BIRZEIT UNIVERSITY

Faculty of Engineering and Technology Department of Electrical and Computer Engineering

ENCS5141—Intelligent Systems Laboratory

Case Study #1—Data Cleaning and Feature Engineering for the Titanic Dataset.

Prepared by: Dunia Jaser - 1201345

Instructor: Dr. Mohammad Jubran

Assistant: Eng. Hanan Awawdeh

Date: March 29, 2024

Abstract

This experiment, focusing on the Titanic dataset, aimed to practically introduce the theoretical basics of preparing the dataset for machine learning analysis. This includes loading the dataset, performing initial data exploration, addressing data quality issues, analyzing feature relevance, encoding categorical variables, splitting the dataset, scaling or normalizing features, applying dimensionality reduction, and validating the pre-processing pipeline with a machine learning model. Finally, the results of the pre-processing will be compared with those from the raw data to gauge improvement in model performance.

Contents

1 Introduction

The dataset of the Titanic is a significant source that encompasses all pertinent facts about the people and how their fate was sealed in this tragic journey. The main objective of our research is to discover these patterns and determinants behind survival with the help of artificial intelligence. Therefore, as we explore the dataset, our aspiration is not only to add to historical knowledge but also to get better at predicting future events.

1.1.Data Cleaning

Data cleaning is the process of fixing or removing incorrect, corrupted, incorrectly formatted, duplicate, or incomplete data within a dataset. When combining multiple data sources, there are many opportunities for data to be duplicated or mislabeled. If data is incorrect, outcomes and algorithms are unreliable, even though they may look correct. There is no one absolute way to prescribe the exact steps in the data cleaning process because the processes will vary from dataset to dataset. But it is crucial to establish a template for your data cleaning process so you know you are doing it the right way every time.

1.2. Feature Engineering

Feature engineering, in data science, refers to manipulation — addition, deletion, combination, mutation — of your data set to improve machine learning model training, leading to better performance and greater accuracy. Effective feature engineering is based on sound knowledge of the business problem and the available data sources.

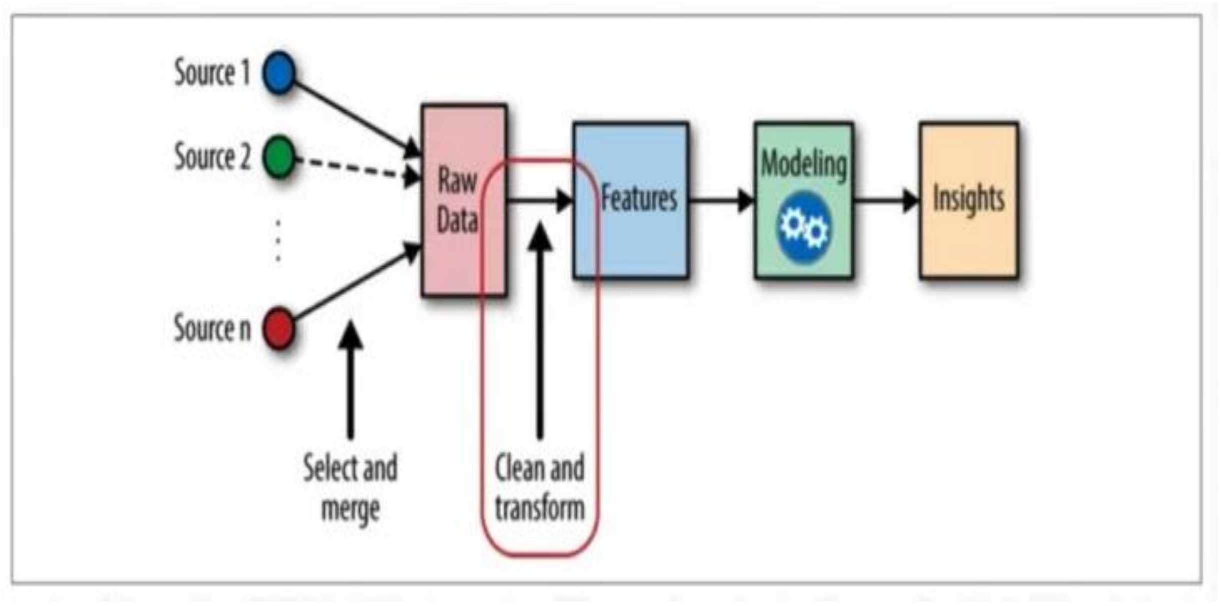


FIGURE 1: FEATURE ENGINEERING IN ML LIFECYCLE DIAGRAM

To ensure that the Titanic dataset is well-prepared for machine learning analysis, the objective is to use data cleaning and feature engineering methods. Two scientific questions will guide this study:

- o What steps should be taken to handle the missing values and outliers that are part of the Titanic data, in order to ensure high-quality information?
- o What factors are critical in determining the survival of passengers, and how can we represent these factors quantitatively?

- How does the performance of a machine learning model trained on pre-processed data compare to that of a model trained on raw data thereby quantifying the impact of data pre-processing steps such as feature filtering, transformation, and dimensionality reduction on model efficacy?

The study aims to establish if a series of pre-processing steps can result in improving the performance level of ML models in forecasting survival rates on the Titanic. Answering these questions helps to highlight the impacts that diligent data pre-processing has on the accuracy and reliability of machine learning models.

2 Procedure and Discussion

2.1. Data Preparation

The Titanic dataset was loaded using Seaborn's function as shown below:

```
import seaborn as sns
import numpy as np
# load dataset titanic
df = sns.load_dataset('titanic')
```

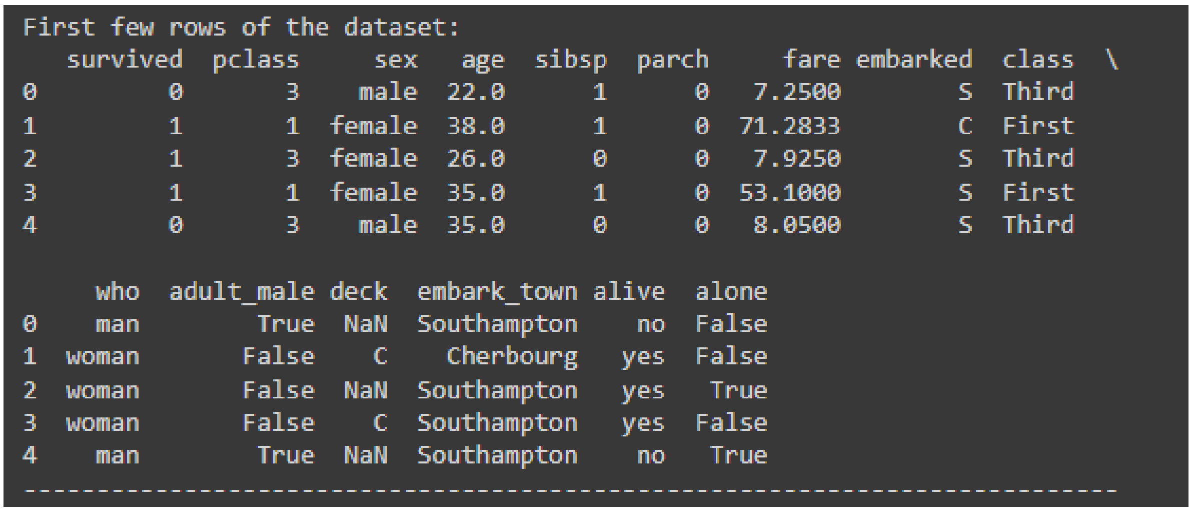
LISTING 2.1 LOADING THE TITANIC DATASET

In the initial stage of analysis, a deep dive into the Titanic dataset was conducted. This step was critical in gaining a deep understanding of the dataset, which subsequently informed the data cleaning and preparation strategy.

