Data Preprocessing

Agenda:

0	Data Pre-processing
1	Data Understanding
2	Check for Datatypes
3	Handle Null Values
4	Handle Outliers
5	Visualization
6	Remove Duplicates
7	Data Splitting
8	Normalization
9	Encoding

0. What is Data Pre-processing

What is Data Pre-processing:

- Data Pre-processing means applying different operations over the data to make it clean, organized, and consistent.
- Data can have many problems; such as, Non-Informative data, In-consistent data, duplicated data, etc.
- Data Pre-processing is about applying operations over the data to solve these problems.
- In this chapter, we will go through each of these problems and see how to solve it. Where we will apply all these concepts, in practice, on a dataset called Titanic Dataset.

Titanic Dataset:

- Titanic dataset is a data collected, about the individuals who were aboard the Titanic on its first and last voyage.
- This dataset was collected to study the behavior followed in rescuing people, to understand the reasons behind the survival of some categories of people and drowning of others.

Read Titanic Dataset:

- The first step in any project, is to read the csv/xlsx file that represents the Titanic dataset.
- As you can see the dataset has 12 rows (samples), and 891 columns (attributes).

```python
df = pd.read_csv("train.csv")
df
```

	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
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

1. Data Understanding

Data Understanding:

- Data Understanding means studying and understanding the data, including knowing:
  - What is the task or the target of the data.
  - How much each column is important to the target.
- For example, in Titanic dataset:
  - Our task is to know if a person survived or died, which means that the target is the column called “Survived”.
  - Columns like “PassengerId”, “Name”, “Ticket”, are not important to the target, so we will drop these columns.
- Note that all the columns except for the target columns are called features.

Drop Un-necessary Columns:

```
df.drop(["PassengerId", "Name", "Ticket"], axis=1, inplace = True)
df.head()
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked
0	0	3	male	22.0	1	0	7.2500	NaN	S
1	1	1	female	38.0	1	0	71.2833	C85	C
2	1	3	female	26.0	0	0	7.9250	NaN	S
3	1	1	female	35.0	1	0	53.1000	C123	S
4	0	3	male	35.0	0	0	8.0500	NaN	S

2. Check for Datatypes

Check for Dtypes:

- Sometimes there are mistakes in representing the columns datatypes, for example:
  - A numerical column could be represented as categorical, or a categorical column represented as object datatype, and so on.
- It's our responsibility to check for such mistakes and correct them.

- For example, in Titanic dataset:
  - Columns "Survived", "Pclass", "SibSp", and "Parch" are int dtypes with small number of possible unique values, so we will change them to be categorical.
  - Columns "Sex", and "Embarked" are string columns so we will change them to be categorical.

Change In-correct Datatypes:

- **Display Datatypes**

```
dtypes = df.dtypes
n_uniq = df.nunique()
pd.DataFrame({"Dtypes": dtypes, "Num_Uniqe": n_uniq}).T
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked
Dtypes	int64	int64	object	float64	int64	int64	float64	object	object
Num_Uniqe	2	3	2	88	7	7	248	147	3

- **Change In-correct Datatypes**

```
cols = ["Pclass", "SibSp", "Parch", "Sex", "Embarked", "Survived"]
df[cols] = df[cols].astype('category')
pd.DataFrame(df.dtypes).T
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked
0	category	category	category	float64	category	category	float64	object	category

3. Handle Null Values

Null Values:

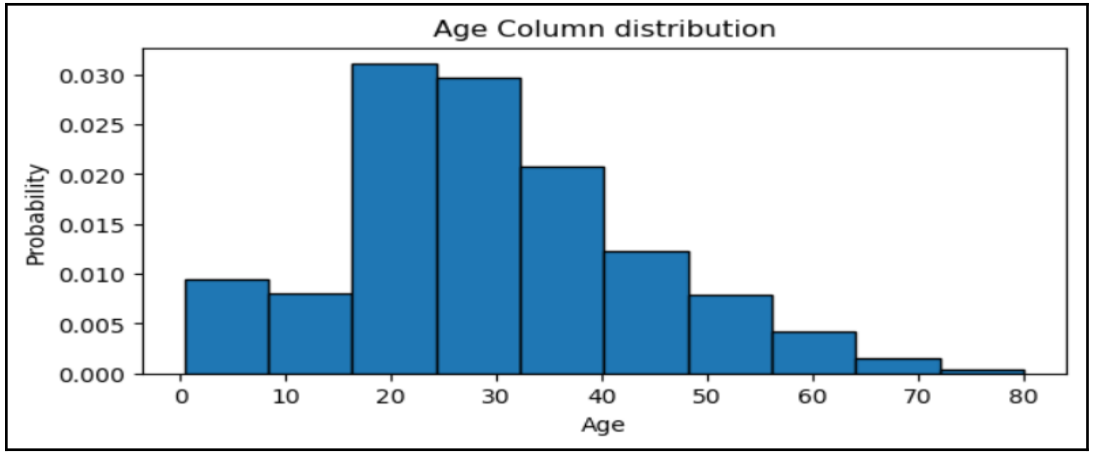
- Null Values must be handled, so that the data becomes complete and ready for machine learning.
- There are 3 options to handle Null values, that depend on the Null Values amount in each column:
  - If the column contains a small number of Null Values, you can simply delete rows that contain null values in this column.
  - If the column contains a huge number of Null Values, you should delete this column.
  - If the column contains a large number of Null Values, but not very large, you can replace null values with mean, median, or Mode.

Replace Null Values:

- In the 3rd option:
  - We replace Null Values with Mode if the column is categorical.
  - We replace Null Values with Mean if the column is numerical & normally distributed.
  - We replace Null Values with Median if the column numerical & not-normally distributed.

Check for Null Values:

- In Titanic dataset:
  - Embarked has only two null values, So we can just drop them.
  - Cabin has about 77% null values, So we will drop this column.
  - Age has about 20% null values, So we will replace null values with the median since Age is not-normally distributed (right skewed).



Handle Null Values:

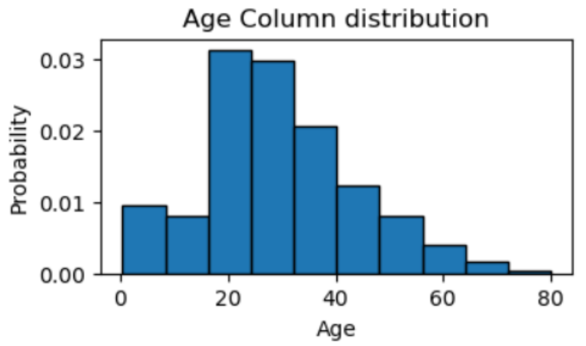
Check for Null Values

```python
null = df.isnull().sum()
ratio = null / df.shape[0]
pd.DataFrame({"Null_sum": null, "Ratio": ratio}).T
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked
Null_sum	0.0	0.0	0.0	177.000000	0.0	0.0	0.0	687.000000	2.000000
Ratio	0.0	0.0	0.0	0.198653	0.0	0.0	0.0	0.771044	0.002245

Replace Null Values in Age Column

```python
plt.figure(figsize=(4, 2))
plt.hist(df['Age'], density=True, edgecolor="black")
plt.title("Age Column distribution")
plt.xlabel("Age")
plt.ylabel("Probability")
plt.show()
```




```python
median = df["Age"].median()
df["Age"].fillna(median, inplace=True)
```

Drop Null Values in Embarked Column

```python
df = df.dropna(subset=['Embarked'])
```

Drop Cabin Column

```python
df = df.drop("Cabin", axis=1)
```

Make sure Null Values are removed

```python
pd.DataFrame(df.isnull().sum()).T
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	0	0	0	0	0	0	0

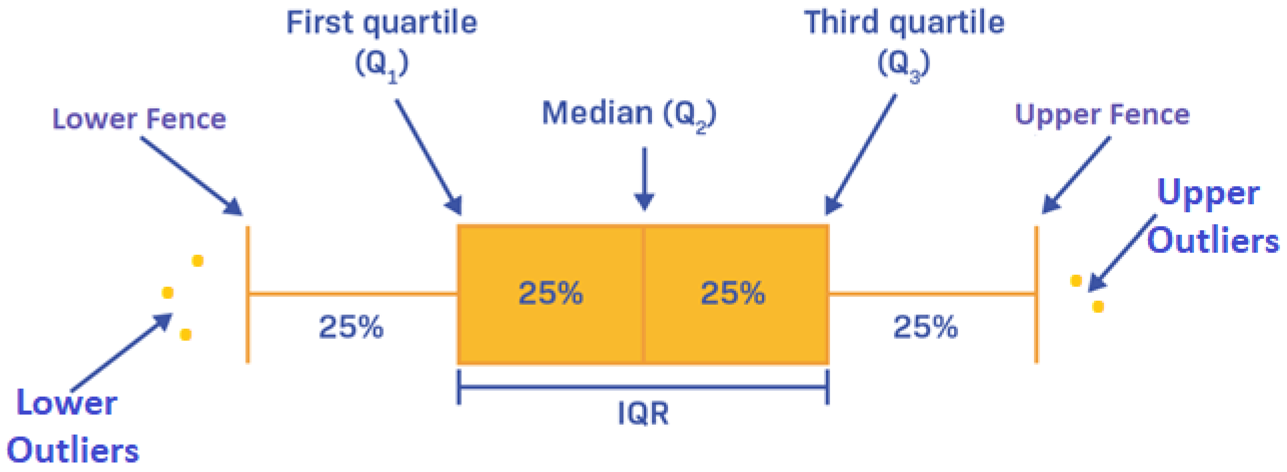
4. Handle Outliers

Outliers:

- Outliers are values within the data, that has extremely large or extremely small values, for example, people ages usually cannot exceed 100 years, if you find a 150 years-old man then this one is considered to be an outlier.
- Sometimes we need to remove outliers because they could cause many problems; such as, introducing misleading information about the data, for example there could 150 years-old man in real-life, but this is a very special case that doesn't represent the normal human ages.
- Also some statistical measures could be sensitive to outliers; such as mean, and this is another reason for why we would like to remove outliers.

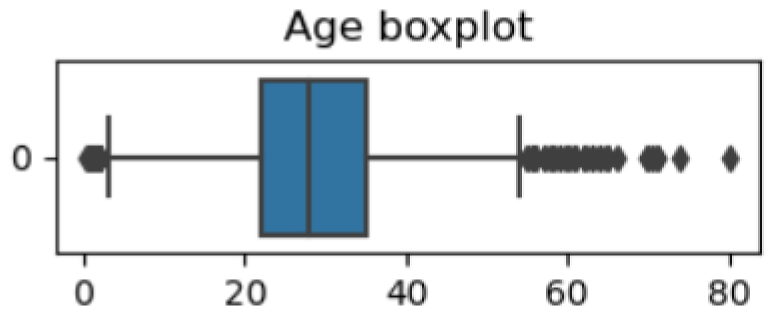
Check for Outliers:

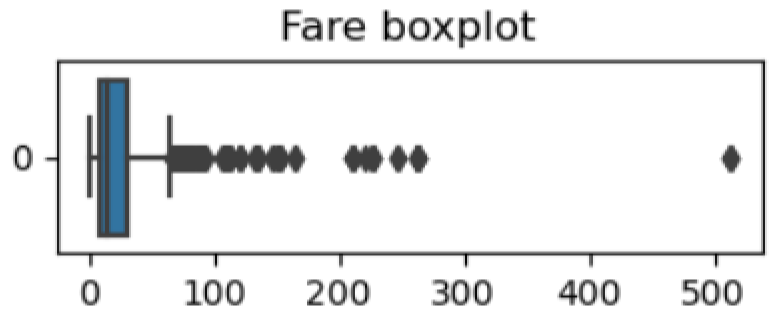
- To check for outliers we Quartiles, where:
  - All values, greater than the Upper-Fence, are considered to be outliers.
  - All values, smaller than the Lower-Fence, are considered to be outliers.
- Also, there is a graph called boxplot, which is useful for quickly checking the outliers.



Handle Outliers:

- There are two options to handle outliers:
  - The 1st option is to treat outliers as if they were Null Values.
  - The 2nd option is to replace Upper-Outliers with Upper-Fence and replace Lower-Outliers with Lower-Fence.
- We Usually tend to do the Second option.
- In Titanic dataset, there are two columns that contains outliers; "Age", and "Fare".

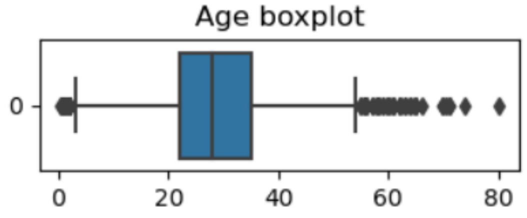


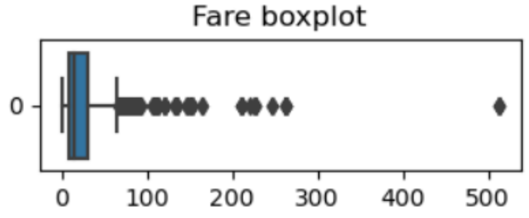


Handle Outliers:

1- Check for Outliers

