TITANIC

MACHINE LEARNING FROM DISASTER

Training Structure

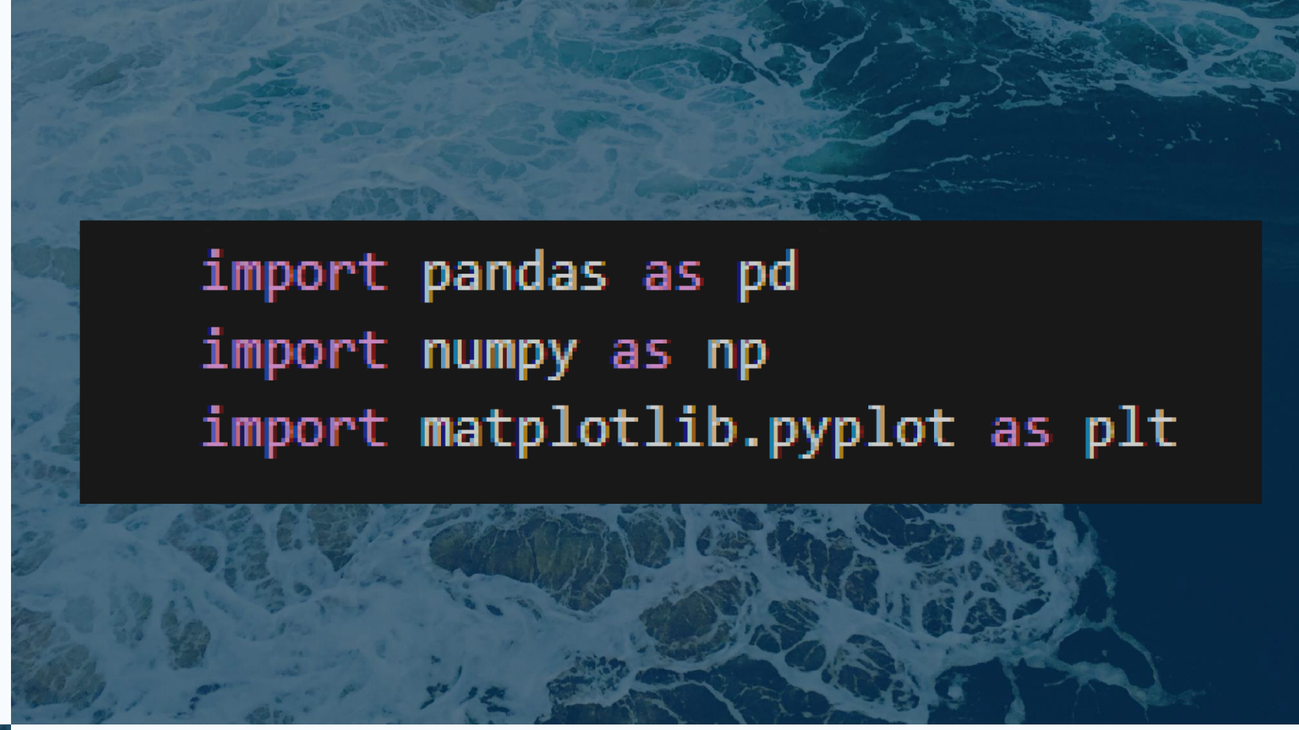
Issues to Tackle:

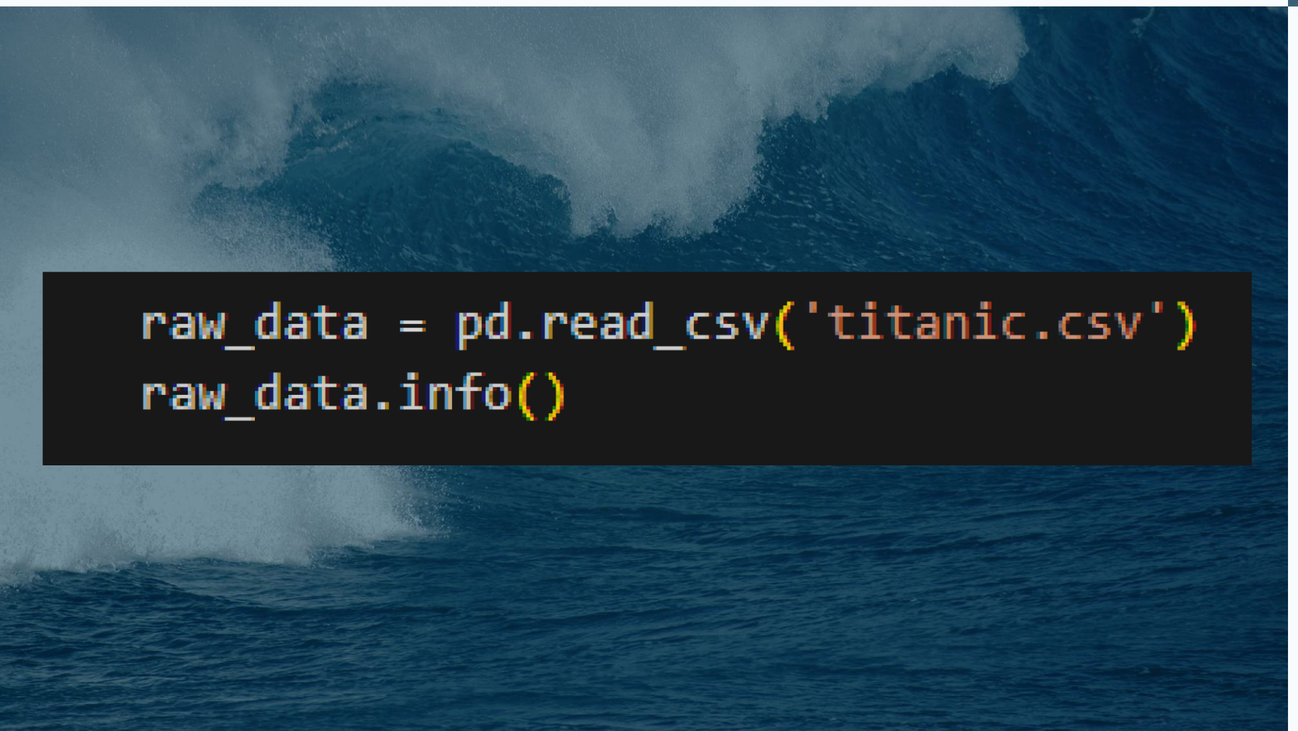
The goal is typically to predict

whether a passenger survived or not based on various features provided in the dataset. The dataset includes information about passengers such as age, sex, ticket class, fare, and more.

Library

Pandas is used for data manipulation and analysis, NumPy is for numerical operations and array handling, and Matplotlib is for creating visualizations from your data. These libraries are often used together in data science and machine learning workflows.





DATASET

loading the Titanic dataset using pd.read_csv and then checking the information about the DataFrame using info(). The info() method provides a concise summary of the DataFrame, including the data types and the presence of missing values.

NULL CLEANING FROM DATASET

```
raw_data.isna().sum()
```

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

The isna().sum() method is used to count the number of missing values (NaN or null values) in each column of a DataFrame. It's a useful way to quickly assess the extent of missing data in your dataset.

```
raw_data.drop(columns=['Cabin'], inplace = True)
```

```
#mean vs median to fill missing ages
print(raw_data.Age.mean())
print(raw_data.Age.median())
```

```
raw_data.Age.fillna(raw_data.Age.median(),inplace = True)
```

```
raw_data.Embarked.mode()[0]
raw_data.Embarked.fillna(raw_data.Embarked.mode()[0], inplace = True)
```

DATATYPE CLEANING FROM DATASET

```
from sklearn.preprocessing import OneHotEncoder
```

The OneHotEncoder is commonly used for converting categorical data into a one-hot encoded format, which is suitable for feeding into machine learning models.

```
preprocessed_data["Sex"] = (preprocessed_data["Sex"] == "male").astype(int)
preprocessed_data["Sex"].unique()
```

converting the 'Sex' column in the DataFrame preprocessed_data to numerical values (0 for 'female' and 1 for 'male'). The .astype(int) method is used to convert the boolean result of the comparison to integers.

```
preprocessed_data["Pclass"] = (preprocessed_data["Pclass"]).astype(int)
preprocessed_data["Pclass"].unique()
```

Same as the last column

```
encoder = OneHotEncoder()
ed = encoder.fit_transform(preprocessed_data[["Embarked"]]).toarray()
ed = pd.DataFrame(ed,columns=encoder.get_feature_names_out())
preprocessed_data = pd.concat([preprocessed_data, ed],axis = 1)
```

1. encoder = OneHotEncoder(): Creates an instance of the OneHotEncoder class.
2. ed = encoder.fit_transform(preprocessed_data[["Embarked"]]).toarray(): Applies the one-hot encoding to the 'Embarked' column and converts the sparse matrix result to a dense NumPy array.
3. ed = pd.DataFrame(ed, columns=encoder.get_feature_names_out()): Converts the one-hot encoded array to a DataFrame with appropriate column names.
4. preprocessed_data = pd.concat([preprocessed_data, ed], axis=1): Concatenates the original DataFrame (preprocessed_data) with the new one-hot encoded DataFrame (ed) along the columns (axis=1).

TRAIN-TEST-SPLIT

```
#features and labels (x and y)
X = data.drop(columns=['Survived'], axis = 1)
Y = data['Survived']
```

- **Features (X):X** is assigned the DataFrame**data** with the 'Survived' column removed. This is often done when you want to use all other columns in the dataset as features for prediction.

Labels (Y):Y is assigned the 'Survived' column of the DataFrame **data**. This column represents the target variable that you want to predict.

After defining **X** and **Y** this way, you can use them to train machine learning models.

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_validation_test, Y_train, Y_validation_test = train_test_split(X,Y , test_size=0.4, random_state=100)

X_validation, X_test, Y_validation, Y_test =train_test_split(X,Y , test_size=0.5, random_state=100)
```

- **First Split (X_train, X_validation_test, Y_train, Y_validation_test):** Here, 60% of the data is used for training (**X_train**, **Y_train**), and 40% is used for a combined validation and test set (**X_validation_test**, **Y_validation_test**).
- **Second Split (X_validation, X_test, Y_validation, Y_test):** From the validation_test set, 50% is used for validation (**X_validation**, **Y_validation**), and the remaining 50% is used for the test set (**X_test**, **Y_test**).

The **random_state** parameter is set to 100 to ensure reproducibility, meaning that if you run the same code with the same random seed, you will get the same splits each time. This is useful for getting consistent results, especially when you are experimenting with different models or parameters.

MODEL IMPLEMENTATION

```
from sklearn.linear_model import LinearRegression

# Assuming X_train and Y_train are your training features and labels
linear_model = LinearRegression()
linear_model.fit(X_train, Y_train)
```

- **from sklearn.linear_model import LinearRegression**: This imports the **LinearRegression** class from scikit-learn, which is used for linear regression modeling.
- **linear_model = LinearRegression()**: This creates an instance of the linear regression model.
- **linear_model.fit(X_train, Y_train)**: This fits (trains) the linear regression model on your training data, where **X_train** is the input features, and **Y_train** is the corresponding target labels.