We began by displaying the first few rows of the dataset to get a preliminary view of its structure and the types of data it contained

```
print("First few rows of the dataset:")
print(df.head())
print("Summary of the dataframe (info):")
print(df.info())
print("Descriptive statistics of the dataset:")
print(df.describe())
print("Missing values in each column:")
print(df.isnull().sum())
```

LISTING 2. 2 DATA EXPLORATION AND SUMMARY STATISTICS IN PYTHON



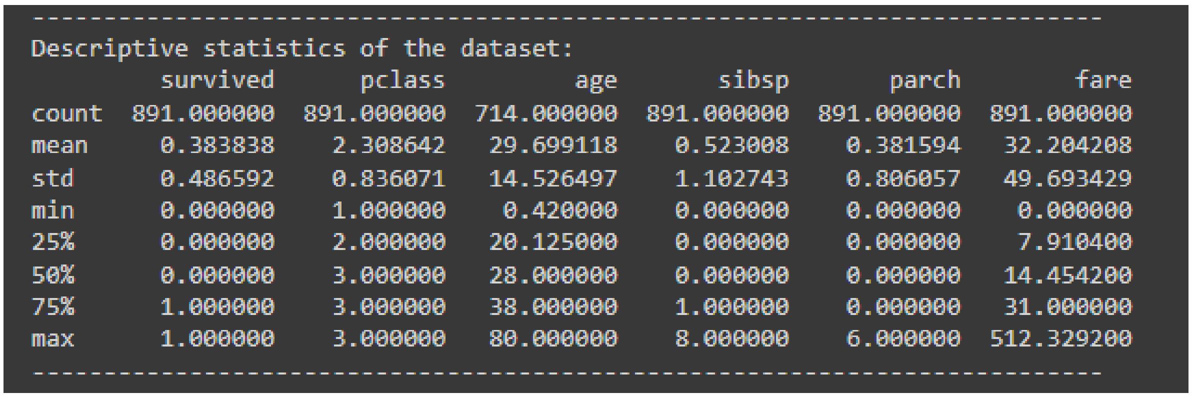
```
First few rows of the dataset:
   survived  pclass     sex   age  sibsp  parch     fare embarked  class  \
0         0       3    male  22.0      1      0   7.2500        S  Third
1         1       1  female  38.0      1      0  71.2833        C  First
2         1       3  female  26.0      0      0   7.9250        S  Third
3         1       1  female  35.0      1      0  53.1000        S  First
4         0       3    male  35.0      0      0   8.0500        S  Third

     who  adult_male deck  embark_town alive  alone
0    man        True  NaN  Southampton    no  False
1  woman       False    C    Cherbourg   yes  False
2  woman       False  NaN  Southampton   yes   True
3  woman       False    C  Southampton   yes  False
4    man        True  NaN  Southampton    no   True
------------------------------------------------------------------------------
```

FIGURE 2- 1 FIRST FEW ROWS OF THE DATASET

The features existing in the Titanic dataset include survival status, passenger class (pclass), sex, age, the number of siblings/spouses aboard (sibsp), the number of parents/children aboard (parch), ticket fare, port of embarkation (embarked), passenger class by name (class), who they were (man, woman, or child), if they were an adult male, the deck they were on, the town they embarked from (embark_town), whether they survived (alive), and if they were alone

The DataFrame for the Titanic dataset consists of 891 entries with indexes 0 to 890. It contains 15 columns of various data types including integers, floats, objects (strings), categories, and booleans. Not all columns contain complete data; in particular, the "Age" column contains 714 non-zero entries, the "Embarked" and "Embark_town" columns each contain 889 non-zero entries, and the "Deck" column contains only 203 non-zero entries. Showing the presence of missing persons shows values. It has a mix of data types: integers and floats for numerical values, categories for categorical data, and objects for text and additional categorical information, along with boolean values for binary attributes.

The dataset included a total of 891 passengers, of whom 342 (or 38.38% average survival rate) survived the tragic event, while 549 (or 61.62%) did not survive. These data highlight the significant loss of life during the disaster and form a key element of our analysis as they provide a clear numerical perspective on the survival outcomes on Titanic



```
----------------------------------------------------------------------
Descriptive statistics of the dataset:
        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
----------------------------------------------------------------------
```

