Part 1: Research Question

A. Purpose of Data Analysis

1. Research question: What are the expected future revenue trends for the telecom company over the next four months?

2. Objective and goals: The objective of this project is 3-fold. First, I want to decompose the data to help identify trends and inspect for seasonality. Next, I will build and tune an ARIMA model to forecast future revenue. Lastly, I will evaluate the model's performance.

Part 2: Method Justification

Summary of Assumptions

1. Stationarity: Stationarity assumes that the statistical properties (mean, variance, etc.) remain consistent over time. Stationarity is important to ensure the model remains stable and effectively project future data (De Jesus, 2022).

2. Autocorrelated Data: Autocorrelation assumes that the data is inherently correlated at regular intervals. ARIMA models require that the previous data is predictive of the future data in order to form the models.

Part 3: Data Preperation

Data Cleaning Summary

1. Line Graph: This link will take you to the code and visualization for the initial line graph and realization of the time series.

2. Time Setting: The data consists of daily records therefor I indexed the date column starting at 2023-01 -01 through the 730 entries. I feel confident there were no gaps in the records due to the lack of null values and/or outliers. The data was recorded for 730 days which is 2 years.

3. Stationarity: In the line graph we can see a clear upward trend. This indicates a lack of stationarity and therefore I applied first-order differencing to ensure stationarity. After differencing I ran an adf test which resulted in a pvalue of < 0.05 confirming stationarity.

4. Preperation Steps: To prepare the data for the model I ensured there were no null values, differenced the data to ensure stationarity, and split the data using an 80/20 training/test split.

5. Clean Data: The cleaned data set is attatched seperatly to this submission.

Part 4: Model Identification and Analysis

Analysis of the time series data set

- Seasonal Component: I determined that the data does not have a seasonal component. This was determined by data decomposition and spectral density plots. Neither analysis show significant patterns that would suggest the presence of seasonality.

- Trend analysis: The original line graph showed a clear trend present in the data. This was corrected by differencing the data and the decomposition shows that there is no clear trend after differencing.

- ACF & PACF: The ACF and PACF showed a correlation at lag 1 which suggests the use of an AR(1) model.

- Spectral Density: The spectral density plot shows the data as peaks over frequency. The lack of significant spikes reinforces the absence of seasonality.

- Decomposed Data & Residuals : The decomposed data shows the lack of trend, seasonality, and autocorrelation via the residual plot.

- ARIMA Model Identification : Given the interpretation of the ACF and PACF in combination with the results of the AIC the best model would be the first-order autoregressive model, ARIMA(1,1,0).

1. Forecast: I used the model to forecast over the test period and then plotted the results over the actuals to compare and visualize the performance.

2. Output and Calculations: The model performed well with an MSE of 5.6327 and MAE of 1.8764. The low prediction errors show a good model fit.

3. Code has been linked throughout, is attached, and available below.

Part 5: Data Summary and Implications

Summary

1. The ARIMA model predicts revenues to remain stable over the next 4 months. The model was selected based on interpretation of the results from the ACF, PACF, and AIC indicating the ARIMA(1,1,0) was the best model for the time series. The results were forecasted over

the same period of the test set using a 95% confidence interval. The confidence interval provides a range of outcomes and visualizes the inherent uncertainty in the forecast. The forecast length was chosen because it simplifies the model construction by preventing errors when comparing the test data to the forecasted data and it is long enough to satisfy the requirements of the research question. The model performed well with an MSE of 5.6327 and MAE of 1.8764.

2. This visualization shows the forecasted data overlayed with the actual test data to visualize performance. This demonstrates the model's ability to accurately predict future values.

3. The recommendation based on the model performance and outcome is to use the model to forecast short term revenue. This model should be updated regularly the capture any emerging trends.

Sources

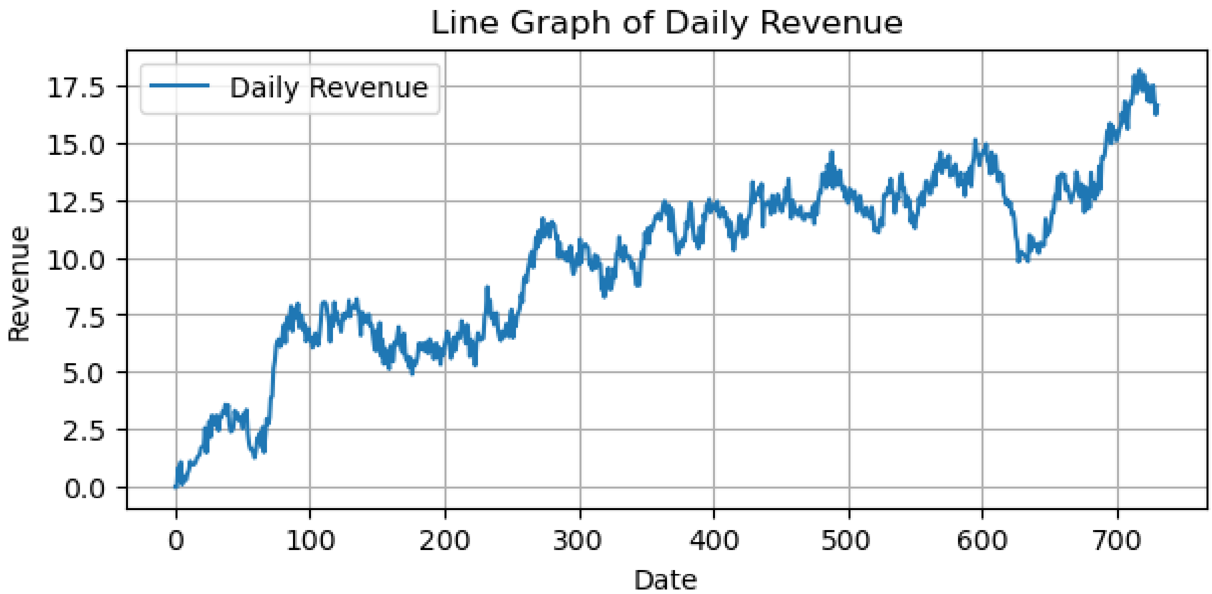
In [155…

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Load the uploaded CSV file
file_path = 'C:/Users/jhall/Desktop/D213/teleco_time_series .csv'
data = pd.read_csv(file_path)
```

Line Graph

In [157…

```
# Plotting the time series
plt.figure(figsize=(7, 3))
plt.plot(data.index, data['Revenue'], label='Daily Revenue')
plt.title('Line Graph of Daily Revenue')
plt.xlabel('Date')
plt.ylabel('Revenue')
plt.grid(True)
plt.legend()
plt.show()
```



Back

In [159…
```
# Display the first few rows of the dataset to understand its structure
data.head(), data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731 entries, 0 to 730
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   Day      731 non-null    int64  
 1   Revenue  731 non-null    float64
dtypes: float64(1), int64(1)
memory usage: 11.6 KB
```

Out[159]:
```
(   Day   Revenue
 0    1  0.000000
 1    2  0.000793
 2    3  0.825542
 3    4  0.320332
 4    5  1.082554,
 None)
```

Data Cleaning

In [161…
```
# Checking for null values
data.isnull().sum()
```

Out[161]:
```
Day        0
Revenue    0
dtype: int64
```

In [162…
```
# Checking for duplicated data
data.duplicated().sum()
```

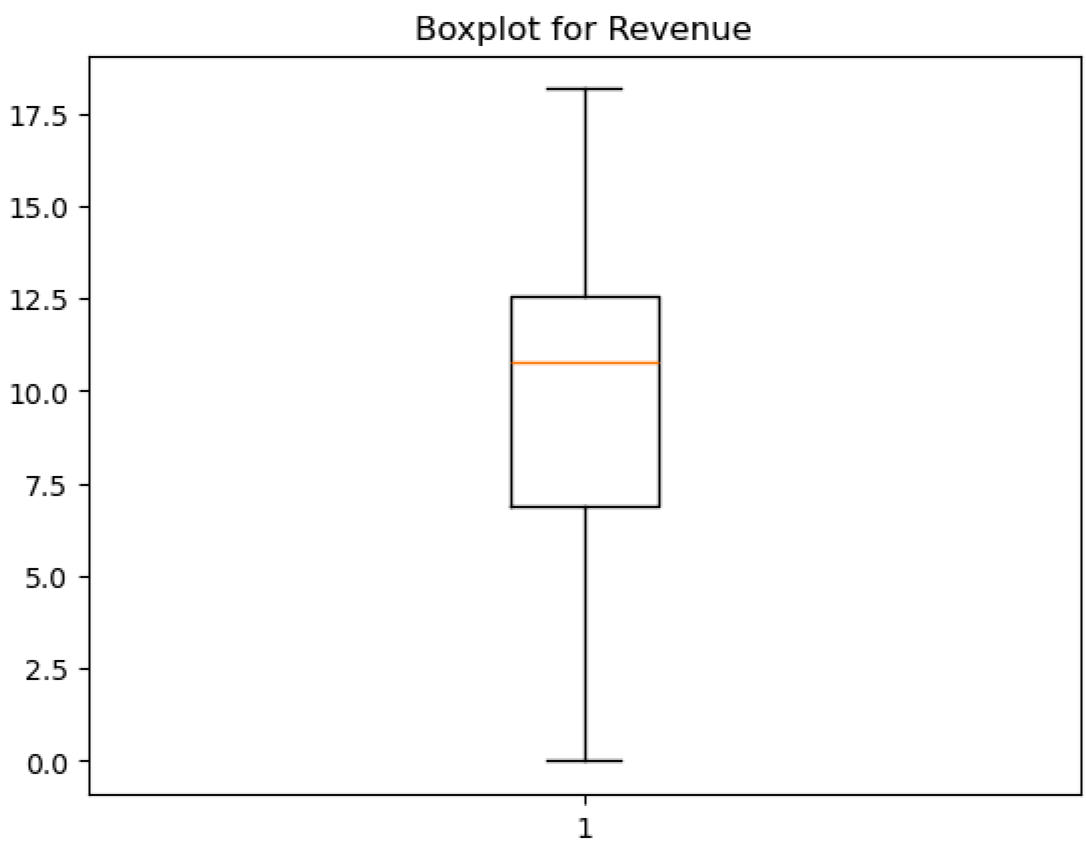
Out[162]:
```
0
```

In [163…
```
# Checking data type
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731 entries, 0 to 730
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   Day      731 non-null    int64  
 1   Revenue  731 non-null    float64
dtypes: float64(1), int64(1)
memory usage: 11.6 KB
```

In [164…

```
# Checking for outliers
plt.boxplot(data['Revenue'])
plt.title('Boxplot for Revenue')
plt.show()
```



Back

Differencing

In [167…

```
# Create a new DataFrame to store the differenced data
diff_data = data.copy()

# Difference the 'Revenue' column to make it stationary
diff_data['Revenue_Diff'] = diff_data['Revenue'].diff()

# Drop the first row with NaN value after differencing
diff_data = diff_data.dropna()

# Display the first few rows of the cleaned and differenced DataFrame
diff_data.head()
```

Out[167]:

	Day	Revenue	Revenue_Diff
1	2	0.000793	0.000793
2	3	0.825542	0.824749
3	4	0.320332	-0.505210
4	5	1.082554	0.762222
5	6	0.107654	-0.974900

ADF

Back

In [169...

```
from statsmodels.tsa.stattools import adfuller

# ADF
adf_result = adfuller(diff_data['Revenue_Diff'].dropna())
print(f"ADF Statistic: {adf_result[0]:.4f}")
print(f"p-value: {adf_result[2]:.4f}")
```

```
ADF Statistic: -44.8745
p-value: 0.0000
```

Time Setting

Back

In [171...

```
# Indexing the date values
data['Date'] = pd.date_range(start='2023-01-01', periods=len(data), freq='D')
data.set_index('Date', inplace=True)
```

Test Training Split

Back

In [173...

```
# Determine the split index (80% of the data)
split_index = int(len(data) * 0.8)

# Split the data into training and testing sets
train_df = data.iloc[:split_index]
test_df = data.iloc[split_index:]

# Display the number of rows in each set
print("Number of rows in train_df:", len(train_df))
print("Number of rows in test_df:", len(test_df))

# Display the first few rows of train_df and test_df
train_df.head(), test_df.head()
```

```
Number of rows in train_df: 584
Number of rows in test_df: 147
```

Out[173]:

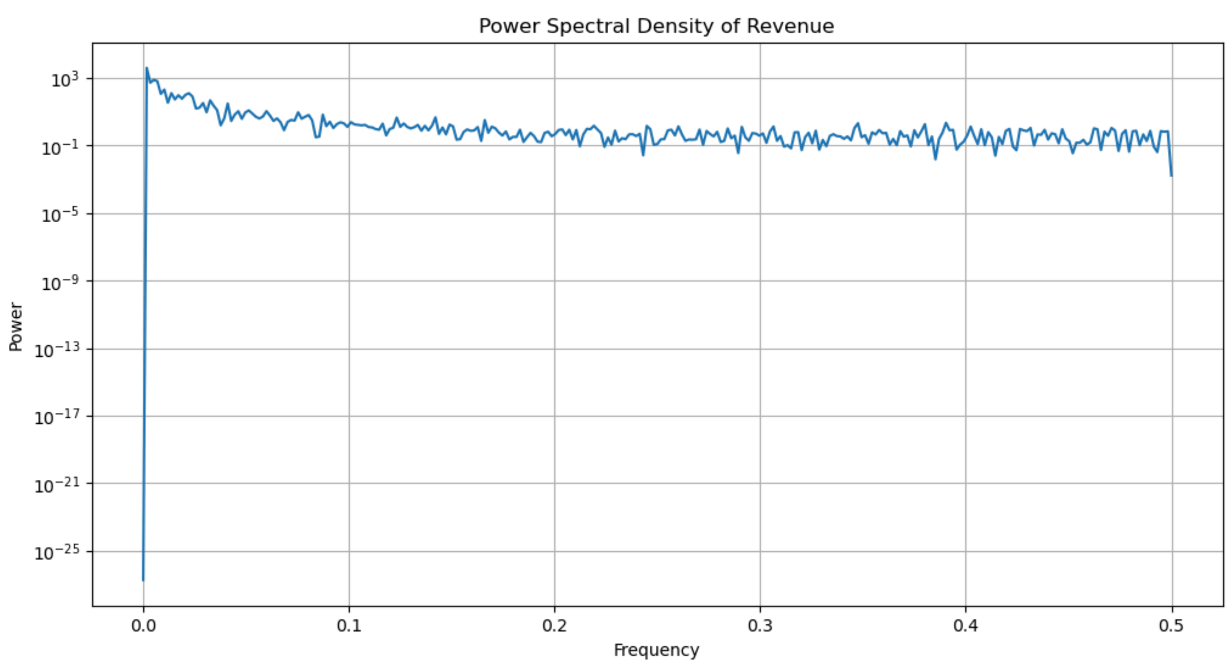
```
(            Day   Revenue
 Date
 2023-01-01    1  0.000000
 2023-01-02    2  0.000793
 2023-01-03    3  0.825542
 2023-01-04    4  0.320332
 2023-01-05    5  1.082554,
             Day    Revenue
 Date
 2024-08-07  585  13.684826
 2024-08-08  586  13.152903
 2024-08-09  587  13.310290
 2024-08-10  588  12.665601
 2024-08-11  589  13.660658)
```

Spectral Density

Back

In [175…

```python
import matplotlib.pyplot as plt
from scipy.signal import periodogram

# Perform spectral density analysis on the 'Revenue_Diff' column
frequencies, power_spectral_density = periodogram(train_df['Revenue'].dropna())

# Plot the power spectral density
plt.figure(figsize=(12, 6))
plt.semilogy(frequencies, power_spectral_density)
plt.title('Power Spectral Density of Revenue')
plt.xlabel('Frequency')
plt.ylabel('Power')
plt.grid(True)
plt.show()
```

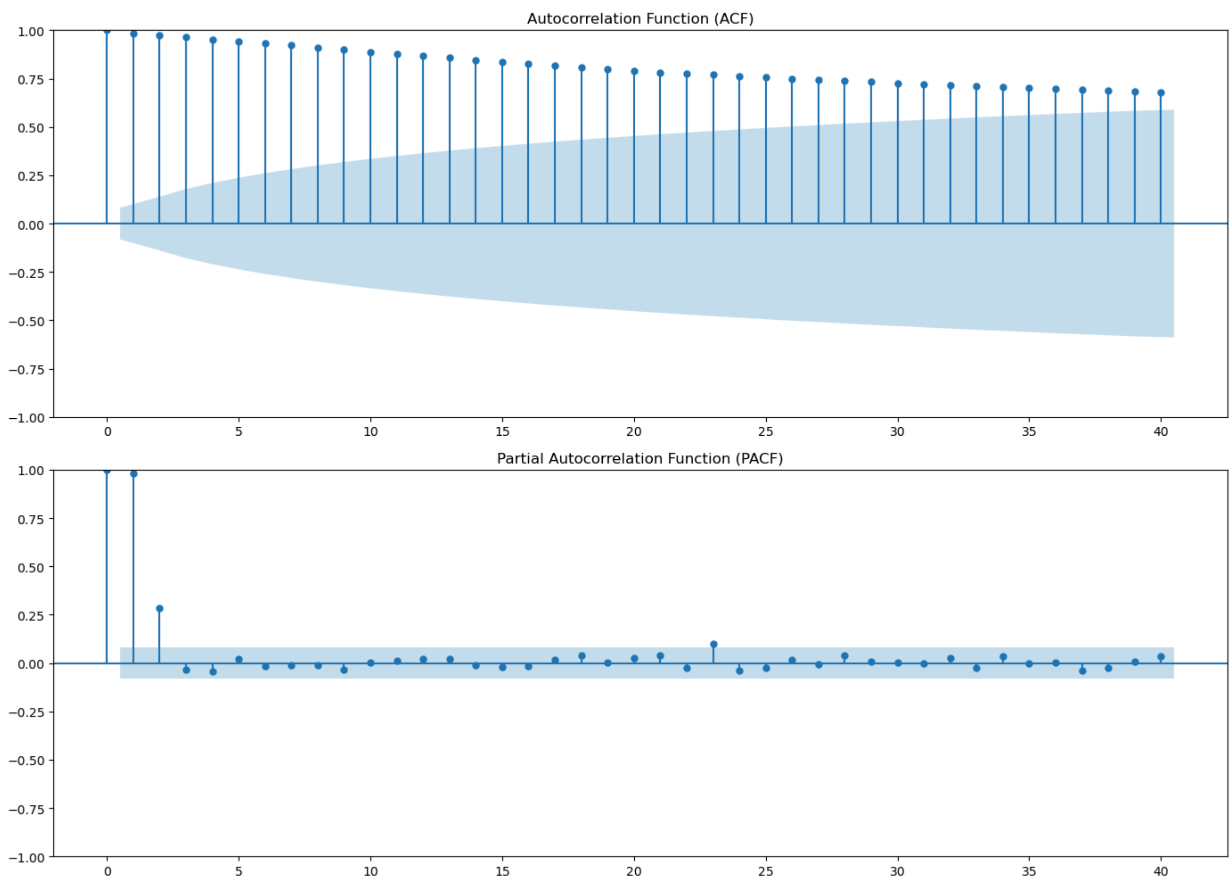


ACF/PACF

Back

In [177…

```
import statsmodels.api as sm
import matplotlib.pyplot as plt

# Perform ACF and PACF plots on the 'Revenue_Diff' column of train_df
fig, ax = plt.subplots(2, 1, figsize=(14, 10))

# ACF plot
sm.graphics.tsa.plot_acf(train_df['Revenue'].dropna(), lags=40, ax=ax[0])
ax[0].set_title('Autocorrelation Function (ACF)')

# PACF plot
sm.graphics.tsa.plot_pacf(train_df['Revenue'].dropna(), lags=40, ax=ax[1], method='ywm
ax[1].set_title('Partial Autocorrelation Function (PACF)')

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



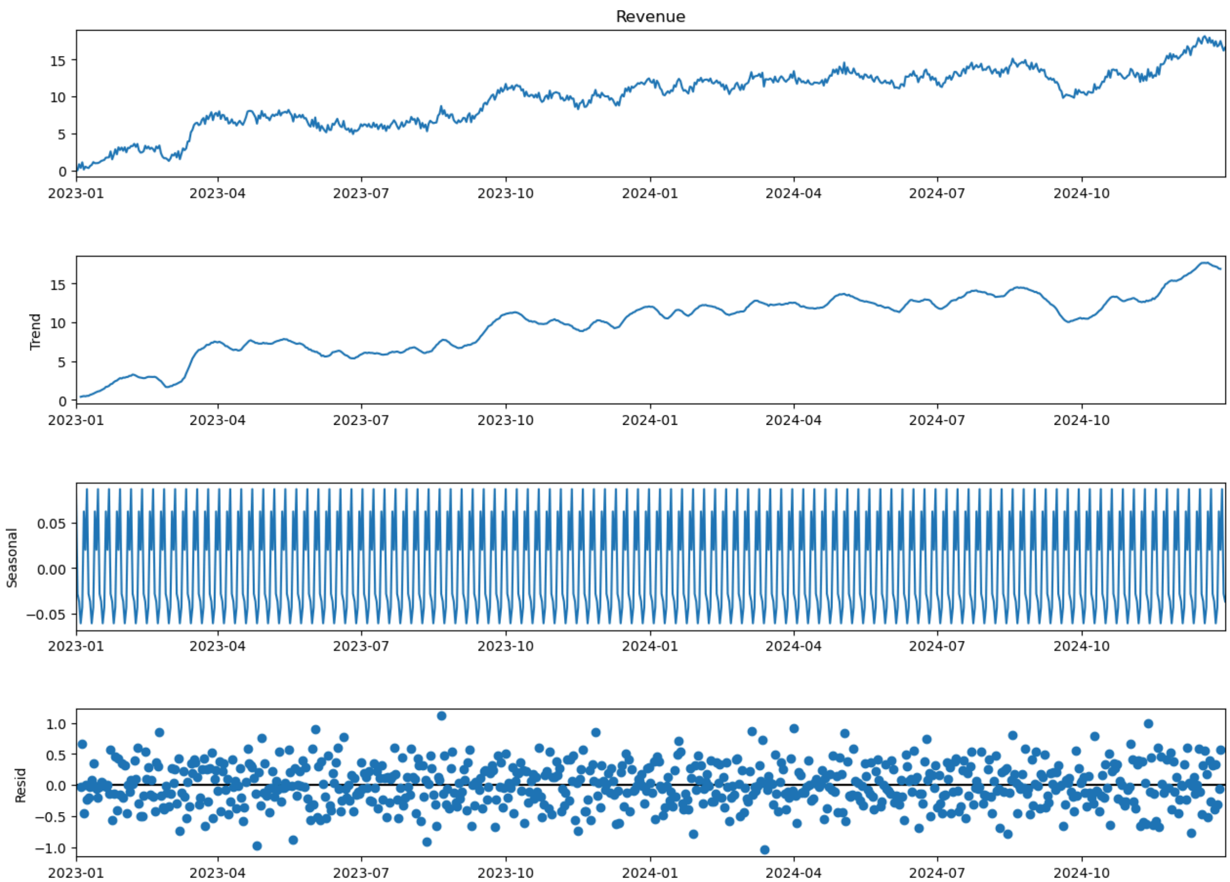
Decomposition

Back

In [179…

```
from statsmodels.tsa.seasonal import seasonal_decompose
```

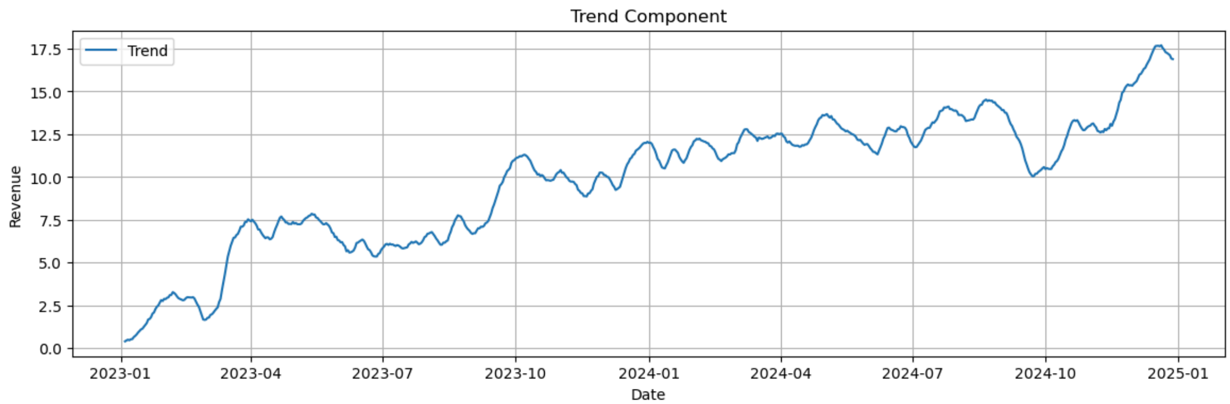
```python
# Ensure the data is set to a datetime index
if not pd.api.types.is_datetime64_any_dtype(data.index):
    data['Date'] = pd.date_range(start='2023-01-01', periods=len(data), freq='D')
    data.set_index('Date', inplace=True)

# Perform decomposition
decomposition = seasonal_decompose(data['Revenue'], model='additive')

# Plot the decomposition
fig = decomposition.plot()
fig.set_size_inches(14, 10)
plt.show()
```

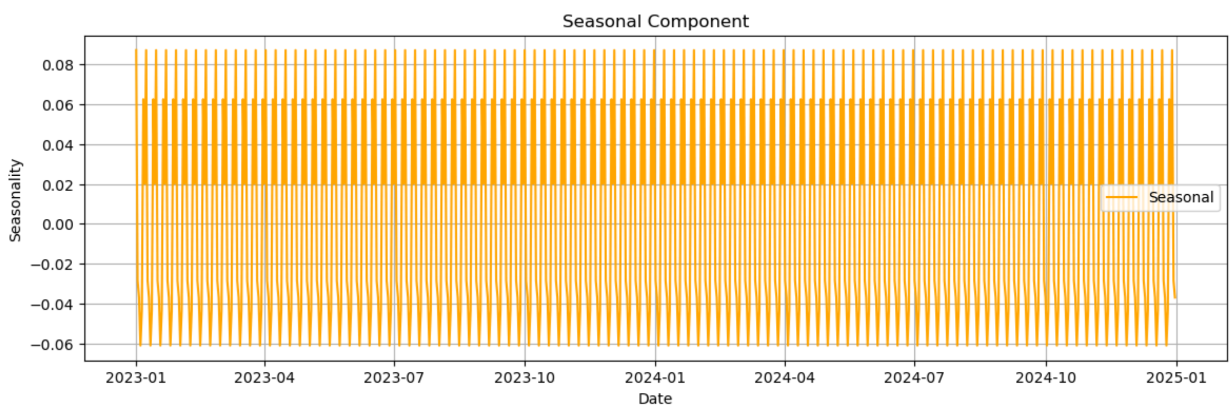


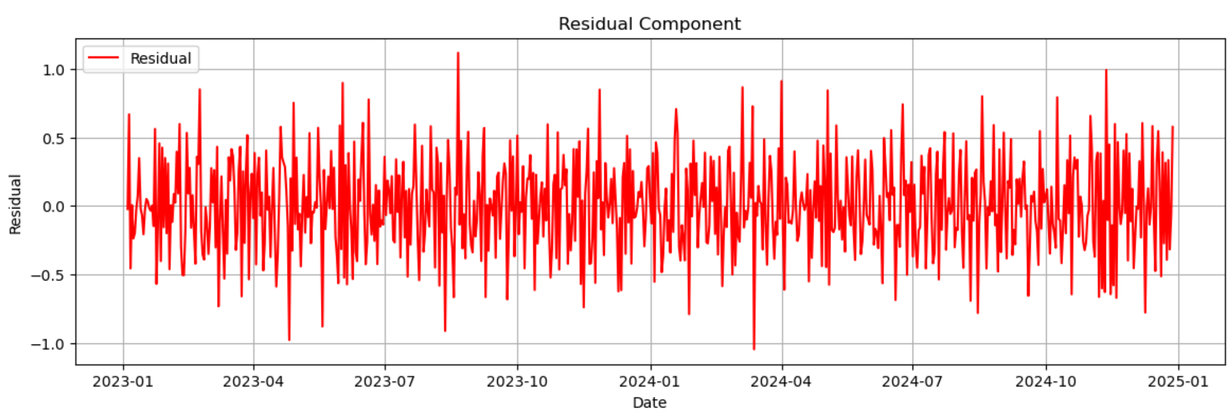
In [180…

```python
# Plotting each component separately
decomposition = seasonal_decompose(data['Revenue'], model='additive')

# Extract individual components
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

# Plot the trend component
plt.figure(figsize=(14, 4))
plt.plot(trend, label='Trend')
plt.title('Trend Component')
plt.xlabel('Date')
plt.ylabel('Revenue')
plt.legend()
plt.grid(True)
plt.show()

# Plot the seasonal component
```

```
plt.figure(figsize=(14, 4))
plt.plot(seasonal, label='Seasonal', color='orange')
plt.title('Seasonal Component')
plt.xlabel('Date')
plt.ylabel('Seasonality')
plt.legend()
plt.grid(True)
plt.show()

# Plot the residual component
plt.figure(figsize=(14, 4))
plt.plot(residual, label='Residual', color='red')
plt.title('Residual Component')
plt.xlabel('Date')
plt.ylabel('Residual')
plt.legend()
plt.grid(True)
plt.show()
```







AIC

Back

In [182…

```
import warnings
import itertools
from statsmodels.tsa.arima.model import ARIMA

# Suppress warnings for cleaner output
warnings.filterwarnings('ignore')

# Define the range of parameters for p, d, and q
p = d = q = range(0, 3)

# Generate all possible combinations of p, d, q
pdq_combinations = list(itertools.product(p, d, q))

# Track the best model based on the lowest AIC
best_aic = float('inf')
best_pdq = None
best_model = None

# Iterate over each combination of p, d, q
for pdq in pdq_combinations:
    try:
        # Fit the ARIMA model
        model = ARIMA(train_df['Revenue'], order=pdq).fit()
        current_aic = model.aic

        # Update the best model if the current AIC is lower
        if current_aic < best_aic:
            best_aic = current_aic
            best_pdq = pdq
            best_model = model
    except Exception as e:
        continue

# Display the best model order and its AIC
print(f"Best ARIMA order: {best_pdq} with AIC: {best_aic}")
```

```
Best ARIMA order: (1, 1, 0) with AIC: 774.0353132415657
```

In [217…

```
# Fit the ARIMA model using the best hyperparameters found from auto_arima
from statsmodels.tsa.arima.model import ARIMA

# Best hyperparameters from auto_arima
best_order = (1, 1, 0)

# Fit the ARIMA model using the training data
fitted_arima_model = ARIMA(train_df['Revenue'], order=best_order, trend='t').fit()

# Print the summary of the fitted ARIMA model
fitted_arima_model_summary = fitted_arima_model.summary()
fitted_arima_model_summary
```

Out[217]:

SARIMAX Results

Dep. Variable:	Revenue	**No. Observations:**	584
Model:	ARIMA(1, 1, 0)	**Log Likelihood**	-383.523
Date:	Wed, 20 Nov 2024	**AIC**	773.046
Time:	22:56:33	**BIC**	786.151
Sample:	01-01-2023	**HQIC**	778.154
	- 08-06-2024		
Covariance Type:	opg		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
x1	0.0230	0.013	1.727	0.084	-0.003	0.049
ar.L1	-0.4605	0.036	-12.663	0.000	-0.532	-0.389
sigma2	0.2181	0.014	16.020	0.000	0.191	0.245

Ljung-Box (L1) (Q):	0.00	**Jarque-Bera (JB):**	1.79
Prob(Q):	0.96	**Prob(JB):**	0.41
Heteroskedasticity (H):	0.97	**Skew:**	-0.07
Prob(H) (two-sided):	0.85	**Kurtosis:**	2.77

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [219…

```
# Fit the SARIMAX model using the best order from the AIC result
sarimax_model = SARIMAX(train_df['Revenue'], order=(1, 1, 0,), trend='t').fit()

# Print the summary of the SARIMAX model
sarimax_model_summary = sarimax_model.summary()
sarimax_model_summary
```

Out[219]:

SARIMAX Results

Dep. Variable:	Revenue	**No. Observations:**	584
Model:	SARIMAX(1, 1, 0)	**Log Likelihood**	-384.508
Date:	Wed, 20 Nov 2024	**AIC**	775.016
Time:	22:56:34	**BIC**	788.120
Sample:	01-01-2023	**HQIC**	780.124
	- 08-06-2024		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
drift	5.816e-05	5.85e-05	0.994	0.320	-5.65e-05	0.000
ar.L1	-0.4595	0.036	-12.632	0.000	-0.531	-0.388
sigma2	0.2193	0.014	15.955	0.000	0.192	0.246

Ljung-Box (L1) (Q):	0.00	**Jarque-Bera (JB):**	1.73
Prob(Q):	0.98	**Prob(JB):**	0.42
Heteroskedasticity (H):	0.97	**Skew:**	-0.07
Prob(H) (two-sided):	0.81	**Kurtosis:**	2.77

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

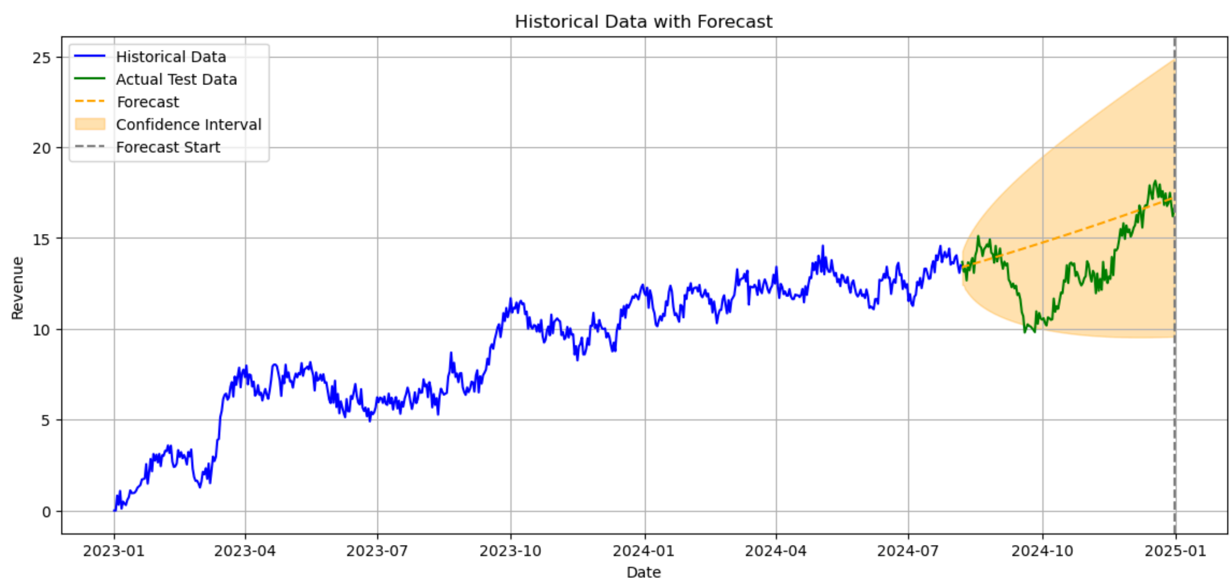
Model Visualization

Back

In [222...

```
# Forecast over 4 months out. Future Steps = test data to avoide errors
future_steps = 147  # 4 months of daily data

# Generate the forecast
forecast = sarimax_model.get_forecast(steps=future_steps)

# Extract forecasted mean and confidence intervals
forecast_mean = forecast.predicted_mean
forecast_ci = forecast.conf_int()

# Create an index for the forecast period
last_date = train_df.index[-1]  # Last date in the training data
forecast_index = pd.date_range(start=last_date + pd.Timedelta(days=1), periods=future_

# Combine historical data with forecast
forecast_df = pd.DataFrame({'Forecast': forecast_mean}, index=forecast_index)

# Plot historical data and forecast
```

```python
plt.figure(figsize=(14, 6))
# Plot historical data
plt.plot(train_df.index, train_df['Revenue'], label='Historical Data', color='blue')
plt.plot(test_df.index, test_df['Revenue'], label='Actual Test Data', color='green')
# Plot forecast
plt.plot(forecast_df.index, forecast_df['Forecast'], label='Forecast', linestyle='--',
# Plot confidence intervals
plt.fill_between(forecast_index,
                 forecast_ci.iloc[:, 0],
                 forecast_ci.iloc[:, 1],
                 color='orange', alpha=0.3, label='Confidence Interval')

# Add labels and title
plt.title('Historical Data with Forecast')
plt.xlabel('Date')
plt.ylabel('Revenue')
plt.axvline(x=test_df.index[-1], color='gray', linestyle='--', label='Forecast Start')
plt.legend()
plt.grid(True)
plt.show()

from sklearn.metrics import mean_squared_error, mean_absolute_error
mse = mean_squared_error(test_df['Revenue'], forecast_mean[:len(test_df)])
mae = mean_absolute_error(test_df['Revenue'], forecast_mean[:len(test_df)])
print(f"Mean Squared Error: {mse}")
print(f"Mean Absolute Error: {mae}")
```



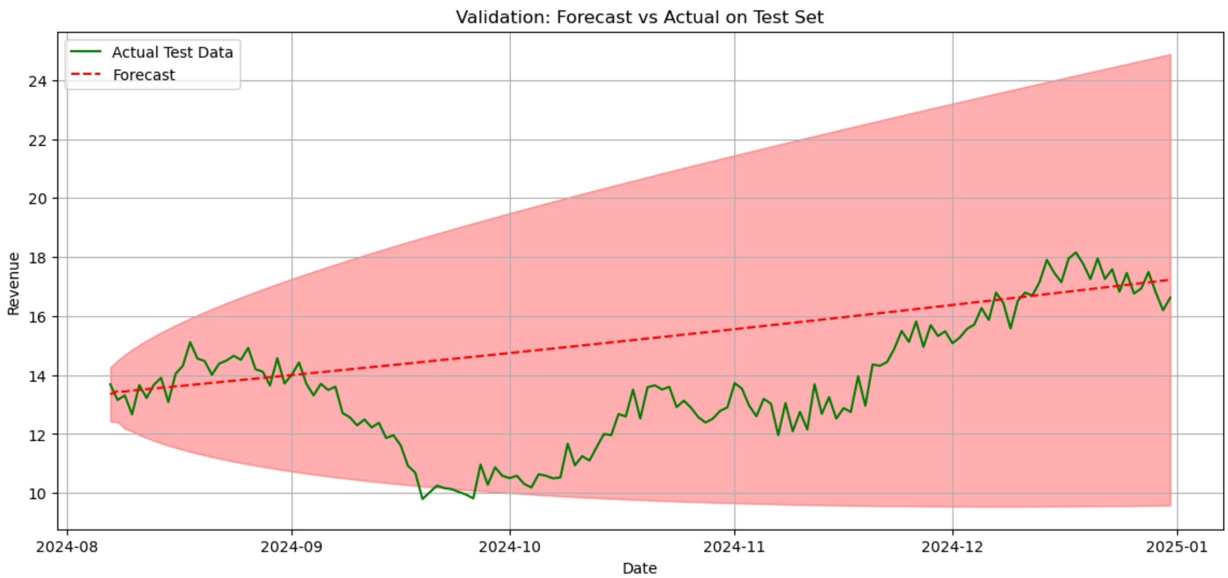

```
Mean Squared Error: 5.632726461622653
Mean Absolute Error: 1.8764163465534704
```

In [224…

```python
# Validate the SARIMAX model by comparing forecast with the actual test set

# Calculate error metrics for the comparison
mse_test = mean_squared_error(test_df['Revenue'], forecast_mean)
mae_test = mean_absolute_error(test_df['Revenue'], forecast_mean)

# Display the error metrics
print(f"Validation on Test Set:")
print(f"Mean Squared Error (MSE): {mse_test:.4f}")
print(f"Mean Absolute Error (MAE): {mae_test:.4f}")

# Plot actual vs. forecast for visual validation
```

```python
plt.figure(figsize=(14, 6))
plt.plot(test_df.index, test_df['Revenue'], label='Actual Test Data', color='green')
plt.plot(test_df.index, forecast_mean, label='Forecast', color='red', linestyle='--')
plt.fill_between(test_df.index, forecast_ci.iloc[:, 0], forecast_ci.iloc[:, 1], color=

plt.title('Validation: Forecast vs Actual on Test Set')
plt.xlabel('Date')
plt.ylabel('Revenue')
plt.legend()
plt.grid(True)
plt.show()
```

```
Validation on Test Set:
Mean Squared Error (MSE): 5.6327
Mean Absolute Error (MAE): 1.8764
```



In [226…

```python
import joblib

# Save the SARIMAX model to a file
SARIMAX_D213 = 'C:/Users/jhall/Desktop/D213/SARIMAXD213.pkl'
joblib.dump(sarimax_model, SARIMAX_D213)

print(f"Model saved successfully as {SARIMAX_D213}")
```

```
Model saved successfully as C:/Users/jhall/Desktop/D213/SARIMAXD213.pkl
```

Clean Data

Back

In [229…

```python
# Extract and save the cleaned dataset
cleaned_data_path = 'C:/Users/jhall/Desktop/D213/clean_teleco_time_series .csv'
diff_data.to_csv(cleaned_data_path, index=True)
print(f"Cleaned dataset saved successfully at {cleaned_data_path}")
```

```
Cleaned dataset saved successfully at C:/Users/jhall/Desktop/D213/clean_teleco_time_s
eries .csv
```

Sources

Back

F. This project is done in a Jupyter Notebook. Attached are PDF and ipynb versions.

G Code Sources

McKinney, W. (2010). Data structures for statistical computing in Python. In Proceedings of the 9th Python in Science Conference (pp. 51-56). SciPy. doi:10.25080/Majora-92bf1922-00a.

Seabold, S., & Perktold, J. (2010). Statsmodels: Econometric and statistical modeling with Python. In Proceedings of the 9th Python in Science Conference (pp. 57-61). SciPy. doi:10.25080/Majora-92bf1922-011.

Virtanen, P., et al. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. Nature Methods, 17(3), 261-272. doi:10.1038/s41592-019-0686-2

H. Web Sources Scaler. (n.d.). Stationary data in machine learning. Scaler Topics. Retrieved from https://www.scaler.com/topics/machine-learning/stationary-data-in-machine-learning/

Scribbr. (n.d.). Akaike Information Criterion. Scribbr. Retrieved from https://www.scribbr.com/statistics/akaike-information-criterion/

Nau, R. (n.d.). ARIMA models. Duke University. Retrieved from https://people.duke.edu/~rnau/411arim.htm

Towards Data Science. (2019, September 23). Time Series Forecasting — ARIMA models. Towards Data Science. Retrieved from https://towardsdatascience.com/time-series-forecasting-arima-models-7f221e9eee06

Real Statistics. (n.d.). ARIMA differencing. Real Statistics. Retrieved from https://real-statistics.com/time-series-analysis/arima-processes/arima-differencing/#:~
=An%20autoregressive%20integrated%20moving%20average,process%20with%20first%2Dorder%2