Wait time modeling for traffic at US-Canada Border

Data Management & Regression Basics

Abstract

In this project, I examine the effect of date/time and weather on the wait time that a traveler faces while crossing the Peace Bridge between Canada and the United States. Exploratory Data Analysis (EDA) is used for analysis to compute the regression results. The Logistic Regression is used to predict/estimate when travel will be delayed by 'x minutes' due to specific hours or weather type. The model is tested via dataset of wait time (y-variable) which is the dependent variable and time hours (x-variable) & weather conditions (x-variable) which are the independent variables for the years 2010-2016 for training the model and 2017 for testing the model. The findings illustrate how, for different hours & weather conditions plays roles in predicting wait time for a given data. The model's predictability is 55% which can further help travelers analyze/plan their trip in advance with respect to further improvements in the model.

Introduction

The Peace Bridge is an international bridge between Canada and the United States at the east end of Lake Erie at the source of the Niagara River, about 20 km upriver of Niagara Falls. It connects Buffalo, New York, in the United States to Fort Erie, Ontario, in Canada. It is operated and maintained by the binational Buffalo and Fort Erie Public Bridge Authority. The Canadian border sees significant traffic flow from the US. The Government of Canada collects detailed information about wait times at its various crossings. I analyze one such crossing i.e. the Peace Bridge crossing. This analysis will help the travelers to plan their trip during the least congestion hours along with weather conditions and also get an idea of the average wait time during the trip.

Data, Methodology & EDA

There are two datasets that have been used for the project. The first dataset is taken from the Government of Canada website relating to the wait time vehicles at 26 crossing at US-Canada Border for the period of 2010-2017. It has the information regarding CBSA office, location, updated (contains date and time), commercial flow and traveler flow. The last 2 columns tell us if there was no delay, if it was closed or if there was delay. I analyze the Peace Bridge crossing for the same period. The second dataset is about the hourly weather conditions from the US government website for the period of 2010-2017 for the Erie County, Buffalo, New York. It has various information like date, hourly present weather type, hourly precipitation, hourly sky conditions, etc. The Peace Bridge comes under The Erie County on the US side is relevant to the weather condition at the bridge. Moreover, the US side is important as the analysis is based on traffic flow from the US to Canada.

From the wait time dataset, I first merge all the data files for the years 2010-2017 and filter it for the Peace Bridge under CBSA office. Dropping the Commercial Flow column which shows commercial vehicle wait time in the dataset. Removing rows that has 'Missed Entry' in the Travelers flow which shows traveler wait time in the dataset. Changing the data type of the date column to extract information. For the wait time under Travelers flow, changing 'No delay' to 0. From the weather data, I extract the 2 relevant columns that are date and HourlyPresentWeatherType which contains various weather codes that are explained in their dataset documentation. Changing the data type of the date column to extract information. Both the datasets had hourly data with some datapoints relating to minutes as well, which were not consistent. Assuming that weather will doesn't changes within minutes, I cluster for both the dataset's time under 'date' column to 'hours' i.e. 2:04pm will be same as 2:54pm and be shown as 2 with the assumption that weather does not change much in an hour. Merging both the datasets to perform analysis. The table below (Fig. 1) shows the descriptive statistics of the merged sample dataset (n=40,841). It can be seen through min & max that time is from 0-23 hours, year from 2010-2017, Logistic variables as 0 & 1 and wait time rages from 0 (No Delay) to 75 (delay time). The merged dataset is then divided into 2 parts as one for the period 2010-2016 and the second for the year 2017. The EDA, later in report is done for the period of 2010-2016.

	Year	TimeHour	Wait_Time	TravellerLogit
count	40841.000000	40841.000000	40841.000000	40841.000000
mean	2014.604099	11.169389	1.332289	0.031904
std	1.559211	7.178114	3.508725	0.175747
min	2010.000000	0.000000	0.000000	0.000000
25%	2015.000000	5.000000	0.000000	0.000000
50%	2015.000000	11.000000	0.000000	0.000000
75%	2015.000000	18.000000	1.000000	0.000000
max	2017.000000	23.000000	75.000000	1.000000

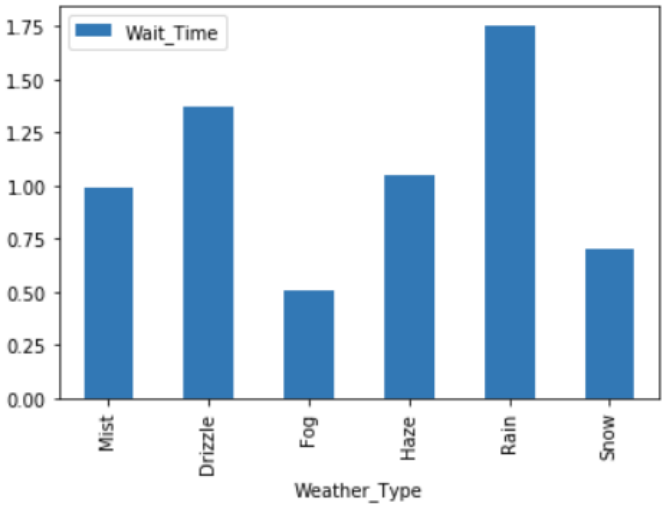
Fig. 1 Descriptive Statistics of Sample Dataset

For weather type column, as per the documentation provided, there can be 3 categories of codes (recorded by different organizations) for each cell and each category itself contains various codes with first one being most accurate to weather condition recorded. Therefore, extracting the most accurate weather code and checking for their occurrences in the dataset (Fig. 2).

SN:03	19025	SN:03	Snow
RA:02	10476	RA:02	Rain
BR:1	6058	BR:1	Mist
DZ:01	1283	DZ:01	Drizzle
HZ:7	774	HZ:7	Haze
FG:2	274	FG:2	Fog
FU:3	23	FU:3	Smoke
PL:06	18	PL:06	Ice Pellets
GS:08	5	GS:08	Small Hail an/or Snow Pellets
SQ:2	2	SQ:2	Squalls

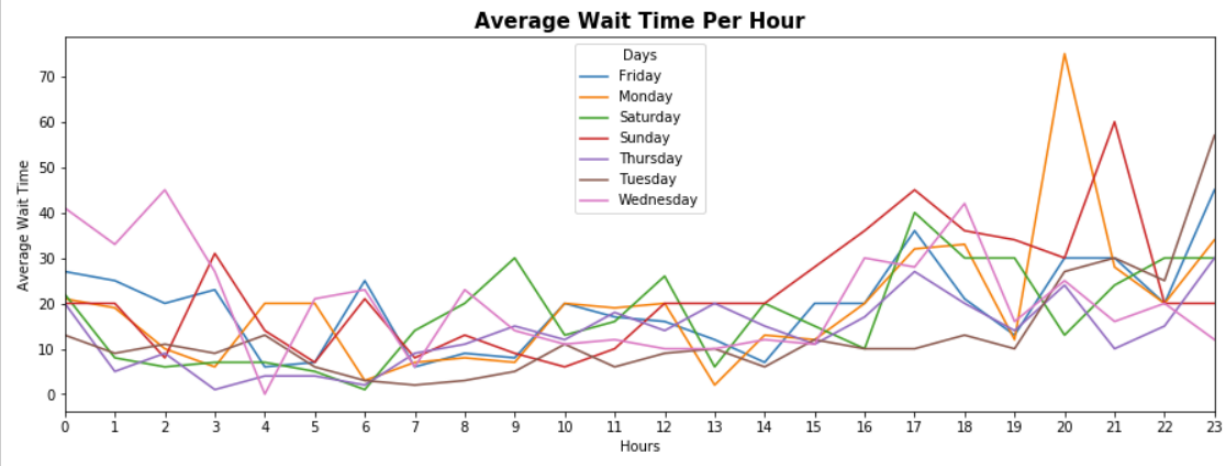
Fig. 2 Weather Type described

The above information (Fig. 2) shows various weather codes which are defined on the right side. It seems truly relevant to the wait time as all these factors can hamper the speed of vehicles and cause traffic. These factors can make road slippery, reduce the visibility, so on. But on the other hand these factors can result in no delay as well, since weather forecasting has been so advanced that travelers see the forecast and let's say if it is forecasted to snow then they might ignore the travel, resulting in less vehicles on bridge and therefore less/no wait time. There can be both the cases. It can be also seen that the last 4 weather types have very insufficient data and hence they are removed for better analysis.



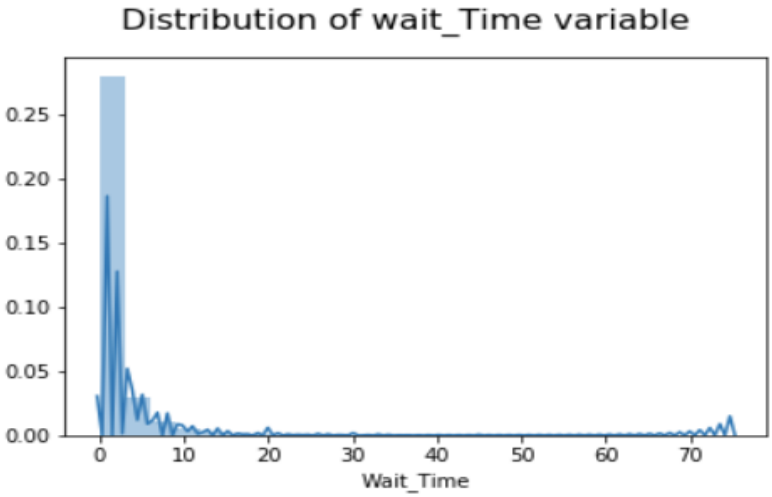
(Fig. 3) Average wait time per weather type

Above (Fig. 3) can be seen the average wait for the rest of the weather types for 2010-2016. It can be seen that 'Rain causes most of the wait times as compared to other weather conditions. Snow is relatively low as fresh snowfall does not cause much slowdown and moreover, Canada and US both facing heavy snowfalls during season, travelling can be avoided by people during those times.



(Fig. 4) Average wait time per hour for the days of a week

Further moving to wait time's relation to hours of the day, the above chart (Fig. 4) depicts the average wait time per hour for all 7 days of the week at all 24 hours of the day for 2010-2016 and we can clearly see that maximum wait time is during Friday & Sunday late evening and the least wait time in the weekdays. Moreover from 12 am to 8 am there is not much difference between weekday and weekend. These patterns can help people to plan their travel day and time effectively and efficiently. Based on above chart, hours are then grouped in 4 categories as morning (4am-9am), afternoon (10am-15pm), evening (16pm-21pm) and night (22pm-3am) for our model.



(Fig. 5) Distribution of Wait_Time Variable

To estimate the Logistic regression model, it is important to divide the dependent varable meaning the y-variable i.e. wait time into 0 & 1. To do so, I looked at the distribution of the wait time variable (Fig. 5), which show that is right skewed distribution, meaning most of the numbers in wait time variable are 0,1,2..below 5 but extends till 75. Therefore, the mean is 1.33, which could have been used to split the variable into 0 & 1 but any traveler would not be interested in wait times for 1 minute. Therefore, on assessing various real reviews on google, travelers suggesting 5-10 minute as a usual wait time on Peace Bridge, I take 10 minutes as the point of distributing the numbers to 0 & 1 i.e. wait time < 10 will be 0, else 1.

Regression Modeling & Results

I used Logistic regression analysis to model wait time (Y variable) as a function of time of day and weather type. Since the time hour and weather variables are categorical variables, I have converted them to dummy variables to perform regression. Two things are to be done in regression analysis, first is to estimate/train the model for the time period of 2010-2016 and then testing this model for the data of 2017.

Before training the model for 2010-2016, I checked for the distribution of values of Logistic Wait Time variable which is 0 (No delay) and 1 (delay). It can be seen (Fig. 6) that distribution is not balanced/scaled. Out of total, 39538 (96.8%) entries are 0 i.e. No Delay and rest 1,303 (3.2%) entries are 1 i.e. Delay. The reason being mean of the wait time distribution is 1.33 and most of the entries for wait time were below 5 minutes of wait time. And on the other hand, our model takes 10 minutes as a threshold. It is therefore necessary to rebalance the distribution for better analysis, else the logistic model will interpret everything as 0.

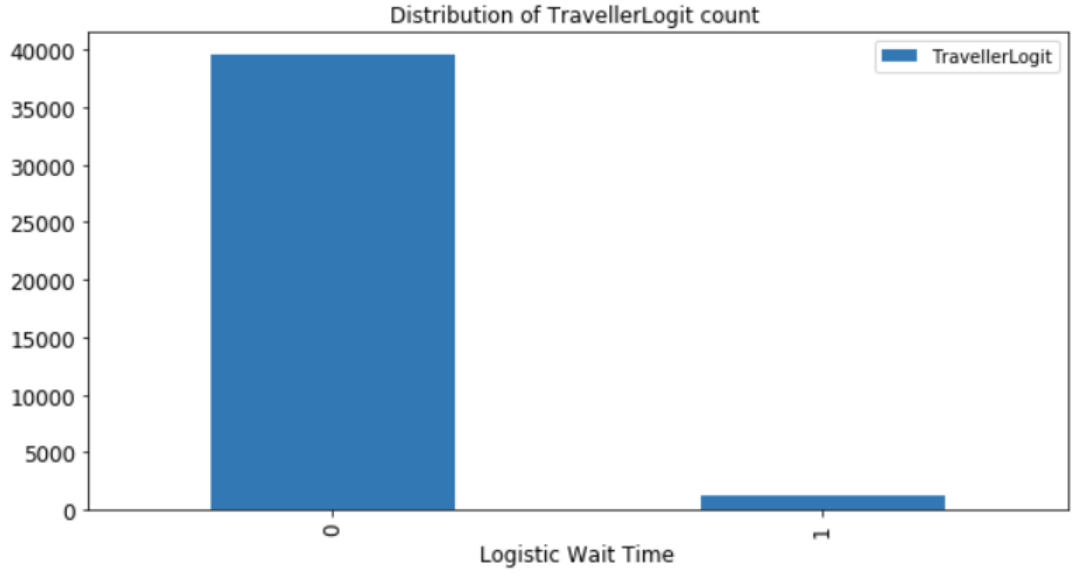


Fig. 6 Distribution of 0s & 1s for Logistic Wait time (Y-variable)

To rebalance the above discussed variable, I used imblearn library and use Synthetic Minority Oversampling Technique (SMOTE) function from it. SMOTE involves duplicating examples in the minority class i.e. 1 in our logistic variable, although these examples don't add

any new information to the model. Instead, new examples can be synthesized from the existing examples. This is a type of data augmentation for the minority class. After using SMOTE, I have the balanced sample and below (Fig. 7) is the description of the same.

```
Length of oversampled data is  74022
Number of no subscription in oversampled data 37011
Number of subscription 37011
Proportion of no subscription data in oversampled data is  0.5
Proportion of subscription data in oversampled data is  0.5
```

(Fig. 7) Description of Logistic Wait Time (Y- variable) after using SMOTE

After obtaining the balanced sample, to train my model, I use sklearn library of python which fits the model and estimates the best representative function for the data points (could be a line, polynomial or discrete borders around). Since, sklearn does not give a summary of regression for r^2 or p-values, I just find intercept as [-4.53324031] and coefficients as [1.84670296 2.87057454 0.39445646 1.93627102 2.02961899 2.3562689 0.20712074 1.56192448 3.04619541 2.1696095] for ['Hour_status_afternoon', 'Hour_status_evening', 'Hour_status_morning', 'Hour_status_night', 'Weather_Type_BR:1', 'Weather_Type_DZ:01', 'Weather_Type_FG:2', 'Weather_Type_HZ:7', 'Weather_Type_RA:02', 'Weather_Type_SN:03'] respectively. Coefficient tells us how much the dependent variable (y variable- Wait Time) is expected to increase when that independent variable (x variables) increases by one, holding all the other independent variables constant. With that representation and model, I can calculate new data points or say predict the y variable for given data points. I have used both the time hours (all the 4 categories of time) and the weather types as x-variables and Logistic Wait times (0 & 1) as y-variables to train the model. I then use the model to predict the y-variable of the data for 2017 using the same x-variables of 2017 dataset and compare it with actual y-variable points for the dataset. I get the accuracy of logistic regression on test set of 2017 as 0.55 or 55%. It can be looked at the summary (Fig. 8) below.

	precision	recall	f1-score	support
0	0.91	0.52	0.66	2477
1	0.20	0.70	0.31	424
accuracy			0.55	2901
macro avg	0.55	0.61	0.49	2901
weighted avg	0.81	0.55	0.61	2901

(Fig. 8) Summary of Logistic Regression

Since the accuracy is not 100% i.e. the model does not predict fully we look at the confusion matrix (Fig. 9) for the different cases that would have happened and it can be clearly

seen below that 1287 (44.3%) of the total data points were true positive and 297 (10.2%) of the data points were true negatives i.e were estimated correctly whereas rest of the others were false.

1287 (True Positives)	**1190** (False Negatives)
127 (False Positives)	**297** (True Negatives)

(Fig. 9) Confusion matrix

Conclusion

The findings illustrate accuracy of prediction through the logistic regression model as 55%. It also illustrates that how hours of a day like evening impacts wait time the most. Moreover, how the weather conditions impact the wait for the traveler. The model is based on 2 major factors, but delays can also be impacted by other factors too. Some of the other factors that would affect the wait time and the congestion on the roads are:

1. Commercial Vehicle: There could have been another analysis for comparing commercial and non-commercial traveler wait time. Commercial vehicle are usually bigger than cars like trucks and take more space on road. There is no separate lane on highway for those big vehicles. Therefore. analysis could be based on comparing the wait time in general and then comparing the wait time for each when the other one is closed to be little specific.

2. Accidents: Accidents on road can lead to traffic for a few miles until the damaged vehicle is taken off the roads. This can be correlated to weather conditions as well. Bad weather can lead to accidents.

3. Special events, university 'Move-in-day', concerts, etc.: There can be traffic on special days such as major events of games, big concerts and also university opening when many people go out to experience it. If there events come on weekends, it can cause even more traffic than the usual weekend.

4. Number of counters open for the immigration process: To travel from one country to another, cars have to stop at border for immigration. If there are a smaller number of counters open, it would automatically increase the waiting hours.

5. Seasonal Shopping – Holiday shopping around major mall areas was indicated as another source of non-commuting congestion, particularly on weekends between Thanksgiving and Christmas

Lastly to conclude, based on the whole analysis, it is definite that there is traffic congestion at US-Canada border at specific weather conditions and peak hours but there can be other factors, if taken into consideration, may give better results and analysis. Therefore, travelers should plan their travel accordingly to avoid rush and long waiting hours.

Resources

Weather Dataset https://www.ncdc.noaa.gov/cdo-web/datasets

Wait time Dataset https://open.canada.ca/data/en/dataset/000fe5aa-1d77-42d1-bfe7-458c51dacfef

Others:

https://en.wikipedia.org/wiki/Peace_Bridge

https://ops.fhwa.dot.gov/congestion_report/chapter2.htm

https://ops.fhwa.dot.gov/congestion_report/chapter2.htm

https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/

Coding Appendix

```
# importing relevant libraries
import pandas as pd
import numpy as np
import datetime
from datetime import date
import warnings
import os
from sklearn.preprocessing import OneHotEncoder
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import seaborn as sns
warnings.filterwarnings('ignore')
```

```
# importing wait time data
colNames = ('CBSA Office', 'Location', 'Updated', 'Commercial Flow', 'Travellers Flow')
wait_time_df = pd.DataFrame(columns = colNames)

for subdir, dirs, files in os.walk("../data/wait time/"):
    for file in files:
        df = pd.read_csv(f'{subdir}{file}')
        if df.columns[4] != 'Travellers Flow':
            df = df.rename(columns = {df.columns[4]:'Travellers Flow'})
        wait_time_df = wait_time_df.merge(df, how='outer')
# filteration of data to get data of Peace Bridge and Travellers flow only
peace_bridge_df = (wait_time_df.loc[wait_time_df['CBSA Office'] == 'Peace Bridge'])
peace_bridge_df = peace_bridge_df.reset_index(drop = True)
peace_bridge_df = peace_bridge_df.drop(columns ='Commercial Flow', axis = 1)
# dropping duplicates that incurred while merging data files(during import)
peace_bridge_df = peace_bridge_df.drop_duplicates()
# dropping rows that contain wait time for travellers flow as Missed entry
peace_bridge_df = peace_bridge_df[-peace_bridge_df['Travellers Flow'].isin(['Missed entry'])]
# importing weather data
data2 = pd.read_csv("../data/weather_data.csv")
# filtering data to get date and weather type only
weather_data = data2[['DATE', 'HourlyPresentWeatherType']]
# dropping null values
weather_data = weather_data.dropna(axis = 0, how = 'any')
# changing the type of column as datetime
weather_data['DATE'] = weather_data['DATE'].astype('datetime64[h]')
peace_bridge_df['Updated'] = peace_bridge_df['Updated'].astype('datetime64[h]')
# dropping duplicates for merging the two data sets
```

```
weather_data = weather_data.drop_duplicates(subset='DATE', keep="first")

weather_data = weather_data.rename(columns={'DATE': 'Updated'})

peace_bridge_and_weather_merged = peace_bridge_df.merge(weather_data, on='Updated',
how='inner')

# importing weather codes data

weather_codes = pd.read_csv("../data/weather_codes.csv")

# creating a dictionary of weather codes

weather_dict = dict(zip(weather_codes['weather_code'], weather_codes['weather_type']))

# function to pick the weather code from the weather type column

def parse_weather(weather_string):
    tok = weather_string.split('|')
    tok = [i.strip() for i in tok]
    for section in tok:
        if section != '':
            tok_2 = section.split(' ')
            for symbol in tok_2:
                symbol  = symbol[1:] if symbol.startswith(('-', '+')) else symbol
                if symbol in weather_dict:
                    return symbol
    return np.nan

# calling the above function to get the code for weather_type

peace_bridge_and_weather_merged['HourlyPresentWeatherType'] =
peace_bridge_and_weather_merged['HourlyPresentWeatherType'].apply(parse_weather)

peace_bridge_and_weather_merged = peace_bridge_and_weather_merged.dropna(axis = 0, how
= 'any')

# renaming weather column

peace_bridge_and_weather_merged =
peace_bridge_and_weather_merged.rename(columns={'HourlyPresentWeatherType':
'Weather_Type'})

# Extracting Date
```

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

# Days Names of the days

peace_bridge_and_weather_merged['Date'] =
peace_bridge_and_weather_merged['Date'].astype('datetime64[ns]')

peace_bridge_and_weather_merged['Days'] =
peace_bridge_and_weather_merged['Date'].dt.weekday_name

# Day Status of the days in week

peace_bridge_and_weather_merged['DayStatus'] =
peace_bridge_and_weather_merged['Date'].dt.weekday

days =
{0:'Weekday',1:'Weekday',2:'Weekday',3:'Weekday',4:'Weekday',5:'Weekend',6:'Weekend'}

peace_bridge_and_weather_merged['DayStatus'] =
peace_bridge_and_weather_merged['DayStatus'].apply(lambda x:days[x])

# Extracting year

peace_bridge_and_weather_merged['Year'] =
pd.DatetimeIndex(peace_bridge_and_weather_merged['Updated']).year

# Extracting hour

peace_bridge_and_weather_merged['TimeHour'] =
pd.to_datetime(peace_bridge_and_weather_merged['Updated']).dt.hour

# Changing No Delay entries in WaitTime Column

peace_bridge_and_weather_merged['Travellers Flow'] =
np.where(peace_bridge_and_weather_merged['Travellers Flow'] == 'No delay', 'No Delay',
peace_bridge_and_weather_merged['Travellers Flow'])

peace_bridge_and_weather_merged['Wait_Time'] =
np.where(peace_bridge_and_weather_merged['Travellers Flow'] == 'No Delay', 0,
peace_bridge_and_weather_merged['Travellers Flow'])

peace_bridge_and_weather_merged['Wait_Time'] =
peace_bridge_and_weather_merged['Wait_Time'].astype('int32')

# descriptive statistics of dataset

peace_bridge_and_weather_merged.describe()

# EDA- looking at the distribution of Wait time

print(peace_bridge_and_weather_merged['Wait_Time'].describe())
```

```
plt.figure()

sns.distplot(peace_bridge_and_weather_merged['Wait_Time'], bins=25, hist_kws={'alpha': 0.4})

plt.show()

# WaitTime column for Logistic Regression

peace_bridge_and_weather_merged['TravellerLogit'] =
np.where(peace_bridge_and_weather_merged['Wait_Time']<10, 0, 1)

# TravellerLogit count chart

import matplotlib.pyplot as plt

TravellerLogit_count = peace_bridge_and_weather_merged['TravellerLogit'].value_counts()

ax = TravellerLogit_count.plot(kind='bar', title ="TravellerLogit count chart", figsize=(10, 5),
legend=True, fontsize=12)

ax.set_xlabel("Logistic Wait Time", fontsize=12)

plt.show()

# Splitting the data into 2 parts as 2010-2016 (training period) & 2017 (testing period)

is_2017 = peace_bridge_and_weather_merged['Year'] == 2017

final_df_2017 = peace_bridge_and_weather_merged[is_2017]

final_df_2017.head()

is_2010_to_2016 =  [not i for i in is_2017]

final_df_2010_to_2016 = peace_bridge_and_weather_merged[is_2010_to_2016]

# 2010-2016 data

# plotting figure showing days and hours in a day for wait time data

fig, ax = plt.subplots(figsize=(10,5))

Traveller_avg_waitTime = final_df_2010_to_2016.groupby(["TimeHour",
"Days"])['Wait_Time'].max().unstack().plot(ax=ax)

plt.ylabel('Average Wait Time', fontsize = 10)

plt.xlabel('Hours', fontsize = 10)

plt.title('Average Wait Time Per Hour',x=0.5, y=1,fontsize =15,fontweight="bold")

hour_ticks = np.arange(0, 24, 1)

ax.set_xticks(hour_ticks)
```

```
plt.show()
# function to pick the weather code from the weather type column using the graph above
def get_hour_status(time):
    if time >= 0 and time <= 3:
        return 'night'
    elif time >= 4 and time <= 9:
        return 'morning'
    elif time >= 10 and time <= 15:
        return 'afternoon'
    elif time >= 16 and time <= 21:
        return 'evening'
    elif time >= 22 and time <= 23:
        return 'night'
    else:
        raise Exception(f'Invalid time = {time}')
# calling the function above
final_df_2010_to_2016['Hour_status'] =
final_df_2010_to_2016['TimeHour'].apply(get_hour_status)
final_df_2017['Hour_status'] = final_df_2017['TimeHour'].apply(get_hour_status)
# checking the counts of entries of weather types
final_df_2010_to_2016['Weather_Type'].value_counts()
# Removing weather types with insufficient data
final_df_2010_to_2016 =
final_df_2010_to_2016[~final_df_2010_to_2016['Weather_Type'].isin(['SQ:2', 'GS:08', 'PL:06',
'FU:3'])]
final_df_2017 = final_df_2017[~final_df_2017['Weather_Type'].isin(['SQ:2', 'GS:08', 'PL:06',
'FU:3'])]
# comparing the average wait time during different weather types
print(final_df_2010_to_2016[['Weather_Type',
'Wait_Time']].groupby(['Weather_Type']).mean())
```

```
weather_average_wait_time = final_df_2010_to_2016[['Weather_Type',
'Wait_Time']].groupby(['Weather_Type']).mean().plot(kind='bar')

weather_average_wait_time.set_xticklabels(('Mist', 'Drizzle', 'Fog', 'Haze', 'Rain', 'Snow'))

plt.show()

# Regression analysis

# Applying OneHotEncoding using get_dummies from pandas

dummies_df_2010_to_2016 = pd.get_dummies(final_df_2010_to_2016, columns=['Hour_status',
'Weather_Type'])

dummies_df_2017 = pd.get_dummies(final_df_2017, columns=['Hour_status', 'Weather_Type'])

# choosing the columns for x variables for training the model

target_columns = []

for col in dummies_df_2010_to_2016.columns:

    if col.startswith('Hour_status_') or col.startswith('Weather_Type_'):

        target_columns.append(col)

# creating x & y variables for training and testing of models for both the datasets

X_train_2010_to_2016 = dummies_df_2010_to_2016.loc[:, target_columns]

y_train_2010_to_2016 = dummies_df_2010_to_2016.loc[:,['TravellerLogit']]

X_test_2017 = dummies_df_2017.loc[:, target_columns]

y_test_2017 = dummies_df_2017.loc[:,['TravellerLogit']]

# balancing the logistic variable

os = SMOTE(random_state=0)

# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

columns = X_train_2010_to_2016.columns

os_data_X, os_data_y=os.fit_sample(X_train_2010_to_2016, y_train_2010_to_2016)

os_data_X = pd.DataFrame(data=os_data_X,columns=columns)

os_data_y= pd.DataFrame(data=os_data_y, columns=['TravellerLogit'])

# we can Check the numbers of our data

print("Length of oversampled data is ",len(os_data_X))
```

```
print("Number of no subscription in oversampled
data",len(os_data_y[os_data_y['TravellerLogit']==0]))

print("Number of subscription",len(os_data_y[os_data_y['TravellerLogit']==1]))

print("Proportion of no subscription data in oversampled data is
",len(os_data_y[os_data_y['TravellerLogit']==0])/len(os_data_X))

print("Proportion of subscription data in oversampled data is
",len(os_data_y[os_data_y['TravellerLogit']==1])/len(os_data_X))

# training the logistic model

logistic_reg = LogisticRegression()

logistic_reg.fit(X_train_2010_to_2016, y_train_2010_to_2016)

logistic_reg.coef_

logistic_reg.intercept_

# predicting/testing the data for 2017 based on the model & getting the accuracy score

y_pred_2017 = logistic_reg.predict(X_test_2017)

print('Accuracy of logistic regression on test set of 2017:
{:.2f}'.format(logreg.score(X_test_2017, y_test_2017)))

# creating a confusion matrix to check for false/true positives/negatives

confusion_matrix = confusion_matrix(y_test_2017, y_pred_2017)

print(confusion_matrix)

# extracting the summary report for the model

print(classification_report(y_test_2017, y_pred_2017))
```