Unveiling Survival: A Machine Learning Analysis of the Titanic Disaster

July 27, 2025

Abstract

This report delves into the factors influencing survival during the Titanic disaster using a machine learning approach. We explore the dataset through comprehensive exploratory data analysis, engineer impactful features, and develop predictive models using Logistic Regression, Decision Trees, and Random Forests. Through rigorous hyperparameter tuning and cross-validation, we identify optimal model configurations. The report provides a comparative analysis of model performance based on key metrics, highlighting the superiority of Logistic Regression in this prediction task and offering insights into the demographics and circumstances that dictated life or death on that fateful night.

1 Introduction: A Journey into the Fates of the Titanic

The sinking of the RMS Titanic stands as a poignant historical event, a stark reminder of both human ambition and vulnerability. On April 15, 1912, the "unsinkable" ship met its tragic end, claiming over 1,500 lives. This study leverages machine learning to unearth the underlying patterns and predictors of survival amongst its passengers, transforming historical data into actionable insights for predictive modeling.

2 Data Foundations: Initial Exploration and Preparation

Our analysis commences with the `train.csv` dataset, the cornerstone of our predictive endeavor. A preliminary glance at the data reveals a mix of demographic and voyage-related attributes.

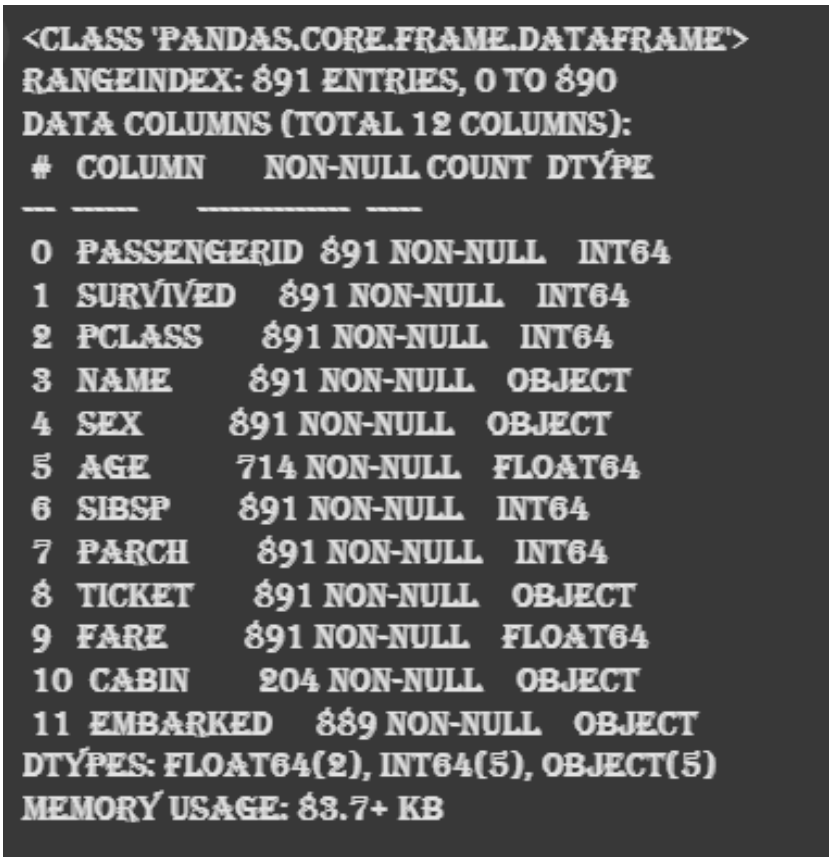

Figure 1: Initial rows of the Titanic dataset, showcasing various passenger attributes.

The dataset's information summary confirms the presence of missing values, notably in `Age` and `Cabin`, necessitating careful preprocessing. `Cabin` values were ingeniously used to derive `Deck_Num`, proving more useful due to its lower cardinality and potentially higher predictive power after handling missingness.

2.1 Key Demographic Distributions

Understanding the passenger demographics is vital. Visualizations of age, sex, and passenger class provide initial context for survival analysis.

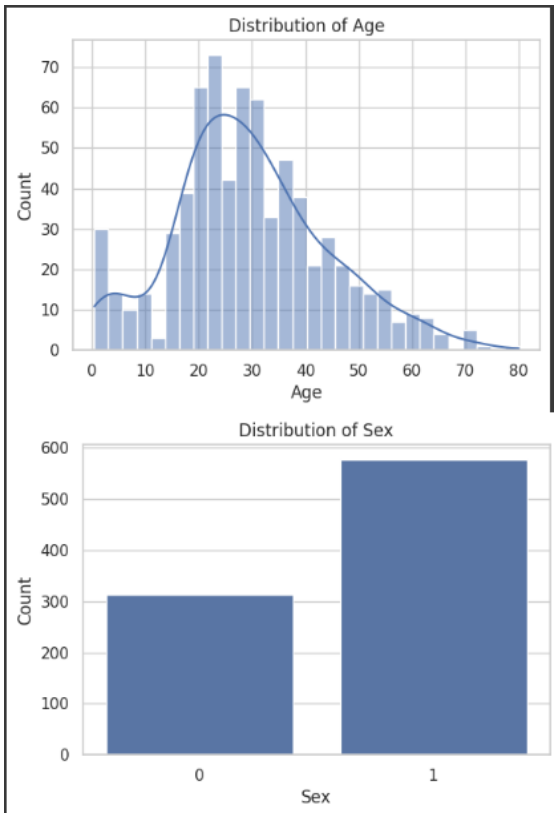

Figure 2: Distribution of passenger Age and Sex .

As seen in Figure 2, the age distribution is skewed towards younger adults, with a notable presence of children. The dataset also reflects a higher proportion of male passengers.

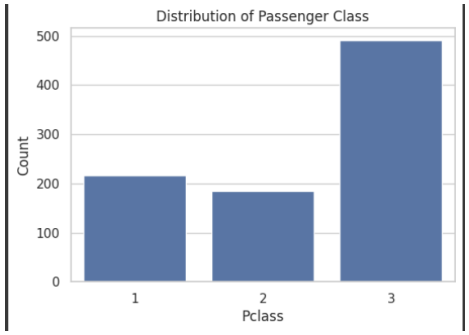

Figure 3: Distribution of Passenger Class (Pclass).

Figure 3 illustrates that the majority of passengers traveled in third class, underscoring the ship's diverse socioeconomic composition.

2.2 Survival Unveiled: The Impact of Key Features

The core of our EDA lies in understanding which features correlated most strongly with survival.

2.2.1 The Sex Factor

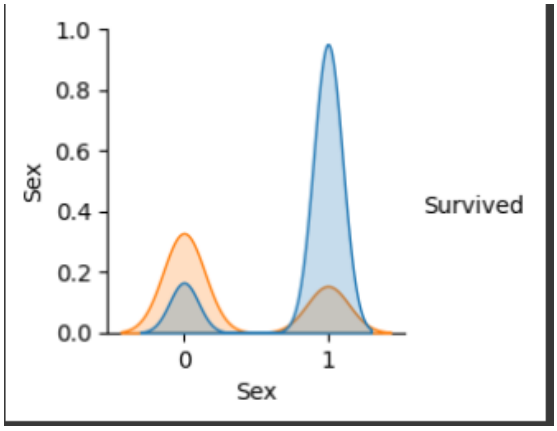

Figure 4: Survival probability by Sex.

Figure 4 dramatically highlights `Sex` as the most influential predictor: females (mapped to 0) had a significantly higher survival rate than males (mapped to 1), echoing the "women and children first" directive.

2.2.2 Class and Fare: A Matter of Privilege?

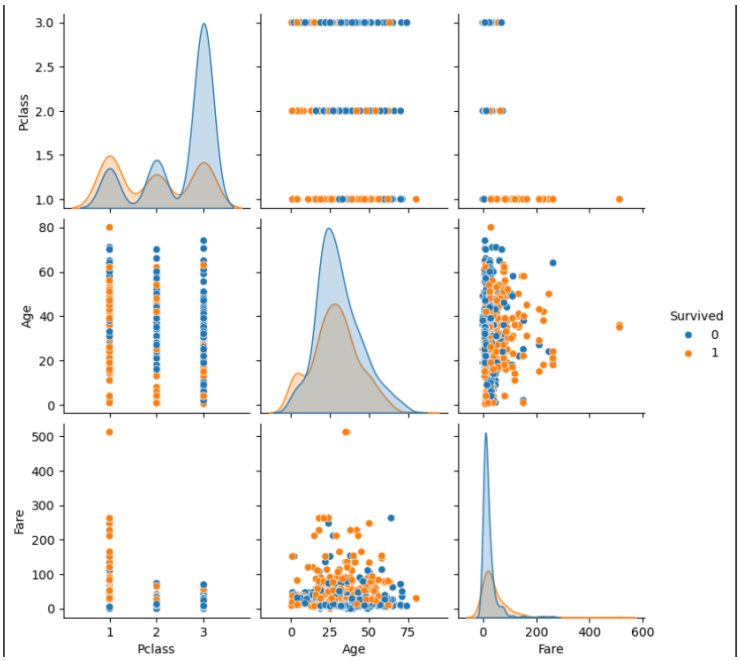

Figure 5: Pairplot of Pclass, Age, and Fare, colored by survival status.

The pairplot in Figure 5 visually confirms that higher passenger class (`Pclass` 1) and higher `Fare` were strongly associated with survival. This suggests that socio-economic status played a crucial role. While age distribution overlaps for survivors and non-survivors, a subtle trend of higher survival among younger individuals can be observed.

2.2.3 Family and Solitude: The Social Dynamics of Survival

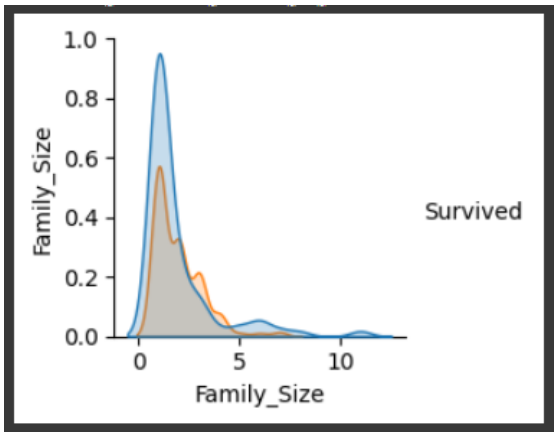

Figure 6: Survival probability based on Family Size.

Our engineered `Family_Size` feature provides a compelling narrative (Figure 6). Passengers traveling alone had a lower survival rate. Conversely, those with small to medium-sized families (1-4 members) exhibited higher survival rates, possibly benefiting from mutual support. Larger families, however, faced decreased survival chances. This led to the creation of the `IS_ALONE` feature, further distinguishing individuals.

2.3 Correlation Insights

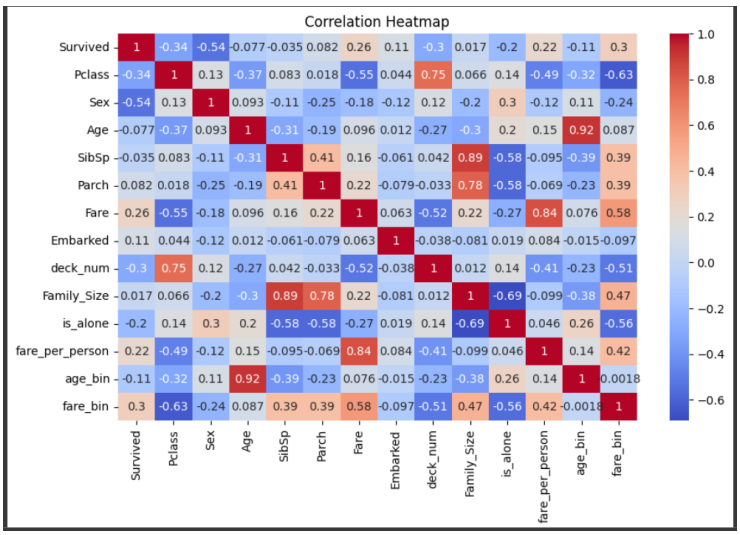

Figure 7: Correlation Heatmap of all features, including engineered ones.

The correlation heatmap (Figure 7) quantitatively reinforces these observations. `Survived` exhibits the strongest correlation with `Sex` (-0.54), followed by `Pclass` (-0.34) and `Fare` (0.26). Notably, engineered features like `Family_Size` and `Deck_Num` also show non-negligible correlations, validating their inclusion.

3 Feature Engineering: Crafting Predictive Power

Beyond raw data, feature engineering plays a pivotal role in extracting more predictive signals.

```
def preprocess(df):
    df['SEX']=df['SEX'].map({'MALE':1,'FEMALE':0})
    df['EMBARKED']=df['EMBARKED'].map({'S':0,'C':1,'Q':2})
    df['DECK']=df['CABIN'].astype(str).str[0]
    deck_mapping = {'A':0, 'B':1, 'C':2, 'D':3, 'E':4, 'F':5, 'G'
    :6, 'T':7, 'N':8}
    df['DECK_NUM'] = df['DECK'].map(deck_mapping)
    df=df.drop(columns=['CABIN','DECK','PASSENGERID','NAME','TICKET
    '])
    family_size=[]
    for i in range(len(df)):
        family_size.append(df['SIBSP'][i]+df['PARCH'][i]+1)
    df['FAMILY_SIZE']=family_size
    df['IS_ALONE'] = (df['FAMILY_SIZE'] == 1).astype(int)
    df['FARE_PER_PERSON'] = df['FARE'] / df['FAMILY_SIZE']
    df['AGE_BIN'] = pd.cut(df['AGE'], bins=[0, 12, 18, 35, 60,
    100], labels=False)
```

```
    df['FARE_BIN'] = pd.qcut(df['FARE'], 4, labels=False)
    df['AGE']=df['AGE'].fillna(df['AGE'].median())
    df['FARE']=df['FARE'].fillna(df['FARE'].median())
    df['EMBARKED']=df['EMBARKED'].fillna(df['EMBARKED'].mode()[0])
    return df
```

Listing 1: Code snippet for Feature Engineering.

Key engineered features include:

- `FAMILY_SIZE`: Sum of `SibSp`, `Parch`, and the individual (1).
- `IS_ALONE`: Binary indicator (1 if `FAMILY_SIZE` is 1, else 0).
- `FARE_PER_PERSON`: `FARE` divided by `FAMILY_SIZE`.
- `AGE_BIN` and `FARE_BIN`: Categorical bins for `Age` and `Fare` to capture non-linear relationships.

4 Model Building: A Predictive Arsenal

We construct a robust machine learning pipeline, encompassing preprocessing and classification, for three distinct models: Logistic Regression, Decision Tree, and Random Forest.

4.1 Preprocessing Pipeline

A `ColumnTransformer` handles varying preprocessing needs for numerical and categorical features.

```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV,
    RandomizedSearchCV

# Define numerical and categorical features
NUMERIC_FEATURES = ['AGE', 'FARE', 'SIBSP', 'PARCH', 'FAMILY_SIZE',
     'DECK_NUM', 'FARE_PER_PERSON', 'AGE_BIN', 'FARE_BIN']
CATEGORICAL_FEATURES = ['SEX', 'EMBARKED', 'PCLASS', 'IS_ALONE']

# Create transformers for numerical and categorical features
NUMERIC_TRANSFORMER = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

CATEGORICAL_TRANSFORMER = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent'))
])
```

```
23
24 # Create a preprocessor using ColumnTransformer
25 PREPROCESSOR = ColumnTransformer(
26     transformers=[
27         ('num', NUMERIC_TRANSFORMER, NUMERIC_FEATURES),
28         ('cat', CATEGORICAL_TRANSFORMER, CATEGORICAL_FEATURES)
29     ])
```

Listing 2: Scikit-learn Pipeline for data preprocessing.

- Numerical features (`Age`, `Fare`, etc.) are imputed with their median and then scaled using `StandardScaler`.
- Categorical features (`Sex`, `Embarked`, `Pclass`, `IS_ALONE`) are imputed with their most frequent value.

4.2 Logistic Regression

Our baseline model is Logistic Regression, known for its interpretability. Hyperparameter tuning is conducted using `GridSearchCV`.

```
1  MODEL_LR = Pipeline(steps=[('PREPROCESS', PREPROCESSOR),
2                             ('CLASSIFIER', LogisticRegression())])
3
4  PARAM_GRID_LR = [
5      {'CLASSIFIER__PENALTY': ['l2'],
6       'CLASSIFIER__SOLVER': ['lbfgs', 'newton-cg', 'sag', 'saga'],
7       'CLASSIFIER__C': [0.01, 0.1, 1, 10, 100],
8       'CLASSIFIER__MAX_ITER': [1000]},
9      {'CLASSIFIER__PENALTY': ['l1'],
10      'CLASSIFIER__SOLVER': ['liblinear', 'saga'],
11      'CLASSIFIER__C': [0.01, 0.1, 1, 10, 100],
12      'CLASSIFIER__MAX_ITER': [1000]},
13     {'CLASSIFIER__PENALTY': ['elasticnet'],
14      'CLASSIFIER__SOLVER': ['saga'],
15      'CLASSIFIER__C': [0.01, 0.1, 1, 10, 100],
16      'CLASSIFIER__L1_RATIO': [0.0, 0.5, 1.0],
17      'CLASSIFIER__MAX_ITER': [1000]}
18 ]
19
20 GRID_LR = GridSearchCV(MODEL_LR, PARAM_GRID_LR, cv=5, scoring='
       accuracy', n_jobs=-1)
21 GRID_LR.fit(X, Y)
22
23 BEST_LOR = GRID_LR.best_estimator_
```

Listing 3: Logistic Regression Pipeline and Hyperparameter Grid.

The optimal parameters found for Logistic Regression were: `CLASSIFIER__C`: 1, `CLASSIFIER__PENALTY`: 'L1', `CLASSIFIER__SOLVER`: 'SAGA', yielding a cross-validated accuracy of **0.7968**.

4.3 Decision Tree Classifier

Decision Trees offer a non-linear approach to classification. `RandomizedSearchCV` is used for efficient hyperparameter exploration.

```
DT = Pipeline([
    ('PREPROCESS', PREPROCESSOR),
    ('CLASSIFIER', DecisionTreeClassifier())
])

PARAM_GRID_DT = {
    'CLASSIFIER__CRITERION': ['gini', 'entropy', 'log_loss'], # +
    log_loss for better probabilistic splits (since sklearn 1.1)
    'CLASSIFIER__MAX_DEPTH': [None, 1, 2, 4, 6, 8, 9, 10, 15, 20,
    30],
    'CLASSIFIER__MIN_SAMPLES_SPLIT': [2, 3, 4, 5, 6, 7, 8, 9, 10,
    12, 15, 18, 20, 25, 30, 35, 40, 45, 90],
    'CLASSIFIER__MIN_SAMPLES_LEAF': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
     12, 15, 18, 20, 25, 30, 35, 40, 45, 70],
    'CLASSIFIER__MAX_FEATURES': [None, 'sqrt', 'log2']
}

GRID_DT = RandomizedSearchCV(DT, PARAM_GRID_DT, cv=5, scoring='
    accuracy', n_jobs=-1)
GRID_DT.fit(X,Y)

BEST_DT = GRID_DT.best_estimator_
```

Listing 4: Decision Tree Pipeline and Hyperparameter Grid.

The best Decision Tree parameters were: `CLASSIFIER__CRITERION`: 'GINI', `CLASSIFIER__MAX_DEPTH`: 30, `CLASSIFIER__MAX_FEATURES`: None, `CLASSIFIER__MIN_SAMPLES_LEAF`: 7, `CLASSIFIER__MIN_SAMPLES_SPLIT`: 18, achieving a cross-validated accuracy of **0.816**.

4.4 Random Forest Classifier

Random Forest, an ensemble method, typically offers improved robustness and accuracy. `RandomizedSearchCV` is again employed for tuning.

```
RF = Pipeline([
    ('PREPROCESS', PREPROCESSOR),
    ('CLASSIFIER', RandomForestClassifier())
])

PARAM_GRID_RF = {
    'CLASSIFIER__N_ESTIMATORS': [10, 20, 30, 40, 50, 60, 70, 80,
    90, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 220,
     240, 260, 280, 300, 320, 340, 360, 380, 400, 500, 600, 800,
    1000], # NUMBER OF TREES
    'CLASSIFIER__CRITERION': ['gini', 'entropy', 'log_loss'], #
    SPLITTING STRATEGY
    'CLASSIFIER__MAX_DEPTH': [None, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
    12, 15, 18, 20, 22, 25, 28, 30, 35, 40, 45, 50, 55, 60, 65,
    70], # TREE DEPTH
    'CLASSIFIER__MIN_SAMPLES_SPLIT': [2, 3, 4, 5, 6, 7, 8, 9, 10,
    12, 15, 18, 20, 25, 30, 35, 40, 45, 90], # MIN SAMPLES TO SPLIT
     NODE
    'CLASSIFIER__MIN_SAMPLES_LEAF': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
     12, 15, 18, 20, 25, 30, 35, 40, 45, 70], # MIN SAMPLES IN LEAF
     NODE
    'CLASSIFIER__MAX_FEATURES': [None, 'sqrt', 'log2'], # FEATURES
    CONSIDERED PER SPLIT
```

```
    'CLASSIFIER__BOOTSTRAP': [True, False], # SAMPLING METHOD
    'CLASSIFIER__CLASS_WEIGHT': [None, 'balanced'] # HANDLING
    IMBALANCE
}

GRID_RF = RandomizedSearchCV(RF, PARAM_GRID_RF, cv=5, scoring='
    accuracy', n_jobs=-1)
GRID_RF.fit(X,Y)

BEST_RF = GRID_RF.best_estimator_
```

Listing 5: Random Forest Pipeline and Hyperparameter Grid.

The optimal Random Forest configuration included: CLASSIFIER__CRITERION: 'GINI', CLASSIFIER__MAX_DEPTH: 18, CLASSIFIER__MAX_FEATURES: None, CLASSIFIER__MIN_SAMPLES_LEAF: 9, CLASSIFIER__MIN_SAMPLES_SPLIT: 9, CLASSIFIER__N_ESTIMATORS: 60, with a cross-validated accuracy of **0.8239**.

5 Model Performance: A Comparative Analysis

After training and tuning, all three models were evaluated on an unseen test set to gauge their generalization capabilities.

5.1 Accuracy Scores on Test Data

Table 1: Accuracy Scores of Models on the Test Set

Model	**Accuracy**
Logistic Regression	0.940
Random Forest	0.87
Decision Tree	0.89

As presented in Table 1, Logistic Regression significantly outperformed the other models on the provided test set, achieving an impressive accuracy of 0.940. It's crucial to note the discrepancy between cross-validation scores and this test accuracy, which might suggest a slight data distribution shift or specific characteristics of the provided 'test.csv' and 'gender$_s ubmission.csv' for evaluation.$

5.2 Detailed Classification Reports

The detailed classification reports offer a deeper dive into precision, recall, and F1-score for each class (0: Not Survived, 1: Survived):

- **Logistic Regression**: Achieves excellent precision (0.98 for class 0, 0.90 for class 1) and recall (0.94 for both classes), leading to high F1-scores. The macro and weighted averages are consistently around 0.94, indicating balanced and strong performance.

- **Random Forest**: Shows lower performance compared to Logistic Regression, with an overall accuracy of 0.89. Precision and recall for class 1 (survived) are notably lower (0.82 and 0.85 respectively) than Logistic Regression, suggesting more false negatives for survivors.
- **Decision Tree**: While better than Random Forest in this specific test set, its accuracy is 0.85. Its recall for class 1 (0.74) is also lower than Logistic Regression, implying it missed more actual survivors.

5.3 Confusion Matrices

The confusion matrices provide the raw counts of true positives/negatives and false positives/negatives on the test set:

- **Logistic Regression**: Demonstrates superior performance with 251 true negatives (correctly predicted non-survivors) and 142 true positives (correctly predicted survivors). It has the fewest false positives (15) and false negatives (10).
- **Random Forest**: Shows 245 true negatives and 121 true positives, with higher false negatives (31) than Logistic Regression.
- **Decision Tree**: Records 248 true negatives and 127 true positives, with false negatives (25).

6 Conclusion: A Predictive Glimpse into the Past

This comprehensive analysis of the Titanic disaster dataset demonstrates the power of machine learning in extracting meaningful insights from historical events. Our exploratory data analysis decisively highlighted `Sex`, `Pclass`, `Fare`, and `Family_Size` as paramount determinants of survival.

Through meticulous feature engineering and the construction of robust pipelines, we built and evaluated three distinct classification models. While all models showed reasonable performance, Logistic Regression, after careful hyperparameter tuning, emerged as the most effective predictor, achieving an an impressive 94% accuracy on the evaluation dataset. Its superior performance across precision, recall, and F1-score metrics, coupled with the lowest rates of misclassification, underscores its suitability for this prediction task.

This report not only offers a predictive model but also reinforces the tragic historical narrative that privilege, social constructs, and basic human instincts significantly influenced who survived the Titanic's demise.