Assignment 2- Linear Regression and Logestic Regression

Group 4 - Jhalak Sadana, Nisarg Desai, Sruthi Kommera, Varalaxmi Etta

Introduction

This report presents an analysis of the publicly available dataset of waiting times on the USA-Canada border. The scope of this report is limited to only analyzing the ‘Travellers Flow’ on the ‘Queenston-Lewiston’ bridge based on wait time as a function of time of day, weekday or weekend, other holidays and month.

We use the publicly available dataset - Historical Border Wait Times between the years 2010-2014 for the analysis. To assist in the analyses, we use the pandas library of python for data manipulation, numpy for mathematical function, sci-kit learn and statsmodels for regression analysis, and seaborn and matplotlib for plotting the data.

Process

In this process,we have used multiple packages for building the models for wait time as target variable which are easy to use open-source data analysis and built on top of Python programming language to utilize the data by following below steps:

Steps:

1) From the imported data given, we have filtered out desired Bridge data which is Lewiston Bridge and cleaned up with omitting 'Commercial flow' column for question ourposes, No Delays marked as '0' based on the website(0-10 minutes waiting time is '0' minutes), diregarding the 'Closed' as they are like outliers to the Data.

2) Then, We have extracted 'X' variables from column 'Updated' for function of time of day, weekday/weekend, other holidays, and month.

3) We built 2 Models: First, OLS(Liear Regression) using wait time as a continuous variable and Logistic Regression using wait time as a categorical variable.

4) Furthermore, we used the entire data as train and taken 2015 Q1 as test data to check on how models are accurate and stable.

Importing necessary libaries

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings # scipy has some internal issues that comes up as warning
warnings.filterwarnings('ignore')
import seaborn as sns
import matplotlib as plt
import holidays
import statsmodels.formula.api as sm
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
import holidays
```

Dataset Description

This dataset gives an insight into the border waiting times at the USA-Canada border, and is obtained from the Canadian government’s website. This data set is updated periodically once every hour to indicate wait times for both travelers and commercial flow through all of the borders between USA-Canada. It indicates a ‘No Delay’ which essentially means that there is no major time delay, and has values of up to 420 minutes.

In [5]:
```
df=pd.read_csv('C:/Users/Vara lakshmi/OneDrive/Desktop/RIT/Fall Sem/680-BANA/BANA-Assignemnt 2/bwt-taf-2010-201
df=df[df['CBSA Office']=='Queenston-Lewiston Bridge']     # filtering only Lewiston Bridge
df.head()                                                 # Printing the 5 observations of the data
```

Out[5]:

	CBSA Office	Location	Updated	Commercial Flow	Travellers Flow
369413	Queenston-Lewiston Bridge	Queenston, ON	2014-04-04 13:06 EDT	No Delay	No Delay
369414	Queenston-Lewiston Bridge	Queenston, ON	2014-04-04 12:05 EDT	No Delay	No Delay
369415	Queenston-Lewiston Bridge	Queenston, ON	2014-04-04 11:09 EDT	No Delay	No Delay
369416	Queenston-Lewiston Bridge	Queenston, ON	2014-04-04 11:07 EDT	No Delay	No Delay
369417	Queenston-Lewiston Bridge	Queenston, ON	2014-04-04 11:07 EDT	No Delay	No Delay

Data Cleaning Procedures

To get a more accurate analysis from the dataset, it is necessary to clean the dataset. Cleaning the dataset refers to removing unnecessary data, fixing inconsistencies in formatting, removing null values or nonsensical outliers. We start by filtering the dataset of 900,000 +rows to just the one bridge of 'Queenston-Lewiston Bridge'. This is followed by renaming 'Travellers Flow' to 'TravellersFlow' to avoid errors and easier use for later functions. We then switch 'No delay' to 0 and change travelers flow into an integer form inorder to perform mathematical operations. We then drop columns, which are not needed, and reset the index so we can access the updated dataset. This is followed by converting the 'Updated' column to date time type inorder to extract more information in later stages.

In [6]:
```
df = df.rename({'Travellers Flow': 'TravellersFlow'}, axis=1)       #  Replacing 'No Delay' with '0'
df['TravellersFlow']=df['TravellersFlow'].replace('No Delay',0)     # Renaming "Travellers Flow" to "Travellers
df = df[(df['TravellersFlow'] != 'Closed')]                         # Omitting "Closed" observations
df = df.drop(['Commercial Flow','Location','CBSA Office'],axis=1)   # dropping "Commercial Flow", "Location", "
df.reset_index(inplace = True , drop = True)                        # Resetting the Index
```

In [7]:
```
df['TravellersFlow'] = df['TravellersFlow'].astype(int)             # Changing "TravellersFlow" to integer
df['Updated'] = pd.to_datetime(df['Updated'])                       # Converting scalar to Date and time.
df.info()                                                           # Getting the stats of the data.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 44222 entries, 0 to 44221
Data columns (total 2 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Updated         44222 non-null  datetime64[ns, tzlocal()]
 1   TravellersFlow  44222 non-null  int64
dtypes: datetime64[ns, tzlocal()](1), int64(1)
memory usage: 691.1 KB
```

In [8]:
```
df['Day'] = df['Updated'].dt.day_name()                   # Extracting Day as a variable
df['Month'] = pd.DatetimeIndex(df['Updated']).month       # Extracting Month as a variable
df['Hour'] = pd.DatetimeIndex(df['Updated']).hour         # Extracting hour as a variable
df['Weekday'] = df['Day'].apply(lambda x: 1 if x == 'Saturday' or x == 'Sunday' else 0) # Getting holiday sched
df['holidayUS'] = pd.Series(df.Updated).apply(lambda x: holidays.CountryHoliday('US').get(x)).values.astype('bo
df.head()
```

Out[8]:

	Updated	TravellersFlow	Day	Month	Hour	Weekday	holidayUS
0	2014-04-04 13:06:00-04:00	0	Friday	4	13	0	0
1	2014-04-04 12:05:00-04:00	0	Friday	4	12	0	0
2	2014-04-04 11:09:00-04:00	0	Friday	4	11	0	0
3	2014-04-04 11:07:00-04:00	0	Friday	4	11	0	0
4	2014-04-04 11:07:00-04:00	0	Friday	4	11	0	0

Above table shows Weekday as 0 and weekend as 1, and other holiday, here considered 'federal US holidays', are 1 and non-holidays are 0.

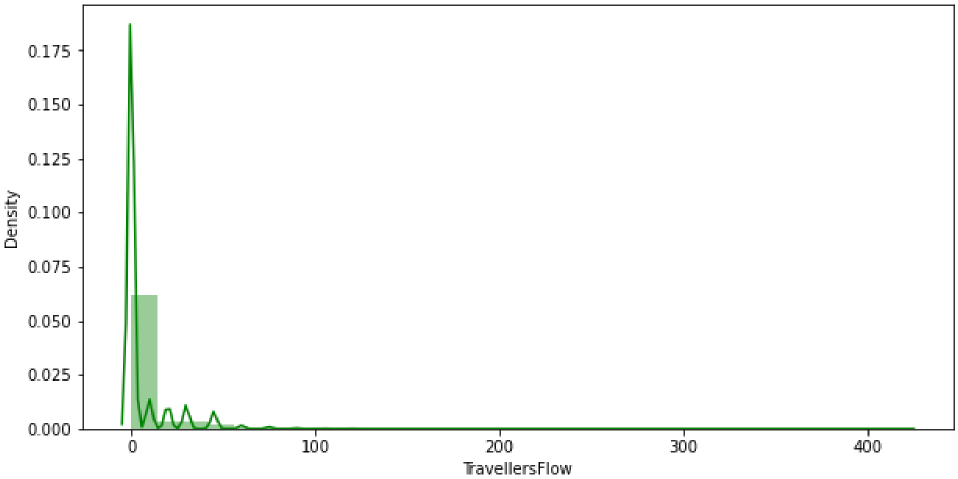
Exploratory Data Analysis (EDA)

Exploratory Data Analysis [EDA] is a preliminary,high level analysis of the dataset. It involves getting the type of variables, nature of variables and to get an idea of the relationships between variables. We also plot the results to see more clearly the relationships between them. Plots are a great way to look for similarities in relationships of variables. They can be later categorized based on their values. The graph we see is right skewed and we can see that there are outliers and inorder to improve the results, we eliminate the outliers. We then find the similar relationships and plot them in order to categorize them into groups.

In [9]:
```
print(df['TravellersFlow'].describe())                                            # Printing the Dataframe
```

```
plt.figure(figsize=(10, 5))                                              # plotting the graph
sns.distplot(df['TravellersFlow'], color='g', bins=30, hist_kws={'alpha': 0.4})     # Giving the measures to pl
plt.show()
```

```
count    44222.000000
mean         5.280290
std         13.800454
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max        420.000000
Name: TravellersFlow, dtype: float64
```



Travellers Flow is at a mean of 5.2 mins, and there are only a few outlier values. We can eliminate the outlier to further improve the analysis. We remove the least number of occuring values of 135,150 and 420 mins.

In [10]:
```
df.dtypes                        # Getting the variable datatypes
```

Out[10]:
```
Updated           datetime64[ns, tzlocal()]
TravellersFlow                        int64
Day                                  object
Month                                 int64
Hour                                  int64
Weekday                               int64
holidayUS                             int64
dtype: object
```

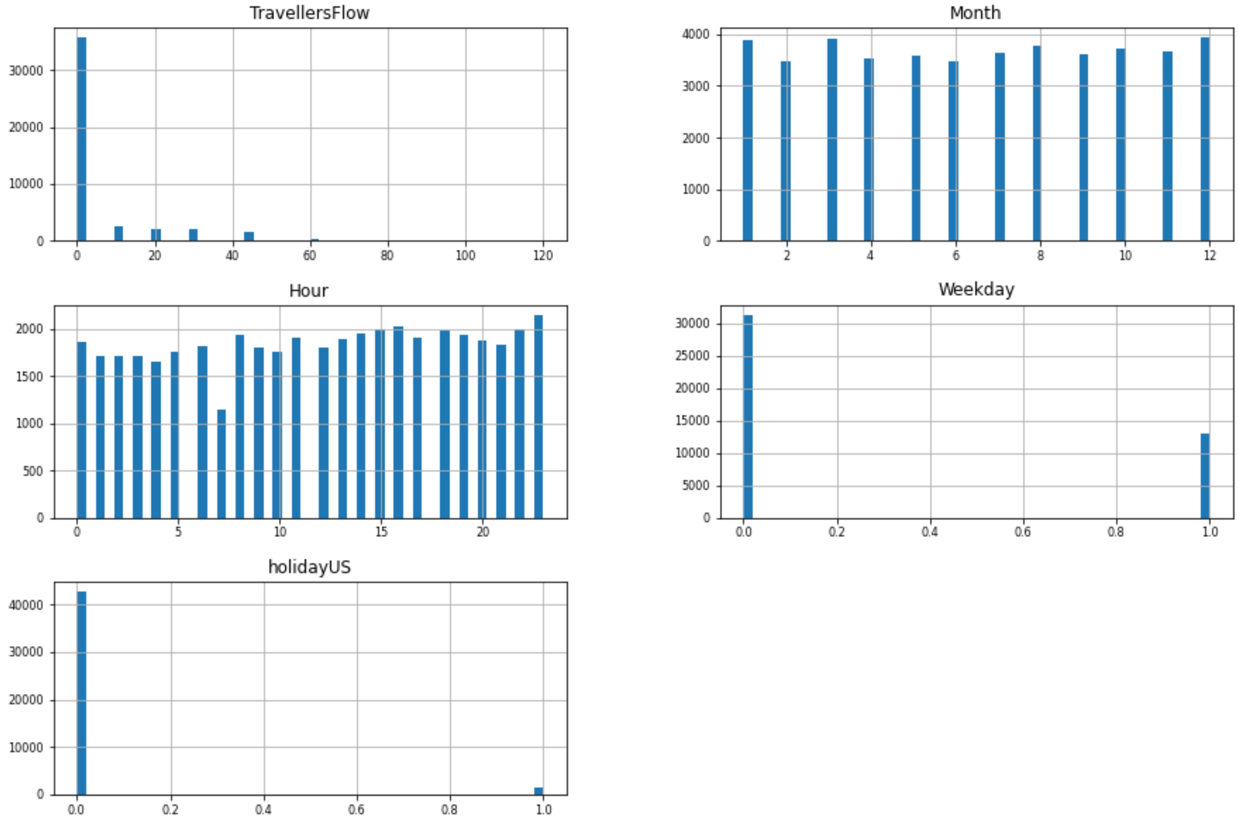
Plotting numerical data

In [11]:
```
indexes = df[ (df['TravellersFlow'] == 135) | (df['TravellersFlow']==150)| (df['TravellersFlow']==420)].index
df.drop(indexes,inplace=True)                #eliminating 135,150,420 as they are outliers
```

We have disregarded the outliers from the Travellers flow and he data for holidayUS is sorted to 1 or 0 category. There is alot of similarity in certain time intervals and month of the year. So, we need to futher categorize hours and month.

In [12]:
```
df_num = df.select_dtypes(include = ['float64', 'int64', 'int32'])    # Plotting variables to check for the dis
df_num.hist(figsize=(15, 10), bins=50, xlabelsize=8, ylabelsize=8)    # fixing the measurements
```

Out[12]:
```
array([[<AxesSubplot:title={'center':'TravellersFlow'}>,
        <AxesSubplot:title={'center':'Month'}>],
       [<AxesSubplot:title={'center':'Hour'}>,
        <AxesSubplot:title={'center':'Weekday'}>],
       [<AxesSubplot:title={'center':'holidayUS'}>, <AxesSubplot:>]],
      dtype=object)
```



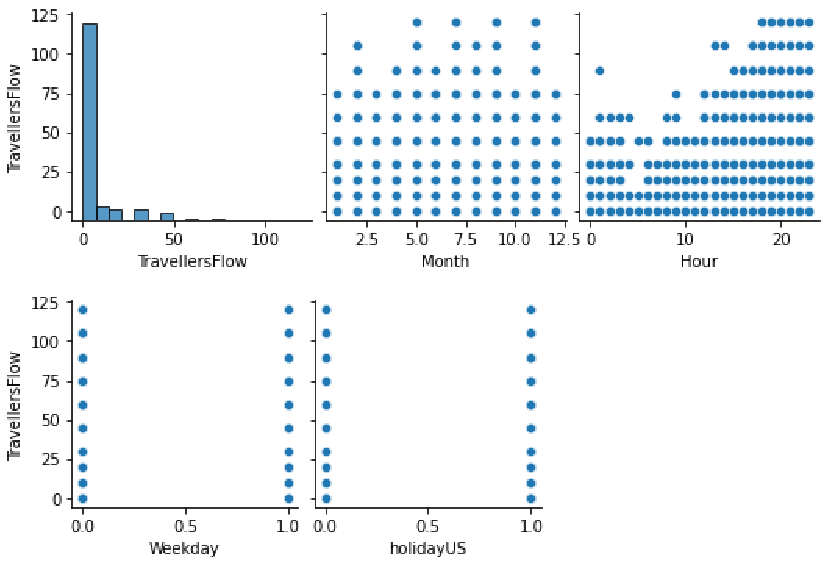
As we can see from the above graphs, Months and hours have the similar trends data which can be grouped for better analysis of the model. For grouping,we go by grouping them by their average mean values.

In [13]:
```
df.corr()    # checking the correlation between variables
```

Out[13]:

	TravellersFlow	Month	Hour	Weekday	holidayUS
TravellersFlow	1.000000	0.076903	0.287491	0.157983	0.019054
Month	0.076903	1.000000	0.001427	0.000781	0.039367
Hour	0.287491	0.001427	1.000000	0.001987	-0.004990
Weekday	0.157983	0.000781	0.001987	1.000000	-0.067512
holidayUS	0.019054	0.039367	-0.004990	-0.067512	1.000000

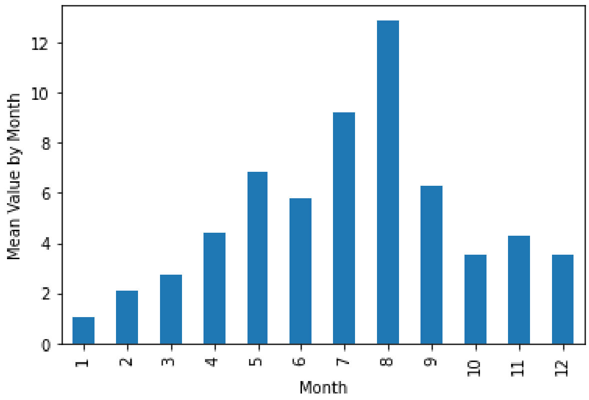
In [14]:
```
for i in range(0, len(df_num.columns), 3):
    sns.pairplot(data=df_num,
                 x_vars=df_num.columns[i:i+3],
                 y_vars=['TravellersFlow'])
plt.show()
```



In [16]:
```
AvgTimeMonth = df.groupby('Month')['TravellersFlow'].mean()      # grouping based on the average time taken by
```

