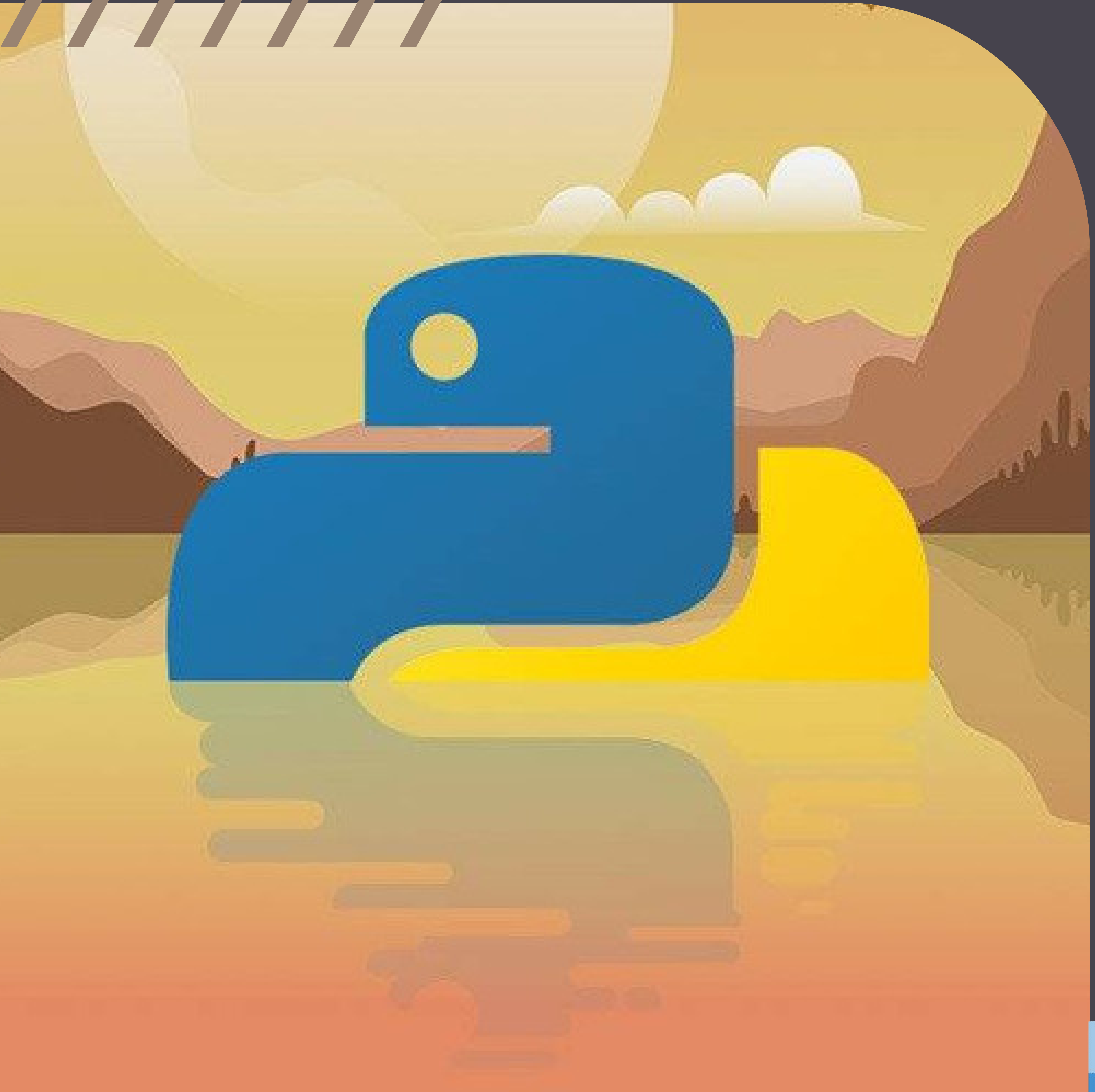


Data Cleansing & Exploratory Data Analysis

Created by
Seaborn Kusto

Official Team

Malka Rusyd Abdussalam

Trianto Haryo Nugroho

Maulana Ishaq Siregar

Fauzi Wardah Ali

Rizki Afrinal

Table of Content

Data Cleansing

Presentations are communication tools that can be used as demonstrations, lectures, spee

Importing Libraries

We will start by importing the libraries we will require for data cleansing. These include Pandas, NumPy, Matplotlib, and Seaborn

- Pandas is a Python library for data analysis.
- Numpy is a Python library for mathematical operations.
- Matplotlib is a Python library for data visualization.
- Seaborn is a Python library for data visualization and exploratory data analysis.

```
[ ] # Import the required Libraries (pandas, numpy, matplotlib, and seaborn)

    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    %matplotlib inline
    import seaborn as sns
```

Uploading Dataset

The next step after importing libraries is uploading the dataset from the storage location to google colab. The dataset that we will use is the Titanic.csv dataset obtained from Kaggle.

```
[ ] # Uploading dataset to google colab from existing location

    from google.colab import files
    files.upload()
```

```
Choose Files No file chosen     Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving Titanic.csv to Titanic.csv
{'Titanic.csv': b'PassengerId,Survived,Pclass,Name,Sex,Age,SibSp,Parch,Ticket,Fare,Cabin,Embarked\r\n1,0,3,"Braund, Mr. Owen Harris",male,22,1,0,A/5 21171,7.25,,S\r\n2,1,1,"Cuming:
```

Reading Dataset

After the dataset is uploaded then we read the contents of the dataset.

```
[ ] # Read the Titanic.csv dataset

    df = pd.read_csv('Titanic.csv')
```

Displaying DataFrame Information

The info() method prints information about 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 (non-null values). Note: the info() method actually prints the info.

We have to check the state of data in each variable/column before taking any action in data manipulation and data cleaning

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 891 entries, 1 to 891
Data columns (total 11 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Survived  891 non-null    int64
 1   Pclass    891 non-null    int64
 2   Name      891 non-null    object
 3   Sex       891 non-null    object
 4   Age       714 non-null    float64
 5   SibSp     891 non-null    int64
 6   Parch     891 non-null    int64
 7   Ticket    891 non-null    object
 8   Fare      891 non-null    float64
 9   Cabin     204 non-null    object
 10  Embarked  889 non-null    object
dtypes: float64(2), int64(4), object(5)
memory usage: 83.5+ KB
```

Column Age

The describe() method prints information about the data description starting with count, mean, standard deviation, min, and max. The value_counts() method prints information counting for each value. And the plot() method display graphs according to the format we specify

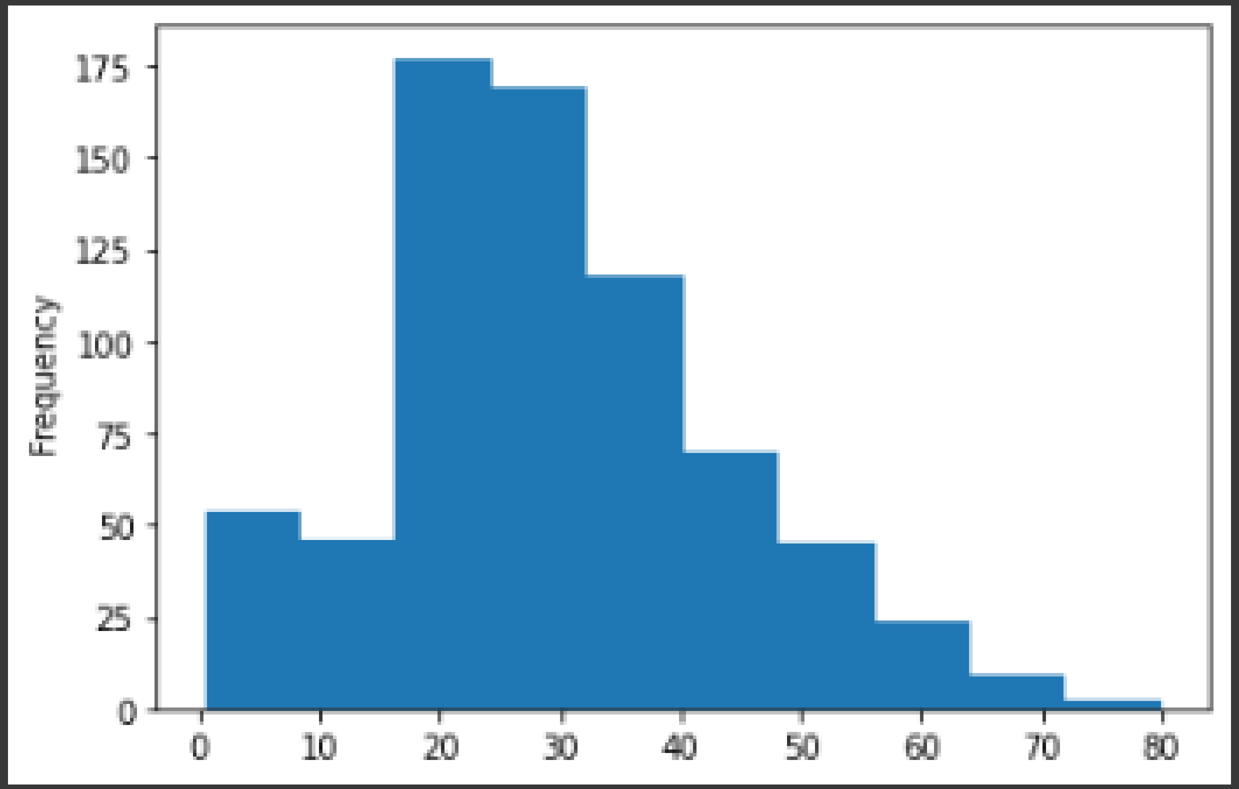
```
[ ]  df['Age'].describe()

count    714.000000
mean      29.699118
std       14.526497
min        0.420000
25%       20.125000
50%       28.000000
75%       38.000000
max       80.000000
Name: Age, dtype: float64
```

```
[ ]  df['Age'].value_counts()

24.00    30
22.00    27
18.00    26
19.00    25
28.00    25
         ..
36.50     1
55.50     1
0.92      1
23.50     1
74.00     1
Name: Age, Length: 88, dtype: int64
```

```
[ ]  df['Age'].plot(kind='hist');
```



Column Age

because the Age coulumn shows skewness, so we have to apply an imputation with median, and we can check data information again.

```
[ ]  val =df['Age'].median()
     df['Age'] = df['Age'].fillna(val)
```

```
[ ]  df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          891 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

Column Cabin

the sum of data entry is 891 yet the Cabin Column is 204. it means there must be NULL in the Cabin column. and we can show Cabin column proportion

```
[ ]  df['Cabin'].value_counts()

     B96 B98        4
     G6             4
     C23 C25 C27    4
     C22 C26        3
     F33            3
                   ..
     E34            1
     C7             1
     C54            1
     E36            1
     C148           1
     Name: Cabin, Length: 147, dtype: int64
```

It showed that Cabin column value has too much unique data and also the Cabin clumn info is not quite informative to describe survived data. So we better remove the Cabin column

```
[ ]  df.drop('Cabin', axis=1, inplace = True)
```

Column Embarked

the sum of data entry is 891 yet the Embarked Column is 889. We can show the Embarked column proportion. Embarked column is categorical data

```
df['Embarked'].value_counts()

S    644
C    168
Q     77
Name: Embarked, dtype: int64
```

we will apply an imputation on Embarked column. so we check the data type of the EMbarked column first. Embarked column is categorical data so the imputation is using mode. from the proportion of EMbarked column, S appeared the most. so S is the mode

```
[ ] val = df.Embarked.mode().values[0]
    df['Embarked'] = df.Embarked.fillna(val)
```

After all data cleansing. we can check information data. There's no more missing value

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          891 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Embarked     891 non-null    object 
dtypes: float64(2), int64(5), object(4)
memory usage: 76.7+ KB
```

Data Manipulation
Column SibSp and Column Parch

We will do data manipulation. Manipulation here doesn't mean changing the data value but to ease a machine to read data. SibSp column (sibling Spouse) ia a column that state the number of siblings or partner came with the pessenger. Parch(Parent Childern) column is a column that state the number of parents or children came with the pessenger.
we will make a new column that shows whether the pessenger is alone or coming with their family.
So we can show the new data

```
[ ]  df['Alone'] = df['SibSp'] + df['Parch']
```

```
[ ]  df['Alone'][df['Alone']>0]='With Family'
     df['Alone'][df['Alone']==0]='With Family'
```

```
[ ]  df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	Alone
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	S	With Family
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C	With Family
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	S	With Family
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	S	With Family
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	S	With Family

Data Visualization
Realtion between sex column and survived column

let's see the survived Sex column proportion and compare it with the Sex column which is not survived. And we can show the visualization of Sex column which is surivived and not survived

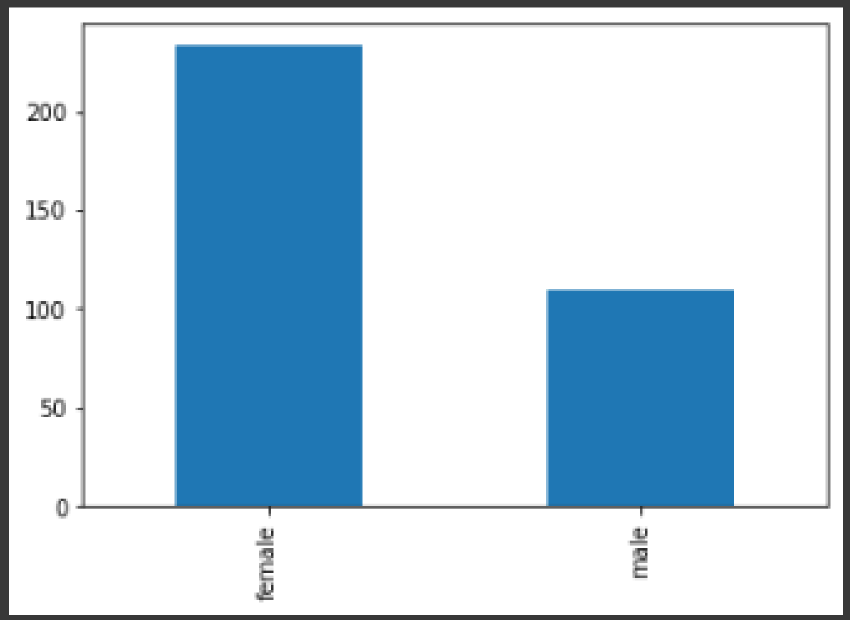
```
[ ]  df.Sex[df['Survived']==1].value_counts()

     female    233
     male      109
     Name: Sex, dtype: int64
```

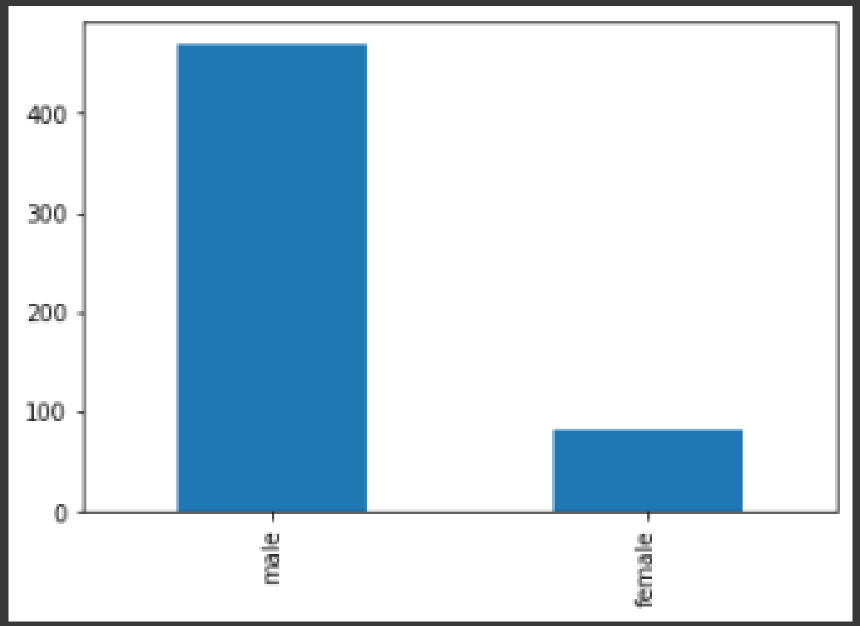
```
[ ]  df.Sex[df['Survived']==0].value_counts()

     male      468
     female     81
     Name: Sex, dtype: int64
```

```
[ ]  df.Sex[df['Survived']==1].value_counts().plot(kind='bar');
```




```
[ ]  df.Sex[df['Survived']==0].value_counts().plot(kind='bar');
```

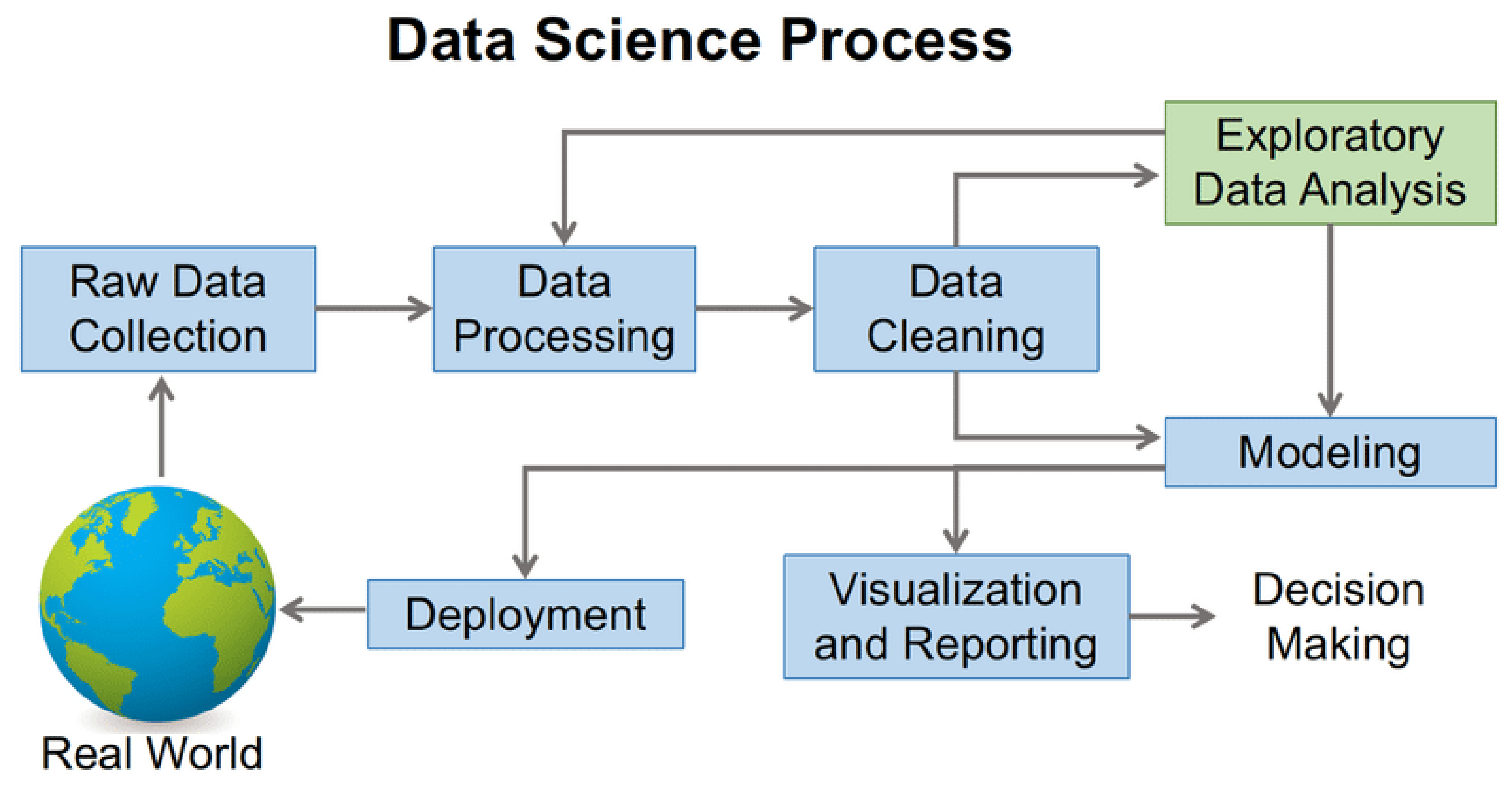


Exploratory Data Analysis

Exploratory Data Analysis, or EDA, is an important step in any Data Analysis or Data Science project. EDA is the process of investigating the dataset to discover patterns, and anomalies (outliers), and form hypotheses based on our understanding of the dataset.

EDA involves generating summary statistics for numerical data in the dataset and creating various graphical representations to understand the data better.

Before we delve into EDA, it is important to first get a sense of where EDA fits in the whole data science process.



Importing Libraries

We will start by importing the libraries we will require for performing EDA. These include Pandas, NumPy, Matplotlib, and Seaborn

- Pandas is a Python library for data analysis.
- Numpy is a Python library for mathematical operations.
- Matplotlib is a Python library for data visualization.
- Seaborn is a Python library for data visualization and exploratory data analysis.

```
[ ]  # Import the required Libraries (pandas, numpy, matplotlib, and seaborn)

     import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     %matplotlib inline
     import seaborn as sns
```

Uploading Dataset

The next step after importing libraries is uploading the dataset from the storage location to google colab. The dataset that we will use is the Titanic.csv dataset obtained from Kaggle.

```
[ ] # Uploading dataset to google colab from existing location

    from google.colab import files
    files.upload()
```

```
Choose Files  No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving Titanic.csv to Titanic.csv
{'Titanic.csv': b'PassengerId,Survived,Pclass,Name,Sex,Age,SibSp,Parch,Ticket,Fare,Cabin,Embarked\r\n1,0,3,"Braund, Mr. Owen Harris",male,22,1,0,A/5 21171,7.25,,S\r\n2,1,1,"Cuming:
```

Reading Dataset

After the dataset is uploaded then we read the contents of the dataset.

```
[ ] # Read the Titanic.csv dataset

    df = pd.read_csv('Titanic.csv')
```

Displaying Top 5 Rows

Returns the top 5 rows of the dataset to have a look at how our dataset looks like.

```
[ ] # Displaying the top 5 rows of the dataset

    df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Changing Index

Pandas default index starts from 0, while the dataset index of the PassengerId column starts from 1. Then we will use the index dataset of column PassengerId.

```
[ ] # Changing the index to start from 1

    df = pd.read_csv('Titanic.csv', index_col=0)
```

```
[ ] # Displaying the top 5 rows to see the changing of the index

    df.head()
```

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
PassengerId											
1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Displaying DataFrame Information

The info() method prints information about 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 (non-null values). Note: the info() method actually prints the info.

```
[ ]  # Displaying the information details about the DataFrame

     df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 891 entries, 1 to 891
Data columns (total 11 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Survived  891 non-null    int64
 1   Pclass    891 non-null    int64
 2   Name      891 non-null    object
 3   Sex       891 non-null    object
 4   Age       714 non-null    float64
 5   SibSp     891 non-null    int64
 6   Parch     891 non-null    int64
 7   Ticket    891 non-null    object
 8   Fare      891 non-null    float64
 9   Cabin     204 non-null    object
 10  Embarked  889 non-null    object
dtypes: float64(2), int64(4), object(5)
memory usage: 83.5+ KB
```

Checking Missing Value (NaN)

Checking whether there is a missing value (NaN) and also counting the number of the missing value in each columns in the dataset.

```
[ ]  # Displaying number of NaN (missing value) from the dataset

     df.isnull().sum()

     Survived      0
     Pclass        0
     Name          0
     Sex           0
     Age         177
     SibSp         0
     Parch         0
     Ticket        0
     Fare          0
     Cabin       687
     Embarked      2
     dtype: int64
```

Displaying Descriptive Statistics

Looking at descriptive statistic parameters for the dataset: Count, Mean, Standard Deviation, Maximum and Minimum , Quartile (25%, 50% and 75%).

```
# Displaying descriptive statistics for the dataset

df.describe()
```

	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Displaying Unique Values in a Column

Displaying all of the unique value and its data types in a Column.

```
[ ] # Displaying all of the unique values in the Sex Column.

    df.Sex.unique()

    array(['male', 'female'], dtype=object)

[ ] # Displaying all of the unique values in the Pclass Column.

    df.Pclass.unique()

    array([3, 1, 2])

[ ] # Displaying all of the non unique values in the Pclass Column.

    df.Pclass.nunique()
```

Displaying Proportion of Unique Values

Displays the data proportion of its unique values for the categoric data type.

```
[ ]  # Displaying the data proportion of its unique values for the Sex Column.

     df.Sex.value_counts()

     male      577
     female    314
     Name: Sex, dtype: int64
```

```
[ ]  # Displaying the data proportion of its unique values for the Embarked Column.

     df.Embarked.value_counts()

     S    644
     C    168
     Q     77
     Name: Embarked, dtype: int64
```

Displaying Shape (Number of Rows and Columns)

Displays the data proportion of its unique values for the categoric data type.

```
[ ]  # Displaying the number of rows and the number of columns of the dataset

     df.shape

     (891, 11)
```

Checking Duplicate Data

Checking the number of duplicate Data for each column.

```
[ ]  # Checking the number of duplicate data for each column

     df[df.duplicated()]
```

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
PassengerId											

Removing Duplicate Data

Remove all of duplicate data from the dataset.

```
[ ] # Remove the duplicate data from the dataset

    df.drop_duplicates()
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...
887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
889	0	3	Johnston, Miss. Catherine Helen 'Carrie'	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 11 columns

Embarked Column

Embarked column has 2 null data on PassangerId numer 62 and 830

```
df[df.Embarked.isnull()]
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
62	1	1	Icard, Miss. Amelie	female	38.0	0	0	113572	80.0	B28	NaN
830	1	1	Stone, Mrs. George Nelson (Martha Evelyn)	female	62.0	0	0	113572	80.0	B28	NaN

Embarked Column

Embarked is categoric data, we can use mode for imputation missing data in Embarked Column

```
val = df.Embarked.mode().values[0]
df["Embarked"]=df.Embarked.fillna(val)
```

Proportion Embarked

Before Imputation

```
df.Embarked.value_counts()
```

```
S    644
C    168
Q     77
Name: Embarked, dtype: int64
```

After Imputation

```
df.Embarked.value_counts()
```

```
S    646
C    168
Q     77
Name: Embarked, dtype: int64
```

Embarked Column

Change the object data in Embraked ('S', 'C', 'Q') to numerical data (0, 1, 2)

```
df.Embarked = df.Embarked.map({"S":0, "C":1, "Q":2})
#change data object to data numeric
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 891 entries, 1 to 891
Data columns (total 11 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Survived  891 non-null    int64  
 1   Pclass    891 non-null    int64  
 2   Name      891 non-null    object 
 3   Sex       891 non-null    object 
 4   Age       714 non-null    float64
 5   SibSp     891 non-null    int64  
 6   Parch     891 non-null    int64  
 7   Ticket    891 non-null    object 
 8   Fare      891 non-null    float64
 9   Cabin     204 non-null    object 
 10  Embarked  891 non-null    int64  
dtypes: float64(2), int64(5), object(4)
```

Age Column

Data Titanic has 891 row, in Age Column only 714 row, its mean Age Column has 177 missing data

```
df.info()
```

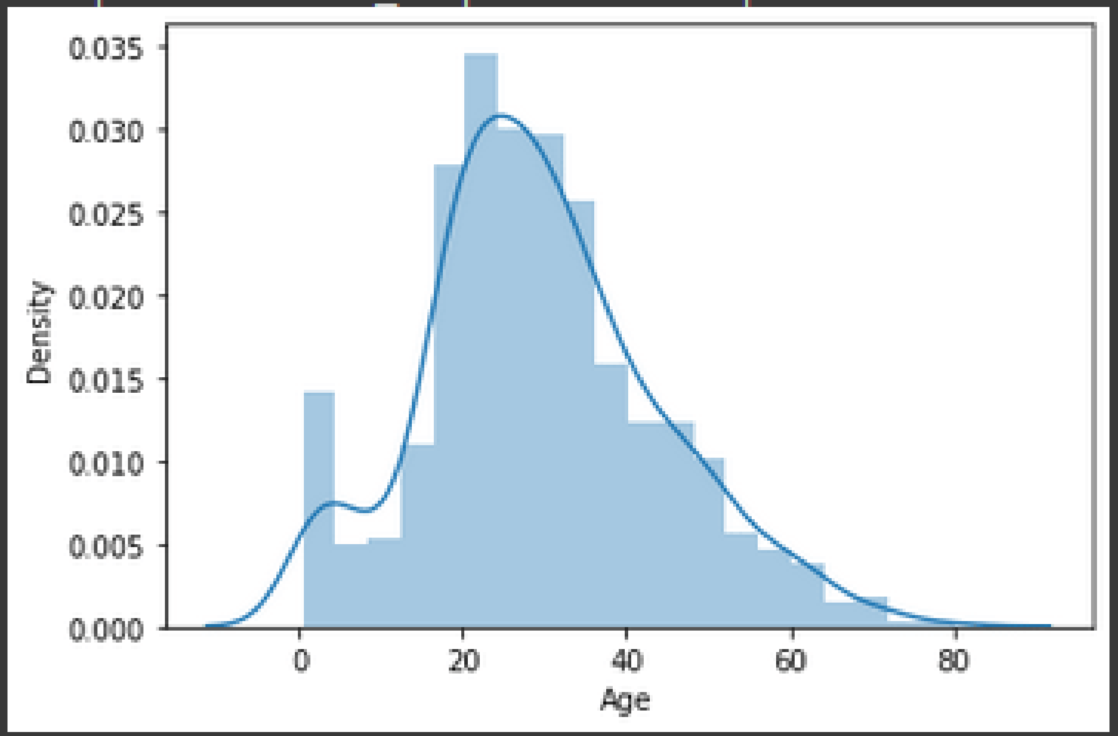
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 891 entries, 1 to 891
Data columns (total 11 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Survived  891 non-null    int64
 1   Pclass    891 non-null    int64
 2   Name      891 non-null    object
 3   Sex       891 non-null    object
 4   Age       714 non-null    float64
 5   SibSp     891 non-null    int64
 6   Parch     891 non-null    int64
 7   Ticket    891 non-null    object
 8   Fare      891 non-null    float64
 9   Cabin     204 non-null    object
 10  Embarked  891 non-null    int64
dtypes: float64(2), int64(5), object(4)
```

Age Column

Age Colum has Skewness Distribution, because of that we can use median for imputaion missing data

```python
import seaborn as sns
sns.distplot(df["Age"])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distribu
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f5c5ff77a
```



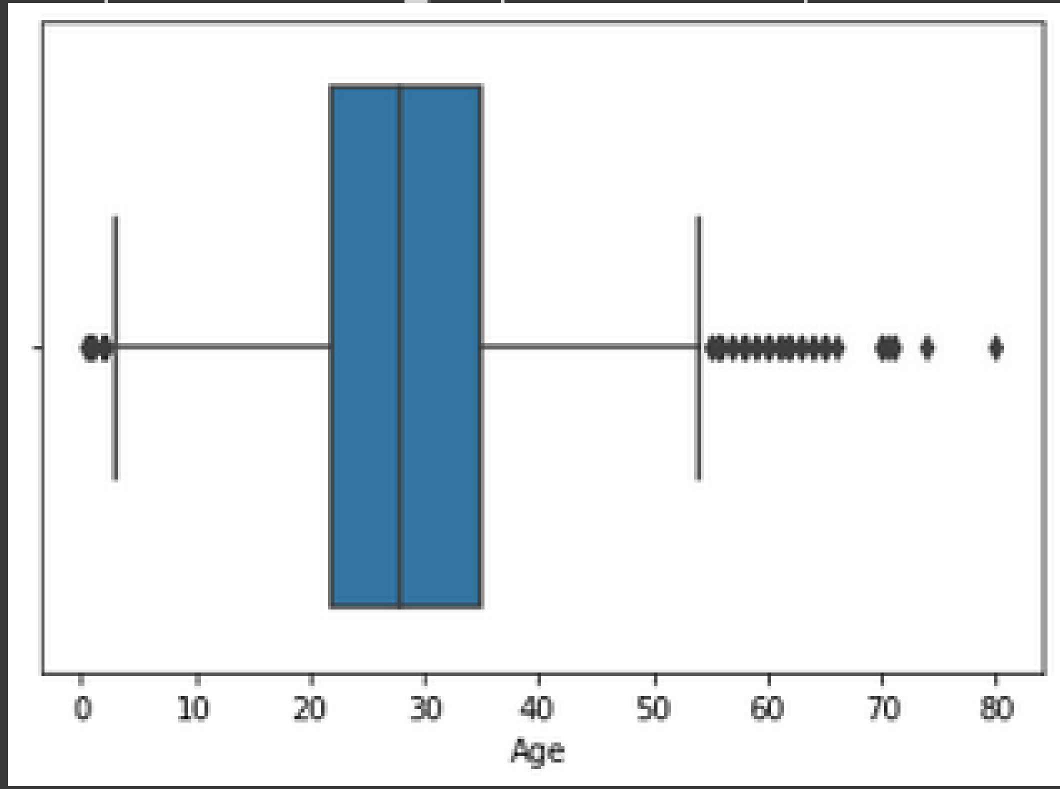

```python
val = df.Age.median()
df["Age"] = df.Age.fillna(val)
```

Age Column

Visualitation data Age Column

```
sns.boxplot(df["Age"])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_de
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7fe372
```



in this plot we can see the outliers, in this case the outliers are passengers aged in range (0, +-5) and more than +- 55

Sex Column

Data in Sex column only has two unique data, that is male and female . we want to convert data object to data numerical

0 for male and 1 for female

```
df.Sex.map({'male':0, 'female':1})
```

```
PassengerId
1      0
2      1
3      1
4      1
5      0
      ..
887    0
888    1
889    1
890    0
891    0
Name: Sex, Length: 891, dtype: int64
```

Drop Data

Drop data is used when the existing data is uninformative and has a lot of unique data., in this case we use drop data in Cabin Column, Name Column and Ticket Column

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 891 entries, 1 to 891
Data columns (total 11 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Survived  891 non-null    int64
 1   Pclass    891 non-null    int64
 2   Name      891 non-null    object
 3   Sex       891 non-null    object
 4   Age       714 non-null    float64
 5   SibSp     891 non-null    int64
 6   Parch     891 non-null    int64
 7   Ticket    891 non-null    object
 8   Fare      891 non-null    float64
 9   Cabin     204 non-null    object
 10  Embarked  891 non-null    int64
```

```
df.drop("Cabin", axis = 1, inplace = True)
df.drop("Name", axis = 1, inplace = True)
df.drop('Ticket', axis = 1, inplace = True)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 891 entries, 1 to 891
Data columns (total 8 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Survived  891 non-null    int64
 1   Pclass    891 non-null    int64
 2   Sex       891 non-null    int64
 3   Age       891 non-null    float64
 4   SibSp     891 non-null    int64
 5   Parch     891 non-null    int64
 6   Fare      891 non-null    float64
 7   Embarked  891 non-null    int64
dtypes: float64(2), int64(6)
memory usage: 62.6 KB
```

Data Survived Visualitation

Proportion of Survived data

```
df.Survived.value_counts()
```

```
0    549
1    342
Name: Survived, dtype: int64
```

Data Survived Visualization

Making data frame from Survived data

```
df_survived = pd.DataFrame(df.Survived.value_counts())
```

```
df_survived['Status'] = ['dies','alive']
```

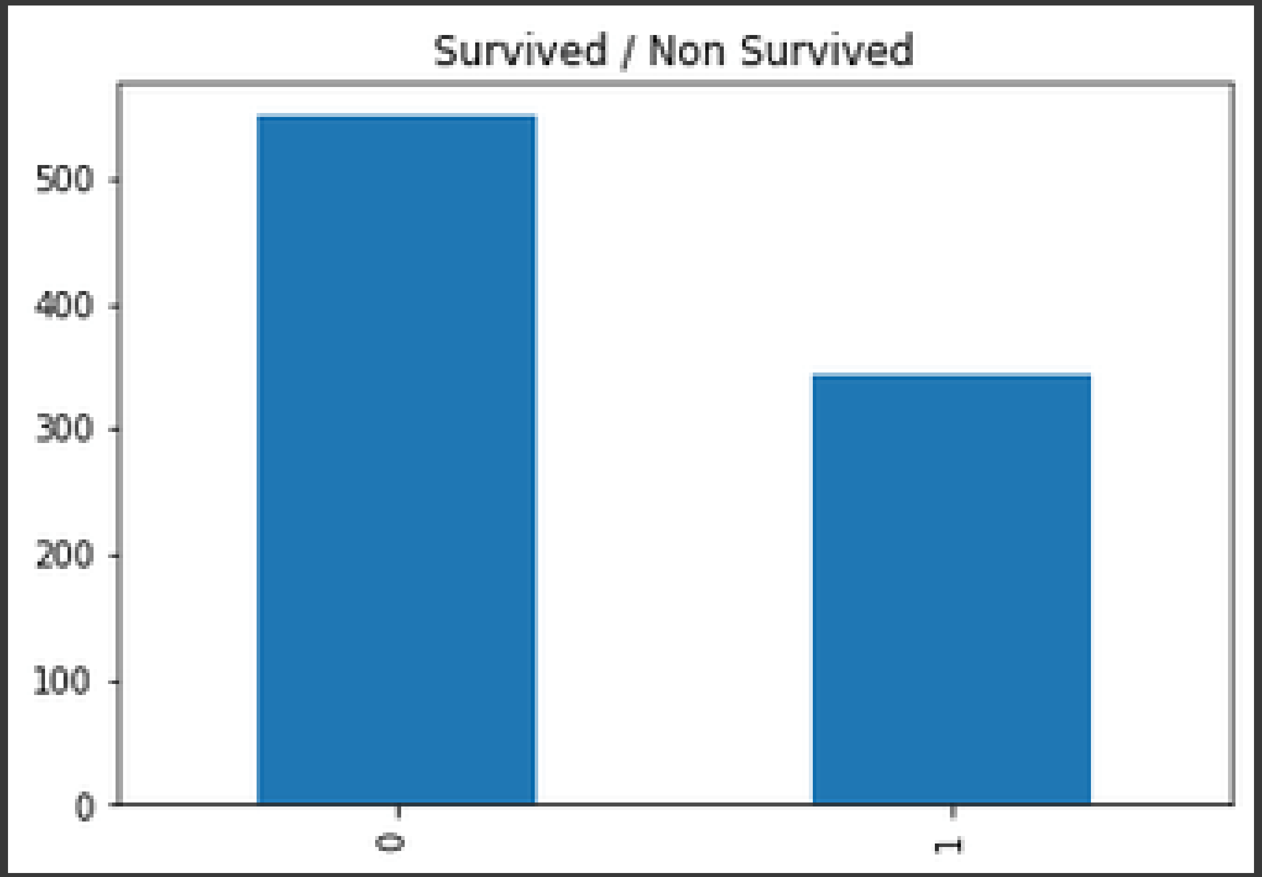
```
df_survived
```

	Survived	Status
0	549	dies
1	342	alive

Data Survived Visualization

Show chart using matplotlib.pyplot module

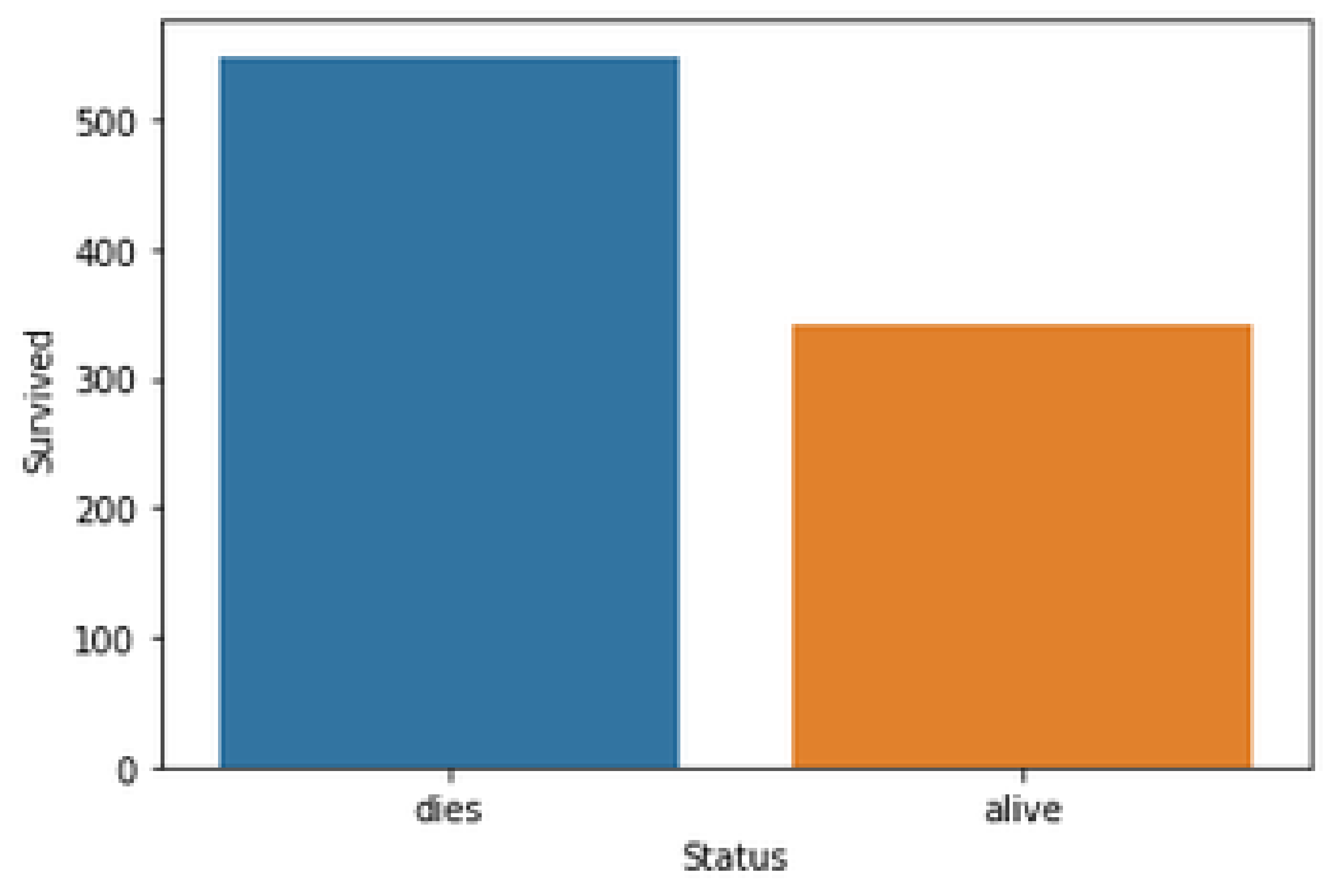
```
df.Survived.value_counts().plot(kind = "bar");
plt.title("Survived / Non Survived");
```



Data Survived Visualization

Show chart using seaborn module

```
sns.barplot(x = 'Status', y = 'Survived', data = df_survived);
```



Thank You

Seaborn Kusto