FIGURE 2- 2 DESCRIPTIVE STATISTICS OF THE DATASET

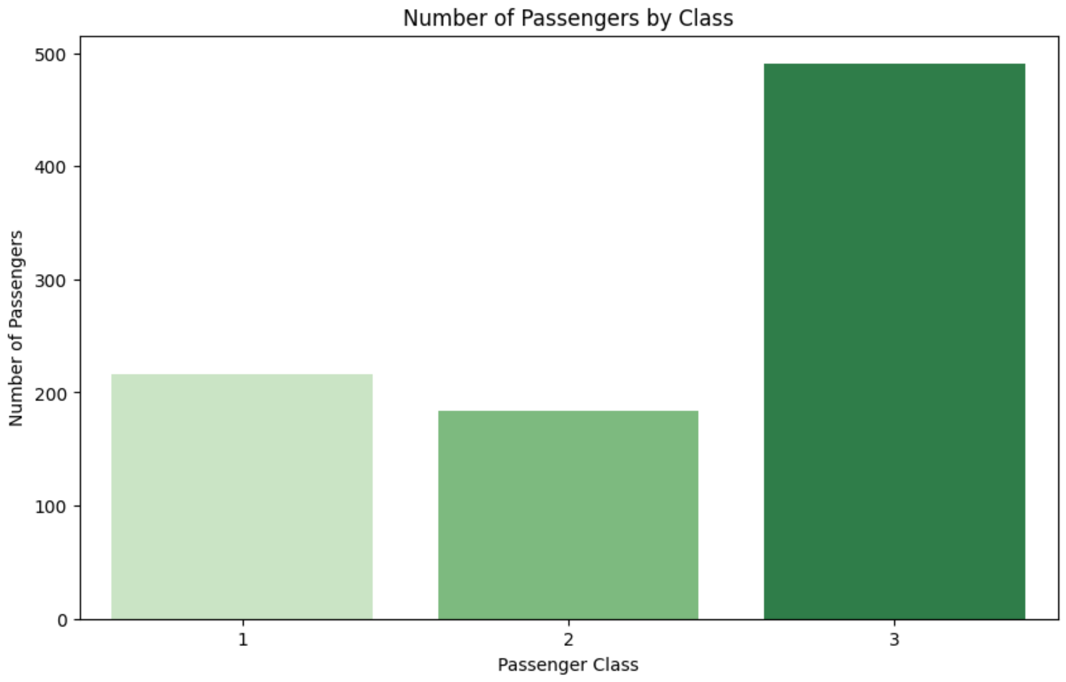


FIGURE 2- 3 SURVIVAL PASSENGERS BY CLASS

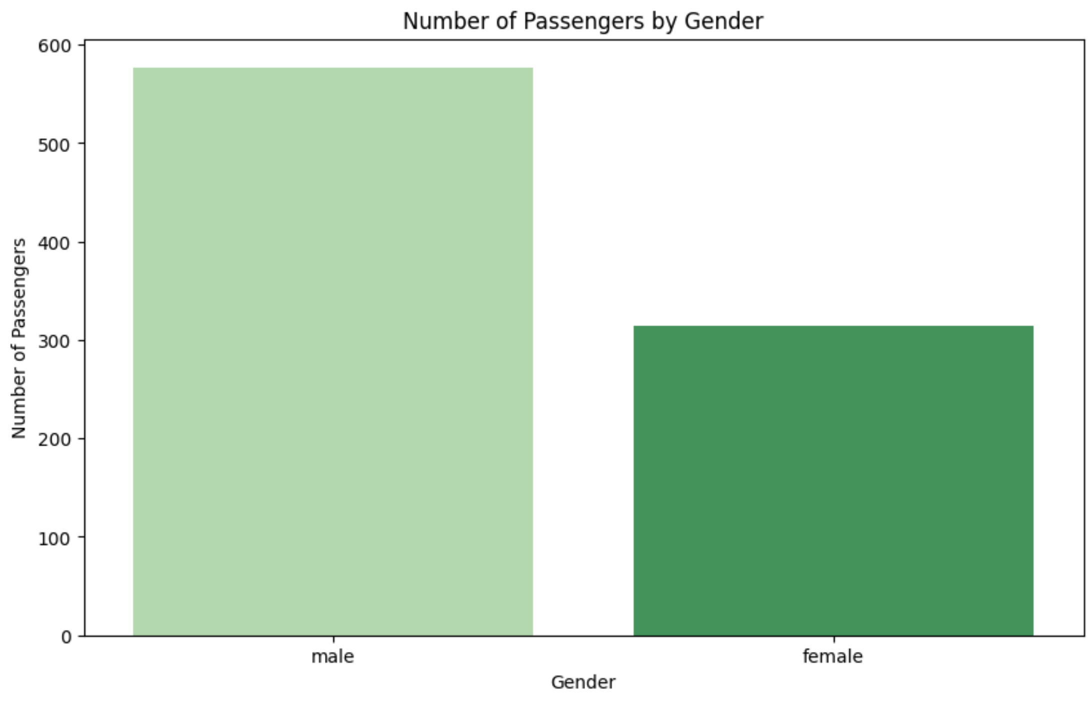


FIGURE 2- 4 SURVIVAL PASSENGERS BY GENDER

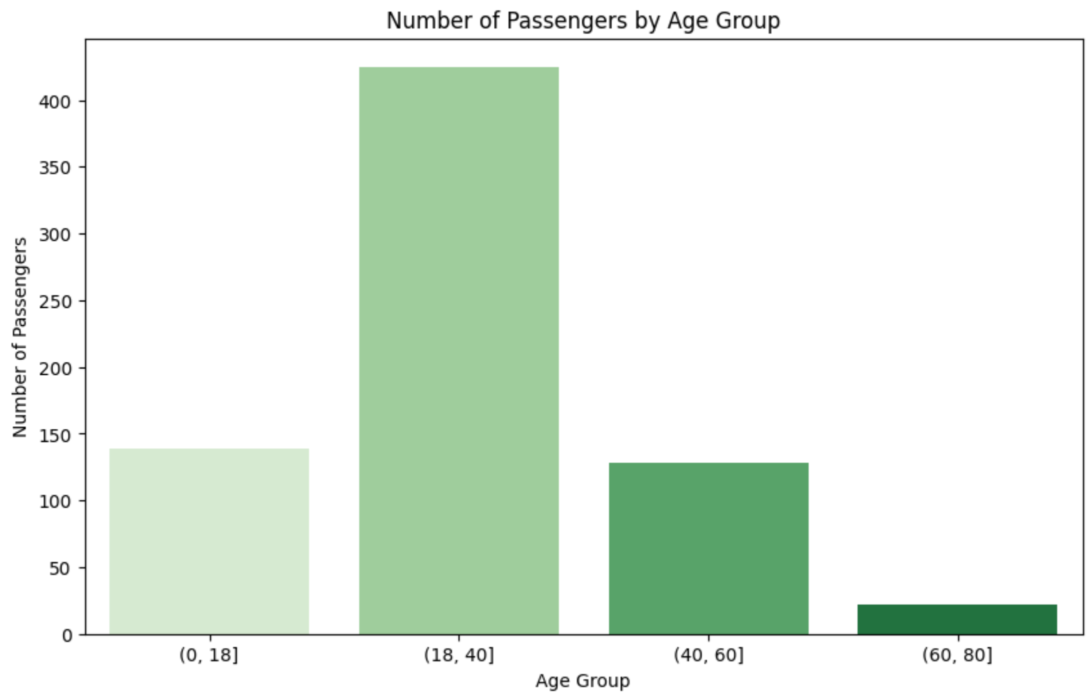


FIGURE 2- 5 SURVIVAL PASSENGERS BY AGE GROUP

The Titanic dataset reveals that survival rates varied widely across classes, genders, and age groups. First-class passengers had the highest survival rate at 62.96%, followed by second-class at 47.28%, and third-class at 24.24%. Gender-wise, females had a notably higher survival rate of 74.20%, compared to males at 18.89%. Age-wise, children and teenagers (0-18 years) showed a survival rate of 50.36%, adults (18-40 years) at 38.82%, middle-aged individuals (40-60 years) at 39.06%, and older adults (60-80 years) had the lowest at 22.73%, indicating that class, gender, and age significantly influenced survival chances on the Titanic.

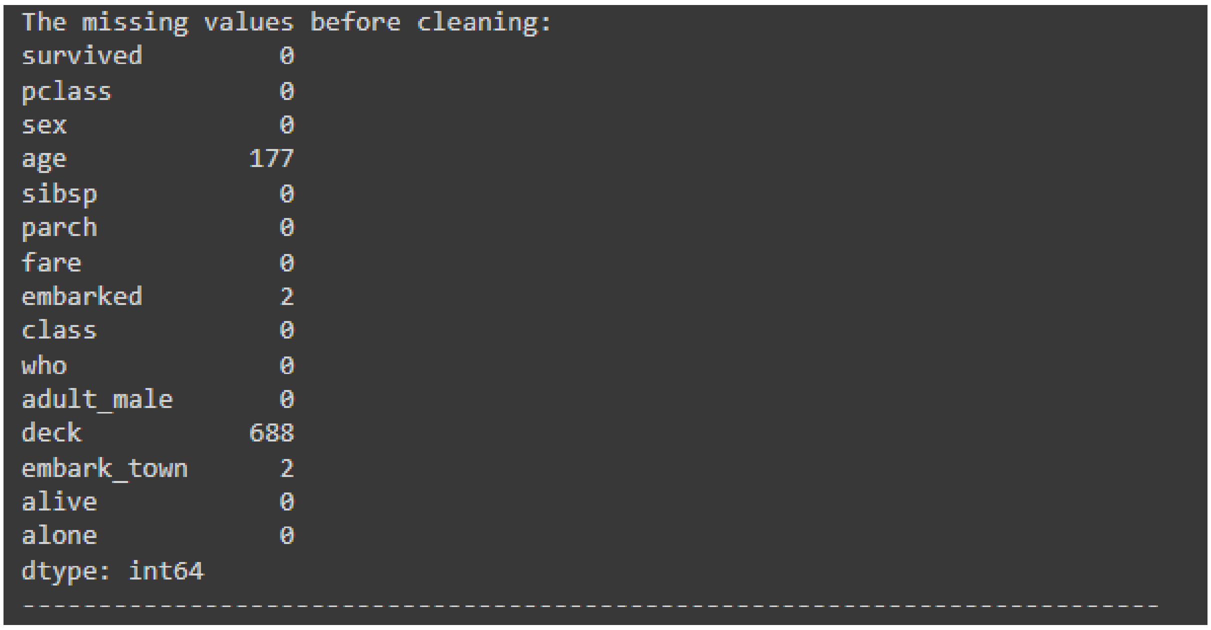
2.2. Data Cleaning

2.2.1. Missing Values

Missing values within the dataset was addressed as shown below:

```
print("The missing values before cleaning:")
print(df.isnull().sum())
```

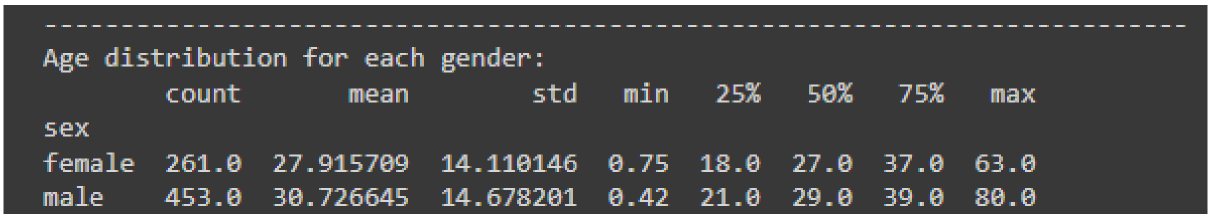
LISTING 2. 3 MISSING VALUES COUNT FOR EACH FEATURE



```
The missing values before cleaning:
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64
---------------------------------------------------------------------------
```

FIGURE 2- 6 NUMBER OF MISSING VALUES IN EACH FEATURE

Note that there is a large amount missing in the 'age' and 'deck' categories. In order to improve data accuracy, the column labeled 'deck' with a high number of missing values was eliminated from the dataset. Further examination concentrated on the 'age' characteristic, uncovering unique age patterns among different genders, which were clarified through descriptive data and depicted using histograms to display the age breakdown by gender, alongside a density curve for a more thorough insight.



```
---------------------------------------------------------------------------
Age distribution for each gender:
        count       mean        std   min   25%   50%   75%   max
sex
female  261.0  27.915709  14.110146  0.75  18.0  27.0  37.0  63.0
male    453.0  30.726645  14.678201  0.42  21.0  29.0  39.0  80.0
```

FIGURE 2- 7 AGE DISTRIBUTION FOR EACH GENDER

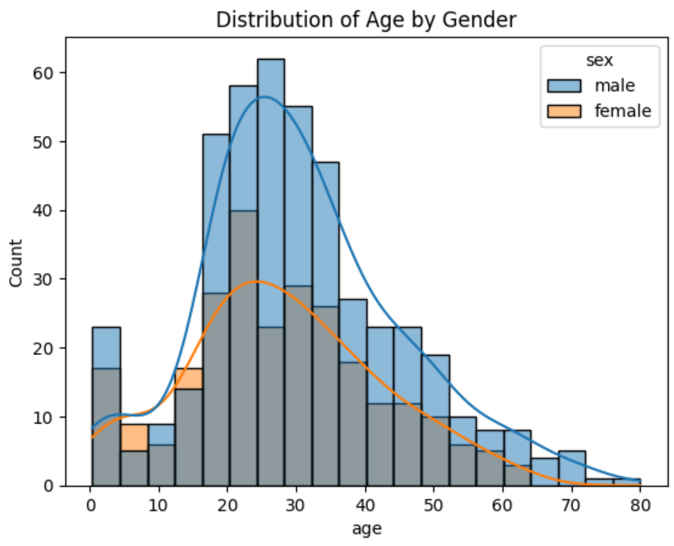


FIGURE 2- 8 AGE DISTRIBUTION

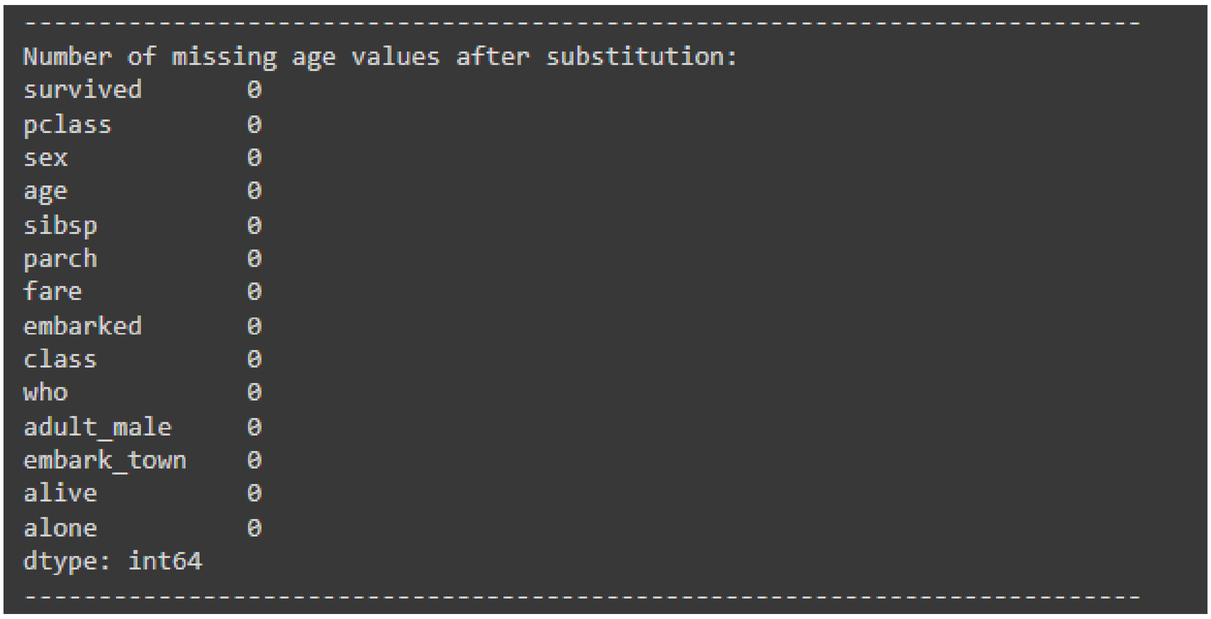
The chart above shows that most passengers, whether they were men (in blue) or women (in orange), were younger, with a high number of people in their twenties and thirties. The way the bars and lines go towards the right, higher for younger ages and lower for older ages, tells us that there were fewer elderly passengers. This kind of pattern, where the chart stretches out more on the right, suggests that the distribution of ages is skewed to the right for both men and women.

The median ages for men and women were calculated and utilized to impute missing age data since the distributions for age were comparable between genders. The most common points of embarkation, along with their corresponding town names, were similarly imputed in cases where this information was absent. Following these procedures, a verification was conducted, confirming that the missing age data had been effectively rectified. Consequently, the dataset's quality was significantly enhanced, rendering it more suitable for further analysis.

```
median_age_by_sex = df_cleaned.groupby('sex')['age'].median()
print("The median of age for each male and female:")
print(median_age_by_sex)
# Replace missing age values with the corresponding gender's median age
for sex, median_age in median_age_by_sex.items():
        df_cleaned.loc[(df_cleaned['sex'] == sex) & df_cleaned['age'].isnull(), 'age'] = median_age
# Impute missing values for 'embarked' and 'embark_town' with the mode value
imputer_embarked = SimpleImputer(strategy='most_frequent')
df_cleaned['embarked'] = imputer_embarked.fit_transform(df_cleaned[['embarked']])
df_cleaned['embark_town'] = imputer_embarked.fit_transform(df_cleaned[['embark_town']])
```

LISTING 2. 4 HANDLING MISSING VALUES

After that the Titanic dataset no longer contained any missing values.



```
---------------------------------------------------------------------------
Number of missing age values after substitution:
survived       0
pclass         0
sex            0
age            0
sibsp          0
parch          0
fare           0
embarked       0
class          0
who            0
adult_male     0
embark_town    0
alive          0
alone          0
dtype: int64
---------------------------------------------------------------------------
```

FIGURE 2- 9 NUMBER OF MISSING AGE VALUES AFTER SUBSTITUTION

2.2.2. Outliers

To detect outliers in the dataset, the interquartile range (IQR) method and box plot were employed.

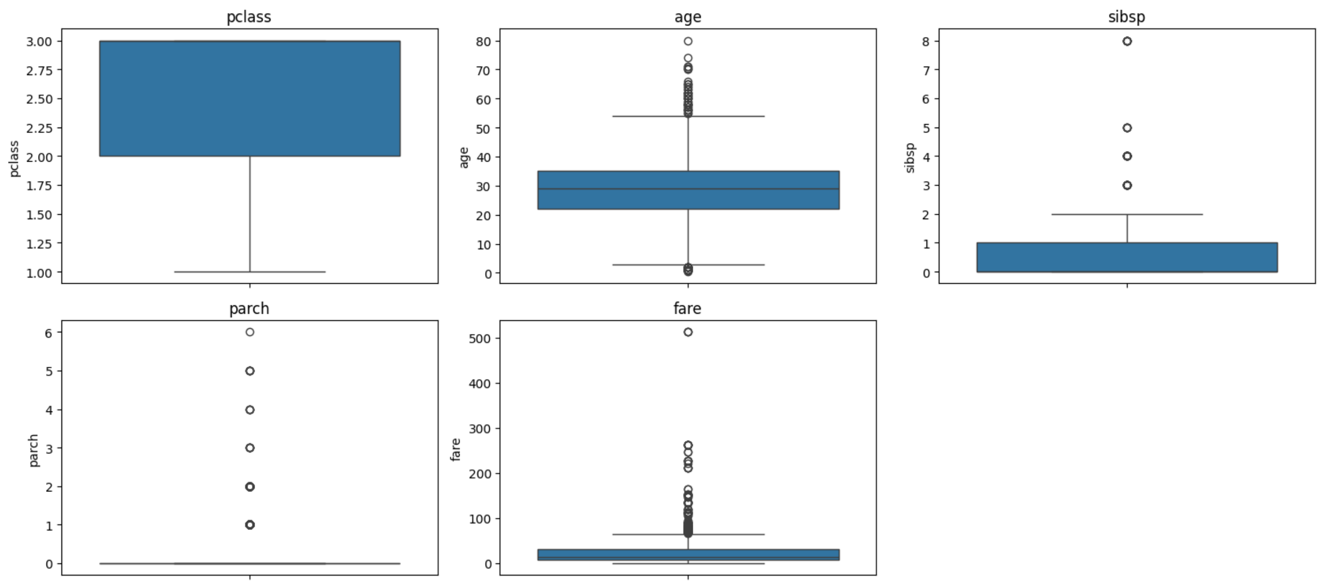


FIGURE 2- 10 BOX PLOT ANALYSIS FOR OUTLIER DETECTION IN TITANIC DATASET VARIABLES

To deal with the outliers found in the Titanic dataset, a capping technique was used. This included establishing a specific value that sets the limit for outliers. Values above this limit were replaced with either the limit itself or the closest value within the allowed range, reducing the impact of outliers on future analyses.

```
for col in numeric_cols:
    Q1 = df_cleaned[col].quantile(0.25)
    Q3 = df_cleaned[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    # Capping the outliers
    df_cleaned[col] = np.where(df_cleaned[col] > upper_bound, upper_bound, df_cleaned[col])
    df_cleaned[col] = np.where(df_cleaned[col] < lower_bound, lower_bound, df_cleaned[col])
```

LISTING 2. 5 HANDLING OUTLIERS

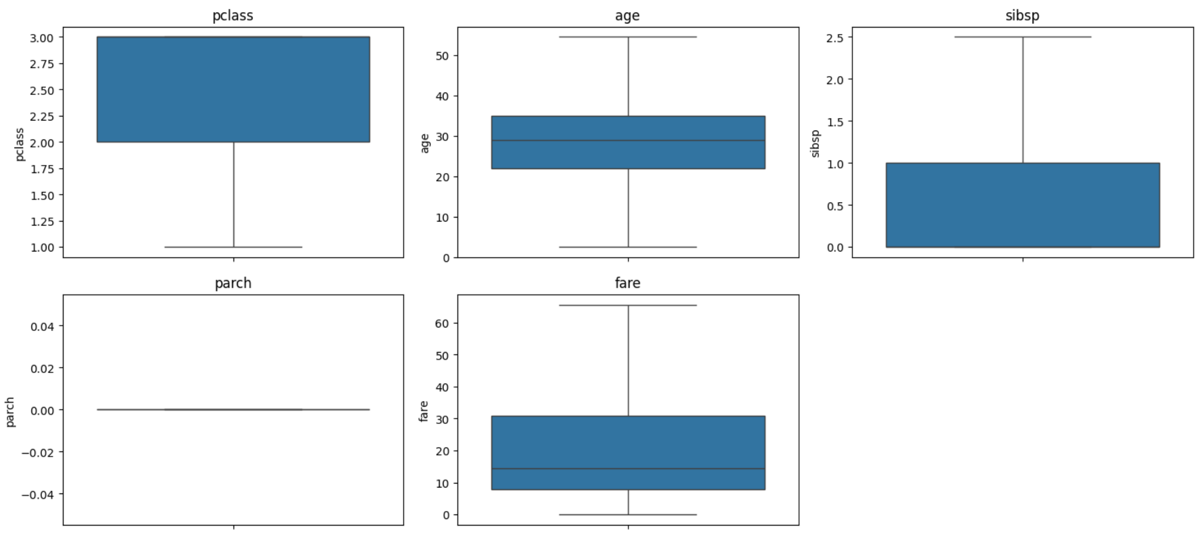


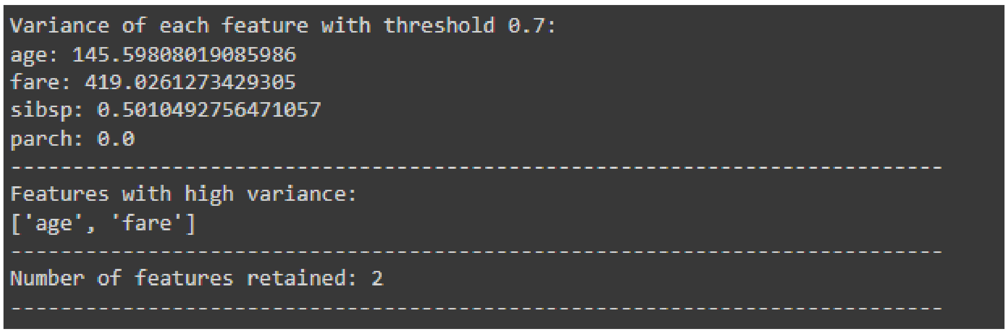
FIGURE 2- 11 BOX PLOT ANALYSIS AFTER THE DETECTION OF OUTLIER

2.3. Feature Engineering

Variance was utilized as a factor to determine the most crucial characteristics. A VarianceThreshold selector was configured in the scikit-learn library with a threshold of 0.7. The selector was next applied to the numerical columns of the refined dataset, effectively removing features with low variance that are unlikely to have a substantial impact on the model's predictive ability.

```
from sklearn.feature_selection import VarianceThreshold
threshold_value = 0.7
selector = VarianceThreshold(threshold=threshold_value)
selector.fit(df_cleaned[numeric_cols])
df_high_variance = df_cleaned[numeric_cols][df_cleaned[numeric_cols].columns[selector.get_support()]]
```

LISTING 2. 6 VARIANCE FEATURE SELECTION



```
Variance of each feature with threshold 0.7:
age: 145.59808019085986
fare: 419.0261273429305
sibsp: 0.5010492756471057
parch: 0.0
--------------------------------------------------------------------------
Features with high variance:
['age', 'fare']
--------------------------------------------------------------------------
Number of features retained: 2
--------------------------------------------------------------------------
```

FIGURE 2- 12 VARIANCE OF EACH FEATURE WITH THRESHOLD 0.7

The examination verified that the attributes 'age' and 'fare' are expected to be important for the predictive modeling job because they offer significant insights because of their diversity. As a result, these two characteristics were kept for additional modeling stages, so we can drop the unnecessary features like 'parch'. The procedure successfully decreased the amount of features to a more controllable level, concentrating on the variables that are most likely to have a significant impact on the predictive capability of the machine learning model.

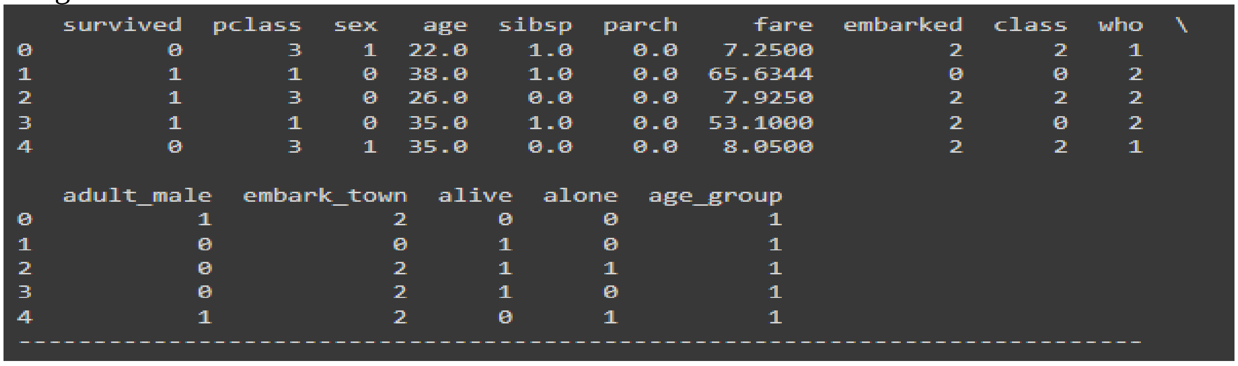
2.4. Encoding

For the categorical variables in the dataset, the LabelEncoder from the scikit-learn preprocessing module was employed. This encoder converts each category within the variables into a numerical format that is suitable for use in machine learning algorithms. Each categorical column was transformed individually, assigning a unique integer to each category within the column. This step is crucial as it allows the machine learning model to interpret and process the categorical data efficiently.

```
from sklearn.preprocessing import LabelEncoder
categorical_cols = df_cleaned.select_dtypes(include=['object',
'category','boolean']).columns
 label_encoder = LabelEncoder()
for column in categorical_cols:
                             df_cleaned[column]          =
label_encoder.fit_transform(df_cleaned[column])
```

LISTING 2. 7 ENCODING CATEGORICAL FEATURES

The transformed data was then reviewed to confirm the successful encoding of the categorical variables:



```
   survived  pclass  sex   age  sibsp  parch     fare  embarked  class  who  \
0         0       3    1  22.0    1.0    0.0   7.2500         2      2    1
1         1       1    0  38.0    1.0    0.0  65.6344         0      0    2
2         1       3    0  26.0    0.0    0.0   7.9250         2      2    2
3         1       1    0  35.0    1.0    0.0  53.1000         2      0    2
4         0       3    1  35.0    0.0    0.0   8.0500         2      2    1

   adult_male  embark_town  alive  alone  age_group
0           1            2      0      0          1
1           0            0      1      0          1
2           0            2      1      1          1
3           0            2      1      0          1
4           1            2      0      1          1
--------------------------------------------------------------------------
```

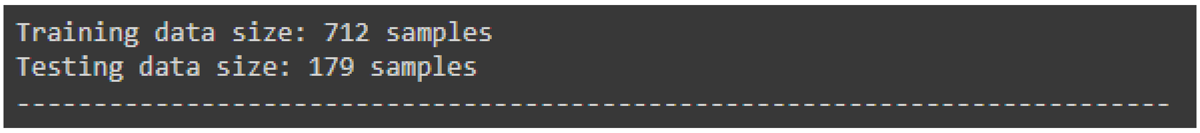
FIGURE 2- 13 THE TRANSFORMED DATA

2.5. Data Splitting

The data was divided into two separate parts in order to make it easier to train and evaluate the machine learning model. By making use of the train_test_split function from scikit-learn's model_selection module, 80% of the data was assigned for training, with the remaining 20% reserved for testing. The 'survived' column, which is the target variable, was taken out from the input features in order to create the X (features) and y (target) arrays. A state parameter was randomly assigned to maintain consistency in the split process. The accuracy of the partitioning process was confirmed by validating the final counts of the training and testing data samples.

```
from sklearn.model_selection import train_test_split

X = df_cleaned.drop('survived', axis=1)  # 'survived' is the target in the Titanic dataset

X = X.drop('alive', axis=1)

y = df_cleaned['survived']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Verify the split

print(f"Training data size: {X_train.shape[0]} samples")

print(f"Testing data size: {X_test.shape[0]} samples")
```

LISTING 2. 8 DATA SPLITTING



```
Training data size: 712 samples
Testing data size: 179 samples
----------------------------------------------------------------------
```

FIGURE 2- 14 NUMBER OF SAMPLES IN TRAINING AND TESTING DATASET

2.6. Scaling or Normalization

The StandardScaler tool was used to ensure that all features in the training data were scaled to have similar sizes, making each feature equally important. Subsequently, the test data were scaled in the same manner. As a result, the scaled training and test data are now prepared for further analysis

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)
```

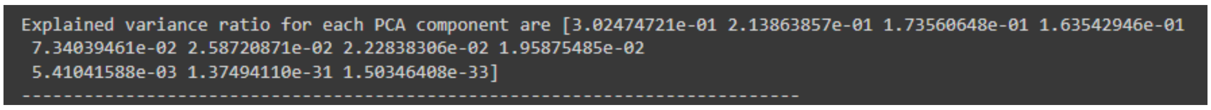
LISTING 2. 9 SCALING DATA

2.7. Dimensionality Reduction

To optimize the dataset and focus on the most informative aspects, Principal Component Analysis (PCA) was implemented using the sklearn.decomposition module. This method converted the scaled data for training and testing into a new set of variables called principal components that are linearly uncorrelated.

```
from sklearn.decomposition import PCA
import pandas as pd
feature_names = X_train.columns
# Apply PCA
pca = PCA(n_components=X_train_scaled.shape[1])
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)
components = pca.components_
# Map the components to the original features
components_df = pd.DataFrame(components, columns=feature_names, index=[f'PC{i+1}' for i in range(components.shape[0])])
print(f"Explained variance ratio for each PCA component are {pca.explained_variance_ratio_}")
```

LISTING 2. 10 DIMENSIONALITY REDUCTION



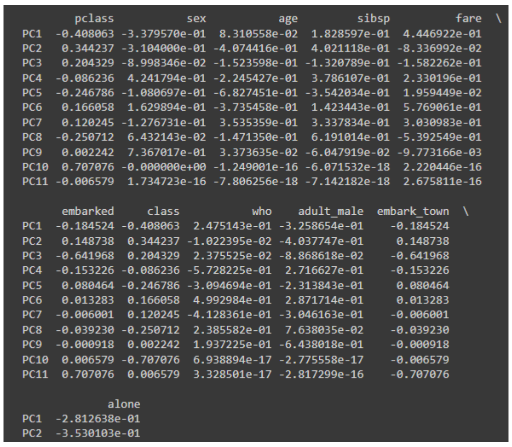
```
Explained variance ratio for each PCA component are [3.02474721e-01 2.13863857e-01 1.73560648e-01 1.63542946e-01
 7.34039461e-02 2.58720871e-02 2.22838306e-02 1.95875485e-02
 5.41041588e-03 1.37494110e-31 1.50346408e-33]
------------------------------------------------------------------------
```

FIGURE 2- 15 EXPLAINED VARIANCE RATIOS FROM THE PCA COMPONENTS

The explained variance ratio indicates the proportion of the dataset's total variance that is captured by each principal component:

- The first principal component captures 30.2% of the variance in the data.
- The second component accounts for 21.38% of the variance.
- The third component contains 17.35% of the variance.
- The fourth and subsequent components each explain a smaller percentage of the variance, with the numbers decreasing significantly for the later components.

The smallest values (for example, those in the range of e-31 or e-33) are extremely close to zero, showing that these elements do not represent any substantial variability in the data and are typically unhelpful.



```
        pclass           sex           age         sibsp          fare  \
PC1  -0.408063 -3.379570e-01  8.310558e-02  1.828597e-01  4.446922e-01
PC2   0.344237 -3.104000e-01 -4.074416e-01  4.021118e-01 -8.336992e-02
PC3   0.204329 -8.998346e-02 -1.523598e-01 -1.320789e-01 -1.582262e-01
PC4  -0.086236  4.241794e-01 -2.245427e-01  3.786107e-01  2.330196e-01
PC5  -0.246786 -1.080697e-01 -6.827451e-01 -3.542034e-01  1.959449e-02
PC6   0.166058  1.629894e-01 -3.735458e-01  1.423443e-01  5.769061e-01
PC7   0.120245 -1.276731e-01  3.535359e-01  3.337834e-01  3.030983e-01
PC8  -0.250712  6.432143e-02 -1.471350e-01  6.191014e-01 -5.392549e-01
PC9   0.002242  7.367017e-01  3.373635e-02 -6.047919e-02 -9.773166e-03
PC10  0.707076 -0.000000e+00 -1.249001e-16 -6.071532e-18  2.220446e-16
PC11 -0.006579  1.734723e-16 -7.806256e-18 -7.142182e-18  2.675811e-16

      embarked     class           who    adult_male  embark_town  \
PC1  -0.184524 -0.408063  2.475143e-01 -3.258654e-01    -0.184524
PC2   0.148738  0.344237 -1.022395e-02 -4.037747e-01     0.148738
PC3  -0.641968  0.204329  2.375525e-02 -8.868618e-02    -0.641968
PC4  -0.153226 -0.086236 -5.728225e-01  2.716627e-01    -0.153226
PC5   0.080464 -0.246786 -3.094694e-01 -2.313843e-01     0.080464
PC6   0.013283  0.166058  4.992984e-01  2.871714e-01     0.013283
PC7  -0.006001  0.120245 -4.128361e-01 -3.046163e-01    -0.006001
PC8  -0.039230 -0.250712  2.385582e-01  7.638035e-02    -0.039230
PC9  -0.000918  0.002242  1.937225e-01 -6.438018e-01    -0.000918
PC10  0.006579 -0.707076  6.938894e-17 -2.775558e-17    -0.006579
PC11  0.707076  0.006579  3.328501e-17 -2.817299e-16    -0.707076

             alone
PC1  -2.812638e-01
PC2  -3.530103e-01
```

FIGURE 2- 16 THE PCA COMPONENTS AND THEIR ASSOCIATION WITH ORIGINAL FEATURES

For example, features like 'sex', 'pclass', and 'fare' have a greater impact on PC1 due to their larger absolute values.

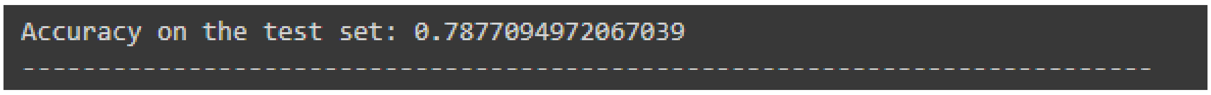
2.8. Model Training and Evaluation

2.8.1. Model Training on Pre-processed Data

In order to verify the pre-processing pipeline, a Random Forest classifier was used by utilizing the RandomForestClassifier from the sklearn.ensemble module. The effectiveness of reducing the dimensionality in capturing important dataset features was evaluated by training the model on PCA-transformed training data.

```
from sklearn.ensemble import RandomForestClassifier
from        sklearn.metrics        import        accuracy_score,
classification_report


clf = RandomForestClassifier(random_state=42)
clf.fit(X_train_pca, y_train)
# Predict on the test set
y_pred = clf.predict(X_test_pca)
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy on the test set: {accuracy}")
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

LISTING 2. 11 RANDOM FOREST MODEL EVALUATION ON PCA-TRANSFORMED DATA



```
Accuracy on the test set: 0.7877094972067039
-----------------------------------------------------------------------
```

FIGURE 2- 17 RANDOM FOREST MODEL ACCURACY RESULT ON TEST DATA

The model achieved an accuracy of 78.77% on the test set, indicating the percentage of correctly predicted instances.

Classification Report:

TABLE 2- 1 CLASSIFICATION REPORT ON PRE-PROCESSED DATA

Survived	precision	recall	f1-score	support
no	0.82	0.82	0.82	105
yes	0.74	0.74	0.74	74
accuracy			0.79	179
macro avg	0.78	0.78	0.78	179
weighted avg	0.79	0.79	0.79	179

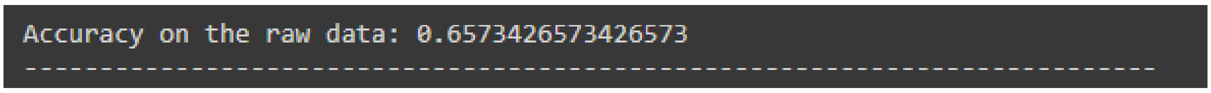
The Random Forest model showed an equal ability to classify, with precision, recall, and f1-scores staying similar for both classes, indicating a balanced performance in accurately recognizing and categorizing examples. In summary, the model's accuracy on the test data was approximately 79%, showing a strong ability to make accurate predictions on new data.

2.8.2. Model Training on Raw-Data

To know the impact of pre-processing on data, the RandomForestClassifier was employed to train the model using the raw data extracted from the Titanic dataset. The dataset was first filtered to include only numerical columns. These columns were chosen for their direct numerical nature, allowing for straightforward model training without the need for categorical encoding.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
import seaborn as sns


df_raw = sns.load_dataset('titanic')


# Selecting only numeric columns for simplicity as they don't need categorical encoding
numeric_cols = ['age', 'sibsp', 'parch', 'fare']
df_raw = df_raw[numeric_cols + ['survived']].dropna()
# Splitting the data
X_raw = df_raw.drop('survived', axis=1)
y_raw = df_raw['survived']
X_train_raw, X_test_raw, y_train_raw, y_test_raw = train_test_split(X_raw, y_raw, test_size=0.2, random_state=42)
clf_raw = RandomForestClassifier(random_state=42)
clf_raw.fit(X_train_raw, y_train_raw)
y_pred_raw = clf_raw.predict(X_test_raw)
accuracy_raw = accuracy_score(y_test_raw, y_pred_raw)
print(f"Accuracy on the raw data: {accuracy_raw}")


print("Classification Report for raw data:")
print(classification_report(y_test_raw, y_pred_raw))
```

LISTING 2. 12 RANDOM FOREST MODEL EVALUATION ON RAW DATA



```
Accuracy on the raw data: 0.6573426573426573
------------------------------------------------------------
```

FIGURE 2. 1 17 RANDOM FOREST MODEL ACCURACY RESULT ON TEST DATA

The model’s accuracy on the test data is 62.16%, showing a modest level of predictive power. There’s potential to improve this by exploring different feature engineering strategies as shown before, tuning the model’s parameters, or applying more complex algorithms to increase accuracy.

Classification Report:

TABLE 2- 2 CLASSIFICATION REPORT ON RAW DATA

Survived	precision	recall	f1-score	support
no	0.69	0.78	0.74	87
yes	0.58	0.46	0.51	56
accuracy			0.66	143
macro avg	0.64	0.62	0.62	143
weighted avg	0.65	0.66	0.65	143

The classification report shows that the model is more accurate at predicting non-survivors, with precision at 0.69 and recall 0.78 than with survivors, which have precision of 0.58 and recall 0.46, with an accuracy of 66%. Consequently, the F1-score is more balanced with non-survivors being higher at 0.74 than survivors, which has the F1-score of 0.51, suggesting better performance in the non-survivor classification.

2.9. Result Comparisons

As shown before, pre-processing has led to an overall better performing model, not just in terms of accuracy but also in the individual precision, recall, and F1-scores for both classes. It's clear that the data preparation steps contributed to enhancing the model's predictive power.

3 Conclusion

Data cleaning and feature selection which are used to pre-process the Titanic dataset, in addition to transformation, reduction, and normalization, led to a notable improvement in the model's predictive accuracy. As shown before, the model trained on the pre-processed data achieved a test accuracy of approximately 78%, up from 66% when using the raw data. Moreover, there was an improvement in the precision, recall, and F1-scores across both classes (survivors and non-survivors), highlighting the benefits of thorough data cleaning and feature engineering. These results validate the pre-processing steps taken, showing how they improve the model's capacity to make accurate predictions on new data by generalizing better. It's clear that such pre-processing steps are important in improving machine learning model performance, as evidenced by the case study's results.