FIT5196 Assessment 2

Student Name: DISHI JAIN

Student ID: 30759307

Libraries used:

- import pandas as pd (for accessing DataFrames)
- import ast (for ast.literal_eval)
- import nltk (for sentiment intensity analyer)
- import nltk.sentiment.vader as va (for sentiment intensity analyer)
- import numpy as np (for np.where)
- nltk.download('vader_lexicon') (for sentiment intensity analyer)

1. Introduction

This assignment is focused on data cleansing and finding outliers in the data. Here the data represents orders places in Australia in different stores. We need to clean the data and fix any errors that it may have. Also we need to find the missing data and impute it from any other attribute that we can. We also need to detect and remove outliers from the data.

2. Importing Libraries

In [ ]:
```
import pandas as pd
import ast
import nltk
import nltk.sentiment.vader as va
import numpy as np
from math import sin, cos, sqrt, atan2, radians
from sklearn.linear_model import LinearRegression

nltk.download('vader_lexicon')
```

3. Analysing Dirty Data

In this task we have to find errors in the file and clean or fix those errors. These errors can be in any form example in date column, diatancce column, etc. We need to find and fix such errors.

In [ ]:
```
#reading dirty_data csv file
dirty_data = pd.read_csv("30759307_dirty_data.csv")
outlier_data = pd.read_csv("30759307_outlier_data.csv")
```

In [ ]:
```
#finding the shape of the data and displaying its few top records
print (dirty_data.shape)
dirty_data.head(10)
#get order of column
cols = dirty_data.columns.tolist()
```

In [ ]:
```
#getting information about the data in terms of data type of columns and count of
dirty_data.info()
```

In [ ]:
```
#describing the numerical data to get the count,min,max and some other attributes
dirty_data.describe()
```

From the above .describe() we can see that all the numerical columns have 500 entries. Hence there are no missing entries present. We can also see that the maximum value in the column distance_to_nearest_warehouse is approximately 5 which is very far away from the mean. So this could be an error. Also looking at the customer_lat and customer_long columns we can see that there exists and error as for this location the latitdes are negative and longitudes are positive.

In [ ]:
```
#describing the caegorical data to get the count,freq,unique and some other attri

dirty_data.describe(include=['O'])
```

From the above .describe() we can see that all the categorical columns have 500 entries. Hence there are no missing entries present. We can also see that there are 500 unique order_ids hence no errors exist in that. There are 492 unique customer_ids which is also okay as a customer may have multiple orders. We can also see that season has 8 unique entries which is wrong as there are 4 known seasons to us. Also we know that there are only 3 warehouses data but in the nearest_warehouse column we can see 6 unique entries. Hence this is also an error.

3.1 Checking the duplicated values in order_id column

In [ ]:
```
dirty_data[dirty_data.duplicated(['order_id'])]
```

Hence there are no duplicate order_id.

3.2 Now checking the duplicate values in customer_id column

In [ ]:
```
dirty_data[dirty_data.duplicated(['customer_id'],keep=False)].sort_values(by='cus
```

From the above output we can see that there are duplicate records for the customers. This is valid as a customer can order multple times from the company. However we find that in the latitude and longitude columns there are wrong values in some rows. This will be corrected later.

3.3 Checking for date column's format

In [ ]:
```
#creating another dataframe which contains three separate columns of year,month,D
coldate = dirty_data['date'].apply(lambda s: pd.Series({'year': s.split('-')[0],
                            'month':s.split('-')[1], 'Date':s.split('-')[2]}))
coldate = coldate.astype(int)
```

In [ ]:
```
#finding the month entries which are greater than 12
coldate[(coldate.month > 12)]
```

Hence there are months that are greater than 12. This is invalid as a month entry cannot be greater than 12.

Now I'll check if there are any rows with month and date value both greater than 12.

In [ ]:
```
coldate[(coldate.month > 12) & (coldate.Date > 12)]
```

Hence there are no months and Date columns that are greater than 12. If we would have received a row where both of them would have been gretaer than 12 then we wouldn't be able to handle it.

Example, If a row was like 2019-14-17, then we wouldn't know the month of the date at all. Hence we would have to handle it differently.

For us however we do not have such scenario where both month and Date columns have value greater than 12. Hence we can simply check for values in month and Date columns. Where I believe month contains values of Dates and Dates contain values of month.

In [ ]:
```
new_error_a = coldate[(coldate.month > 12) & (coldate.Date <= 12)]
new_error_a
```

In these rows we can simply interchange the month value and Date value.

In [ ]:
```
#interchanging or swapping the month and date values
coldate[['month','Date']] = coldate[['Date','month']].where((coldate.month > 12)
```

In [ ]:
```
coldate[(coldate.month > 12) & (coldate.Date <= 12)]
```

Hence now there are no values where month is greater than 12

In [ ]:
```
#checking again for inccorect entries
coldate[(coldate.month < 12) & (coldate.Date < 12)]
```

Hence for such a scenario we assume that the month is correctly placed.

In [ ]:
```
coldate = coldate.astype(str)
```

In [ ]:
```
#finding errors where year column has values of length less than 4 or date or mor
new_error_b = coldate[(coldate.year.str.len() < 4) | (coldate.month.str.len() > 2
new_error_b
```

As the desired format is yyyy-mm-dd, hence for the above we can swap the values where Date column has years and year column has dates.

In [ ]:
```
#swapping date and year values
coldate[['year','Date']] = coldate[['Date','year']].where((coldate.year.str.len()
```

In [ ]:
```
#checking again for inccorect entries
coldate[(coldate.year.str.len() < 4) | (coldate.month.str.len() > 2) | (coldate.D
```

Now no incorrect dates exists in the data. Hence we can combine and write back the dates back to the file

In [ ]:
```
#writing the corrected dates back to the column
dirty_data['date'] = coldate['year'].astype(str) + '-' + coldate['month'].astype(
```

Hence now all date values are correct in the original data frame dirty_data

3.4 Now checking the nearest warehouse column

In [ ]:
```
dirty_data.nearest_warehouse.value_counts()
```

Hence we see above that there are -

6 entries of thompson and 193 entries of Thompson.

10 entries of nickolson and 184 entires of Nickolson.

6 entries of bakers and 101 entries of Bakers

Hence we need to correct this error as thompson and Thompson means the same. Similarly for nickolson and Nickolson and similarly for bakers and Bakers.

We can do this by using the replce option.

In [ ]:
```
new_error_c = dirty_data[dirty_data.nearest_warehouse.isin(['bakers','nickolson',
new_error_c
```

In [ ]:
```
#replacing incorrect entries with correct ones
dirty_data.nearest_warehouse.replace({"bakers": "Bakers", "nickolson": "Nickolsor
```

In [ ]:
```
#checking the unique values again
dirty_data.nearest_warehouse.value_counts()
```

Hence now the values are corrected.

In [ ]:

3.5 Checking for the item names in the shopping cart column. There must be 10 unique items as given to us

In [ ]:
```
item_list = []
for i in dirty_data.shopping_cart:
    for j in ast.literal_eval(i):
        item_list = item_list + [j[0]]

item_set = set(item_list)
print(item_set)
print(len(item_set))
```

Hence there are 10 unique items

3.6 Now to check for the column order_price, we can check for negative values. To do this we can do the following

In [ ]:
```
dirty_data[dirty_data['order_price'] < 0]
```

3.7 Now analysing the column delivery_charges

In [ ]:
```
dirty_data[dirty_data['delivery_charges'] < 0]
```

3.8 Analysing longitude and latitude columns, we know that a latitude should be negative and longitude should be positive. (in this case)

In [ ]:
```
dirty_data[['customer_lat','customer_long']].head()
```

In [ ]:
```
#checking the case when latitude > longitude
dirty_data[['customer_lat','customer_long']].where(dirty_data.customer_lat > dirt
```

Hence there are 27 wrong entries in the data. To correct this we can swap the values

In [ ]:
```
new_error_d = dirty_data[(dirty_data.customer_lat > dirty_data.customer_long)]
new_error_d
```

In [ ]:
```
#swapping the incorrect lat,long values
dirty_data[['customer_lat','customer_long']] = dirty_data[['customer_long','custo
```

In [ ]:
```
#checking again for the error
dirty_data[['customer_lat','customer_long']].where(dirty_data.customer_lat > dirt
```

Hence now the values are correct.

3.9 Checking for negative values in coupon discount

In [ ]:
```
dirty_data[dirty_data['coupon_discount'] < 0]
```

Hence no negative coupon discounts are there. Everything is correct.

3.10 Checking for negative values in order_total

In [ ]:
```
dirty_data[dirty_data['order_total'] < 0]
```

Hence no negative order_total are there. Everything is correct.

3.11 Checking for Season values

In [ ]:
```
dirty_data.season.value_counts()
```

Hence there are incorrect values of winter,summer,autumn,spring. I will convert these into Winter,Summer,Autumn,Spring

In [ ]:
```
new_error_e = dirty_data[dirty_data.season.isin(['winter','autumn','summer','spri
new_error_e
```

In [ ]:
```
#replacing incorrect values with correct ones in season column
dirty_data.season.replace({"winter": "Winter", "autumn": "Autumn", "summer":"Sumn
```

In [ ]:
```
dirty_data.season.value_counts()
```

Now the values are correct.

3.12 Checking for is_expedited_delivery values

In [ ]:
```

```

In [ ]:
```
dirty_data.is_expedited_delivery.value_counts()
```

Hence only True and False values exists. This is correct

3.13 Checking for distance_to_nearest_warehouse values

In [ ]:
```
#findng negative values in the column
dirty_data[dirty_data['distance_to_nearest_warehouse'] < 0]
```

Hence no negative values exist in the column.

3.13 Checking for latest_customer_review NA values

In [ ]:
```
dirty_data['latest_customer_review'].isna().sum()
```

Hence no null values exist.

3.14 Checking for is_happy_customer NA values

In [ ]:
```
dirty_data['is_happy_customer'].isna().sum()
```

Hence no null values exist.

3.15 Checking for is_happy_customer NA values

In [ ]:
```
dirty_data.is_happy_customer.value_counts()
```

Hence no null values exist.

3.16 Now to check the sentiments i.e. happy customer or not

In [ ]:
```
#object of SentimentIntensityAnalyzer
sentiment_analyzer = va.SentimentIntensityAnalyzer()
```

In [ ]:
```

```

In [ ]:
```
#creating a separate dataframe that contains the polarity score as analyzed by th
is_happy_data = dirty_data['latest_customer_review'].apply(lambda x: pd.Series({'
```

In [ ]:
```
type(is_happy_data)
```

In [ ]:
```
is_happy_data.head()
```

In [ ]:
```
#if the sentiment score is >= 0.05 then customer is happy else she/he is sad
is_happy_data['new_ishappy_customer'] = np.where(is_happy_data['new_sentiment_sco
```

In [ ]:
```
is_happy_data.head()
```

In [ ]:
```

```

In [ ]:
```
#comparing the actual is_happy_customer column with the analyzed one
pd.crosstab(is_happy_data["new_ishappy_customer"], dirty_data["is_happy_customer"
```

Hence there are

130 instances where dirty_data file had False as happy_customer value and the customer was not happy. Hence correct data.

342 instances where dirty_data file had True as happy_customer value and the customer was happy. Hence correct data.

6 instances where dirty_data file had True as happy_customer value but the customer was not happy. Hence incorrect data.

22 instances where dirty_data file had False as happy_customer value but the customer was actually happy. Hence incorrect data.

In [ ]:
```
new_error_i = is_happy_data[is_happy_data["new_ishappy_customer"] != dirty_data['
new_error_i
```

In [ ]:
```

```

Hence the correct values determined by the sentiment intensity solver are used in the dirty_data file

In [ ]:
```
#using the analyzed sentiments by SentimentIntensityAnalyzer as the correct colum
dirty_data['is_happy_customer'] = is_happy_data['new_ishappy_customer']
```

In [ ]:
```
#comparing again using crosstab
pd.crosstab(is_happy_data["new_ishappy_customer"], dirty_data["is_happy_customer"
```

In [ ]:
```

```

3.17 Checking date and seasons whether the seasons are correct based on the date

We know,

Spring - September,October,November

Autumn - March,April,May

Summer - December,January,February

Winter - June,July,August

Hence we can check whether the season column in dirty_data is correct or not

In [ ]:

```

```

In [ ]:

```
#converting the dates to the desired format
dirty_data['date'] = pd.to_datetime(dirty_data['date'], format='%Y-%m-%d')
```

In [ ]:

```
dirty_data.dtypes
```

In [ ]:

```
#finding only the months from the date to compare with seasons later on
dirty_data['month_only'] = dirty_data['date'].dt.month
```

In [ ]:

```
# dirty_data['new_season'] = np.where(dirty_data['month_only'] >= 0.05 , 'True' ,
```

In [ ]:

```
#creating new column new_season to calculate correct seasons
dirty_data['new_season'] = np.where((dirty_data.month_only < 6) & (dirty_data.mon
                           np.where((dirty_data.month_only < 9) & (dirty_data.mon
                           np.where((dirty_data.month_only < 12) & (dirty_data.mo
```

In [ ]:

```
#comparing new_season and original one
pd.crosstab(dirty_data["season"], dirty_data["new_season"])
```

Hence, there is some incorrect data.

1 instance where dirty_data had season as Autumn but the actual season was Spring

2 instances where dirty_data had season as Autumn but the actual season was Summer

2 instances where dirty_data had season as Autumn but the actual season was Winter

2 instances where dirty_data had season as Spring but the actual season was Autumn

1 instance where dirty_data had season as Spring but the actual season was Summer

1 instance where dirty_data had season as Summer but the actual season was Spring

3 instances where dirty_data had season as Winter but the actual season was Autumn

1 instance where dirty_data had season as Winter but the actual season was Spring

3 instances where dirty_data had season as Winter but the actual season was Summer

Hence, we can use the correct season column i.e. new_season as the correct season column and drop the original season column as it contains incorrect values

In [ ]:
```
new_error_f = dirty_data[dirty_data.season != dirty_data.new_season]
new_error_f
```

In [ ]:
```
#dropping the season column original one and using the new calculated one
dirty_data.drop('season',axis = 1 ,inplace = True)
dirty_data.drop('month_only', axis = 1,inplace=True)
dirty_data.rename(columns={'new_season': 'season'}, inplace=True)
```

In [ ]:
```

```

3.18 Now finding whether the nearest warehouse column is correct or not

In [ ]:
```
warehouse_data = pd.read_csv("warehouses.csv")
warehouse_data.set_index('names', inplace=True)
```

In [ ]:
```
#reference from www.stackoverflow.com
#function to find distance between two locations and respective nearest warehouse
def find_nearest_warehouse(lat1,lon1):

    # given radius of earth in km
    R = 6378.0

    #converting lat long values to radian
    lat1 = radians(lat1)
    lon1 = radians(lon1)
    #converting lat long values of three warehouses to radian
    nick_lat = radians(warehouse_data.loc['Nickolson','lat'])
    nick_lon = radians(warehouse_data.loc['Nickolson','lon'])
    thomp_lat = radians(warehouse_data.loc['Thompson','lat'])
    thomp_lon = radians(warehouse_data.loc['Thompson','lon'])
    baker_lat = radians(warehouse_data.loc['Bakers','lat'])
    baker_lon = radians(warehouse_data.loc['Bakers','lon'])

    dlon_nic = nick_lon - lon1
    dlat_nic = nick_lat - lat1

    a = sin(dlat_nic / 2)**2 + cos(lat1) * cos(nick_lat) * sin(dlon_nic / 2)**2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))

    #variable to store distance from given point to nickolson warehouse
    distance_nic = R * c


    dlon_thomp = thomp_lon - lon1
    dlat_thomp = thomp_lat - lat1

    a = sin(dlat_thomp / 2)**2 + cos(lat1) * cos(thomp_lat) * sin(dlon_thomp / 2)
    c = 2 * atan2(sqrt(a), sqrt(1 - a))

    #variable to store distance from given point to thompson warehouse
    distance_thomp = R * c


    dlon_baker = baker_lon - lon1
    dlat_baker = baker_lat - lat1

    a = sin(dlat_baker / 2)**2 + cos(lat1) * cos(baker_lat) * sin(dlon_baker / 2)
    c = 2 * atan2(sqrt(a), sqrt(1 - a))

    #variable to store distance from given point to bakers warehouse
    distance_baker = R * c

    warehouse_dict = {'Nickolson':distance_nic,'Thompson':distance_thomp,'Bakers'
    warehouse_name = min(warehouse_dict, key = lambda k : warehouse_dict[k])
    distance_value = min(warehouse_dict.values())
    #returning the nearest warehouse name and distance
    return warehouse_name,distance_value
```

In [ ]:
```
#calculating the new warehouse distance and new nearest warehouse values
dirty_data['new_nearest_warehouse'],dirty_data['new_distance_warehouse'] = zip(*d
```

In [ ]:
```
dirty_data.head()
```

Firstly looking at Nearest Warehouse original and new data

In [ ]:
```
pd.crosstab(dirty_data["nearest_warehouse"], dirty_data["new_nearest_warehouse"])
```

Hence, there is some incorrect data.

2 instances where dirty_data had nearest_warehouse as Bakers but the actual nearest_warehouse was Nickolson

6 instances where dirty_data had nearest_warehouse as Bakers but the actual nearest_warehouse was Thompson

2 instances where dirty_data had nearest_warehouse as Nickolson but the actual nearest_warehouse was Bakers

6 instances where dirty_data had nearest_warehouse as Nickolson but the actual nearest_warehouse was Thompson

2 instances where dirty_data had nearest_warehouse as Thompson but the actual nearest_warehouse was Bakers

2 instances where dirty_data had nearest_warehouse as Thompson but the actual nearest_warehouse was Nickolson

Hence, we can use the correct nearest_warehouse column i.e. new_nearest_warehouse as the correct column and drop the original nearest_warehouse column as it contains incorrect values

In [ ]:
```
new_error_g = dirty_data[dirty_data['nearest_warehouse'] != dirty_data['new_neare
new_error_g
```

In [ ]:
```
#dropping the original column and using the new calculated one
dirty_data.drop('nearest_warehouse',axis = 1 ,inplace = True)
dirty_data.rename(columns={'new_nearest_warehouse': 'nearest_warehouse'}, inplace
```

Now looking at original and new nearest_distance_warehouse. We can observe that the new_distance_warehouse has numbers to larger decimal places. Hence for correct comparison we can round this column to 4 places as in the same format of distance_to_nearest_warehouse

In [ ]:
```
dirty_data['new_distance_warehouse'] = dirty_data['new_distance_warehouse'].round
```

In [ ]:
```
check_distance_nearest = dirty_data['distance_to_nearest_warehouse'] == dirty_dat

check_distance_nearest.value_counts()
```

Hence there are 33 values that are incorrect in the column distance_to_nearest_warehouse.

To get the correct data we can simply use the new column new_distance_warehouse as it contains the correct values.

In [ ]:
```
new_error_h = dirty_data[dirty_data['distance_to_nearest_warehouse'] != dirty_dat
new_error_h
```

In [ ]:
```
#dropping the original column and using the new calculated one

dirty_data.drop('distance_to_nearest_warehouse',axis = 1 ,inplace = True)
dirty_data.rename(columns={'new_distance_warehouse': 'distance_to_nearest_wareho
```

3.19 ORDER PRICE

We can calculate the order price by findng the price of each item in the cart. This is done by using the shopping cart column and the quantity column.

Firstly I have created a function find_df which will create columns that will include the quantity , items only and the length and frequency of each item tuple. This will be used in linear algebra to solve for the price of each individual product.

Next I have ceated a function that is used to simply find the solution of the equatons created by np.array(). The linear algebra multiplies the matrices where first matrix is the quantities of the product and the second matrix is the given price of the cart. Using this the price of each item can be calculated. Using this I have reated a dicttionary finally that will store the product as the key and its price as the value

In the next function I have used the calculated prices and the carts to get the new column new_order_price. This will be used to find the correct order price and hence can be used to compare the given dirty data and the calculated data for order price.

In [ ]:
```
dirty_data.head()
```

In [ ]:
```
from ast import literal_eval
def find_df(check_df):
    df = check_df.copy()
    #df = check_df.filter(['shopping_cart'])

    #converrting the string literals of shopping cart to lists
    df['shopping_cart'] = df['shopping_cart'].apply(lambda x: literal_eval(str(x)
    df['shopping_cart'] = df['shopping_cart'].apply(sorted)
    #df['order_price'] = check_df.filter(['order_price'])

    #list will store dictionaries, where key will be item and value will be quant
    new_list = []
    for i in df.shopping_cart:
        new_dict = {}
        for j in i:
            new_dict[j[0]] = j[1]
        new_list.append(new_dict)


    #column to store just the items
    df['items_only'] = ''

    #column to store just the quantities
    df['quantity_only'] = ''

    #column to store just the length of the cart
    df['length'] = ''

    for index,i in enumerate(new_list):

        temp_item_list = []
        temp_quantity_list = []
        for j,k in i.items():
            temp_item_list.append(j)
            temp_quantity_list.append(k)

        df.at[index,'items_only'] = temp_item_list
        df.at[index,'quantity_only'] = temp_quantity_list
        df.at[index,'length'] = len(temp_item_list)


    df['items_only'] = df['items_only'].apply(lambda x :tuple(x))

    #column to store frequency of the repeated items in the tuple
    df['occurence'] = df['items_only'].apply(lambda x: (df['items_only'] == x).su

    return df



def find_price(df):

    #selecting only those rows where length is equal to occurence as it will be u
    df = df[df.length == df.occurence]

    occurence_val = df.occurence.value_counts().index.tolist()
```

```
    occurence_occurence = df.occurence.value_counts().tolist()




    f_dic = {}
    for i in occurence_val:
        find_item = []
        new_df = df[df.occurence == i]
        find_item = new_df.items_only.value_counts().index.to_list()
        for j in find_item:
            try:
                #coefficients of quantities
                cal_a = np.array(list(new_df.quantity_only[new_df.items_only == j

                #coefficients of price
                cal_b = np.array(list(new_df.order_price[new_df.items_only == j])
                cal_price = np.linalg.solve(cal_a,cal_b)

                #dctionary to store the item name and price
                f_dic.update(dict(zip(new_df['items_only'][new_df['items_only']==

            except:
                continue

    return f_dic



def fill_new_column(df,f_dic):

    for i in df.index:
        item_tuple = df.at[i,'items_only']
        quant_list = df.at[i,'quantity_only']
        for k in item_tuple:
            try:
                #filling the calculated order price new one
                df.at[i,'new_order_price'] = df.at[i,'new_order_price'] + f_dic[k
            except:
                continue
    return df
```

Here I have used the outlier_dataset to calculate the price of the items as it will not have any incorect order_price values. Hence I have used it here

In [ ]:
```
#using the outlier dataset to calculate the price of items
out_find_price = find_df(outlier_data)
dic_price_final = find_price(out_find_price)
```

In [ ]:
```
out_find_price
```

In [ ]:
```
data_find_price = find_df(dirty_data)
data_find_price['new_order_price'] = 0
data_find_price = fill_new_column(data_find_price,dic_price_final)
```

In [ ]:
```
data_find_price.head()
```

In [ ]:
```
len(data_find_price)
```

In [ ]:
```
data_find_price['order_price'][data_find_price['order_price'] != data_find_price[
```

Hence 54 times we can say that the order price calculated is wrong

To correct this error, we can use the new_order_price column and drop the original one.

In [ ]:
```
#dropping the original column and using the new calculated one

data_find_price.drop('order_price',axis = 1 ,inplace = True)
data_find_price.rename(columns={'new_order_price': 'order_price'}, inplace=True)
```

In [ ]:
```
len(data_find_price)
```

3.20 Checking for order total

Now we can check whether the calculation of order_price, discount and delivery charge was done correctly or not.