```python
num_cols = df.select_dtypes("number").columns
plt.figure(figsize=(8, 1))
for i, col in enumerate(num_cols):
    plt.subplot(1, 2, i+1)
    sns.boxplot(df[col], orient="h")
    plt.title(f"{col} boxplot")
```



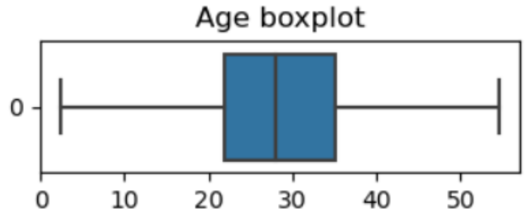


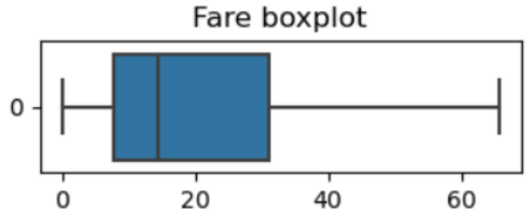
2- Remove Outliers

```python
for col in num_cols:
    Q1 = df[col].quantile(.25)
    Q3 = df[col].quantile(.75)
    IQR = Q3 - Q1
    Lower_Fence = Q1 - 1.5 * IQR
    Upper_Fence = Q3 + 1.5 * IQR
    Lower_Outliers = df[df[col] < Lower_Fence][col].values
    Upper_Outliers = df[df[col] > Upper_Fence][col].values
    df[col].replace(Lower_Outliers, Lower_Fence, inplace=True)
    df[col].replace(Upper_Outliers, Upper_Fence, inplace=True)
```

3- Make Sure Outliers are Removed

```python
num_cols = df.select_dtypes("number").columns
plt.figure(figsize=(8, 1))
for i, col in enumerate(num_cols):
    plt.subplot(1, 2, i+1)
    sns.boxplot(df[col], orient="h")
    plt.title(f"{col} boxplot")
```





5. Visualization

Visualization:

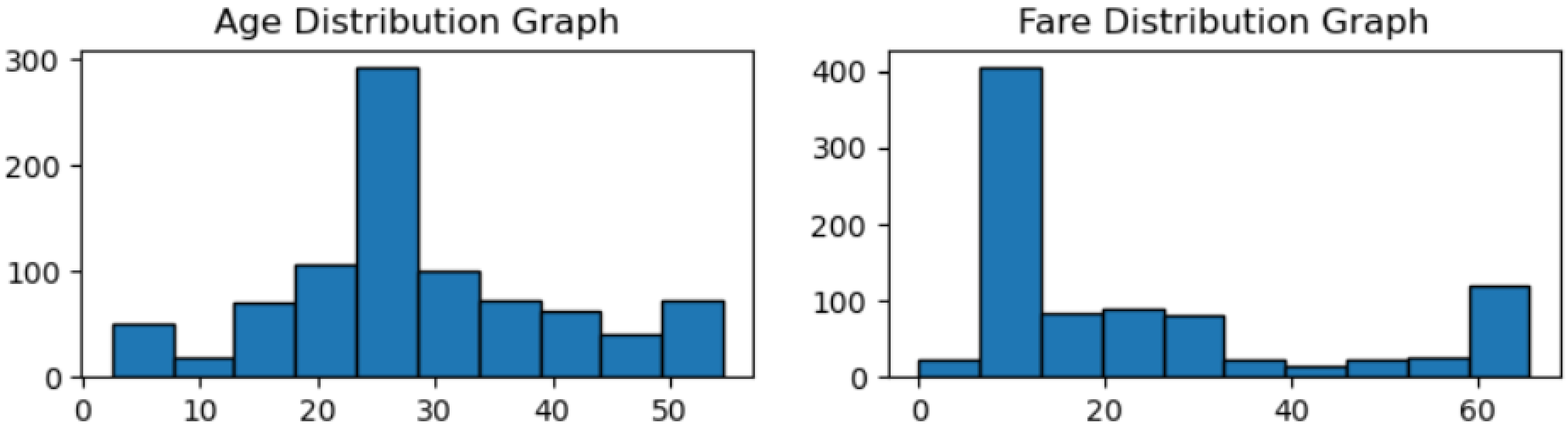
- Visualization is the process of creating graphs about our data, that helps use well understand & explore our data.
- Types of Graphs:
  A. Data Distribution Graphs.
  B. Outlier Detection Graphs.
  C. Relationship Graphs.
- The most famous libraries used for visualization are Matplotlib & Seaborn.

A. Data Distribution Graphs:

- Numerical Data Distribution Graphs:
  - Histogram.
  - KDE Plot.
- Categorical Data Distribution Graphs:
  - Count Plot.
  - Pie Plot.

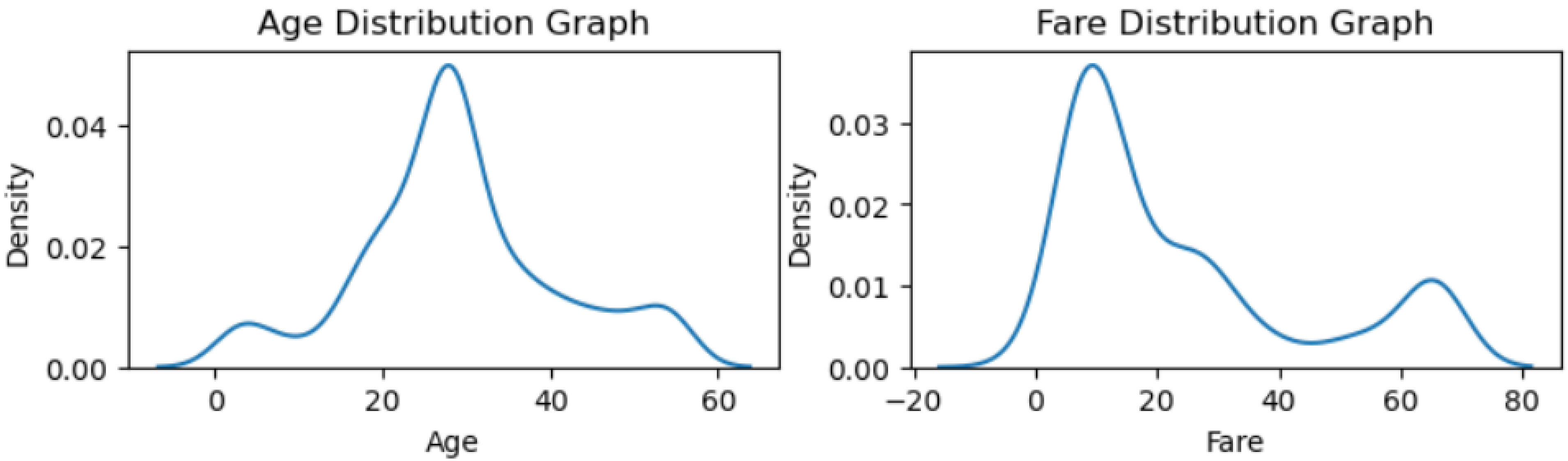
Numerical Data Distribution (Histogram):

```
num_cols = df.select_dtypes("number").columns
plt.figure(figsize=(9, 2))
for i, col in enumerate(num_cols):
    plt.subplot(1, 2, i+1)
    plt.hist(df[col], edgecolor="black")
    plt.title(f"{col} Distribution Graph")
plt.show()
```



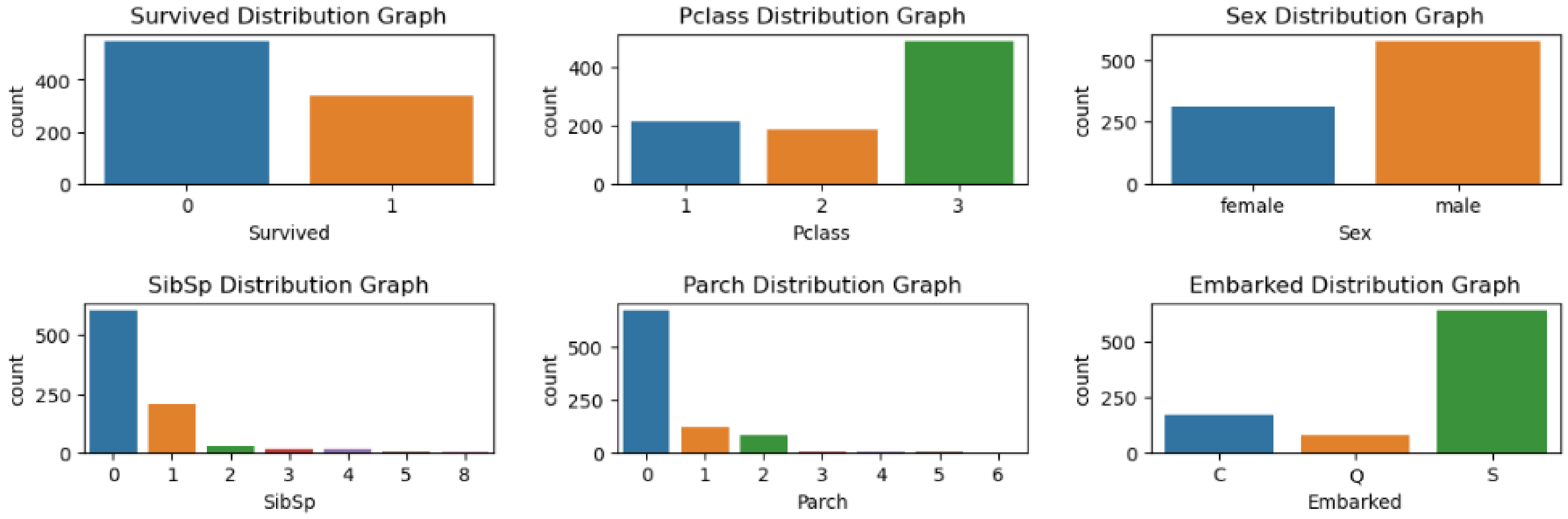
Numerical Data Distribution (Kde Plot):

```
num_cols = df.select_dtypes("number").columns
plt.figure(figsize=(9, 2))
for i, col in enumerate(num_cols):
    plt.subplot(1, 2, i+1)
    sns.kdeplot(df[col])
    plt.title(f"{col} Distribution Graph")
plt.show()
```



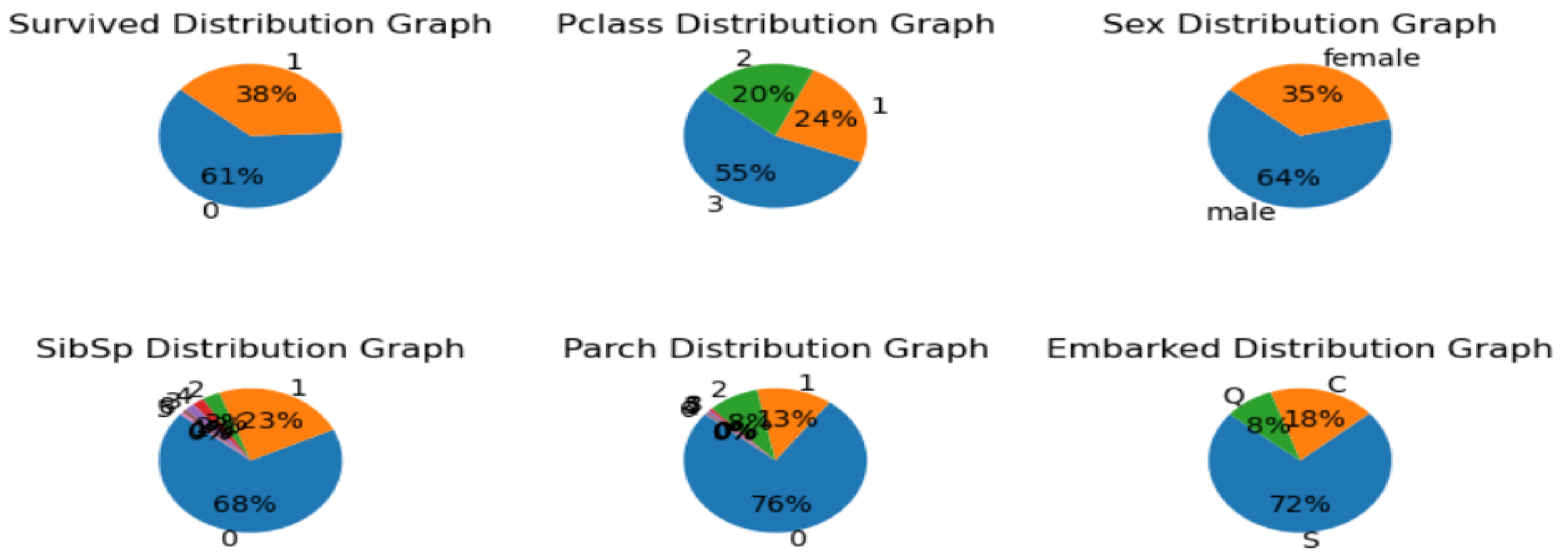
Categorical Data Distribution (Count Plot):

```python
cat_cols = df.select_dtypes("category").columns
plt.figure(figsize=(14, 4))
for i, col in enumerate(cat_cols):
    plt.subplot(2, 3, i+1)
    sns.countplot(x=col, data=df)
    plt.title(f"{col} Distribution Graph")
plt.subplots_adjust(hspace=.8, wspace=.3)
plt.show()
```



Categorical Data Distribution (Pie Plot):

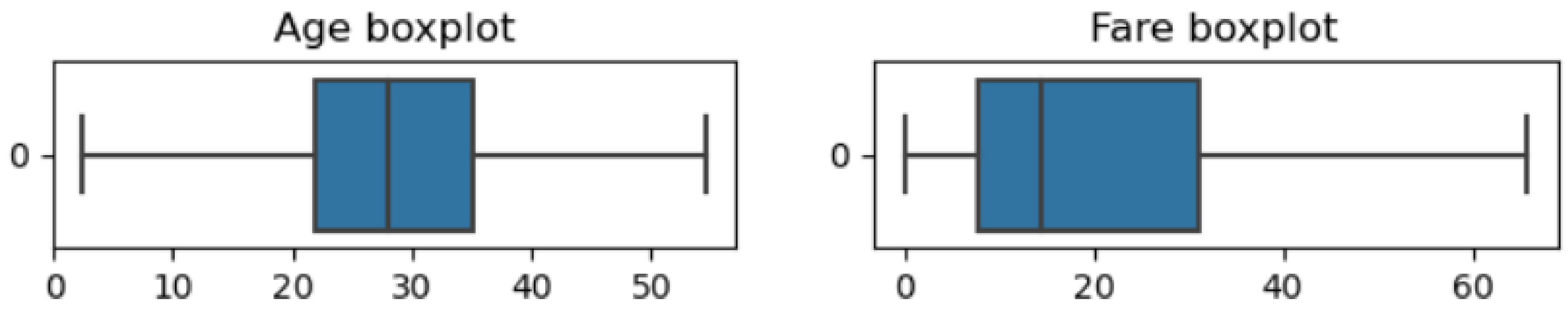
```python
cat_cols = df.select_dtypes("category").columns
plt.figure(figsize=(9, 4))
for i, col in enumerate(cat_cols):
    plt.subplot(2, 3, i+1)
    unique = df[col].value_counts()
    count = unique.values
    categories = unique.index
    plt.pie(count, labels = categories, startangle=140, autopct='%1.1d%%')
    plt.title(f"{col} Distribution Graph")
plt.subplots_adjust(hspace=.8, wspace=.3)
plt.show()
```



B. Outlier Detection Graphs:

- The easiest way to check for outliers, is through displaying a graph called box plot.

```python
num_cols = df.select_dtypes("number").columns
plt.figure(figsize=(8, 1))
for i, col in enumerate(num_cols):
    plt.subplot(1, 2, i+1)
    sns.boxplot(df[col], orient="h")
    plt.title(f"{col} boxplot")
```



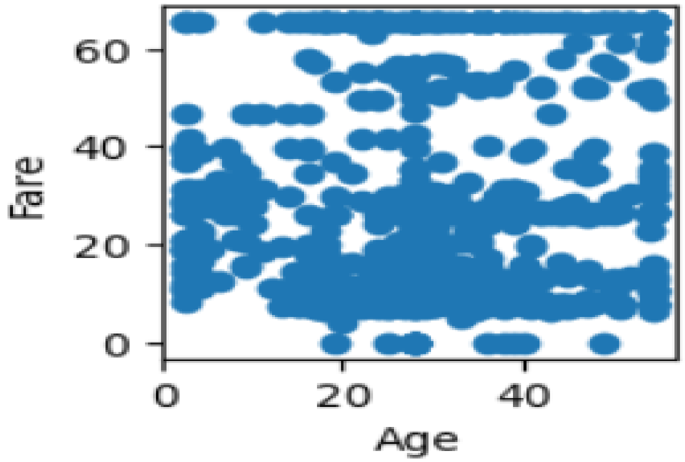
C. Relationship Graphs:

- The idea of this type of graphs is to visualize the relationship between two columns, usually between each feature & the target.
- The dtypes of the two columns defines which graph to use, for example:
  - If the two columns are numerical, we could use scatter plot, pair plot, line plot, or heat map.
  - If we have one numerical & one categorical, then we use bar plot.
  - If the two columns are categorical, then we use heat map.

Numerical/Numerical Relationship (Scatter Plot):

- Get the cartesian coordinates for all the data points in 2d space, where the two dimensions are the two columns.

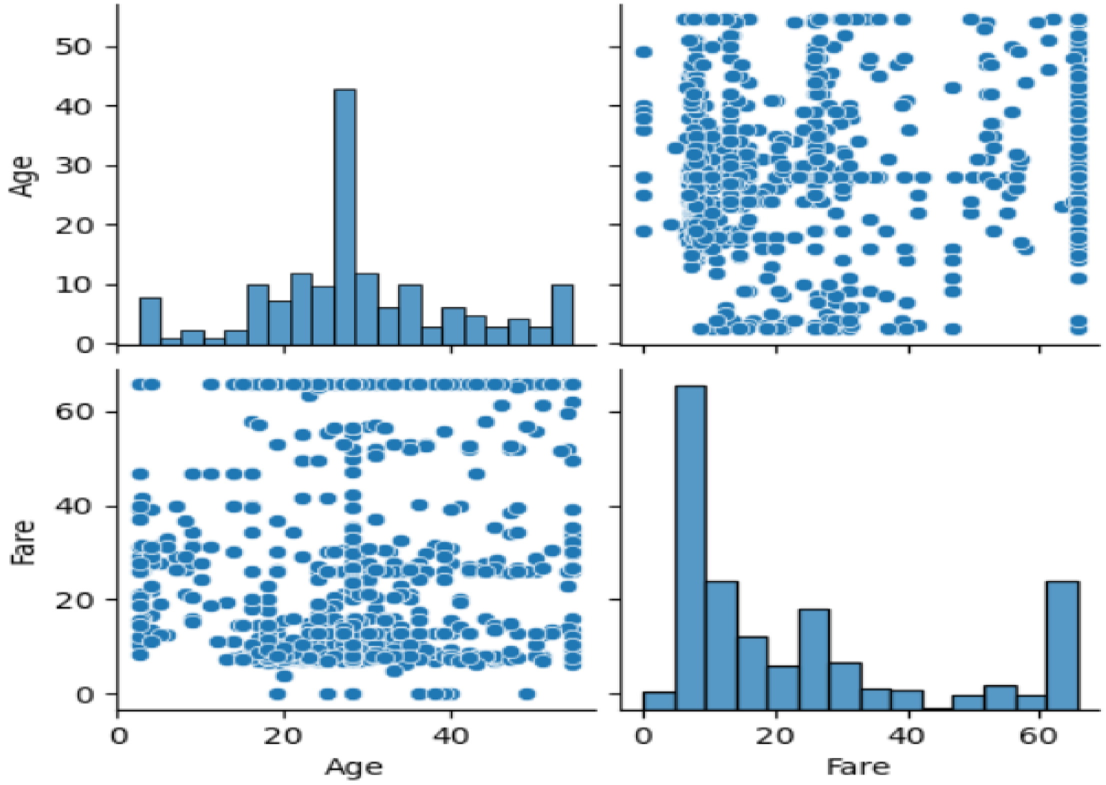
```python
plt.figure(figsize=(2, 2))
plt.scatter(df["Age"], df["Fare"])
plt.xlabel("Age")
plt.ylabel("Fare")
plt.show()
```



Numerical/Numerical Relationship (Pair Plot):

- Is a graph that gets scatter plot between each two numerical columns.

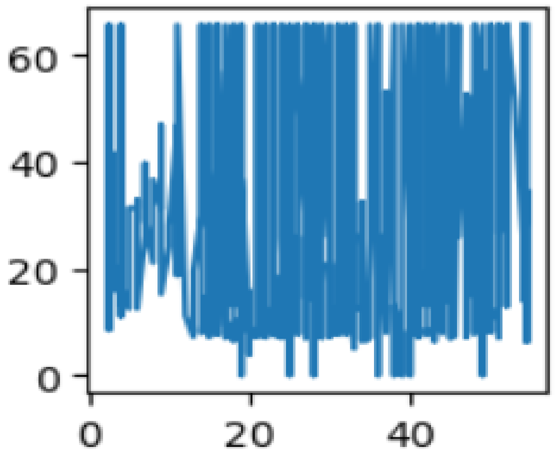
```
sns.pairplot(df)
```



Numerical/Numerical Relationship (Line Plot):

- Is the same as scatter plot, but it connect the points with each other with lines, so the data should be sorted first.

```
sorted_df = df.sort_values(by="Age")
plt.figure(figsize=(2, 2))
plt.plot(sorted_df["Age"], sorted_df["Fare"])
plt.show()
```



Numerical/Numerical Relationship (Heat Map):

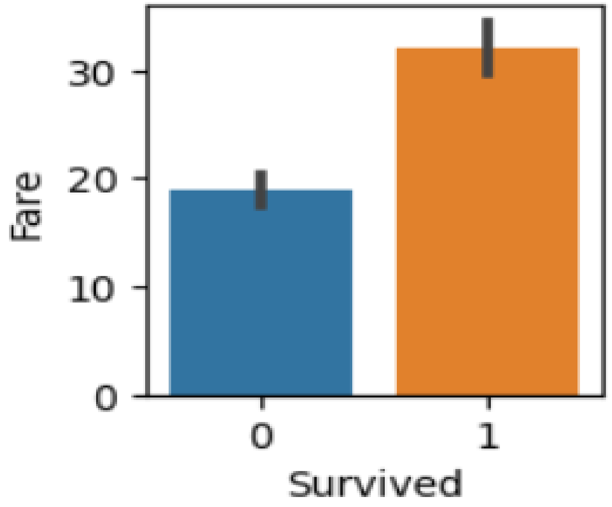
- Is used to show how high values in a 2D-matrix are, this is useful if you want to visualize the correlation matrix of your data.

```
corr = df.corr()
plt.figure(figsize=(2, 2))
sns.heatmap(corr, annot=True)
plt.show()
```

Numerical/Categorical Relationship (Bar Plot):

- Numerical columns values are aggregated based on the unique values in the categorical column.

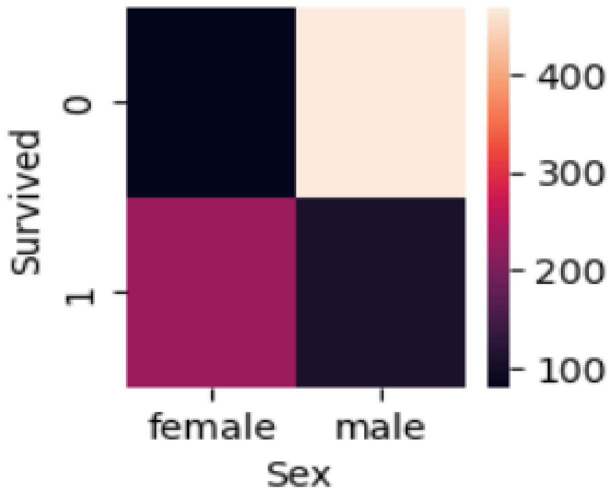
```
plt.figure(figsize=(2, 2))
sns.barplot(x="Survived", y="Fare", data=df)
plt.show()
```



Categorical /Categorical Relationship (Heat Map):

- We first calculate the frequency of each possible pair of unique values form the two columns, then we display the result as a heat map.

```
plt.figure(figsize=(2, 2))
agg = df.pivot_table(index="Survived", columns="Sex", values="Age", aggfunc=len)
sns.heatmap(agg)
plt.show()
```



6. Remove Duplicates

Duplicates:

- Duplicates refers to the rows of the dataset that is repeated.
- It's preferred to remove these rows because they don't add value to data, which means that they don't introduce new information, which is considered to be waste of memory & resources.
- This also could increase the computation time.
- In Titanic dataset, there are about 129 duplicated rows.

Remove Duplicates:

- **Check for Duplicates**

```
df.duplicated().sum()
```

```
129
```

- **Remove Duplicates**

```
df.drop_duplicates(inplace=True)
```

- **Make Sure that Duplicates are Removed**

```
df.duplicated().sum()
```

```
0
```

7. Data Splitting

Data Splitting:

- Data splitting means dividing the columns of the dataset, into Features & a Target.
- The Target is the column we are most interested to study, while the Features are the columns that helps us understand more about the Target.
- Usually the features are called “X”, while the Target is called “y”.
- In Titanic dataset, the target is the “Survived” column, while the features are the other columns.

Split the Data:

```
X = df.drop("Survived", axis=1)
y = df[["Survived"]]
```

```
X
```

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	3	male	22.0	1	0	7.2500	S
1	1	female	38.0	1	0	65.6563	C
2	3	female	26.0	0	0	7.9250	S
3	1	female	35.0	1	0	53.1000	S
4	3	male	35.0	0	0	8.0500	S
...	...	...	...	...	...	...	...
885	3	female	39.0	0	5	29.1250	Q
887	1	female	19.0	0	0	30.0000	S
888	3	female	28.0	1	2	23.4500	S
889	1	male	26.0	0	0	30.0000	C
890	3	male	32.0	0	0	7.7500	Q

760 rows × 7 columns

```
y
```

	Survived
0	0
1	1
2	1
3	1
4	0
...	...
885	0
887	1
888	0
889	1
890	0

760 rows × 1 columns

8. Normalization

Normalization:

- Normalization is transforming the data so that all the numerical columns have the same scale, that's why normalization is also called Scaling.
- This scale is usually between 0 & 1, by applying a normalization technique called MinMax Scaler.
- Steps to calculate MinMax Scaler for each column:
  1. The 1st step is called fit:
     - Calculate the Min & Max values of the column.
  2. The 2nd step is called transform:
     - The new values of the columns are calculated using this formula: (X - Min) / (Max - Min), where X refers to the column's values.

Apply Normalization:

```python
from sklearn.preprocessing import MinMaxScaler
num_cols = X.select_dtypes("number").columns
scaler = MinMaxScaler()
scaler.fit(X[num_cols])
X[num_cols] = scaler.transform(X[num_cols])
```

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	3	male	0.375000	1	0	0.110424	S
1	1	female	0.682692	1	0	1.000000	C
2	3	female	0.451923	0	0	0.120704	S
3	1	female	0.625000	1	0	0.808757	S
4	3	male	0.625000	0	0	0.122608	S
...	...	...	...	...	...	...	...
885	3	female	0.701923	0	5	0.443598	Q
887	1	female	0.317308	0	0	0.456925	S
888	3	female	0.490385	1	2	0.357163	S
889	1	male	0.451923	0	0	0.456925	C
890	3	male	0.567308	0	0	0.118039	Q

760 rows × 7 columns

9. Encoding

Encoding:

- Encoding means representing the string values as numbers so that the machine can understand them, where computers can only apply mathematical operations over numbers.
- There are three main Encoding techniques to use, but to decide which one to choose, we divide string values into 2 types:
  - Nominal, where order of the unique values doesn't matter, for example, in shoes colour "Red" is not greater or less than "Yellow".
  - Ordinal, where order matters, for example, in shoes size "Large" is greater than "Medium".

Encoding Techniques:

- Encoding Techniques are:
  1. **Ordinal Encoding**:
     - Used for ordinal columns.
  2. **One Hot Encoding**:
     - Used for nominal columns with small number of unique values.
  3. **Binary Encoding**:
     - Used for nominal columns with large number of unique values.
- In Titanic dataset, Sex & Embarked are both nominal so we will apply One Hot Encoding.

Apply Encoding:

```
from category_encoders import OneHotEncoder
encoder = OneHotEncoder(cols = str_cols, drop_invariant=True)
X = encoder.fit_transform(X)
```

	Pclass	Sex_1	Sex_2	Age	SibSp	Parch	Fare	Embarked_1	Embarked_2	Embarked_3
0	3	1	0	0.375000	1	0	0.110424	1	0	0
1	1	0	1	0.682692	1	0	1.000000	0	1	0
2	3	0	1	0.451923	0	0	0.120704	1	0	0
3	1	0	1	0.625000	1	0	0.808757	1	0	0
4	3	1	0	0.625000	0	0	0.122608	1	0	0
...	...	...	...	...	...	...	...	...	...	...
885	3	0	1	0.701923	0	5	0.443598	0	0	1
887	1	0	1	0.317308	0	0	0.456925	1	0	0
888	3	0	1	0.490385	1	2	0.357163	1	0	0
889	1	1	0	0.451923	0	0	0.456925	0	1	0
890	3	1	0	0.567308	0	0	0.118039	0	0	1

760 rows × 10 columns

Thank You