# FIT5196: Data Wranling

## Ayush Sharma ¶

### 30823293

## Introduction:

In this assignment we are provided with three different csv files that are dirty_data, missing_data and outlier_data. For each file we need to perform Exploratory data analysis and do different tasks like in dirty data we have to clean the data and find out the correct value for the incorrect value using python. making date in correct format or finding out correct order price or total order amount. or checking if longitude and latitude is correct etc. Whereas in the missing_data file we have to impute values for all the missing values in data set wherever there is null. for the outlier_file we need to find out the outliers in the data set and remove the outliers.

## Importing Library:

Importing the required libraries

In [ ]:

```
import pandas as pd
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from math import sin, cos, sqrt, atan2, radians
import numpy as np
import matplotlib.pylab as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
```

## Reading CSV File of Dirty Data, Missing Data and Outlier Data

In [ ]:

```
#reading Dirty_data csv file into a pandas dataframe
dirty_data=pd.read_csv('30823293_dirty_data.csv')
missing_data=pd.read_csv('30823293_missing_data.csv')
outlier_data=pd.read_csv('30823293_outlier_data.csv')
```

## Describing content of data and Exploring it

As we can see in the very first entry below the format of date was supposed to be YYYY-MM-DD but it is YYYY-DD-MM so we will reformat the date and change it to YYYY-MM-DD and look for cases that also need to be reformated. After that in the df.tail comand we can see that 494 is different and the latitude and longitude are interchanged so we will change all the columns if they have such mistake

In [ ]:

```
# displaying top 10 in dirty data
dirty_data.head(10)
```

In [ ]:

```
# displaying last 10 in dirty data
dirty_data.tail(10)
```

In [ ]:

```
dirty_data.info()
```

In [ ]:

```
dirty_data.describe()
```

## 1.) Checking All the Columns

- Checking the warehouse names and changing the errors
- checking the dates format and changing it into correct format i.e. YYYY-MM-DD
- Checking if all seasons are corresponding to the correct month
- finding the correct distance from warehouse and the correct warehouse
- finding if all the customer are happy or not
- findinng the correct order price and order total

In the next 2 cells we have checked the nearest warehouse column and replaced the column as some of the values are repeated in these column but in actual they are same and there are only 3 warehouses i.e Thompson, Nickolson, Bakers as they represent the same warehouse but they are stored in different ways i.e capitalised and lower case so we will convert all the lowe cases to capitalised.

In [ ]:

```
# Producing the distinct type of values and there count
dirty_data.nearest_warehouse.unique()
```

In [ ]:

```
# making all the values capitalised
dirty_data.nearest_warehouse=dirty_data.nearest_warehouse.str.capitalize()
dirty_data.nearest_warehouse.value_counts()
```

## Date Re-formating:

Earlier I mentioned that there are some dates in YYYY-DD-MM or MM-DD-YYYY format so we will retify the error and reformat the dates in the correct order of YYYY-MM-DD so I have created seperate columns for date month and year and check all the values in they do not belong to that column we will interchange between them like if a month has 2019 or 28 so we will check it and swap it to the correct column and then merge after rectifying the error.

In [ ]:

```
# Splitting the date column and making day month and year column in dataframe
for i,j in dirty_data.iterrows():
    x= dirty_data.loc[i]['date'].split("-")
    dirty_data.loc[i,'year']=x[0]
    dirty_data.loc[i,'month']=x[1]
    dirty_data.loc[i,'day']=x[2]
```

In [ ]:

```
# itterating over each row and checking year,month, day column if they are in wrong positio
for i, j in dirty_data.iterrows():
    if int(j['day'])>31 :
        dirty_data.loc[i,['year','day']]=dirty_data.loc[i,['day','year']].values

    if int(j['month'])>31:
        dirty_data.loc[i,['year','month']]=dirty_data.loc[i,['month','year']].values

    if int(j['month'])>12:
        dirty_data.loc[i,['day','month']]=dirty_data.loc[i,['month','day']].values
dirty_data['date']=dirty_data['year'].astype(str)+'-'+\
                    dirty_data['month'].astype(str)+'-'+dirty_data['day'].astype(str)
```

**Checking if there is no wrong value in the year month and date column**

In [ ]:

```
# checking all values in year
dirty_data['year'].unique()
```

In [ ]:

```
# checking all values in day
dirty_data['day'].unique()
```

In [ ]:

```
# checking all values in month
dirty_data['month'].unique()
```

## Fixing Seasons

- After analysing the warehouse column in the dirty data we had a simple issue in the season column. We can clearly See that some rows in season column are capitalised while some are in in lower case so we will keep all the values in capitalised form so we will replace that column.
- Another problem we encounter in season column was that some dates where in wrong format so we will check if that season is wrong for that date we change those seasons and set them to actual season in Australia.

In [ ]:

```
# looking at each season
dirty_data.season.unique()
```

In [ ]:

```
# capitalising and printing all the values for season
dirty_data.season=dirty_data.season.str.capitalize()
dirty_data.season.unique()
```

The Seasons printed below are those entries that have their seasons updated as wrong so we will replace these season with the correct seasons which we have done in the cell after the next cell.

In [ ]:

```
# checking all the wrong season based on the month column
for i, j in dirty_data.iterrows():
    if (dirty_data.loc[i]['month']=='12' or dirty_data.loc[i]['month']=='01' \
        or dirty_data.loc[i]['month']=='02') and dirty_data.loc[i]['season']!='Summer':
        print(i, dirty_data.loc[i]['season'])

    if (dirty_data.loc[i]['month']=='03' or dirty_data.loc[i]['month']=='04' \
        or dirty_data.loc[i]['month']=='05') and dirty_data.loc[i]['season']!='Autumn':
        print(i, dirty_data.loc[i]['season'])

    if (dirty_data.loc[i]['month']=='06' or dirty_data.loc[i]['month']=='07' \
        or dirty_data.loc[i]['month']=='08') and dirty_data.loc[i]['season']!='Winter':
        print(i, dirty_data.loc[i]['season'])

    if (dirty_data.loc[i]['month']=='09' or dirty_data.loc[i]['month']=='10' \
        or dirty_data.loc[i]['month']=='11') and dirty_data.loc[i]['season']!='Spring':
        print(i, dirty_data.loc[i]['season'])
```

In [ ]:

```
# replacing all the above values that has wrong season
for i, j in dirty_data.iterrows():
    if (dirty_data.loc[i]['month']=='12' or dirty_data.loc[i]['month']=='01' or\
        dirty_data.loc[i]['month']=='02') and dirty_data.loc[i]['season']!='Summer':
        dirty_data.at[i,'season']='Summer'

    if (dirty_data.loc[i]['month']=='03' or dirty_data.loc[i]['month']=='04' or \
        dirty_data.loc[i]['month']=='05') and dirty_data.loc[i]['season']!='Autumn':
        dirty_data.at[i,'season']='Autumn'

    if (dirty_data.loc[i]['month']=='06' or dirty_data.loc[i]['month']=='07' or \
        dirty_data.loc[i]['month']=='08') and dirty_data.loc[i]['season']!='Winter':
        dirty_data.at[i,'season']='Winter'

    if ((dirty_data.loc[i]['month']=='09' or dirty_data.loc[i]['month']=='10' or \
         dirty_data.loc[i]['month']=='11') and dirty_data.loc[i]['season']!='Spring'):
        dirty_data.at[i,'season']='Spring'
```

**checking if all wrong season columns are replaced**

In [ ]:

```
#Final check for all the seasons replaced if wrong
for i, j in dirty_data.iterrows():
    if (dirty_data.loc[i]['month']=='12' or dirty_data.loc[i]['month']=='01' \
        or dirty_data.loc[i]['month']=='02') and dirty_data.loc[i]['season']!='Summer':
        print(i, dirty_data.loc[i]['season'])

    if (dirty_data.loc[i]['month']=='03' or dirty_data.loc[i]['month']=='04' \
        or dirty_data.loc[i]['month']=='05') and dirty_data.loc[i]['season']!='Autumn':
        print(i, dirty_data.loc[i]['season'])

    if (dirty_data.loc[i]['month']=='06' or dirty_data.loc[i]['month']=='07' \
        or dirty_data.loc[i]['month']=='08') and dirty_data.loc[i]['season']!='Winter':
        print(i, dirty_data.loc[i]['season'])

    if (dirty_data.loc[i]['month']=='09' or dirty_data.loc[i]['month']=='10' \
        or dirty_data.loc[i]['month']=='11') and dirty_data.loc[i]['season']!='Spring':
        print(i, dirty_data.loc[i]['season'])
```

## Longitude latitude interchange

- In the beginning we have found out that some longitude and latitude are interchanged so we will interchange these values.
- As we can see most of the customer_long starts with `144.9` and most of the custpmer_lat starts with `-37.8` but there are few values that are interchanged so we will interchange such values where there is any customer_lat which is greater that 0 as latitudes are all in -ve form

In [ ]:

```
dirty_data.customer_lat.unique()
```

In [ ]:

```
dirty_data.customer_long.unique()
```

In [ ]:

```
# interchanging all the values if they are wrong they will be interchanged
for i, j in dirty_data.iterrows():
    if j['customer_lat']>0:
        dirty_data.loc[i,['customer_lat','customer_long']]\
        =dirty_data.loc[i,['customer_long','customer_lat']].values
```

Now if we see these column all the values are now replaced

In [ ]:

```
dirty_data.customer_lat.unique()
```

In [ ]:

```
dirty_data.customer_long.unique()
```

## Customer review

We have encounter an issue that correcponding to some customer reviews the is_happy customer is incorrectly filled so for that we will use a library SentimentIntensityAnalyzer where if a `compound score` is greater than `0.05` we will set the column as True i,e customer is happy otherwise we will set it as false which means the customer is not happy

In [ ]:

```
# instanciating sentiment analyser
analyser = SentimentIntensityAnalyzer()

# creating new colummn for polarity scores
dirty_data['rating'] = dirty_data['latest_customer_review']\
                    .apply(lambda row: analyser.polarity_scores(row))

# creating a column that has only compound score for each sentiment analysis
dirty_data['compound'] = dirty_data['rating']\
                        .apply(lambda score_dict: score_dict['compound'])
# inputing all the correct value for customer happiness
#i.e. if happy it is tru else it is false if the value of compound score is less than 0.05

dirty_data['is_happy_customer_new'] = dirty_data['compound']\
                        .apply(lambda x: True if x>= 0.05 else False)

# creating a new column for that
dirty_data["is_happy_customer"]=dirty_data['is_happy_customer_new']
```

In [ ]:

```
# if there is no review it will be set as true i.e. customer is happy
rowCount = dirty_data.count()
rangeofDf = rowCount.iloc[0]
for i in range (0, rangeofDf):
    if dirty_data.loc[i, 'latest_customer_review'] == 'None':
        dirty_data.loc[i, 'is_happy_customer'] = True
```

## Distance from the warehouse

Now in this part we need to fix the error in the nearest warehouse column by calculating the distance between the customer latitude and customer longitude with warehouse latitude and warehouse longitude and for that we have used `haversine distance formulate` so we will be able to find the distance and the difference between the two wrong entries and for this we have used warehouse csv to get the long and latitude of the warehouse

In [ ]:

```
warehouse=pd.read_csv('warehouses.csv')
#calculating distance


def distance(x1,y1,warehouse_name):
    Radius = 6378.0
    x2 = radians(warehouse['lat'][warehouse['names'] == warehouse_name])
    y2  =  radians(warehouse['lon'][warehouse['names'] == warehouse_name])
    dist_long = y2 - y1
    dist_lat = x2 - x1
    mat = sin(dist_lat / 2)**2 + cos(x1) * cos(x2) * sin(dist_long / 2)**2
    mat1 = 2 * atan2(sqrt(mat), sqrt(1 - mat))
    distance = Radius * mat1
    #for i in distance:
    return distance, warehouse_name
```

In [ ]:

```

for i in range(len(dirty_data['order_id'])):
    final_list = []
    dis_list = []
    name_list = []
    for j in warehouse['names']:
        dis,name = distance(radians(dirty_data.customer_lat[i])\
                           ,radians(dirty_data.customer_long[i]),j)
        dis_list.append(dis)
        name_list.append(name)
    final_list = list(zip(dis_list,name_list))
    final_dis,final_name = min(final_list)[0],min(final_list)[1]
    dirty_data.at[i,'new_distance_to_nearest_warehouse'] = round(final_dis,4)
    dirty_data.at[i,'new_nearest_warehouse'] = final_name

dirty_data.nearest_warehouse=dirty_data.new_nearest_warehouse
dirty_data.distance_to_nearest_warehouse=dirty_data.new_distance_to_nearest_warehouse
```

## Order Price and Order total

- Another issue that we encountered was that the order price and the order total were different based on the coumn shopping cart so for that what we have done is we have splitted the shopping cart and created a matrix of all the occurance of the product and using `np.linalg` we have calculated correct price for each item in the shopping cart with this we have found out the correct order total and order price.

In [ ]:

```
# missing_copy = missing_data[]
missing_copy = missing_data[missing_data.order_price.notnull()]

shopping_list=[]
for i, j in missing_copy.iterrows():
    lst=missing_copy.loc[i]['shopping_cart']
    x=lst.split("), (")
    shopping_dict={}
    for each in x:
        shopping_dict[each.split(",")[0].replace("(","").replace("[","")\
            .replace("\'","")]=int(each.split(",")[1].replace(")","")\
                                   .replace("]","").replace("\'","").strip())
    shopping_list.append(shopping_dict)


uniqueLst=[]
for each in shopping_list:
    for e1 in each:
        uniqueLst.append(e1)


listshopping_dict=[]
uniqueListFinal=[]


for h in uniqueLst:
    if(h not in uniqueListFinal):
        uniqueListFinal.append(h)


len(uniqueListFinal)
for l in shopping_list:
    shopping_mapping=[]
    for each in uniqueListFinal:
        if(each in(l.keys())):
            shopping_mapping.append(l.get(each))
        else:
            shopping_mapping.append(0)
    listshopping_dict.append(shopping_mapping)
```

In [ ]:

```
price_list = []
for i, j in missing_copy.iterrows():
    price_list.append(missing_copy.loc[i, "order_price"])
```

In [ ]:

```
shopping_array = np.array(listshopping_dict[:10])
price_array = np.array(price_list[:10])
linear_price = np.linalg.solve(shopping_array, price_array)
linear_price=list(linear_price)
```

In [ ]:

```
priceDict={}
i=0
for each in uniqueListFinal:
    priceDict[each]=linear_price[i]
    i=i+1
```

In [ ]:

```
dirty_data["new_order_price"] = 0
i=0
for x in dirty_data.shopping_cart:
    total_amount=0
    for x in eval(x):
        if x[0] in priceDict.keys():
            total_amount=total_amount+x[1]*priceDict[x[0]]
    dirty_data.at[i,"new_order_price"]=total_amount
    i+=1
```

In [ ]:

```

dirty_data['new_order_total'] = 0
dirty_data['new_order_total']=(((100-dirty_data['coupon_discount'])/100)\
                               *dirty_data['new_order_price'])+dirty_data['delivery_charges
```

- As we can see the new order price is different in the 9th and 12th entry in the output of below cell we will replace this value and with the new correct value so that we can make sure we dont replace correct values with the wrong ones so in the cell after the next cell we will replace all such wrong values for both order price and order total.
- Also we can see in the 487th entry that the order price and new order price is same but the order total is different so we will replace that too as well

In [ ]:

```
dirty_data[["order_price","new_order_price",'coupon_discount'\
            ,'delivery_charges','new_order_total','order_total']].head(15)
```

In [ ]:

```
dirty_data[["order_price","new_order_price",'coupon_discount'\
            ,'delivery_charges','new_order_total','order_total']].tail(15)
```

In [ ]:

```
for i, j in dirty_data.iterrows():
    if dirty_data.loc[i]['order_total']==dirty_data.loc[i]['new_order_total'] and \
    dirty_data.loc[i]['order_price']!=dirty_data.loc[i]['new_order_price']:
        dirty_data.at[i,'order_price']=dirty_data.loc[i]['new_order_price']
    if dirty_data.loc[i]['order_total']!=dirty_data.loc[i]['new_order_total'] and\
    dirty_data.loc[i]['order_price']==dirty_data.loc[i]['new_order_price']:
        dirty_data.at[i,'order_total']=dirty_data.loc[i]['new_order_total']
```

In the below 2 cells we have confirmed that all three of the values mentioned above are replaced and we now have the correct value.

In [ ]:

```
dirty_data[["order_price","new_order_price",'coupon_discount','delivery_charges','new_order
```

In [ ]:

```
dirty_data[["order_price","new_order_price",'coupon_discount','delivery_charges','new_order
```

In [ ]:

```
dirty_data.shape
```

## Part 2: Missing Data

In the part 2 of the assignment we have some missing values in our data so we will impute these values. In the cell below we have find out the number of values that are missing in the data frame where the following has the number of missing data:

- nearest_warehouse: `55`
- order_price: `15`
- delivery_charges: `40`
- order_total: `15`
- distance_to_nearest_warehouse: `31`
- is_happy_customer: `40`

In [ ]:

```
#display all column with missing values
missing_data.isnull().sum()
```

### Nearest Warehouse:

- We have created a funtion above in the dirty data part named `distance` and we have called it here to find

out the distance and the nearest warehourse where there is a missing value.
- In the below 3 cells have calculated the distance from the warehouse for each entry and found out the nearest warehourse to the customer longitude and latiude using haversine distance
- And in the end I have replaced all the missing values in the nearest warehouse and distance to warehouse from the new nearest warehouse and new distance to warehouse.

In [ ]:

```
#calculating distance

for i in range(len(missing_data['order_id'])):
    final_list = []
    dis_list = []
    name_list = []
    for j in warehouse['names']:
        dis,name = distance(radians(missing_data.customer_lat[i]),radians(missing_data.cust
        dis_list.append(dis)
        name_list.append(name)
    final_list = list(zip(dis_list,name_list))
    final_dis,final_name = min(final_list)[0],min(final_list)[1]
    missing_data.at[i,'new_distance_to_nearest_warehouse'] = round(final_dis,4)
    missing_data.at[i,'new_nearest_warehouse'] = final_name

#missing_data.nearest_warehouse=missing_data.new_nearest_warehouse
#missing_data.distance_to_nearest_warehouse=missing_data.new_distance_to_nearest_warehouse
```

In [ ]:

```
# replacing all the null values with new warehouse values
for i,j in missing_data.iterrows():
    if pd.isnull(j['nearest_warehouse']):
        missing_data.loc[i,['nearest_warehouse','new_nearest_warehouse']]=missing_data.loc[

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

In [ ]:

```
for i,j in missing_data.iterrows():
    if pd.isnull(j['distance_to_nearest_warehouse']):
        missing_data.loc[i,['distance_to_nearest_warehouse','new_distance_to_nearest_wareho


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

In [ ]:

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

## Happy Customer review:

is_happy_customer is a column that has a true or false value in form of 0 and 1 and some values are missing so we will impute these values and find out the correct value for these.

In [ ]:

```
# instanciating sentiment analyser
analyser = SentimentIntensityAnalyzer()

# creating new colummn for polarity scores
missing_data['rating'] = missing_data['latest_customer_review']\
                    .apply(lambda row: analyser.polarity_scores(row))

# creating a column that has only compound score for each sentiment analysis
missing_data['compound'] = missing_data['rating']\
                        .apply(lambda score_dict: score_dict['compound'])
# inputing all the correct value for customer happiness
#i.e. if happy it is tru else it is false if the value of compound score is less than 0.05

missing_data['is_happy_customer_new'] = missing_data['compound']\
                        .apply(lambda x: 1.0 if x>= 0.05 else 0.0)

# creating a new column for that
for i,j in missing_data.iterrows():
    if pd.isnull(j['is_happy_customer']):
        missing_data.loc[i,['is_happy_customer','is_happy_customer_new']]=missing_data.loc[
```

Checking if all the values have been imputed as we can see we have imputed all the missing values and we can confirm it below.

In [ ]:

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

## Missing order price and order total

- now we will update all the missing order price and order total using the linear equation solver we have used earlier to get order price and order total so we will replace such values using the same way and the price we have found for that

In [ ]:

```
# finding out the price for each value in dataframe
missing_data["new_order_price"] = 0
i=0
for x in missing_data.shopping_cart:
    total_amount=0
    for x in eval(x):
        if x[0] in priceDict.keys():
            total_amount=total_amount+x[1]*priceDict[x[0]]
    missing_data.at[i,"new_order_price"]=total_amount
    i+=1
```

In [ ]:

```
# calculating order total
missing_data['new_order_total'] = 0
missing_data['new_order_total']=(((100-missing_data['coupon_discount'])/100)\
                               *missing_data['new_order_price'])+missing_data['delivery_cha
```

In [ ]:

```
# adding all the values to missing order price and order total
for i,j in missing_data.iterrows():
    if pd.isnull(j['order_price']):
        missing_data.loc[i,['order_price','new_order_price']]=missing_data.loc[i,['new_orde
    if pd.isnull(j['order_total']):
        missing_data.loc[i,['order_total','new_order_total']]=missing_data.loc[i,['new_orde

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

# Missing Delivery charge

from above part we have the missing order total and order price we will find out delivery charges for all missing values

In [ ]:

```
missing_data['new_delivery_charge'] = 0
missing_data['new_delivery_charge']=missing_data['order_total']-(((100-missing_data['coupon
                               *missing_data['new_order_price'])
```

In [ ]:

```
for i,j in missing_data.iterrows():
    if pd.isnull(j['delivery_charges']):
        missing_data.loc[i,['delivery_charges','new_delivery_charge']]=missing_data.\
        loc[i,['new_delivery_charge','delivery_charges']].values
```

**Showing all the columns that we have missing data are removed**

In [ ]:

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

# Part3 Outlier Data:

In the outlier file we have to remove all the outlier in the delivery charges column for that we have used a linear regression model to predict the delivery charge and find out the difference between each of them after that we have remove those values from the data where there is a outlier in the difference between the actual delivery charge and orignal delivery charge. in the plot below we have show that there are outliers in the delivery charge column and for each season there is are outliers

In [ ]:

```
#Plot for outlier in delivery charge
plt.figure(figsize=(15,4))
sns.boxplot( x = outlier_data["delivery_charges"] )
```

In [ ]:

```
#Plot for outlier in delivery charge and season
plt.figure(figsize=(10,8))
sns.boxplot(y='delivery_charges',x='season',hue='season',data=outlier_data)
plt.show()
```

In [ ]:

```
#Adding  a column with numeric value to predict delivery charge
datasize=dirty_data.count().iloc[0]
for x in range (0, datasize):
    if dirty_data.loc[x, 'is_expedited_delivery'] == True:
           dirty_data.loc[x, 'new_is_expedited_delivery'] = 1.0
    else:
            dirty_data.loc[x, 'new_is_expedited_delivery'] = 0.0

    if dirty_data.loc[x, 'is_happy_customer'] == True:
            dirty_data.loc[x, 'new_is_happy_customer'] = 1.0
    else:
            dirty_data.loc[x, 'new_is_happy_customer'] = 0.0
```

In [ ]:

```
#seperating all data based on season
summer_df=dirty_data[['is_expedited_delivery','distance_to_nearest_warehouse'\
                      ,'is_happy_customer','delivery_charges']][dirty_data.season=='Summer'
autumn_df=dirty_data[['is_expedited_delivery','distance_to_nearest_warehouse'\
                      ,'is_happy_customer','delivery_charges']][dirty_data.season=='Autumn'
winter_df=dirty_data[['is_expedited_delivery','distance_to_nearest_warehouse'\
                      ,'is_happy_customer','delivery_charges']][dirty_data.season=='Winter'
spring_df=dirty_data[['is_expedited_delivery','distance_to_nearest_warehouse'\
                      ,'is_happy_customer','delivery_charges']][dirty_data.season=='Spring'
```

## Fitting Linear regression model:

We have used dirty data to fit in linear regression model on the using is_expedited_delivery, distance_to_nearest_warehouse, is_happy_customer columns and target variable is delivery charges

In [ ]:

```
lm=LinearRegression()
summer_model=lm.fit(summer_df[['is_expedited_delivery','distance_to_nearest_warehouse'\
                      ,'is_happy_customer']],summer_df.delivery_charges)
autumn_model=lm.fit(autumn_df[['is_expedited_delivery','distance_to_nearest_warehouse'\
                      ,'is_happy_customer']],autumn_df.delivery_charges)
winter_model=lm.fit(winter_df[['is_expedited_delivery','distance_to_nearest_warehouse'\
                      ,'is_happy_customer']],winter_df.delivery_charges)
spring_model=lm.fit(spring_df[['is_expedited_delivery','distance_to_nearest_warehouse'\
                      ,'is_happy_customer']],spring_df.delivery_charges)
```

In [ ]:

```
datasize1=outlier_data.count().iloc[0]
for x in range (0, datasize1):
    if outlier_data.loc[x, 'is_expedited_delivery'] == True:
           outlier_data.loc[x, 'new_is_expedited_delivery'] = 1.0
    else:
            outlier_data.loc[x, 'new_is_expedited_delivery'] = 0.0

    if outlier_data.loc[x, 'is_happy_customer'] == True:
            outlier_data.loc[x, 'new_is_happy_customer'] = 1.0
    else:
            outlier_data.loc[x, 'new_is_happy_customer'] = 0.0
```

In [ ]:

```
outlier_summer_df=outlier_data[outlier_data.season=='Summer']
outlier_autumn_df=outlier_data[outlier_data.season=='Autumn']
outlier_winter_df=outlier_data[outlier_data.season=='Winter']
outlier_spring_df=outlier_data[outlier_data.season=='Spring']
```

## Predicting:

Now we are predicting the delivery charges for the model we have made and then we will remove all the outlier using this predicted value

In [ ]:

```
#predicting values using linear regression
outlier_summer_df['predicted_delivery_charges']= summer_model\
.predict(outlier_summer_df[['is_expedited_delivery','distance_to_nearest_warehouse'\
                       ,'is_happy_customer']])


outlier_autumn_df['predicted_delivery_charges'] = autumn_model\
.predict(outlier_autumn_df[['is_expedited_delivery','distance_to_nearest_warehouse'\
                       ,'is_happy_customer']])

outlier_winter_df['predicted_delivery_charges']= winter_model\
.predict(outlier_winter_df[['is_expedited_delivery','distance_to_nearest_warehouse'\
                       ,'is_happy_customer']])

outlier_spring_df['predicted_delivery_charges'] = spring_model\
.predict(outlier_spring_df[['is_expedited_delivery','distance_to_nearest_warehouse'\
                       ,'is_happy_customer']])
```

## Removing outlier

Now we are removing outlier by finding out the IQR and Lower and upper bound and remove all the values that are below and above these values

In [ ]:

```

outlier_df = pd.concat([outlier_summer_df, outlier_autumn_df, outlier_winter_df, outlier_sp
outlier_df
outlier_df['delivery_charges_difference'] = outlier_df['delivery_charges'] - outlier_df['pr
```

In [ ]:

```
import statistics

outlier_df_spring= outlier_df[outlier_df.season=="Spring"]
#print(outlier_df_summer)
lst=outlier_df_spring['delivery_charges_difference'].values.tolist()
lst.sort()
median=statistics.median(lst)
lst1=lst[0:int(len(lst)/2)]
lst2=lst[int(len(lst)/2):len(lst)]
median1=statistics.median( lst1)
print("median1")
print(median1)
median2= statistics.median(lst2)
print("median2")
print(median2)
IQR= median2-median1
print('IQR')
print(IQR)

lower_bond= median1- 1.5*(IQR)
upper_bond= median2+ 1.5 *(IQR)
print(lower_bond)
print(upper_bond)
outlier_df_spring_new=outlier_df_spring.copy()
for i,j in outlier_df_spring.iterrows():

    if(outlier_df_spring.loc[i,'delivery_charges_difference']<lower_bond):
        outlier_df_spring_new.drop(i,inplace=True)
    if(outlier_df_spring.loc[i,'delivery_charges_difference']>upper_bond):
        outlier_df_spring_new.drop(i,inplace=True)
len(outlier_df_spring_new)
```

In [ ]:

```
import statistics

outlier_df_summer= outlier_df[outlier_df.season=="Summer"]
#print(outlier_df_summer)
lst=outlier_df_summer['delivery_charges_difference'].values.tolist()
lst.sort()
median=statistics.median(lst)
lst1=lst[0:int(len(lst)/2)]
lst2=lst[int(len(lst)/2):len(lst)]
median1=statistics.median( lst1)
print("median1")
print(median1)
median2= statistics.median(lst2)
print("median2")
print(median2)
IQR= median2-median1
print('IQR')
print(IQR)

lower_bond= median1- 1.5*(IQR)
upper_bond= median2+ 1.5 *(IQR)
print(lower_bond)
print(upper_bond)
outlier_df_summer_new=outlier_df_summer.copy()
for i,j in outlier_df_summer.iterrows():

    if(outlier_df_summer.loc[i,'delivery_charges_difference']<lower_bond):
        outlier_df_summer_new.drop(i,inplace=True)
    if(outlier_df_summer.loc[i,'delivery_charges_difference']>upper_bond):
        outlier_df_summer_new.drop(i,inplace=True)
len(outlier_df_summer)
```

In [ ]:

```
import statistics

outlier_df_winter= outlier_df[outlier_df.season=="Winter"]
#print(outlier_df_summer)
lst=outlier_df_winter['delivery_charges_difference'].values.tolist()
lst.sort()
median=statistics.median(lst)
lst1=lst[0:int(len(lst)/2)]
lst2=lst[int(len(lst)/2):len(lst)]
median1=statistics.median( lst1)
print("median1")
print(median1)
median2= statistics.median(lst2)
print("median2")
print(median2)
IQR= median2-median1
print('IQR')
print(IQR)

lower_bond= median1- 1.5*(IQR)
upper_bond= median2+ 1.5 *(IQR)
print(lower_bond)
print(upper_bond)
outlier_df_winter_new=outlier_df_winter.copy()
for i,j in outlier_df_winter.iterrows():

    if(outlier_df_winter.loc[i,'delivery_charges_difference']<lower_bond):
        outlier_df_winter_new.drop(i,inplace=True)
    if(outlier_df_winter.loc[i,'delivery_charges_difference']>upper_bond):
        outlier_df_winter_new.drop(i,inplace=True)
len(outlier_df_winter)
len(outlier_df_winter_new)
```

In [ ]:

```
import statistics

outlier_df_autumn= outlier_df[outlier_df.season=="Autumn"]
#print(outlier_df_summer)
lst=outlier_df_autumn['delivery_charges_difference'].values.tolist()
lst.sort()
median=statistics.median(lst)
lst1=lst[0:int(len(lst)/2)]
lst2=lst[int(len(lst)/2):len(lst)]
median1=statistics.median( lst1)
print("median1")
print(median1)
median2= statistics.median(lst2)
print("median2")
print(median2)
IQR= median2-median1
print('IQR')
print(IQR)

lower_bond= median1- 1.5*(IQR)
upper_bond= median2+ 1.5 *(IQR)
print(lower_bond)
print(upper_bond)
outlier_df_autumn_new=outlier_df_autumn.copy()
for i,j in outlier_df_autumn.iterrows():

    if(outlier_df_autumn.loc[i,'delivery_charges_difference']<lower_bond):
        outlier_df_autumn_new.drop(i,inplace=True)
    if(outlier_df_autumn.loc[i,'delivery_charges_difference']>upper_bond):
        outlier_df_autumn_new.drop(i,inplace=True)
len(outlier_df_autumn)
len(outlier_df_autumn)
```

In [ ]:

```
finalDF=pd.concat([outlier_df_autumn_new,outlier_df_spring_new,outlier_df_summer_new,outlie
```

## Plots to show Outliers are removed

As we can see below all the outliers are removed from our final data and there are no outlier point that are showing in both the plots thus we can say now that outliers have been handles and we have removed them now we will check all the files and remove any extra columns

In [ ]:

```
plt.figure(figsize=(10,8))
sns.boxplot(x='delivery_charges_difference',data=finalDF)
plt.show()
```

In [ ]:

```
plt.figure(figsize=(10,8))
sns.boxplot(x='delivery_charges_difference',y='season',data=finalDF)
plt.show()
```

# Creating Final Datframe Frame for Writing CSV file:

In [ ]:

```
dirty_data.shape
```

In [ ]:

```
missing_data.shape
```

In [ ]:

```
finalDF.shape
```

In [ ]:

```
dirty_data.dtypes
```

In [ ]:

```
missing_data.dtypes
```

In [ ]:

```
finalDF.dtypes
```

In [ ]:

```
dirty_data.drop(['year','month','day','rating','compound','is_happy_customer_new'\
                 ,'new_distance_to_nearest_warehouse','new_nearest_warehouse','new_order_pr
                 ,'new_order_total','new_is_expedited_delivery'\
                 ,'new_is_happy_customer'],axis=1,inplace=True)
```

In [ ]:

```
missing_data.drop(['new_distance_to_nearest_warehouse','new_nearest_warehouse','rating','co
                   ,'is_happy_customer_new','new_order_price','new_order_total'\
                   ,'new_delivery_charge'],axis=1,inplace=True)
```

In [ ]:

```
finalDF.drop(['new_is_expedited_delivery','new_is_happy_customer'\
              ,'predicted_delivery_charges','delivery_charges_difference'],axis=1,inplace=T
```

In [ ]:

```
dirty_data.shape
```

In [ ]:

```
missing_data.shape
```

In [ ]:

```
finalDF.shape
```

## Writting Outlier dataframe into CSV

In [ ]:

```
dirty_data.to_csv('30823293_dirty_data_solution.csv',index=False)
missing_data.to_csv('30823293_missing_data_solution.csv',index=False)
finalDF.to_csv('30823293_outlier_data_solution.csv',index=False)
```

## Conclusion:

From this assignment I have learned how to analyze and wrangle data. I have learned how to work with different formats of data like handling of the date column. I also learned how to do sentiment analysis using the given functions and libraries in python. This allowed me to learn how by just a text we can find out the sentiment of the person. I also learned how to handle and calculate distances between two points given their latitude and longitude values. This assignment also allowed me to learn about handling the missing data and the outliers data. I learned to calculate the outlier values and use its concept to remove them accordingly.

### Reference:

https://stackoverflow.com/questions/26886653/pandas-create-new-column-based-on-values-from-other-columns-apply-a-function-o (https://stackoverflow.com/questions/26886653/pandas-create-new-column-based-on-values-from-other-columns-apply-a-function-o) https://stackoverflow.com/questions/19412462/getting-distance-between-two-points-based-on-latitude-longitude (https://stackoverflow.com/questions/19412462/getting-distance-between-two-points-based-on-latitude-longitude)