Catching Flights: Python report

A study of the efficiency of commercial flights within the United States with Python programming.

By Nur Aisha binte S Burhan [redacted]

---

1 Introduction

In this report, we dive into the punctuality of commercial flights in the United States with the use of Python Programming. Python is a programming language with great versatility; it is often used for web development, automation, data science, and machine learning. Due to its simple syntax, it is well-suited for beginners (Worsley, 2022). The next few pages will contain Python codes we have used in the assessment of this study.

In [1]:
```
import warnings
warnings.filterwarnings('ignore')
```

2 Data

The 2009 ASA Statistical Computing and Graphics Data Expo entails flight arrival and departure details for all commercial flights within the US, from October 1987 to April 2008 (Harvard Dataverse, 2008). As this is a large dataset, the data we will be using in this particular study consists of years 1989-1990, 1994-1996, 2000-2002, and 2006-2007. We have chosen these particular subset of years to ensure a reliable and accurate analysis as we compare the data over time.

Simalarly to R Programming, Python also has packages which are designed to add specialized functionality. They are composed of modules, which are essentially codes that are made to be reusable (Udacity, 2021). To get access to the packages that we want, we use the `import` function.

Before importing data, you will need to set your directory. The directory sets the default location of any files you would like to import. You can find your current working directory with the `os.getcwd()` function. To change directories, use `os.chdir("insert designated path")`. These functions are available in the `os` package.

In [2]:
```
import os
os.chdir("/Users/nuraisha/Desktop/ST2195 CW/DATASET")
```

The files are saved in .csv format. We import the files in by using the `pd.read_csv()` function, available in the pandas package, and assign them to their respective years. Pandas is a Python package widely utilized for tasks in data science, machine learning,

and data analysis. `encoding` is the process where the sequence of code points is converted into a set of bytes, for efficient storage of these strings. 'latin-1' is a character encoding standard.

In [3]:
```
import pandas as pd
Y1989 = pd.read_csv("1989.csv", encoding='latin-1')
Y1990 = pd.read_csv("1990.csv", encoding='latin-1')
Y1994 = pd.read_csv("1994.csv", encoding='latin-1')
Y1995 = pd.read_csv("1995.csv", encoding='latin-1')
Y1996 = pd.read_csv("1996.csv", encoding='latin-1')
Y2000 = pd.read_csv("2000.csv", encoding='latin-1')
Y2001 = pd.read_csv("2001.csv", encoding='latin-1')
Y2002 = pd.read_csv("2002.csv", encoding='latin-1')
Y2006 = pd.read_csv("2006.csv", encoding='latin-1')
Y2007 = pd.read_csv("2007.csv", encoding='latin-1')
```

We then bind the separate files into one, using the `pd.concat()` function.

In [4]:
```
Dataset = pd.concat([Y1989,Y1990,Y1994,Y1995,Y1996,Y2000,Y2001,Y2002,Y2006,Y
```

After loading the data, our first step should always be to inspect them and understand the different parameters and variables. `DataFrame.info()` shows a concise summary of the dataframe. The information contains the number of columns, column labels, column data types, memory usage, range index, and the number of cells in each column for non-null values. `DataFrame.head(n= )` shows the first n rows where n=5 by default.

3 Findings

3.1 Period with minimal delays

We define optimal by the period whereby the number of early and punctual flights are at its highest. In this section, we aim to find the top 3 most common optimal date and time of flight.

We start off by filtering the data to non-delayed flights by including data where every type of delay has a value of less than, or equal to zero. Square brackets `[]` are used to access rows from a DataFrame. The & sign means 'and', which gives the intercept of observations from each condition.

In [5]:
```
#Filtering the data
nondel_flights = Dataset[(Dataset['ArrDelay'] <= 0) & (Dataset['DepDelay'] <
    & (Dataset['WeatherDelay'] <= 0) & (Dataset['NASDelay'] <= 0) & (Dataset
    & (Dataset['LateAircraftDelay'] <= 0)]
```

Next, we select the relevant columns using double square brackets `[[]]`, where the interior brackets are for list, and the outside brackets are for the indexing operator.

In [6]:
```
#Selecting relevant columns
nondel_schedule = nondel_flights[['Month', 'DayOfWeek', 'CRSDepTime']]
```

We then tabulate the number of observations using the `value_counts()` function. We also reset and name the index using the `rest_index()` function. Afterwards, we find the top 3 values by using the `nlargest(n,col)` function.

Dot notation is a way to select columns in a dataframe. It indicates that you're accessing data or behaviors for a particular object type.

In [7]:
```
#Finding frequency of each schedule
schedfreq = nondel_schedule.value_counts().reset_index(name='Frequency')

#Finding top 3 values
schedfreq.nlargest(3, 'Frequency')
```

Out[7]:

	Month	DayOfWeek	CRSDepTime	Frequency
0	11	4	600	2199
1	10	2	600	2156
2	5	3	600	2134

Therefore, the most common schedule that has the highest number of punctual flights is a 6am flight on a Thursday in the month of November, followed by a 6am flight on a Tuesday in October and a 6am flight on a Wednesday in May.

3.2 Efficiency of older planes

In this section, we analyse the efficiency of old planes by comparing the average delay duration over time.

Before we begin, it is crucial to note that we do not have sufficient data to fully analyse the efficiency of older planes as there is no specific column in the available data to determine whether the aircraft is older or newer. Therefore, we proceed in the assumption that the aircrafts are renewed each year.

As defined by BTS, a flight is considered delayed when it arrived 15 or more minutes than the schedule(Bureau of Transportation Statistics, -). Thus we first filter to our desired data where arrival and departure delay has a value of more than 15 minutes. The | sign means 'or'. Instead of 'and', we use 'or' with the intention to collate data that gives the union of observations from each condition.

In [8]:
```
#Filtering the data
delayed_flights = Dataset[(Dataset['ArrDelay'] > 15) | (Dataset['DepDelay']
```

Next, we find the total duration of delay by summing the columns together.

In [9]:
```
#Finding total duration of delay
delayed_flights['Totaldelay']=delayed_flights.iloc[:,[14,15,24,25,26,27,28]]
```

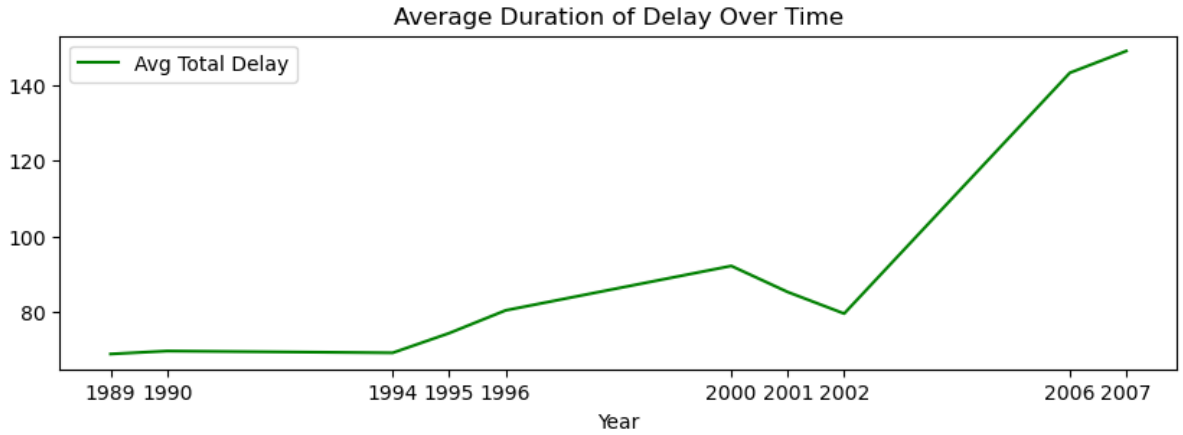
We then find the yearly average duration of delay by grouping the dataframe by the column 'Year' using the `groupby()` function, then finding the mean, using the `mean()` function.

In [10]:
```
#Finding yearly average duration of delay
avgdel_year = pd.DataFrame(delayed_flights.groupby('Year')['Totaldelay'].mea
.rename(columns={"Totaldelay": "Avg Total Delay"})
display(avgdel_year)
```

Year	Avg Total Delay
1989	68.990946
1990	69.775705
1994	69.302306
1995	74.482636
1996	80.498853
2000	92.229580
2001	85.342674
2002	79.653684
2006	143.194183
2007	148.943807

`from, import` imports a specific member or members of the library. We import the `pyplot` function from the `matplotlib` package to plot the data.

In [11]:
```
from matplotlib import pyplot as plt
#Plotting the graph
avgdel_year.plot(y='Avg Total Delay', use_index=True, #Assigning columns to
                 c='green', #Choosing a color for the plot
                 xticks=avgdel_year.index) #Assigning names to the axes
plt.title("Average Duration of Delay Over Time") #Naming plot
plt.gcf().set_size_inches(10, 3) #Resizing plot
plt.show()
```



Plot above shows that the average duration of delay has increased over the year.

3.3 Flight destinations

In this section, we look into the trend of flight paths over the years.

We first filter to our desired data frame to exclude cancelled and diverted flights.

In [12]:
```
#Setting the dataset for this section
nocanc_flights = Dataset[(Dataset['Cancelled'] == 0) & (Dataset['Diverted']
```

We tabulate the frequency of each flight paths each year, then form the flight path in a new column.

In [13]:
```
#Finding yearly frequency of each flight paths
sum_flights_year = nocanc_flights[['Origin', 'Dest', 'Year']].value_counts()

#Forming the flight path
sum_flights_year["Flight"] = sum_flights_year['Origin'] +"-"+ sum_flights_ye
display(sum_flights_year)
```

	Origin	Dest	Year	Counts	Flight
0	SFO	LAX	1990	25622	SFO-LAX
1	LAX	SFO	1990	25284	LAX-SFO
2	LAX	SFO	1989	21375	LAX-SFO
3	SFO	LAX	1989	21207	SFO-LAX
4	LAX	LAS	1996	17472	LAX-LAS
...	...	...	...	...	...
37021	IAD	SYR	1990	1	IAD-SYR
37022	IAD	SWF	1990	1	IAD-SWF
37023	BDL	ISP	1995	1	BDL-ISP
37024	IAD	RDU	1994	1	IAD-RDU
37025	RSW	MIA	1995	1	RSW-MIA

37026 rows × 5 columns

Each year has different total observations. Thus, for accuracy purposes, we calculate the proportion of the frequency of the flight paths by the total observations. To do so, we find the total yearly observations first. Afterwards, we merge the data to the current data frame (sum_flights_year) by Year, using the `pd.merge()` function. To calculate the proportion ratio, we divide the yearly frequency of each flight paths by their corresponding yearly total observations.

In [14]:
```
#Finding total yearly observations
totalyearobs = pd.DataFrame(nocanc_flights[['Year']].value_counts().reset_in

#Merging to data frame 'sum_flights_year'
new_dataset = pd.merge(sum_flights_year,totalyearobs, on ='Year', how ="left

#Calculation proportion
new_dataset['proportion'] = new_dataset['Counts']/new_dataset['Total']
```

There are about 37026 different flight paths recorded in the dataset. Therefore, we will filter the data frame to only include the top 30 flight paths based on their total frequency. In doing so, we can identify the popular flight paths within the US.

Similar to the earlier part in this section, we tabulate the total frequency of each flight paths, then form the flight path in a new column.

In [15]:
```
#Finding frequency of each flight paths
sum_flights = nocanc_flights[['Origin', 'Dest']].value_counts().reset_index(

#Forming the flight path
sum_flights["Flight"] = sum_flights['Origin'] +"-"+ sum_flights["Dest"]
```

We then create a list of the flight paths with the top 30 highest total frequency.

In [16]:
```
#Finding the top 30 flight paths
topflightpaths = list((sum_flights.nlargest(30, 'Counts'))['Flight'])
```

We filter the new data frame (new_dataset) to include only the flight paths listed in the list (topflightpaths).

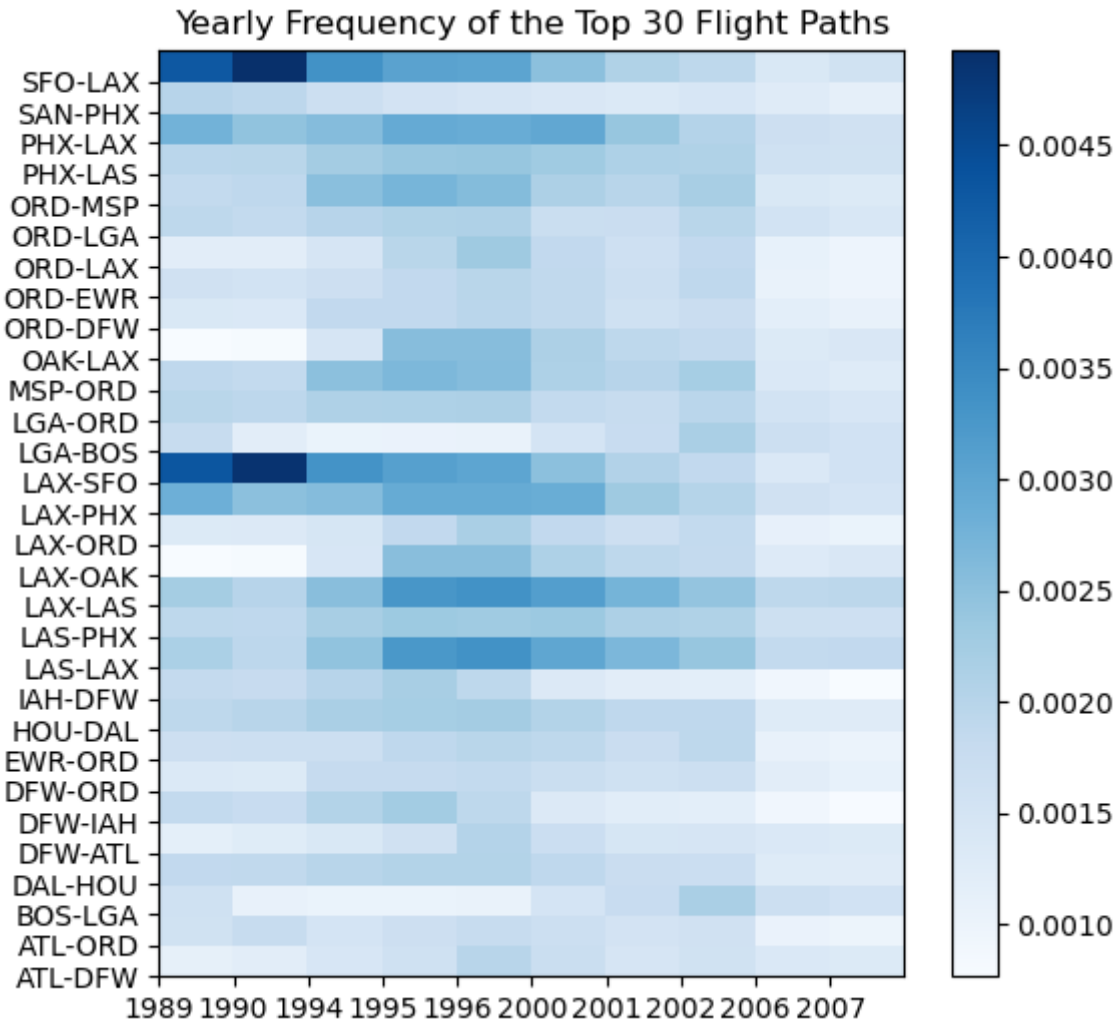
In [17]:
```
#Subsetting the list in sum_flights_year
freqtopflightpaths = new_dataset.query('Flight in @topflightpaths')
```

To visualise the data, we plot a heat map. To do so, we create a pivot table first to summarise and neatly organise the data by its parameters.

In [18]:
```
#Creating pivot table
pivot = freqtopflightpaths.pivot_table(index=['Flight'], columns='Year', val
```

We then create the heat map. NumPy is the fundamental package for scientific computing in Python. In this portion, we used the `arange()` function to convert the data set into an array for plot labelling purposes.

In [19]:
```
#Creating heat map
from matplotlib import pyplot as plt
import numpy as np
plt.pcolor(pivot, cmap='Blues')
plt.yticks(np.arange(len(pivot.index.values)),
              labels=pivot.index.values)
plt.xticks(np.arange(len(pivot.columns.values)),
              labels=pivot.columns.values)
plt.gcf().set_size_inches(6, 6)
plt.title("Yearly Frequency of the Top 30 Flight Paths")
plt.colorbar()
plt.show()
```



3.4 Cascading Failures

Cascading failures in the aviation industry can occur when delays in one airport create delays in other airports, resulting in a domino effect that can cause widespread disruption of air travel. In this section, we will demonstrate how we detect such failures.

To begin, we will take row '43210' as an example. We then find the relating flights by filtering the data to the same flight number, tail number and flight date.

In [20]:
```
#Choosing a random observation
pd.set_option('display.max_columns', None)
display(Y2007.iloc[2277704])
```

```
Year                    2007
Month                      4
DayofMonth                25
DayOfWeek                  3
DepTime               1641.0
CRSDepTime              1610
ArrTime               1853.0
CRSArrTime              1814
UniqueCarrier             NW
FlightNum               1177
TailNum                N677MC
ActualElapsedTime      132.0
CRSElapsedTime         124.0
AirTime                100.0
ArrDelay                39.0
DepDelay                31.0
Origin                   BDL
Dest                     DTW
Distance                 548
TaxiIn                     7
TaxiOut                   25
Cancelled                  0
CancellationCode         NaN
Diverted                   0
CarrierDelay               3
WeatherDelay               0
NASDelay                   8
SecurityDelay              0
LateAircraftDelay         28
Name: 2277704, dtype: object
```

In [21]:
```
#Finding flights with the same flight number and tail number
E2 = Y2007[(Y2007['FlightNum'] == 1177) & (Y2007['TailNum'] == 'N677MC')
           & (Y2007['Month'] == 4) & (Y2007['DayofMonth'] == 25)]
```

In [22]:
```
pd.set_option('display.max_columns', None)
display(E2)
```

	Year	Month	DayofMonth	DayOfWeek	DepTime	CRSDepTime	ArrTime	CRSArrT
2277704	2007	4	25	3	1641.0	1610	1853.0	1
2277726	2007	4	25	3	1936.0	1912	2014.0	1

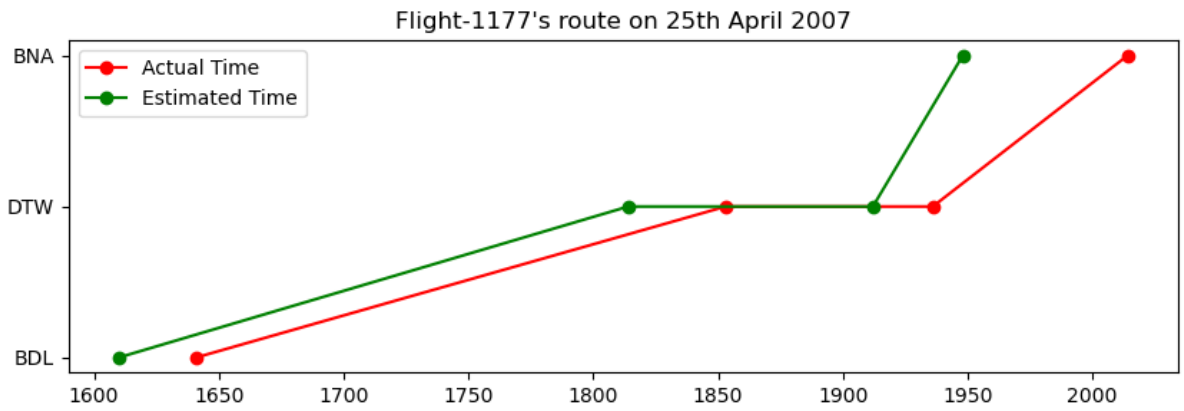
We then subset the data, keeping only the actual and estimated timings.

In [23]:
```
#Forming a filtered data frame of the actual time of arrival and departure
act_arrtime = (E2[['Dest','ArrTime']]).rename({'Dest': 'Airport', 'ArrTime':
act_deptime = (E2[['Origin', 'DepTime']]).rename({'Origin': 'Airport', 'DepT
#Arranging 'Time' in ascending order
act_time = (pd.concat([act_arrtime, act_deptime])).sort_values(by=['Time'])

#Repeating the steps for estimated time
est_arrtime = (E2[['Dest','CRSArrTime']]).rename({'Dest': 'Airport', 'CRSArr
est_deptime = (E2[['Origin', 'CRSDepTime']]).rename({'Origin': 'Airport', 'C
est_time = pd.concat([est_arrtime, est_deptime]).sort_values(by=['Time'])
```

Next, we plot a graph to visualise the time path.

In [24]:
```
#Plotting the graph
plt.plot(act_time['Time'], act_time['Airport'], linestyle="-", marker="o", c
plt.plot(est_time['Time'], est_time['Airport'], linestyle="-", marker="o", c
plt.title("Flight-1177's route on 25th April 2007")
plt.gcf().set_size_inches(10, 3)
plt.legend()
plt.show()
```



It can be observed from the figure above that the delay in flight-1177's departure from BDL airport led to a more than proportionate delay as it landed at BNA airport.

3.5 Predicting delays

We take the following steps to predict the duration of delay based on the airline.

Data pre-processing refers to the process of transforming raw data into a more refined, cleaned data set (Mesevage, 2021). We do so by removing all the null-valued columns and rows using the `dropna()` function. We then filter the data by creating a subset of the relevant columns.We create a new column for the total delay where we sum the columns 'ArrDelay' and 'DepDelay'.

In [25]:
```
#Drop rows with missing values
Dataset.dropna

#drop columns, i.e. subset relevant columns
Dataset1 = Dataset[["UniqueCarrier"]]
Dataset1['Totaldelay'] = Dataset['ArrDelay'] + Dataset["DepDelay"]
```

Next, we define variables X and y where X represents the independent variable while y represents the dependent variables whose values are to be predicted.

The `reshape()` function allows us to reshape an array without changing its data. It fundamentally means changing the shape of an array where the shape of the array is determined by the number of elements in each dimension. In this case, we reshape our data using `array.reshape(-1, 1)` as our X and y variables both have a single feature.

In [26]:
```
X = Dataset1["UniqueCarrier"].values.reshape(-1,1)
y = Dataset1["Totaldelay"].values.reshape(-1,1)
```

We move on to dividing the data to train and set. The 'train' data is used to train, or develop, the model, and the 'test' data is to test the accuracy of said model. We import the `train_test_split` function from the `sklearn.model_selection` package to split the data. The split percentages are as follow: 75% for training, and 25% for testing.

In [27]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.75, r
```

We now attempt to train our algorithm. For that, we need to import the `LinearRegression` function from the `sklearn.linear_model` package. We instantiate it, and call the `fit()` function along with our training data. The linear regression model is often used for analysis prediction (Statistics Solutions, 2013). The regression can be used to recognize the extent of the effect that the independent variable(s) have on a dependent variable, which in this case, the effect of the airline company have on the duration of delay.

In [ ]:
```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

We then use our test data and see how accurately our model predicts the total delay.

In [ ]:
```
y_pred = regressor.predict(X_test)
```

To make predictions, we create a new set of data, and test it using the model.

In [ ]:
```
#Creating new data
X_newdata = ["AS", "DL", "HA"]

#Test against the model
y_pred = regressor.predict(X_newdata)
```

4 Conclusion

To conclude, the study of commercial flight efficiency in the US with the application of Python programming underlines the significance of data analysis and informed decision-making in the aviation sector.

As demonstrated in this report, we have performed various analyses, create visualisations and develop code chunks and models with the use of Python programming. Python programming is a useful tool for observing and organizing information; it is versatile, it can be customized to suit different needs, and it is easy to use even for those who are new to it.

References

(Worsley, 2022), "What is Python? - The Most Versatile Programming Language: https://www.datacamp.com/blog/all-about-python-the-most-versatile-programming-language

(Udacity, 2021), "What Is a Python Package?": https://www.udacity.com/blog/2021/01/what-is-a-python-package.html

(Harvard Dataverse, 2008), "Data Expo 2009: Airline on time data": https://doi.org/10.7910/DVN/HG7NV7

(Bureau of Transportation Statistics, -), "Airline On-Time Statistics and Delay Causes": https://www.transtats.bts.gov/ot_delay/ot_delaycause1.asp

(Mesevage, 2021), “What Is Data Preprocessing & What Are The Steps Involved?”: https://monkeylearn.com/blog/data-preprocessing/

(Statistics Solutions, 2013), “What is Linear Regression”: https://www.statisticssolutions.com/free-resources/directory-of-statistical-analyses/what-is-linear-regression/

In [ ]:
```

```