Assignment 6 - Advanced Data Management & Regressions

By Dana DiVincenzo, Pyone Myat Maw, and Chithira Pugazhenthi

Introduction

In this assignment, we were given a csv file which contained information about traffic at the Canadian-US border. This data has been collected by the Canada Government Agency, and wait times are one of the statistical variables that can be useful for visitors and travelers to plan ahead their times to cross through the border. The goal of the assignment was to use feature engineering to create data that could be used in an OLS and Logistic regression equation to predict delay times at the Queenston-Lewiston Bridge location, based on factors such as the location of crossing, date and time of crossing, and whether the passenger was commercial or travel. The results and possible improvements to the models were then discussed.

Setup and Prepwork

In this section we import useful packages, clean the data, add columns of data, and do further manipulation to make later analysis easier.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.formula.api as sm
from datetime import timezone
import holidays
import datetime
from statsmodels.formula.api import ols , logit
import warnings # scipy warnings to be skipped
warnings.filterwarnings('ignore')
import io

warnings.filterwarnings('ignore')
mydir = "C:/Users/Owner/Downloads/"
f1 = "bwt-taf-2010-2014-eng.csv"
fTrafficFlow = mydir + f1
dTrafficFlow =pd.read_csv(fTrafficFlow)
```

In [2]:
```
# filtering only "Queenston-Lewiston Bridge" from dataset into dataframe
dTrafficFlow= dTrafficFlow[(dTrafficFlow['CBSA Office'] == 'Queenston-Lewiston
Bridge') & (dTrafficFlow.Location == "Queenston, ON")]
# reseting the index numbers
dTrafficFlow.reset_index(inplace = True)
# removing index column
dTrafficFlow = dTrafficFlow.drop(columns="index")
# renaming the columns to remove spaces
dTrafficFlow.rename(columns = {'Commercial Flow':'Commercial_Flow'}, inplace =
True)
dTrafficFlow.rename(columns = {'Travellers Flow':'Travellers_Flow'}, inplace =
True)
```

In [3]:
```
# converting contents of "Updated" column to time format
dTrafficFlow.Updated = pd.to_datetime(dTrafficFlow.Updated)
# creating "date" &"time" columns in dataframe
dTrafficFlow['Date'] = pd.to_datetime(dTrafficFlow['Updated']).dt.date
dTrafficFlow['Time'] = pd.to_datetime(dTrafficFlow['Updated']).dt.time
```

In [4]:
```
# creating "Hour","DayofWeek" & "Month" columns in dataframe
dTrafficFlow['Hour'] =  pd.to_datetime(dTrafficFlow['Updated']).dt.strftime("%
H")
dTrafficFlow['DayofWeek'] =  pd.to_datetime(dTrafficFlow['Updated']).dt.strfti
me("%w")
dTrafficFlow['Month'] =  pd.to_datetime(dTrafficFlow['Updated']).dt.strftime(
"%m")
```

In [5]:
```
# setting a list to store all the dates of holidays
USHolidays = list(holidays.US(years = dTrafficFlow.Updated.dt.year.unique()).k
eys())
# creating "isHoliday",& "isWeekend" columns in dataframe
dTrafficFlow['isHoliday'] = dTrafficFlow['Updated'].isin(USHolidays)
weekends = ["0","6"] # Saturday -6 Sunday-0
dTrafficFlow['isWeekend'] = dTrafficFlow['DayofWeek'].isin(weekends)
```

Combining Commercial Flow and Travellers Flow into one delay column

We wanted to have one delay column that represented both commercial flow and travellers flow. However, the values in those columns were a variety of numbers, along with the strings "no delay," "closed," and "not applicable," so we could not just simply combine the columns. First, we had to make 2 copies of the data frame, one which focused on commercial and one which focused on travellers, and in each one we dropped all of the "closed" and "not applicable" values. We then changed all of the "no delay" values to 0, because it represented a 0 minute wait time. We were then able to find the true averages of the wait times for travellers flow and commercial flow. We then used those averages in the original dataframe to replace all of the "not applicable" and "closed" values, which allowed us to keep the average for the column accurate without having to drop too many rows.

In [6]:
```
comFlowdf = dTrafficFlow.copy(deep=True)
```

In [7]:
```
comFlowdf.drop(comFlowdf[comFlowdf['Commercial_Flow'] == 'Not Applicable'].ind
ex, inplace=True)
comFlowdf.drop(comFlowdf[comFlowdf['Commercial_Flow'] == 'Closed'].index, inpl
ace=True)
comFlowdf["Commercial_Flow"] = comFlowdf["Commercial_Flow"].replace(["No Dela
y"] , 0).astype(int)
comFlowdf.loc[:,"Commercial_Flow"].mean()
```

Out[7]: 0.4349448263386397

In [8]:
```
travFlowdf = dTrafficFlow.copy(deep=True)
```

In [9]:
```
travFlowdf.drop(travFlowdf[travFlowdf['Travellers_Flow'] == 'Not Applicable'].
index, inplace=True)
travFlowdf.drop(travFlowdf[travFlowdf['Travellers_Flow'] == 'Closed'].index, i
nplace=True)
travFlowdf["Travellers_Flow"] = travFlowdf["Travellers_Flow"].replace(["No Del
ay"] , 0).astype(int)
travFlowdf.loc[:,"Travellers_Flow"].mean()
```

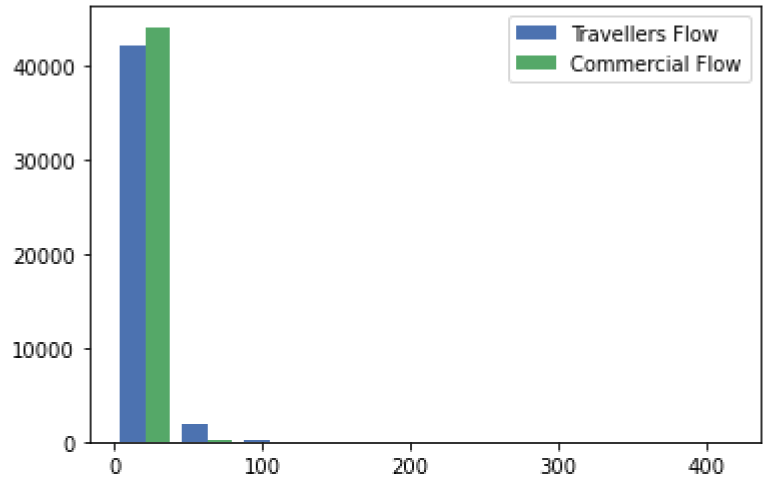
Out[9]: 5.280290353217856

In [10]:
```
# converting 'No Delay' into 0 and Closed and Not Applicable to the averages o
f the column
dTrafficFlow["Commercial_Flow"] = dTrafficFlow["Commercial_Flow"].replace(["No
t Applicable","Closed"] , 0.4349)
dTrafficFlow["Travellers_Flow"] = dTrafficFlow["Travellers_Flow"].replace(["No
t Applicable","Closed"] , 5.2803)
dTrafficFlow["Commercial_Flow"] = dTrafficFlow["Commercial_Flow"].replace(["No
Delay"] , 0).astype(float)
dTrafficFlow["Travellers_Flow"] = dTrafficFlow["Travellers_Flow"].replace(["No
Delay"] , 0).astype(float)
```

Calculating our delay time variable

In the next section, we decide how we would like to take the data from the Travellers Flow and Commerial Flow columns and use it to make one delay time variable.

In [11]:
```
# this is a histogram of the delays for travellers and commercial - we can see
that most delays are around 0 minutes
plt.style.use('seaborn-deep')
plt.hist([dTrafficFlow["Travellers_Flow"], dTrafficFlow["Commercial_Flow"]], l
abel=['Travellers Flow', 'Commercial Flow'])
plt.legend(loc='upper right')
plt.show()
```



Looking at the histogram above, which shows the travellers flow and commercial flow, the wait times for both are almost always 0. Although travellers do have delays more often than commercial users, they are similar enough where we decided it would be appropriate to average the two columns to get one general delay column.

In [12]:
```
# creating a new column for the total delay time, averaging the two delay colu
mns
dTrafficFlow["Total_delay"] = (dTrafficFlow["Commercial_Flow"] + dTrafficFlow[
"Travellers_Flow"])/2
```

Making the Month variable categorical

In the cell below, we brake the Month column data into a categorical variable, based on which quarter of the year the month falls into. We also tested out breaking the Month data into the seasons of the year, but after running the OLS model, that gave us a lower R^2 value.

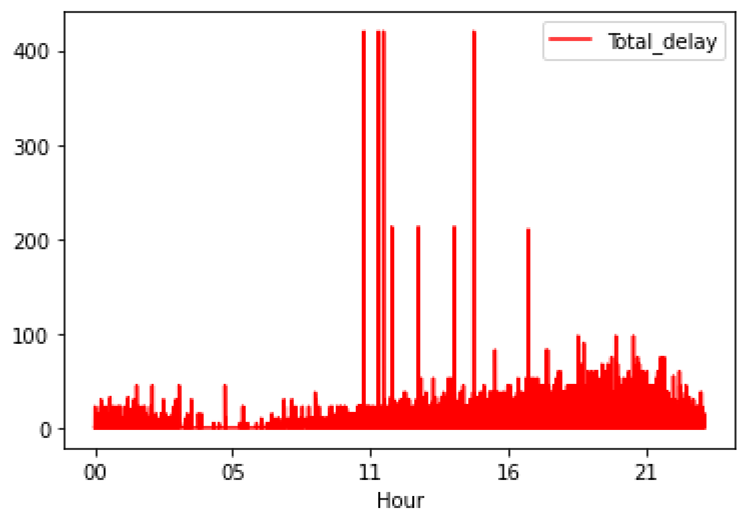
In [13]:
```
def quarte(x):
    x = int(x)
    if x <= 3:
        return "Q1"
    elif x <= 6:
        return "Q2"
    elif x <= 9:
        return "Q3"
    elif x <= 12 :
        return "Q4"
    else:
        return "NotApplicable"

dTrafficFlow['Quarters'] = dTrafficFlow['Month'].apply(quarte)
```

Making the Hour variable categorical and adding it as a new column

In the next section, we analyze the data in the Hour column to decide which method of breaking it up would have the best impact on our regression models.

In [14]:
```
# here we are sorting the data frame by month, so our following graph will dis
play correctly
sorteddf = dTrafficFlow.sort_values('Hour', ascending = True).reset_index(drop
=True)
```

In [15]:
```
sorteddf.plot(kind='line',x='Hour',y='Total_delay',color='red')
plt.show()
```



This graph shows us Total_delay vs Hour. We used this information to determine how we would break up the Hour column data into categorical variables. We made sure that times with similar delay values, as shown in the graph, were in one category. We also tested out breaking the Hour data up into different 6 hour windows, but rejected those, because breaking it up based on the graph provided us with the highest R^2 value in our OLS model.

In [16]:
```
# breaking Hour up into 4 different categories
def tod(x):
    x = int(x)
    if x < 4 or x >= 22:
        return "Night"
    elif x < 10:
        return "Morning"
    elif x < 16:
        return "Afternoon"
    else:
        return "Evening"

dTrafficFlow['TimeofDay'] = dTrafficFlow['Hour'].apply(tod)
```

Running the OLS Regression Model and Analyzing Results

Below, we run an OLS Regression model with Total_delay as the dependant variable. We also tested this model with Travellers_Flow as the dependant variable, and with Commercial_Flow as the dependant variable. Commercial_Flow gave us the lowest R^2 value (.004), while Travellers_Flow gave us the highest (.183). Although Travellers_Flow had a higher R^2 value than Total_delay (.135), we wanted to focus on Total_delay because it gave us information about both columns.

In [17]:
```
# training the ols on the continuous variable of wait time.
formula = "Total_delay ~ TimeofDay + Quarters + isHoliday + isWeekend"
model_ols = ols(formula = formula ,data= dTrafficFlow).fit()
model_ols.summary()
```

Out[17]:

OLS Regression Results

Dep. Variable:	Total_delay	**R-squared:**	0.135
Model:	OLS	**Adj. R-squared:**	0.134
Method:	Least Squares	**F-statistic:**	859.3
Date:	Sat, 21 Nov 2020	**Prob (F-statistic):**	0.00
Time:	09:25:58	**Log-Likelihood:**	-1.5404e+05
No. Observations:	44234	**AIC:**	3.081e+05
Df Residuals:	44225	**BIC:**	3.082e+05
Df Model:	8		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	0.5400	0.101	5.346	0.000	0.342	0.738
TimeofDay[T.Evening]	3.6858	0.104	35.439	0.000	3.482	3.890
TimeofDay[T.Morning]	-2.8751	0.108	-26.698	0.000	-3.086	-2.664
TimeofDay[T.Night]	-1.8680	0.105	-17.784	0.000	-2.074	-1.662
Quarters[T.Q2]	2.4632	0.107	23.126	0.000	2.254	2.672
Quarters[T.Q3]	4.0918	0.105	38.807	0.000	3.885	4.298
Quarters[T.Q4]	0.9426	0.105	8.996	0.000	0.737	1.148
isHoliday[T.True]	-2.8692	3.523	-0.814	0.415	-9.774	4.036
isWeekend[T.True]	2.1554	0.082	26.153	0.000	1.994	2.317

Omnibus:	92673.568	**Durbin-Watson:**	0.857
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1136871316.673
Skew:	17.631	**Prob(JB):**	0.00
Kurtosis:	787.594	**Cond. No.**	115.

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

Shown by the adjusted R-Squared value of the OLS model, we can determine that 13.4 percent of the variability in delay is due to the general time of day (TimeofDay), the quarter of the year (Quarters), if it's a holiday (isHoliday), and if it's a weekend (isWeekend). When we look at the coefficient values, we can see that Afternoon, Q1, isHoliday = False, and isWeekend = False do not appear. We suspect that this is because the regression model is encountering these values first and setting them as a baseline. This means that all the other coefficients are relative to an afternoon in quarter 1, that is not a holiday or a weekend. The coefficients also show us that Evening, Q2, Q3, Q4, and isWeekend = True have a positive effect on the delay, while Morning, Night, and isHoliday = True have a negative effect on the delay.

Making Total_delay categorical and adding it as a new column

In order to run a logistic regression model, we had to break up the dependant variable, Total_delay, into a categorical variable. Below we find the mean of Total_delay, and break the data into low and high groups based on that mean.

In [18]:
```
# we know the most of the values are 0 due to the histograms above
# this is the mean of Total_delay
dTrafficFlow.loc[:,"Total_delay"].mean()
```

Out[18]: 2.8576175860198045

In [19]:
```
# making delay time a categorical variable, with low = 0 and high = 1 options, based on the average
dTrafficFlow['Delay'] = dTrafficFlow['Total_delay'].apply(lambda x: 1 if x >= 2.8576175860198045 else 0)
```

The "low delay" group ended up being values that were less than 2.858, and the "high delay" group were values that were greater than or equal to 2.858. Although this was the way we felt was most appropriate for the assignment, it may not be logical in the real world. Many may not consider a delay of 3 minutes as a delay at all. Ideally, we would have more groups, such as no delay, low delay, medium delay, high delay, and extremely high delay.