```
AvgTimeMonth.plot(kind='bar',ylabel='Mean Value by Month')       # Plotting the average time taken
```

Out[16]: <AxesSubplot:xlabel='Month', ylabel='Mean Value by Month'>

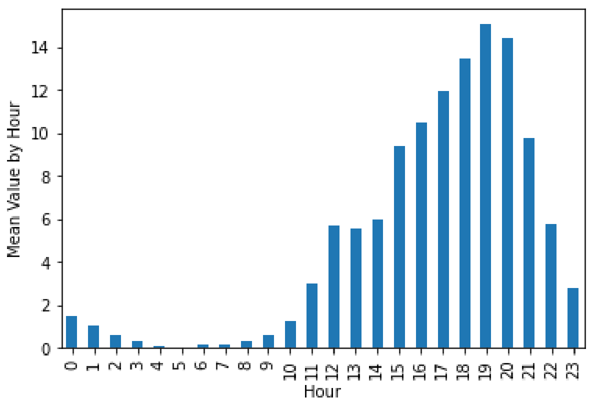


Now based on the monthly avg waiting time above, we will categorize in to four categories for the analysis.

In [41]:
```
df['CategoryMonth'] = df['Month'].apply(lambda x: 'A' if x == 1 or x == 2 or x == 3 else 'B' if x == 4 or x ==
                                        x == 12 else 'C' if x == 5 or x == 6 or x == 7 or x == 9  else 'D')
```

In [18]:
```
AvgTimeHour = df.groupby('Hour')['TravellersFlow'].mean()   # grouping based on the average time taken by Hour
AvgTimeHour.plot(kind='bar',ylabel='Mean Value by Hour')    # Plotting the average time taken and hour
```

Out[18]: <AxesSubplot:xlabel='Hour', ylabel='Mean Value by Hour'>



In [44]:
```
df['Hour']=df['Hour'].apply(int)
df['CategoryHours'] = df['Hour'].apply(lambda x: 'H1' if x == 2 or x == 3 or x == 4 or x == 5 or x == 6 or x ==
                                        else 'H2' if x == 0 or x == 1 or x == 10 or x == 11 or x == 23
                                        else 'H3' if x == 12 or x == 13 or x == 14 or x == 15 or x == 22
                                        else 'H4')
df.head()
```

Out[44]:

	Updated	TravellersFlow	Day	Month	Hour	Weekday	holidayUS	CategoryMonth	CategoryHours
0	2014-04-04 13:06:00	0	Friday	4	13	0	0	B	H3
1	2014-04-04 12:05:00	0	Friday	4	12	0	0	B	H3
2	2014-04-04 11:09:00	0	Friday	4	11	0	0	B	H2
3	2014-04-04 11:07:00	0	Friday	4	11	0	0	B	H2
4	2014-04-04 11:07:00	0	Friday	4	11	0	0	B	H2

Ordinary Least Square Regression

In [45]:
```
df['Hour']=df['Hour'].apply(str)              # converting Hour to string
df['Weekday']=df['Weekday'].apply(str)        # converting Weekday to string
df['holidayUS']=df['holidayUS'].apply(str)    # converting USholiday to string
df['Month']=df['Month'].apply(str)            # converting Month to string
```

In [46]:
```
import statsmodels.formula.api as sm          # importing statmodels as library
lm = sm.ols(formula='TravellersFlow ~ holidayUS + CategoryMonth + Weekday + CategoryHours',data=df).fit() # rur
lm.summary()   # summary of the model
```

Out[46]:

OLS Regression Results

Dep. Variable:	TravellersFlow	**R-squared:**	0.184
Model:	OLS	**Adj. R-squared:**	0.184
Method:	Least Squares	**F-statistic:**	1244.
Date:	Thu, 17 Nov 2022	**Prob (F-statistic):**	0.00
Time:	20:52:49	**Log-Likelihood:**	-1.7433e+05
No. Observations:	44222	**AIC:**	3.487e+05
Df Residuals:	44213	**BIC:**	3.488e+05
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-4.5707	0.153	-29.804	0.000	-4.871	-4.270
holidayUS[T.1]	2.9467	0.339	8.684	0.000	2.282	3.612
CategoryMonth[T.B]	2.2072	0.167	13.243	0.000	1.881	2.534
CategoryMonth[T.C]	5.4600	0.157	34.778	0.000	5.152	5.768
CategoryMonth[T.D]	6.5441	0.186	35.187	0.000	6.180	6.909
Weekday[T.1]	4.5429	0.131	34.727	0.000	4.286	4.799
CategoryHours[T.H2]	1.7282	0.167	10.326	0.000	1.400	2.056
CategoryHours[T.H3]	6.3332	0.166	38.130	0.000	6.008	6.659
CategoryHours[T.H4]	12.3791	0.158	78.496	0.000	12.070	12.688

Omnibus:	52617.162	**Durbin-Watson:**	0.579
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	28467963.540
Skew:	5.777	**Prob(JB):**	0.00
Kurtosis:	126.760	**Cond. No.**	6.98

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Analysing the results of OLS Model

1) 18.4 % of waiting time is explained by this model when wait time taken as numerical variable, which is not a very accurate model.

2) All the co-efficients of the model are significant as they are approaching 0 which is less than 5% and time taken increases as the variables increase baed on the categories.

3) Intercept signifies that when all X is 0, the minimum waiting time expected will still be 4.2 minutes

Evaluating the model

In [47]:
```
train = df.copy()   # copying the dataframe to "train"
```

In [50]:
```
test=pd.read_csv('C:/Users/Vara lakshmi/OneDrive/Desktop/RIT/Fall Sem/680-BANA/BANA-Assignemnt 2/bwt-taf-2015-0
test=test[test['CBSA Office']=='Queenston-Lewiston Bridge']         # filtering only Lewiston Bridge
test = test.rename({'Travellers Flow': 'TravellersFlow'}, axis=1)   # Renaming Travellers Flow to "TravellersFl
test['TravellersFlow']=test['TravellersFlow'].replace('No delay',0) # Replacing 'No Delay' with '0'
test = test[(test['TravellersFlow'] != 'Closed')]                   # Omitting "Closed" observations
test = test[(test['TravellersFlow'] != 'Missed entry')]             # Omitting "Missed entry" observations
test = test[(test['TravellersFlow'] != 'Not applicable')]           # Omitting "Not applicable" observations
test = test[(test['TravellersFlow'] != 'Temporarily closed')]       # Omitting "Temporarily closed" observatio
```

In [69]:
```
test = test.drop(['Commercial Flow','Location','CBSA Office'],axis=1)

test.reset_index(inplace = True , drop = True)               # resetting the index
test['TravellersFlow'] = test['TravellersFlow'].astype(int)    # Changing "TravellersFlow" to integer
test['Updated'] = pd.to_datetime(test['Updated'])            # Converting scalar to Date and time
```

```
test['Day'] = test['Updated'].dt.day_name()                     # Extracting Day as a variable
test['Month'] = pd.DatetimeIndex(test['Updated']).month         # Extracting Month as a variable
test['Hour'] = pd.DatetimeIndex(test['Updated']).hour           # Extracting hour as a variable
test['Weekday'] = test['Day'].apply(lambda x: 1 if x == 'Saturday' or x == 'Sunday' else 0) # Categorsing weeke
test['holidayUS'] = pd.Series(test.Updated).apply(lambda x: holidays.CountryHoliday('US').get(x)).values.astype
indexes = test[ (test['TravellersFlow'] == 135) | (test['TravellersFlow']==150)| (test['TravellersFlow']==420)]
test.drop(indexes,inplace=True) #eliminating 135,150,420 as they are outliers

#now on the basis of the values of the avg month time we will categories the month in four categories for the a
test['CategoryMonth'] = test['Month'].apply(lambda x: 'A' if x == 1 or x == 2 or x == 3 else 'B' if x == 4 or x
                                    x == 12 else 'C' if x == 5 or x == 6 or x == 7 or x == 9  else 'D')

test['Hour']=test['Hour'].apply(int)                          # converting "hour" to integer
test['CategoryHours'] = test['Hour'].apply(lambda x: 'H1' if x == 2 or x == 3 or x == 4 or x == 5 or x == 6 or
                                    else 'H2' if x == 0 or x == 1 or x == 10 or x == 11 or x == 23
                                    else 'H3' if x == 12 or x == 13 or x == 14 or x == 15 or x == 22
                                    else 'H4')

test['Hour']=test['Hour'].apply(str)                          # converting "hour" to string
test['Weekday']=test['Weekday'].apply(str)                    # converting "weekday" to integer
test['holidayUS']=test['holidayUS'].apply(str)                # converting "USholiday" to integer
test['Month']=test['Month'].apply(str)                        # converting "Month" to integer
```

In [87]:
```
train = df.copy()           # copying the data to train
X = train.loc[:, ~train.columns.isin(['TravellersFlow', 'Updated','Month','Hour','Day'])] # giving X variables
y = train['TravellersFlow'] # giving Y variables
X = pd.get_dummies(X)       # creating dummy variables for all the values in X
```

In [89]:
```
X = X.loc[:, ~X.columns.isin(['Day_Monday', 'Weekday_0','holidayUS_0','CategoryMonth_A','CategoryHours_H1'])]
x_train, x_test,y_train,y_test = train_test_split(X,y,test_size =0.1)
x_train.head() # printing the first 5 observations
```

Out[89]:

	Weekday_1	holidayUS_1	CategoryMonth_B	CategoryMonth_C	CategoryMonth_D	CategoryHours_H2	CategoryHours_H3	C
23551	1	0	1	0	0	0	1	
3164	1	0	0	0	0	0	0	
477	1	0	0	0	0	0	0	
13535	0	0	0	0	0	1	0	
37955	0	0	1	0	0	0	0	

In [90]:
```
clf = LinearRegression()     # defining the linear regression
clf.fit(x_train,y_train)     # fitting the linear regression
```

Out[90]:
```
LinearRegression()
```

In [91]:
```
clf.predict(x_test)          # predicting the test data in to train model
```

Out[91]:
```
array([-2.83991167, 17.74354335, 13.22678513, ..., 10.02761181,
        8.35706542, -2.83991167])
```

In [92]:
```
clf.score(x_test,y_test) # accuracy of training data
```

Out[92]:
```
0.1964524922892592
```

In [93]:
```
y_test
```

Out[93]:
```
13926     0
10957    45
31289    60
34913     0
3433      0
         ..
22041     0
41730     0
4731     10
28567     0
37271     0
Name: TravellersFlow, Length: 4423, dtype: int32
```

In [96]:
```
Xtest = test.loc[:, ~test.columns.isin(['TravellersFlow', 'Updated','Month','Hour','Day'])]
ytest = test['TravellersFlow']
Xtest = pd.get_dummies(Xtest)
Xtest = Xtest.loc[:, ~Xtest.columns.isin(['Day_Monday', 'Weekday_0','holidayUS_0','CategoryMonth_A','CategoryHo
```

In [97]:
```
Xtest.head()    # printing the first 5 rows of df
```

Out[97]:

	Weekday_1	holidayUS_1	CategoryMonth_B	CategoryHours_H2	CategoryHours_H3	CategoryHours_H4
0	0	0	1	1	0	0
1	0	0	1	1	0	0
2	0	0	1	1	0	0
3	0	0	1	1	0	0
4	0	0	1	1	0	0

Since the new dataset will have only months of Quarter 1, based on our earlier categorization of the variables of month, we only use the category A and B, as C and D categories have months which are not included in the first quarter.

In [98]:
```
x_test.head()   # printing the first 6 rows of df
```

Out[98]:

	Weekday_1	holidayUS_1	CategoryMonth_B	CategoryMonth_C	CategoryMonth_D	CategoryHours_H2	CategoryHours_H3	(
13926	0	0	0	0	0	1	0	
10957	1	0	0	1	0	0	0	
31289	0	0	0	1	0	0	0	
34913	0	0	0	0	0	0	0	
3433	0	0	1	0	0	0	0	

In [99]:
```
Xtest['CategoryMonth_C'] = 0 # As category for C and D is missing we will make it 0
Xtest['CategoryMonth_D'] = 0
```

In [100...
```
clf.predict(Xtest)   # predicting the values of test data
```

Out[100]:
```
array([3.0576253, 3.0576253, 3.0576253, ..., 3.0576253, 3.0576253,
       3.0576253])
```

In [62]:
```
clf.score(Xtest,ytest) #  gives the mean accuracy
```

Out[62]:
```
-1.7117368716521186
```

Analysing the prediction Results of OLS Regression

1) Based on the clf score which is -1.7 meaning the accuracy score is as low it can be meaning the built linear model is not predicting the results as accurately we need.

Logistic Regression

In [101...
```
df.head() # printing the first 6 rows
```

Out[101]:

	Updated	TravellersFlow	Day	Month	Hour	Weekday	holidayUS	CategoryMonth	CategoryHours
0	2014-04-04 13:06:00	0	Friday	4	13	0	0	B	H3
1	2014-04-04 12:05:00	0	Friday	4	12	0	0	B	H3
2	2014-04-04 11:09:00	0	Friday	4	11	0	0	B	H2
3	2014-04-04 11:07:00	0	Friday	4	11	0	0	B	H2
4	2014-04-04 11:07:00	0	Friday	4	11	0	0	B	H2

For a logistical model, we add a new column to change 'TravellersFlow' to categorize it into 0 or 1 based on our delay. Based on above EDA, we decided to go with 0-10 mins as '0' or No Delay and the rest as '1' or Delay.

In [114...
```
df['TravellersFlowBinary']=df['TravellersFlow'].apply(lambda x:0 if x<=10 else 1)        # changing timetaken t
df = df.loc[:, ~df.columns.isin(['TravellersFlow', 'Updated','Month','Hour','Day'])]
variables = "TravellersFlowBinary ~ Weekday +  holidayUS + CategoryMonth + CategoryHours "
logitModel = sm.logit(formula = variables ,data = df ).fit()                             # fitting the Logistic
logitModel.summary()                                                                     # getting summary of t
```

```
Optimization terminated successfully.
         Current function value: 0.290145
         Iterations 9
```

Out[114]:

Logit Regression Results

Dep. Variable:	TravellersFlowBinary	**No. Observations:**	44222
Model:	Logit	**Df Residuals:**	44213
Method:	MLE	**Df Model:**	8
Date:	Thu, 17 Nov 2022	**Pseudo R-squ.:**	0.2734
Time:	21:13:38	**Log-Likelihood:**	-12831.
converged:	True	**LL-Null:**	-17658.
Covariance Type:	nonrobust	**LLR p-value:**	0.000

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-6.7668	0.113	-60.067	0.000	-6.988	-6.546
Weekday[T.1]	1.1263	0.033	34.044	0.000	1.061	1.191
holidayUS[T.1]	0.5511	0.089	6.198	0.000	0.377	0.725
CategoryMonth[T.B]	0.8782	0.056	15.803	0.000	0.769	0.987
CategoryMonth[T.C]	1.7342	0.051	33.884	0.000	1.634	1.835
CategoryMonth[T.D]	2.0405	0.056	36.662	0.000	1.931	2.150
CategoryHours[T.H2]	2.0504	0.111	18.451	0.000	1.833	2.268
CategoryHours[T.H3]	3.5393	0.105	33.826	0.000	3.334	3.744
CategoryHours[T.H4]	4.3987	0.103	42.540	0.000	4.196	4.601

Analysing the results of Logistic Model

1) 27.34 % of waiting time is explained by this model when wait time taken as categorical variable, which is quite not good model but better explained then OLS model.

2) All the co-efficients of the model are significant as they are approaching 0 which is less than 5% and has positive effect on the wait time.

3) Pseudo Rsq is between 0.2 and 0.4, this represents excellent fit for the model.

Evaluating the model

In [117…

```
test['TravellersFlowBinary']=test['TravellersFlow'].apply(lambda x:0 if x<=10 else 1)       # categorising the
test = test.loc[:, ~test.columns.isin(['TravellersFlow', 'Updated','Month','Hour','Day'])]   #
test.head()
```

Out[117]:

	Weekday	holidayUS	CategoryMonth	CategoryHours	TravellersFlowBinary
0	0	0	B	H2	0
1	0	0	B	H2	0
2	0	0	B	H2	0
3	0	0	B	H2	0
4	0	0	B	H2	0

In [118…

```
y = df['TravellersFlowBinary']
X = df.loc[:, df.columns!='TravellersFlowBinary']
X = pd.get_dummies(X)
X = X.loc[:, ~X.columns.isin(['Weekday_0', 'holidayUS_0','CategoryMonth_A','CategoryHours_H1'])]
x_train, x_test,y_train,y_test = train_test_split(X,y,test_size =0.1)
```

In [119…

```
model = LogisticRegression()                           # defining the model
model.fit(x_train, y_train)                            # fitting the model
predictions = model.predict(x_test)                    # predicting the data
print(classification_report(y_test, predictions))  # printing the model
```

```
              precision    recall  f1-score   support

           0       0.89      0.99      0.93      3834
           1       0.68      0.20      0.31       589

    accuracy                           0.88      4423
   macro avg       0.78      0.59      0.62      4423
weighted avg       0.86      0.88      0.85      4423
```

In [120…
```
ytest = test['TravellersFlowBinary']
Xtest = test.loc[:, df.columns!='TravellersFlowBinary']      # "omitting TravellersFlowBinary"
Xtest = pd.get_dummies(Xtest)                                # creating dummy variables
Xtest = Xtest.loc[:, ~Xtest.columns.isin(['Weekday_0', 'holidayUS_0','CategoryMonth_A','CategoryHours_H1'])]
```

In [121…
```
Xtest['CategoryMonth_C'] = 0  # As category for C and D is missing we will make it 0
Xtest['CategoryMonth_D'] = 0
```

In [123…
```
finalPredicitons = model.predict(Xtest)                # predicting the test data
print(classification_report(ytest, finalPredicitons))  # printing the results of predictions
```

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99     38322
           1       0.00      0.00      0.00       637

    accuracy                           0.98     38959
   macro avg       0.49      0.50      0.50     38959
weighted avg       0.97      0.98      0.98     38959
```

Analysing prediction results of Logistic Regression

1) Based on the clf score which is -1.7 meaning the accuracy score is as low it can be meaning the built linear model is not predicting the results as accurately we need.

2) Precision: Correct positive predictions relative to total positive predictions

Recall: Correct positive predictions relative to total actual positives

F1-score gives the accuracy of the model.The higher the score,the higher the accuracy.

Conlusions and Comparison of Models

After training the model, the precision of 89 Precent was achieved i.e. the model predicted no delay with a 89 % accuracy for 10% of the data which was tested in the same data set. The same model with data of Quarter 1, 2015 provided a precision of 98 % when there was no delay . The accuracy of Linear regression model is negative whereas in the case of logistic regression the accuracy was 98 %

The accuracy in case of linear regression can be increased if we convert the predicted value to the nearest values which are multiplied by either 5 or 10 and then compare it with the wait time values .

The Logistic model can be improved if more variance for dependent variable can be explained by reducing the independent variables used for regression. If this method doesn't improve the variance then the best method is to keep all the independent variable or try to categorize this variables more accurately.This in turn will improve the prediction.

The test data was only taken for quarter 1 , if data for entire year was considered , the accuracy score in case of logistic regression might have increased.

Additional features to improve the model

1) We can add more dependent variables in order to further improve the accuracy and see impacts on the flow.

2) Using feature encoding, where we treat all the. variables into groups. [ex. Italy 1, India 2, USA 3 etc]

3) Selecting features which affect the model the most and standardizing them. This will prevent values from over powering each other.

In [ ]: