DIGITAL ASSIGNMENT 1

MACHINE LEARNING MODELS TO PREDCIT THE SURVIVAL OF PASSENGERS IN TITANIC

MEIC501L MACHINE LEARNING FOR COMMUNICATIONS

DONE BY:

DURGA SITALAKSHMI S

Registration number :24MEC0024

Mail ID: durgasitalakshmi.s2024@vitstudent.ac.in

Phone number: +91 9731850288

Aim:

The aim is to predict the survival of passengers on the Titanic using various machine learning models.

Objectives:

1. **Data Preprocessing:**

- Handle missing data by imputing or removing incomplete entries.
- Convert categorical data into numerical form for compatibility with machine learning models.
- Ensure the data is properly scaled or normalized where necessary.

2. **Feature Engineering:**

- Identify and select relevant features that could influence survival, such as age, gender, ticket class, and embarked location.
- Generate new features, if necessary, to improve model performance.

3. **Model Implementation:**

- Implement and train the following machine learning models on the processed dataset:
  - Naive Bayes
  - Decision Tree
  - Random Forest
  - K-Nearest Neighbors (KNN)
  - Support Vector Machine (SVM)

4. **Model Evaluation:**

- Evaluate the performance of each model using key metrics such as accuracy, precision, recall, F1-score, and confusion matrix.
- Compare the effectiveness of the different models based on these metrics to identify the best-performing model.

5. **Prediction and Analysis:**

- Use the trained models to predict the survival of passengers in the test dataset.
- Analyse and interpret the results to understand which features most significantly contributed to the survival prediction.

1. ABOUT THE DATASET

The Titanic dataset, sourced from the infamous sinking of the RMS Titanic in 1912, is one of the most well-known datasets in the data science and machine learning community. The disaster, which led to the loss of over 1,500 lives, has been extensively analyzed, and this dataset captures key details about the passengers who were on board.

Historical Context:

On April 10, 1912, the RMS Titanic, one of the largest and most luxurious ships at the time, set sail from Southampton, England, on its maiden voyage to New York City. On the night of April 14, the ship struck an iceberg in the North Atlantic Ocean and sank in the early hours of April 15. The limited number of lifeboats, combined with chaotic evacuation procedures, led to a high fatality rate, particularly among certain demographics.

Attributes of the Titanic Dataset:

1. **PassengerId**: A unique identifier for each passenger.
2. **Survived**: Binary variable indicating whether the passenger survived (1) or not (0). This is the target variable.
3. **Pclass**: Ticket class of the passenger, indicating socio-economic status:

- 1 = First Class (Upper)
- 2 = Second Class (Middle)
- 3 = Third Class (Lower)

4. **Name**: Full name of the passenger.
5. **Sex**: Gender of the passenger (male or female).
6. **Age**: Age of the passenger in years. Some entries may have missing values.
7. **SibSp**: Number of siblings or spouses the passenger had aboard the Titanic.
8. **Parch**: Number of parents or children the passenger had aboard the Titanic.
9. **Ticket**: Ticket number of the passenger.
10. **Fare**: Amount of money the passenger paid for the ticket.
11. **Cabin**: Cabin number assigned to the passenger. This attribute may have many missing values.
12. **Embarked**: Port of embarkation, indicating where the passenger boarded the Titanic:

- C = Cherbourg
- Q = Queenstown
- S = Southampton

2. THE CODE

1) IMPORTING LIBRARIES

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

2) LOADING DATA

```
train_data = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")
```

3) EXPLORATORY DATA ANALYSIS (EDA)

```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

```
pd.pivot_table(train_data, index="Survived", values=["Age", "SibSp", "Parch", "Fare"])
```

```python
x = pd.DataFrame(
    (
        pd.pivot_table(
            train_data,
            index="Survived",
            columns="Sex",
            values="Ticket",
            aggfunc="count",
        )
    )
)
print()
print(
    pd.pivot_table(
        train_data, index="Survived", columns="Pclass", values="Ticket", aggfunc="count"
    )
)
print()
print(
    pd.pivot_table(
        train_data,
        index="Survived",
        columns="Embarked",
        values="Ticket",
        aggfunc="count",
    )
)
print()
x
```

```
Pclass      1   2    3
Survived
0          80  97  372
1         136  87  119

Embarked   C   Q    S
Survived
0         75  47  427
1         93  30  217


[9]:
```

Sex	female	male
Survived		
0	81	468
1	233	109

4) DATA CLEANING

```
train_data.isnull().sum()
```

```
[10]:

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
```

```
train_data = train_data.drop(columns=["PassengerId", "Cabin", "Name", "Ticket"])
```

```
[12]:
```

```
train_data["Age"] = train_data["Age"].fillna(train_data["Age"].mean())
```

```
[13]:
```

```
train_data.isnull().sum()
```

```
[13]:

Survived    0
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    2
dtype: int64
```

5) FEATURE ENGINEERING

```
from sklearn.preprocessing import LabelEncoder

cols = ["Sex", "Embarked"]
le = LabelEncoder()
for col in cols:
    train_data[col] = le.fit_transform(train_data[col])

train_data.head()
```

[17]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	1	22.0	1	0	2.110213	2
1	1	1	0	38.0	1	0	4.280593	0
2	1	3	0	26.0	0	0	2.188856	2
3	1	1	0	35.0	1	0	3.990834	2
4	0	3	1	35.0	0	0	2.202765	2

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

[18]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	1	22.000000	1	0	2.110213	2
1	1	1	0	38.000000	1	0	4.280593	0
2	1	3	0	26.000000	0	0	2.188856	2
3	1	1	0	35.000000	1	0	3.990834	2
4	0	3	1	35.000000	0	0	2.202765	2
...	...	...	...	...	...	...	...	...
886	0	2	1	27.000000	0	0	2.639057	2
887	1	1	0	19.000000	0	0	3.433987	2
888	0	3	0	29.699118	1	2	3.196630	2
889	1	1	1	26.000000	0	0	3.433987	0
890	0	3	1	32.000000	0	0	2.169054	1

6) SETTING UP THE PARAMETERS FOR THE MODEL

```
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from tabulate import tabulate
import numpy as np

def classify(model):
    x_train, x_test, y_train, y_test = train_test_split(
        X, y, test_size=0.25, random_state=40
    )
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)

    # Calculate Accuracy
    accuracy = model.score(x_test, y_test)

    # Generate Classification Report
    report = classification_report(y_test, y_pred, output_dict=True)

    # Generate Confusion Matrix
    conf_matrix = confusion_matrix(y_test, y_pred)

    # Prepare Data for Tabulate
    table = []
    table.append(["Accuracy", "", "", f"{accuracy:.2f}", len(y_test)])

    for label in ['0', '1']:
        precision = report[label]['precision']
        recall = report[label]['recall']
        f1_score = report[label]['f1-score']
        support = report[label]['support']
        table.append([label, f"{precision:.2f}", f"{recall:.2f}", f"{f1_score:.2f}", support])

    # Macro Average
    macro_avg = report['macro avg']
    table.append(["Macro average", f"{macro_avg['precision']:.2f}", f"{macro_avg['recall']:.2f}", f"{macro_avg['f1-score']:.2f}", len(y_test)])

    # Weighted Average
    weighted_avg = report['weighted avg']
    table.append(["Weighted average", f"{weighted_avg['precision']:.2f}", f"{weighted_avg['recall']:.2f}", f"{weighted_avg['f1-score']:.2f}", len(y_tes

    # Print the Table
    print(tabulate(table, headers=["", "Precision", "Recall", "F1-score", "Support"], tablefmt="grid"))

    # Print Accuracy
    print(f"\nAccuracy: {accuracy:.2f}")

    # Print Confusion Matrix
    print("\nConfusion Matrix:")
    print(conf_matrix)

    # Cross Validation Score
    score = cross_val_score(model, X, y, cv=5)
    print("\nCV SCORE:", np.mean(score))
```

7) NAÏVE BAYES CLASSIFIER

```
def classify_naive_bayes():
    model_nb = GaussianNB()
    classify(model_nb)

classify_naive_bayes()
```

```
-------------------+-------------+----------+------------+-----------+
                   | Precision   | Recall   |  F1-score  |  Support  |
===================+=============+==========+============+===========+
 Accuracy          |             |          |       0.79 |       223 |
-------------------+-------------+----------+------------+-----------+
 0                 | 0.86        | 0.76     |       0.8  |       128 |
-------------------+-------------+----------+------------+-----------+
 1                 | 0.72        | 0.83     |       0.77 |        95 |
-------------------+-------------+----------+------------+-----------+
 Macro average     | 0.79        | 0.79     |       0.79 |       223 |
-------------------+-------------+----------+------------+-----------+
 Weighted average  | 0.80        | 0.79     |       0.79 |       223 |
-------------------+-------------+----------+------------+-----------+

Accuracy: 0.79

Confusion Matrix:
[[97 31]
 [16 79]]

CV SCORE: 0.7677233067604042
```

8) KNN

```
# Example usage for K-Nearest Neighbors
def classify_knn(n_neighbors=5):
    model_knn = KNeighborsClassifier(n_neighbors=n_neighbors)
    classify(model_knn)

classify_knn(n_neighbors=5)
```

```
+-------------------+-------------+----------+------------+-----------+
|                   | Precision   | Recall   |  F1-score  |  Support  |
+===================+=============+==========+============+===========+
| Accuracy          |             |          |       0.75 |       223 |
+-------------------+-------------+----------+------------+-----------+
| 0                 | 0.75        | 0.86     |       0.8  |       128 |
+-------------------+-------------+----------+------------+-----------+
| 1                 | 0.76        | 0.61     |       0.68 |        95 |
+-------------------+-------------+----------+------------+-----------+
| Macro average     | 0.76        | 0.73     |       0.74 |       223 |
+-------------------+-------------+----------+------------+-----------+
| Weighted average  | 0.75        | 0.75     |       0.75 |       223 |
+-------------------+-------------+----------+------------+-----------+

Accuracy: 0.75

Confusion Matrix:
[[110  18]
 [ 37  58]]

CV SCORE: 0.7744397715146569
```

9) RANDOM FOREST CLASSIFIER

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()
classify(model)
```

```
+------------------+-------------+----------+------------+-----------+
|                  | Precision   | Recall   |   F1-score |   Support |
+==================+=============+==========+============+===========+
| Accuracy         |             |          |       0.81 |       223 |
+------------------+-------------+----------+------------+-----------+
| 0                | 0.80        | 0.89     |       0.84 |       128 |
+------------------+-------------+----------+------------+-----------+
| 1                | 0.82        | 0.69     |       0.75 |        95 |
+------------------+-------------+----------+------------+-----------+
| Macro average    | 0.81        | 0.79     |       0.8  |       223 |
+------------------+-------------+----------+------------+-----------+
| Weighted average | 0.81        | 0.81     |       0.8  |       223 |
+------------------+-------------+----------+------------+-----------+

Accuracy: 0.81

Confusion Matrix:
[[114  14]
 [ 29  66]]

CV SCORE: 0.8159500345238844
```

10) SUPPORT VECTOR MACHINE

```
from sklearn.svm import SVC
from sklearn.datasets import make_classification

# Create a dataset
X, y = make_classification(n_samples=100, n_features=10, random_state=42)

# Initialize the SVM model
svm_model = SVC()

# Call the classify function with SVM model
classify(svm_model)
```

```
+------------------+-------------+----------+------------+-----------+
|                  | Precision   | Recall   |   F1-score |   Support |
+==================+=============+==========+============+===========+
| Accuracy         |             |          |       0.84 |        25 |
+------------------+-------------+----------+------------+-----------+
| 0                | 0.93        | 0.82     |       0.88 |        17 |
+------------------+-------------+----------+------------+-----------+
| 1                | 0.70        | 0.88     |       0.78 |         8 |
+------------------+-------------+----------+------------+-----------+
| Macro average    | 0.82        | 0.85     |       0.83 |        25 |
+------------------+-------------+----------+------------+-----------+
| Weighted average | 0.86        | 0.84     |       0.84 |        25 |
+------------------+-------------+----------+------------+-----------+

Accuracy: 0.84

Confusion Matrix:
[[14  3]
 [ 1  7]]

CV SCORE: 0.93
```

11) DATA CLEANING FOR TESTING DATASET

```
[26]: X_test = test.drop(columns=["PassengerId", "Name", "Cabin", "Ticket"], axis=1)

      X_test
```

[26]:

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	3	male	34.5	0	0	7.8292	Q
1	3	female	47.0	1	0	7.0000	S
2	2	male	62.0	0	0	9.6875	Q
3	3	male	27.0	0	0	8.6625	S
4	3	female	22.0	1	1	12.2875	S
...	...	...	...	...	...	...	...
413	3	male	NaN	0	0	8.0500	S
414	1	female	39.0	0	0	108.9000	C
415	3	male	38.5	0	0	7.2500	S
416	3	male	NaN	0	0	8.0500	S
417	3	male	NaN	1	1	22.3583	C

418 rows × 7 columns

```
[27]: from sklearn.preprocessing import LabelEncoder

      cols = ["Sex", "Embarked"]
      le = LabelEncoder()

      for col in cols:
          X_test[col] = le.fit_transform(X_test[col])

      X_test.head()
```

[27]:

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	3	1	34.5	0	0	7.8292	1
1	3	0	47.0	1	0	7.0000	2
2	2	1	62.0	0	0	9.6875	1
3	3	1	27.0	0	0	8.6625	2
4	3	0	22.0	1	1	12.2875	2

```
X_test["Age"] = X_test["Age"].fillna(X_test["Age"].mean())
X_test["Fare"] = X_test["Fare"].fillna(X_test["Fare"].mean())

X_test.isnull().sum()
```

```
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    0
dtype: int64
```

```python
X_test = test.drop(columns=["PassengerId", "Name", "Cabin", "Ticket"], axis=1)

X_test["Age"] = X_test["Age"].fillna(X_test["Age"].mean())
X_test["Fare"] = X_test["Fare"].fillna(X_test["Fare"].mean())

X_test.isnull().sum()

from sklearn.preprocessing import LabelEncoder

cols = ["Sex", "Embarked"]
le = LabelEncoder()

for col in cols:
    X_test[col] = le.fit_transform(X_test[col])

X_test.head()
X_test
```

[29]:

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	3	1	34.50000	0	0	7.8292	1
1	3	0	47.00000	1	0	7.0000	2
2	2	1	62.00000	0	0	9.6875	1
3	3	1	27.00000	0	0	8.6625	2
4	3	0	22.00000	1	1	12.2875	2
...	...	...	...	...	...	...	...

12) TESTING PARAMETERS

```python
pred = model.predict(X_test)
pred
```

[30]:

```
array([1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0,
       1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1,
       1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1,
       1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0,
       0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1,
       1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
       0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1,
       1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1,
       0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1,
       1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1,
       0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0,
       1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0,
       0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0],
      dtype=int64)
```

```
submit = pd.read_csv("gender_submission.csv")
```

[32]:

```
submit["Survived"] = pred

submit
```

[32]:

	PassengerId	Survived
0	892	1
1	893	0
2	894	0
3	895	1
4	896	0
...	...	...
413	1305	1
414	1306	1
415	1307	1
416	1308	1
417	1309	0

418 rows × 2 columns

3.THE MATHEMATICAL MODEL FOR RANDOM FOREST

ABOUT RANDOM FOREST

Random Forest is a popular machine learning algorithm that belongs to the ensemble learning family.

It's built on the concept of decision trees, but it combines multiple decision trees to improve predictive accuracy and control overfitting.

How Random Forest Works

1. **Decision Trees:** Random Forest creates multiple decision trees. Each tree is built on a random subset of the data (with replacement, known as bootstrapping) and a random subset of features.
2. **Ensemble:** The algorithm combines the predictions from all these individual trees to make a final prediction. For classification, the most frequent class is chosen, and for regression, the average prediction is taken.

The Mathematics Behind Random Forest

Regression Problems

When using the Random Forest Algorithm to solve regression problems, you are using the mean squared error (MSE) to how your data branches from each node.

$$MSE = \frac{1}{N} \sum_{i=1}^{N}(fi - yi)^2$$

Where *N* is the number of data points, *fi* is the value returned by the model and *yi* is the actual value for data point *i*.

This formula calculates the distance of each node from the predicted actual value, helping to decide which branch is the better decision for your forest. Here, *yi* is the value of the data point you are testing at a certain node and *fi* is the value returned by the decision tree.

Classification Problems

When performing Random Forests based on classification data, you should know that you are often using the Gini index, or the formula used to decide how nodes on a decision tree branch.

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

This formula uses the class and probability to determine the Gini of each branch on a node, determining which of the branches is more likely to occur. Here, *pi* represents the relative frequency of the class you are observing in the dataset and *c* represents the number of classes.

You can also use entropy to determine how nodes branch in a decision tree.

$$Entropy = \sum_{i=1}^{C} -p_i * \log_2(p_i)$$

Entropy uses the probability of a certain outcome in order to make a decision on how the node should branch. Unlike the Gini index, it is more mathematical intensive due to the logarithmic function used in calculating it.

CALCULATION OF RANDOM FOREST

Step 1: Select Features and Data

For simplicity, let's consider the following features:

- Pclass (Ticket class: 1, 2, 3)

- Sex (Gender: Male = 0, Female = 1)
- Age (Age of the passenger)

We'll use the following sample data for this tree:

Passenger Name	Pclass	Sex	Age	Survived (Label)
Nasser	2	1	14	1
Saundercock	3	0	20	0
Vestrom	3	1	14	0
Hewlett	2	1	55	1
McGowan	3	1	15	1

Step 2: Splitting Criteria

A decision tree splits the data based on the feature that provides the best separation between classes. For simplicity, we'll use a basic splitting rule:

- First split on Sex: if the passenger is female (Sex = 1), check further splits; otherwise, the passenger is male and may have a different survival probability.

Step 3: Build the Tree

Let's construct a very basic tree:

1. Root Node: Split based on Sex:
   - If Sex = 1 (Female), go to the next level.
   - If Sex = 0 (Male), predict Survived = 0.
2. Next Level (Females Only):
   - Split based on Pclass:
     - If Pclass = 1 or 2, predict Survived = 1.
     - If Pclass = 3, go to the next level.
3. Next Level (Pclass = 3, Females Only):
   - Split based on Age:
     - If Age < 16, predict Survived = 1.
     - If Age ≥ 16, predict Survived = 0.

Step 4: Make Predictions Using the Tree

	A	B	C	D	E
1	**Passenger**	**Tree 1**	**Tree 2**	**Tree 3**	**Final Prediction (Majority Vote)**
2	Nasser	1	1	1	1
3	Sandstrom	1	0	1	1
4	Bonnell	1	1	1	1
5	Saundercock	0	0	0	0
6	Andersson	0	0	0	0
7	Vestrom	0	0	0	0
8	Hewlett	1	1	1	1
9	Rice	0	0	0	0
10	Williams	1	1	1	1
11	Vander Planke	0	0	0	0
12	Masselmani	1	1	1	1
13	Fynney	0	0	0	0
14	Beesley	1	1	1	1
15	McGowan	1	0	1	1

Now, let's use this decision tree to predict the survival of each passenger:

1. Nasser:
    - Sex = 1 (Female)
    - Pclass = 2 → Predict Survived = 1
2. Saundercock:
    - Sex = 0 (Male) → Predict Survived = 0
3. Vestrom:
    - Sex = 1 (Female)
    - Pclass = 3

- Age = 14 → Predict Survived = 1

4. Hewlett:
   - Sex = 1 (Female)
   - Pclass = 2 → Predict Survived = 1
5. McGowan:
   - Sex = 1 (Female)
   - Pclass = 3
   - Age = 15 → Predict Survived = 1

Final Predictions from This Tree

Based on the tree logic:

- Nasser: Survived
- Saundercock: Did not survive
- Vestrom: Survived
- Hewlett: Survived
- McGowan: Survived