```
#%% md
D.  Summarize the data cleaning process by doing the following:

1.  Provide a line graph visualizing the realization of the time series.

2.  Describe the time step formatting of the realization, including any gaps in measurement and the length of the sequence.

3.  Evaluate the stationarity of the time series.

4.  Explain the steps used to prepare the data for analysis, including the training and test set split.

5.  Provide a copy of the cleaned dataset.
#%% md
Step 1: Install pmdarima library for AutoARIMA
#%%
!pip install pmdarima
#%% md
Step 2: Upload Data from Files and Read into Dataframe
#%%
import pandas as pd

# Read data from a CSV file
file_path = (r"C:\Users\gabri\OneDrive\Documents\Education\WGU\MSDA\D603 - "
             r"Machine Learning\D603 Task 3\medical_clean.csv")
df = pd.read_csv(file_path)

#%% md
Step 3: D1. Provide a line graph visualizing the realization of the time series.
#%%
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
33
34 revData = df
35
36
37 # Convert the 'Day' column to datetime format
   starting from 1/1/2023
38 revData['Date'] = pd.to_datetime('2023-01-01') + pd.
   to_timedelta(revData['Day'] - 1, unit='D')
39
40 # Set the 'Date' column as the index and set the
   frequency to daily
41 revData.set_index('Date', inplace=True)
42 revData = revData.asfreq('D')
43 revData = revData.drop('Day', axis=1)
44
45 # Plot the time series data
46 plt.figure(figsize=(10, 6))
47 plt.plot(revData['Revenue'])
48 plt.xlabel('Date')
49 plt.ylabel('Revenue')
50 plt.title('Time Series of Revenue')
51 plt.show()
52
53
54
55 #%% md
56 Step 4: D2: The time series data is formatted with
   daily time steps, starting from January 1, 2023. The
    'Day' column has been converted to a datetime format
   , and the 'Date' column is set as the index with a
   daily frequency. The dataset contains 731
   observations, representing a continuous sequence of
   daily revenue data over 2 years without any gaps in
   measurement.
57
58 #%%
59
60 # Check for missing data
61 print("If there were no missing rows, there would be
   731 rows of minute data")
62 print("The actual length of the DataFrame is:", len(
```

```
revData))
if len(revData) == 731:
    print("There are no missing values.\n")
else:
    # Everything
    set_everything = set(range(731))

    # The revenue index as a set
    set_revenue = set(revData.index)

    # Calculate the difference
    set_missing = set_everything - set_revenue

    # Print the difference
    print("Missing rows: ", set_missing)

    # Fill in the missing rows
    revData = revData.reindex(range(731), method='ffill')


#%% md
Step 5: D3: Evaluate the stationarity of the time series. The time series is not stationary.
#%%
from statsmodels.tsa.stattools import adfuller

# Function to check the stationarity of a time series
def check_stationarity(series, significance_level=0.05):
    result = adfuller(series.dropna())
    p_value = result[1]
    print('ADF Statistic:', result[0])
    print('p-value:', result[1])
    return p_value <= significance_level

# Check stationarity
is_stationary = check_stationarity(revData['Revenue'])
print(f'The time series is {"stationary" if
```

```
 97 is_stationary else "not stationary"}.')
 98
 99 #%% md
100 Since the time series is not stationary, the data
    will be differenced. The time series must be
    stationary, otherwise the analysis may be unreliable
    .
101 #%%
102 # Track the number of differencing steps
103 differencing_steps = 0
104
105 # Apply differencing and recheck for stationarity
    until the data becomes stationary
106 if not check_stationarity(revData['Revenue']):
107     print("The time series is not stationary.
    Applying differencing.")
108     revData_diff= revData.diff()
109     revData_diff.dropna(inplace=True)
110     differencing_steps += 1
111 #%% md
112 Step 6: D4. Explain the steps used to prepare the
    data for analysis, including the training and test
    set split.
113
114 The data preparation process involved the following
    steps:
115 1. Reading the CSV file into a DataFrame.
116 2. Checking for missing rows.
117 3. Converting the 'Day' column to a datetime format
    starting from January 1, 2023.
118 4. Setting the 'Date' column as the index and
    setting the frequency to daily.
119 5. Splitting the data into training and test sets,
    with 80% of the data used for training and 20% for
    testing.
120 6. Evaluate for stationarity.
121 7. Non-stationary data must be differenced.
122 8. Analyze differenced data.
123
124 #%%
125 # Split the data into training and test sets
```

```
126 train_size = int(len(revData_diff) * 0.8)
127 train, test = revData_diff.iloc[:train_size],
    revData.iloc[train_size:]
128
129 train_original, test_original = revData.iloc[:
    train_size], revData.iloc[train_size:]
130 #%% md
131 Step 7: D5. Provide a copy of the cleaned dataset.
132 #%%
133 # Save the cleaned dataset to a CSV file
134 revData_diff.to_csv('cleaned_data.csv')
135
136 #%% md
137 E.  Analyze the time series dataset by doing the
    following:
138
139 1.  Report the annotated findings with
    visualizations of your data analysis, including the
    following elements:
140
141     •   trends
142
143     •   the autocorrelation function
144
145     •   the spectral density
146
147     •   the decomposed time series
148
149     •   confirmation of the lack of trends in the
    residuals of the decomposed series
150
151 2.  Identify an autoregressive integrated moving
    average (ARIMA) model that accounts for the observed
     trend and seasonality of the time series data.
152
153 3.  Perform a forecast using the derived ARIMA model
     identified in part E2.
154
155 4.  Provide the output and calculations of the
    analysis you performed.
156 #%% md
```

```
Step 8: E1. Report the annotated findings with visualizations of your data analysis, including the following elements:

• trends

• the autocorrelation function

• the spectral density

• the decomposed time series
#%%
from scipy.stats import shapiro, norm, probplot

# Plot the differenced time series
plt.figure(figsize=(12, 4))
plt.plot(revData_diff.index, revData_diff['Revenue'], label='Differenced Time Series')
plt.title('Differenced Time Series')
plt.xlabel('Date')
plt.ylabel('Revenue')
plt.legend()
plt.grid(True)
plt.show()

# Check for normality of the differenced data
stat, p = shapiro(revData_diff['Revenue'])
print('Shapiro-Wilk Test: Statistics=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
    print('Differenced data follows a normal distribution (fail to reject H0)')
else:
    print('Differenced data does not follow a normal distribution (reject H0)')

# Plot the histogram and Q-Q plot of the differenced data
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
```

```
plt.hist(revData_diff['Revenue'], bins=30, density=True, alpha=0.6, color='g')
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
p = norm.pdf(x, revData_diff['Revenue'].mean(), revData_diff['Revenue'].std())
plt.plot(x, p, 'k', linewidth=2)
plt.title('Histogram of Differenced Data')
plt.xlabel('Revenue')
plt.ylabel('Density')

plt.subplot(1, 2, 2)
probplot(revData_diff['Revenue'], dist="norm", plot=plt)
plt.title('Q-Q Plot of Differenced Data')

plt.tight_layout()
plt.show()


#%% md
Comments: The above graphs indicate that any trends in the data are the result of a random walk. The differenced time series is stationary, meaning that the mean and standard deviation do not change with time. The histogram of the values shows a normal distribution.
#%%
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
print(revData_diff)

# Plot the ACF and PACF on the differenced dataset
fig, axes = plt.subplots(2,1)
# Plot the ACF
plot_acf(revData_diff, lags=20, ax=axes[0])
# Plot the PACF
plot_pacf(revData_diff, lags=20, ax=axes[1])
plt.show()

#%% md
```

```
Comments: The autocorrelation does not show
    seasonality. It also suggests that an AR(2) model
    would fit.
#%%
from scipy.signal import periodogram

# Plot the spectral density
frequencies, spectrum = periodogram(revData_diff['
    Revenue'])
plt.figure(figsize=(12, 4))
plt.semilogy(frequencies, spectrum)
plt.title('Spectral Density')
plt.xlabel('Frequency')
plt.ylabel('Spectrum')
plt.grid(True)
plt.show()

# Apply FFT to the differenced data
fft_result = np.fft.fft(revData_diff['Revenue'])
freq = np.fft.fftfreq(len(revData_diff['Revenue']))

# Plot the FFT result
plt.figure(figsize=(12, 4))
plt.plot(freq, np.abs(fft_result))
plt.title('FFT of Differenced Data')
plt.xlabel('Frequency')
plt.ylabel('Amplitude')
plt.grid(True)
plt.show()
#%% md
Comments:
The above graphs show a lack of seasonlity, with no
    outliers or remarkable behavior.
#%%
from statsmodels.tsa.seasonal import
    seasonal_decompose
# Decompose the time series
decomposition = seasonal_decompose(revData_diff,
    model='additive')
trend = decomposition.trend
seasonal = decomposition.seasonal
```

```
residual = decomposition.resid

plt.figure(figsize=(12, 8))
plt.subplot(411)
plt.plot(revData_diff, label='Differenced Time Series')
plt.legend(loc='best')
plt.subplot(412)
plt.plot(trend, label='Trend')
plt.legend(loc='best')
plt.subplot(413)
plt.plot(seasonal, label='Seasonality')
plt.legend(loc='best')
plt.subplot(414)
plt.plot(residual, label='Residuals')
plt.legend(loc='best')
plt.tight_layout()
plt.show()

# Confirm lack of trends in the residuals
plt.figure(figsize=(12, 4))
plt.plot(residual, label='Residuals')
plt.title('Residuals of Decomposed Series')
plt.xlabel('Date')
plt.ylabel('Residuals')
plt.legend()
plt.grid(True)
plt.show()

#%% md
Comments: The above decomposed series further illustrate the lack of seasonlity and positive trend or drift in the data. This time series is the product of a random walk.
#%% md
E2. Identify an autoregressive integrated moving average (ARIMA) model that accounts for the observed trend and seasonality of the time series data.
#%% md
Training:
#%% md
```

```
293
294 #%%
295 import itertools
296 from statsmodels.tsa.arima.model import ARIMA
297
298 # Perform grid search to determine the best
    parameters for ARIMA model on training data
299 p = range(0, 5)
300 d = differencing_steps
301 q = range(0, 5)
302 pdq = list(itertools.product(p, [d], q))
303
304 best_aic_train = np.inf
305 best_pdq_train = None
306 best_model_train = None
307
308 for param in pdq:
309     try:
310         model = ARIMA(train, order=param)
311         results = model.fit()
312         if results.aic < best_aic_train:
313             best_aic_train = results.aic
314             best_pdq_train = param
315             best_model_train = results
316     except:
317         continue
318
319 print(f'Best ARIMA model for training data: ARIMA{
    best_pdq_train} - AIC:{best_aic_train} \n\n')
320 print(best_model_train.summary())
321
322 # Calculate residuals and mean absolute error for
    training data
323 residuals_train = best_model_train.resid
324 mae_train = np.mean(np.abs(residuals_train))
325 print('Mean absolute error for training data:',
    mae_train)
326
327 #%% md
328 Test:
329 #%%
```

```
330 # Perform grid search to determine the best
    parameters for ARIMA model on test data
331 best_aic_test = np.inf
332 best_pdq_test = None
333 best_model_test = None
334
335 for param in pdq:
336     try:
337         model = ARIMA(test, order=param)
338         results = model.fit()
339         if results.aic < best_aic_test:
340             best_aic_test = results.aic
341             best_pdq_test = param
342             best_model_test = results
343     except:
344         continue
345
346 print(f'Best ARIMA model for test data: ARIMA{
    best_pdq_test} - AIC:{best_aic_test}')
347 print(best_model_test.summary())
348
349 # Calculate residuals and mean absolute error for
    test data
350 residuals_test = best_model_test.resid
351 mae_test = np.mean(np.abs(residuals_test))
352 print('Mean absolute error for test data:', mae_test
    )
353
354 #%% md
355 E3. Perform a forecast using the derived ARIMA model
     identified in part E2.
356 #%%
357 # Fit ARIMA model on test data
358 model_test = ARIMA(test_original, order=
    best_pdq_test)
359 results_test = model_test.fit()
360 print(results_test.summary())
361
362 # Calculate mean absolute error for test data
363 mae_test = np.mean(np.abs(results_test.resid))
364 print('Mean absolute error for test data:', mae_test
```

```
)

# Calculate MAE for the forecast created by the training data on the test data
forecast_test = best_model_train.get_forecast(steps=len(test))
mean_forecast_test = forecast_test.predicted_mean
mae_forecast_test = np.mean(np.abs(mean_forecast_test - test['Revenue']))
print('Mean absolute error for training forecast on test data:', mae_forecast_test)


# Generate predictions and confidence intervals for test data
prediction = results_test.get_prediction(start=-146, end=len(test) + 90)  # Extend forecast by 90 days
mean_prediction = prediction.predicted_mean
confidence_intervals = prediction.conf_int()
lower_limit = confidence_intervals.iloc[:, 0]
upper_limit = confidence_intervals.iloc[:, 1]


# Calculate the 90-day moving average for the test set and extend it by an additional 90 days using the ARIMA model
extended_test = pd.concat([train_original['Revenue'], test_original['Revenue'], mean_prediction[-90:]])
moving_average_90_extended = extended_test.rolling(window=90).mean()

# Plot the training data, test data, and the 90-day moving average with its confidence interval
plt.figure(figsize=(12, 4))
plt.plot(train.index, train_original['Revenue'], label='Training Data')
plt.plot(test.index, test_original['Revenue'], label='Observed (Test Set)')
plt.plot(moving_average_90_extended.index, moving_average_90_extended, color='b', label='90-Day Moving Average')
```

```
390 plt.title('Training Data, Test Data, and 90-Day
    Moving Average with 90-Day Extension')
391 plt.xlabel('Date')
392 plt.ylabel('Revenue (Millions)')
393 plt.legend()
394 plt.grid(True)
395 plt.show()
396
397
398 # Extend the moving average by an additional 90 days
399 extended_moving_average = pd.concat([revData['
    Revenue'], mean_prediction[-90:]]).rolling(window=90
    ).mean()
400
401 # Plot the moving average for the entire dataset
    with the 90-day forecast and confidence interval
402 plt.figure(figsize=(12, 4))
403 plt.plot(revData.index, revData['Revenue'], label='
    Original Time Series')
404 plt.plot(extended_moving_average.index,
    extended_moving_average, color='r', label='Moving
    Average')
405 plt.plot(mean_prediction.index[-90:],
    mean_prediction[-90:], color='b', label='90-Day
    Forecast')
406 plt.fill_between(lower_limit.index[-90:],
    lower_limit[-90:], upper_limit[-90:], color='pink',
    alpha=0.3, label='Confidence Interval')
407 plt.title('Moving Average for the Entire Dataset
    with 90-Day Forecast')
408 plt.xlabel('Date')
409 plt.ylabel('Revenue (Millions)')
410 plt.legend()
411 plt.grid(True)
412 plt.show()
413
414
415 # Plot the final 90 days of the test data, the 90-
    day forecast with the confidence interval, and the
    90-day moving average
416 plt.figure(figsize=(12, 4))
```

```
417 plt.plot(test.index[-90:], test['Revenue'][-90:],
    label='Observed (Test Set)')
418 plt.plot(mean_prediction.index[-90:],
    mean_prediction[-90:], color='r', label='Forecast')
419 plt.fill_between(lower_limit.index[-90:],
    lower_limit[-90:], upper_limit[-90:], color='pink')
420 plt.plot(extended_moving_average.index[-180:],
    extended_moving_average[-180:], color='b', label='90
    -Day Moving Average')
421 plt.title('Final 90 Days of Test Data, 90-Day
    Forecast, and 90-Day Moving Average')
422 plt.xlabel('Date')
423 plt.ylabel('Revenue (Millions)')
424 plt.legend()
425 plt.grid(True)
426 plt.show()
427
428 #%% md
429 E4. Provide the output and calculations of the
    analysis you performed.
430 #%%
431 print(f'Best ARIMA model for training data: ARIMA{
    best_pdq_train} - AIC:{best_aic_train}')
432 print('Mean absolute error for training data:',
    mae_train)
433
434 print('Mean absolute error for training forecast on
    test data:', mae_forecast_test)
435
436 print(f'Best ARIMA model for test data: ARIMA{
    best_pdq_test} - AIC:{best_aic_test}')
437 print('Mean absolute error for test data:', mae_test
    )
438 #%% md
439 Step F: Summarize Your Findings and Assumptions
440 1. Discuss the results of your data analysis,
    including the following:
441
442 *   The Selection of an ARIMA Model
443
444         The ARIMA model was selected using the using
```

```
444  a grid search and the ARIMA method, which
    identified the best model parameters based on the
    lowest AIC. The AIC finds models that fit well and
    penalizes for overfitting, meaning that the model
    with the lowest AIC will not be over- or under-
    fitted. The selected model accounts for the (lack of
    ) observed trend and seasonality in the time series
    data. The best parameters (p, d, q) are (2, 1, 1)
    respectively, which fits the discoveries of the ACF
    plot.
445
446 *   The Prediction Interval of the Forecast
447
448         The prediction interval of the forecast
    provides a range within which the true values are
    expected to lie with a certain probability, with
    alpha set to 0.05. The confidence intervals were
    calculated for both the test data forecast and the
    extended forecast.
449
450 *   A Justification of the Forecast Length
451
452         The forecast was extended to 90 days past
    the final date to provide a comprehensive view of
    the future revenue trends over the subsequent
    financial quarter. This length was chosen as most
    financial metrics, like revenue, are measured by
    quarter (three months), allowing for better planning
     and decision-making. With two years of data, only
    one quarter was appropriate.
453
454 *   The Model Evaluation Procedure and Error Metric
455
456         The model evaluation procedure involved
    calculating the Mean Absolute Error (MAE) for the
    training and test forecasts. The MAE is a common
    error metric used to assess the accuracy of the
    forecast by measuring the absolute difference
    between the observed and predicted values. The best
    forecast should minimize the MAE, which while the
    training data had a lower MAE of 0.3587 compared to
```

```
test's 0.4752, when the training forecast was applied to the test data, the MAE spiked. This means that using fresher data is better for forecasting.

#%% md
F2.  Provide an annotated visualization of the forecast of the final model compared to the test set that includes the following:

•   the original output with the new prediction line and confidence cone

•   correct labeling
#%%
# Plot the moving average for the entire dataset with the 90-day forecast and confidence interval
plt.figure(figsize=(12, 4))
plt.plot(revData.index, revData['Revenue'], label='Original Time Series')
plt.plot(extended_moving_average.index, extended_moving_average, color='r', label='Moving Average')
plt.plot(mean_prediction.index[-90:], mean_prediction[-90:], color='b', label='90-Day Forecast')
plt.fill_between(lower_limit.index[-90:], lower_limit[-90:], upper_limit[-90:], color='pink', alpha=0.3, label='Confidence Interval')
plt.title('Moving Average for the Entire Dataset with 90-Day Forecast')
plt.xlabel('Date')
plt.ylabel('Revenue (Millions)')
plt.legend()
plt.grid(True)
plt.show()
```