coursework part 2

In this report we will be conducting an analysis on US flight details records from 1990 to 1999. We will focus on these years to avoid black swan events such as those that occured in 2001 and 2008 and have consistent data

Data inspection

We will start by importing necessary libraries

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

Then appload the data

In [ ]:
```
df_1990 = pd.read_csv("1990.csv") #importing the csv files
pd.set_option('display.max_columns', None) #this allows us to view all the columns of th
df_1990.head() # viewing the first few rows
```

In [ ]:
```
df_1990.shape #get the number of rows and columns
```

we have a large number of rows and 29 attributes

In [ ]:
```
df_1990.columns
```

A description of each of these attributes will be attached in the appendix

In [ ]:
```
df_1990.info()
```

This shows us the types of each variable , There are 3 categorical variables and 26 numerical

Data cleaning

we will start looking for general anomalities in our data in order to get more precise insights First we need some statistical insights on our numerical variables

In [ ]:
```
df_1990.describe().T
```

The data seems coherent with the description :it's the data of year 1990 , times are well within the range (0001;2400)... However there is a problem with the minmal value of "ActualElapsedTime" and "CRSElapsedTime" : a negative value expressing time duration seems not logical thus we will see how much of this data we have

In [ ]:
```
df_1990[((df_1990['CRSElapsedTime']<=0) | (df_1990['ActualElapsedTime']<=0) )].shape
```

the number of flawed data is small comapred to the population so it wouldn't be harmful to delete those lines if necessary

now we will inspect our categorical data

In [ ]:
```
df_1990.describe(include=object).T
```

this tells us that most flights are from ATL (Hartsfield-Jackson Atlanta International Airport) to ORD (O'Hare International Airport)

Best Time and Day of Week to minimize delays

(a) What are the best times and days of the week to minimise delays each year? in order to answer this question we start by creating a table containing each day with its corresponding mean delay

In [ ]:
```
df_1990.groupby(['Year','DayOfWeek']).agg({'DepDelay':'mean', 'ArrDelay':'mean'})
```

Choice of delay metric

when looking closely at the data we find that when a plane is diverted we have NaN values for the ArrDelay (since that only happens for emergencies usually , we don't care about arrival delay anymore) and this represents a considerable part of the data so that would affect our conclusions

In [ ]:
```
df_1990[(df_1990['Diverted']==1)]
```

In [ ]:
```
df_1990[(df_1990['Diverted']==1)].shape
```

Dropping NaN values wouldn't be the optimal solution and when relying only on departure time we may get biased data since we wouldn't take into consideration delay variation during the flight (delay could increase or decrease) Thus we decided to construct an estimator that sums departure and arrival delays and devides them by 2 and if a value is missing we use the othre one (second best estimator)

In [ ]:
```
copy=df_1990.copy() # we first make a copy of the data frame
copy['DepDelay'].fillna(copy['ArrDelay'], inplace=True) # we replace the nan values of D
copy['ArrDelay'].fillna(copy['DepDelay'], inplace=True) # we replace the nan values of A
copy["Delay"] = (copy['DepDelay'] + copy['ArrDelay']) / 2 # we get the mean
copy["Delay"]
```

Now we get the more consistent response

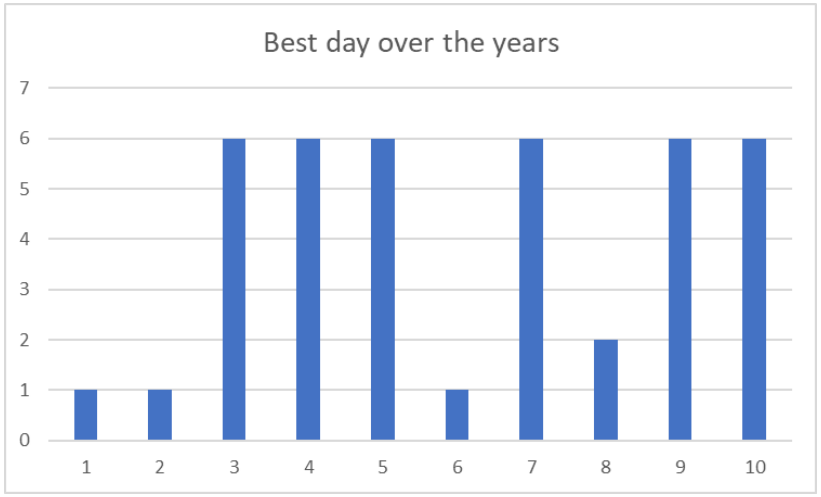
Best Day

In [ ]:
```
Delay_by_day=copy.groupby(['Year','DayOfWeek']).agg({'Delay':'mean'})
Delay_by_day
```

In [ ]:
```
Delay_by_day["Delay"].idxmin()
```

So the best day of the week to minimize delay would be Sunday

And over the years we get :



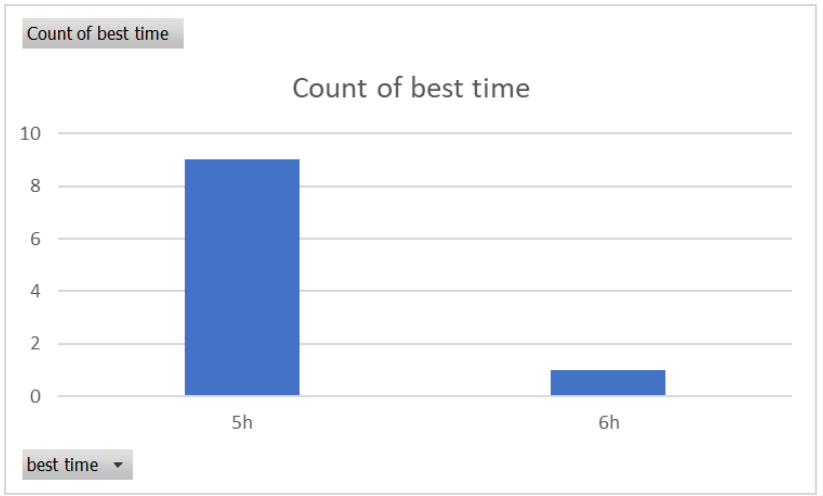
Best time

In [ ]:
```
copy2 = df_1990.copy() # we start by making a copy of the data frame

copy2 = copy2.dropna(subset=['DepTime']) # we drop the NaN values


# we use the same technique for the calculation of the matric as before

copy2['DepDelay'].fillna(copy2['ArrDelay'], inplace=True) # we replace the nan values of
copy2['ArrDelay'].fillna(copy2['DepDelay'], inplace=True) # we replace the nan values of
copy2["Delay"] = (copy2['DepDelay'] + copy2['ArrDelay']) / 2 # we get the mean



# This part is for data manipulation

copy2['DepTime'] = copy2['DepTime'].astype(str) # we convert the column's type into stri
copy2['DepTime'] = copy2['DepTime'].str.split('.').str[0] # we do this to get rid of the


# Define a function to extract hour information
def extract_hour(time_str):
    if len(time_str) == 3 and time_str[:2].isdigit():
        return int(time_str[:1])
    elif len(time_str) == 4 and time_str[:2].isdigit():
        return int(time_str[:2])
    else:
        return None

# Apply the function to 'DepTime' column to extract hour
copy2['hour'] = copy2['DepTime'].apply(extract_hour)


# Group by hour
grouped_by_hour = copy2.groupby('hour').agg({'Delay':'mean'})
```

```
grouped_by_hour
```

In [ ]:
```
grouped_by_hour["Delay"].idxmin()
```

after repeating this over the years we get that :



So the best time of the day to minimize delay would be earlier in the morning (around 5 and 6)

Age of planes and delays relationship

We will start by uploading the palnes file

In [ ]:
```
Planes=pd.read_csv('plane-data.csv')
df_1995= pd.read_csv('1995.csv')
Planes.tail()
```

We need to put the data we need in the same row in order to calculate a correlation coefficient and see how true that assumption is

In [ ]:
```
copy3 = df_1995.copy() # we prepare a copy

copy3['TailNum'] = copy3['TailNum'].astype(str) # we convert this column type into an ob

# we merge the columns containing year and the copy based on the tailnum
age_df=pd.merge(copy3, Planes[['tailnum','year']], left_on='TailNum',right_on='tailnum',


age_df['age'] = 2008 - age_df['year'] # we add a coulumn containing the age
```

We then prepare our delay metric

In [ ]:
```
age_df['DepDelay'].fillna(age_df['ArrDelay'], inplace=True) # we replace the nan values
age_df['ArrDelay'].fillna(age_df['DepDelay'], inplace=True) # we replace the nan values
age_df["Delay"] = (age_df['DepDelay'] + age_df['ArrDelay']) / 2 # we get the mean
age_df["Delay"]
```

Then we make a new data frame containing the two variables we're intersted in

In [ ]:
```
grouped_by_age= age_df.groupby('age').agg({'Delay':'mean'})
grouped_by_age.reset_index(inplace=True) # we get age as a seperate column
grouped_by_age
```

We plot the barplot to visualize the data

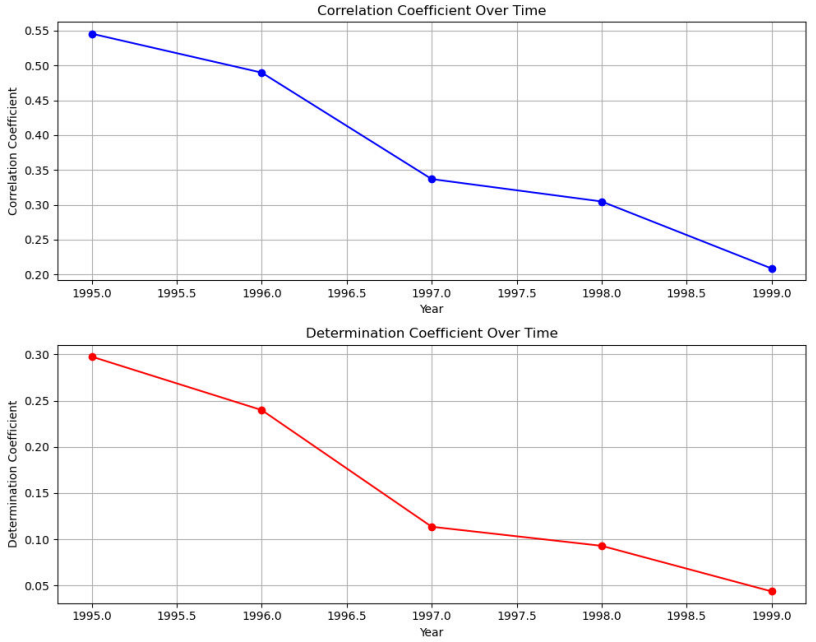
In [ ]:
```
plt.bar(grouped_by_age['age'], grouped_by_age['Delay'], color='blue')

# Add labels and title
plt.xlabel('Age')
plt.ylabel('Delays')
plt.title('Delays vs Age')

# Show the plot
plt.show()
```

We see there is a downward tendency in the slope, this means that there is probably a negative relationship between age and delays .

A probable reason for that is that newer planes may have some issues initially when deployed whereas older planes have already gone through that phase and are probably more stable and consistent as they get older

In [ ]:
```
corr_coef = grouped_by_age['Delay'].corr(grouped_by_age['age'])
print('r = ',corr_coef, 'r^2 = ',corr_coef**2)
```

the correlation coefficient is significant , the age of the plane explains about 30% of the delays that happen
After repeating this over the years we get this plot



we can see that as the years go , the effect diminishes over time , this may be explained by the advance in technology which improves the adaptation of planes quickly

Logistic regression

(c) For each year, fit a logistic regression model for the probability of diverted US flights using as many features as possible from attributes of the departure date, the scheduled departure and arrival times, the coordinates and distance between departure and planned arrival airports, and the carrier. Visualize the coefficients across years.

First we start by exploring the relationships between "Diverted" and the other variables , to choose the variables we will depend on in our model

In [ ]:
```
copy4 = df_1995.copy()
copy4
```

In [ ]:
```
copy4 = copy4.applymap(pd.to_numeric, errors='coerce')
corr=copy4.corrwith(copy4["Diverted"])
corr_df=pd.DataFrame(corr, columns=['Correlation'])
corr_df
```

after getting the results we notice that taxi in is the best regressor we have, upon further inspection we notice an intresting relationship between TaxiIn and Diverted : Whenever Diverted = 1, TaxiIn = 0

In [ ]:
```
# we compare the shape of the data frame where "Diverted"=1 to the shape of the datafram

copy4[(copy4['Diverted']==1)].shape == copy4[((copy4['TaxiIn']==0) & (copy4['Diverted']=
```

but not whenever TaxiIn = 0 Diverted = 1

In [ ]:
```
# we compare the shape of the data frame where "Diverted"=1 to the shape of the datafram

copy4[(copy4['TaxiIn']==0)].shape == copy4[((copy4['TaxiIn']==0) & (copy4['Diverted']==1
```

So it would be better to make a dummy variable x=1 when TaxiIn =0 and x=0 whenever TaxiIn !=0 and use it as an indicator to avoid giving extra weight for the large values where as we only care if it's equal to 0 or not

In [ ]:
```
copy4['Taxi_dum']= (copy4['TaxiIn']==0).astype(int)
copy4
```

In [ ]:
```
cor = copy4['Diverted'].corr(copy4['Taxi_dum'])
print('r = ',cor, 'r^2 = ',cor**2)
```

That is way higher than the result before so this is a better regressor

We also could use the observation we made earlier where whenever 'ArrDelay' had a NaN value the the plane was diverted , the correlation table was not able to capture this relation due to it being a 'NaN' value , so we could further test this information by creating a new dummy variable 'Arr_dum'

In [ ]:
```
copy4['Arr_dum'] = (copy4['ArrTime'].isnull()).astype(int)
copy4
```

In [ ]:
```
cor2 = copy4['Diverted'].corr(copy4['Arr_dum'])
print('r = ',cor2, 'r^2 = ',cor2**2)
```

this is also a very good regressor we decided to use these two variables

we decided to use the top 4 variables (to include some controle variables) The variables are : 'Taxi_dum', 'DepDelay', 'CRSElapsedTime' and 'Distance'

Data cleaning

we start by cleaning the data

In [ ]:
```
model_data=copy4[['Taxi_dum','Arr_dum','TaxiIn','ArrDelay', 'DepDelay','CRSElapsedTime',
model_data.isnull().sum()
```

We will fill the missing values with mean

In [ ]:
```
model_data = model_data.fillna(model_data.mean())

model_data.isnull().sum()
```

In [ ]:
```
model_data
```

Modeling

We start by specifying the features and target set

In [ ]:
```
# importing libraries
from sklearn.model_selection import train_test_split

# Splitting the features and the target
X = model_data.drop(columns=["Diverted"])
Y = model_data['Diverted']

# Split the data into training and test sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2)
```

Regression

we run our regression

In [ ]:
```
from sklearn.linear_model import LogisticRegression

logmodel = LogisticRegression(max_iter=1000)

logmodel.fit(X_train,Y_train)
```

In [ ]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay

# Generate predictions with the best model
Y_pred = logmodel.predict(X_test)

# Create the confusion matrix
cm = confusion_matrix(Y_test, Y_pred)

ConfusionMatrixDisplay(confusion_matrix=cm).plot();
```

we will now inspect the classification report

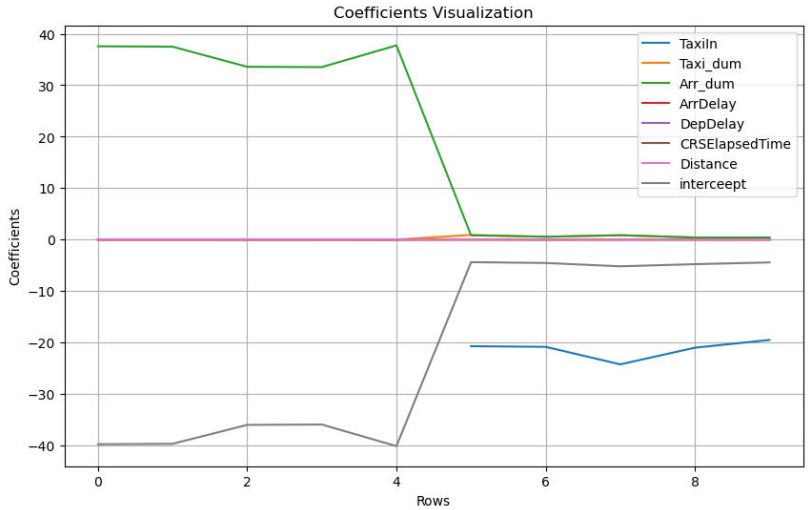
In [ ]:
```
from sklearn.metrics import classification_report

print(classification_report(Y_test, Y_pred))#, target_names=target_names))
```

and finally we get the coefficients

In [ ]:
```
coefficients = logmodel.coef_
intercept = logmodel.intercept_

print("Coefficients des variables :", coefficients)
print("Intercept :", intercept)
```

Data visualisation

Repeating this across the years we get a line plot



we can see that the efffect of missing data in the first 5 years is very remarquable , however that highlights the power and effect of the introduced variable and it it seems stable across the years too