To do so I have used the concept -

Order Total = Order Price - Coupon Discount(in %) + Delivery Charge

In [ ]:
```
data_find_price[['order_price','coupon_discount','delivery_charges','order_total'
```

In [ ]:
```
#creating new order_total column with correct values
data_find_price['new_order_total'] = data_find_price['order_price'] - ((data_finc
```

In [ ]:
```
data_find_price.head()
```

In [ ]:
```
data_find_price[data_find_price['order_total'] != data_find_price['new_order_tota
```

In [ ]:
```

```

Hence there are wrongly calculated values in the order_total. The correct values are present in the new_order_total column which have been calculated above.

Now simply, we can use the new_order_total column as the order_total column and can drop the original order_total column with wrong values.

Hence the values have been corrected now

In [ ]:
```
data_find_price.drop('order_total',axis = 1 ,inplace = True)
data_find_price.rename(columns={'new_order_total': 'order_total'}, inplace=True)
```

In [ ]:
```
data_find_price.head()
```

In [ ]:
```
len(data_find_price)
```

In [ ]:

In [ ]:

3.19 changing the order of the columns as they were originally

In [ ]:
```
data_find_price = data_find_price[cols]
import datetime
```

3.20 changing the data types of the columns as they were originally

In [ ]:
```
data_find_price.info()
```

In [ ]:
```
data_find_price.date = data_find_price.date.astype(str)
```

In [ ]:
```
data_find_price.info()
```

Writing the data to output file

In [ ]:
```
data_find_price.to_csv('30759307_dirty_data_solution.csv',index=False)
```

4. MISSING DATA ANALYSIS

In this task we are required to find the missing data and impute the missing values based on other related columns.

In [ ]:
```
#reading missing data file
missing_data = pd.read_csv("30759307_missing_data.csv")
```

In [ ]:
```
#information about the dataframe
missing_data.info()
```

In [ ]:
```
cols2 = missing_data.columns.tolist()
```

In [ ]:
```
#finding the count of missing values in each column
missing_data.isnull().sum()
```

4.1 The missing nearest_warehouse and the distance_to_nearest_warehouse can be imputed by using the latitude and longitude values of the customer and the location of the warehouses.

In [ ]:
```
#checking the null distance values
missing_data[missing_data['distance_to_nearest_warehouse'].isnull()].head()
```

In [ ]:
```
#checking the null warehouse name values
missing_warehou = missing_data[missing_data['nearest_warehouse'].isnull()]

missing_data = missing_data.drop(missing_warehou.index)

#creating two new columns for the missing data about warehouse and distance to ne
missing_warehou['new_nearest_warehouse'],missing_warehou['new_distance_warehouse'


#deleting the incorrect column
del missing_warehou['nearest_warehouse']

del missing_warehou['distance_to_nearest_warehouse']

missing_warehou.rename({'new_nearest_warehouse':'nearest_warehouse','new_distance
```

In [ ]:
```
#merging the two dataframes back together
missing_data = pd.concat([missing_data,missing_warehou],sort=False)
```

In [ ]:

In [ ]:

Hence now checking for the nearest warehouse and the distance to nearest warehouse, we can check null values in the new generated columns above

In [ ]:
```
missing_data.distance_to_nearest_warehouse = missing_data.distance_to_nearest_war
```

In [ ]:
```
missing_data[missing_data['nearest_warehouse'].isnull()].head()
```

In [ ]:
```
missing_data[missing_data['distance_to_nearest_warehouse'].isnull()].head()
```

Hence no null values exist in these columns.

Hence we can use these new columns as the correct columns.

In [ ]:

In [ ]:

In [ ]:
```
#again checking the sum of rows with mssing values in the dataframe
missing_data.isnull().sum()
```

4.2 Now to calculate the missing order_prices from the functions used in Dirty Data Analysis part

Now to calculate the missing values of the order_prices we can use the functions created in Dirty Data Analysis task

In [ ]:
```

data_miss_price = find_df(missing_data)
data_miss_price['new_order_price'] = 0
missing_data = fill_new_column(data_miss_price,dic_price_final)
```

In [ ]:
```
#again checking the sum of rows with mssing values in the dataframe

missing_data.isnull().sum()
```

In [ ]:
```


missing_data.drop('order_price',axis = 1 ,inplace = True)
missing_data.rename(columns={'new_order_price': 'order_price'}, inplace=True)
```

Hence the new_order_price contains no null values. So we can simply use that as the new column.

In [ ]:
```
#again checking the sum of rows with mssing values in the dataframe

missing_data.isnull().sum()
```

4.3 To find the missing delivery charges we can use the equation below-

order_total = order_price - ((coupon_discount*order_price)/100) + delivery_charges

Hence,

delivery_charges = order_total - order_price + ((coupon_discount*order_price)/100)

In [ ]:
```
# #filling the column with the correct delivery charges
missing_data['delivery_charges'] = missing_data['order_total'] - missing_data['or
```

In [ ]:
```
#again checking the sum of rows with mssing values in the dataframe

missing_data.isnull().sum()
```

4.4 Now to calculate the missing values of the order total, we can use the updated order prices, coupon discount and delivery charges.

In [ ]:
```
miss_order = missing_data[missing_data['order_total'].isnull()].head()
```

In [ ]:
```
#creating new order_total column with correct values
miss_order['new_order_total'] = miss_order['order_price'] - ((miss_order['coupon_
```

In [ ]:
```
missing_data = missing_data.drop(miss_order.index)
```

In [ ]:
```
#dropping the original incorrect column and using the new calculated column value
miss_order.drop('order_total',axis = 1 ,inplace = True)
miss_order.rename(columns={'new_order_total': 'order_total'}, inplace=True)
```

In [ ]:
```
missing_data = pd.concat([missing_data,miss_order],sort=False)
```

In [ ]:
```
#again checking the sum of rows with mssing values in the dataframe

missing_data.isnull().sum()
```

4.5 Now to find the missing values of is_happy_customer column, we can use the Sentiment Intensity Analysis approach as used above.

In [ ]:
```
#finding null is_happy_customer values
missing_data[missing_data['is_happy_customer'].isnull()].head()
```

In [ ]:
```
#finding the correct sentiments of the customer based on the reviews
find_happy_data = missing_data['latest_customer_review'].apply(lambda x: pd.Serie

#customer is happy so True is polarity score >= 0.5 else False
find_happy_data['new_ishappy_customer'] = np.where(find_happy_data['new_sentiment
```

In [ ]:
```
#using this new is_happy_customer as the final column
missing_data['is_happy_customer'] = find_happy_data['new_ishappy_customer']
```

In [ ]:
```
missing_data[missing_data['is_happy_customer'].isnull()].head()
```

Hence the column has been imputed with the correct values based on the reviews of the customer

In [ ]:
```
#again checking the sum of rows with mssing values in the dataframe

missing_data.isnull().sum()
```

4.6 Changing the order of the columns as they were originally

In [ ]:
```
missing_data = missing_data[cols2]
```

In [ ]:
```
missing_data.info()
```

Writing the file back to a csv

In [ ]:
```
missing_data.to_csv('30759307_missing_data_solution.csv',index=False)
```

In [ ]:
```
```

5. OUTLIER DATA ANALYSIS

In [ ]:
```
outlier_data = pd.read_csv("30759307_outlier_data.csv")
```

In [ ]:
```
outlier_data.head()
```

In [ ]:
```
outlier_data.boxplot('delivery_charges', figsize=(10, 30))
```

However it is not good to just determine the outliers based on its values. We can identify outliers based on the columns that it depends upon. These are season, distance_to_nearest_warehouse, whether the customer wants an expedited delivery and whether the customer was happy or not with previous order.

To find the relationship with these, we can draw a boxplot based on these column values.

In [ ]:
```
outlier_data.boxplot('delivery_charges', by = ['season'] , figsize=(20, 20))
```

Hence we can see that the Winter column has outliers above approximately 100. So if we remove all outliers in the data above 100 then we would lose a large portion of the data for the seasons spring and summer. Hence it is not a good approach to simply remove the outliers based on season.

In [ ]:
```
outlier_data.boxplot('delivery_charges', by = ['is_expedited_delivery'] , figsize
```

Again by just observing and removing outliers from the column is_expedited_delivery is not a good approach

In [ ]:
```
outlier_data.boxplot('delivery_charges', by = ['is_happy_customer'] , figsize=(2(
```

In [ ]:
```
outlier_data.boxplot('delivery_charges', by = ['season','is_happy_customer'] , fi
```

In [ ]:
```
outlier_data.boxplot('delivery_charges', by = ['season','is_happy_customer','is_e
```

Hence we can create separate dataframes based on the season, and then based on the values of the is_expedited_delivery column and is_happy_customer column, we can remove the outliers.

In [ ]:
```

```

In [ ]:
```
summer_outlier = outlier_data[outlier_data.season == 'Summer']
winter_outlier = outlier_data[outlier_data.season == 'Winter']
autumn_outlier = outlier_data[outlier_data.season == 'Autumn']
spring_outlier = outlier_data[outlier_data.season == 'Spring']
```

In [ ]:
```

```

In [ ]:
```
summer_outlier.boxplot('delivery_charges', by = ['is_expedited_delivery','is_happ
```

Hence in this it will be easier to remove the outliers as we can use the combinations of True and False for the two columns is_expedited_delivery and is_happy_customer.

In [ ]:
```
winter_outlier.boxplot('delivery_charges', by = ['is_expedited_delivery','is_happ
```

Hence in this it will be easier to remove the outliers as we can use the combinations of True and False for the two columns is_expedited_delivery and is_happy_customer.

In [ ]:
```
autumn_outlier.boxplot('delivery_charges', by = ['is_expedited_delivery','is_happ
```

In [ ]:
```
spring_outlier.boxplot('delivery_charges', by = ['is_expedited_delivery','is_happ
```

Hence in this it will be easier to remove the outliers as we can use the combinations of True and False for the two columns is_expedited_delivery and is_happy_customer.

In [ ]:
```
print(summer_outlier.shape)
print(winter_outlier.shape)
print(autumn_outlier.shape)
print(spring_outlier.shape)
```

Hence this is the shape of the dataframes before removing outliers.

In [ ]:
```
def reframe_df(df):
    #getting True and False combinations of the two columns

    df_true_true = df[(df.is_expedited_delivery == True) & (df.is_happy_customer
    df_true_false = df[(df.is_expedited_delivery == True) & (df.is_happy_customer
    df_false_true = df[(df.is_expedited_delivery == False) & (df.is_happy_custome
    df_false_false = df[(df.is_expedited_delivery == False) & (df.is_happy_custom


    #calculating the q1 value
    q1 = np.quantile(df_true_true['delivery_charges'], .25)
    #calculating the q3 value
    q3 = np.quantile(df_true_true['delivery_charges'], .75)
    #calculating the iqr value
    iqr = q3-q1
    upper_range = q3+1.5*iqr
    lower_range = q1-1.5*iqr
    #removing outliers below and above q3*1.5iqr
    df_true_true = df_true_true[(df_true_true.delivery_charges <= upper_range) &


    #calculating the q1 value
    q1 = np.quantile(df_true_false['delivery_charges'], .25)
    #calculating the q3 value
    q3 = np.quantile(df_true_false['delivery_charges'], .75)
    #calculating the iqr value
    iqr = q3-q1
    upper_range = q3+1.5*iqr
    lower_range = q1-1.5*iqr
    df_true_false = df_true_false[(df_true_false.delivery_charges <= upper_range)

    #calculating the q1 value

    q1 = np.quantile(df_false_true['delivery_charges'], .25)
    #calculating the q3 value
    q3 = np.quantile(df_false_true['delivery_charges'], .75)
    #calculating the iqr value
    iqr = q3-q1
    upper_range = q3+1.5*iqr
    lower_range = q1-1.5*iqr
    df_false_true = df_false_true[(df_false_true.delivery_charges <= upper_range)


    #calculating the q1 value

    q1 = np.quantile(df_false_false['delivery_charges'], .25)
    #calculating the q3 value
    q3 = np.quantile(df_false_false['delivery_charges'], .75)
    #calculating the iqr value
    iqr = q3-q1
    upper_range = q3+1.5*iqr
    lower_range = q1-1.5*iqr
    df_false_false = df_false_false[(df_false_false.delivery_charges <= upper_rar

    #merginf the datasets back together
    merged_df = pd.concat([df_true_true, df_true_false,df_false_true,df_false_fal
```

```
    return merged_df
```

In [ ]:
```
summer_outlier = reframe_df(summer_outlier)
winter_outlier = reframe_df(winter_outlier)
autumn_outlier = reframe_df(autumn_outlier)
spring_outlier = reframe_df(spring_outlier)
```

In [ ]:
```
#diaplaying the shape of the datasets again after removing outliers
print(summer_outlier.shape)
print(winter_outlier.shape)
print(autumn_outlier.shape)
print(spring_outlier.shape)
```

Finally, we can merge all of them.

In [ ]:
```
final_df_outlier = pd.concat([summer_outlier, winter_outlier,spring_outlier,autur
```

In [ ]:
```
final_df_outlier.dtypes
```

In [ ]:
```
final_df_outlier.to_csv('30759307_outlier_data_solution.csv',index=False)
```

Hencce the outliers have been removed

Summary

From this assignment we have used and learned a lot about wrangng and data cleaning. The following take aways from this for me are -

Sentiment Analysis - It was interesting to determine how from a sentecne we can determine the sentiments of an individual. By directly using the libraries and functions in python we can analyse and interpret how a person is feeling by measuring the polarity scores derieved from the text.

Data Cleansing - I learned how to handle different data types and how to format that in the required condition. I learned how to handle and check the format of date column and also learned how to deal with bool type columns.

Geographical Distance - I have also learned how to calculate distance between two points given their latitutde and longitude values. It was interesting to find out the nearest warehouses based on the values given to us

Graphical Representatons - I have analysed and learnt about the boxplot and the inter quartile ranges. I also learned how to fix the outliers

Dealing wth missing data - I have learned how to deal with missing values and how to impute them from other related column

References

www.stackoverflow.com (http://www.stackoverflow.com)

www.w3schools.com (http://www.w3schools.com)

In [ ]: