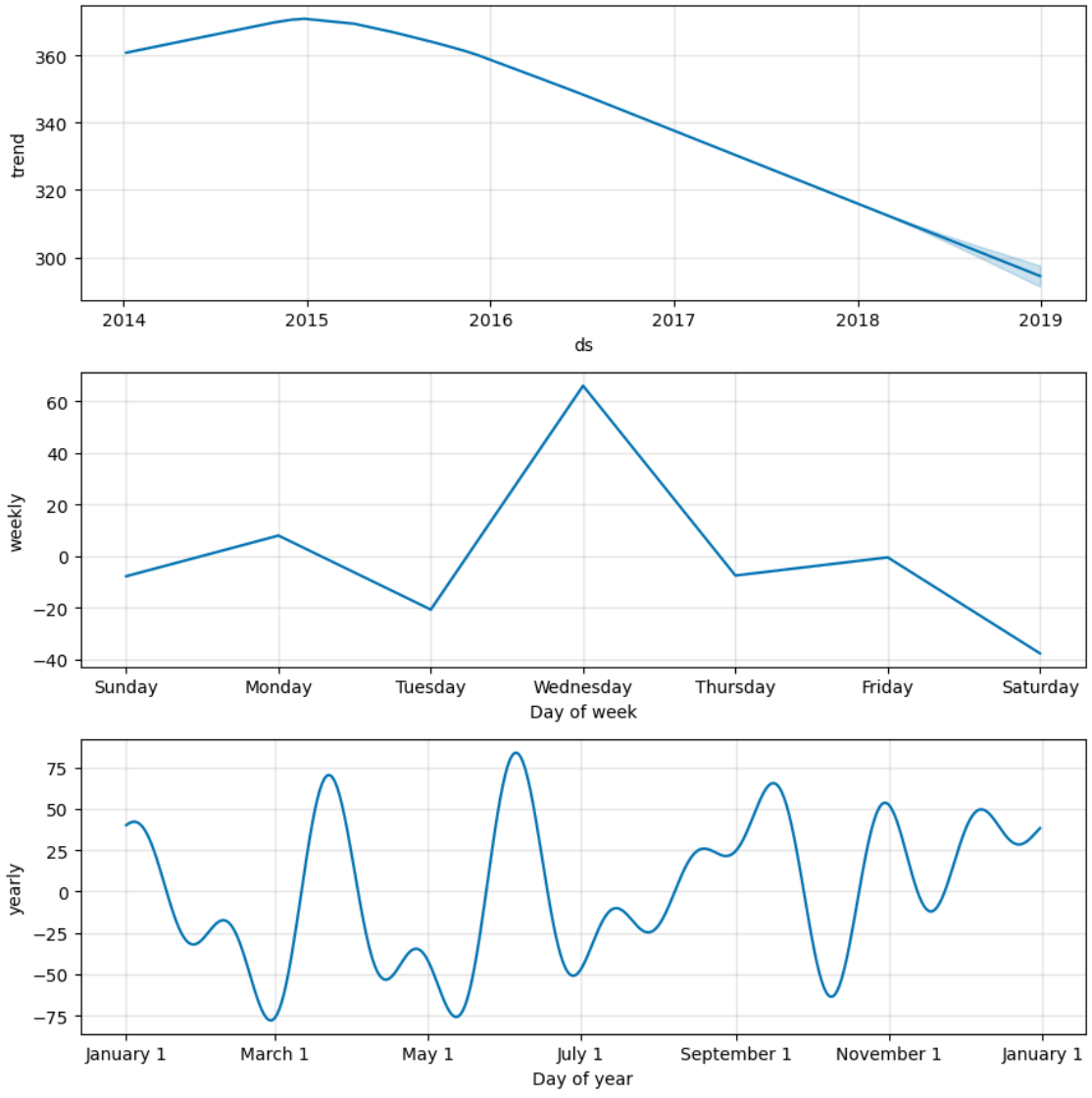
```
19:57:44 - cmdstanpy - INFO - Chain [1] start processing
19:57:44 - cmdstanpy - INFO - Chain [1] done processing
```

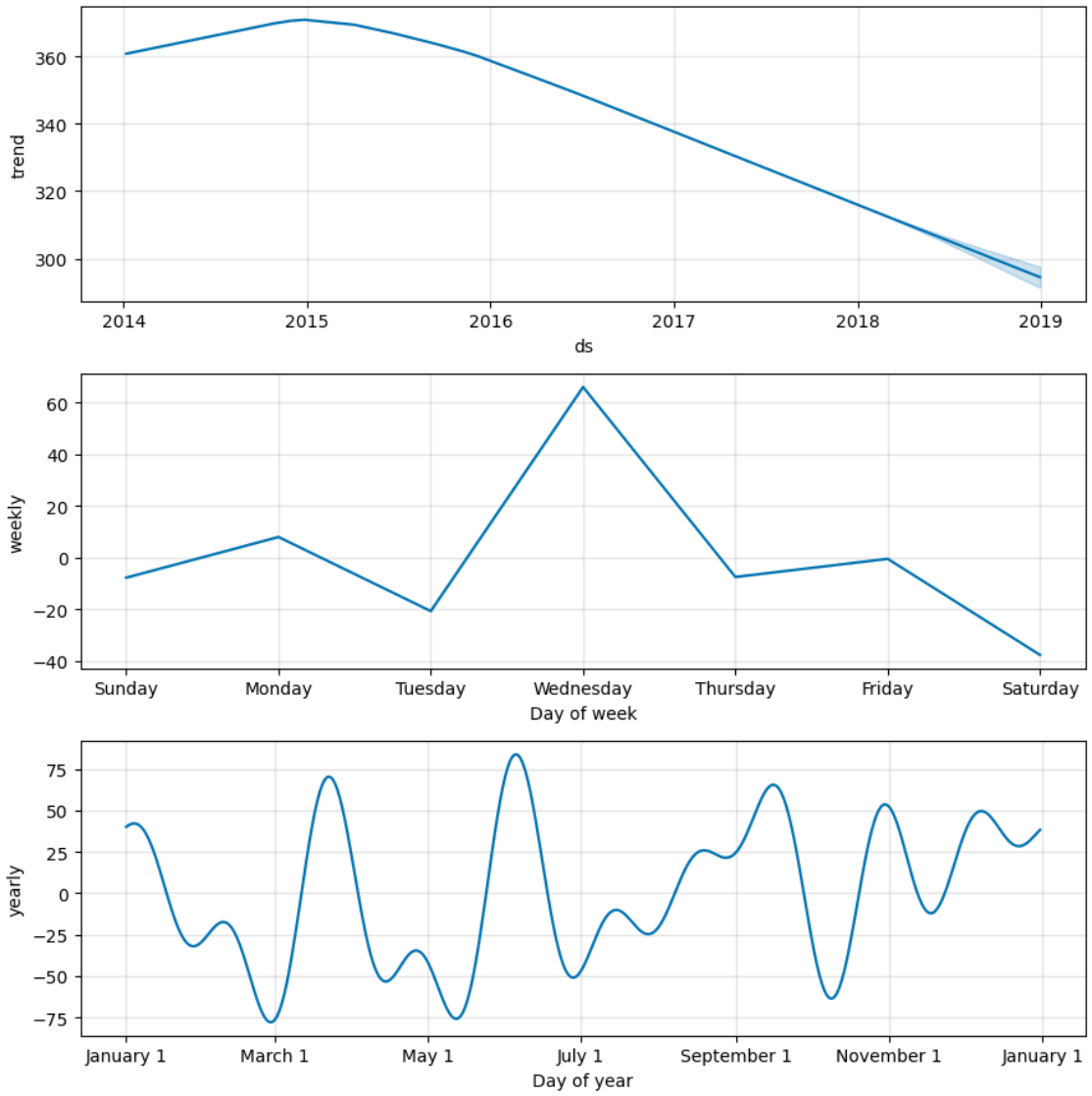
[22]:
```
print("Length of forecast:", len(forecast))
print("Length of test set:", len(test))
```

```
Length of forecast: 1238
Length of test set: 2015
```

[23]:
```
#### Visualize Each Components[Trends,yearly]
model.plot_components(forecast)
```

[23]:





Model Evaluation

[24]:
```
future = model.make_future_dataframe(periods=90, freq='D')
forecast = model.predict(future)
```

[25]:
```
import numpy as np

# Assuming 'test' is your actual DataFrame with the ground truth values
y_true = test['y'].values

# Remove NaNs from y_true
y_true = y_true[~np.isnan(y_true)]
```

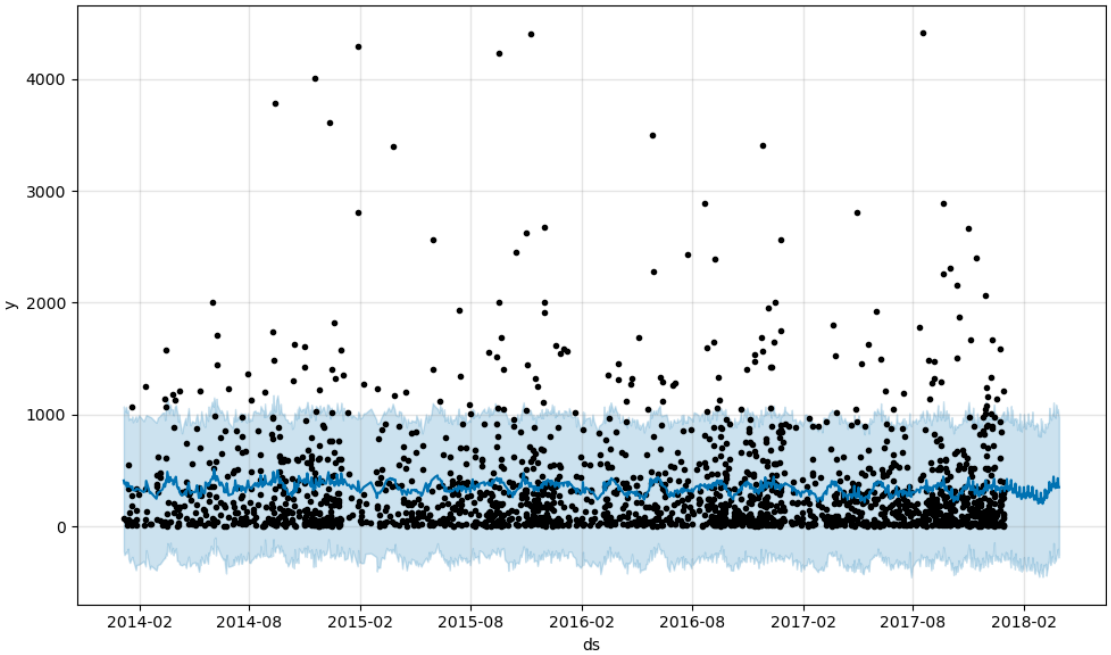
[26]:
```
y_true = y_true[~np.isnan(y_true)]
```

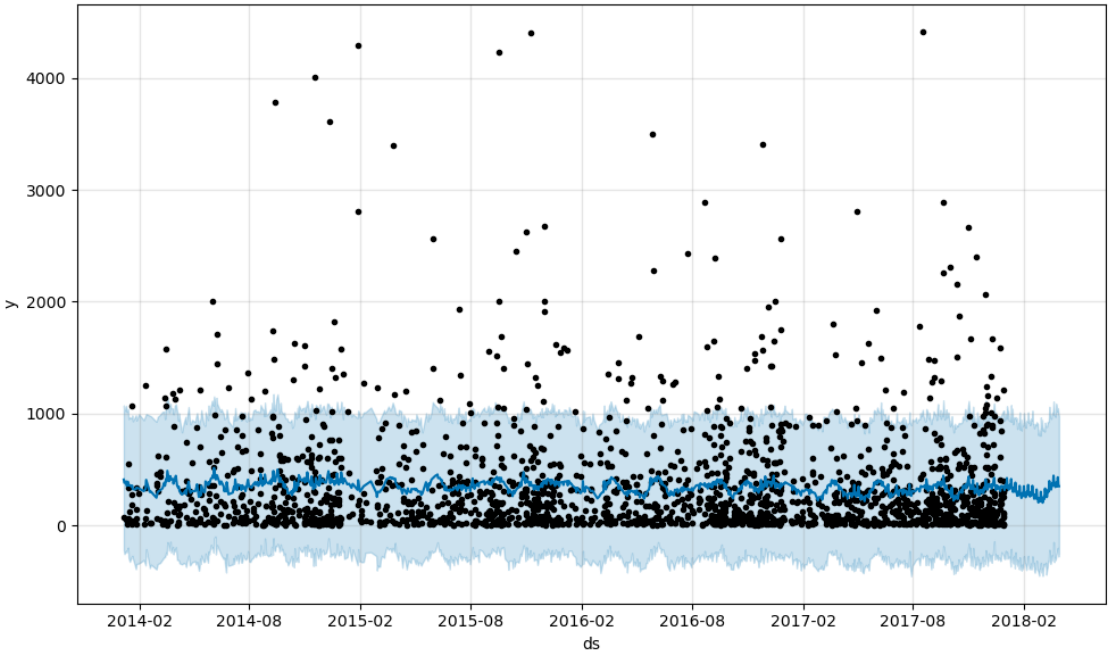
[27]:
```
import numpy as np
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Raw series
y_true = test['y'].values
y_pred = forecast['yhat'][-len(test):].values  # or adjust index depending on␣
 ↪forecast range

# Step 1: Align lengths (take the shorter one)
min_len = min(len(y_true), len(y_pred))
y_true = y_true[-min_len:]
y_pred = y_pred[-min_len:]

# Step 2: Remove NaNs from both
valid_mask = ~np.isnan(y_true) & ~np.isnan(y_pred)
y_true_clean = y_true[valid_mask]
y_pred_clean = y_pred[valid_mask]

# Step 3: Evaluate
print("MAE:", mean_absolute_error(y_true_clean, y_pred_clean))
print("RMSE:", np.sqrt(mean_squared_error(y_true_clean, y_pred_clean)))
```

```
MAE: 329.38915883026334
RMSE: 555.2939303525181
```

[28]:
```
model.plot(forecast)
```

[28]:




4000
3000
2000
1000
0
y
2014-02
2014-08
2015-02
2015-08
2016-02
2016-08
2017-02
2017-08
2018-02
ds

[ ]: