Deakin University

SIT724 – Group Task Submission

End Term Assessment – Part 1 and Part 2

Submitted by

Prince Park, Suraj Mathew, Shailesh Pande

Group 23

Date 8TH October 2023

Contents

Read the Data

	train_id	name	item_condition_id	category_name	brand_name	price	shipping	clean_description
0	128037	Bundle for Sassy Sisters	3	Women/Tops & Blouses/Blouse	NaN	16.0	0	max cleo black dress paper crane black tank to...
1	491755	PINK VS TANK	2	Women/Tops & Blouses/Tank, Cami	NaN	17.0	0	sequin pink sign sequins missing gently worn
2	470924	Funko Pop Unmasked Cyclops	1	Kids/Toys/Action Figures & Statues	Funko	30.0	1	box great condition comes soft pop protector p...
3	491263	Baby Roshe Runs	3	Kids/Boys 2T-5T/Shoes	Nike	19.0	0	baby black nike roshe runs size 5c
4	836489	Baby Girl Ralph Lauren dresses	3	Kids/Girls 0-24 Mos/Dresses	Ralph Lauren	24.0	0	2 polo dresses 3 months wore washed dreft pink...
...	...	...	...	...	...	...	...	...
355803	760377	Beats By Dre Solo White	3	Electronics/TV, Audio & Surveillance/Headphones	Beats	45.0	1	beats dre solo white gently used work great
355804	780889	4 New Leap Frog Leapster Learning Games	1	Kids/Toys/Learning & Education	Leap Frog	9.0	1	viewing 4 new leap frog leapster learning game...
355805	650579	Torrid bra size 42ddd	3	Women/Underwear/Bras	Torrid	20.0	1	couple places lace snagged tell fairly good co...
355806	481154	Vans shoes	2	Men/Shoes/Fashion Sneakers	VANS	23.0	0	size 11
355807	361073	Kendra Scott Alex earrings in Magenta	2	Women/Jewelry/Earrings	Kendra Scott	38.0	1	description yet

Table 1: Data as provided

- EDA - Basic analysis of the loaded data frame to assess the structure , columns etc.
- Dimensions of the data provided
  - *The number of rows (observations) is 355808*
  - *The number of columns (variables) is 8*
- Check for Duplicates and remove duplicates
  - *The number of rows duplicated are : 48572*
  - *Duplicated rows are dropped from the data set.*
  - *Number of observations in the data after removing duplicated rows are : 307236*
- Information on features provided in the data

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 307236 entries, 0 to 355807
Data columns (total 8 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   train_id           307236 non-null  int64
 1   name               307236 non-null  object
 2   item_condition_id  307236 non-null  int64
 3   category_name      305911 non-null  object
 4   brand_name         175941 non-null  object
 5   price              307236 non-null  float64
 6   shipping           307236 non-null  int64
 7   clean_description  307070 non-null  object
dtypes: float64(1), int64(3), object(4)
memory usage: 21.1+ MB
```

  - *There are missing values in columns 'category_name' , 'brand_name' , 'clean_description '*
  - *'train_id' , item_condition_id' and 'shipping' features are in numeric data type. These are id's only and should be read as 'object type'*

➢ Convert data type of features

```
'''
Convert train_id,item_condition_id,shipping to object type
'''
#make a list of columns names where the datatype of the columns need to be changed
object = ['train_id','item_condition_id','shipping']

#Using for loop to iterate over the list and change the datatype
for i in object:
    df[i] = df[i].astype('object')
```

Code 1: Code to change datatypes of features

The data type of following features

- *'train_id'*
- *'item_condition_id'*
- *'shipping']*

is converted to 'object' data type

```
# Getting the 5 Point Summary of the only numeric datatype in the dataframe (Price)

# get the details in a data frame ,numbers are rounded to two places of decimals
df['price'].describe().to_frame().round(2)
```

Code 2: Code to change data type of a feature

➢ Five-point statistical summary on ' price '

	price
count	307236
mean	27
std	38
min	0
25%	10
50%	17
75%	29
max	2000

Table 2:Five Point Statistical Summary on 'price'

```
# Number of observations where price is zero
print('The number of transactions where price is zero:', len(df[df['price']==0]))
```

Code 3: Code for ascertaining the number of transaction with price = 0

- *The number of transactions where price is zero: 169*

➢ Analysis of items with price zero

```
# slice the data for price == 0 ,and construct a pivot to count the number of transactions
# for each 'item_condition_id' which has price = 0.

a = pd.pivot_table(data    =  df[df['price']==0],
                   index   = ['item_condition_id','price'],
                   values  = 'train_id',
                   aggfunc = 'count',
                   margins = 'Total',
                   margins_name= 'Total')
display(a)
```

Code 4: Code for Price zero pivot

		train_id
item_condition_id	price	
1	0.0	57
2	0.0	62
3	0.0	48
4	0.0	2
Total		169

Table 3: Price Zero Pivot

- *There are 57 transactions which belong to item_condition_id = 1 , which is the best quality, where price is zero.*
- *Data provided could be a catalog or transaction table. In both cases, it is unlikely that any item will have zero price. We could group the price by categories and ingest a measure of central tendency.*
- *We decided to remove those observations that have price zero.*

```
# create data frame by considering only those data points where price in not equal to zero
df = df[df['price'] != 0]
```

Code 5: Code to filter out observations with price = 0

- *The number of rows(observations) after above operation: 307067*

➢ Description of 'Object' data type features provided in the data

	count	unique	top	freq
train_id	307236	307236	128037	1
name	307236	277067	Bundle	446
item_condition_id	307236	5	1	132492
category_name	305911	1135	Women/Athletic Apparel/Pants, Tights, Leggings	12364
brand_name	175941	3046	PINK	11438
shipping	307236	2	0	170057
clean_description	307070	267826	description yet	17104

Table 4: Description of 'object' data type features

- *Observation*
  - *There are 276k unique values in 'name'*
  - *There are 5 unique values in 'item_condition_id'*
  - *There are 1135 unique values in 'category_name'*
  - *There are 3045 unique values in 'brand_name'*
  - *There are 2 unique values in 'shipping' paying status*
  - *There are 267k unique descriptions under 'clean_description'*
- *We will need to study further the various 'object' type features*
- *We will define a function which will give us the value counts in numbers and percentage and also a visual representation of the same for object type features in a data frame*

```
'''
Define a function which will give the value counts of a categorical feature in numbers and percentage and a count plot
for visualization with a vertical / horizontal orientation , depending on the count of uniwue values of the feature
'''
def value_count_info(df, feature):
    a = df[feature].value_counts()  # find value counts in numbers
    b = round(df[feature].value_counts(normalize=True) * 100, 2)  # find value counts in %

    # create a datframe to display the value counts both in numbers and in percentage
    value_count_df = pd.DataFrame({'count': a,'Percentage': b})

    # rename the index column as the 'feature'
    value_count_df = value_count_df.rename_axis(feature)

    # Format the 'Percentage' column to show as '%'
    value_count_df['Percentage'] = value_count_df['Percentage'].apply(lambda x: f"{x:.2f}%")

    # Determine the pivot display options and the plot orientation based on the number of unique values

    if len(a) > 5:
        # if unique values in feature > 5

        print('\n')   # create a space line
        print('Value Count Analysis of :',feature)

        display(value_count_df.head()) # display only top 5 rows of the pivot

        plt.figure(figsize=(20, 5))  # Adjust the figure size for horizontal plot
        g = sns.barplot(data=value_count_df.head(),  # make a bar plot for only top 5 rows of the pivot
                        x=value_count_df.head()['count'],
                        y=value_count_df.head().index,
                        orient='h')  # Horizontal orientation of bar plot if unique values are more than 5
        for i in g.containers:
            g.bar_label(i, fmt='%d')  # annotate the plot and display counts as integers
        plt.xlabel('Count',fontweight='bold')           # display the x-label
        plt.ylabel(feature,fontweight='bold')  # Set the ylabel as the feature name
        plt.title('Bar plot showing Value Counts of top 5 : ' + feature,fontweight='bold')

    else:   # if unique values of the feature are less than 5

        print('\n')
        print('Value Count Analysis of :',feature)

        display(value_count_df) #  display the complete pivot
        plt.figure(figsize=(20, 5))  # Adjust the figure size for vertical plot
        g = sns.barplot(data=value_count_df,  # plot the vertical bar plot
                        x=value_count_df.index,
                        y=value_count_df['count'])
        for i in g.containers:
            g.bar_label(i, fmt='%d')  # Display counts as integers
        plt.xlabel(feature,fontweight='bold')
        plt.ylabel('Count',fontweight='bold')  # Set the ylabel as 'Count'
        plt.title('Bar plot showing Value Counts of : ' + feature,fontweight='bold')

    plt.show()

    return
```

Code 6: Code for value count and graph

- *We create a list of features on which we seek to get more information and iterate through a for loop code with the above defined function*

```
obj_col = ['item_condition_id','shipping','category_name','brand_name','name','clean_description']

for i in obj_col:
    value_count_info(df,i)

```

Code 7: Code to run the value count function in a loop

➢ Value Count Analysis of 'item_condition_id'

```
Value Count Analysis of : item_condition_id
```

item_condition_id	count	Percentage
1	132435	43.13%
3	89856	29.26%
2	77604	25.27%
4	6703	2.18%
5	469	0.15%

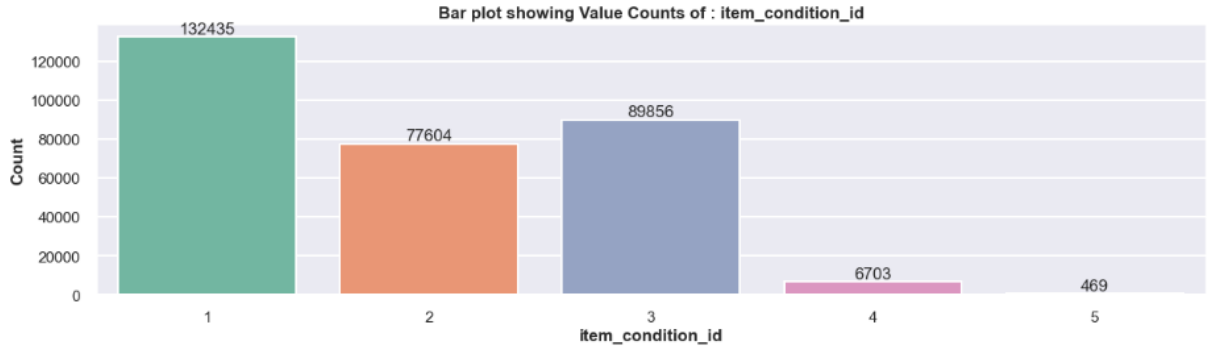

Table 5: Value Count Analysis of 'item_condition_id"

➢ Value Count Analysis of 'shipping

```
Value Count Analysis of : shipping
```

shipping	count	Percentage
0	169954	55.35%
1	137113	44.65%

Table 6: Value Count Analysis of 'shipping'

➢ Value Count Analysis of ‘category_name’

```
Value Count Analysis of : category_name
```

category_name	count	Percentage
Women/Athletic Apparel/Pants, Tights, Leggings	12360	4.04%
Women/Tops & Blouses/T-Shirts	9651	3.16%
Beauty/Makeup/Face	7137	2.33%
Beauty/Makeup/Lips	6200	2.03%
Electronics/Video Games & Consoles/Games	5531	1.81%

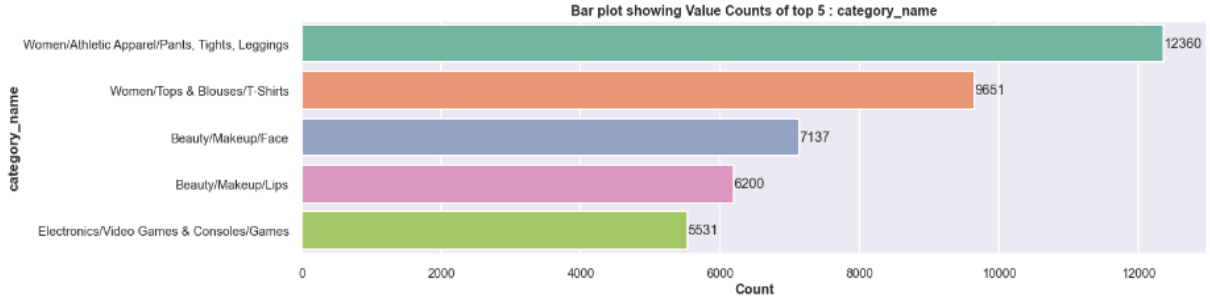

Table 7: Value Count Analysis of ‘category_name’

➢ Value Count Analysis of ‘brand_name’

Value Count Analysis of : brand_name

brand_name	count	Percentage
PINK	11433	6.50%
Nike	11322	6.44%
Victoria's Secret	9955	5.66%
LuLaRoe	6322	3.60%
Apple	3430	1.95%

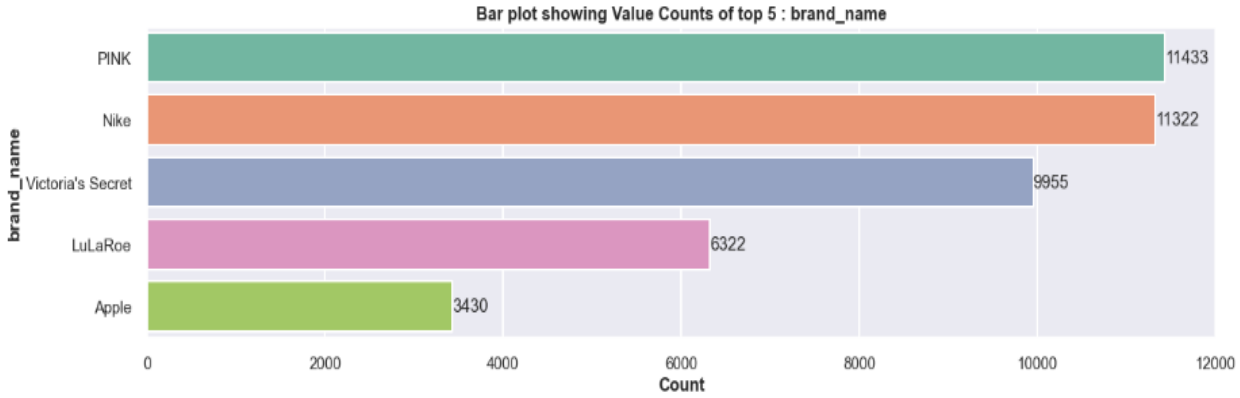

Table 8: Value Count Analysis of 'brand_name'

- Value Count Analysis of 'name'

Value Count Analysis of : name

name	count	Percentage
Bundle	446	0.15%
BUNDLE	94	0.03%
Reserved	94	0.03%
Converse	93	0.03%
Romper	91	0.03%

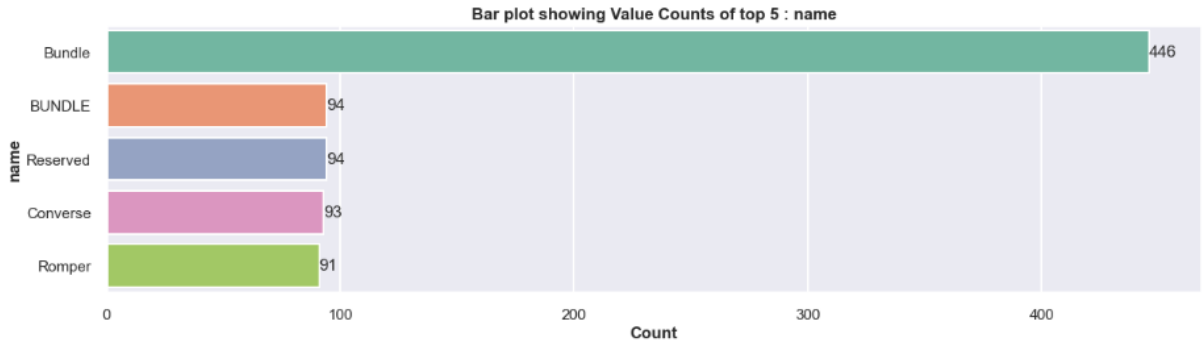

Table 9: Value Count Analysis of 'name'

➢ Value Count Analysis of 'clean_description'

Value Count Analysis of : clean_description

clean_description	count	Percentage
description yet	17095	5.57%
brand new	1107	0.36%
new	1095	0.36%
good condition	606	0.20%
great condition	478	0.16%

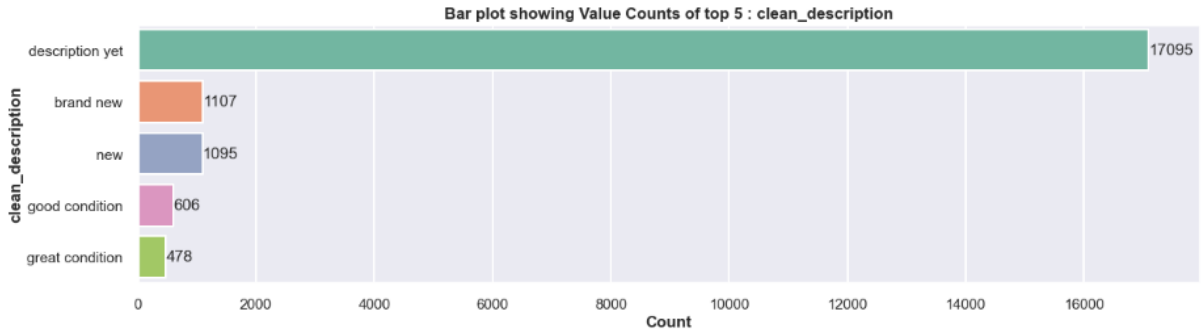

Table 10: Value Count Analysis of 'clean_description'

- *Observations*
  - *Bundle and BUNDLE in the feature 'name' is a repetition*
  - *Description under 'clean_description' have string values 'brand new' and 'new', 'good condition' , 'great condition', which are ambiguous*
  - *Since the count of such discrepancies is insignificant , we do not make any changes*
  - *Information on other features as displayed above is self-explanatory*

Question 1.1

Find the missing values:

➢ Write the function **missing_values_table** which will find the missing values in the data frame

```
'''
• 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);
'''
def missing_values_table(df):
    a = df.isnull().sum()  # find missing values if any in all features

    # filter the information to see only those features where missing values is more than 0
    a = a[a>0]
    a = a.to_frame()       # construct a data frame
    a = a.rename(columns={0:'Missing_Value_Count'})  # rename the column
    a = a.rename_axis('Features with Missing Values') # rename the index column
    a = a.sort_values(by='Missing_Value_Count',ascending=False) # sort from high to low

    # add a col which shows % of missing data
    a['% Missing Values'] = round(a['Missing_Value_Count'] / df.shape[0]*100,2)
    a
    return(a)
```

Code 8: Function for finding missing values in features in a data frame

- *The output of the above defined function will display all the features with missing values in ascending order along with the percentage.*

```
# use the function defined above to ascertain the missing values in the given data set
missing_values_table(df)
```

Features with Missing Values	Missing_Value_Count	% Missing Values
brand_name	131227	42.74
category_name	1323	0.43
clean_description	166	0.05

Table 11: Missing Values in Features

- Can the Missing Values be imputed by the means of the respective column
  - *Code to find the data type of the features that have 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)
'''
# get the names of the features which have missing values in a list
index_list = missing_values_table(df).index.tolist()
index_list
```

```
['brand_name', 'category_name', 'clean_description']
```

Code 9: Code to find features with missing values

```
# ascertain the data type of the features which have missing values

df[index_list].dtypes
```

```
brand_name           object
category_name        object
clean_description    object
dtype: object
```

- *There are 3 columns that have missing values [brand_name, category_name, clean_description].*
- *All the 3 columns are of object/string data type.*
- *The mean value cannot be imputed as the categories here are vast and not limited. These columns cannot be converted into their numerical equivalent through label encoding. Hence the measure of central tendency (mean) cannot be used over here.*
- *Using the mode is a possibility, but then it could change the meaning of the dataset if 42% of the brand name column needs to be imputed. For the items that have an item category, we can impute the brand name as "Others".*

Question 1.2

Find the price information from the data

➢ Write code to print the median price of the items in the data

```
'''
• Write code to print the median price of the items in the data;
'''

print('\n')
median_price = df['price'].median() # find the median price of the column 'price'

# Printing the median value of the the price column for the entire dataset
print ("The median value of the Price Column is:", median_price)
print('\n');


# visualize the median price
plt.figure(figsize = (15,3));
g2= sns.boxplot(df.price,
            showfliers=False, # will not display the outliers
            width=0.5,
            orient ='h');
plt.axvline(x=median_price, c='red'); #draw a red colured vertical line at the median

plt.text(median_price, -0.55, f'Median Price: {median_price}', color='red',
         fontsize=15, ha='center') # add text to a plot

plt.show(g2)
```

Code 10: Code to find the median price

- The median price

```
The median value of the Price Column is: 17.0
```

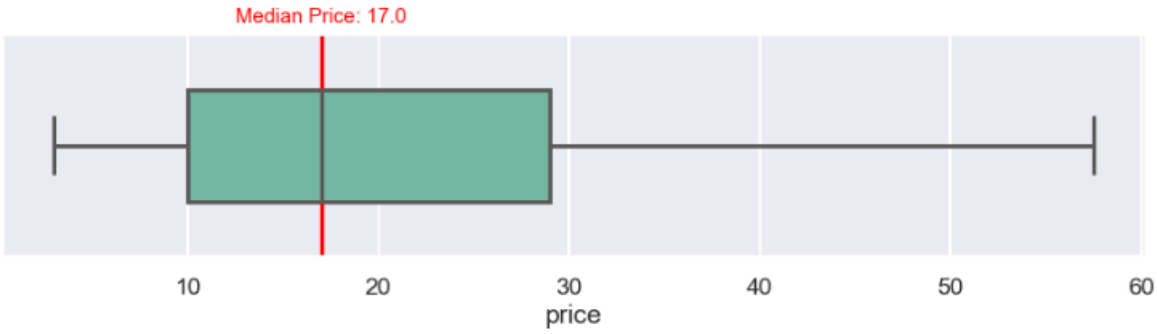

Figure 1: Median Price of all items

- 90th percentile value on the price

```
'''
• What is the 90th percentile value on the price;
'''

# find the 90th percentile
price_90percentile = df['price'].quantile(q=90/100)
print('\n');
print("The 90th percentile of the Price Column is:", price_90percentile)
print('\n');


# visualize the 90th percentile price
plt.figure(figsize = (15,3));
g3 = sns.boxplot(df.price,
            showfliers=False, # will not display the outliers
            width=0.5,
            orient ='h')

# draw a red colured vertical line at the 90th Percentile
plt.axvline(x=price_90percentile, c='red')
plt.text(price_90percentile, -0.55, f'90th Percentile: {price_90percentile}',
         color='red', fontsize=15, ha='center')

plt.show(g3)
```

Code 11: Code to find the 90th percentile of all prices

```
The 90th percentile of the Price Column is: 51.0
```

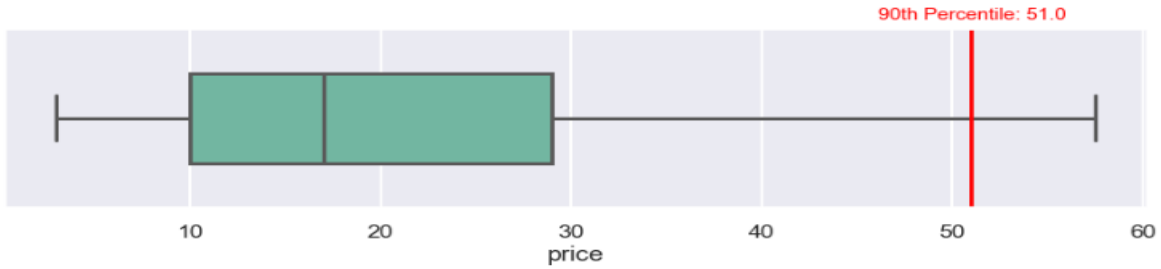

Figure 2:Boxplot showing the 90th percentile price

- Draw the histogram chart for the price of the items in the data with 50 bins.

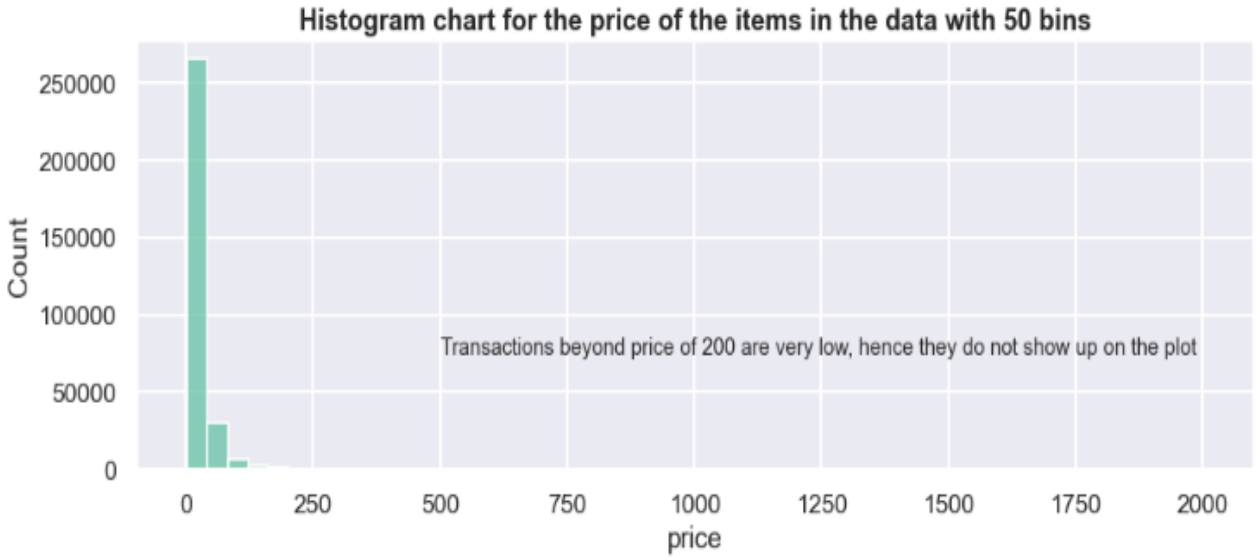

Figure 3: Histogram of price with 50 bins

- *The histogram does not show beyond 4 bins because number of transactions in bins beyond 200 are very low relative to the first few bins.*

- *To visualize the spread of transactions across the bins better, we slice the data frame into different price ranges as under and view the Histograms*

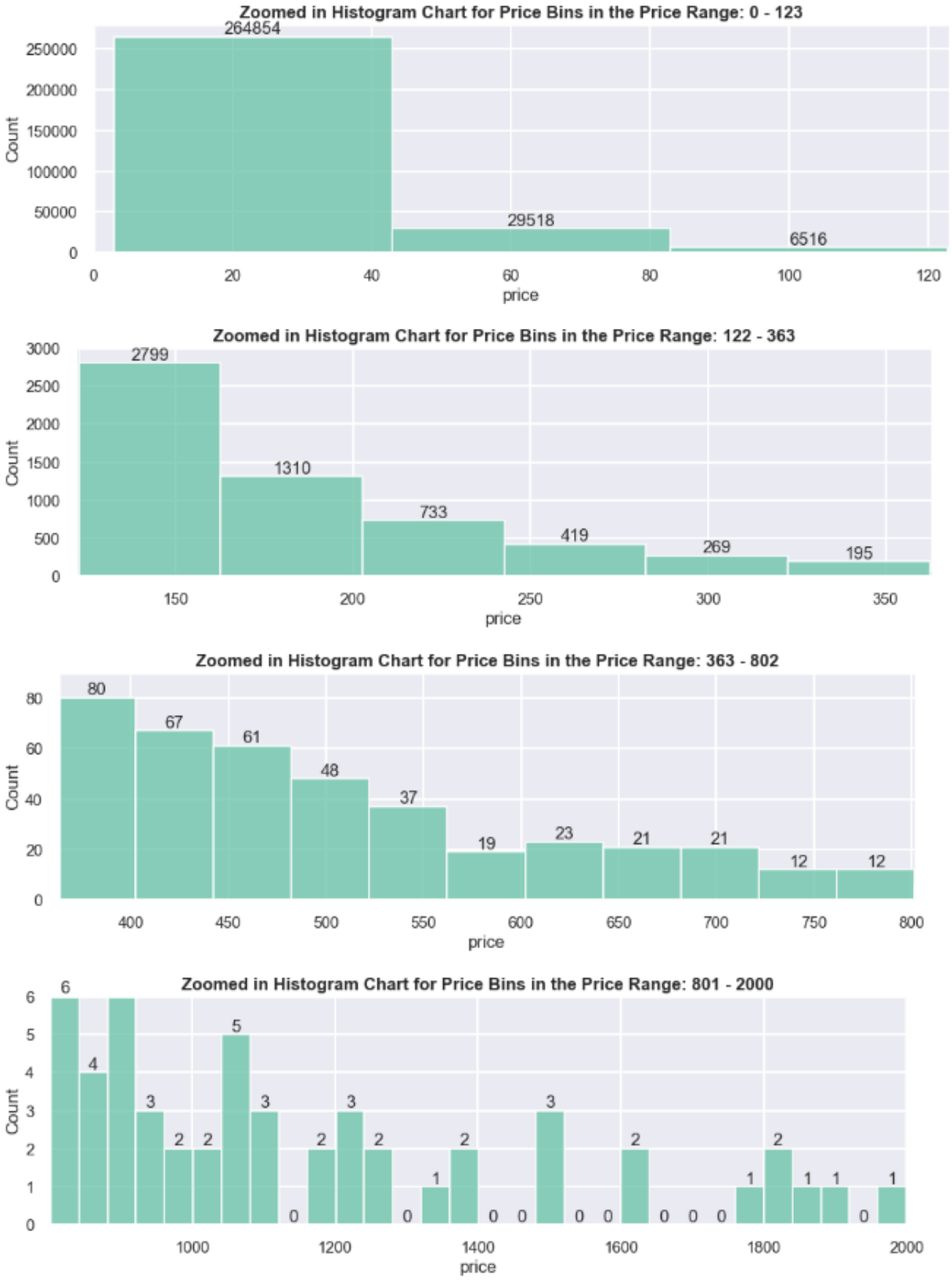

Figure 4: Histogram of price bins at different price ranges

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.

  - *Shipping status 0 implies : Shipper has paid*
  - *Shipping status 1 implies : Buyer has paid*
  - *Code to find the percentage of the items that are paid by the buyers*
  - *We have defined a function earlier 'value_count_info' which would give us the output required, but we write the code again to map the numeric values of shipping status with their actual meaning*

```
'''
• Write code to find out the percentage of the items that are paid by the buyers.
'''
# shipping : 0 implies that seller has paid
# shipping : 1 implies that buyer has paid

# we have defined a function to do this, however in this code we map the 0 and 1 under 'shipping' to their actual meaning

# find the value counts in mumbers
shipping_count=df.shipping.value_counts()

# find the value counts in percentage
shipping_percent=df.shipping.value_counts(normalize=True)*100

# make a dataframe on shipping info
shipping_info_df = pd.DataFrame({'Count':shipping_count,
                                 'Percentage':shipping_percent})

shipping_info_df.rename_axis('Shipping',inplace = True)

shipping_mapping = {0: 'Seller Paid', 1: 'Buyer Paid'}  # map the shipping charges according to the numeric codes

shipping_info_df.index = shipping_info_df.index.map(shipping_mapping)  # replace 0 and 1

# display the "Percentage" with '%' sign
shipping_info_df['Percentage'] = shipping_info_df['Percentage'].apply(lambda x: f"{x:.2f}%")

display(shipping_info_df)

plt.figure(figsize = (15,5));
sns.set_theme('poster',font_scale=0.7,palette='Set2')
g9 = sns.barplot(data = shipping_info_df,
                 x = shipping_info_df.index,
                 y = shipping_info_df['Count'])
for i in g9.containers: g9.bar_label(i,)
plt.title('Shipping Charges Information')
plt.show(g9)

```

Code 12: Code to find percentage of transactions paid by biller / shipper

➢ Shipping Charges : Value Count Analysis

Shipping	Count	Percentage
Seller Paid	169954	55.35%
Buyer Paid	137113	44.65%

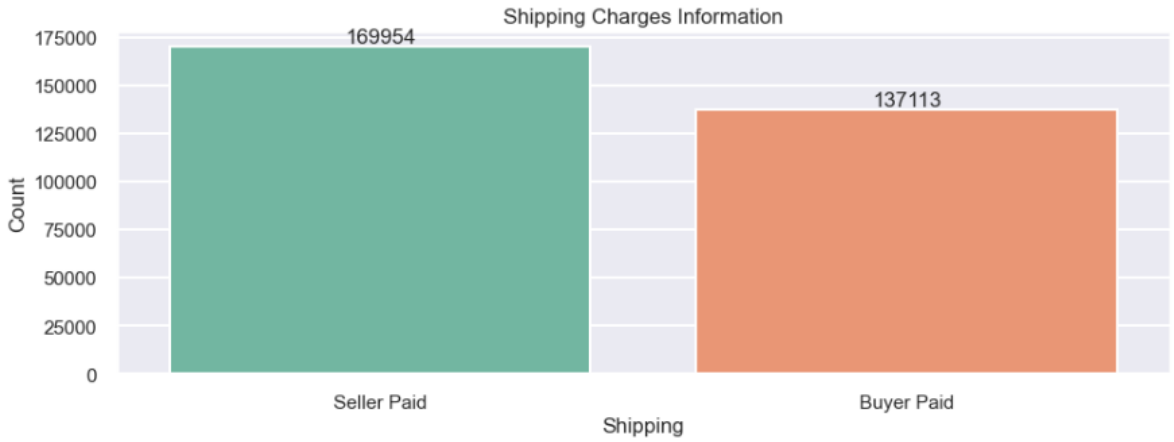

Figure 5:Shipping Payment Status visualization

- *Seller pays 55 % of the times and Buyer pays 45 % of the times.*

➢ Draw (two) histogram graphs in one plot on the price for seller pays shipping and buyer pays shipping (50 bins).

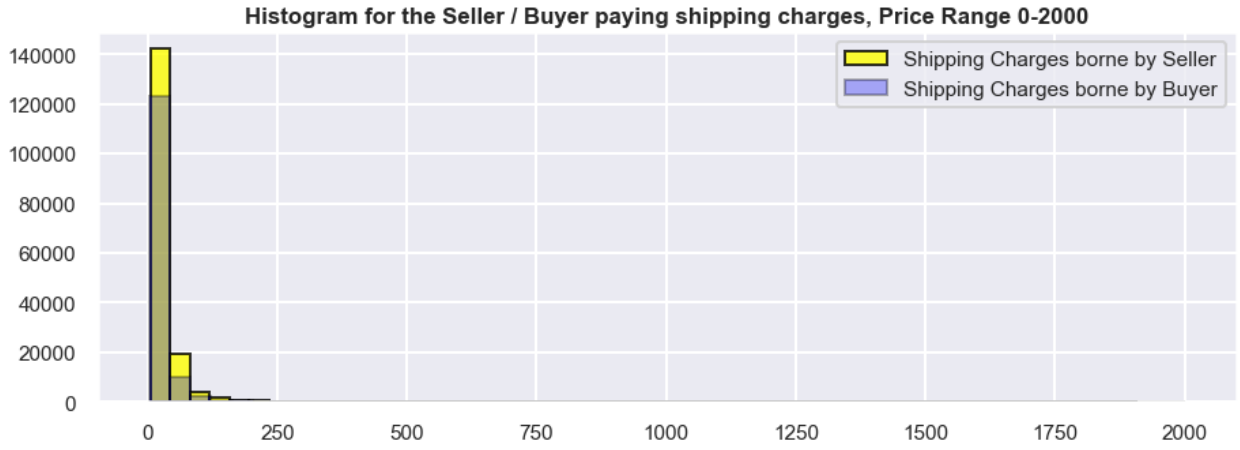

Figure 6: Histogram showing Shipping status

- *Histogram graphs for shipping payment status after slicing the data frame on different price range for better visualization*

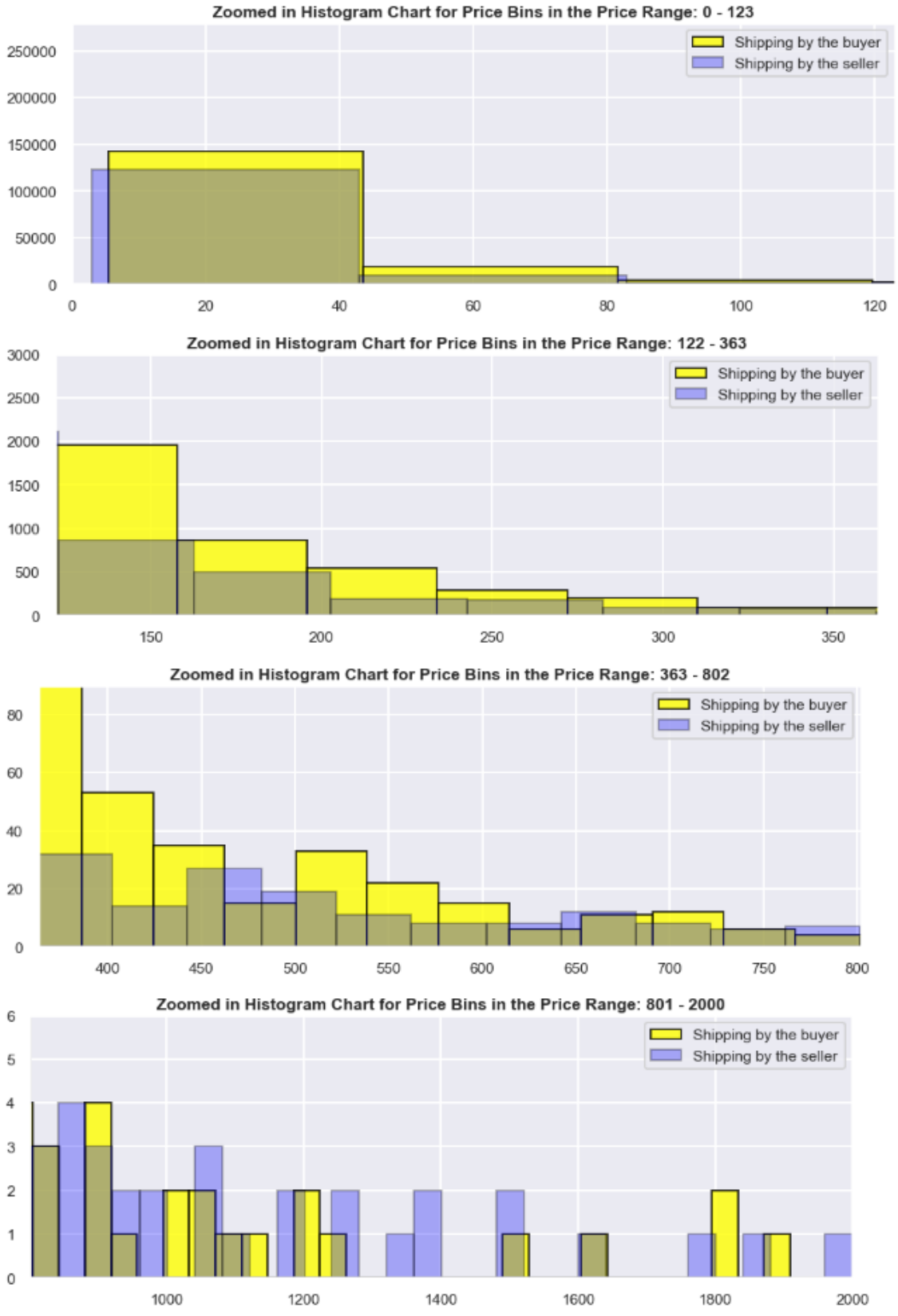

Figure 7: Histogram showing Shipping status on different price ranges

- *Towards the higher end of the price range , instances of buyers bearing the shipping charges increases.*
- *At the lower price bracket Seller bears the shipping charges more often.*

➢ When buying the items online, is the price higher 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).

- *Code to compare the median prices when seller / buyer pays the shipping cost*

```
'''
• 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).
'''
# create a pivot table with 'shipping' as the index . Aggregating Function is median for 'price'
a = pd.pivot_table(data = df,
                   index = 'shipping',
                   values= 'price',
                   aggfunc= 'median')

a = a.rename_axis('Shipping Charges') # rename the index column

a = a.rename(columns = {'price':'Median Price'}) # rename the column

shipping_mapping = {0: 'Seller Paid', 1: 'Buyer Paid'}  # map the shipping charges

a.index = a.index.map(shipping_mapping)  # replace 0 and 1

display(a.T)

plt.figure(figsize=(12,7))
g19 = sns.barplot(data = a,
                  x = a.index,
                  y = 'Median Price')
for i in g19.containers: g19.bar_label(i,)
plt.title('Median Prices vs Shipping Charges',fontweight = 'bold')

plt.show(g19)
```

Code 13: Code to compare the median prices when seller / buyer pays the shipping cost

Shipping Charges	Seller Paid	Buyer Paid
Median Price	19.0	14.0

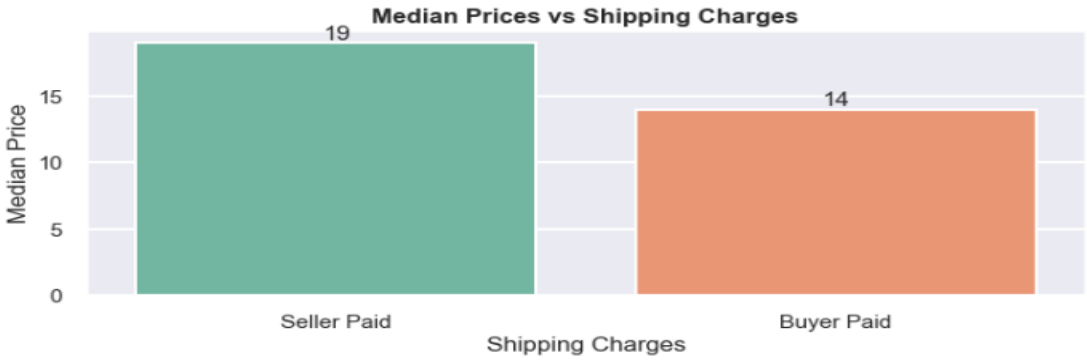

Figure 8: Median Price Comparison with shipping cost

- *Median Price when shipper bears the shipping cost is higher*

Question 1.4

Item condition information

- Write a code to print the count of the rows on each number (value) in column item_condition_id.

- *We call the function that we have defined earlier*

```
'''
• Write the code to find out (print) the count of the rows on each number (value) in column item_condition_id.
'''

# we call the function that we have defined in our EDA - code # 17 above

value_count_info(df,'item_condition_id')
```

Code 14: Code to print the count of the rows on each number (value) in column 'item_condition_id'

Value Count Analysis of : item_condition_id

item_condition_id	count	Percentage
1	132435	43.13%
3	89856	29.26%
2	77604	25.27%
4	6703	2.18%
5	469	0.15%

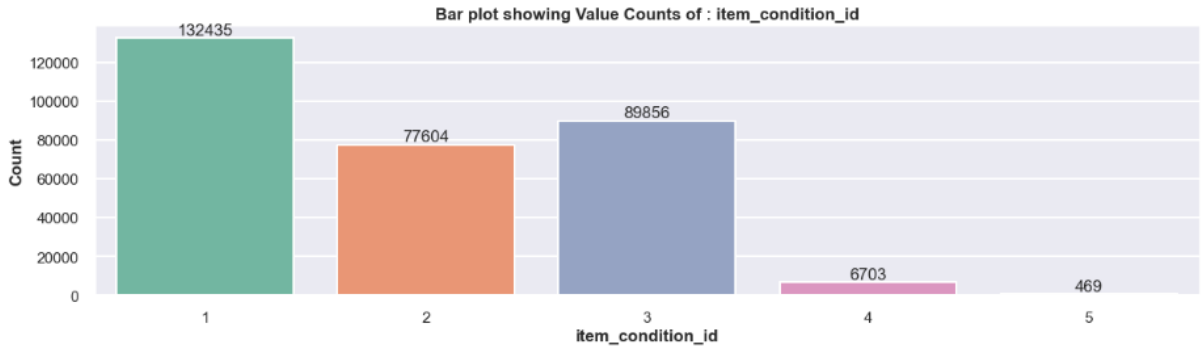

Figure 9: Count plot of the rows on each number (value) in column and visualize the same in a bar plot

- Draw the boxplot graphs (one plot) on the price for each item condition value and find out whether the better condition of the item could have a higher median price

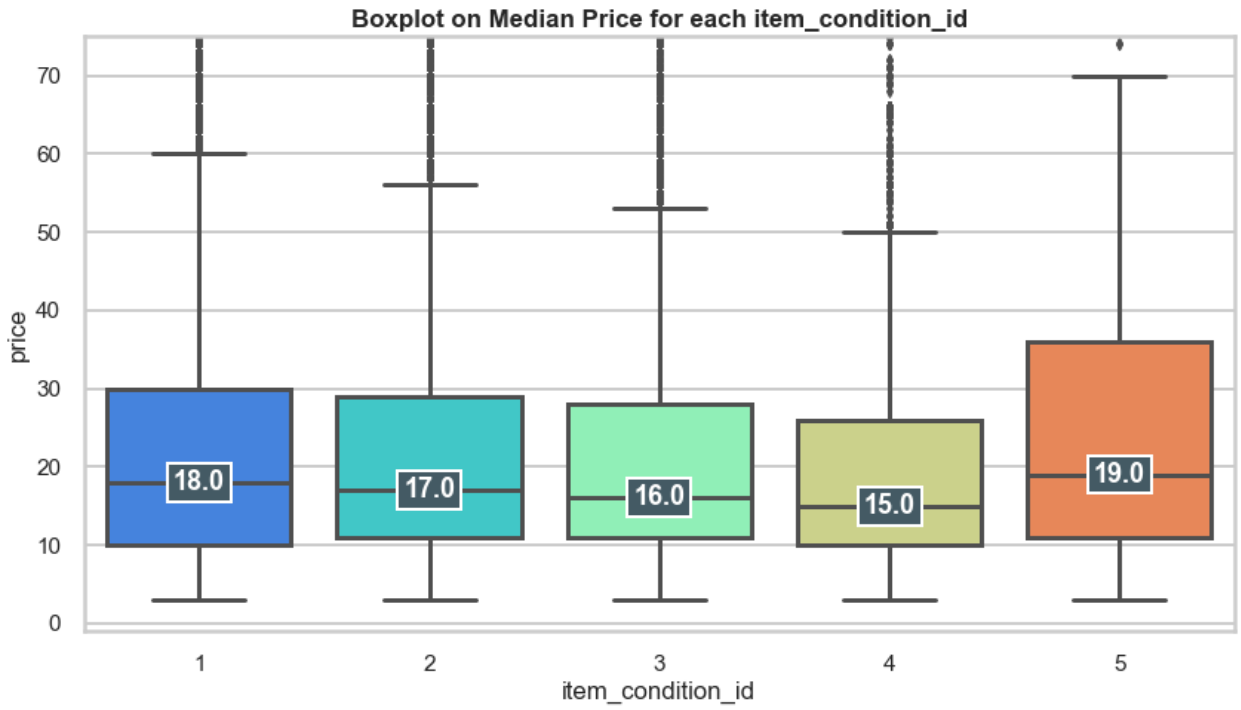

Figure 10: Boxplot graph on the price for each item condition value

- *items with item_condition_id = 1 (best condition) , median price : 18*
- *items with item_condition_id = 5 (poorest condition) , median price : 19*
- *We know from earlier analysis that items that have item_condition_id = 5 , constitute only 0.15 % of the total transactions. (469 transactions on item _condition_id =5 , out of a total of 307067 transactions)*

Question 1.5

Analysis of Categories

- Write the code to find out (print) how many unique categories you could find from column category_name.
  - *Code to find the number of unique names in 'category_name'*

```
'''
• Write the code to find out (print) how many unique categories you could find from column category_name.
'''
a = df['category_name'].nunique() # find number of unique categories in the feature 'category_name'

print('The number of unique categories in the dataset is:',a)
```

Code 15: Code to find number of unique category names

- *The number of unique categories in the dataset is: 1135*

➢ Write a code to print the top 3 categories which have the worst condition. (item_condition_id == 5)

```
'''
• For the items with worst condition only (highest value from item_condition_id),
write code to (print) find out the top 3 categories (now you probably understand the findings you had in Question 1.4).
'''
a = df[df['item_condition_id'] == 5] # slice the data frame for only those transactions which invlove item_condition_id =5

b = a['category_name'].value_counts()  # number value counts of category name
b = b.to_frame().rename(columns={'category_name':'count'}) # convert to frame and rename column

c = a['category_name'].value_counts(normalize=True)*100  # value counts of category name in percenatge
c = c.to_frame().rename(columns={'category_name':'Percentage'})

c['Percentage'] = c['Percentage'].apply(lambda x: f"{x:.1f}%") # display 'Percentage' in % format, upto 1 decimal place

d = a.groupby('category_name').median()   # find the median price of each category
d = d.rename(columns={'price':'Median_Price'}) # rename column

worst_condition_items_info = pd.concat([b,c,d],axis = 1) # create a combined data frame
display(worst_condition_items_info.head(3)) # view only the top three categories

print("Total number of Transactions involving 'item_condition_id' = 5 are :", len(df[df['item_condition_id'] == 5]))
```

Code 16: Code to print the top 3 categories which have the worst condition (item_condition_id ==5)

- *We find the top 3 categories of item_condition_id ==5, along with their median place*

	count	Percentage	Median_Price
Electronics/Cell Phones & Accessories/Cell Phones & Smartphones	115	24.6%	38.0
Electronics/Video Games & Consoles/Games	37	7.9%	18.0
Electronics/Video Games & Consoles/Consoles	31	6.6%	24.0

```
Total number of Transactions involving 'item_condition_id' = 5 are : 469
```

Table 12: Table showing top3 categories of item_condition_id 5

- *The top 3 categories where the item_condition_id is 5 are*

  Electronics/Cell Phones & Accessories/Cell Phones & Smartphones - ~25%

  Electronics/Video Games & Consoles/Games - ~8%

  Electronics/Video Games & Consoles/Consoles - ~7%

- *Everything belongs to the electronics category. Therefore, the median price in question 1.4 is the highest where item condition is 5, as electronics are high priced products thereby pushing the median price higher as almost 39% of the items fall under the category of Electronics*

Question 1.6

- Write a function to split the text content in the column category_name by '/ ' character. For missing values (Nan), the results from splitting should be "Category Unknown", "Category Unknown", "Category Unknown".

```
'''
• Write the function (must be function) to split the text content (string value in each row) in column
category_name by '/' character. you need to handle the exception in the function for those has missing values (NaN).
For missing values (NaN), the results from splitting should be "Category Unknown", "Category Unknown", "Category Unknown".
'''
# define a function 'split_column'

def split_column(DF, column_name,string):

    # make a new data frame with columns made by splitting a given column in the given data frame where ever there is '/'
    split_data = DF[column_name].str.split('/',
                                           expand=True, # new columns will be created in a new data frame
                                           n=2)         # splitting will stop after 2 splits. So max 3 cols will be added

    # name the new columns created
    split_data.columns = ['main_cat', 'subcat_1', 'subcat_2']

    # fill missing values in three new columns with a given string value
    split_data = split_data.fillna(string)

    # concatenate the newly created dataframe of split columns with the original dadatframe,so that new columns are added
    # to the original dataframe
    DF = pd.concat([DF, split_data], axis=1)

    return DF


```

Code 17: Code to write function to split column string

- *Use the newly created function to split the column in a copy of the original data frame*

```
'''
Use the above function you wrote to create three new columns main_cat,subcat_1 and
subcat_2 with corresponding values from the result of splitting. Print out the dataframe to
show the top 5 rows for three new columns main_cat,subcat_1 and subcat_2.
'''

df2 = split_column(df2,'category_name','Category Unknown')
df2.head()
```

	train_id	name	item_condition_id	category_name	brand_name	price	shipping	clean_description	main_cat	subcat_1	subcat_2
0	128037	Bundle for Sassy Sisters	3	Women/Tops & Blouses/Blouse	NaN	16.0	0	max cleo black dress paper crane black tank to...	Women	Tops & Blouses	Blouse
1	491755	PINK VS TANK	2	Women/Tops & Blouses/Tank, Cami	NaN	17.0	0	sequin pink sign sequins missing gently worn	Women	Tops & Blouses	Tank, Cami
2	470924	Funko Pop Unmasked Cyclops	1	Kids/Toys/Action Figures & Statues	Funko	30.0	1	box great condition comes soft pop protector p...	Kids	Toys	Action Figures & Statues
3	491263	Baby Roshe Runs	3	Kids/Boys 2T-5T/Shoes	Nike	19.0	0	baby black nike roshe runs size 5c	Kids	Boys 2T-5T	Shoes
4	836489	Baby Girl Ralph Lauren dresses	3	Kids/Girls 0-24 Mos/Dresses	Ralph Lauren	24.0	0	2 polo dresses 3 months wore washed dreft pink...	Kids	Girls 0-24 Mos	Dresses

Table 13: Data frame with split columns

- *We check if the missing values in the newly created columns have been filled correctly or not.*

In [47]:
```
# check if missing values in the newly created columns were replaced by 'Category Unkown'.

df2[df2['main_cat']=='Category Unknown'].head()  # view those rows where main_cat == 'Category Unknown'
```

Out[47]:

	train_id	name	item_condition_id	category_name	brand_name	price	shipping	clean_description	main_cat	subcat_1	subcat_2
121	420010	Authentic Leather Jacket	2	NaN	Black Rivet	16.0	0	100 authentic leather great condition worn twi...	Category Unknown	Category Unknown	Category Unknown
246	755244	Bundle 40 Dollars	3	NaN	NaN	46.0	0	description yet	Category Unknown	Category Unknown	Category Unknown
248	1001910	Rae Dunn darling mug	2	NaN	Rae Dunn	22.0	0	rae dunn darling mug	Category Unknown	Category Unknown	Category Unknown
315	942795	PRIDE	1	NaN	NaN	8.0	1	pride lot 2 pairs socks one purple white says ...	Category Unknown	Category Unknown	Category Unknown
621	168998	NOS Vintage mini-doll tie dye pencils	2	NaN	NaN	19.0	0	fantastic brand new old stock 60 comes display...	Category Unknown	Category Unknown	Category Unknown

- We can see the string 'Category Unknown' in the new columns

In [48]:
```
# check for missing values in the newly created columns

split_cols = ['main_cat','subcat_1','subcat_2']   # create a list of names of features

for i in split_cols:  # check for null values in a for loop
    print('Missing Values in',i,'are:', df2[i].isnull().sum())
```

```
Missing Values in main_cat are: 0
Missing Values in subcat_1 are: 0
Missing Values in subcat_2 are: 0
```

- There are no missing values in the newly created columns , which implies that all missing values have been correctly filled by the given string value of 'Category Unknown'

Code 18: Code for checking correctness of filling missing values in the new columns

Question 1.7

Analysis of the three new columns created after splitting the column '**category_name**'

- Write code (or function) to change the text (value in each row) from the new three columns to lowercase.
  - *Code to convert a column string to lower case*

```
'''
• Write code (or function) to change the text (value in each row) from the new three columns to lowercase.
'''
split_cols
```

```
['main_cat', 'subcat_1', 'subcat_2']
```

```
# run a loop code to change the string to lower case only for the newly created columns

for i in split_cols:
    df2[i] = df2[i].str.lower()

df2[split_cols].head(3)
```

Code 19: Code to convert a column string to lower case

- *Output of code to convert columns to lower case – top 3 rows*

	main_cat	subcat_1	subcat_2
0	women	tops & blouses	blouse
1	women	tops & blouses	tank, cami
2	kids	toys	action figures & statues

Table 14: Table showing conversion to lower case

- Draw the bar chart to find out the top 5 most popular main categories (in column main_cat) in the data (only showing the top 5).

```
Value Count Analysis of : main_cat
```

main_cat	count	Percentage
women	137595	44.81%
beauty	42976	14.00%
kids	35520	11.57%
electronics	25196	8.21%
men	19396	6.32%

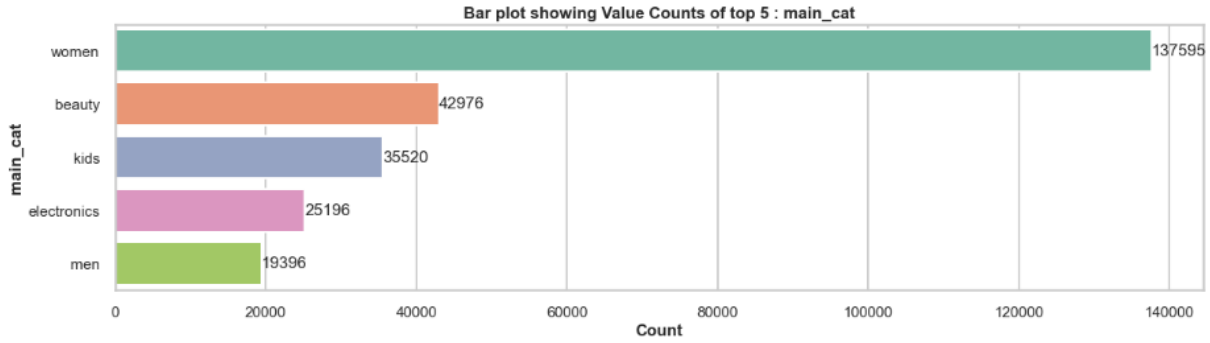

Figure 11: Top 5 Main Categories according to number of times transacted

- Write code (or function) to (print) find out how many unique main categories (in column main_cat), unique first sub-categories (in column subcat_1) and unique second sub-categories (in column subcat_2) respectively.

- *Code to find unique values in selected features*

```
'''
• Write code (or function) to (print) find out how many unique main categories (in column main_cat),
unique first sub-categories (in column subcat_1) and unique second sub-categories (in column subcat_2) respectively.
'''
split_cols = ['main_cat', 'subcat_1', 'subcat_2']

for i in split_cols:
    print('The number of unique categories in',i,'are :',df2[i].nunique())

```

```
The number of unique categories in main_cat are : 11
The number of unique categories in subcat_1 are : 114
The number of unique categories in subcat_2 are : 789
```

Code 20: Code to find unique values in features

Question 1.8

Exploring the price and categories.

- Write a code to print the median price for all the categories in the new column **main_cat**.
  - *Code to print the median price for all the categories in the new column main_cat along with a bar plot for better visualization*

```
'''
• Write code to (print) find out the median price for all the categories in new column main_cat.
'''
# create a pivot table
a = pd.pivot_table(data   = df2,
                   index  = 'main_cat',
                   values = 'price',
                   aggfunc= 'median').sort_values(by = 'price', ascending = False) # sort the median Price high to low

a = a.rename(columns={'price':'Median Price'})  # rename the price column correctly

display(a)

plt.figure(figsize=(15,5))
g22 = sns.barplot(data = a,
                  y = a.index,
                  x = a['Median Price'],
                  orient='h')
for i in g22.containers: g22.bar_label(i,)
plt.title('Bar Chart to show median price for all the categories "main_cat"')
plt.show(g22)
```

Code 21: Code to find median prices for all the categories in the new column main_cat with bar plot

main_cat	Median Price
men	21.0
women	19.0
category unknown	18.0
home	18.0
sports & outdoors	16.0
vintage & collectibles	16.0
beauty	15.0
electronics	15.0
kids	14.0
other	14.0
handmade	11.0

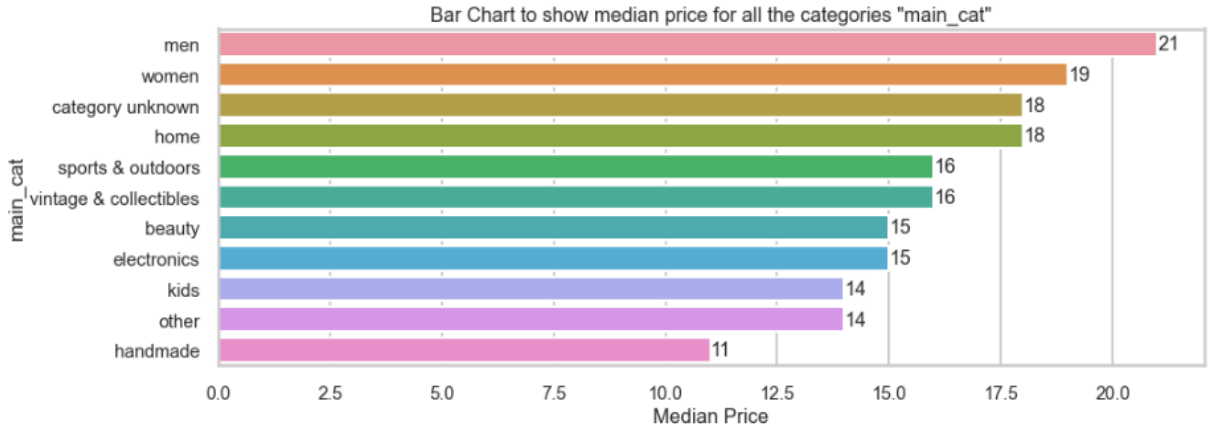

Figure 12: Top 10 main Categories according to Median Prices

- Draw a bar chart to find out the top 10 most expensive first sub-categories (in the column subcat_1) in the data.
  - *Code to find the categories in subcat_1 with highest mean and median prices. Construct a pivot table with index as 'subcat_1' for which the price is aggregated to find Maximum / Mean / Median prices of subcat_1*

```
'''
• Draw the bar chart to find out the top 10 most expensive first sub-categories (in column subcat_1) in the data.
'''
# create a pivot to view the max , mean and median prices of items in subcat_1
a = pd.pivot_table(data   =  df2,
                   index  = 'subcat_1',
                   values = 'price',
                   aggfunc= ['max','mean','median'])

a = a.sort_values(by = ('max','price'),ascending=False)  # sort from high to low the 'max' price

a = round(a.head(10),0)  # view the top 10
display(a)

a.head(10).plot(kind ='barh', figsize = (20,8));
```

Table 15: Subcat_1 – top 10 categories based on mean ,median and max prices

- *Top 10 subcat_1 categories based on Maximum Price*

subcat_1	max price
makeup	2000.0
cell phones & accessories	1909.0
women's handbags	1850.0
jewelry	1800.0
computers & tablets	1400.0
bags and purses	1250.0
collectibles	1050.0
category unknown	1000.0
men's accessories	850.0
daily & travel items	840.0

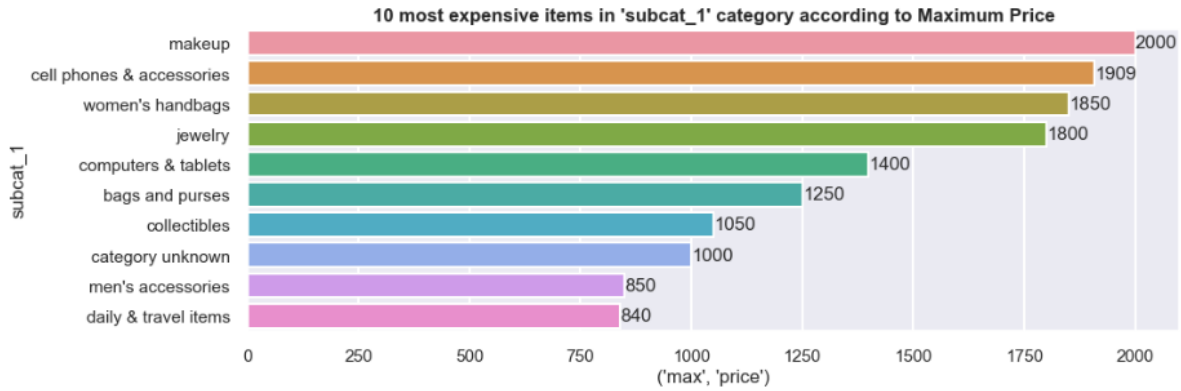

Figure 13: Top 10 subcat_1 categories based on Maximum Price

- *Top 10 subcat_1 categories based on Mean and Median Price*

subcat_1	mean price
computers & tablets	82.3
strollers	81.7
cameras & photography	70.8
women's handbags	58.8
bags and purses	58.1
footwear	47.5
musical instruments	45.4
suits	43.3
golf	43.2
shoes	41.7

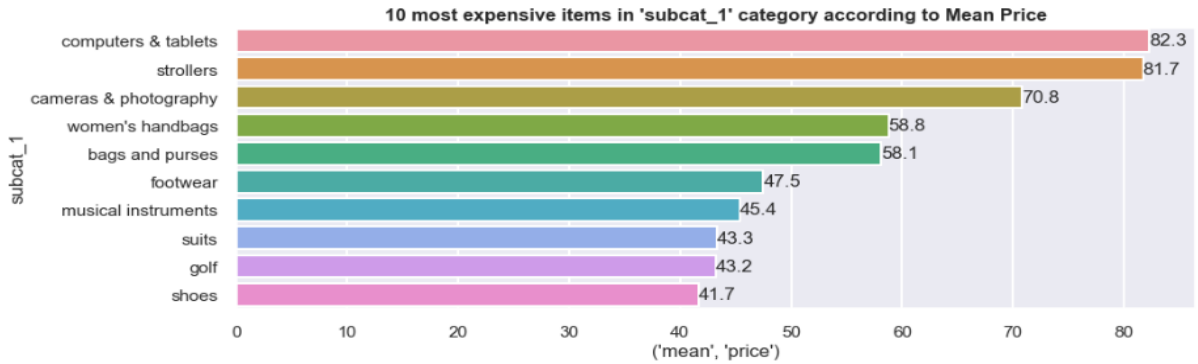

Figure 14: Top 10 subcat_1 categories based on Mean Price

subcat_1	median price
computers & tablets	40.0
cameras & photography	40.0
strollers	39.0
suits	33.0
blazers & sport coats	32.5
footwear	31.0
women's handbags	31.0
musical instruments	31.0
shoes	29.0
bags and purses	26.0

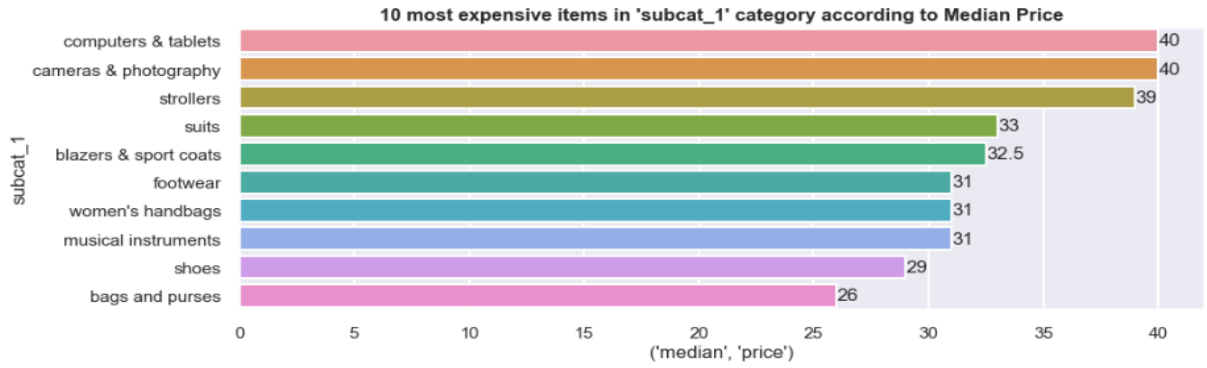

Figure 15: Top 10 subcat_1 categories based on Median Price

- Draw a bar chart to find out the top 10 cheapest second sub-categories (in column subcat_2) in the data.

- *Construct a pivot table with index 'subcat_2' for which prices are aggregated to ascertain the mean and median and they are sorted in low to high order. Top 10 rows will display the subcat_2 categories which have the lowest minimum / mean / median prices.*

subcat_2	min price	mean price	median price
100 years or older	3.0	26.7	22.0
lanyard	3.0	11.0	9.0
lamps& accessories	3.0	22.5	16.0
knit top	3.0	19.9	15.0
knee-length	3.0	29.7	24.0
...	...	...	...
standard	44.0	90.0	76.0
art doll	45.0	45.0	45.0
television	46.0	46.0	46.0
feather beds	60.0	60.0	60.0
entertaining/serving	87.0	87.0	87.0

Table 16: Pivot table showing cheapest subcat_2 items

- *Lowest Priced subcat_2 items based on Minimum Price*

subcat_2	min price
100 years or older	3.0
lamps& accessories	3.0
knit top	3.0
knee-length	3.0
kitchen utensils & gadgets	3.0
kids' room décor	3.0
video gaming merchandise	3.0
kids	3.0
keychain	3.0
leg warmers	3.0

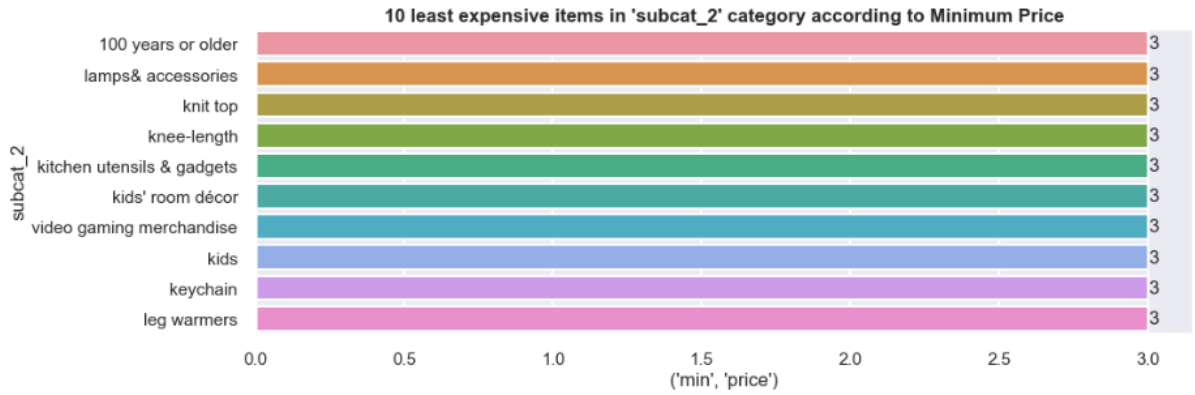

Figure 16: Subcat_2 bottom 10 categories based on Min Prices

- *Lowest Priced subcat_2 items based on Mean Price*

subcat_2	mean price
historical, military	3.0
photography	3.0
necktie	3.0
mirror	5.0
hair relaxers	5.0
sticker	6.0
patch	6.0
fixture	6.0
graphic design	6.0
pinback button	7.0

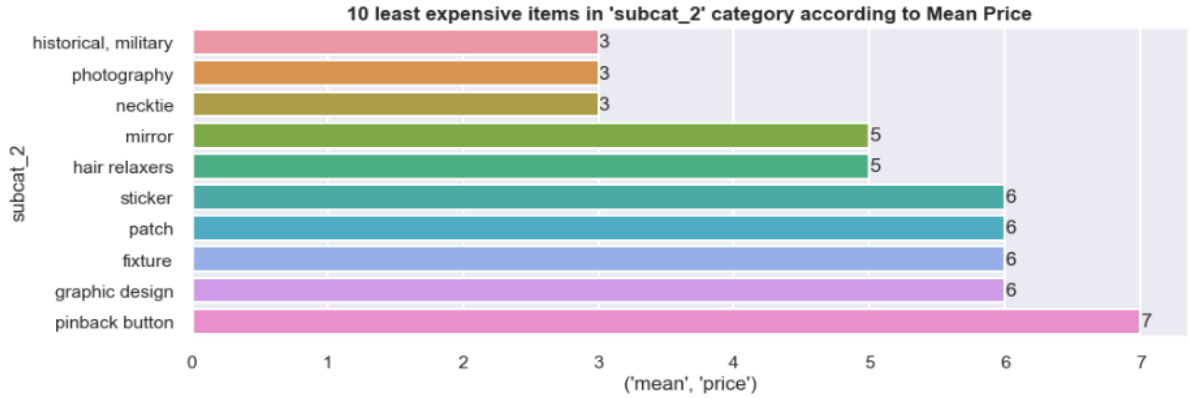

Figure 17: Subcat_2 bottom 10 categories based on Mean Prices

- *Lowest Priced subcat_2 items based on Median Price*

subcat_2	median price
historical, military	3.0
photography	3.0
necktie	3.0
sticker	4.0
print	4.0
mirror	5.0
hair relaxers	5.0
patch	5.0
pinback button	5.0
magnet	6.0

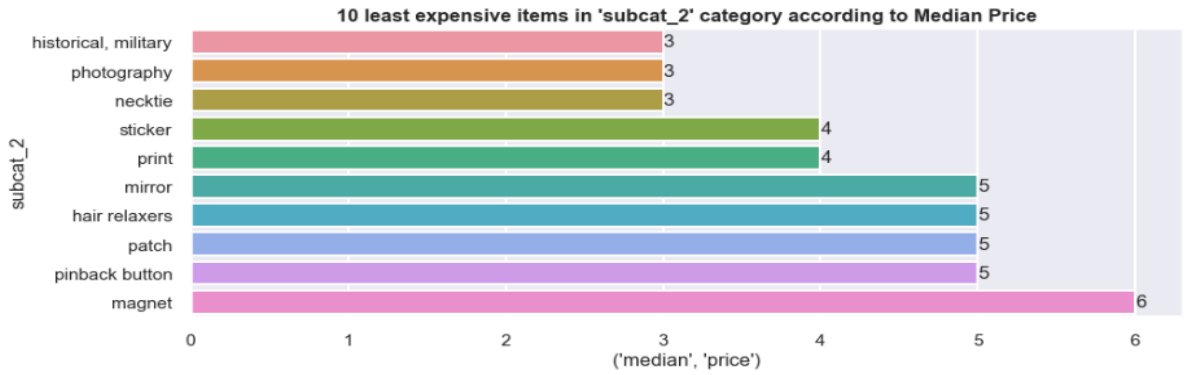

Figure 18: Subcat_2 bottom 10 categories based on Median Prices

Question 1.9

Exploring the price and brand.

- Write code to (print) find out the median price for all the brands (fill Nan with 'brand unavailable').
  - *Code to (print) find out the median price for all the brands (fill Nan with 'brand unavailable') and to plot a bar chart for the top 10 brands with highest median prices for visualization.*

```
"""
fill NaN values in the column 'barnd_name' with string 'brand unavailable'
"""
print('Number of transactions where brand name is not mentioned :',df2['brand_name'].isna().sum())

# fill missing values in the feature 'brand_name' with string value 'brand unavailable'
df2['brand_name'] = df2['brand_name'].fillna('brand unavailable')

# check if brand_name missing values have been filled or not
a = df2[df2['brand_name'] == 'brand unavailable'].shape[0]
print("Count of transactions where missing values in 'brand_name' is filled with string 'brand unavailable':", a )

```

Code 22: Code to fill the missing values with "brand_unavailable"

- *Number of transactions where brand name is not mentioned : 131227*
- *Count of transactions =missing values in 'brand_name' is filled with string 'brand unavailable': 131227*

Code 23: Code to find median price of brands

```
'''
• Write code to (print) find out the median price for all the brands
'''

# find the median price of brands and sort by price high to low
a = df2.groupby('brand_name').median().sort_values(by='price',ascending=False)
a = a.rename(columns = {'price':'Median Price'})  # rename the column from 'price' to 'Median Price'

display(a)
```

Code 24: Code for Median Prices of Brand

brand_name	Median Price
Tiffany Designs	359.0
Stuart Weitzman	329.0
Blendtec	280.0
IBM	275.0
MICHELE	254.0
...	...
A.B.S. by Allen Schwartz	3.0
Genica	3.0
Toys R Us Plush	3.0
Gold Bond	3.0
Jinx	3.0

3046 rows × 1 columns

Table 17: Median Prices of Brands

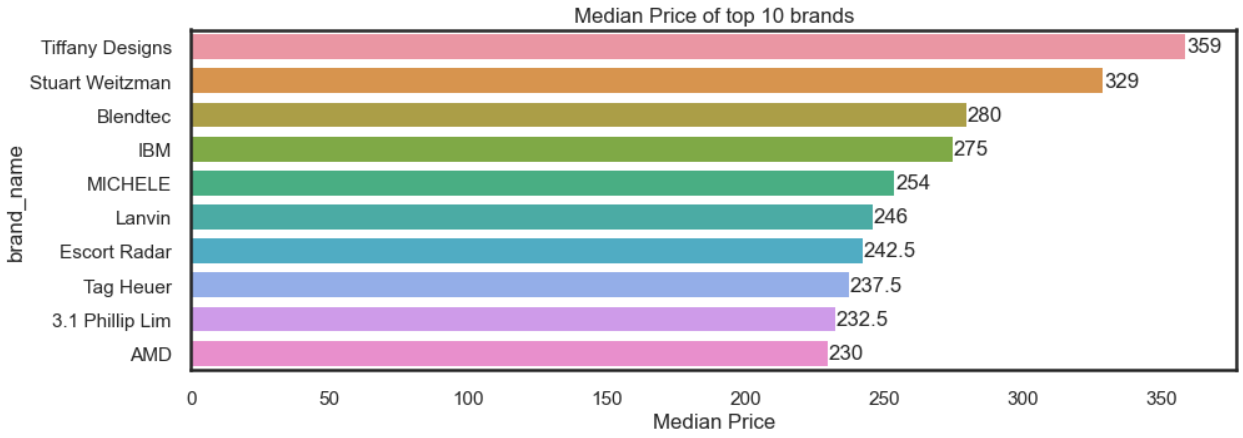

Figure 19: Top Brands based on highest Median Price

Question 1.10

Item Description Analysis

- Draw the word cloud chart by using the column clean_description.

'clean_description' consists of these words in each row

```
0         max cleo black dress paper crane black tank to...
1              sequin pink sign sequins missing gently worn
2         box great condition comes soft pop protector p...
3                         baby black nike roshe runs size 5c
4         2 polo dresses 3 months wore washed dreft pink...
                                ...
355803          beats dre solo white gently used work great
355804    viewing 4 new leap frog leapster learning game...
355805    couple places lace snagged tell fairly good co...
355806                                              size 11
355807                                   description yet
Name: clean_description, Length: 307236, dtype: object
```

- *A list of 'stop words' is created including 'punctuation marks' by importing the necessary libraries*

```
# create a list consisting stopwords and punctuaion marks

stopwords = nltk.corpus.stopwords.words('english')
punctuations = list(string.punctuation)

list_stopwords = stopwords + punctuations
```

Code 25: Code to ascertain stop words

- *A function is defined to make a word cloud from the corpus of words in which the input parameters are a word corpus and a name of a data frame.*

```python
'''
Draw the wordcloud chart by using the column clean_description.
'''

# create a function to generate word cloud with 3 input parameters
def word_cloud_image(df,feature_name,df_name):
    words = ' '.join(df[feature_name].astype(str))   # create the word corpus
    wordcloud = WordCloud(stopwords=list_stopwords,
                          background_color='black',
                          width = 1000,
                          height = 700,
                          random_state=seed).generate(words)

    plt.figure(figsize=(10,10))
    plt.imshow(wordcloud)
    plt.title(f'Word Cloud from feature : {feature_name} of data set {df_name}',fontweight = 'bold')
    plt.axis('off')
    plt.show()
    return
```

Code 26: Function to output word cloud image

- *To draw the word cloud chart by using the column ' clean_description ' from the full data frame*

```python
# utilize the function to generate the word cloud
word_cloud_image(df2,'clean_description','df2')
```

- *The word cloud*

Word Cloud from feature : clean_description of data set df2

Figure 20: Word Cloud for words in 'clean_description' for full data frame

- Divide the data with quantiles of the price (1st , 2nd , 3rd and 4th quantile)

  - *The data frame is sliced into 4 parts based on the 4 quantiles of price. A new column is introduced in the data frame with the labels Q1, Q2, Q3, Q4*

```
'''
• Divide the data with quantiles of the price (using qcut from pandas to obtain the first/second/third/fourth quantile).
'''
quantiles = pd.qcut(df2['price'],
                    q=4,                              # will slice the dataframe into 4 quantiles
                    labels=['Q1', 'Q2', 'Q3', 'Q4'])  # label the price quantile

df2['quantiles'] = quantiles   # create a new column in the data frame which will display the quamtiles
```

```
# slice the dataframe as per the different quantiles

Q1_df = df2[df2['quantiles']=='Q1']
Q2_df = df2[df2['quantiles']=='Q2']
Q3_df = df2[df2['quantiles']=='Q3']
Q4_df = df2[df2['quantiles']=='Q4']

```

```
# ascertain the min and max values of the prices for the 4 quantiles

q = [Q1_df,Q2_df,Q3_df,Q4_df]

for i in range(len(q)):
    print('Min and Max values for prices are:',
          q[i].price.describe().loc[['min','max']].to_list(),'for quantile',f'Q{i + 1}')

```

Code 27: Code for creating a new feature based on 4 quantiles on price

  - *4 new data frames are created based on the newly created Quantile labels.*

```
Q1_df = df[df['quantiles']=='Q1']
Q2_df = df[df['quantiles']=='Q2']
Q3_df = df[df['quantiles']=='Q3']
Q4_df = df[df['quantiles']=='Q4']
```

Code 28: Slice the data frame into price quantiles

  - *The price range of the 4 quantiles is ascertained*

```
q = [Q1_df,Q2_df,Q3_df,Q4_df]

for i in range(len(q)):
    print('Min and Max values for prices are:',q[i].price.describe().loc[['min','max']].to_list(),'for quantile',f'Q{i + 1}'
```

Code 29: Code to ascertain the price range of the 4 quantiles on price

  - *Min and Max values for prices are: [3.0, 10.0] for quantile Q1*
  - *Min and Max values for prices are: [10.5, 17.0] for quantile Q2*
  - *Min and Max values for prices are: [18.0, 29.0] for quantile Q3*
  - *Min and Max values for prices are: [30.0, 2000.0] for quantile Q4*

- Draw the word cloud by using the column clean_description on each quantile of price data.

```
# ascertain the number of transactions in each quantile

q = [Q1_df,Q2_df,Q3_df,Q4_df]

for i in range(len(q)):
    print('Transaction Count for quantile 'f'Q{i + 1} is :', len(q[i]))
```

```
Transaction Count for quantile Q1 is : 77874
Transaction Count for quantile Q2 is : 78814
Transaction Count for quantile Q3 is : 74628
Transaction Count for quantile Q4 is : 75751
```

Figure 21: : Word Cloud of words from feature "clean_description" of data frame Q1 quntile

Figure 22: Word Cloud of words from feature "clean_description" of data frame Q2 quntile

Figure 23: Word Cloud of words from feature "clean_description" of data frame Q3 quantile

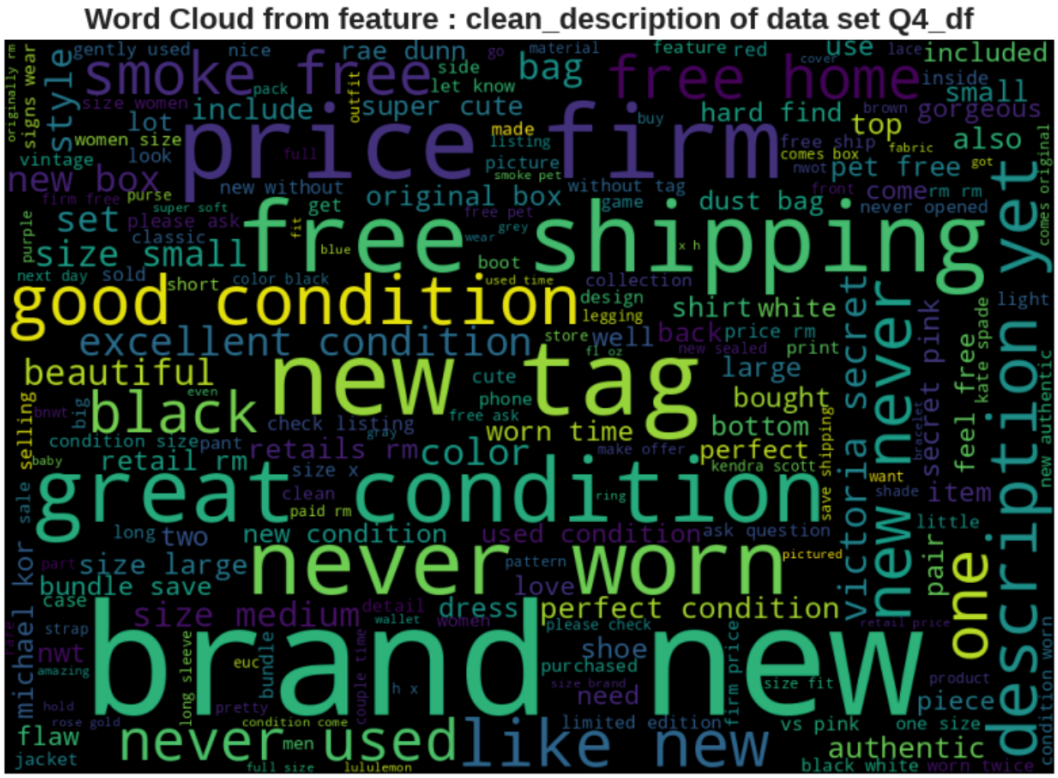

Figure 24: Word Cloud of words from feature "clean_description" of data frame Q4 quantile

Question 2.1

The dataset is the New York City Taxi Demand. The raw data is from the NYC Taxi and Limousine Commission. The data consists of aggregating the total number of taxi passengers into 30-minute buckets.

Read data and explore the time series.

First 5 rows of the data

timestamp	value
2014-07-01 00:00:00	10844
2014-07-01 00:30:00	8127
2014-07-01 01:00:00	6210
2014-07-01 01:30:00	4656
2014-07-01 02:00:00	3820

Last 5 rows of the data

timestamp	value
2015-01-31 21:30:00	24670
2015-01-31 22:00:00	25721
2015-01-31 22:30:00	27309
2015-01-31 23:00:00	26591
2015-01-31 23:30:00	26288

NUmber of data points in the data frame are: 10320

Table 18: Time Series Data on New York Taxi Demand

- *The start date is 1st July 2014*
- *The last date is 31st Jan 2015*
- *Data is for number of taxi riders in every half hour interval.*
- *Total number of observations are 10320*

Information on the data

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 10320 entries, 2014-07-01 00:00:00 to 2015-01-31 23:30:00
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   value   10320 non-null  int64
dtypes: int64(1)
memory usage: 161.2 KB
```

- *Index column is in date time format and there are no missing values*

Time Series Plot for the entire period from 1st July 2014 to 31st Jan 2015.

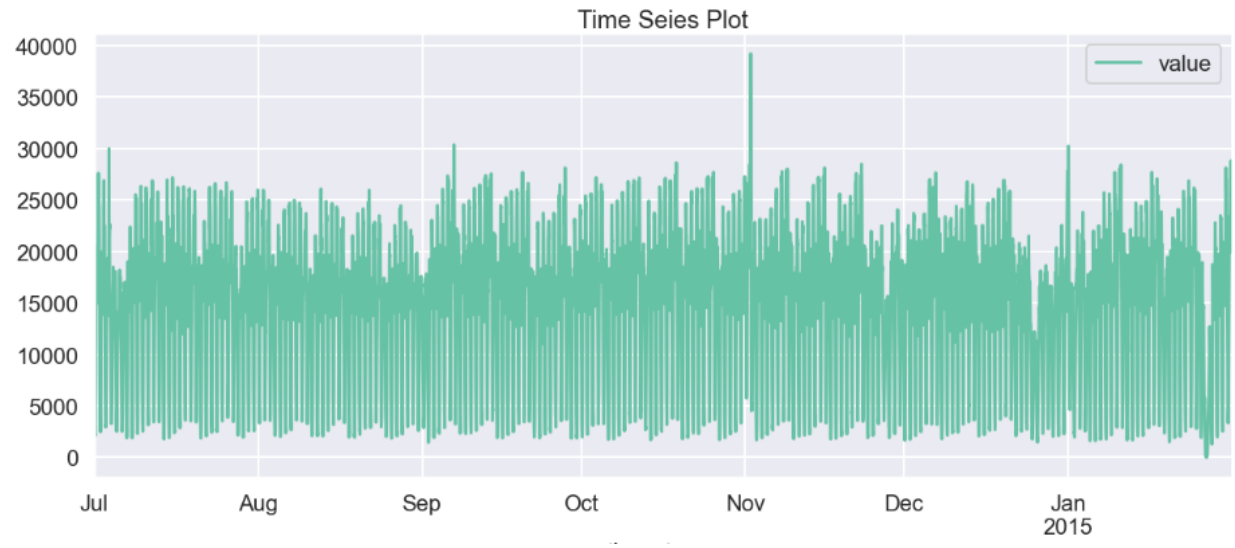

Figure 25: Time Series Plot

Bar plot depicting the total monthly demand (sum)

	0	1	2	3	4	5	6
year	2014	2014	2014	2014	2014	2014	2015
month	7	8	9	10	11	12	1
Total Rides	22311198	21695693	22497659	23937235	22308660	22042382	21426889

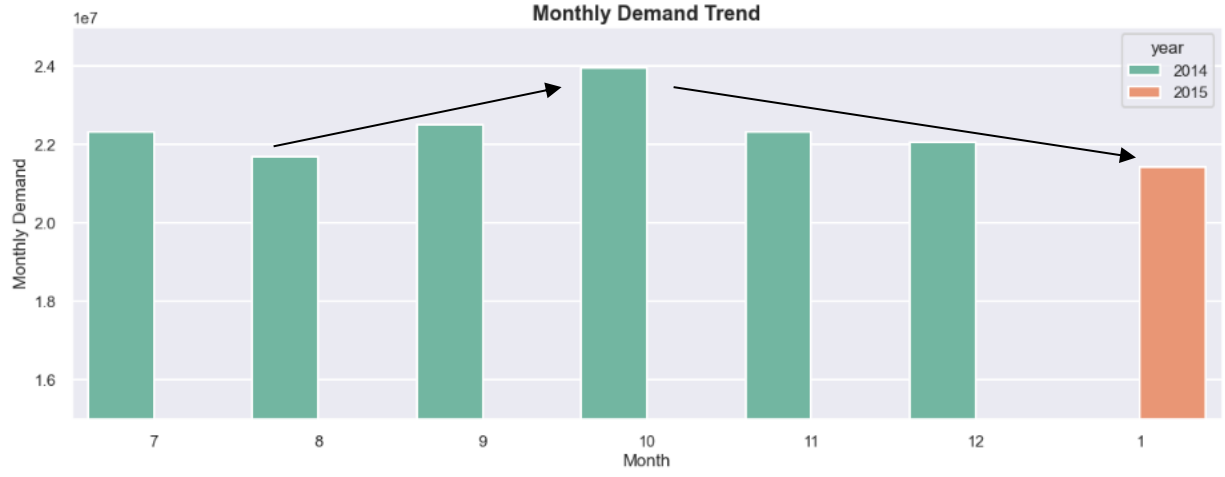

Figure 26: Bar plot showing monthly total demand

- *October month saw the demand peak*
- *Nov 2014, Dec 2014 and then Jan 2015 are showing a steady decline.*
- *Aug 2014 to Oct 2014 saw a steady increase in demand.*

Bar plot showing variation in demand according to dates within a month

Month Date	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Daily Demand	5159166	5044292	5078536	4831805	4983462	5146137	5197271	5309747	5255182	5304335	5255902	5275480	5366018	5230449	5229797

Month Date	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Daily Demand	5302004	5250073	5369313	5270219	5236237	5100868	5150100	5103166	4980949	4729473	4563639	4284431	4580632	4810807	4995727	3824499

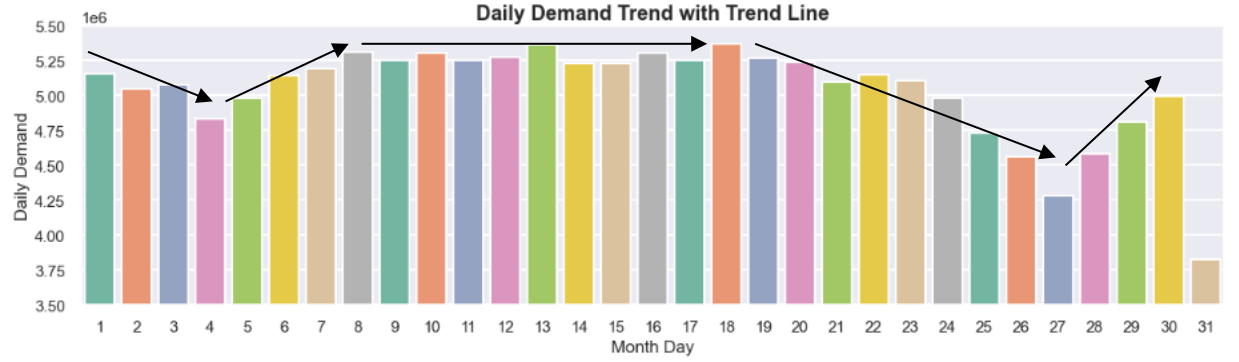

Figure 27: Bar plot showing total taxi demand on calendar days

- *The trends of daily demand according to the date of the month is shown above.*

Bar plot showing variation in demand according to time within a day

Hour of the day	0	1	2	3	4	5	6	7	8	9	10	11
Hourly Demand	6296122	4669140	3444457	2495045	1833559	1540893	3136845	5327804	6595224	6889104	6922719	7281544

Hour of the day	12	13	14	15	16	17	18	19	20	21	22	23
Hourly Demand	7659422	7650318	7924883	7585604	6477726	7766845	9399540	9843478	9277652	9234956	8984049	7982787

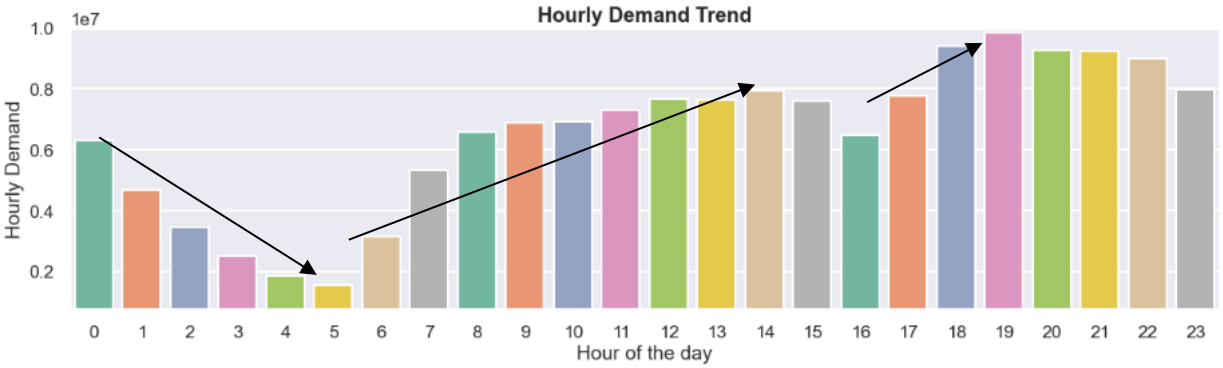

Figure 28:Hourly variation in demand during a day

- *The demand starts dipping after midnight until 6 am.*
- *Peak demand is at 8 pm*

- Create two new data frames **df_day** and **df_hour** by aggregating the demand value on daily and hourly level.

    - *Code to create two data frames , one aggregated by the day and the other aggregated by the hour. We need to keep in mind that the data provided is for half hour buckets.*

```
df_day = df.resample('D').sum() # data available is every half hour, so we sum up for a day to get daily data
df_hour = df.resample('H').sum() # data available is every half hour, so we sum up 2 half hour data to get hourly data

print('\n','Number of Data points in the data frame df_day are',df_day.shape[0])

print('\n','First 5 rows of the data frame df_day')
display(df_day.head(5))
print('\n','Last 5 rows of the data frame df_day')
display(df_day.tail(5))

print('\n','Number of Data points in the data frame df_hour are',df_hour.shape[0])

print('\n','First 5 rows of the data frame df_hour')
display(df_hour.head(5))
print('\n','Last 5 rows of the data frame df_hour')
display(df_hour.tail(5))
```

Code 30: : Code to up sample the series to hourly and day

df_day

Number of Data points in the data frame df_day are 215

First 5 rows of the data frame df_day

timestamp	value
2014-07-01	745967
2014-07-02	733640
2014-07-03	710142
2014-07-04	552565
2014-07-05	555470

Last 5 rows of the data frame df_day

timestamp	value
2015-01-27	232058
2015-01-28	621483
2015-01-29	704935
2015-01-30	800478
2015-01-31	897719

Table 19: df_day data frame

df_hour

Number of Data points in the data frame df_hour are 5160

First 5 rows of the data frame df_hour

timestamp	value
2014-07-01 00:00:00	18971
2014-07-01 01:00:00	10866
2014-07-01 02:00:00	6693
2014-07-01 03:00:00	4433
2014-07-01 04:00:00	4379

Last 5 rows of the data frame df_hour

timestamp	value
2015-01-31 19:00:00	56577
2015-01-31 20:00:00	48276
2015-01-31 21:00:00	48389
2015-01-31 22:00:00	53030
2015-01-31 23:00:00	52879

Table 20: df_hour data frame

- Plot the demand value in two-line charts for both df_day and df_hour data frames.

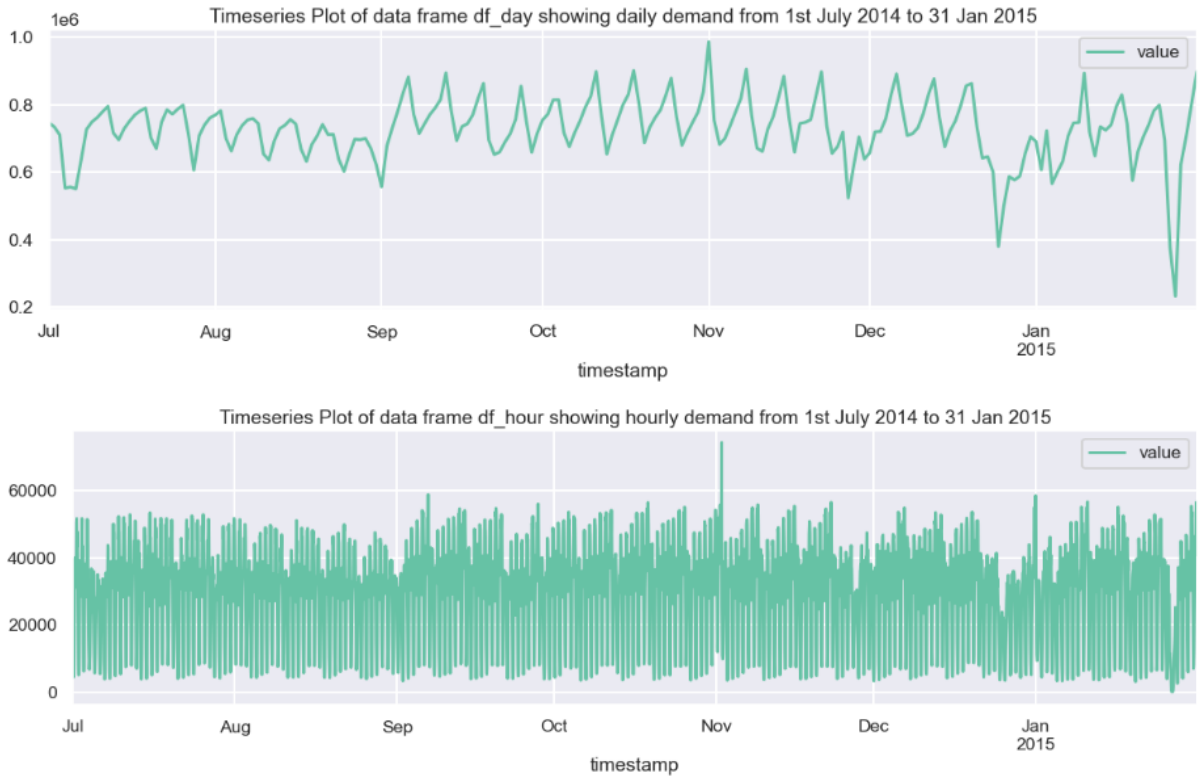

Figure 29: Plot df_day and df_hour

- Plot the seasonal decomposition components (Trend, Seasonal, Residual) from df_day data frame, 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)?

- *Multiplicative model is chosen because the residuals are smaller than those obtained from the additive model*

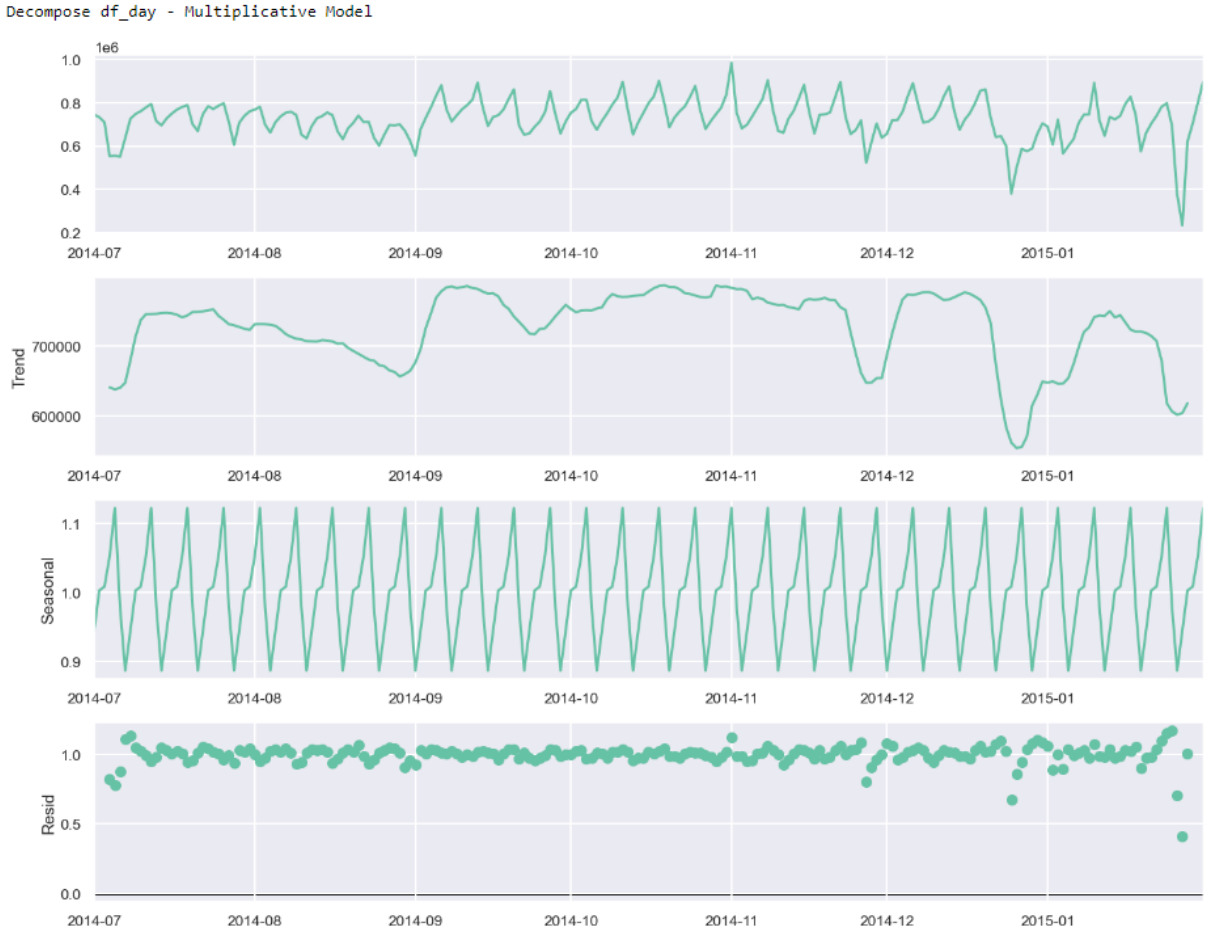

Figure 30: Multiplicative Decomposition plot of df_day

Augmented Dickey-Fuller test

The Augmented Dickey-Fuller test is an unit root test which determines whether there is a unit root and subsequently whether the series is non-stationary.

The hypothesis in a simple form for the ADF test is:

- H_0 : The Time Series has a unit root and is thus non-stationary.
- H_1 : The Time Series does not have a unit root and is thus stationary.

We would want the series to be stationary for building ARIMA models and thus we would want the p-value of this test to be less than the α value.

Test Statistic Value : -3.4481
p_value: 0.0094

- *Since the p_value is less than α =0.05 (Significance level of 5 %) , we reject the Null Hypothesis, and conclude that there is not enough statistical evidence to prove that the series is stationary*

Question 2.2

ARIMA and other models for forecasting

- Create the **acf** and **pacf** plots for df_day data frame.

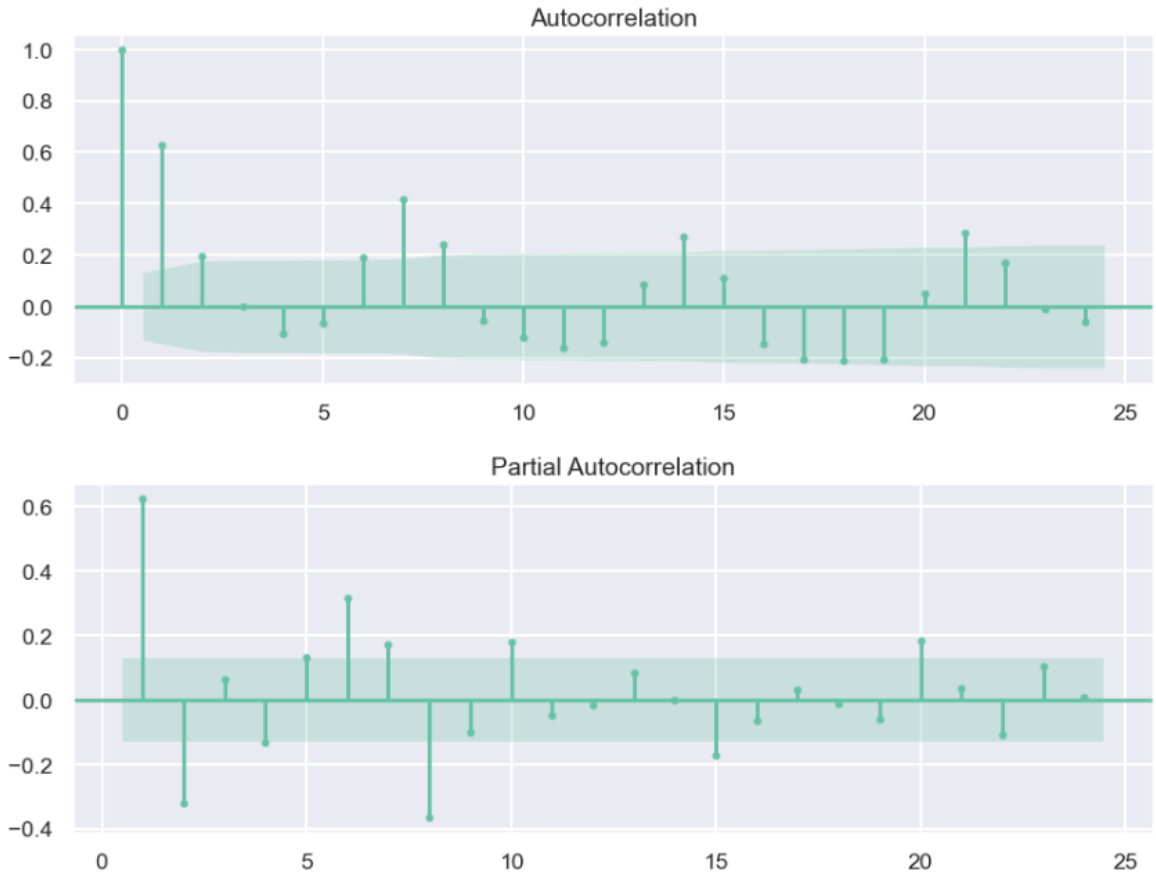

Figure 31: ACF and PACF plots

- Find the best model with different parameters on ARIMA model.
  The parameter range for p,d,q is all from [0, 1, 2]. Find the best model with lowest Mean Absolute Error from 27 choices based on the time from ”Jul-01-2014” to ”Dec-01-2014”.

  Brute Force Method

  - *We create 27 combinations of p ,d, and q when they can take any value from 0 to 2.*

```
import itertools
p = d = q = range(0, 3)

pdq = list(itertools.product(p, d, q))
for i in range(0,len(pdq)):
    print('Model: {}'.format(pdq[i]))


print("\n")
print('The number of combinations of p,d,and q are :',len(pdq))
```

Code 31: Code to create 27 combinations of (p, d, q)

```
Model: (0, 0, 0)
Model: (0, 0, 1)
Model: (0, 0, 2)
Model: (0, 1, 0)
Model: (0, 1, 1)
Model: (0, 1, 2)
Model: (0, 2, 0)
Model: (0, 2, 1)
Model: (0, 2, 2)
Model: (1, 0, 0)
Model: (1, 0, 1)
Model: (1, 0, 2)
Model: (1, 1, 0)
Model: (1, 1, 1)
Model: (1, 1, 2)
Model: (1, 2, 0)
Model: (1, 2, 1)
Model: (1, 2, 2)
Model: (2, 0, 0)
Model: (2, 0, 1)
Model: (2, 0, 2)
Model: (2, 1, 0)
Model: (2, 1, 1)
Model: (2, 1, 2)
Model: (2, 2, 0)
Model: (2, 2, 1)
Model: (2, 2, 2)


The number of combinations of p,d,and q are : 27
```

Split the data into train and test data sets. Train Data starts from Jul-01-2014 to Dec-01-2014.

```
start_date = '2014-07-01'
end_date ='2014-12-01'
start_date = pd.to_datetime(start_date)
end_date = pd.to_datetime(end_date)
```

```
train_df_day = df_day[(df_day.index <= end_date)]
train_df_day
```

```
test_df_day = df_day[(df_day.index > end_date)]
test_df_day
```

Code 32: Code to create the train and test data

- *The train data set*

timestamp	value
2014-07-01	745967
2014-07-02	733640
2014-07-03	710142
2014-07-04	552565
2014-07-05	555470
...	...
2014-11-27	523184
2014-11-28	616841
2014-11-29	704360
2014-11-30	638317
2014-12-01	656814

154 rows × 1 columns

Table 21: Train Data set on df_day

Train data has 154 observations

- *The test data set*

timestamp	value
2014-12-02	719097
2014-12-03	720080
2014-12-04	758203
2014-12-05	829996
2014-12-06	890958
...	...
2015-01-27	232058
2015-01-28	621483
2015-01-29	704935
2015-01-30	800478
2015-01-31	897719

61 rows × 1 columns

Table 22: Test Data set on df_day

Test data set has 61 observations

Code to fit the ARIMA model in a loop with all the 27 combinations of p, d and q found earlier

```
# import the libraries

from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_absolute_percentage_error,mean_squared_error

```

```
# create an empty data frame to store the results of the order and MAPE
ARIMA_MAPE = pd.DataFrame(columns=['param','MAPE'])
ARIMA_MAPE
```

```


for param in pdq:
    # fit the ARIMA model on the train data set
    ARIMA_model = ARIMA(train_df_day.values, order=param).fit()

    # make the predictions for the period where actual value is known - which is the test data period
    y_predicted = ARIMA_model.forecast(steps=len(test_df_day))

    # actual value as avaiilable in the test data set
    y_true = test_df_day.values

    # calculate the Mean Absolute Percentage Error
    MAPE = round(mean_absolute_percentage_error(y_true, y_predicted)*100,4)

    # Append the result to ARIMA_MAPE data frame created earlier
    ARIMA_MAPE = ARIMA_MAPE.append({'param':param,'MAPE':MAPE},ignore_index=True)

   # print('ARIMA{} - MAPE: {:.2f}'.format(param, MAPE))  # Print the parameters and MAPE

ARIMA_MAPE.sort_values(by='MAPE',inplace=True)
display(ARIMA_MAPE.reset_index())

best_param = ARIMA_MAPE.iloc[0]['param']
print('\n','The (p,d,q) combination which provides the lowest MAPE figure is:',best_param)
```

Code 33: Code to run the ARIMA model on the 27 combinations of (p,d,q)

ARIMA model results

	index	param	MAPE
0	18	(2, 0, 0)	16.5584
1	14	(1, 1, 2)	16.5845
2	19	(2, 0, 1)	16.5864
3	2	(0, 0, 2)	16.5970
4	10	(1, 0, 1)	16.5994
5	1	(0, 0, 1)	16.6045
6	0	(0, 0, 0)	16.6221
7	11	(1, 0, 2)	16.6918
8	9	(1, 0, 0)	16.7092
9	20	(2, 0, 2)	16.7972
10	21	(2, 1, 0)	16.8062
11	22	(2, 1, 1)	16.8817
12	5	(0, 1, 2)	16.9742
13	13	(1, 1, 1)	17.0188
14	25	(2, 2, 1)	17.0383
15	4	(0, 1, 1)	17.0847
16	12	(1, 1, 0)	17.1110
17	3	(0, 1, 0)	17.1382
18	7	(0, 2, 1)	17.2110
19	23	(2, 1, 2)	18.2033
20	16	(1, 2, 1)	19.3619
21	8	(0, 2, 2)	22.3498
22	17	(1, 2, 2)	22.9302
23	15	(1, 2, 0)	30.6080
24	24	(2, 2, 0)	67.0222
25	6	(0, 2, 0)	89.3169
26	26	(2, 2, 2)	98.4856

```
The (p,d,q) combination which provides the lowest MAPE figure is: (2, 0, 0)
```

Table 23: ARIMA models -parameters and MAPE results

- Use the best model in above steps to forecast the demand for period from ”Jan-01-2015” to ”Jan-31-2015”.

    - *Data is split into train and test sets . Train data set consists of all data before 1st Jan 2015. Test data is all data after and including 1st Jan 2015.*

      Tran and Test Data set

Train Data Set, First and last three observations

timestamp	value
2014-07-01	745967
2014-07-02	733640
2014-07-03	710142

timestamp	value
2014-12-29	588023
2014-12-30	655665
2014-12-31	704941

Number of Observations in Train Data Set are : 184

Test Data Set, First and last three observations

timestamp	value
2015-01-01	690407
2015-01-02	606716
2015-01-03	722115

timestamp	value
2015-01-29	704935
2015-01-30	800478
2015-01-31	897719

Number of Observations in Test Data Set are : 31

Table 24: Train and Test Data set, demarcation date 1st Jan 2015

- *Predictions for the period of the test data*

	Actual Demand	Predictions-ARIMA Model
2015-01-01	690407	733675
2015-01-02	606716	742578
2015-01-03	722115	740928
2015-01-04	565709	736575
2015-01-05	600132	733322
2015-01-06	631877	731914
2015-01-07	707015	731752
2015-01-08	745416	732072
2015-01-09	746839	732403
2015-01-10	892664	732588
2015-01-11	718725	732639
2015-01-12	647742	732624
2015-01-13	734397	732594
2015-01-14	724099	732573
2015-01-15	739819	732564
2015-01-16	795094	732564
2015-01-17	828957	732566
2015-01-18	743123	732568
2015-01-19	575177	732569
2015-01-20	660452	732570
2015-01-21	703946	732570
2015-01-22	739376	732569
2015-01-23	782348	732569
2015-01-24	798498	732569
2015-01-25	694262	732569
2015-01-26	375311	732569
2015-01-27	232058	732569
2015-01-28	621483	732569
2015-01-29	704935	732569
2015-01-30	800478	732569
2015-01-31	897719	732569

Table 25: Predications of the ARIMA model

➢ Plot the predicted value and the true demand value from ”Jan-01-2015” to ”Jan-31-2015”.

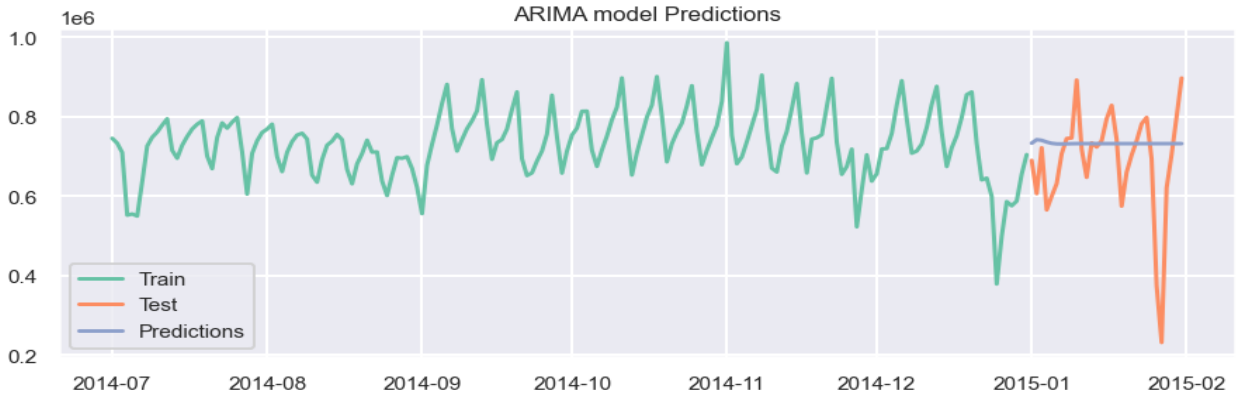

Figure 32: ARIMA model predictions

- *Evaluation of the model with performance metrics of **RMSE** and **MAPE***

```
# code to return only the values in the time series
y_true = test_df_day.values

# calculate the Root Mean Square Error
RMSE = np.sqrt(mean_squared_error(y_true,y_predicted)).astype('int')

# calculate the Mean Absolute Percenatge Error
MAPE = round(mean_absolute_percentage_error(y_true,y_predicted)*100,2)

# create a dataframe to store the results of the model with performance metrics of RMSE and MAPE
result_Arima = pd.DataFrame({'RMSE':RMSE,
                             'MAPE':MAPE},
                            index = ['Arima Model'])
result_Arima
```

	RMSE	MAPE
Arima Model	137647	18.9

Table 26: Performance Metrics of RMSE and MAPE for ARIMA model

➢ Exponential Smoothening model for forecasting for demand value from ”Jan-01-2015” to ”Jan-31-2015” based on historical data ”Jul-01-2014” to ”Dec-01-2014”.

Holt Winters Exponential Smoothening, Trend: Additive, Seasonal: Multiplicative

```
# import the libraries
from statsmodels.tsa.api import ExponentialSmoothing
```

```
# Initializing the Triple Exponential (level , trend, seasonal) Smoothing Model on the train data set

exp_TES_mul = ExponentialSmoothing(train_df_day,
                                   seasonal='mul',    # seasonal order is assumed as multiplicative
                                   trend= 'add',
                                   initialization_method='estimated')
# fit the model on the model
model_fit_mul = exp_TES_mul.fit()

# make the predictions
y_predicted = model_fit_mul.forecast(steps=len(test_df_day))
```

```
print("\033[1m" 'Holt Winters Exponential Smoothening, Trend: Additive, Seasonal: Multiplicative, Estimated Parameters')
display(model_fit_mul.params)
```

```
==Holt Winters Exponential Smoothening, Trend: Additive, Seasonal: Multiplicative, Estimated Parameters ==

{'smoothing_level': 0.7121428571428572,
 'smoothing_trend': 0.02967261904761905,
 'smoothing_seasonal': 0.2878571428571428,
 'damping_trend': nan,
 'initial_level': 608718.6285714282,
 'initial_trend': 15432.942857142909,
 'initial_seasons': array([1.01144179, 1.03785507, 1.04386587, 1.02236436, 1.02820033,
       0.93953127, 0.91674131]),
 'use_boxcox': False,
 'lamda': None,
 'remove_bias': False}
```

Code 34: Code for Holt Winters Exponential Smoothening Model , Seasonality -Multiplicative

- *Making the forecasts and adding the predictions to the **predictions data frame** for comparison*

```
# make a combined data frame with actual value and the predicted value (in whole number)
predictions_df= pd.concat([predictions_df,y_predicted.astype('int')]
                          ,axis=1).rename(columns={0:'Predictions-Holt Winters Multiplicative Model'})
predictions_df
```

Code 35: Code for making data frame for predictions Exponential Multiplicative Model

- *Predictions*

	Actual Demand	Predictions-ARIMA Model	Predictions-Holt Winters Multiplicative Model
2015-01-01	690407	733675	632272
2015-01-02	606716	742578	778798
2015-01-03	722115	740928	858320
2015-01-04	565709	736575	770119
2015-01-05	600132	733322	700846
2015-01-06	631877	731914	714619
2015-01-07	707015	731752	687090
2015-01-08	745416	732072	635454
2015-01-09	746839	732403	782715
2015-01-10	892664	732588	862633
2015-01-11	718725	732639	773987
2015-01-12	647742	732624	704363
2015-01-13	734397	732594	718203
2015-01-14	724099	732573	690533
2015-01-15	739819	732564	638636
2015-01-16	795094	732564	786631
2015-01-17	828957	732566	866947
2015-01-18	743123	732568	777854
2015-01-19	575177	732569	707880
2015-01-20	660452	732570	721786
2015-01-21	703946	732570	693976
2015-01-22	739376	732569	641818
2015-01-23	782348	732569	790548
2015-01-24	798498	732569	871261
2015-01-25	694262	732569	781722
2015-01-26	375311	732569	711397
2015-01-27	232058	732569	725370
2015-01-28	621483	732569	697419
2015-01-29	704935	732569	645000
2015-01-30	800478	732569	794465
2015-01-31	897719	732569	875574

Code 36: Predictions Exponential Multiplicative Model

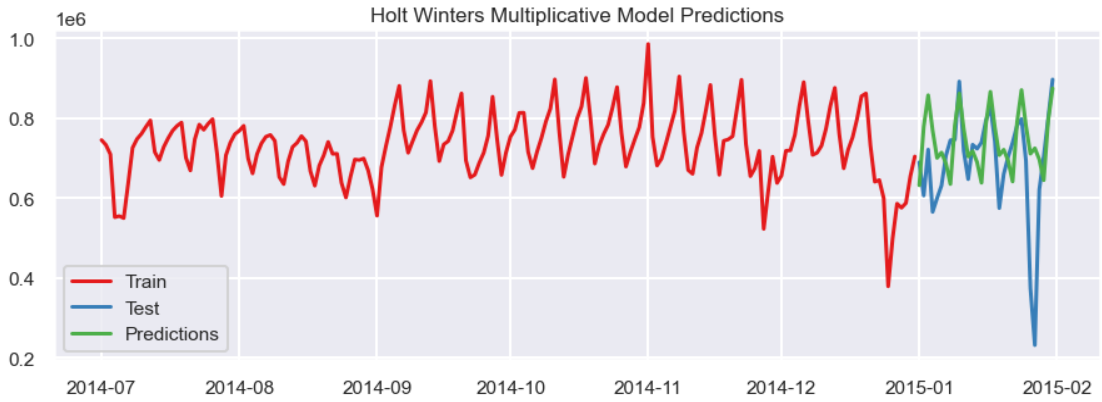

Figure 33: Plot predictions Exponential Multiplicative Model

- *Evaluation of the model with performance metrics of **RMSE** and **MAPE***

	RMSE	MAPE
Arima Model	137647.0	18.90
Holt Winters Multiplicative Model	133899.0	19.05

Table 27: Performance metrics - Exponential Multiplicative Model

Holt Winters Exponential Smoothening, Trend: Additive, Seasonal: Additive

```
exp_TES_add = ExponentialSmoothing(train_df_day,
                                   seasonal='add',
                                   trend= 'add',
                                   initialization_method='estimated')



model_fit_add = exp_TES_add.fit()


y_predicted = model_fit_add.forecast(steps=len(test_df_day))



```

```
print("\033[1m" '==Holt Winters Exponential Smoothening, Trend: Additive, Seasonal: Additive, Estimated Parameters ==')
model_fit_add.params
```

```
==Holt Winters Exponential Smoothening, Trend: Additive, Seasonal: Additive, Estimated Parameters ==

{'smoothing_level': 0.7121428571428572,
 'smoothing_trend': 0.02967261904761905,
 'smoothing_seasonal': 0.2878571428571428,
 'damping_trend': nan,
 'initial_level': 608718.6285714282,
 'initial_trend': 15432.942857142909,
 'initial_seasons': array([  6158.59285714,  26300.45      ,  30857.41428571,  17944.83571429,
        22896.45      , -42255.12142857, -61902.62142857]),
 'use_boxcox': False,
 'lamda': None,
 'remove_bias': False}
```

Code 37: Code Holt Winters Exponential Smoothening, Trend: Additive, Seasonal: Additive

- *Predictions*

	Actual Demand	Predictions-ARIMA Model	Predictions-Holt Winters Multiplicative Model	Predictions-Holt Winters Additive Model
2015-01-01	690407	733675	632272	642654
2015-01-02	606716	742578	778798	772968
2015-01-03	722115	740928	858320	843652
2015-01-04	565709	736575	770119	768248
2015-01-05	600132	733322	700846	699867
2015-01-06	631877	731914	714619	711232
2015-01-07	707015	731752	687090	687867
2015-01-08	745416	732072	635454	646942
2015-01-09	746839	732403	782715	777256
2015-01-10	892664	732588	862633	847940
2015-01-11	718725	732639	773987	772536
2015-01-12	647742	732624	704363	704155
2015-01-13	734397	732594	718203	715520
2015-01-14	724099	732573	690533	692155
2015-01-15	739819	732564	638636	651230
2015-01-16	795094	732564	786631	781544
2015-01-17	828957	732566	866947	852228
2015-01-18	743123	732568	777854	776824
2015-01-19	575177	732569	707880	708443
2015-01-20	660452	732570	721786	719808
2015-01-21	703946	732570	693976	696442
2015-01-22	739376	732569	641818	655518
2015-01-23	782348	732569	790548	785832
2015-01-24	798498	732569	871261	856516
2015-01-25	694262	732569	781722	781112
2015-01-26	375311	732569	711397	712731
2015-01-27	232058	732569	725370	724096
2015-01-28	621483	732569	697419	700730
2015-01-29	704935	732569	645000	659806
2015-01-30	800478	732569	794465	790120
2015-01-31	897719	732569	875574	860804

Table 28: Predictions - Exponential Smoothening, Trend: Additive, Seasonal: Additive

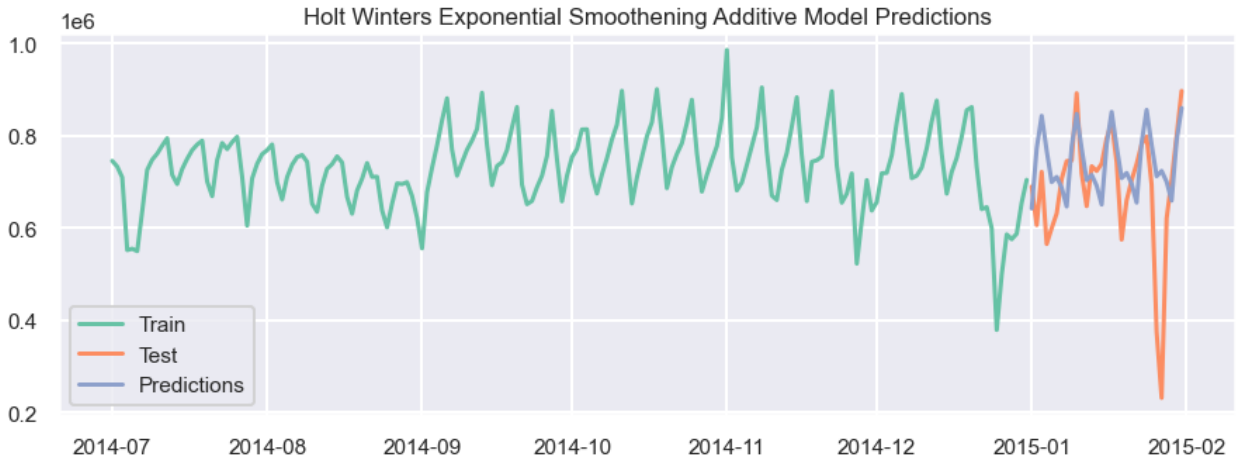

Figure 34: Predictions plot - Exponential Smoothening, Trend: Additive, Seasonal: Additive

- *Evaluation of the model with performance metrics of **RMSE** and **MAPE***

	RMSE	MAPE
Arima Model	137647.0	18.90
Holt Winters Multiplicative Model	133899.0	19.05
Holt Winters Additive Model	131632.0	18.60

Table 29: Performance Metrics - Exponential Smoothening Additive Model

Holt Winters Additive Model gives the best result

Question 2.3

Anomaly within the df_day data frame.

- Create the Weekday column according to the timestamp column in **df_day** data frame. The value in Weekday column should be from [**'Monday', 'Tuesday', 'Wednesday', 'Thursday',' Friday', 'Saturday', 'Sunday'**].

  - *Code to introduce a new column 'weekday' in the df_day data frame*

```
'''
introduce a new column in the data frame df_day by the name 'weekday' which will display numeric week days,
'0' being Monday and '6' being Sunday
'''
df_day['weekday'] = df_day.index.weekday
display(df_day.weekday.value_counts().to_frame().sort_index())
weekday_mapping = {0: 'Monday',
                   1: 'Tuesday',
                   2: 'Wednesday',
                   3: 'Thursday',
                   4: 'Friday',
                   5: 'Saturday',
                   6: 'Sunday'}

# replace numbers by weekday names as mapped above
df_day['weekday'] = df_day['weekday'].replace(weekday_mapping)

display(df_day)
```

Code 38: Code to introduce new column showing weekdays

	weekday
0	30
1	31
2	31
3	31
4	31
5	31
6	30

timestamp	value	weekday
2014-07-01	745967	Tuesday
2014-07-02	733640	Wednesday
2014-07-03	710142	Thursday
2014-07-04	552565	Friday
2014-07-05	555470	Saturday
...	...	...
2015-01-27	232058	Tuesday
2015-01-28	621483	Wednesday
2015-01-29	704935	Thursday
2015-01-30	800478	Friday
2015-01-31	897719	Saturday

215 rows × 2 columns

Table 30: df_day data frame with new column showing weekdays

- *Run a check that the numeric week days value counts match with the weekday names value counts.*

	weekday
Monday	30
Tuesday	31
Wednesday	31
Thursday	31
Friday	31
Saturday	31
Sunday	30

Table 31: Week Day names value counts of df_day

- Create the **Hour, Day, Month, Year, Month_day** (numeric format on day of the month), **Lag** (yesterday's demand value), and **Rolling Mean** (rolling 7 days mean demand value, minimized period is 1) 7 new columns in df_day data frame according to the timestamp column.

Code to introduce new columns 'Hour', 'Day', 'Month', 'Month Name', 'Year', 'Lag', 'Rolling Mean'

```
Month_mapping = {1:'January',
                 2:'February',
                 3:'March',
                 4:'April',
                 5:'May',
                 6:'June',
                 7:'July',
                 8:'August',
                 9:'September',
                10:'October',
                11:'November',
                12:'December'}


df_day['Hour']          = df_day.index.hour  # introduce column 'hour', this column will always display the hr on top - '0'
df_day['Day']           = df_day.index.day   # introduce column 'day' showing calendar dates
df_day['Month']         = df_day.index.month # introduce column 'month', which will show numeric months.1==Jan and 12==Dec
df_day['Month_Name']    = df_day['Month'].replace(Month_mapping)          # replace month numbers with month names
df_day['Year']          = df_day.index.year                               # introduce column "Year"
df_day['Lag']           = df_day['value'].shift(1)         # introduce column 'Lag' which shows the value of one day before
df_day['Rolling_Mean']  = df_day['value'].rolling(7).mean().round(2)          # introduce column 'Rolling Mean' of 7 days.

# run a check if the numeric months have been correctly replaced by month names,by counting the number of days in a month
display(pd.pivot_table(data = df_day,
                       index = ['Year','Month_Name'],
                       values= 'Day',
                       aggfunc='count',
                       sort=False)) # this will ensure that the month names come in their natural order.

display(df_day)
```

Code 39: Code to introduce columns 'Hour', 'Day', 'Month', 'Month Name', 'Year', 'Lag', 'Rolling Mean'

- *Check number of days in a month to confirm that the numeric calendar months are correctly replaced by the month names*

		Day
Year	**Month_Name**	
2014	**July**	31
	August	31
	September	30
	October	31
	November	30
	December	31
2015	**January**	31

Table 32: Count of days in a Calendar Month

- *df_day with the new columns*

timestamp	value	weekday	Hour	Day	Month	Month_Name	Year	Lag	Rolling_Mean
2014-07-01	745967	Tuesday	0	1	7	July	2014	NaN	NaN
2014-07-02	733640	Wednesday	0	2	7	July	2014	745967.0	NaN
2014-07-03	710142	Thursday	0	3	7	July	2014	733640.0	NaN
2014-07-04	552565	Friday	0	4	7	July	2014	710142.0	NaN
2014-07-05	555470	Saturday	0	5	7	July	2014	552565.0	NaN
...	...	...	...	...	...	...	...	...	...
2015-01-27	232058	Tuesday	0	27	1	January	2015	375311.0	617971.29
2015-01-28	621483	Wednesday	0	28	1	January	2015	232058.0	606190.86
2015-01-29	704935	Thursday	0	29	1	January	2015	621483.0	601270.71
2015-01-30	800478	Friday	0	30	1	January	2015	704935.0	603860.71
2015-01-31	897719	Saturday	0	31	1	January	2015	800478.0	618035.14

215 rows × 9 columns

Table 33: df_day with 7 new columns

- Using Isolation Forest with above crafted features in df_day to find out the date which is identified as 'outlier'.

- *Information on features of df_day*

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 215 entries, 2014-07-01 to 2015-01-31
Freq: D
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   value         215 non-null    int64
 1   weekday       215 non-null    object
 2   Hour          215 non-null    int64
 3   Day           215 non-null    int64
 4   Month         215 non-null    int64
 5   Month_Name    215 non-null    object
 6   Year          215 non-null    int64
 7   Lag           214 non-null    float64
```

```
 8   Rolling_Mean   209 non-null    float64
dtypes: float64(2), int64(5), object(2)
memory usage: 16.8+ KB
```

- *There are missing values 'Lag' and 'Rolling Mean'.*
- *We drop the columns with missing values , because the algorithm of 'Isolation Forest' will not be able to handle missing values. No information will be lost from the data , as these columns were in any case derived from other columns of the data frame.*

Revised Data Frame

timestamp	value	Hour	Day	Month	Year
2014-07-01	745967	0	1	7	2014
2014-07-02	733640	0	2	7	2014
2014-07-03	710142	0	3	7	2014
2014-07-04	552565	0	4	7	2014
2014-07-05	555470	0	5	7	2014
...	...	...	...	...	...
2015-01-27	232058	0	27	1	2015
2015-01-28	621483	0	28	1	2015
2015-01-29	704935	0	29	1	2015
2015-01-30	800478	0	30	1	2015
2015-01-31	897719	0	31	1	2015

215 rows × 5 columns

Table 34: df_day revised after dropping columns with missing values

- *Code to apply the Isolation Forest Algorithm to identify the outliers in df_day data set*

```
'''
Using Isolation Forest with above crafted features in df_day to find out the date which is
identified as 'outlier'.
'''
from sklearn.ensemble import IsolationForest

# Initialize the object Isolation Forest which will identify outliers at a contamination level of 5 %.
# The amount of contamination of the data set is the proportion of outliers in the data set

model = IsolationForest(contamination=0.02)  # Adjust the contamination parameter as needed
model.fit(a)                                 # fit the model

outliers = model.predict(a)                  # predict the outliers

df_day['outlier'] = outliers           # introduce a column 'outliers' - outliers are represented by '-1' and others as '1'

df_day_outlier = df_day[(df_day['outlier'] < 0)] # view the data frame containing only the outliers

display(df_day_outlier)

```

Code 40: Code for Isolation Forest algorithm to identify outliers

- *The set of data points identified as outliers by the Isolation Forest Algorithm at 2% Contamination*

timestamp	value	weekday	Hour	Day	Month	Month_Name	Year	Lag	Rolling_Mean	outlier
2014-12-25	379302	Thursday	0	25	12	December	2014	600096.0	673910.43	-1
2015-01-10	892664	Saturday	0	10	1	January	2015	746839.0	698521.71	-1
2015-01-26	375311	Monday	0	26	1	January	2015	694262.0	679170.43	-1
2015-01-27	232058	Tuesday	0	27	1	January	2015	375311.0	617971.29	-1
2015-01-31	897719	Saturday	0	31	1	January	2015	800478.0	618035.14	-1

Table 35: Outliers in the data set

- *Count of outliers in the data set*

	outlier
1	210
-1	5

Table 36: Count of outliers in the data set

- *There are a total of 5 outliers identified by the Isolation Forest algorithm in the data set comprising of 215 data points. We shall visualize these outliers in a scatter plot for better understanding.*

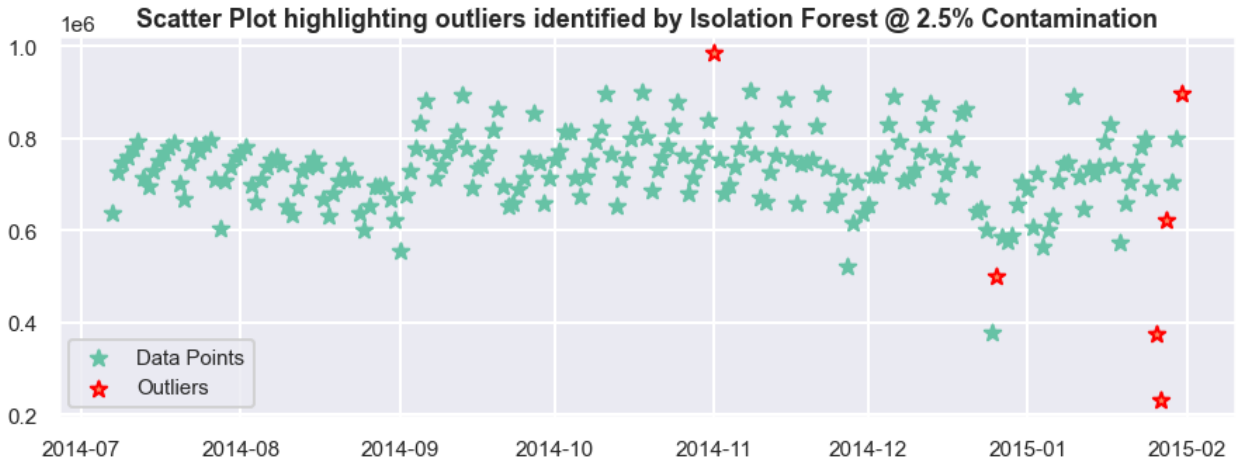

Figure 35: Plot for the outliers

The Isolation Forest algorithm has identified the outliers as is evidenced in the plot above.

*********** End of Report***********

Retrospection

1) What did you have learned from your team members from the second assignment?

- The way we approached this assignment was to individually tackle the whole assignment first without any discussion among the group members.
- This allowed each of us to understand what is asked and tackle it based on our own research and approach.
- Next, we shared our attempts on a google drive and each of us went through each other's work and the approaches used.
- Some of us resorted to straight line codes and some used functions and classes to tackle the questions.
- The learning here was the same solution can be arrived at through multiple ways and we learnt those multiple ways by assessing and discussing each other's work.
- Next, we got into a series of code review sessions and connects through zoom meetings, and WhatsApp Calls, explaining to each other the pros and cons of the approach taken by each of us.
- We then narrowed down the approach in the final code book based on 3 factors.

1) Simplicity and Maintainability

2) Re-usability

3) Efficiency.

2) What is the contribution of each team member for finishing the second assignment?

- Everyone worked on the entire assessment.
- In terms of code book preparation, Suraj handled and explained 1.1 to 1.5 in Part I,
- Prince handled and explained 1.6 to 1.10 in Part I and
- Shailesh handled and explained 2.1 to 2.3 (Time Series portion).
- In terms of the report preparation Shailesh took up the overall consolidation and then modification and templatization was done by both Prince and Suraj. Video again was a collective effort each one explaining the portion that was allocated.

References

Videos

1) Krish Naik (25 Feb 2022) 'Live Day 1 – Exploratory Data Analysis and Stock Analysis with Time Series Data', YouTube, accessed 2nd October 2023, Link
2) Nachiketa Hebbar (5 Sep 2022) 'ARIMA Model in Python | Time Series Forecasting #6, YouTube, accessed 3rd October 2023, Link

Course Material

1) Deakin Study Material (n.d) 'Week 2 Data Manipulation', Modern Data Science SIT742, Deakin University
2) Deakin Study Material (n.d) 'Week 4 Data Analytics 1 Time Series, Modern Data Science SIT742, Deakin University
3) Deakin Study Material (n.d) 'Week 4 Data Analytics 2, Isolation Forest, Modern Data Science SIT742, Deakin University
4) UT Austin Study Material (n.d) 'Week 3 ARIMA Models' Time Series Forecasting Course, UT Austin

Websites – Articles

1) Akshara_416 (13 September 2023), Anomaly detection using Isolation Forest – A Complete Guide, Analytics Vidhya , accessed 1st October 2023.
2) Amol Mavuduru (24 Nov 2021), How to perform anomaly detection with the Isolation Forest algorithm, Medium , accessed 30th September 2023.