RIT - BANA680; Date: 10/28/2022; Contributors: Fei Manheche, Pranav Belmannu, Prathyusha Netheti, Sai Likhitha Devireddy

Regression Analysis

Introduction:

Time is the most crucial element in today's world and everything revolves around how to maximise efficiency.One important issue we face today is waiting at Borders or tolls during rush hours.In this project we aim to create a model to undestand at what time of the day or during which day of the month there is a maximum rush. This helps the traveller to plan their itenarary accordingly there by reducing one's border wait time.

About the Data:

- This data has been collected by the Government of Canada.The data presented here has been collected from 2010-2014.
- The data includes the location of Canada Border Service Agency(CBSA),location of the border crossing,date and time of crossing the,type of travellers vehicle and their respective wait time.
- The date and time have been collected from different time zones but we will be using the Queenston-Lewiston Bridge crossing data in this project.
- The travellers flow column has different wait time values which includes No Delays,Not applicable and Closed.
- For test data analysis we have used the border crossing data from the year 2015.

Scope of Analysis:

- In this analysis an EDA will be conducted on the data and see how each variable affects the wait time and analyse why that variable is affecting the wait time.
- Creating two types of regression models(OLS AND Logistic) with Y(wait time) as dependent variable
- Analysing the outputs of both the regression models

In [1]:
```
# Dependency for calendar holidays for canada.
#pip install holidays
```

In [2]:
```
#importing required libraries
import holidays
import numpy as np
import pandas as pd
import dateutil
import re
```

In [3]:
```
# Reading the data from the stored path
# dir = 'C:/Users/Likitha/Downloads/'
dir = '/Users/fei/Library/CloudStorage/OneDrive-Personal/Documents/RIT/BANA 680 Data Management for Business Analytics/Assignment
bwt_data = 'bwt-taf-2010-2014-eng.csv'

# Data Frames creation
df = pd.read_csv(dir+bwt_data)
```

Data Cleanup of aggregated data in dataset

In this analysis only the border wait time of Lewiston Bridge crossing is required,to extract that we perform the following analysis:

- We have extracted the data of the Lewiston Bridge
- Dropped the values of Not applicable and Closed as they don't add value to the analysis
- Check for the days which are closed

In [4]:
```
# We only want data from Lewiston Bridge crossing for now
indexFiltered = df[df['CBSA Office'] != "Queenston-Lewiston Bridge"].index
df.drop(indexFiltered, axis =0,  inplace=True)
print(df.shape) # Shape before filtering
# df.head()
```

```
(44234, 5)
```

In [5]:
```
df[df['Travellers Flow']=='Closed'].head(2) # Output reduced due to space restriction
```

Out[5]:

	CBSA Office	Location	Updated	Commercial Flow	Travellers Flow
403444	Queenston-Lewiston Bridge	Queenston, ON	2011-04-29 08:35 EDT	Closed	Closed
403446	Queenston-Lewiston Bridge	Queenston, ON	2011-04-29 08:35 EDT	Closed	Closed

We can see that Queenston -Lewston Bridge has been closed from 28th April 2011 morning to 29th April 2011 morning as it was a National mourning day

In [6]:
```
print(df.shape) # Shape before filtering

from dateutil import parser
# Datetime cleanup; Make Date field a Pandas date field with UTC time as standard
# Maybe there is a more efficient method, but works for now - but takes a while to process...
tzmapping = {'ADT': dateutil.tz.gettz('Canada/Atlantic'),
             'CDT': dateutil.tz.gettz('US/Central'),
             'MDT': dateutil.tz.gettz('America/Denver'),
             'PDT': dateutil.tz.gettz('US/Pacific')}
df['Updated'] = df['Updated'].apply(parser.parse, tzinfos=tzmapping)

# Create a copy of the df for Logistic regression where we drop the no delay and other irrelevant cases
df1 = df.copy(deep=True) # let's do a deep copy to not interfere in results

# TO-DO: Analyze
indexFiltered = df[(df['Travellers Flow'] == "Not Applicable") |
        (df['Travellers Flow'] == "Closed")].index
df.drop(indexFiltered, axis =0,  inplace=True)

indexFiltered = df1[(df1['Travellers Flow'] == "Not Applicable") |
        (df1['Travellers Flow'] == "Closed")].index
df1.drop(indexFiltered, axis =0,  inplace=True)


# df1.loc[df['Travellers Flow'] == "Delay", 'Travellers Flow'] = 1

# Change values for the text values on the Travellers flow columns
df.loc[(df['Travellers Flow'] == "No Delay"), 'Travellers Flow'] = 0
df1.loc[(df1['Travellers Flow'] == "No Delay"), 'Travellers Flow'] = 0

# Check if it is clean: returns nothing is good!
df.loc[(df['Travellers Flow'] == "No Delay") |
        (df['Travellers Flow'] == "Not Applicable") |
        (df['Travellers Flow'] == "Closed")]

# df after cleanup to check
print(df.shape) # Shape before filtering
print(df1.shape) # Shape before filtering

# df['Updated'] = pd.to_datetime(df['Updated'], utc=True)
df.head()
df1.head(2)
# We have 2 dfs for the different regression analysis we want to do
# For OLS - df and for Logistic - df1
```

```
(44234, 5)
(44222, 5)
(44222, 5)
```

Out[6]:

	CBSA Office	Location	Updated	Commercial Flow	Travellers Flow
369413	Queenston-Lewiston Bridge	Queenston, ON	2014-04-04 13:06:00-04:00	No Delay	0
369414	Queenston-Lewiston Bridge	Queenston, ON	2014-04-04 12:05:00-04:00	No Delay	0

In [7]:
```
# df1['Travellers Flow'].value_counts() # Output ommitted due to space restriction
```

In [8]:
```
# Add date column to both dfs
df['Date'] = pd.to_datetime(df['Updated']).dt.date
df1['Date'] = pd.to_datetime(df1['Updated']).dt.date

# Add Weekday vs weekend check
df["IsWeekend"] = pd.to_datetime(df["Date"]).dt.weekday >= 5
df1["IsWeekend"] = pd.to_datetime(df1["Date"]).dt.weekday >= 5

df['Travellers Flow'] = df['Travellers Flow'].astype(int)
#Create time variable
df['time']=pd.to_datetime(df['Updated']).dt.time
df1['time']=pd.to_datetime(df1['Updated']).dt.time
#Split date time attributes
df['HourOfDay'] = df['Updated'].dt.hour
df['Year'] = df['Updated'].dt.year
df['Month'] = df['Updated'].dt.month
df['DayOfMonth'] = df['Updated'].dt.day
df['DayOfWeek'] = df['Updated'].dt.dayofweek
# df.info()
# df.head()
#create month variable
df1['HourOfDay'] = df1['Updated'].dt.hour
df1['Year'] = df1['Updated'].dt.year
df1['Month'] = df1['Updated'].dt.month
```

```
df1['DayOfMonth'] = df1['Updated'].dt.day
df1['DayOfWeek'] = df1['Updated'].dt.dayofweek

# df1.head(3) # Output ommitted due to space restriction
```

In [9]:
```
# Let's add a column for CANADA Bank holiday checks
import holidays

#cal = holidays.CA()
df['Date'] = pd.to_datetime(df['Date']).dt.normalize()
canada_holidays = holidays.Canada()

# df.dtypes
df['isCanadaHoliday'] = df['Date'].isin(canada_holidays)
df.head()

#cal = holidays.CA()
df1['Date'] = pd.to_datetime(df1['Date']).dt.normalize()
canada_holidays = holidays.Canada()

# df.dtypes
df1['isCanadaHoliday'] = df1['Date'].isin(canada_holidays)
# df1.head(3) # Output ommitted due to space restriction
```

In [10]:
```
# Let's add a column for US Federal Bank holiday checks
from pandas.tseries.holiday import USFederalHolidayCalendar as calendar
cal = calendar()
df['Date'] = pd.to_datetime(df['Date']).dt.normalize()
holidays = cal.holidays(df['Date'].min(), df['Date'].max())

# df.dtypes
df['isUSHoliday'] = df['Date'].isin(holidays)
df.head()

df1['Date'] = pd.to_datetime(df1['Date']).dt.normalize()
holidays = cal.holidays(df1['Date'].min(), df1['Date'].max())

# df.dtypes
df1['isUSHoliday'] = df1['Date'].isin(holidays)
# df1.head(2) # Output ommitted due to space restriction
```

In [11]:
```
#calculating mean and assigning it to variable'm'
m=pd.to_numeric(df1['Travellers Flow']).mean()
m
```

Out[11]:
```
5.280290353217856
```

EDA

To understand how each variable has an affect on the wait time we have plotted the following analysis to understand the data.

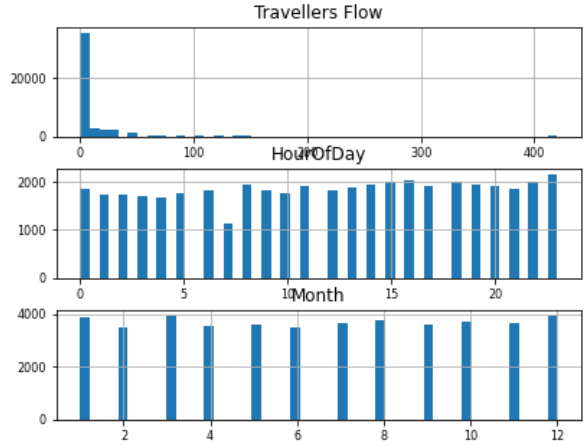
In [12]:
```
#importing required libraries
import seaborn as sns
import matplotlib.pyplot as plt
import warnings # to ignore internal issues that comes up as warning
warnings.filterwarnings('ignore')

df_quant = df.select_dtypes(include = ['datetime64', 'int64', 'float64'])
print('Number of quantitative variables = ',len(df_quant.columns))
print('List of quantitative variables:\n', df_quant.columns.tolist())
df_quant.head(1)
```

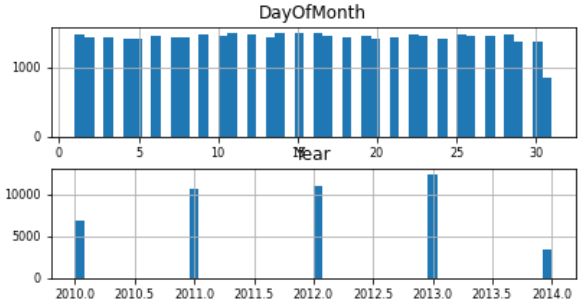
```
Number of quantitative variables =  7
List of quantitative variables:
 ['Travellers Flow', 'Date', 'HourOfDay', 'Year', 'Month', 'DayOfMonth', 'DayOfWeek']
```

Out[12]:

	Travellers Flow	Date	HourOfDay	Year	Month	DayOfMonth	DayOfWeek
369413	0	2014-04-04	13	2014	4	4	4

In [13]:
```
getvars = ['Travellers Flow', 'DayOfMonth', 'HourOfDay', 'Year', 'Month']
df_tmp = df[getvars]
df_tmp.hist(figsize=(15, 5), bins=50, xlabelsize=8, ylabelsize=8)
plt.show()
```





we can see that from the above graphs travellers flow has been more on some days or during some hours.

- In the 'Hour of Day' graph the flow is more during before and after usual office working hours i.e 9am -5 pm.This could be because most of the travellers might have made their plans to travel before or after work.
- In the 'Month' graph we can observe that there is more flow during January,March,July,August and December.This could be due to the summer and winter break during the reespective seasons.

To check for those variables which are qualitative in nature which are also called categorical variables the following query has been performed.

In [14]:
```
df_qual = df.select_dtypes(include = ['O'])
print('Number of qualitative variables = ',len(df_qual.columns))
print('List of qualitative variables:\n', df_qual.columns.tolist())
# df_qual.head(1)
```

```
Number of qualitative variables =  4
List of qualitative variables:
 ['CBSA Office', 'Location', 'Commercial Flow', 'time']
```

In [15]:
```
df_qual['Location'].unique()
```

Out[15]:
```
array(['Queenston, ON'], dtype=object)
```

To find the corelartion between travellers flow and Day of Month,Hour of Day,and year

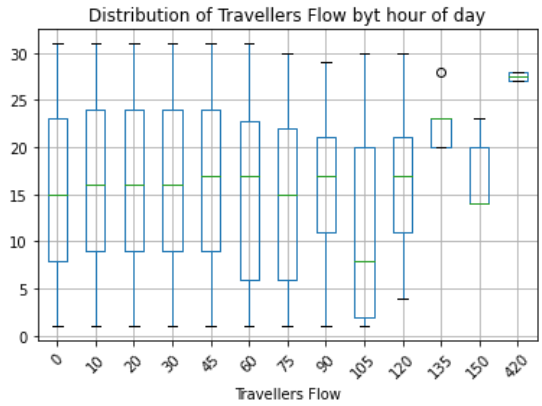
In [16]:
```
df_tmp_corr = df_tmp.corr()['HourOfDay'][:-1]
print(df_tmp_corr,'\n')
```

```
Travellers Flow    0.274833
DayOfMonth         0.000875
HourOfDay          1.000000
Year               0.019881
Name: HourOfDay, dtype: float64
```

From the above output it could be seen that travellers flow is more correlated with 'HourofDay' compared to 'Year' and ' dayofMonth'

In [17]:
```
# print('Value counts for Travellers Flow:\n', df_tmp['HourOfDay'].value_counts())
plt.figure(figsize = (10, 4))
df_tmp.boxplot(column='DayOfMonth',by='Travellers Flow')
plt.xticks(rotation=45)
plt.title('Distribution of Travellers Flow byt hour of day')
plt.suptitle('')
plt.show()
```

```
<Figure size 720x288 with 0 Axes>
```



- This box plot depicts number of travellers by day of the month. It can be seen that there are no travellers in the beginning and end of the month indicating delay time is more in middle of the month

Data cleanup of Test data

Cleaning the test data in order to make it fit for using during Prediction analysis

In [18]:
```
bwt_2015_data_url = "https://www.cbsa-asfc.gc.ca/data/bwt-taf-2015-01-01--2015-03-31-en.csv"
dft = pd.read_csv(bwt_2015_data_url)  # This will download and create our df directly from bwt cbsa site

dft.head(2)
```

Out[18]:

	CBSA Office	Location	Updated	Commercial Flow	Travellers Flow
0	Abbotsford-Huntingdon	Huntingdon, BC	2014-12-31 20:49 PST	No delay	5
1	Abbotsford-Huntingdon	Huntingdon, BC	2014-12-31 21:49 PST	No delay	5

In [19]:
```
# We only want data from Lewiston Bridge crossing for now
indexFilteredt = dft[dft['CBSA Office'] != "Queenston-Lewiston Bridge"].index
dft.drop(indexFilteredt, axis =0,  inplace=True)
print(dft.shape) # Shape before filtering
dft.head(2)
```

```
(38969, 5)
```

Out[19]:

	CBSA Office	Location	Updated	Commercial Flow	Travellers Flow
110480	Queenston-Lewiston Bridge	Queenston, ON	2014-12-31 23:18 EST	No delay	No delay
110481	Queenston-Lewiston Bridge	Queenston, ON	2014-12-31 23:20 EST	No delay	No delay

In [20]:
```
print(dft.shape) # Shape before filtering

from dateutil import parser
# Datetime cleanup; Make Date field a Pandas date field with UTC time as standard
# Maybe there is a more efficient method, but works for now - but takes a while to process...
tzmapping = {'ADT': dateutil.tz.gettz('Canada/Atlantic'),
             'CDT': dateutil.tz.gettz('US/Central'),
             'MDT': dateutil.tz.gettz('America/Denver'),
             'PDT': dateutil.tz.gettz('US/Pacific')}
dft['Updated'] = dft['Updated'].apply(parser.parse, tzinfos=tzmapping)

# Create a copy of the df for Logistic regression where we drop the no delay and other irrelevant cases
dft1 = dft.copy(deep=True) # let's do a deep copy to not interfere in results

#only two records are new in prediction dataset i.e. No delay and Missed entry, we shall drop missing values and add 0 to no dela

indexFiltered = dft[(dft['Travellers Flow'] =='Missed entry')].index
dft.drop(indexFiltered, axis =0,  inplace=True)

indexFiltered = dft1[(dft1['Travellers Flow'] =='Missed entry')].index
dft1.drop(indexFiltered, axis =0,  inplace=True)

# Change values for the text values on the Travellers flow columns
dft.loc[(dft['Travellers Flow'] == "No delay"), 'Travellers Flow'] = 0
dft1.loc[(dft1['Travellers Flow'] == "No delay"), 'Travellers Flow'] = 0

# Check if it is clean: returns nothing is good!
dft.loc[(dft['Travellers Flow'] == "No delay") |
        (dft['Travellers Flow'] == 'Missed entry')]

# df after cleanup to check
print(dft.shape) # Shape before filtering
print(dft1.shape) # Shape before filtering

# df['Updated'] = pd.to_datetime(df['Updated'], utc=True)
dft.head()

# dft1.head() # Output ommitted due to space restriction
# We have 2 dfs for the different regression analysis we want to do
# For OLS - dft and for Logistic - dft1
```

```
(38969, 5)
(38959, 5)
(38959, 5)
```

Out[20]:

	CBSA Office	Location	Updated	Commercial Flow	Travellers Flow
110480	Queenston-Lewiston Bridge	Queenston, ON	2014-12-31 23:18:00-05:00	No delay	0
110481	Queenston-Lewiston Bridge	Queenston, ON	2014-12-31 23:20:00-05:00	No delay	0
110482	Queenston-Lewiston Bridge	Queenston, ON	2014-12-31 23:23:00-05:00	No delay	0
110483	Queenston-Lewiston Bridge	Queenston, ON	2014-12-31 23:25:00-05:00	No delay	0
110484	Queenston-Lewiston Bridge	Queenston, ON	2014-12-31 23:28:00-05:00	No delay	0

In [21]:
```
# Add date column to both dfs
dft['Date'] = pd.to_datetime(dft['Updated']).dt.date
```

```
dft1['Date'] = pd.to_datetime(dft1['Updated']).dt.date

# Add Weekday vs weekend check
dft["IsWeekend"] = pd.to_datetime(dft["Date"]).dt.weekday >= 5
dft1["IsWeekend"] = pd.to_datetime(dft1["Date"]).dt.weekday >= 5

dft['Travellers Flow'] = dft['Travellers Flow'].astype(int)
#Create time variable
dft['time']=pd.to_datetime(dft['Updated']).dt.time
dft1['time']=pd.to_datetime(dft1['Updated']).dt.time
#Split date time attributes
dft['HourOfDay'] = dft['Updated'].dt.hour
dft['Year'] = dft['Updated'].dt.year
dft['Month'] = dft['Updated'].dt.month
dft['DayOfMonth'] = dft['Updated'].dt.day
dft['DayOfWeek'] = dft['Updated'].dt.dayofweek
# df.info()
# df.head()
#create month variable
dft1['HourOfDay'] = dft1['Updated'].dt.hour
dft1['Year'] = dft1['Updated'].dt.year
dft1['Month'] = dft1['Updated'].dt.month
df1['DayOfMonth'] = dft1['Updated'].dt.day
dft1['DayOfWeek'] = dft1['Updated'].dt.dayofweek

# dft1.head() # Output ommitted due to space restriction
```

In [22]:
```
 #Let's add a column for CANADA Bank holiday checks
import holidays
dft['Date'] = pd.to_datetime(dft['Date']).dt.normalize()
canada_holidays = holidays.Canada()

dft['isCanadaHoliday'] = dft['Date'].isin(canada_holidays)
# dft.head() # Output ommitted due to space restriction
```

In [23]:
```
# Let's add a column for US Federal Bank holiday checks
from pandas.tseries.holiday import USFederalHolidayCalendar as calendar
cal = calendar()
dft['Date'] = pd.to_datetime(dft['Date']).dt.normalize()

holidays = cal.holidays(dft['Date'].min(), dft['Date'].max())
# df.dtypes
dft['isUSHoliday'] = dft['Date'].isin(holidays)
dft.head()
dft1['Date'] = pd.to_datetime(dft1['Date']).dt.normalize()
holidays = cal.holidays(dft1['Date'].min(), dft1['Date'].max())
# df.dtypes
dft1['isUSHoliday'] = dft1['Date'].isin(holidays)
# dft1.head() # Output ommitted due to space restriction
```

Divided the delay time of travellers flow into high's and low's by creating a threshold value which is measured as mean of Traveller's flow stored in variable 'm'

In [24]:
```
dft1.loc[pd.to_numeric(dft1['Travellers Flow'])<=m, 'y']= 'low'
dft1.loc[pd.to_numeric(dft1['Travellers Flow']) >m, 'y']= 'high'
```

Converting the high's and low's to 1's and 0's for the logistic model

In [25]:
```
dft1.loc[dft1['y'] =='low', 'yn'] = 0
dft1.loc[dft1['y'] =='high', 'yn'] = 1 #yn numerical
```

In [26]:
```
dft1['yn'] = pd.to_numeric(dft1['yn'])
dft1.head(2) # Output limited due to space restriction
```

Out[26]:

	CBSA Office	Location	Updated	Commercial Flow	Travellers Flow	Date	IsWeekend	time	HourOfDay	Year	Month	DayOfWeek	isUSHoliday	y
110480	Queenston-Lewiston Bridge	Queenston, ON	2014-12-31 23:18:00-05:00	No delay	0	2014-12-31	False	23:18:00	23	2014	12	2	False	low
110481	Queenston-Lewiston Bridge	Queenston, ON	2014-12-31 23:20:00-05:00	No delay	0	2014-12-31	False	23:20:00	23	2014	12	2	False	low

OLS Regression

Linear regression analysis is used to predict the value of a variable based on the value of another variable. The variable you want to predict is called the dependent variable. The variable you are using to predict the other variable's value is called the independent variable.

This form of analysis estimates the coefficients of the linear equation, involving one or more independent variables that best predict the value of the dependent variable. Linear regression fits a straight line or surface that minimizes the discrepancies between predicted and actual output values. There are simple linear regression calculators that use a “least squares” method to discover the best-fit line for a set of paired data. You then estimate the value of X (dependent variable) from Y (independent variable).

OLS is a most commonly used statistical models. The main idea is to fit all dependent variables or factors affecting data on a line using equation in the form of Y=b0+b1X...

In this project, we will model Y(Wait time/travellers flow) in the form of linear equation using variables like hour(time of the day,month, weekday/weekend and holiday.

In [27]:
```
#import the statsmodel package
import statsmodels.formula.api as sm
```

In [28]:
```
df=df.rename(columns={'Travellers Flow':'Travellers_Flow'})
f1='Travellers_Flow~(HourOfDay+Month+IsWeekend+isUSHoliday)'
result =sm.ols(formula=f1,data=df).fit()
result.summary()
```

Out[28]:

OLS Regression Results

Dep. Variable:	Travellers_Flow	**R-squared:**	0.103
Model:	OLS	**Adj. R-squared:**	0.103
Method:	Least Squares	**F-statistic:**	1270.
Date:	Fri, 18 Nov 2022	**Prob (F-statistic):**	0.00
Time:	11:38:19	**Log-Likelihood:**	-1.7641e+05
No. Observations:	44222	**AIC:**	3.528e+05
Df Residuals:	44217	**BIC:**	3.529e+05
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-4.4093	0.174	-25.294	0.000	-4.751	-4.068
IsWeekend[T.True]	4.5705	0.138	33.218	0.000	4.301	4.840
isUSHoliday[T.True]	2.2952	0.380	6.044	0.000	1.551	3.039
HourOfDay	0.5455	0.009	60.967	0.000	0.528	0.563
Month	0.2757	0.018	15.457	0.000	0.241	0.311

Omnibus:	50760.377	**Durbin-Watson:**	0.546
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	20118174.287
Skew:	5.500	**Prob(JB):**	0.00
Kurtosis:	106.911	**Cond. No.**	92.0

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

From the above OLS output R squared value is 0.103 which is not an ideal value for a linear regression.

- Although all the variables included in the equation are significant we might build a better model other than Linear regression to fit the data accurately

Performing a prediction analysis using test data for the above OLS model

In [29]:
```
#predicting results using dft test data
ypred = result.predict(dft)
ypred.head(3)
```

Out[29]:
```
110480    11.444703
110481    11.444703
110482    11.444703
dtype: float64
```

Logistic regression:

Logistic Regression was used in the biological sciences in the early twentieth century. It was then used in many social science applications. Logistic Regression is used when the dependent variable(target) is categorical. For example,

- To predict whether an email is spam (1) or not (0)

- Whether the tumor is malignant (1) or not (0)

Consider a scenario where we need to classify whether an email is a spam or not. If we use linear regression for this problem, there is a need for setting up a threshold based on which classification can be done. Say if the actual class is malignant, the predicted continuous value is 0.4 and the threshold value is 0.5, the data point will be classified as not malignant which can lead to serious consequences in real-time.

In [30]:
```
#To know what are the unique values in each column
# for col in df1:
#   print(df1[col].unique()) # Output ommitted due to space restriction
```

In [31]:
```
df1['Travellers Flow'] = pd.to_numeric(df1['Travellers Flow'])
m=df1['Travellers Flow'].mean()
m
```

Out[31]:
```
5.280290353217856
```

In [32]:
```
df1.loc[df1['Travellers Flow'] <=m, 'y'] = 'low'
df1.loc[df1['Travellers Flow'] >m, 'y'] = 'high'
df1.head(3)
```

Out[32]:

	CBSA Office	Location	Updated	Commercial Flow	Travellers Flow	Date	IsWeekend	time	HourOfDay	Year	Month	DayOfMonth	DayOfWeek	isCan
369413	Queenston-Lewiston Bridge	Queenston, ON	2014-04-04 13:06:00-04:00	No Delay	0	2014-04-04	False	13:06:00	13	2014	4	NaN	4	
369414	Queenston-Lewiston Bridge	Queenston, ON	2014-04-04 12:05:00-04:00	No Delay	0	2014-04-04	False	12:05:00	12	2014	4	NaN	4	
369415	Queenston-Lewiston Bridge	Queenston, ON	2014-04-04 11:09:00-04:00	No Delay	0	2014-04-04	False	11:09:00	11	2014	4	NaN	4	

In [33]:
```
df1.loc[df1['y'] =='low', 'yn'] = 0
df1.loc[df1['y'] =='high', 'yn'] = 1
#yn numerical
```

In [34]:
```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import confusion_matrix
```

In [35]:
```
df1['IsWeekend'] = pd.to_numeric(df1['IsWeekend'])
df1['isUSHoliday'] = pd.to_numeric(df1['isUSHoliday'])
df1['Month'] = pd.to_numeric(df1['Month'])
df1['HourOfDay'] = pd.to_numeric(df1['HourOfDay'])
df1['isCanadaHoliday'] = pd.to_numeric(df1['isCanadaHoliday'])
```

In [36]:
```
logit_model=sm.logit(formula="yn~IsWeekend + isUSHoliday +Month +HourOfDay", data=df1).fit()
```

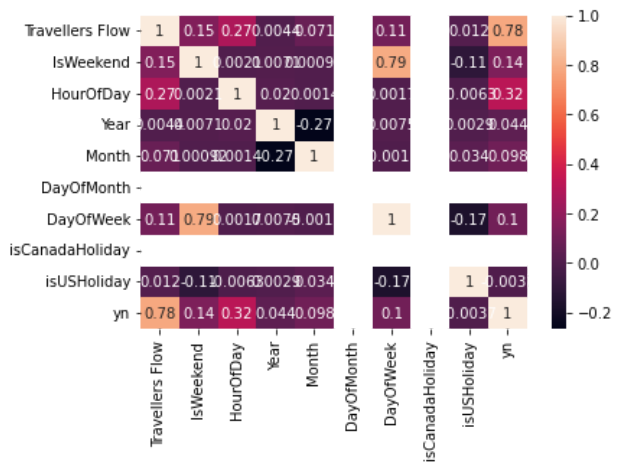
```
Optimization terminated successfully.
         Current function value: 0.421416
         Iterations 7
```

In [37]:
```
df1.corr()
```

Out[37]:

	Travellers Flow	IsWeekend	HourOfDay	Year	Month	DayOfMonth	DayOfWeek	isCanadaHoliday	isUSHoliday	yn
Travellers Flow	1.000000	0.148152	0.274833	0.004437	0.071120	NaN	0.107026	NaN	0.011724	0.781708
IsWeekend	0.148152	1.000000	0.002056	0.007082	0.000915	NaN	0.791133	NaN	-0.108711	0.138859
HourOfDay	0.274833	0.002056	1.000000	0.019881	0.001380	NaN	0.001656	NaN	-0.006265	0.315710
Year	0.004437	0.007082	0.019881	1.000000	-0.266617	NaN	0.007455	NaN	0.002901	0.044242
Month	0.071120	0.000915	0.001380	-0.266617	1.000000	NaN	-0.001642	NaN	0.034488	0.097696
DayOfMonth	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
DayOfWeek	0.107026	0.791133	0.001656	0.007455	-0.001642	NaN	1.000000	NaN	-0.171274	0.102768
isCanadaHoliday	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
isUSHoliday	0.011724	-0.108711	-0.006265	0.002901	0.034488	NaN	-0.171274	NaN	1.000000	-0.003717
yn	0.781708	0.138859	0.315710	0.044242	0.097696	NaN	0.102768	NaN	-0.003717	1.000000

In [52]:
```
import seaborn as sns

# checking correlation using heatmap
#plotting the heatmap for correlation
ax = sns.heatmap(df1.corr(), annot=True)
```



In [39]:
```
# df1['HourOfDay'].value_counts()
```

In [40]:
```
print(logit_model.summary())
```

```
                           Logit Regression Results
==============================================================================
Dep. Variable:                     yn   No. Observations:                44222
Model:                          Logit   Df Residuals:                    44217
Method:                           MLE   Df Model:                            4
Date:                Fri, 18 Nov 2022   Pseudo R-squ.:                  0.1416
Time:                        11:38:20   Log-Likelihood:                -18636.
converged:                       True   LL-Null:                       -21710.
Covariance Type:            nonrobust   LLR p-value:                     0.000
==========================================================================================
                             coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------------
Intercept                 -4.1774      0.048    -86.490      0.000      -4.272      -4.083
IsWeekend[T.True]          0.8314      0.027     30.465      0.000       0.778       0.885
isUSHoliday[T.True]        0.1733      0.080      2.156      0.031       0.016       0.331
Month                      0.0812      0.004     21.575      0.000       0.074       0.089
HourOfDay                  0.1387      0.002     62.817      0.000       0.134       0.143
==========================================================================================
```

- The pseuod R-squared value for this model is 0.1416 which is not satisfactory but it fits better than Linear Regression

Training and testing the model for predictive analysis

In [41]:
```
# Load the tests data to train
y_train = df1['yn']
x_train = df1[['IsWeekend', 'isUSHoliday', 'Month' ,'HourOfDay']]
```

In [42]:
```
# Load the test data to perform predictive analysis
y_test = dft1['yn']
x_test = dft1[['IsWeekend', 'isUSHoliday', 'Month' ,'HourOfDay']]
```

In [43]:
```
logreg = LogisticRegression()
logreg.fit(x_train, y_train)
```

Out[43]:
```
LogisticRegression()
```

In [44]:
```
# Fit (train) model
reg = LogisticRegression()
reg.fit(x_train, y_train)
```

Out[44]:
```
LogisticRegression()
```

In [45]:
```
# Prediction based on test data
pred = reg.predict(x_test)
pred
```

Out[45]:
```
array([0., 0., 0., ..., 0., 0., 0.])
```

In [46]:
```
# To know the accuracy of the model
accuracy = reg.score(x_test, y_test)
accuracy
```

Out[46]:
```
0.9523345055057881
```

- From the given data the model is 95% accurate

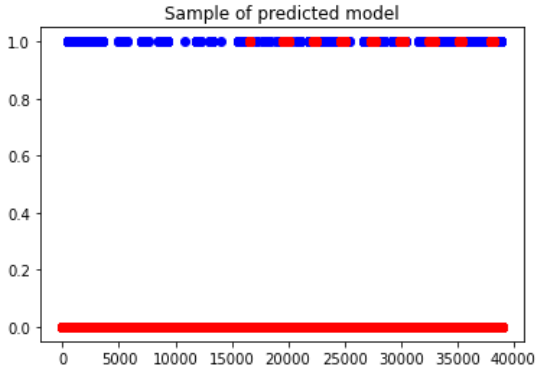
In [47]:
```
y_pred = logreg.predict(x_test)
y_pred
```

Out[47]:
```
array([0., 0., 0., ..., 0., 0., 0.])
```

In [48]:
```
# df1.plot.scatter(y='HourOfDay',x='Travellers Flow', figsize = (16,8))
```

In [49]:
```
# df1.plot.scatter(y='Travellers Flow',x='Date', figsize = (16,6))
```

In [51]:
```
plt.scatter(range(len(y_test)), y_test, color='blue')
plt.scatter(range(len(y_pred)), y_pred, color='red')
plt.title("Sample of predicted model")
plt.show()
```



Analysis of OLS and Logistic Regression

In OLS regression, a linear relationship between the dependent and independent variable is a must, but in logistic regression, one does not assume such things. The relationship between the dependent and independent variable may be linear or non-linear. OLS assumes that the distribution should be normally distributed, but in logistic regression, the distribution may be normal, poisson, or binominal. OLS assumes that there is an equal variance between all independent variables, but ordinal logistic does not assume that there is an equal variance between independent variables.

Linear regression is not a better fit model as seen from the above OLS output. Logistic is performed to overcome the drawbacks of Linear regression and to create a better fit model. But from the Pseudo-R squared value of Logistic regression we can see that the model is not satisfactory. To improve this R-value more variables/features need to be included in calculating the Y variable.

Additional Features to Improve the model fit

- Time of the day and month can be converted into categorical variable since there are 24 hours and 12 months respectively
- Commercial Flow data can be included to the model and see it's impact on traveller's flow
- Time taken to travel between two bridges.
- No. of tolls/check points to reach the destination
- No of cars waiting on the bridge
- Data on factors which might affect wait time like Accidents, Road maintenance, and weather

Conclusion

- Through this project an EDA has been conducted on the data and it can be seen that there are certain days and certain hours with significance where there is maximum rush, which might explain the delay in Traveller's flow.
- From the analysis we observed that during holiday season or before and after office hours saw maximum delay in Travellers flow.
- Two regression models has been built on which prediction analysis has been performed.The data from 2015 has been used to test these models and the Logistic Regression model has given us 95% accuracy in prediction.
- While the prediction analysis has been done based on these models, the models has overall not been satisfactory due to insufficient data.To improve the model fit we might need more data as mentioned in the Additional features.

In [ ]: