CONTENTS

PART I: RESEARCH QUESTION

RESEARCH QUESTION

How can telecommunications companies predict daily for the next 180 days to manage customer churn better?

OBJECTIVES

My goals for this analysis are to identify revenue trends and patterns and conduct spectral density analysis. The first goal is to analyze the historical daily revenue data to identify trends, seasonal patterns, and recurring cycles. This helps us understand how revenue has changed over time. The second goal is to analyze the spectral density of the revenue data to identify cyclical patterns in revenue.

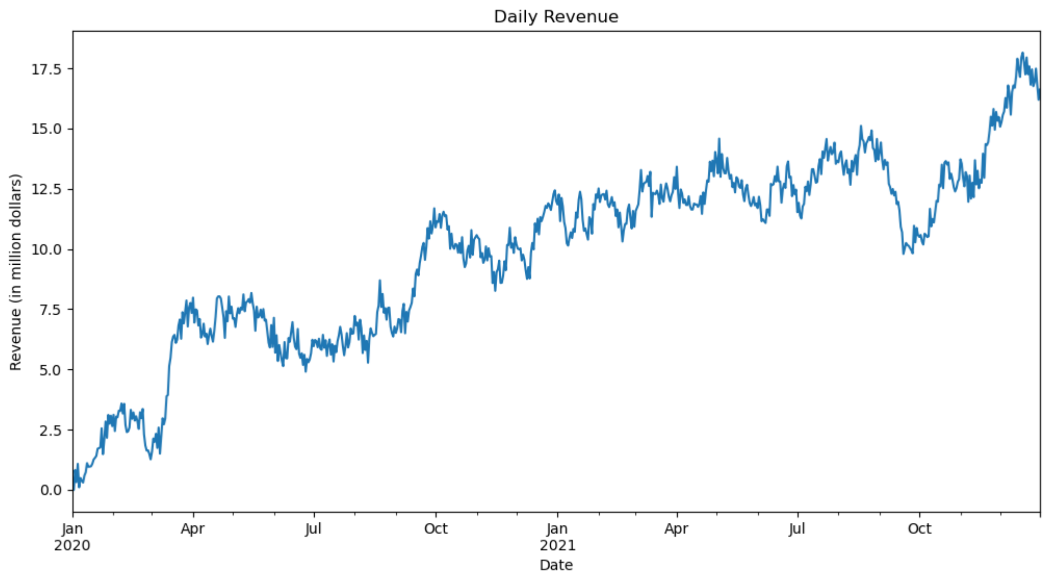
PART II: METHOD JUSTIFICATION

SUMMARY OF ASSUMPTIONS

Time series models require the data to be stationary to make accurate and meaningful predictions. Non-stationary data can lead to unreliable forecasts. A time series is stationary if its statistical properties, such as mean, variance, and autocorrelation, are constant over time.
Autocorrelation in time series analysis assumes that current values are correlated with past values. Significant autocorrelation indicates that future values depend on previous observations. This is crucial for selecting appropriate model like ARIMA.

PART III: DATA PREPARATION

LINE GRAPH VISUALIZATION



TIME STEP FORMATTING

The graph does not exhibit any visible gaps in measurement. Each data point corresponds to a specific day, with no missing entries. This continuity suggests that the revenue data is consistently recorded. The line graph above spans approximately 24 months. This extended period allows us to observe revenue trends over a substantial period. The graph exhibits fluctuations in daily revenue. Peaks indicate periods of higher revenue, while troughs represent lower revenue.

STATIONARITY

A visual inspection of the previous graphs shows a non-stationary time series, as seen in the trends and seasonality that change over time.

```
# Perform ADF test
adf_test = adfuller(df['Revenue'])
print('ADF Test Statistic:', adf_test[0])
print('p-value:', adf_test[1])

# Perform the KPSS test
kpss_test = kpss(df['Revenue'], regression='c')
print('\nKPSS Test Statistic:', kpss_test[0])
print('p-value:', kpss_test[1])

# Check if the p-value is less than 0.05
if adf_test[1] < 0.05 and kpss_test[1] >= 0.05:
    print("\nThe time series is stationary.")
else:
    print("\nThe time series is not stationary.")
```

```
ADF Test Statistic: -1.9246121573101842
p-value: 0.3205728150793961

KPSS Test Statistic: 3.5607139300692454
p-value: 0.01

The time series is not stationary.
```

I used the Augmented Dickey-Fuller (ADF) Test and the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test for further statistical tests. ADF tests the null hypothesis that a unit root is present in the time series. KPSS tests the null hypothesis to find if the time series is stationary. In this analysis, the initial df is not stationary.

```
# Create a new DataFrame with the stationary time series
df_stationary = df.copy()

# Difference the time series to make it stationary
df_stationary['Revenue'] = df_stationary['Revenue'].diff().dropna()

# Drop missing values from df_stationary
df_stationary.dropna(inplace=True)

# Plot the differenced series
df_stationary['Revenue'].plot(figsize=(12, 6), title='Differenced Daily Revenue', xlabel='Day', ylabel='Differenced Revenue (in million dollars)')
plt.savefig("t1_diff-revenue.png")
plt.show()
```

Differencing is used to make the initial df stationary. This process subtracts the previous observation from the current observation. Using diff() transforms the Revenue column into a series of differences between consecutive days, which can help stabilize the mean of the time series.

```
# Perform ADF test on differenced series
adf_test_diff = adfuller(df_stationary['Revenue'])
print('ADF Test Statistic (Differenced):', adf_test_diff[0])
print('p-value:', adf_test_diff[1])

# Perform KPSS test on differenced series
kpss_test_diff = kpss(df_stationary['Revenue'], regression='c')
print('\nKPSS Test Statistic (Differenced):', kpss_test_diff[0])
print('p-value:', kpss_test_diff[1])

# Check if the p-value is less than 0.05
if adf_test_diff[1] < 0.05 and kpss_test_diff[1] >= 0.05:
    print("\nThe differenced time series is stationary.")
else:
    print("\nThe differenced time series is not stationary.")
```

```
ADF Test Statistic (Differenced): -44.874527193876
p-value: 0.0

KPSS Test Statistic (Differenced): 0.06678789990071667
p-value: 0.1

The differenced time series is stationary.
```

Then, reevaluate the differenced series with the ADF and KPSS test. The result says the differenced time series is stationary.

STEPS TO PREPARE THE DATA

1. Initialize all libraries and packages to be used.

```
# Data Handling and Manipulation
import csv
import pandas as pd
import numpy as np

# Visualizations
import matplotlib.pyplot as plt

# Time Series Analysis
import pmdarima as pm
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller, kpss
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from pandas.plotting import autocorrelation_plot

# Modeling and Forecasting
import statsmodels.api as sm
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.arima.model import ARIMAResults

# Performance metrics
from sklearn.metrics import mean_absolute_error, mean_squared_error

# General Utilities
from platform import python_version
import os
from datetime import datetime
from scipy import signal
import warnings
import joblib

# Set warning filters
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.filterwarnings('ignore', category=RuntimeWarning)
from statsmodels.tools.sm_exceptions import InterpolationWarning
warnings.filterwarnings("ignore", category=InterpolationWarning)

# Print Python version (for informational purposes)
print("Python version used for this analysis is: ", python_version())
```

```
Python version used for this analysis is:  3.11.7
```

2. Load the dataset into Pandas DataFrame.

```
# Specifiy CSV file path
file_path = r'C:\Users\kolgi\OneDrive - Western Governors University\D213\churn_task1\teleco_time_series.csv'
# Open the CSV file and read it using DictReader
with open(file_path, 'r') as csvfile:
    csvreader = csv.DictReader(csvfile)

# Read the CSV file into a pandas DataFrame then open head
df = pd.read_csv(file_path)
```

3. View the first five rows of the data frame.
   df.head()

4. Generate descriptive statistics such as the count, mean, standard deviation, minimum, 25th percentile, median, 75th percentile, and maximum values.
   df.describe()
5. Convert Day to datetime formatting.

```python
# Convert 'Day' to datetime format assuming 'Day' represents days since a start date
# Assume the start date is '2020-01-01'
start_date = '2020-01-01'
df['Day'] = pd.to_datetime(start_date) + pd.to_timedelta(df['Day'] - 1, unit='D')

# Set 'Day' as the index
df.set_index('Day', inplace=True)
```

6. Plot the Revenue data.

```python
# Plot the entire revenue data
df['Revenue'].plot(figsize=(12, 6), title='Daily Revenue', xlabel='Date', ylabel='Revenue (in million dollars)')
plt.savefig("t1_revenue.png")
plt.grid(True)
plt.show()
```

7. Check for missing values using .isnull().sum().

```python
# Check for missing values
missing_values = df.isnull().sum()

# Print the count of missing values
print(missing_values)
```

8. Evaluate stationarity of original dataset.

```python
# Perform ADF test
adf_test = adfuller(df['Revenue'])
print('ADF Test Statistic:', adf_test[0])
print('p-value:', adf_test[1])

# Perform the KPSS test
kpss_test = kpss(df['Revenue'], regression='c')
print('\nKPSS Test Statistic:', kpss_test[0])
print('p-value:', kpss_test[1])

# Check if the p-value is less than 0.05
if adf_test[1] < 0.05 and kpss_test[1] >= 0.05:
    print("\nThe time series is stationary.")
else:
    print("\nThe time series is not stationary.")
```

9. Difference the time series to make it stationary. Then, drop missing values from differeced time series. Plot the differenced series.

```python
# Create a new DataFrame with the stationary time series
df_stationary = df.copy()

# Difference the time series to make it stationary
df_stationary['Revenue'] = df_stationary['Revenue'].diff().dropna()

# Drop missing values from df_stationary
df_stationary.dropna(inplace=True)

# Plot the differenced series
df_stationary['Revenue'].plot(figsize=(12, 6), title='Differenced Daily Revenue', xlabel='Day', ylabel='Differenced Revenue (in million dollars)')
plt.savefig("t1_diff-revenue.png")
plt.show()
```

10. Evaluate stationarity of differenced series.

```python
# Perform ADF test on differenced series
adf_test_diff = adfuller(df_stationary['Revenue'])
print('ADF Test Statistic (Differenced):', adf_test_diff[0])
print('p-value:', adf_test_diff[1])

# Perform KPSS test on differenced series
kpss_test_diff = kpss(df_stationary['Revenue'], regression='c')
print('\nKPSS Test Statistic (Differenced):', kpss_test_diff[0])
print('p-value:', kpss_test_diff[1])

# Check if the p-value is less than 0.05
if adf_test_diff[1] < 0.05 and kpss_test_diff[1] >= 0.05:
    print("\nThe differenced time series is stationary.")
else:
    print("\nThe differenced time series is not stationary.")
```

11. Split the dataset with rows 1-584 as train and 585-731 as test.

```python
# Split the dataset
train = df_stationary.iloc[:584]
test = df_stationary.iloc[584:]

# Print their shapes
print("Dataset shape:", df.shape)
print("Train shape:", train.shape)
print("Test shape:", test.shape)
```

12. Save the training and testing into a .csv file.

```python
# Save training dataframe to CSV
train.to_csv('task1_train.csv')

# Save testing dataframe to CSV
test.to_csv('task1_test.csv')
```

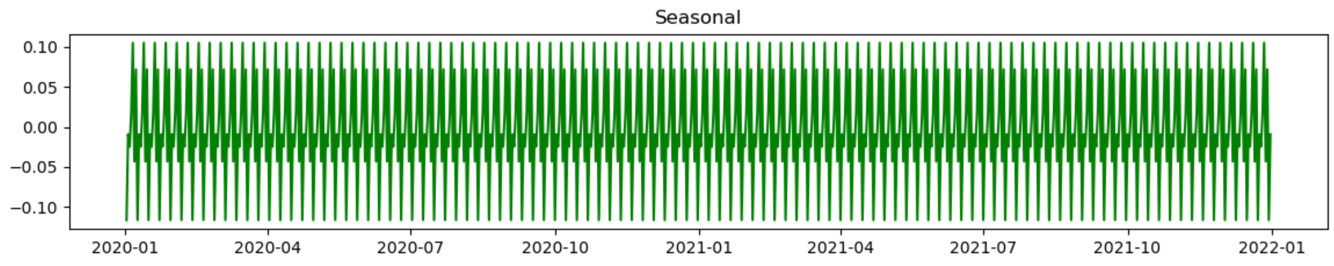
PREPARED DATA SET

Training data attached as *task1_train.csv*. Testing data attached as *task1_test.csv*.

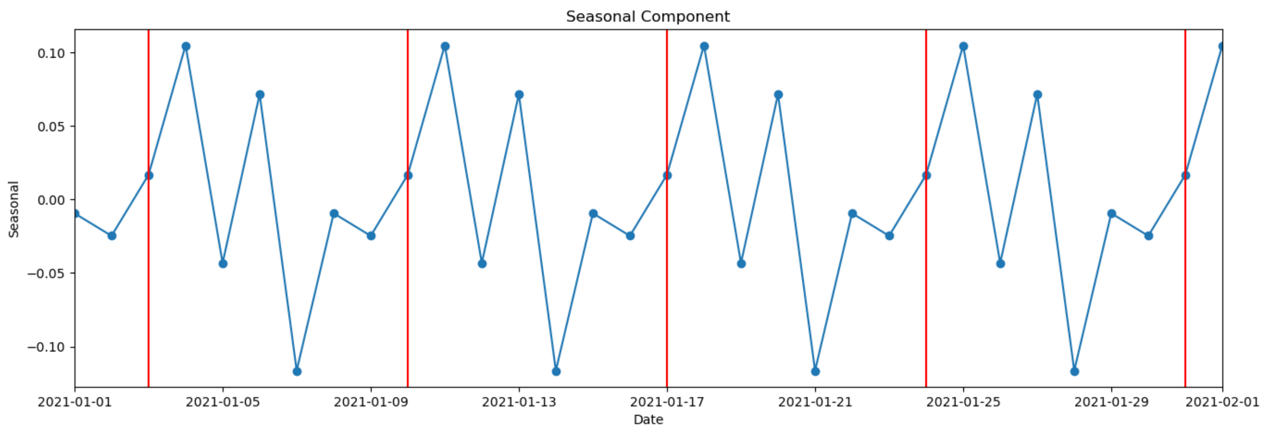
PART IV: MODEL IDENTIFICATION & ANALYSIS

REPORT FINDINGS & VISUALIZATIONS

SEASONALITY

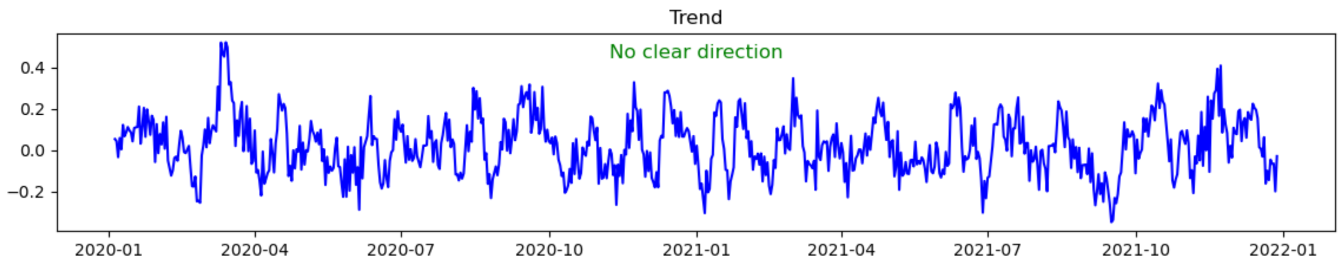


The seasonal component in the graph above represents fluctuations over time that exhibit a repeating pattern. The data points consistently oscillate above and below the zero line. The pattern repeats at fixed intervals which suggest a seasonal effect influencing the data.



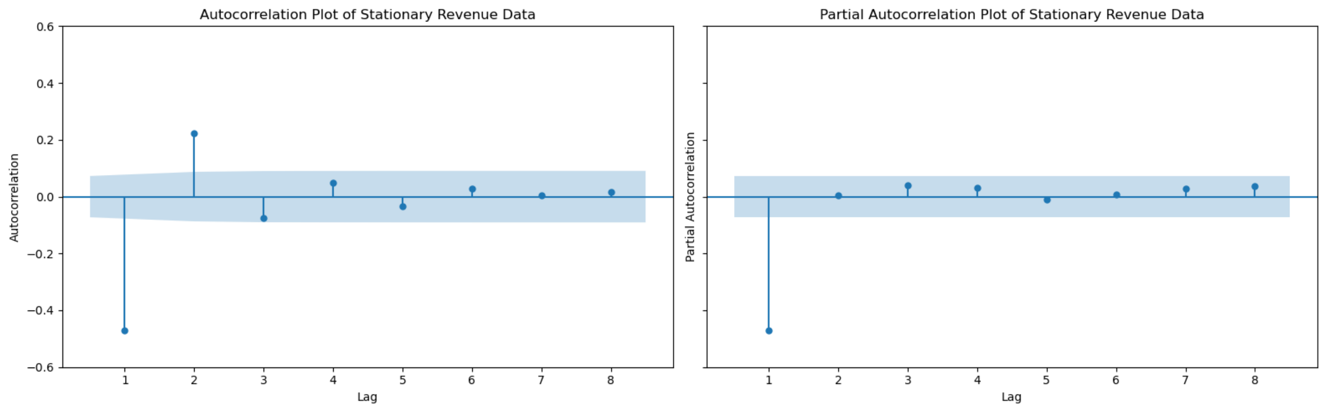
The seasonal component above in this graph exhibits a repeating pattern over a period of approximately one month. The data consistently oscillates above and below the zero line. The peaks signify positive deviations, while troughs represent negative deviations from the baseline.
The red vertical line is drawn at specific points to indicate Mondays. This regular alignment suggests a weakly seasonality effect.

TRENDS



There is no consistent upward or downward trend. This trend graph lacks a clear long-term movement. This variability suggests that there are no visible trends in this data.

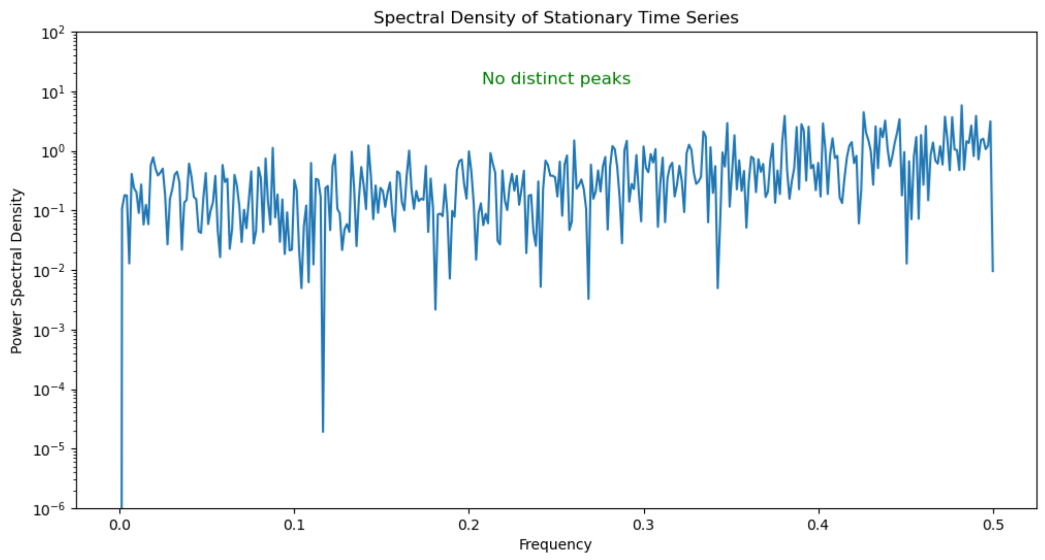
AUTOCORRELATION FUNCTION



The Autocorrelation Function (ACF) measures the correlation between a time series and its lagged values. Significant spikes at specific lags indicate potential moving average (MA) terms. In the ACF plot, lag 1 shows a positive correlation. The other lags fall within the blue-shaded area, which is deemed insignificant.

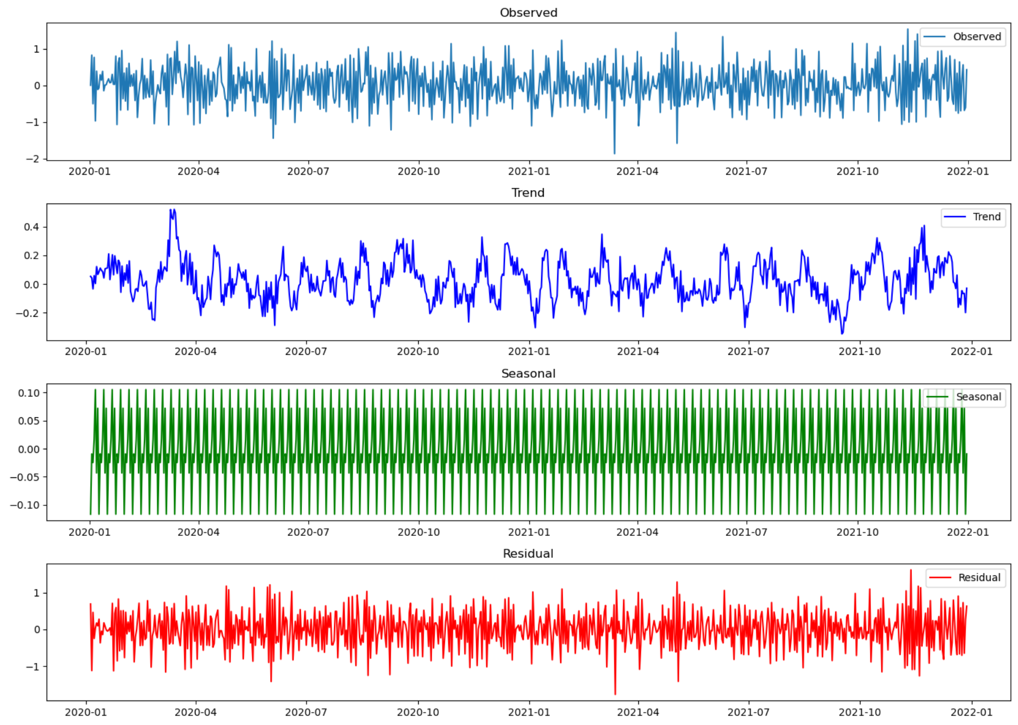
The Partial Autocorrelation Function (PACF) measures the correlation between a time series and its lagged values after removing the effects of shorter lags. Significant spikes at specific lags indicate potential auto-regressive (AR) terms. In the PACF plot, lag 1 has a significant negative spike. The other lags are close to zero and within the blue shade confidence interval.
Since there is only one significant spike in both ACF and PACF plot at lag 1, consider ARIMA(1,0,0) model.

SPECTRAL DENSITY



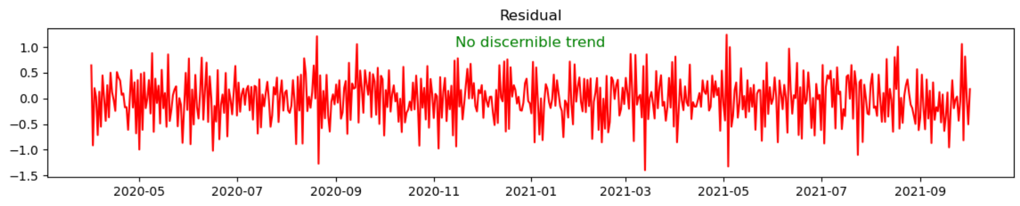
There are no distinct peaks with clear periodic components. Instead, it exhibits a consistent Variance across the frequency spectrum. Furthermore, the stationary nature and flat spectrum indicate that the data behaves randomly without specific frequency patterns.

DECOMPOSED TIME SERIES



The graphs display four components of a time series dataset. The blue line shows the observed data with fluctuations over time. The trend is the dark blue line, which represents a smoothed version of the data that captures the overall trend. The green line highlights the seasonal component. Lastly, the residual component is the red line that shows the remaining noise after accounting for trend and seasonality.

CONFIRMATION OF THE LACK OF TRENDS IN THE RESIDUALS OF THE TIME SERIES



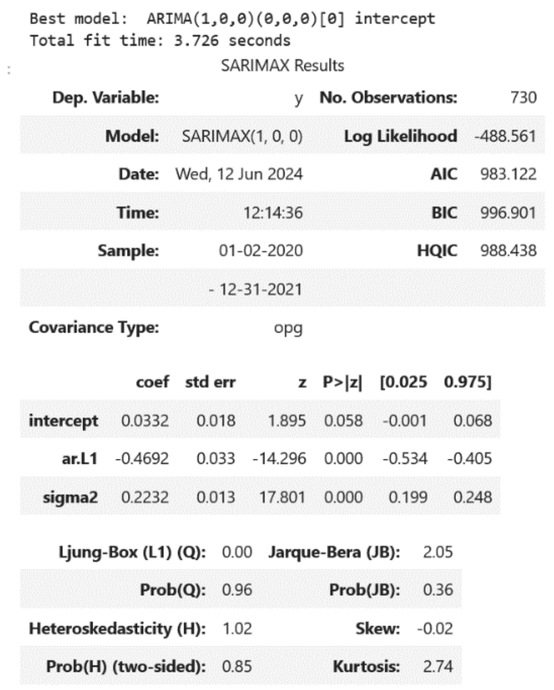
The residuals appear to be randomly scattered above and below a horizontal line at zero. This suggests that there is no discernable trend in how residuals deviate over time. This is a positive sign, as it suggests that the model captures most of the systemic information in the data.

ARIMA MODEL

INITIAL MODEL SELECTION WITH AUTO_ARIMA

```
# Use auto_arima to find the best model
auto_model = pm.auto_arima(df_stationary,
                           seasonal=False,
                           trace=True,
                           error_action='ignore',
                           suppress_warnings=True)
auto_model.summary()
```

I used auto_arima from the pmdarima library to automatically select the best ARIMA model for the stationary revenue data. Since the data was already stationary, the auto_arima function was configured with seasonal=False to focus on non-seasonal models.



```
Best model:  ARIMA(1,0,0)(0,0,0)[0] intercept
Total fit time: 3.726 seconds
```

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	730
Model:	SARIMAX(1, 0, 0)	**Log Likelihood**	-488.561
Date:	Wed, 12 Jun 2024	**AIC**	983.122
Time:	12:14:36	**BIC**	996.901
Sample:	01-02-2020	**HQIC**	988.438
	- 12-31-2021		
Covariance Type:	opg		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
intercept	0.0332	0.018	1.895	0.058	-0.001	0.068
ar.L1	-0.4692	0.033	-14.296	0.000	-0.534	-0.405
sigma2	0.2232	0.013	17.801	0.000	0.199	0.248

Ljung-Box (L1) (Q):	0.00	**Jarque-Bera (JB):**	2.05
Prob(Q):	0.96	**Prob(JB):**	0.36
Heteroskedasticity (H):	1.02	**Skew:**	-0.02
Prob(H) (two-sided):	0.85	**Kurtosis:**	2.74

The best model determined by auto_arima is ARIMA(1,0,0)(0,0,0)[0]. This indicates that the best model is an ARIMA model with one autoregressive term (AR), no differencing, and no moving average (MA) ter. There is no seasonal component as indicated by (0,0,0)[0], and the model includes an intercept. The Akaike Information Criterion, a measure of the relative quality of the model, is 983.122. The lower values indicate a better fit. A significant diagnostic test to note is the Ljung-Box (L1) (Q): 0.00, Prob(Q): 0.96**:** The Ljung-Box test for autocorrelation at lag 1 has a very low Q-value with a high p-value, indicating no significant autocorrelation in the residuals.

The residual diagnostics indicate that the model residuals do not exhibit significant autocorrelation, are normally distributed, and have constant variance.

FITTING ARIMA MODEL TO TRAINING DATA

```
# Fit model into train
from statsmodels.tsa.arima.model import ARIMA

model = ARIMA(train, order=(1, 0, 0), freq='D')
results = model.fit()
results.summary()
```

I used the best model to manually specify the ARIMA model to fit the training data.

SARIMAX Results

Dep. Variable:	Revenue	**No. Observations:**	584
Model:	ARIMA(1, 0, 0)	**Log Likelihood**	-383.946
Date:	Wed, 12 Jun 2024	**AIC**	773.893
Time:	12:14:36	**BIC**	787.002
Sample:	01-02-2020	**HQIC**	779.002
	- 08-07-2021		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.0234	0.013	1.758	0.079	-0.003	0.049
ar.L1	-0.4597	0.036	-12.654	0.000	-0.531	-0.388
sigma2	0.2180	0.014	16.034	0.000	0.191	0.245

Ljung-Box (L1) (Q):	0.00	**Jarque-Bera (JB):**	1.84
Prob(Q):	0.96	**Prob(JB):**	0.40
Heteroskedasticity (H):	0.97	**Skew:**	-0.08
Prob(H) (two-sided):	0.83	**Kurtosis:**	2.77

The AIC for this model is 773.893. The lower values indicate a better fit. The Ljung-Box (L1) (Q): 0.00, Prob(Q): 0.96 means the Ljung-Box test for autocorrelation at lag 1 has a very low Q-value with a high p-value, indicating no significant autocorrelation in the residuals.
The residual diagnostics indicate that the model residuals do not exhibit significant autocorrelation, are normally distributed, and have constant variance.

SEASONAL ARIMA (SARIMA) MODEL SELECTION WITH AUTO_ARIMA

```
# Initiate auto_arima run on original dataset
model_s = pm.auto_arima(df,
                        seasonal=True, m=90,
                        start_p=1, start_q=1,
                        max_p=2, max_q=2,
                        max_P=2, max_Q=2,
                        d=1, D=1,  # Ensures reviewing higher differencing order
                        trace=True,
                        error_action='ignore',
                        suppress_warnings=True)
model_s.summary()
```

```
Best model:  ARIMA(1,1,0)(0,1,0)[90]
Total fit time: 313.532 seconds
```

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	731
Model:	SARIMAX(1, 1, 0)x(0, 1, 0, 90)	**Log Likelihood**	-623.505
Date:	Wed, 12 Jun 2024	**AIC**	1251.010
Time:	12:19:53	**BIC**	1259.933
Sample:	01-01-2020	**HQIC**	1254.473
	- 12-31-2021		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.5007	0.035	-14.337	0.000	-0.569	-0.432
sigma2	0.4107	0.021	19.674	0.000	0.370	0.452

Ljung-Box (L1) (Q):	0.07	**Jarque-Bera (JB):**	4.69
Prob(Q):	0.79	**Prob(JB):**	0.10
Heteroskedasticity (H):	1.13	**Skew:**	-0.01
Prob(H) (two-sided):	0.36	**Kurtosis:**	3.42

I applied auto_arima to the original dataset with seasonal parameters with seasonal=True and m=90 to account for seasonality with a frequency of 90 days. The parameter settings (start_p, start_q, max_p, max_q, max_P, max_Q, d, D) were configured to consider different orders of differencing and seasonal differencing.
The best model determined by auto_arima is ARIMA(1,1,0)(0,1,0)[90]. The ARIMA part indicates one autoregressive term (AR), first-order differencing, and no moving average term (MA). The seasonal part indicates no seasonal AR or MA, first-order seasonal differencing with a

seasonal period of 90. The Ljung-Box (L1) (Q): 0.07, Prob(Q): 0.79: The Ljung-Box test for autocorrelation at lag 1 has a very low Q-value with a high p-value, indicating no significant autocorrelation in the residuals.

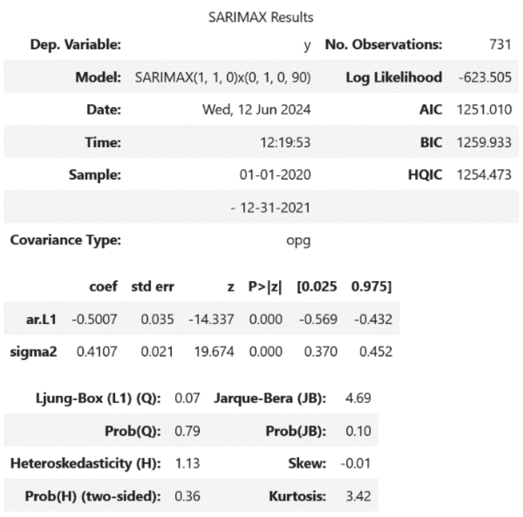
The residual diagnostics indicate that the model residuals do not exhibit significant autocorrelation, are marginally normally distributed, and have constant variance.

FITTING SARIMAX INTO ORIGINAL DATASET

```
from statsmodels.tsa.statespace.sarimax import SARIMAX

# Fit SARIMA model
model_sarima = SARIMAX(df, order=(1,1,0), seasonal_order=(0,1,0,90))
final_results = model_sarima.fit()
final_results.summary()
```

I used the best model to manually specify the ARIMA model to fit the original dataset.



SARIMAX Results

Dep. Variable:	y	**No. Observations:**	731
Model:	SARIMAX(1, 1, 0)x(0, 1, 0, 90)	**Log Likelihood**	-623.505
Date:	Wed, 12 Jun 2024	**AIC**	1251.010
Time:	12:19:53	**BIC**	1259.933
Sample:	01-01-2020	**HQIC**	1254.473
	- 12-31-2021		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.5007	0.035	-14.337	0.000	-0.569	-0.432
sigma2	0.4107	0.021	19.674	0.000	0.370	0.452

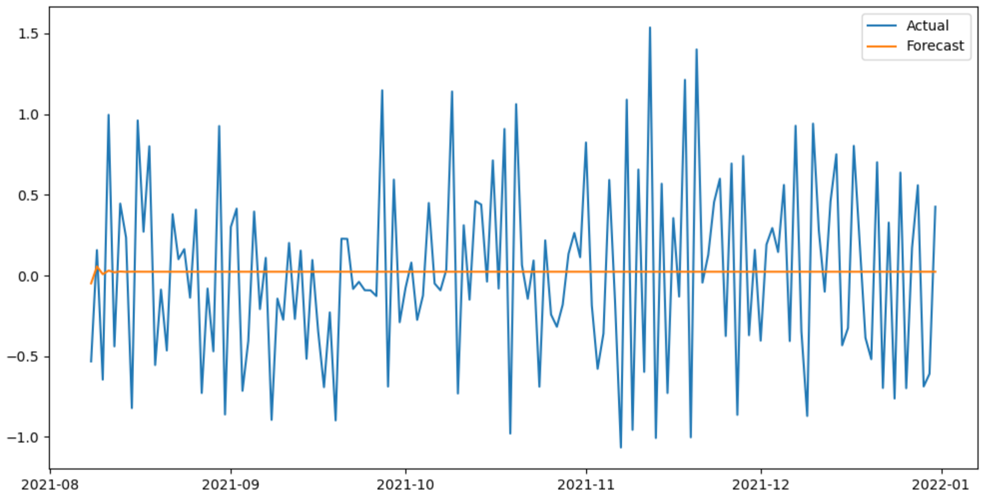
Ljung-Box (L1) (Q):	0.07	**Jarque-Bera (JB):**	4.69
Prob(Q):	0.79	**Prob(JB):**	0.10
Heteroskedasticity (H):	1.13	**Skew:**	-0.01
Prob(H) (two-sided):	0.36	**Kurtosis:**	3.42

The AIC for this model is 1251.010**.** Lower values indicate a better fit.

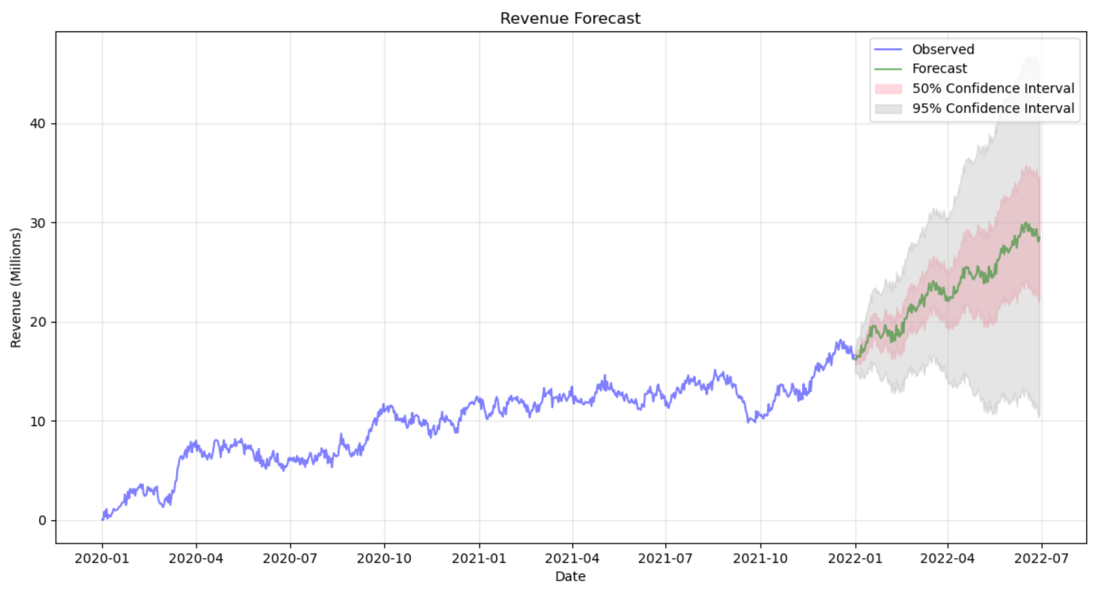
The Ljung-Box test for autocorrelation at lag 1 has a very low Q-value with a high p-value, indicating no significant autocorrelation in the residuals. The residual diagnostics indicate that the model residuals do not exhibit significant autocorrelation, are approximately normally distributed, and have constant variance, suggesting that the model fits the data well.

FORECASTING USING THE ARIMA MODEL

I generated forecasted values for the same duration as the test dataset using the fitted ARIMA model. This is a critical step for evaluating the model's performance by comparing the forecasted values to the actual values of the dataset.



I generated a 180-day forecast using a fitted SARIMA model, the forecasted data, and the 50% and 95% confidence intervals. The visualizations help in understanding how well the model forecasts. future revenue.



OUTPUT & CALCULATIONS

1. Perform seasonal decomposition on the differenced time series.

```
# Decompose the differenced series
decomposition = seasonal_decompose(df_stationary)
```

2. Forecast future values and store the forecast values. The forecast() method generates predictions equal to the test dataset's length. Then, the forecasted values are stored in the forecast variable.

```
# Forecast for the length of the test data
forecast = results.forecast(steps=len(test))
forecast
```

```
2021-08-08   -0.048621
2021-08-09    0.056441
2021-08-10    0.008147
2021-08-11    0.030347
2021-08-12    0.020142
                 ...
2021-12-27    0.023356
2021-12-28    0.023356
2021-12-29    0.023356
2021-12-30    0.023356
2021-12-31    0.023356
Freq: D, Name: predicted_mean, Length: 146, dtype: float64
```

3. This code generates a forecast for the next 10 days using my SARIMAX model. It provides the predicted mean values and the confidence intervals.

```
# The prediction of this model, set to 180 days
final_forecast = final_results.get_forecast(steps=180)

# The predicted mean
mean_forecast = final_forecast.predicted_mean
mean_forecast
```

```
2022-01-01    16.173009
2022-01-02    16.135897
2022-01-03    16.541553
2022-01-04    16.513874
2022-01-05    16.411448
                ...
2022-06-25    28.772319
2022-06-26    29.331427
2022-06-27    28.644399
2022-06-28    28.035574
2022-06-29    28.461559
Freq: D, Name: predicted_mean, Length: 180, dtype: float64
```

4. The confidence intervals give a range of values within which the true future values are expected to fall. The 50% confidence interval is narrower and gives a more likely range, while the 95% confidence intervals is wider, providing a more conservative estimate of uncertainty.

```
# Establishing the lower and upper confidence limits for a 50% confidence interval
confidence_intervals_50 = final_forecast.conf_int(alpha=0.5)
lower_50 = confidence_intervals_50.iloc[:, 0]
upper_50 = confidence_intervals_50.iloc[:, 1]
print("50% Confidence Interval")
print(confidence_intervals_50)

# Establishing the lower and upper confidence limits for a 95% confidence interval
confidence_intervals_95 = final_forecast.conf_int(alpha=0.05)
lower_95 = confidence_intervals_95.iloc[:, 0]
upper_95 = confidence_intervals_95.iloc[:, 1]
print("95% Confidence Interval")
print(confidence_intervals_95)
```

```
50% Confidence Interval
            lower Revenue  upper Revenue
2022-01-01      15.740756      16.605262
2022-01-02      15.652751      16.619043
2022-01-03      15.959720      17.123385
2022-01-04      15.872470      17.155277
2022-01-05      15.704576      17.118320
...                   ...            ...
2022-06-25      22.753208      34.791431
2022-06-26      23.284810      35.378044
2022-06-27      22.570401      34.718396
2022-06-28      21.934319      34.136830
2022-06-29      22.333167      34.589951

[180 rows x 2 columns]
95% Confidence Interval
            lower Revenue  upper Revenue
2022-01-01      14.916948      17.429070
2022-01-02      14.731949      17.539845
2022-01-03      14.850837      18.232269
2022-01-04      14.650053      18.377694
2022-01-05      14.357385      18.465511
...                   ...            ...
2022-06-25      11.281703      46.262936
2022-06-26      11.760884      46.901970
2022-06-27      10.994292      46.294506
2022-06-28      10.306260      45.764889
2022-06-29      10.653390      46.269729
```

CODE

Code attached as churn_task1.ipynb

PART V: DATA SUMMARY & IMPLICATIONS

RESULTS

SELECTION OF AN ARIMA MODEL

To identify the most suitable ARIMA model, I used the auto_arima function, which automatically selects the best parameters for the ARIMA model based on the data. This function tests different combinations of p, d, and q parameters and selects the one with the lowest AIC.

The final and best model is ARIMA(1,1,0)(0,1,0)[90]. The chosen model shown as ARIMA(1,1,0) indicates a first-order autoregressive model (AR=1), first-order differencing (I=1), and no moving average component (MA=0). The Seasonal Component shown as (0,1,0)[90] has no seasonal autoregressive or moving average components but includes seasonal differencing with a period of 90 days.

THE PREDICTION INTERVAL OF THE FORECAST

The forecast for 180 days includes mean forecast and confidence intervals of 50% Confidence Interval and 95% Confidence Interval. The 50% Confidence Interval is narrower, indicating more likely future values. On the other hand, the 95% Confidence Interval shows a wider range, providing a conservative estimate of future values.

A JUSTIFICATION OF THE FORECAST LENGTH

The forecast length of 180 days is based on business relevance, seasonality, and model reliability. Firstly, business relevance specifically focuses on the telecommunications industry. A six-month forecast helps in strategic planning for customer retention efforts and marketing campaigns. Furthermore, given the model's seasonal component with a 90-day period, the 180-day forecast captures two full seasonal cycles. Lastly, a 180-day forecast makes for a more reliable model. Forecasts beyond this period may introduce higher uncertainty, making them less reliable for practical decision-making.

MODEL EVALUATION PROCEDURE AND ERROR METRIC

Performance Metric	*Score*
Mean Absolute Error (MAE)	0.4670
Mean Squared Error (MSE)	0.3246
Root Mean Squared Error (RMSE)	0.5697

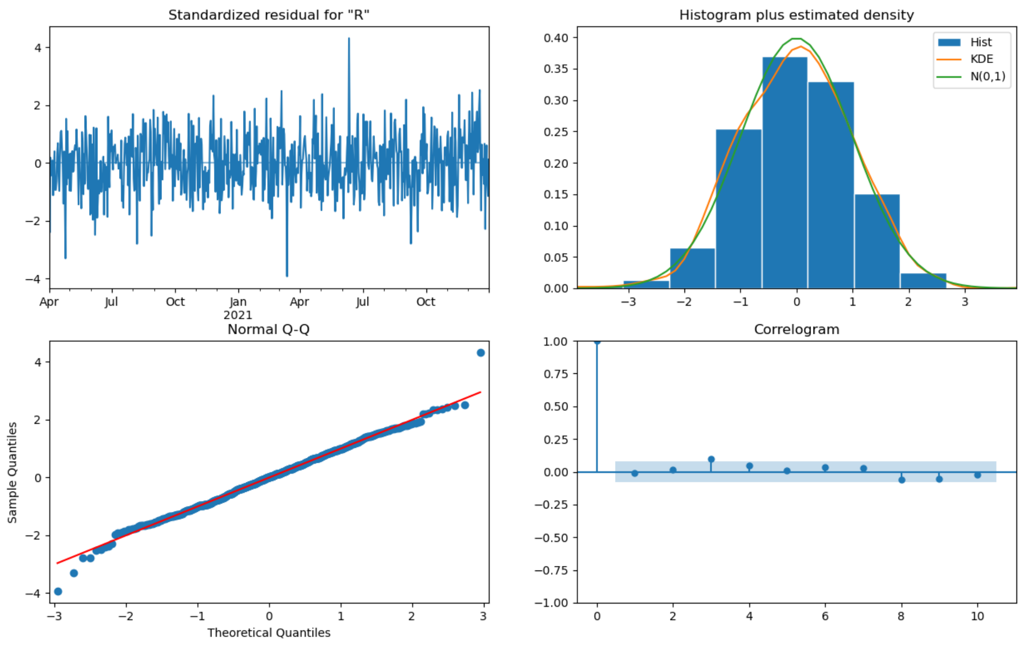
The MAE measures the average magnitude of the error in a set of predictions without considering their direction. On average, the model's predictions are off by about 0.47 million dollars. This indicates how much the forecast deviates from the actual values on a daily basis.

The MSE measures the average of the squares of the errors. This metric is more sensitive to large errors because squaring the differences magnifies the larger errors more than the smaller ones. An MSE of 0.32 indicates that, on average, the square of the difference between the predicted and actual values is about 0.32. This value is less interpretable on its own because it is squared millions of dollars.

The RMSE is the square root of the MSE and provides error metrics in the same units as the original data, thus making it more interpretable. An RMSE of approximately 0.57 indicates that, on average, the magnitude of the model's prediction is about 0.57 million errors.

These performance metrics suggest that while the model has a relatively small average error, there are instances where the error is larger, as indicated by the higher RMSE compared to MAE.

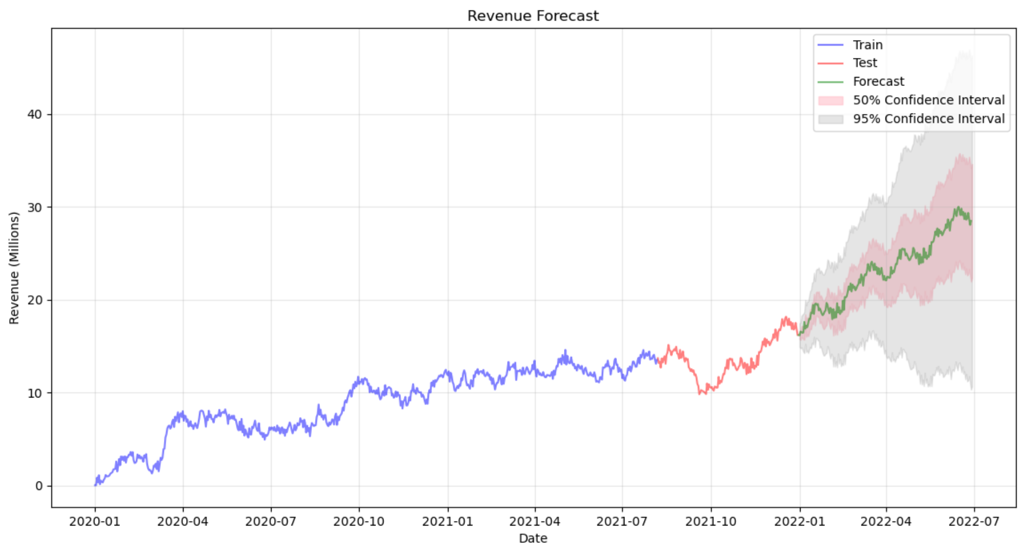
PLOT DIAGNOSTICS



Generate a set of diagnostic plots that help evaluate the model's assumptions and performance using plot_diagnostics().

The first graph is the Standardized Residuals Vs. Time. No clear pattern or trend is evident, which is desirable for well-behaved residuals. The second graph is the Histogram With Estimated Density. The superimposed normal distribution curve suggests that residuals follow a normal distribution. The empirical density line (KDE) also indicates the data distribution. The normal Q-Q plot has dots closely following the reference line. This suggests that residuals are approximately normally distributed. Lastly, the Scale-Locations plot has the residuals plotted against fitted values. The spread of residuals should be uniform across fitted values.

ANNOTATED VISUALIZATION



The graph shows the forecasted revenue extends beyond the actual data. The model predicts revenue growth, but the confidence intervals acknowledge uncertainty.

RECOMMENDATIONS

I have three recommendations for this analysis.

Firstly, integrate forecasting into business strategy. Use the revenue forecasts to make informed decisions about resource allocation. For example, anticipate periods of lower revenue to plan cost-saving measures. On the other hand, identify periods of expected high churn and proactively implement customer retention strategies with targeted promotion.

Furthermore, investigate any significant deviation between the forecasted and actual revenue to understand the relying causes. This could involve deeper analysis into specific periods or customer segments.

Lastly, monitor and update the model regularly. Continuously update the model with new data to improve its accuracy. This will ensure that the model remains relevant and accurate over time. It is also important to retrain the model periodically to capture any shifts in customer behavior.

PART VI: REPORTING

The executed notebook is attached as churn_task1.html.

THIRD-PARTY CODE SOURCES

No sources cited

SOURCES

No sources cited