|  |  |
| --- | --- |
|  |  |

Date: 08/10/2023

Task :Assessment 2

Module – SIG742 (Modern Data Science)

Masters in Data Science



CODE REPORT

Contents

[List of References 2](#_Toc146824319)

[Part 1: 2](#_Toc146824320)

[Group 2](#_Toc146824321)

[Question 1. 1 2](#_Toc146824322)

[Answer 1. 1 -First part 3](#_Toc146824323)

[Output 3](#_Toc146824324)

[Explanation: 4](#_Toc146824325)

[Answer 1.1 – Second part 4](#_Toc146824326)

[Question 1. 2 5](#_Toc146824327)

[Answer 1. 2 – First part 5](#_Toc146824328)

[Output 5](#_Toc146824329)

[Explanation 5](#_Toc146824330)

[Answer 1. 2 – Second part 5](#_Toc146824331)

[Output 6](#_Toc146824332)

[Explanation 6](#_Toc146824333)

[Answer 1. 2 – Third part 6](#_Toc146824334)

[Output 7](#_Toc146824335)

[Explanation 7](#_Toc146824336)

[Question 1. 3 8](#_Toc146824337)

[Answer 1.3 – First Part 8](#_Toc146824338)

[Output 8](#_Toc146824339)

[Explanation 9](#_Toc146824340)

[Answer 1.3 – Second Part 9](#_Toc146824341)

[Output 10](#_Toc146824342)

[Explanation 10](#_Toc146824343)

[Answer 1.3 – Third Part 11](#_Toc146824344)

[Output 13](#_Toc146824345)

[Explanation 13](#_Toc146824346)

# List of References

* NA

# Part 1:

Data Acquisition and Manipulation

There are 10 questions in this part for total 60 marks, and each question is for 5 marks. The quality of

your explanation in the report and video will be 10 marks for all questions.

You are required to use Google Colab to finish all the coding in the code block cell, and provide

sufficient coding comments, and also save the result of running as well.

The (Item\_listing\_category.zip) data used for this part could be found in here. You will need to use

Pandas to read the unzipped (csv) data for starting.

# Group

|  |  |  |  |
| --- | --- | --- | --- |
| Author Name | Deakin ID | Questions Answered | Total |
| Ramchandar Mariappan | 223914532 | 1.1 to 1.3 | 3 |
| Simranjit Singh |  | 1.4 to 1.7 | 4 |
| Uthara Ravichanthar |  | 1.8 to 1.10 | 4 |

## Question 1. 1

|  |
| --- |
| Find the missing values:  • Write the function missing\_values\_table and use the dataframe as the input. The function  should return the information of missing values by column (only for columns which have  missing values and the returned value should be the count of rows has missing values);  • For columns which have missing values, could you impute the missing values with the  mean value of the particular columns? (if you think it could not be done with mean value,  write down the reason in comments and report rather than code |

## Answer 1. 1 -First part

|  |
| --- |
| Write the function missing\_values\_table and use the dataframe as the input. The function  should return the information of missing values by column (only for columns which have  missing values and the returned value should be the count of rows has missing values);  # Define the missing\_values\_table function  def missing\_values\_table(item\_list\_df):      # Calculate the total number of missing values per column      missing\_count = item\_list\_df.isnull().sum()      # Filter columns with missing values (count > 0)      missing\_count = missing\_count[missing\_count > 0]      # Calculate the percentage of missing values per column      missing\_percentage = (missing\_count / len(item\_list\_df)) \* 100      # Create a DataFrame to display the missing value information      missing\_table = pd.DataFrame({          'Missing Values rows': missing\_count,          'Missing Values Percentage': missing\_percentage      })      return missing\_table  # Use the missing\_values\_table function  missing\_info = missing\_values\_table(item\_list\_df)  # Print the missing value information  print("Missing Values Information:")  print(missing\_info) |

## Output

|  |
| --- |
| Missing Values Information:  Missing Values rows Missing Values Percentage  category\_name 1539 0.432537  brand\_name 151956 42.707303  clean\_description 194 0.054524 |

## Explanation:

|  |
| --- |
| 1. The function is written to find the missing values. The def function is called by keeping item\_list\_df. item\_list\_df possess the data set in the form of dataframe   [def missing\_values\_table(item\_list\_df):]   1. Calculation of the null values through isnull function. item\_list\_df.isnull().sum() using this code to count the number of missing values in the dataset   [missing\_count = item\_list\_df.isnull().sum()]   1. Then filter is applied to identify where missing count is greater than 0 . This will filter values that have null values   [missing\_count = missing\_count[missing\_count > 0]]   1. Additionally, calculation of the null values % using (missing\_count / len(item\_list\_df)) \* 100 where missing count was found in step 3. Using len function the total records in the dataset is calculated   [missing\_percentage = (missing\_count / len(item\_list\_df)) \* 100]   1. Data frame is set up using pd.DataFrame and in this Missing count from step 3 and missing percentage from step 4 is used.   [missing\_table = pd.DataFrame({  'Missing Values rows': missing\_count,  'Missing Values Percentage': missing\_percentage  })]   1. The function is closed with the return statement   return missing\_table   1. Now calling the function module to print the missing information. 2. missing\_info = missing\_values\_table(item\_list\_df) -> the defined function is now called for printing the information   [missing\_info = missing\_values\_table(item\_list\_df)]   1. Then the print the missing values.   print("Missing Values Information:"), print(missing\_info)-> using print function to display missing values. |

## Answer 1.1 – Second part

|  |
| --- |
| For columns which have missing values, could you impute the missing values with the  mean value of the particular columns? (if you think it could not be done with mean value,  write down the reason in comments and report rather than code)   1. Category Name 2. Brand Name 3. Clean description   Are the three columns that has missing values. The data types of all three of them are object ie categorical.  All three variables which has missing values are categorical. Categorical variables cannot be imputed using a mean approach. Mean approach can be used only for continuous variables  Mode Imputation: This is a common choice when dealing with categorical data. K-Nearest Neighbors (K-NN) Imputation: This method imputes missing categorical values based on the categories of their nearest neighbors in the dataset |

-------------------------------------------------------------------------------------------------------------------------------------------------------

## Question 1. 2

|  |
| --- |
| Find the price information from the data: • Write code to print the median price of the items in the data; • What is the 90th percentile value on the price; • Draw the histogram chart for the price of the items in the data with 50 bins. |

## Answer 1. 2 – First part

|  |
| --- |
| Write code to print the median price of the items in the data  median\_price = item\_list\_df['price'].median()  print(f'The median price of the items in the data is', median\_price) |

## Output

|  |
| --- |
| The median price of the items in the data is 17.0 |

## Explanation

|  |
| --- |
| 1. median function is used to identify the median value against the target variable price   median\_price = item\_list\_df['price'].median()   1. Using print function to display the median price   print(f'The median price of the items in the data is', median\_price) |

## Answer 1. 2 – Second part

|  |
| --- |
| What is the 90th percentile value on the price  # Calculate the 90th percentile using numpy.percentile  percentile\_90 = np.percentile(item\_list\_df['price'], 90)  print(f'90th Percentile Price:', percentile\_90) |

## Output

|  |
| --- |
| 90th Percentile Price: 51.0 |

## Explanation

|  |
| --- |
| 1. 90th percentile is calculated using the numpy.percentile against the target variable price.   percentile\_90 = np.percentile(item\_list\_df['price'], 90)   1. Using the print function to display the output   print(f'90th Percentile Price:', percentile\_90) |

## Answer 1. 2 – Third part

|  |
| --- |
| Draw the histogram chart for the price of the items in the data with 50 bins.  # Specify the number of bins  num\_bins = 50  # Plot the histogram  plt.hist(item\_list\_df['price'], bins=num\_bins, color='blue', alpha=0.7)  plt.xlabel('Price')  plt.ylabel('Frequency')  plt.title('Histogram of Item Prices')  plt.grid(True)  # Show the plot  plt.show() |

## Output

|  |
| --- |
|  |

## Explanation

|  |
| --- |
| 1. Specifying the number of bins as 50 -   num\_bins = 50   1. Plotting the histogram using plt.hist with input parameters include the Target variable column, bins = 50 , color = blue and alpha value = 0.7 2. Xaxis label is given as price 3. Yaxis label is given as Frequency 4. Title is given as plt.title 5. Grid is kept as true with plt.grid(True)   # Plot the histogram  plt.hist(item\_list\_df['price'], bins=num\_bins, color='blue', alpha=0.7)  plt.xlabel('Price')  plt.ylabel('Frequency')  plt.title('Histogram of Item Prices')  plt.grid(True)   1. Finally displaying the plot using plt.show()   # Show the plot  plt.show() |

-------------------------------------------------------------------------------------------------------------------------------------------------------

## Question 1. 3

|  |
| --- |
| Question 1.3 Exploring the shipping information from the data:  • Write code to find out the percentage of the items that are paid by the buyers.  • Draw (two) histogram graphs in one plot on the price for seller pays shipping and buyer pays shipping (50 bins).  • When buying the items online, do you need to pay higher price if seller pays for the shipping? Write the code to find out (Compare the median price of items paid by buyers and items paid by sellers, and explain the result in the comment and report).  (Optional: You could use the subplot from EDA) |

## Answer 1.3 – First Part

|  |
| --- |
| Write code to find out the percentage of the items that are paid by the buyers. Please Note\* Assuming when shipping is 0 - then paid by buyers & 1 - then paid by sellers [ This is not given in the problem statement]  #Find the value counts of the shipping  item\_list\_df['shipping'].value\_counts()  #Taking the first value which is shipping as 0  items\_buyers = item\_list\_df['shipping'].value\_counts()[0]  #identifying the total records using the len function  leng\_item\_list\_df = len(item\_list\_df)  #percentage is dividing the first value from the value counts and the total  #records found using len function. Also it is rounded for 2 digits  Percentage\_buyers = round(items\_buyers/leng\_item\_list\_df\*100,2)  #print function to print the value  print(f'The percentage of items that are paid by buyers',Percentage\_buyers, '%') |

## Output

|  |
| --- |
| 0 197064  1 158744  Name: shipping, dtype: int64  The percentage of items that are paid by buyers 55.38 % |

## Explanation

|  |
| --- |
| 1. Identifying the value counts of 0’s and 1’s using the column shipping and value count function   item\_list\_df['shipping'].value\_counts()   1. From the list[0] position is hold to identify the shipment charges held by buyers   items\_buyers = item\_list\_df['shipping'].value\_counts()[0]   1. Using len function to identify the total records ie 0s and 1s   leng\_item\_list\_df = len(item\_list\_df)   1. Percentage is calculated by dividing point 2 and point 3 outputs   Percentage\_buyers = round(items\_buyers/leng\_item\_list\_df\*100,2)   1. Using print function to finally display the output   print(f'The percentage of items that are paid by buyers',Percentage\_buyers, '%') |

## Answer 1.3 – Second Part

|  |
| --- |
| Draw (two) histogram graphs in one plot on the price for seller pays shipping and buyer pays shipping (50 bins).  Please Note\* Assuming when shipping is 0 - then paid by buyers & 1 - then paid by sellers [ This is not given in the problem statement]  # Filter the DataFrame to separate data based on shipping category  seller\_pays\_shipping = item\_list\_df[item\_list\_df['shipping'] == 1]['price']  buyer\_pays\_shipping = item\_list\_df[item\_list\_df['shipping'] == 0]['price']  # Specify the number of bins (50 in this case)  num\_bins = 50  # Create subplots for side-by-side histograms  fig, axs = plt.subplots(1, 2, figsize=(12, 6))  # Plot the histogram for seller pays shipping  axs[0].hist(seller\_pays\_shipping, bins=num\_bins, color='blue', alpha=0.7)  axs[0].set\_xlabel('Price')  axs[0].set\_ylabel('Frequency')  axs[0].set\_title('Seller Pays Shipping')  axs[0].grid(True)  # Plot the histogram for buyer pays shipping  axs[1].hist(buyer\_pays\_shipping, bins=num\_bins, color='green', alpha=0.7)  axs[1].set\_xlabel('Price')  axs[1].set\_ylabel('Frequency')  axs[1].set\_title('Buyer Pays Shipping')  axs[1].grid(True)  # Adjust spacing between subplots  plt.tight\_layout()  # Show the plot  plt.show() |

## Output

|  |
| --- |
|  |

## Explanation

|  |
| --- |
| 1. The number of bins is specified as 50   num\_bins = 50   1. Subplot is created with side-by-side histogram with figsize spec as 12,6 2. plt.hist() to create the histogram with input parameters including the dataframe for both buyer and seller , number of bins, color as blue and alpha value is 0.7 . Here alpha means means that the histogram bars will be somewhat transparent, allowing you to see through them to some extent. 3. Xlabel, ylabel and title are included 4. Grid in the visual is set as True   # Plot the histogram for seller pays shipping  axs[0].hist(seller\_pays\_shipping, bins=num\_bins, color='blue', alpha=0.7)  axs[0].set\_xlabel('Price')  axs[0].set\_ylabel('Frequency')  axs[0].set\_title('Seller Pays Shipping')  axs[0].grid(True)  # Plot the histogram for buyer pays shipping  axs[1].hist(buyer\_pays\_shipping, bins=num\_bins, color='green', alpha=0.7)  axs[1].set\_xlabel('Price')  axs[1].set\_ylabel('Frequency')  axs[1].set\_title('Buyer Pays Shipping')  axs[1].grid(True)   1. Display the plot using   plt.show() |

## Answer 1.3 – Third Part

|  |
| --- |
| When buying the items online, do you need to pay higher price if seller pays for the shipping? Write the code to find out (Compare the median price of items paid by buyers and items paid by sellers, and explain the result in the comment and report).  # Filter the DataFrame for items paid by sellers and buyers  seller\_pays\_shipping = item\_list\_df[item\_list\_df['shipping'] == 1]['price']  buyer\_pays\_shipping = item\_list\_df[item\_list\_df['shipping'] == 0]['price']  # Calculate the median price for each category  median\_price\_seller\_pays = seller\_pays\_shipping.median()  median\_price\_buyer\_pays = buyer\_pays\_shipping.median()  # Create a bar plot to compare median prices  plt.bar(['Seller Pays', 'Buyer Pays'], [median\_price\_seller\_pays, median\_price\_buyer\_pays], color=['blue', 'green'])  plt.xlabel('Shipping Category')  plt.ylabel('Median Price')  plt.title('Median Price Comparison: Seller Pays vs. Buyer Pays')  plt.grid(True)  # Show the plot  plt.show()  # Calculate the price difference  price\_difference = median\_price\_buyer\_pays - median\_price\_seller\_pays  # Report the results  print(f'On comparison of when the price is higher between seller and buyer pays for shipping:\n')  print(f'The median price when seller pays is', median\_price\_seller\_pays)  print(f'The median price when buyer pays is', median\_price\_buyer\_pays)  if price\_difference < 0:      print("On average, items paid by sellers have a higher median price.")  elif price\_difference > 0:      print("On average, items paid by buyers have a higher median price.")  else:      print("There is no significant difference in the median prices.") |

## Output

|  |
| --- |
|  |

## Explanation

|  |
| --- |
| 1. The DataFrame is created with filters for the column shipping by classifying the buyer and seller   seller\_pays\_shipping = item\_list\_df[item\_list\_df['shipping'] == 1]['price']  buyer\_pays\_shipping = item\_list\_df[item\_list\_df['shipping'] == 0]['price']   1. Median price for the seller and buyer are then calculated against the variable created from dataframe using the function Median()   median\_price\_seller\_pays = seller\_pays\_shipping.median()  median\_price\_buyer\_pays = buyer\_pays\_shipping.median()   1. Bar plot is created with blue and green color to differentiate how much of shipment is paid by buyers and sellers. 2. Xlabel, Ylabel, Title and Grid is included 3. Display the plot using plt.show()   plt.bar(['Seller Pays', 'Buyer Pays'], [median\_price\_seller\_pays, median\_price\_buyer\_pays], color=['blue', 'green'])  plt.xlabel('Shipping Category')  plt.ylabel('Median Price')  plt.title('Median Price Comparison: Seller Pays vs. Buyer Pays')  plt.grid(True)  # Show the plot  plt.show()   1. The price difference is calculated by subtracting the median price buyer pays and the median price seller pays   price\_difference = median\_price\_buyer\_pays - median\_price\_seller\_pays   1. Results are reported using the print function and it displays the median price that seller pays and buyer pays   print(f'On comparison of when the price is higher between seller and buyer pays for shipping:\n')  print(f'The median price when seller pays is', median\_price\_seller\_pays)  print(f'The median price when buyer pays is', median\_price\_buyer\_pays)   1. Control flow is used and in this case If function is used to compare the price and display the result   if price\_difference < 0:  print("On average, items paid by sellers have a higher median price.")  elif price\_difference > 0:  print("On average, items paid by buyers have a higher median price.")  else:  print("There is no significant difference in the median prices.") |

--------------------------------------------End of Ramchandar----------------------------------------------------------------------------------

## Question 1. 4

|  |
| --- |
| You are required to find out the item condition information from the data. Lower the number (value), the better condition of the item. • Write the code to find out (print) the count of the rows on each number (value) in column item\_condition\_id. • Draw the boxplot graphs (one plot) on the price for each item condition value, and find out out whether the better condition of the item could have higher median price (draw the plot and answer this question in the comment and report). |

## Answer 1. 4 First Part

|  |
| --- |
|  |

## Output

|  |
| --- |
|  |

## Explanation

|  |
| --- |
|  |

--------------------------------------------------------------------------------------------------------------------------------------------------------

# Part II Time series analysis exercise

# Group

|  |  |  |  |
| --- | --- | --- | --- |
| Author Name | Deakin ID | Questions Answered | Total |
| Ramchandar Mariappan | 223914532 | 2.1 | 1 |
| Simranjit Singh |  | 1.4 to 1.7 | 4 |
| Uthara Ravichanthar |  | 1.8 to 1.10 | 4 |

## Question 2.1

|  |
| --- |
| The dataset used here is the New York City Taxi Demand dataset. The raw data is from the NYC Taxi and Limousine Commission. The data included here consists of aggregating the total number of taxi passengers into 30 minute buckets. In this question, we will simply process the data and explore the time series. • Create two new dataframes df\_day and df\_hour by aggregating the demand value on daily and hourly level. • Plot the demand value in two line charts for both df\_day and df\_hour dataframes. • Plot the seasonal decomposition components (Trend, Seasonal, Residual) from df\_day dataframe, also find out the p value from adfuller test. Do you think the df\_day is stationary enough (please explain your reasons in comments and report)? |

## Answer 2.1 First Part

|  |
| --- |
| Create two new dataframes df\_day and df\_hour by aggregating the demand value on daily and hourly level. Plot the demand value in two line charts for both df\_day and df\_hour dataframes  TS\_df['timestamp'] = pd.to\_datetime(TS\_df['timestamp'])  # Create a DataFrame for daily aggregation  df\_day = TS\_df.resample('D', on='timestamp').sum().reset\_index()  # Create a DataFrame for hourly aggregation  df\_hour = TS\_df.resample('H', on='timestamp').sum().reset\_index()  # Set figure size  plt.figure(figsize=(15, 6))  # Plot the demand value on a daily level  plt.subplot(2, 1, 1)  sns.lineplot(data=df\_day, x='timestamp', y='value')  plt.title('Daily Demand Value')  plt.xlabel('Date')  plt.ylabel('Demand Value')  # Plot the demand value on an hourly level  plt.subplot(2, 1, 2)  sns.lineplot(data=df\_hour, x='timestamp', y='value')  plt.title('Hourly Demand Value')  plt.xlabel('Time')  plt.ylabel('Demand Value')  # Adjust spacing between subplots  plt.tight\_layout()  # Show the plots  plt.show() |

## Output

|  |
| --- |
| Learning the data pattern for the demand vs date - Clearly there is a seasonality existing in days and weeks. Also, there is a trend seen in this data with increase from 2014-07 to 2014-11. then decreasing from 2014-11 to 2015-02  Learning the data pattern for the hourly demand, it is clear there is a spike in hours and that could show the seasonality or a peak time where there is high demand for taxi and then there is a drop. This is common and there are lot of examples in this case we can think of, for example morning hours could be peak because there will people commuting to work by taxis. |

## Explanation

|  |
| --- |
| 1. The Data is stored in a dataframe TS\_df and the column timestamp is in the data type format object. The code using pandas pd to datetime converts the timestamp feature to datetime data type   TS\_df['timestamp'] = pd.to\_datetime(TS\_df['timestamp'])   1. Now the new column df\_day is created to store the result of the dataframe. The TS\_df is the existing dataframe containing the time series data and that will be resampled. Resampling is done on the TS\_df dataframe on the daily frequency data ‘D’and the on parameter specifies the column in the TS\_df dataframe that contains column timestamp. The sum is then used to sum the value of the daily frequency data. When resample and sum like functions are used, the data the index of the dataframe will change and reset index helps to reset the data. Similar to the daily frequency data, hourly frequency is used as well   # Create a DataFrame for daily aggregation  df\_day = TS\_df.resampl('D', on='timestamp').sum().reset\_index()  # Create a DataFrame for hourly aggregation  df\_hour = TS\_df.resample('H', on='timestamp').sum().reset\_index()   1. The plt from matplotlib is used to create a figure of figsize(15,6)   # Set figure size  plt.figure(figsize=(15, 6))   1. The plt from matplotlib is used to create a subplot to have two figures one for day and the hour. The sns from seaborn is used to create a line plot with data as df\_day created already with x axis = time stamp and y axis on the values. The title is given with plt.title and similarly the xlables and y labels are named. The same method is used for hourly level data as well   # Plot the demand value on a daily level  plt.subplot(2, 1, 1)  sns.lineplot(data=df\_day, x='timestamp', y='value')  plt.title('Daily Demand Value')  plt.xlabel('Date')  plt.ylabel('Demand Value')  # Plot the demand value on an hourly level  plt.subplot(2, 1, 2)  sns.lineplot(data=df\_hour, x='timestamp', y='value')  plt.title('Hourly Demand Value')  plt.xlabel('Time')  plt.ylabel('Demand Value')   1. Finally the layout is adjusted and displayed using plt.show()   # Adjust spacing between subplots  plt.tight\_layout()  # Show the plot  plt.show() |

## Answer 2.2 Second Part

|  |
| --- |
| Plot the seasonal decomposition components (Trend, Seasonal, Residual) from df\_day dataframe, also find out the p value from adfuller test. Do you think the df\_day is stationary enough (please explain your reasons in comments and report)?  import statsmodels.api as sm  from statsmodels.tsa.stattools import adfuller  # Set the frequency of the DataFrame index to 'D' (daily)  df\_day['timestamp'] = pd.to\_datetime(df\_day['timestamp'])  # Set 'timestamp' as the index of the DataFrame  df\_day.set\_index('timestamp', inplace=True)  # Perform seasonal decomposition using additive decomposition with daily frequency  decomposition = sm.tsa.seasonal\_decompose(df\_day['value'], model='additive')  # Plot the components  plt.figure(figsize=(12, 8))  # Original time series  plt.subplot(4, 1, 1)  plt.plot(df\_day.index, df\_day['value'], label='Original')  plt.title('Original Time Series')  plt.xlabel('Date')  plt.ylabel('Demand Value')  plt.legend()  # Trend component  plt.subplot(4, 1, 2)  plt.plot(decomposition.trend, label='Trend')  plt.title('Trend Component')  plt.xlabel('Date')  plt.ylabel('Trend')  plt.legend()  # Seasonal component  plt.subplot(4, 1, 3)  plt.plot(decomposition.seasonal, label='Seasonal')  plt.title('Seasonal Component')  plt.xlabel('Date')  plt.ylabel('Seasonal')  plt.legend()  # Residual component  plt.subplot(4, 1, 4)  plt.plot(decomposition.resid, label='Residual')  plt.title('Residual Component')  plt.xlabel('Date')  plt.ylabel('Residual')  plt.legend()  # Adjust spacing between subplots  plt.tight\_layout()  # Show the plots  plt.show()  # Perform ADF test to check stationarity on the residual component  adf\_result = adfuller(decomposition.resid.dropna(), autolag='AIC')  p\_value = adf\_result[1]  # Interpret the ADF test result  if p\_value <= 0.05:      stationarity\_result = "The residual series is likely stationary (p-value <= 0.05)"  else:      stationarity\_result = "The residual series is likely non-stationary (p-value > 0.05)"  print(f"ADF Test p-value: {p\_value}")  print(stationarity\_result) |

## Output

|  |
| --- |
| In this case, as the seasonal component is constant over time ie in days, hours where there is a spike and then a drop , we have chosen additive decomposition method to show all the components  Trend: There is little bit of trend seen, with decreasing from 2014 -07 to 2014-09, increasing from 2014-09 to 2014-12  Seasonality: Clear seasonality seen every days/week and it is very consistent pattern. This regular pattern indicates that demand follows a predictable cycle.  Residual: The residual component shows the errors or variations that occur every month. These errors become exposed and repeat in a consistent manner, It signifies there are specific factors influencing demand on a monthly basis.  ADF Test p-value: 8.312345605115968e-12  The residual series is likely stationary (p-value <= 0.05)  **Reasons:**  Adfuller test signifies the stationarity based on the hypothesis Null - Hypothesis : The series is not stationary ( P value >0.05 at 95% confidence)  Reject Null Hypothesis : The series is stationary ( Pvalue <=0.05 at 95% confidence)  On the above adfuller testing, the p value is 8.312345605115968e-12 and it is less than 0.05, therefore we need to reject the null hypothesis and prove that series is stationary |

## Explanation

|  |
| --- |
| 1. Importing the required variables for the stationarity testing. sm is imported from statsmodels.api. Adfuller is imported again from statsmoels.tsa.stattools.   import statsmodels.api as sm  from statsmodels.tsa.stattools import adfuller   1. Though the df\_day is already in the datetime datatype, double checking. The df\_day data frame is converted to datetime object type using pandas pd to datetime. Also the timestamp is set to the index using df\_day.set\_index with inplace=True means with in the same dataframe make the changes   # Set the frequency of the DataFrame index to 'D' (daily)  df\_day['timestamp'] = pd.to\_datetime(df\_day['timestamp'])  # Set 'timestamp' as the index of the DataFrame  df\_day.set\_index('timestamp', inplace=True)   1. Now the seasonal decomposition is made on the df\_day dataframe using the sm function imported from statsmodels and using the seasonal decompose. Here the model is chosen as additive and the values are used to perform the decomposition   # Perform seasonal decomposition using additive decomposition with daily frequency  decomposition = sm.tsa.seasonal\_decompose(df\_day['value'], model='additive')   1. The figure is then plotted using plt imported from matplotlib and with the figsize of (12,8)   # Plot the components  plt.figure(figsize=(12, 8))   1. Then the plot is carried out. The subplot is created to separate the figure in to 4 different parts.The title, xlabel, ylabel and the legend particulars are includes on the plot. The first subplot is depicting the original dataframe values with df\_day ie with out any decomposition. The second subplot includes the trend component using decomposition.trend . Third subplot include the decomposition.seasonal which decomposes the seasonal component. Finally the last subplot exposes the residual in the time series data.   # Original time series  plt.subplot(4, 1, 1)  plt.plot(df\_day.index, df\_day['value'], label='Original')  plt.title('Original Time Series')  plt.xlabel('Date')  plt.ylabel('Demand Value')  plt.legend()  # Trend component  plt.subplot(4, 1, 2)  plt.plot(decomposition.trend, label='Trend')  plt.title('Trend Component')  plt.xlabel('Date')  plt.ylabel('Trend')  plt.legend()  # Seasonal component  plt.subplot(4, 1, 3)  plt.plot(decomposition.seasonal, label='Seasonal')  plt.title('Seasonal Component')  plt.xlabel('Date')  plt.ylabel('Seasonal')  plt.legend()  # Residual component  plt.subplot(4, 1, 4)  plt.plot(decomposition.resid, label='Residual')  plt.title('Residual Component')  plt.xlabel('Date')  plt.ylabel('Residual')  plt.legend()   1. The layout is adjusted and the plt.show()is used to diplay the output plots.   # Adjust spacing between subplots  plt.tight\_layout()  # Show the plots  plt.show()   1. As a next step to identify stationarity , the ADF test is performed with the autolag AIC. The adfuller imported from the statsmodels is used to test the stationarity on the residual component. Then the result obtained is used to generate the P value   # Perform ADF test to check stationarity on the residual component  adf\_result = adfuller(decomposition.resid.dropna(), autolag='AIC')  p\_value = adf\_result[1]   1. The flow statement is used to do the validation on the P value and print the result as a description that is quite visible on the stationarity results. The Pvalue is compared for 0.05 ie 95% confidence interval to validate the stationarity test. Finally based on the if else statement the results are printed with P value.   # Interpret the ADF test result  if p\_value <= 0.05:      stationarity\_result = "The residual series is likely stationary (p-value <= 0.05)"  else:      stationarity\_result = "The residual series is likely non-stationary (p-value > 0.05)"  print(f"ADF Test p-value: {p\_value}")  print(stationarity\_result) |

|  |
| --- |
| 1)Why you decide to choose your solution; - In the context of this question, will take this in to three parts.  Part 1:  I chose to create two new DataFrames, df\_day and df\_hour, to aggregate the demand value on daily and hourly levels because this approach allows for easy visualization and analysis of the time series data at different frequencies. Resampling the data to daily and hourly intervals provides a way to observe trends and patterns at different time scales.  Part 2:  I used line charts to visualize the demand values over time because line charts are suitable for showing how values change over continuous time intervals, which is common in time series analysis.  Part 3:  Also, the additive model is used to check the decomposition of trend, seasonality and residual. The reason for going to additive model as a first step because the magnitude of the seasonal component is relatively constant or linear across the different time points. Also I chosen the Augmented Dickey-Fuller (ADF) test on the residual component to check for stationarity. This solution is chosen because it helps to understand the underlying trends, seasonality, and stationarity of the time series data, which is crucial for time series analysis.  2)Are there any other solutions that could solve the question;  Part 1 & Part 2  There are alternative ways to aggregate and visualize the data. For example, other aggregation functions besides. sum () (e.g., .mean() or .max()) depending on the specific analysis goals. Additionally, we can use different plotting libraries or customize the visualizations further.  Part 3  Alternative decomposition methods (e.g., multiplicative decomposition) could be used based on the characteristics of the time series data. Different statistical tests for stationarity, such as the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test, could be employed as an alternative to the ADF test.  3)Whether your solution is the optimal or not? why?  The chosen solution is a good starting point for analyzing time series data. It's clear, easy to understand, and covers the necessary steps for data preprocessing and analysis. The solution is optimal because it identifies the stationarity of the timeseries analysis and it is very important for predicting the future forecast. Further Optimization is not required in this data set as the required ADF testing helped to prove on the stationarity of the time series with the help of the P value and the hypothesis judgement. Though we can call this as optimal in this context, on a broader context Optimization might involve more sophisticated modeling techniques, parameter tuning, or considering external factors that could impact the time series which are not considered . |

--------------------------------------------End of Ramchandar----------------------------------------------------------------------------------

----------------------------------------------------------------------END----------------------------------------------------------------------

THANK YOU