Collect the dataset:

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link: flightdata.csv - Google Drive

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.

Note: There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Importing The Libraries

Import the necessary libraries as shown in the image. (optional) Here we have used visualisation style as fivethirtyeight.

Read the dataset:

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of the csv file.

Data preparation:

As we have understood how the data is, let's pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

- Handling missing values
- Handling categorical data

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

Handling Missing Values

- Let’s find the shape of our dataset first. To find the shape of our data, the df.shape method is used. To find the data type, df.info() function is used.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11231 entries, 0 to 11230
Data columns (total 26 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   YEAR                 11231 non-null  int64
 1   QUARTER              11231 non-null  int64
 2   MONTH                11231 non-null  int64
 3   DAY_OF_MONTH         11231 non-null  int64
 4   DAY_OF_WEEK          11231 non-null  int64
 5   UNIQUE_CARRIER       11231 non-null  object
 6   TAIL_NUM             11231 non-null  object
 7   FL_NUM               11231 non-null  int64
 8   ORIGIN_AIRPORT_ID    11231 non-null  int64
 9   ORIGIN               11231 non-null  object
 10  DEST_AIRPORT_ID      11231 non-null  int64
 11  DEST                 11231 non-null  object
 12  CRS_DEP_TIME         11231 non-null  int64
 13  DEP_TIME             11124 non-null  float64
 14  DEP_DELAY            11124 non-null  float64
 15  DEP_DEL15            11124 non-null  float64
 16  CRS_ARR_TIME         11231 non-null  int64
 17  ARR_TIME             11116 non-null  float64
 18  ARR_DELAY            11043 non-null  float64
 19  ARR_DEL15            11043 non-null  float64
 20  CANCELLED            11231 non-null  float64
 21  DIVERTED             11231 non-null  float64
 22  CRS_ELAPSED_TIME     11231 non-null  float64
 23  ACTUAL_ELAPSED_TIME  11043 non-null  float64
 24  DISTANCE             11231 non-null  float64
 25  Unnamed: 25          0 non-null      float64
dtypes: float64(12), int64(10), object(4)
memory usage: 2.2+ MB
```

- For checking the null values, df. Isnull () function is used. To sum those null values we use. sum() function. From the below image we found that there are no null values present in our dataset. So we can skip handling the missing values step.

- We will fill in the missing values in the numeric data type using the mean value of that particular column and categorical data type using the most repeated value.

In [94]:
```
dataset=dataset[["FL_NUM","MONTH","DAY_OF_MONTH","ORIGIN","DEST","CRS_ARR_TIME","DEP_DEL15","ARR_DEL15"]]
dataset.isnull().sum()
```

Out[94]:
```
FL_NUM            0
MONTH             0
DAY_OF_MONTH      0
ORIGIN            0
DEST              0
CRS_ARR_TIME      0
DEP_DEL15       107
ARR_DEL15       188
dtype: int64
```

In [95]:
```
dataset[dataset.isnull().any(axis=1)].head(10)
```

Out[95]:

	FL_NUM	MONTH	DAY_OF_MONTH	ORIGIN	DEST	CRS_ARR_TIME	DEP_DEL15	ARR_DEL15
177	2834	1	9	MSP	SEA	852	0.0	NaN
179	86	1	10	MSP	DTW	1632	NaN	NaN
184	557	1	10	MSP	DTW	912	0.0	NaN
210	1096	1	10	DTW	MSP	1303	NaN	NaN
478	1542	1	22	SEA	JFK	723	NaN	NaN
481	1795	1	22	ATL	JFK	2014	NaN	NaN
491	2312	1	22	MSP	JFK	2149	NaN	NaN
499	423	1	23	JFK	ATL	1600	NaN	NaN
500	425	1	23	JFK	ATL	1827	NaN	NaN
501	427	1	23	JFK	SEA	1053	NaN	NaN

Handling Categorical Values

As we can see our dataset has categorical data we must convert the categorical data to integer encoding or binary encoding.

To convert the categorical features into numerical features we use encoding techniques. There are several techniques but in our project we are using manual encoding with the help of list comprehension.

```
for index,row in dataset.iterrows():
    dataset.loc[index,'CRS_ARR_TIME']- math.floor(row['CRS_ARR_TIME']/100)
dataset.head()
```

	FL_NUM	MONTH	DAY_OF_MONTH	ORIGIN	DEST	CRS_ARR_TIME	DEP_DEL15	ARR_DEL15
0	1399	1	1	ATL	SEA	2143	0.0	0.0
1	1476	1	1	DTW	MSP	1435	0.0	0.0
2	1597	1	1	ATL	SEA	1215	0.0	0.0
3	1768	1	1	SEA	MSP	1335	0.0	0.0
4	1823	1	1	SEA	DTW	607	0.0	0.0

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
dataset['DEST']=le.fit_transform(dataset['DEST'])
dataset['ORIGIN']=le.fit_transform(dataset['ORIGIN'])
```

```
x=dataset.iloc[:,0:8].values
y=dataset.iloc[:,8:9].values
```

```
x
```

```
array([[1.399e+03, 1.000e+00, 1.000e+00, ..., 0.000e+00, 1.000e+00,
        0.000e+00],
       [1.476e+03, 1.000e+00, 1.000e+00, ..., 0.000e+00, 0.000e+00,
        1.000e+00],
       [1.597e+03, 1.000e+00, 1.000e+00, ..., 0.000e+00, 1.000e+00,
        0.000e+00],
       ...,
       [1.823e+03, 1.200e+01, 3.000e+01, ..., 0.000e+00, 0.000e+00,
        1.000e+00],
       [1.901e+03, 1.200e+01, 3.000e+01, ..., 0.000e+00, 1.000e+00,
        0.000e+00],
       [2.005e+03, 1.200e+01, 3.000e+01, ..., 0.000e+00, 1.000e+00,
        0.000e+00]])
```

```
from sklearn.preprocessing import OneHotEncoder
oh=OneHotEncoder()
z=oh.fit_transform(x[:,4:5]).toarray()
t=oh.fit_transform(x[:,5:6]).toarray()
```

```
z
```

```
array([[1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       ...,
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.]])
```

```
t
```

```
array([[1., 0.],
       [1., 0.],
       [1., 0.],
       ...,
       [1., 0.],
       [1., 0.],
       [1., 0.]])
```

```
x=np.delete(x,[4,5],axis=1)
```