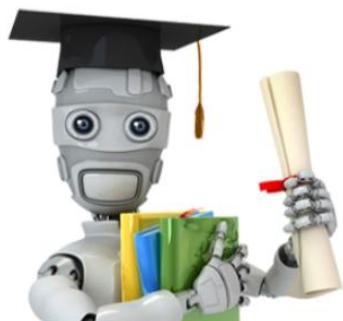


Advanced Machine learning Mastering Course

Introduced by

George Samuel

Master in computer science
Cairo University



Innovisionray.com

2024

Advanced Machine Learning 2024 by
George Samuel

Machine Learning Diploma

Session 3: Data Pre-processing

Agenda:

0	Data Pre-processing
1	Data Understanding
2	Check for Datatypes
3	Handle Null Values
4	Handle Outliers
5	Remove Duplicates
6	Visualization
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**, **Inconsistent 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.



Data Pre-processing

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).

```
1 df = pd.read_csv("train.csv")
2 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... Heikkinen, Miss. Laina	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Allen, Mr. William Henry	male	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Montvila, Rev. Juozas	male	35.0	0	0	373450	8.0500	NaN	S
...
886	887	0	2	Graham, Miss. Margaret Edith	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Johnston, Miss. Catherine Helen "Carrie"	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Behr, Mr. Karl Howell	male	Nan	1	2	W.C. 6607	23.4500	NaN	C
889	890	1	1	Dooley, Mr. Patrick	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3		male	32.0	0	0	370376	7.7500	NaN	Q

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:

```
1 df.drop(["PassengerId", "Name", "Ticket"], axis=1, inplace = True)  
2 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.

Data Pre-processing

Change In-correct Datatypes:

- Display Datatypes

```
1 dtypes = df.dtypes
2 n_uniq = df.nunique()
3 pd.DataFrame({"Dtypes": dtypes, "Num_Unique": n_uniq}).T
```

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

- Change In-correct Datatypes

```
1 cols = ["Pclass", "SibSp", "Parch", "Sex", "Embarked", "Survived"]
2 df[cols] = df[cols].astype('category')
3 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:

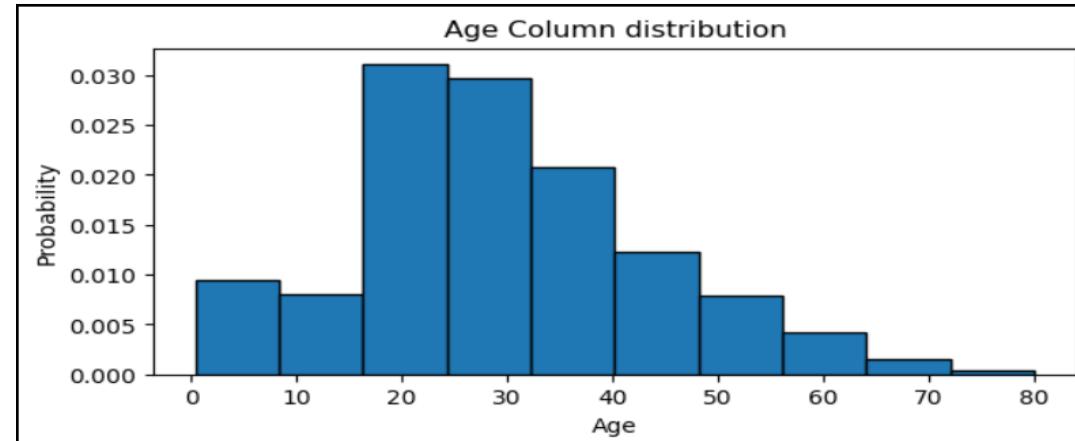
- Null Values must be handled, so that the data becomes complete and ready for machine learning.
- There are 3 options to handle Null values, that depend on the Null Values amount in each column:
 - If the column contains a small number of Null Values, you can simply delete rows that contain null values 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 \(right skewed\)](#).



Data Pre-processing

Handle Null Values:

Check for Null Values

```
1 null = df.isnull().sum()  
2 ratio = null / df.shape[0]  
3 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

Drop Null Values in Embarked Column

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

Drop Cabin Column

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

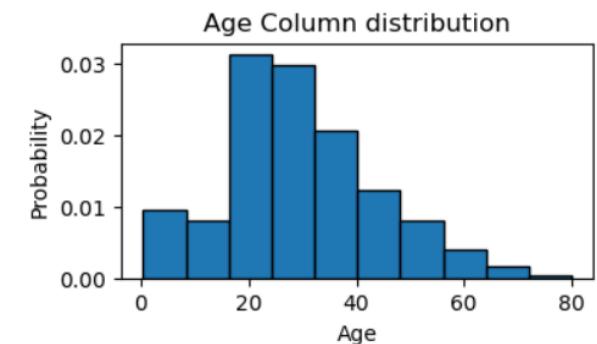
Make sure Null Values are removed

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

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

Replace Null Values in Age Column

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



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

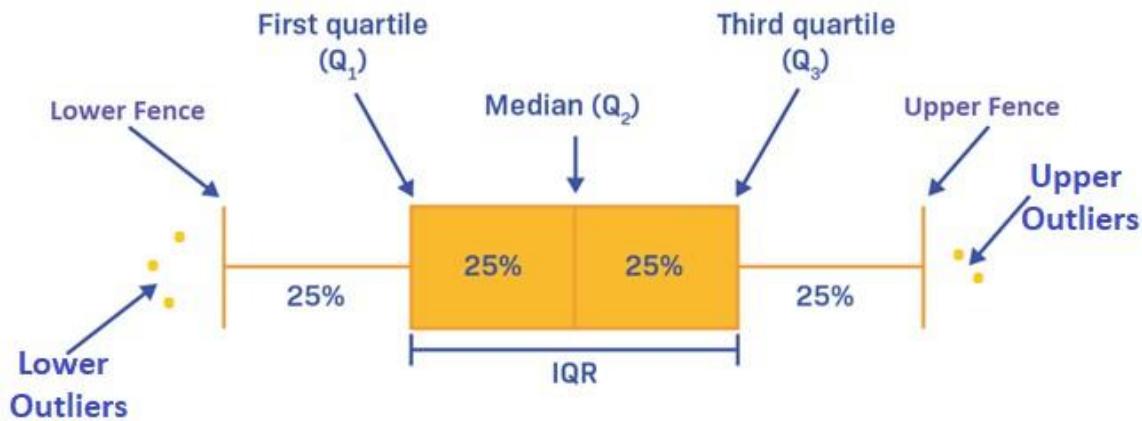
4. Handle Outliers

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.

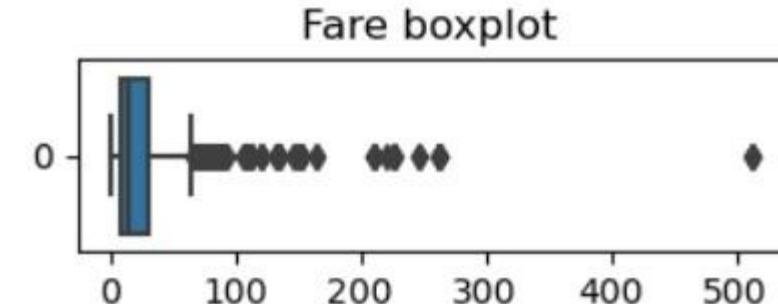
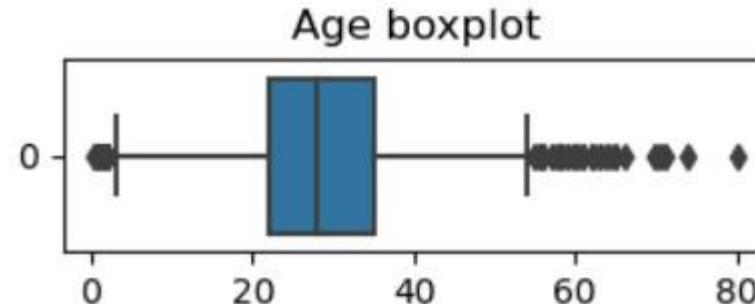
Check for Outliers:

- To check for outliers we use 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”.

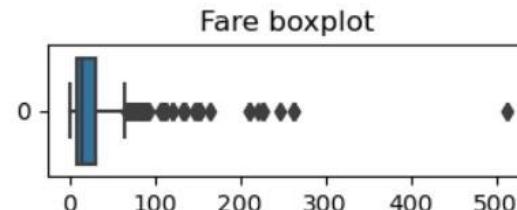
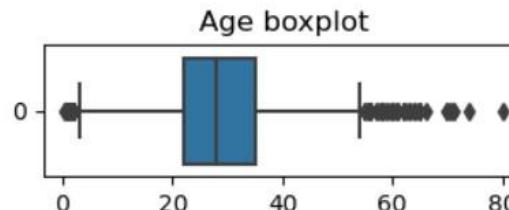


Data Pre-processing

Handle Outliers:

1- Check for Outliers

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

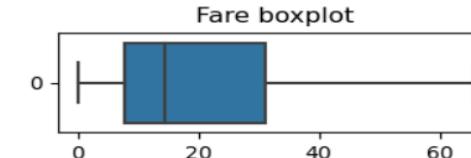
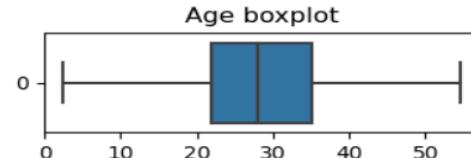


2- Remove Outliers

```
1 for col in num_cols:
2     Q1 = df[col].quantile(.25)
3     Q3 = df[col].quantile(.75)
4     IQR = Q3 - Q1
5     Lower_Fence = Q1 - 1.5 * IQR
6     Upper_Fence = Q3 + 1.5 * IQR
7     Lower_Outliers = df[df[col] < Lower_Fence][col].values
8     Upper_Outliers = df[df[col] > Upper_Fence][col].values
9     df[col].replace(Lower_Outliers, Lower_Fence, inplace=True)
10    df[col].replace(Upper_Outliers, Upper_Fence, inplace=True)
```

3- Make Sure Outliers are Removed

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



5. 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

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

129

- Remove Duplicates

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

- Make Sure that Duplicates are Removed

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

0

6. Visualization

Visualization:

- Visualization is the process of creating graphs about our data, that helps us 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.

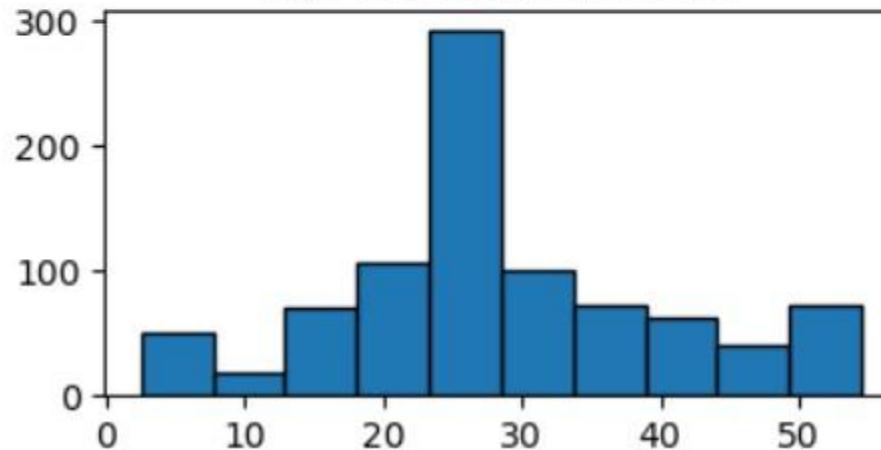
A. Data Distribution Graphs:

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

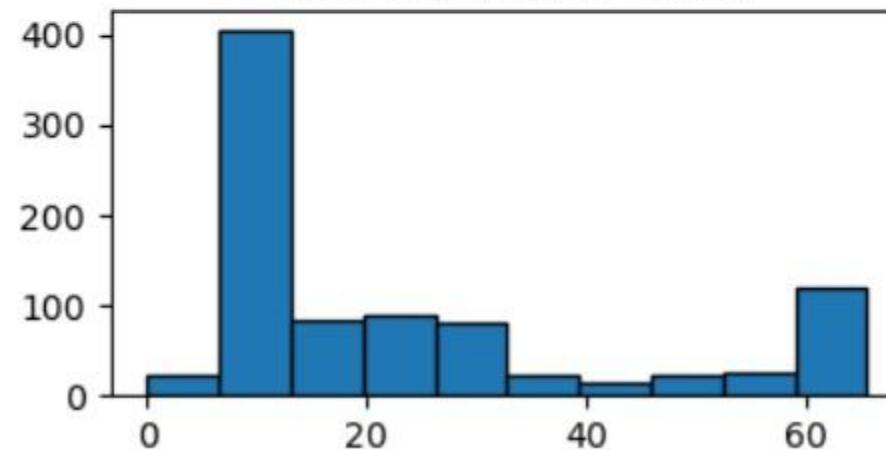
Numerical Data Distribution (Histogram):

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

Age Distribution Graph

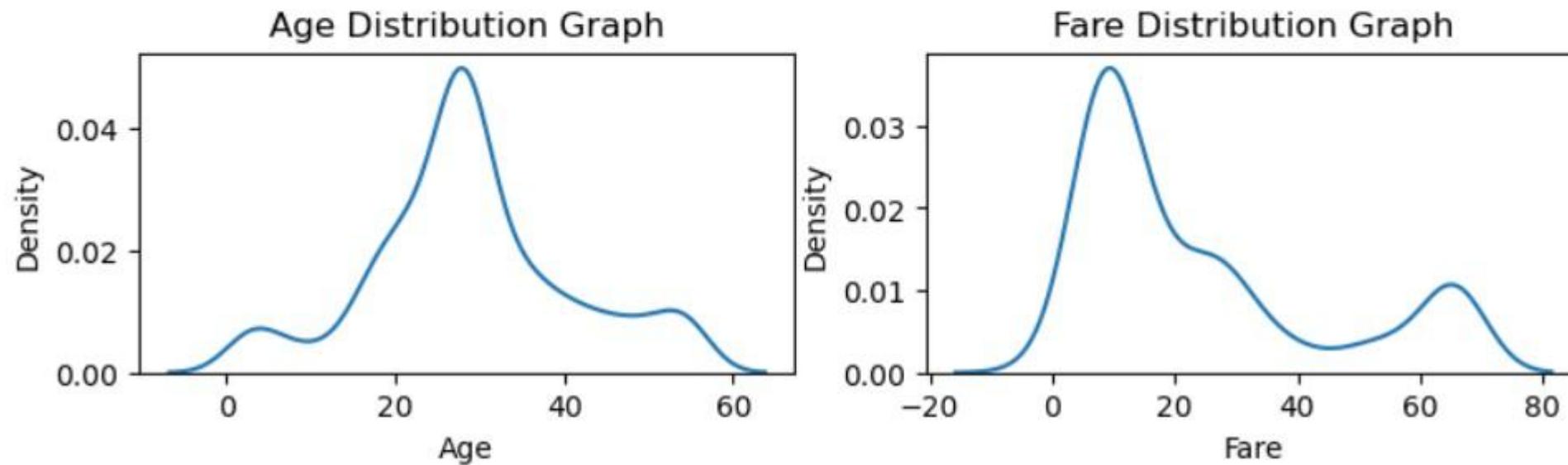


Fare Distribution Graph



Numerical Data Distribution (Kde Plot):

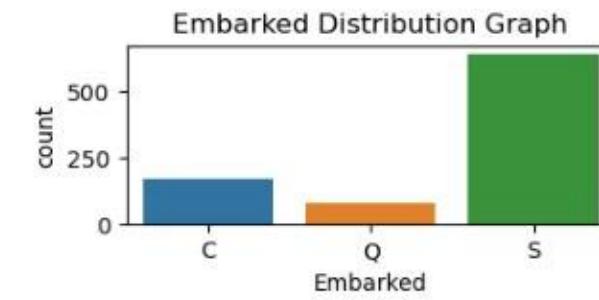
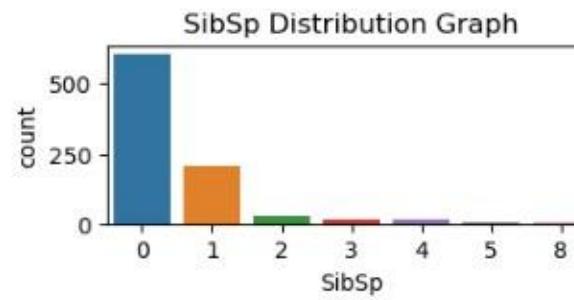
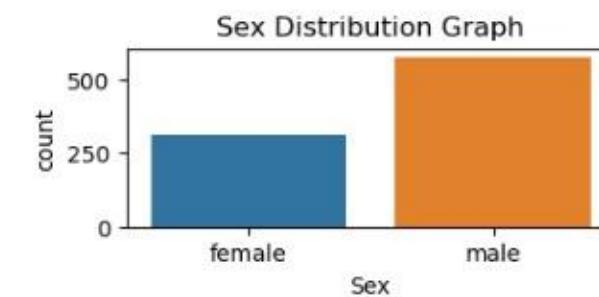
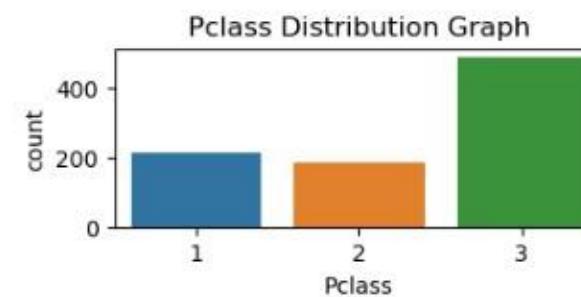
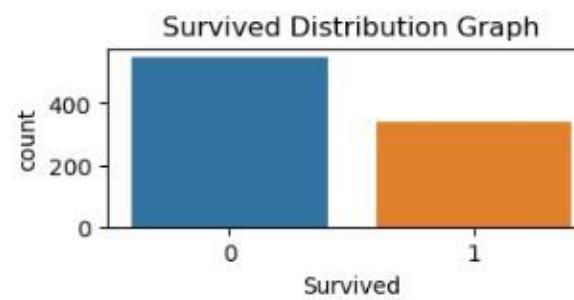
```
1 num_cols = df.select_dtypes("number").columns
2 plt.figure(figsize=(9, 2))
3 for i, col in enumerate(num_cols):
4     plt.subplot(1, 2, i+1)
5     sns.kdeplot(df[col])
6     plt.title(f"{col} Distribution Graph")
7 plt.show()
```



Data Pre-processing

Categorical Data Distribution (Count Plot):

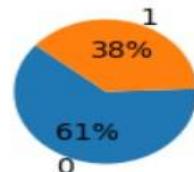
```
1 cat_cols = df.select_dtypes("category").columns
2 plt.figure(figsize=(14, 4))
3 for i, col in enumerate(cat_cols):
4     plt.subplot(2, 3, i+1)
5     sns.countplot(x=col, data=df)
6     plt.title(f"{col} Distribution Graph")
7 plt.subplots_adjust(hspace=.8, wspace=.3)
8 plt.show()
```



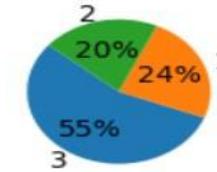
Categorical Data Distribution (Pie Plot):

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

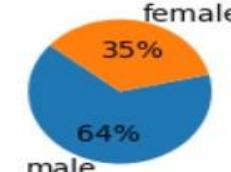
Survived Distribution Graph



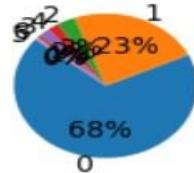
Pclass Distribution Graph



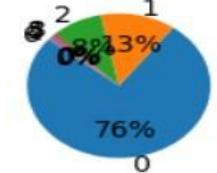
Sex Distribution Graph



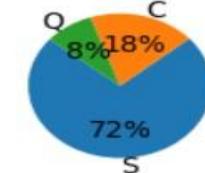
SibSp Distribution Graph



Parch Distribution Graph



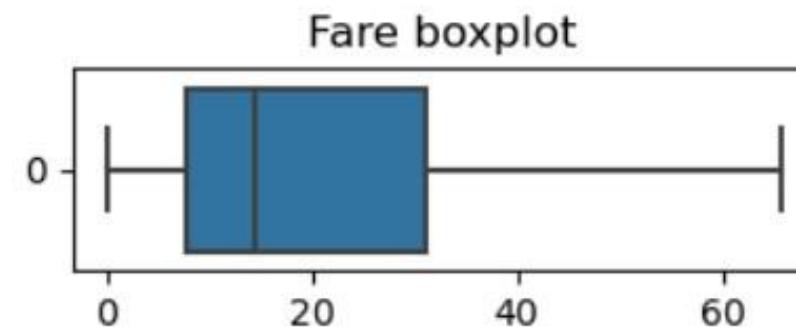
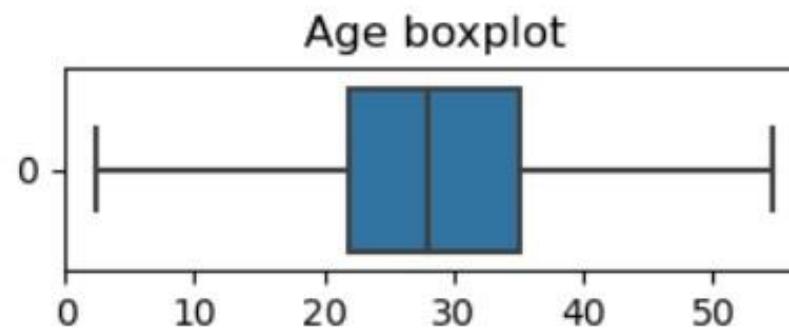
Embarked Distribution Graph



B. Outlier Detection Graphs:

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

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



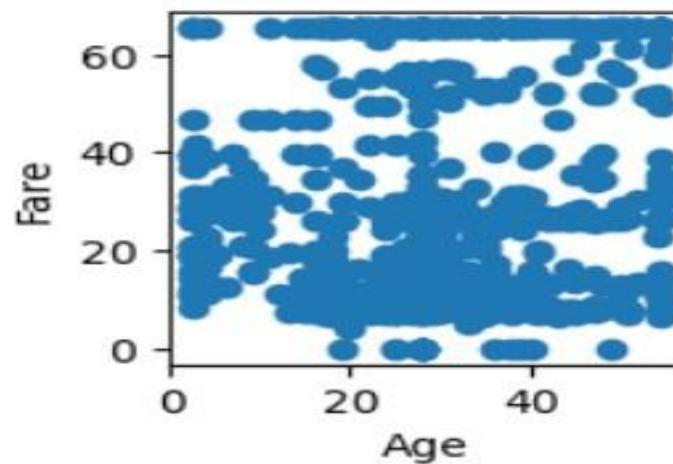
C. Relationship Graphs:

- 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):

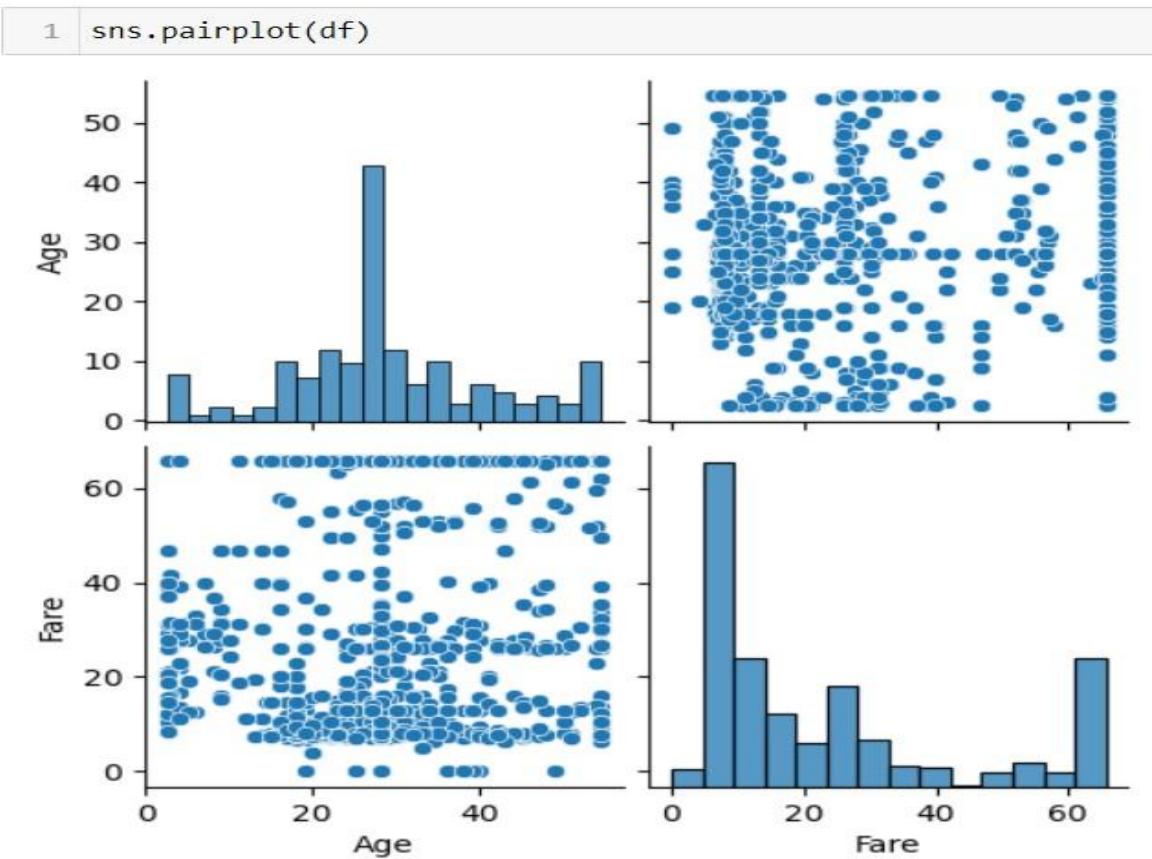
- Get the cartesian coordinates for all the data points in 2d space, where the two dimensions are the two columns.

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



Numerical/Numerical Relationship (Pair Plot):

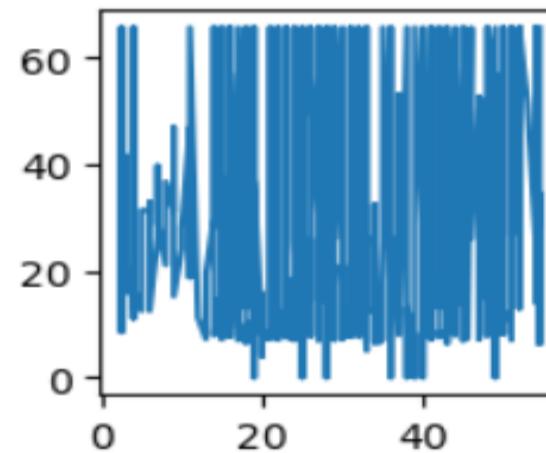
- Is a graph that gets scatter plot between each two numerical columns.



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.

```
1 sorted_df = df.sort_values(by="Age")
2 plt.figure(figsize=(2, 2))
3 plt.plot(sorted_df["Age"], sorted_df["Fare"])
4 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.

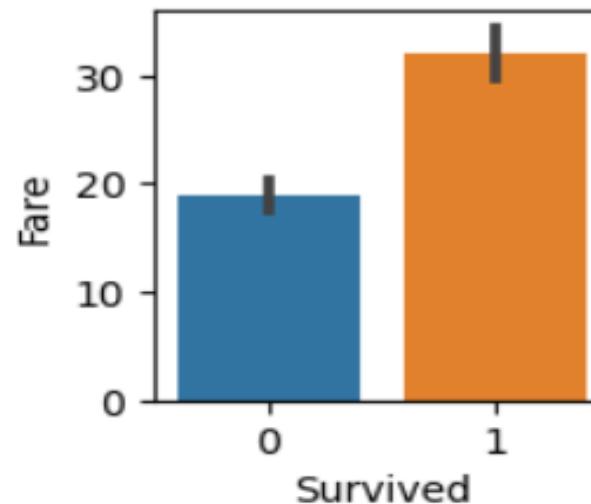
```
1 corr = df.corr()  
2 plt.figure(figsize=(2, 2))  
3 sns.heatmap(corr, annot=True)  
4 plt.show()
```



Numerical/Categorical Relationship (Bar Plot):

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

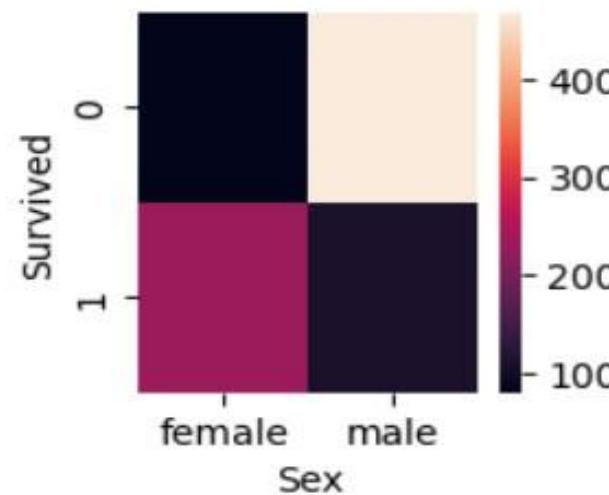
```
1 plt.figure(figsize=(2, 2))
2 sns.barplot(x="Survived", y="Fare", data=df)
3 plt.show()
```



Categorical / Categorical Relationship (Heat Map):

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

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



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.

Data Pre-processing

Split the Data:

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

1 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

1 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: $(X - \text{Min}) / (\text{Max} - \text{Min})$, where X refers to the column's values.

Data Pre-processing

Apply Normalization:

```
1 from sklearn.preprocessing import MinMaxScaler  
2 num_cols = X.select_dtypes("number").columns  
3 scaler = MinMaxScaler()  
4 scaler.fit(X[num_cols])  
5 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**.

Data Pre-processing

Apply Encoding:

```
1 from category_encoders import OneHotEncoder  
2 encoder = OneHotEncoder(cols = str_cols, drop_invariant=True)  
3 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