Running the Logistic Regression Model and Analyzing Results

Below, we run a logistic regression model with Delay as the dependant variable, and TimeofDay, Quarters, and isWeekend as the independant variables. We were unable to include isHoliday, because too high of a percentage of the values were False, and the model rejected it.

In [20]:
```
logitFormula = "Delay ~ TimeofDay + Quarters + isWeekend"
logit_model = logit(formula = logitFormula, data = dTrafficFlow).fit()
logit_model.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.369498
         Iterations 8
```

Out[20]:

Logit Regression Results

Dep. Variable:	Delay	**No. Observations:**	44234
Model:	Logit	**Df Residuals:**	44226
Method:	MLE	**Df Model:**	7
Date:	Sat, 21 Nov 2020	**Pseudo R-squ.:**	0.2634
Time:	09:25:59	**Log-Likelihood:**	-16344.
converged:	True	**LL-Null:**	-22190.
Covariance Type:	nonrobust	**LLR p-value:**	0.000

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
Intercept	-2.6300	0.043	-60.823	0.000	-2.715	-2.545
TimeofDay[T.Evening]	0.9957	0.031	31.822	0.000	0.934	1.057
TimeofDay[T.Morning]	-3.2762	0.087	-37.521	0.000	-3.447	-3.105
TimeofDay[T.Night]	-1.3466	0.042	-32.051	0.000	-1.429	-1.264
Quarters[T.Q2]	1.3199	0.044	29.696	0.000	1.233	1.407
Quarters[T.Q3]	2.1422	0.044	49.202	0.000	2.057	2.228
Quarters[T.Q4]	0.8045	0.045	17.820	0.000	0.716	0.893
isWeekend[T.True]	0.8930	0.029	30.421	0.000	0.835	0.951

Shown by the Pseudo R-Squared value of the Logistic model, we can determine that 26.34 percent of the variability in delay is due to the general time of day (TimeofDay), the quarter of the year (Quarters), and if it's a weekend (isWeekend). When we look at the coefficient values, we can see that Afternoon, Q1, and isWeekend = False do not appear, for the same reason as in the OLS model. The coefficients show us that Evening, Q2, Q3, Q4, and isWeekend = True have a positive effect on the delay, while Morning and Night have a negative effect on the delay.

OLS vs. Logistic Analysis

The OLS and Logistic regression models that we ran analyzed the same variables (with the exception of isHoliday), but the dependent variable in the logistic model was split into a categorical variable. The models returned different R-Squared values (.134 vs. .2634), and the logistic R-Squared value was about double the R-Squared value of the OLS model. This means that the Logistic model is a better predictor of the variability in delay times. This may be because it is easier for the model to predict a general category for delay time rather than an exact value. However, both R-Squred values were very low, so the models are not great predictors. Splitting up the categorical variables TimeofDay and Quarters in different ways may lead to better results. When looking at the coefficient values, the independant variables which were positive in the OLS model were also positive in the logistic model, and the variables that were negative in the OLS model were also negative in the logistic model. This makes sense, if we look at TimeofDay = Morning, for example, because traffic in the very early morning is likely to be low, so each model represents that by giving it a negative coefficient.

Suggested Improvements to the Models

Other features that could improve the model are data about traffic incidents, traffic congestion, weather, and the amount of staffing at the border gate. These factors are related as bad weather and congestion can lead to an increased number of traffic accidents. The delay time and the corresponding number of vehicles/visitors being delayed could potentially be used to understand the reason for delay. A larger dataset with high level of accuracy and detail would have helped to train the model first and to test the data for finer prediction.

Conclusions

In general, this assignment was mostly successsful, as we were able to demonstrate our Pandas, Matplotlib, and Jupyter Notebook skills by cleaning and manipulating data, producting visualizations, and creating a clean report. Unfortunetely, our R-Squared values in both models were very low. However, unlike lab tests, the environmental source of data can have highly skewed data which would result in having lower R-Squared value and lesser probability of accurate prediction. This model provides a fair R-Squared value to help in the analysis. Overall, we are satisfied with the results from this assignment.