NAME: JITESH VINOD ZOPE

ROLL NO.: 69

PID: 246054

BATCH: 3

TITLE: IMPLEMENTATION OF AN END-TO-END MACHINE LEARNING DATA PIPELINE

In [58]:
```
# pip install numpy
# pip install pandas
# pip install matplotlib
!pip install numpy scikit-learn
!pip install numpy seaborn
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: numpy in c:\users\hp\appdata\roaming\python\python39
\site-packages (2.0.2)
Requirement already satisfied: scikit-learn in c:\users\hp\appdata\roaming\python\py
thon39\site-packages (1.6.1)
Requirement already satisfied: joblib>=1.2.0 in c:\users\hp\appdata\roaming\python\p
ython39\site-packages (from scikit-learn) (1.5.3)
Requirement already satisfied: scipy>=1.6.0 in c:\users\hp\appdata\roaming\python\py
thon39\site-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\hp\appdata\roaming\p
ython\python39\site-packages (from scikit-learn) (3.6.0)
WARNING: You are using pip version 20.2.3; however, version 25.3 is available.
You should consider upgrading via the 'c:\program files\python39\python.exe -m pip i
nstall --upgrade pip' command.
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: numpy in c:\users\hp\appdata\roaming\python\python39
\site-packages (2.0.2)
Requirement already satisfied: seaborn in c:\users\hp\appdata\roaming\python\python3
9\site-packages (0.13.2)
Requirement already satisfied: pandas>=1.2 in c:\users\hp\appdata\roaming\python\pyt
hon39\site-packages (from seaborn) (2.3.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in c:\users\hp\appdata\roamin
g\python\python39\site-packages (from seaborn) (3.9.4)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\hp\appdata\roaming
\python\python39\site-packages (from pandas>=1.2->seaborn) (2.9.0.post0)
Requirement already satisfied: tzdata>=2022.7 in c:\users\hp\appdata\roaming\python
\python39\site-packages (from pandas>=1.2->seaborn) (2025.3)
Requirement already satisfied: pytz>=2020.1 in c:\users\hp\appdata\roaming\python\py
thon39\site-packages (from pandas>=1.2->seaborn) (2025.2)
Requirement already satisfied: pillow>=8 in c:\users\hp\appdata\roaming\python\pytho
n39\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (11.3.0)
Requirement already satisfied: cycler>=0.10 in c:\users\hp\appdata\roaming\python\py
thon39\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\hp\appdata\roaming\pyth
on\python39\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (4.59.1)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\hp\appdata\roaming\pytho
n\python39\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (3.2.3)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\hp\appdata\roaming\pytho
n\python39\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.3.0)
Requirement already satisfied: importlib-resources>=3.2.0; python_version < "3.10" i
n c:\users\hp\appdata\roaming\python\python39\site-packages (from matplotlib!=3.6.1,
>=3.4->seaborn) (6.5.2)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\hp\appdata\roaming\pyth
on\python39\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.4.7)
Requirement already satisfied: packaging>=20.0 in c:\users\hp\appdata\roaming\python
\python39\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (25.0)
Requirement already satisfied: six>=1.5 in c:\users\hp\appdata\roaming\python\python
39\site-packages (from python-dateutil>=2.8.2->pandas>=1.2->seaborn) (1.17.0)
Requirement already satisfied: zipp>=3.1.0; python_version < "3.10" in c:\users\hp\a
ppdata\roaming\python\python39\site-packages (from importlib-resources>=3.2.0; pytho
n_version < "3.10"->matplotlib!=3.6.1,>=3.4->seaborn) (3.23.0)
WARNING: You are using pip version 20.2.3; however, version 25.3 is available.
You should consider upgrading via the 'c:\program files\python39\python.exe -m pip i
nstall --upgrade pip' command.
```

In [59]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [60]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
```

In [61]:
```
import seaborn as sns
#Load titanic dataset
titanic_data = sns.load_dataset('titanic')
```

In [62]:
```
print(titanic_data.shape)
```

```
(891, 15)
```

In [63]:
```
print(titanic_data.columns)
```

```
Index(['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'fare',
       'embarked', 'class', 'who', 'adult_male', 'deck', 'embark_town',
       'alive', 'alone'],
      dtype='object')
```

In [64]:
```
print(titanic_data.head())
```

```
   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
```

In [65]:
```
print(titanic_data.tail())
```

```
     survived  pclass     sex   age  sibsp  parch   fare embarked   class  \
886         0       2    male  27.0      0      0  13.00        S  Second
887         1       1  female  19.0      0      0  30.00        S   First
888         0       3  female   NaN      1      2  23.45        S   Third
889         1       1    male  26.0      0      0  30.00        C   First
890         0       3    male  32.0      0      0   7.75        Q   Third

       who  adult_male deck  embark_town alive  alone
886    man        True  NaN  Southampton    no   True
887  woman       False    B  Southampton   yes   True
888  woman       False  NaN  Southampton    no  False
889    man        True    C    Cherbourg   yes   True
890    man        True  NaN   Queenstown    no   True
```

In [66]:
```
#Review the structure of the dataset
print(titanic_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype   
---  ------       --------------  -----   
 0   survived     891 non-null    int64   
 1   pclass       891 non-null    int64   
 2   sex          891 non-null    object  
 3   age          714 non-null    float64 
 4   sibsp        891 non-null    int64   
 5   parch        891 non-null    int64   
 6   fare         891 non-null    float64 
 7   embarked     889 non-null    object  
 8   class        891 non-null    category
 9   who          891 non-null    object  
 10  adult_male   891 non-null    bool    
 11  deck         203 non-null    category
 12  embark_town  889 non-null    object  
 13  alive        891 non-null    object  
 14  alone        891 non-null    bool    
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.7+ KB
None
```

In [67]:
```
print(titanic_data.describe())
```

```
        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
```

In [68]:
```
#Identifying missing values
missing_values = titanic_data.isnull().sum()
print(missing_values)
```

```
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
```

In [69]:
```
#Dealing with missing values
#Dropping columns with excessive missing data
new_titanic_df = titanic_data.drop(columns=['deck'])
```

In [70]:
```
#Imputing median age for missing age data
new_titanic_df['age'].fillna(new_titanic_df['age'].median(),inplace=True)
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_5248\1960786904.py:2: FutureWarning: A valu
e is trying to be set on a copy of a DataFrame or Series through chained assignment
using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  new_titanic_df['age'].fillna(new_titanic_df['age'].median(),inplace=True)
```

In [71]:
```
missing_values = new_titanic_df.isnull().sum()
print(missing_values)
```

```
survived       0
pclass         0
sex            0
age            0
sibsp          0
parch          0
fare           0
embarked       2
class          0
who            0
adult_male     0
embark_town    2
alive          0
alone          0
dtype: int64
```

In [72]:
```
data = new_titanic_df
data['embark_town'].dtype
data['embark_town'].unique()
data['embark_town'].fillna(data['embark_town'].mode()[0] , inplace=True)
data.isnull().sum()
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_5248\1968151471.py:4: FutureWarning: A valu
e is trying to be set on a copy of a DataFrame or Series through chained assignment
using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  data['embark_town'].fillna(data['embark_town'].mode()[0] , inplace=True)
```

Out[72]:
```
survived       0
pclass         0
sex            0
age            0
sibsp          0
parch          0
fare           0
embarked       2
class          0
who            0
adult_male     0
embark_town    0
alive          0
alone          0
dtype: int64
```

In [73]:
```
data = new_titanic_df
data['embarked'].dtype
data['embarked'].unique()
data['embarked'].fillna(data['embarked'].mode()[0] , inplace=True)
data.isnull().sum()
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_5248\2633234362.py:4: FutureWarning: A valu
e is trying to be set on a copy of a DataFrame or Series through chained assignment
using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  data['embarked'].fillna(data['embarked'].mode()[0] , inplace=True)
```

Out[73]:
```
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
```

In [74]:
```
# Step 4 : Encode catagorical variables
le = LabelEncoder()
data['sex'] = le.fit_transform(data['sex'])
data['embarked'] = le.fit_transform(data['embarked'])
```

In [75]:
```
data.head()
```

Out[75]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male
0	0	3	1	22.0	1	0	7.2500	2	Third	man	True
1	1	1	0	38.0	1	0	71.2833	0	First	woman	False
2	1	3	0	26.0	0	0	7.9250	2	Third	woman	False
3	1	1	0	35.0	1	0	53.1000	2	First	woman	False
4	0	3	1	35.0	0	0	8.0500	2	Third	man	True

In [76]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 14 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   survived     891 non-null    int64
 1   pclass       891 non-null    int64
 2   sex          891 non-null    int64
 3   age          891 non-null    float64
 4   sibsp        891 non-null    int64
 5   parch        891 non-null    int64
 6   fare         891 non-null    float64
 7   embarked     891 non-null    int64
 8   class        891 non-null    category
 9   who          891 non-null    object
 10  adult_male   891 non-null    bool
 11  embark_town  891 non-null    object
 12  alive        891 non-null    object
 13  alone        891 non-null    bool
dtypes: bool(2), category(1), float64(2), int64(6), object(3)
memory usage: 79.4+ KB
```

In [77]:
```
#Step 5 : Select features and target
data = data[['pclass' , 'sex' , 'age' , 'fare' , 'embarked' , 'survived']]

X = data[['pclass' , 'sex' , 'age' , 'fare' , 'embarked']]
y = data['survived']
```

In [78]:
```
# Step 6 : Train-test split
X_train , X_test,y_train , y_test = train_test_split(X,y,test_size=0.3, random_stat
```

In [79]:
```
# Step 7 : Train Model
model = LogisticRegression(max_iter = 1000)
model.fit(X_train,y_train)
```

Out[79]:

▾ LogisticRegression

```
LogisticRegression(max_iter=1000)
```

In [80]:
```
# Prediction
y_pred = model.predict(X_test)
```

In [81]:
```
#from sklearn.metrices import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print("Acuracy:",accuracy)
```

```
Acuracy: 0.7947761194029851
```

In [82]:
```
new_passenger = pd.DataFrame({
    'pclass': [3],
    'sex': ['male'],
    'age' : [28],
    'fare' : [7.25],
    'embarked' : ['S']
```

```
})
```

In [83]:
```
new_passenger_encoded = pd.get_dummies(new_passenger)
new_passenger_encoded = new_passenger_encoded.reindex(columns=X.columns,fill_value=
```

In [84]:
```
prediction = model.predict(new_passenger_encoded)
print("Survived" if prediction[0] == 1 else "Did not survive")
```

```
Survived
```

In [85]:
```
new_passenger = pd.DataFrame({
     'pclass': [1,3,2],
    'sex': ['female','male','female'],
    'age' : [38,45,14],
    'fare' : [80.0, 8.05,20.0],
    'embarked' : ['C','S','Q']

})
```

In [86]:
```
new_passenger_encoded = pd.get_dummies(new_passenger)
new_passenger_encoded = new_passenger_encoded.reindex(columns=X.columns,fill_value=
```

In [87]:
```
prediction = model.predict(new_passenger_encoded)
print("Survived" if prediction[0] == 1 else "Did not survive")
```

```
Survived
```

In [88]:
```
predictions = model.predict(new_passenger_encoded)
for i,pred in enumerate(predictions):
    print(f"Passenger {i+1}:",
          "Survived" if pred == 1 else "Did not survive")
```

```
Passenger 1: Survived
Passenger 2: Survived
Passenger 3: Survived
```

TRAINING A MACHINE LEARNING MODEL WITH TITANIC DATASET

In [102…
```
# Importing necessary libraries
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import seaborn as sns
import pandas as pd

# Loading the Titanic dataset
titanic_df = sns.load_dataset('titanic')

# Splitting the dataset
train_data, test_data = train_test_split(titanic_df, test_size=0.2, random_state=42

print(f"Train data shape: {train_data.shape}")
```

```python
print(f"Test data shape: {test_data.shape}")

# Separating target variable
x_train = train_data.drop("survived", axis=1)
y_train = train_data["survived"]

x_test = test_data.drop("survived", axis=1)
y_test = test_data["survived"]

# Convert categorical variables into numerical form
x_train = pd.get_dummies(x_train, drop_first=True)
x_test = pd.get_dummies(x_test, drop_first=True)

# Handling missing values
x_train = x_train.fillna(0)
x_test = x_test.fillna(0)

# Align train and test columns (IMPORTANT)
x_train, x_test = x_train.align(x_test, join="left", axis=1, fill_value=0)

# Initialize Logistic Regression model
logreg = LogisticRegression(max_iter=1000)

# Train the model
logreg.fit(x_train, y_train)

# Make predictions
predictions = logreg.predict(x_test)

# Display evaluation metrics
print("Classification Report:")
print(classification_report(y_test, predictions))

print("Confusion Matrix:")
print(confusion_matrix(y_test, predictions))

print("Accuracy Score:")
print(accuracy_score(y_test, predictions))
```

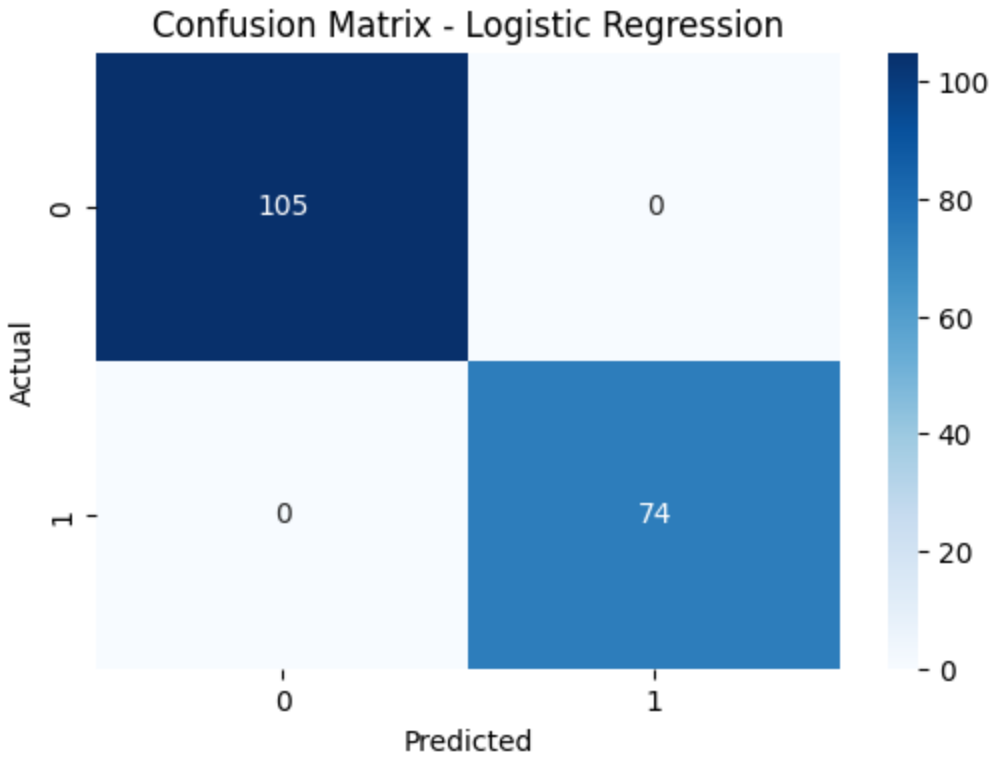
```
Train data shape: (712, 15)
Test data shape: (179, 15)
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       105
           1       1.00      1.00      1.00        74

    accuracy                           1.00       179
   macro avg       1.00      1.00      1.00       179
weighted avg       1.00      1.00      1.00       179

Confusion Matrix:
[[105   0]
 [  0  74]]
Accuracy Score:
1.0
```

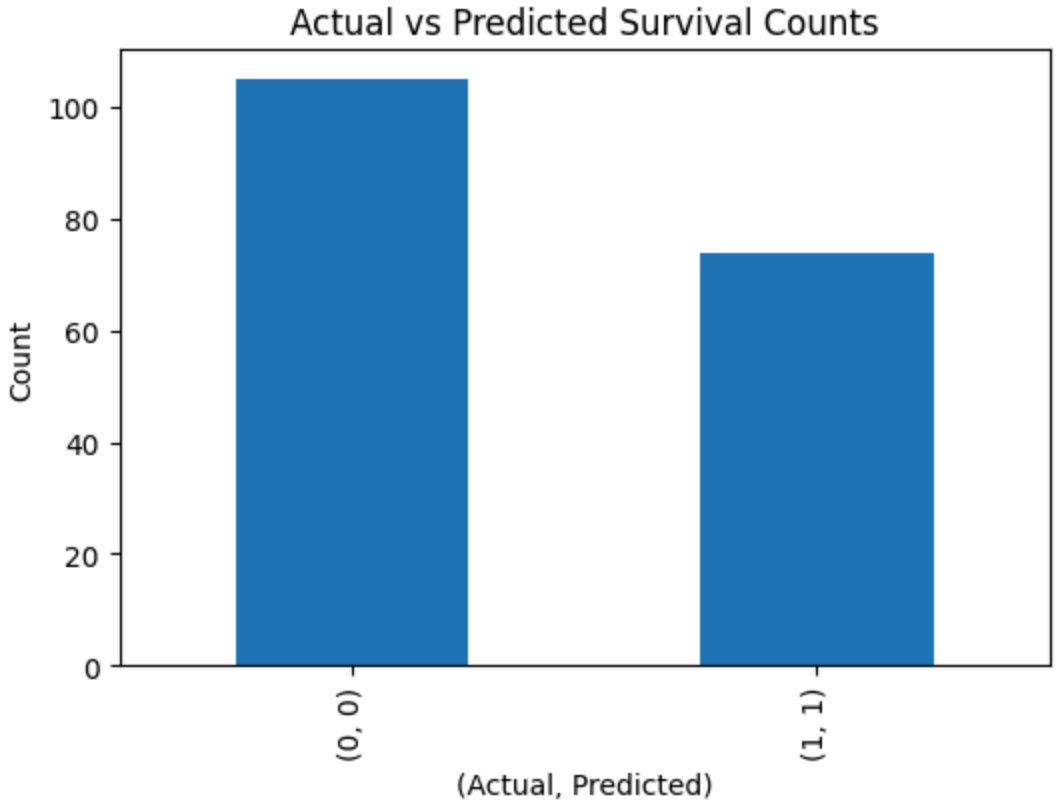
PLOTTING

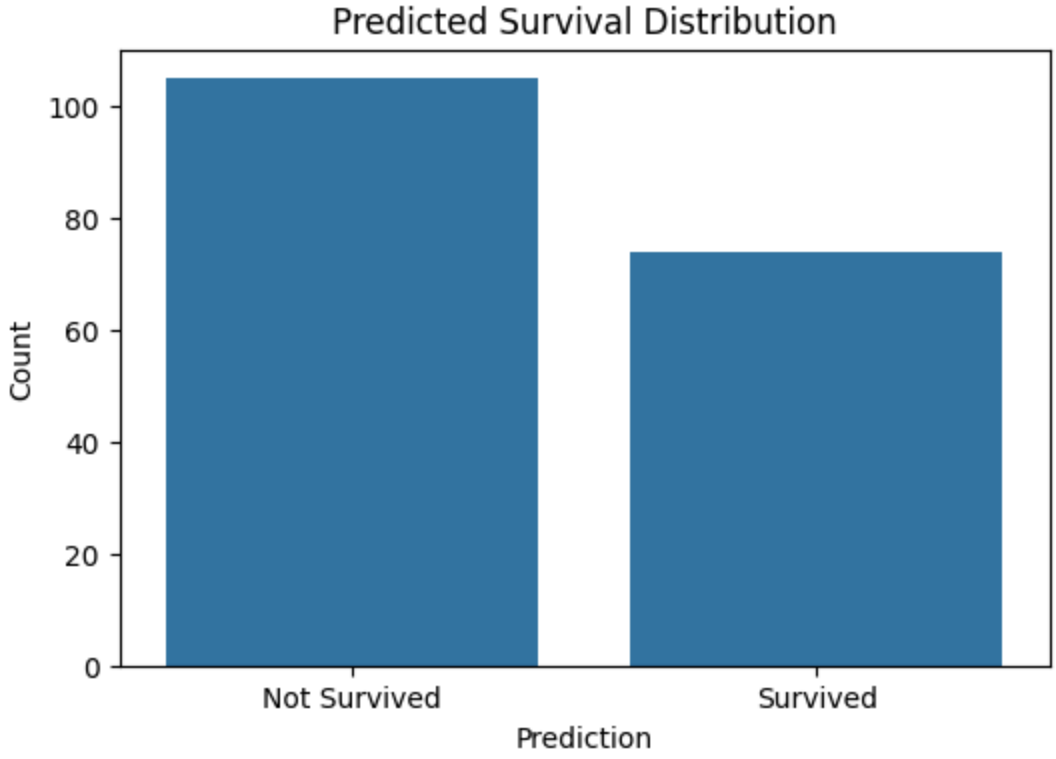
In [109…

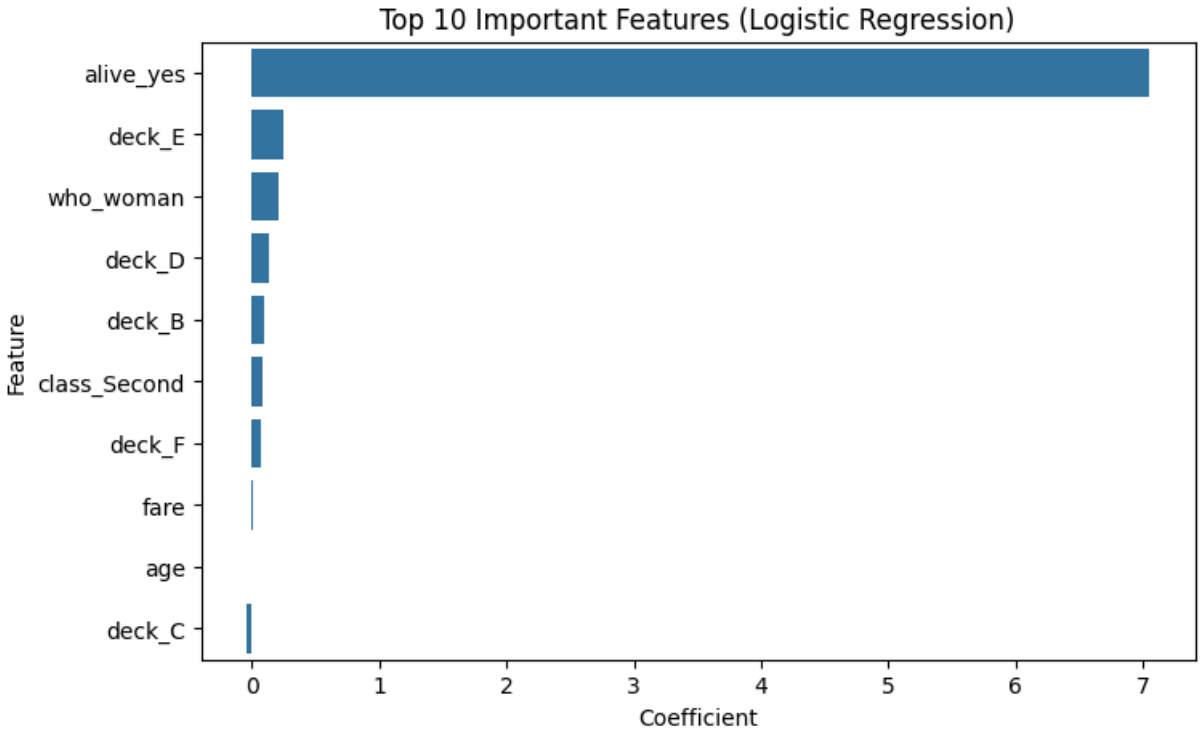
```
import matplotlib.pyplot as plt
import seaborn as sns

cm = confusion_matrix(y_test, predictions)

plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - Logistic Regression")
plt.show()



comparison_df = pd.DataFrame({
    "Actual": y_test.values,
    "Predicted": predictions
})

comparison_df.value_counts().plot(kind="bar", figsize=(6,4))
plt.title("Actual vs Predicted Survival Counts")
plt.xlabel("(Actual, Predicted)")
plt.ylabel("Count")
plt.show()




plt.figure(figsize=(6,4))
sns.countplot(x=predictions)
plt.xticks([0,1], ["Not Survived", "Survived"])
plt.title("Predicted Survival Distribution")
plt.xlabel("Prediction")
plt.ylabel("Count")
plt.show()




coefficients = pd.DataFrame({
    "Feature": x_train.columns,
    "Coefficient": logreg.coef_[0]
})

coefficients = coefficients.sort_values(by="Coefficient", ascending=False).head(10)

plt.figure(figsize=(8,5))
sns.barplot(x="Coefficient", y="Feature", data=coefficients)
plt.title("Top 10 Important Features (Logistic Regression)")
plt.show()
```

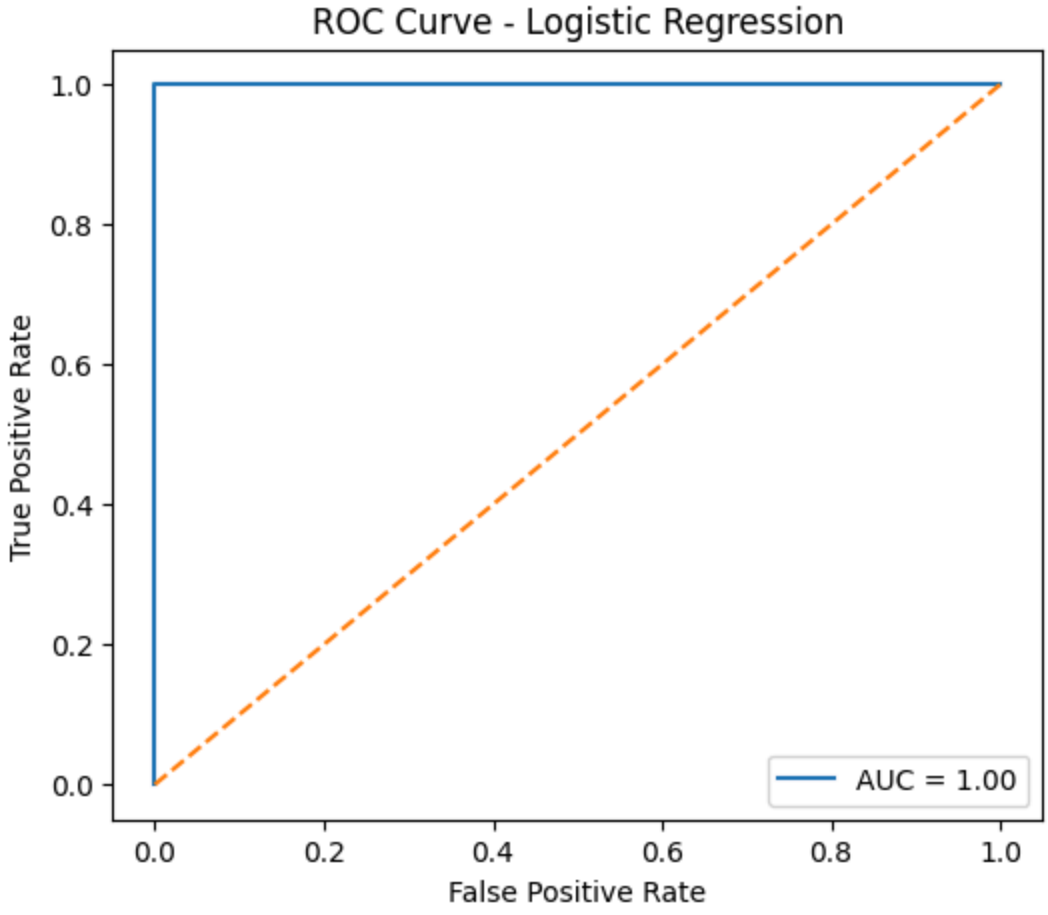
```python
from sklearn.metrics import roc_curve, auc

y_prob = logreg.predict_proba(x_test)[:,1]
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6,5))
plt.plot(fpr, tpr, label=f"AUC = {roc_auc:.2f}")
plt.plot([0,1], [0,1], linestyle="--")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - Logistic Regression")
plt.legend()
plt.show()
```




Actual vs Predicted Survival Counts
Count
100
80
60
40
20
0
(0, 0)
(1, 1)
(Actual, Predicted)


Predicted Survival Distribution
Count
100
80
60
40
20
0
Not Survived
Survived
Prediction





STANDARDIZATION

In [110…

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()

# Standardize the data
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)



logreg = LogisticRegression(max_iter=1000)
logreg.fit(x_train_scaled, y_train)

predictions = logreg.predict(x_test_scaled)

print("Accuracy:", accuracy_score(y_test, predictions))



print("Mean after standardization:", x_train_scaled.mean())
print("Std after standardization:", x_train_scaled.std())



scaler.fit_transform(x_test)   # ❌ NEVER fit on test data
```

```
Accuracy: 1.0
Mean after standardization: -6.833810508195883e-18
Std after standardization: 1.0
```

Out[110…

```
array([[ 0.88742288, -1.34903605,  0.82036305, ..., -0.32394177,
        -1.40830868,  1.19118383],
       [-0.25537349,  0.37905601, -0.55202   , ..., -0.32394177,
         0.7100716 , -0.83950099],
       [ 0.88742288, -0.23413795, -0.55202   , ..., -0.32394177,
         0.7100716 , -0.83950099],
       ...,
       [ 0.88742288,  0.76927035,  0.82036305, ..., -0.32394177,
         0.7100716 ,  1.19118383],
       [-0.25537349, -0.40137266, -0.55202   , ..., -0.32394177,
         0.7100716 ,  1.19118383],
       [ 0.88742288, -1.12605643,  0.82036305, ..., -0.32394177,
         0.7100716 ,  1.19118383]])
```

NORMALIZATION

In [111…

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

# Normalize the data
x_train_norm = scaler.fit_transform(x_train)
x_test_norm = scaler.transform(x_test)
```

```
logreg = LogisticRegression(max_iter=1000)
logreg.fit(x_train_norm, y_train)

predictions = logreg.predict(x_test_norm)

print("Accuracy:", accuracy_score(y_test, predictions))



print("Min value:", x_train_norm.min())
print("Max value:", x_train_norm.max())



scaler.fit_transform(x_test)  # ❌ Wrong
```

```
Accuracy: 1.0
Min value: 0.0
Max value: 1.0
```

Out[111…

```
array([[1.        , 0.        , 0.25      , ..., 0.        , 0.        ,
        1.        ],
       [0.5       , 0.43661972, 0.        , ..., 0.        , 1.        ,
        0.        ],
       [1.        , 0.28169014, 0.        , ..., 0.        , 1.        ,
        0.        ],
       ...,
       [1.        , 0.53521127, 0.25      , ..., 0.        , 1.        ,
        1.        ],
       [0.5       , 0.23943662, 0.        , ..., 0.        , 1.        ,
        1.        ],
       [1.        , 0.05633803, 0.25      , ..., 0.        , 1.        ,
        1.        ]])
```