TITANIC PROJECT

Naima Dzhunushova
Makhabat Zhyrgalbekova

PROJECT OVERVIEW

We are working with the Titanic Data Set from Kaggle, and trying to predict a classification of passengers on Titanic:
survived or deceased

or

DATASET OVERVIEW

The Titanic dataset contains information about the passengers divided into two sets: training and test data. The training set includes passenger information along with the survival outcome, and test set - without. There are **891 entries**, each with **11 attributes** in our train dataset.

COLUMNS IN THE DATASET

- Survived: Indicates if a passenger survived (1) or not (0).
- Pclass: Ticket class, a proxy for socio-economic status (1 = 1st, 2 = 2nd, 3 = 3rd).
- Name: Passenger's name.
- Sex: Passenger's sex.
- Age: Passenger's age in years.
- SibSp: Number of siblings/spouses aboard the Titanic.
- Parch: Number of parents/children aboard the Titanic.
- Ticket: Ticket number.
- Fare: Passenger fare.
- Cabin: Cabin number.
- Embarked: Port of embarkation (C = Cherbourg, Q = Queenstown, S = Southampton).

IMPORTING LIBRARIES

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

import missingno as msno
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import KFold, cross_val_score

import warnings
warnings.filterwarnings('ignore')
```

IMPORTING DATASET

VIEW THE TOP FEW ROWS OF THE DATAFRAME

```
train = pd.read_csv("titanic_train.csv", index_col='PassengerId')
test = pd.read_csv("titanic_test.csv", index_col='PassengerId')

train.head()
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

NUMBER OF ROWS AND COLUMNS

```
1  titanic_Data= pd.concat([train, test], ignore_index=False)
2  rows,columns = titanic_Data.shape
3  print("num of rows in data :",rows)
4  print("the number of column in the data:",columns)
```

```
num of rows in data : 1309
the number of column in the data: 11
```

DESCRIPTION OF DATA

```
1  titanic_Data.info()
2  train.describe().T
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1309 entries, 1 to 1309
Data columns (total 11 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Survived  891 non-null    float64
 1   Pclass    1309 non-null   int64  
 2   Name      1309 non-null   object 
 3   Sex       1309 non-null   object 
 4   Age       1046 non-null   float64
 5   SibSp     1309 non-null   int64  
 6   Parch     1309 non-null   int64  
 7   Ticket    1309 non-null   object 
 8   Fare      1308 non-null   float64
 9   Cabin     295 non-null    object 
 10  Embarked  1307 non-null   object 
dtypes: float64(3), int64(3), object(5)
memory usage: 122.7+ KB
```

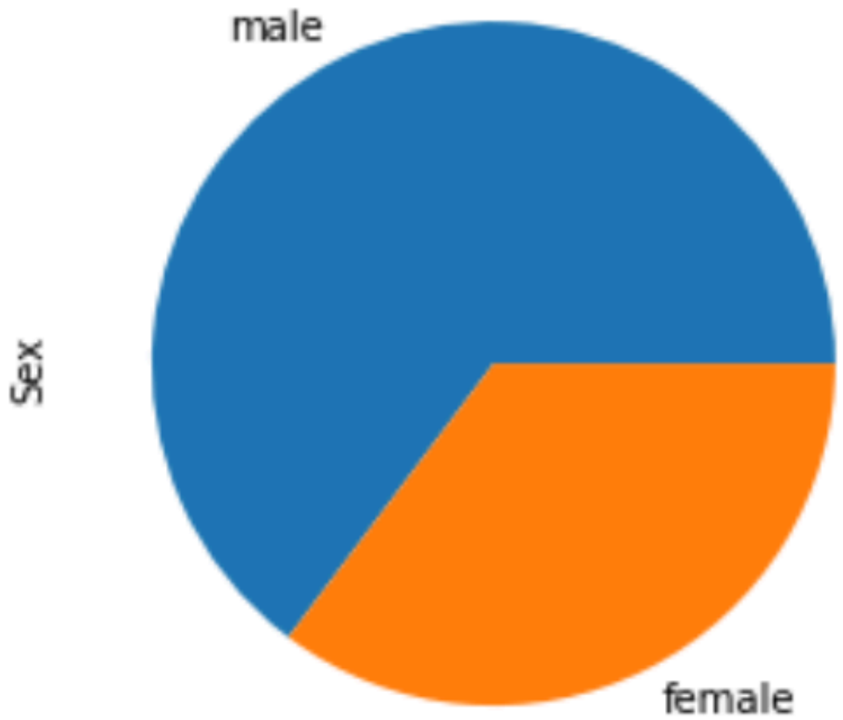
STATISTICS OF DATA

	count	mean	std	min	25%	50%	75%	max
Survived	891.0	0.383838	0.486592	0.00	0.0000	0.0000	1.0	1.0000
Pclass	891.0	2.308642	0.836071	1.00	2.0000	3.0000	3.0	3.0000
Age	714.0	29.699118	14.526497	0.42	20.1250	28.0000	38.0	80.0000
SibSp	891.0	0.523008	1.102743	0.00	0.0000	0.0000	1.0	8.0000
Parch	891.0	0.381594	0.806057	0.00	0.0000	0.0000	0.0	6.0000
Fare	891.0	32.204208	49.693429	0.00	7.9104	14.4542	31.0	512.3292

VISUALISATION OF DATA

```
1  train.Sex.value_counts().plot(kind='pie')
```

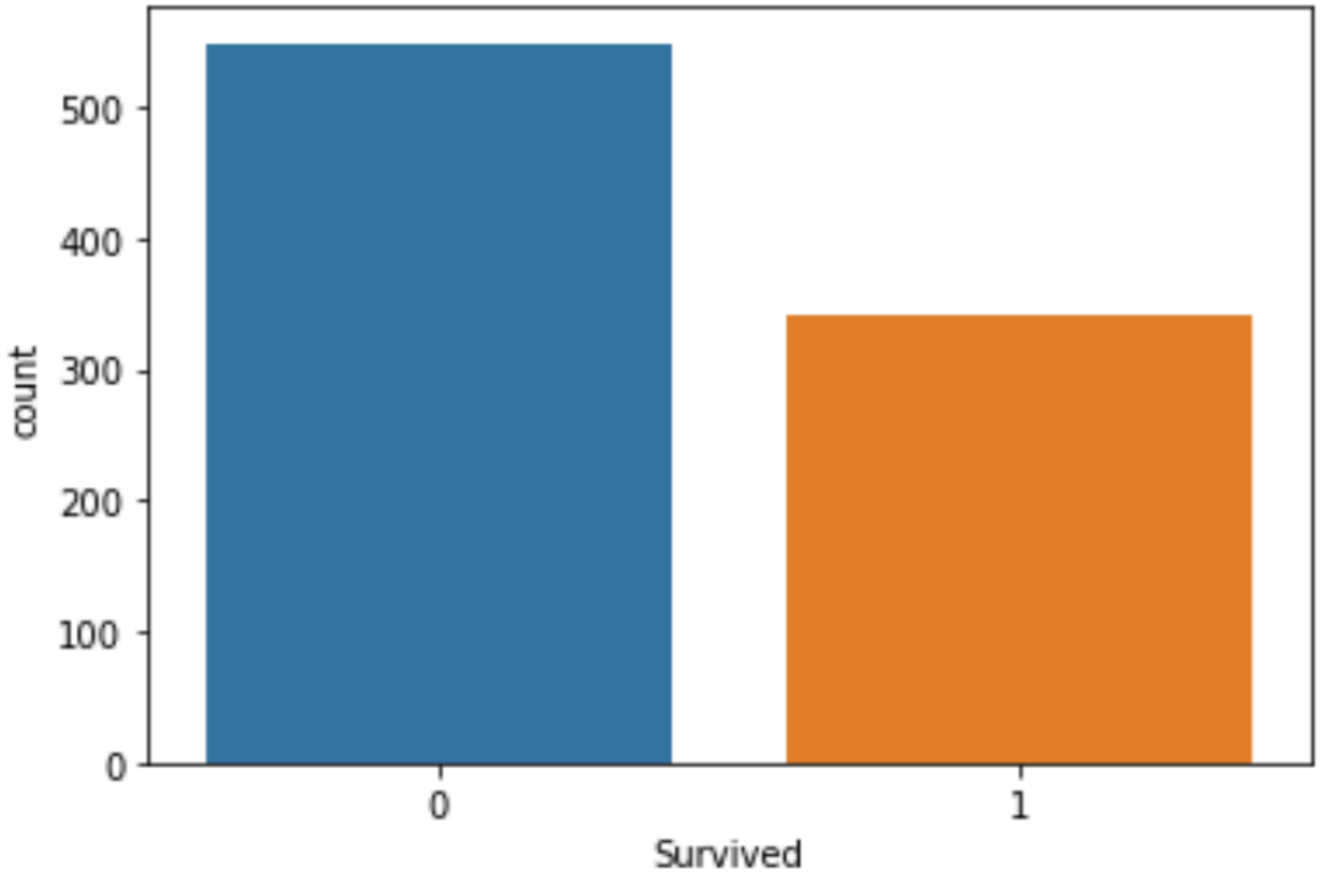
```
<AxesSubplot:ylabel='Sex'>
```



VISUALISATION OF DATA

```
1  sns.countplot(x='Survived',data=train)
```

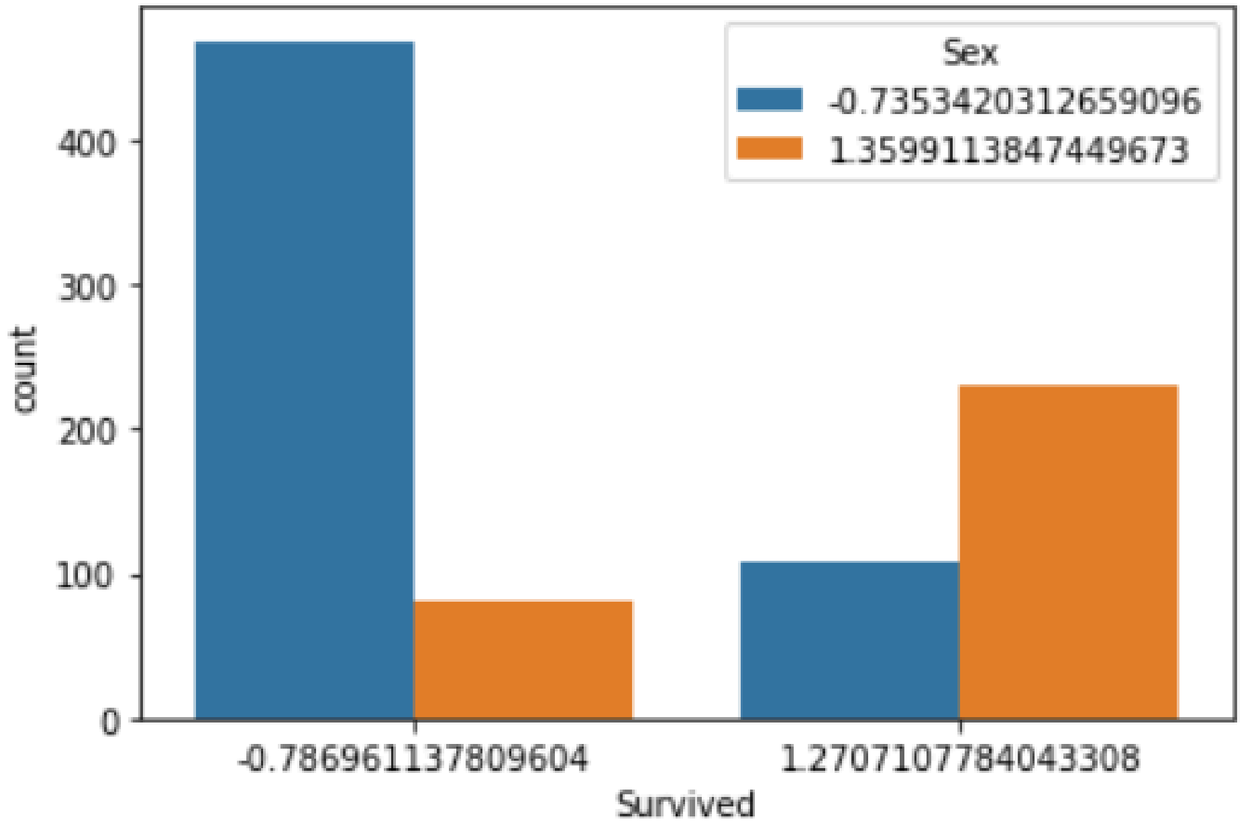
```
<AxesSubplot:xlabel='Survived', ylabel='count'>
```



VISUALISATION OF DATA

```
1  sns.countplot(x='Survived',data=train,hue='Sex')
```

```
<AxesSubplot:xlabel='Survived', ylabel='count'>
```



VISUALISATION OF DATA

```python
women = train.loc[train.Sex == 'female']["Survived"]
rate_women = sum(women)/len(women)

print("% of women who survived:", rate_women)
```

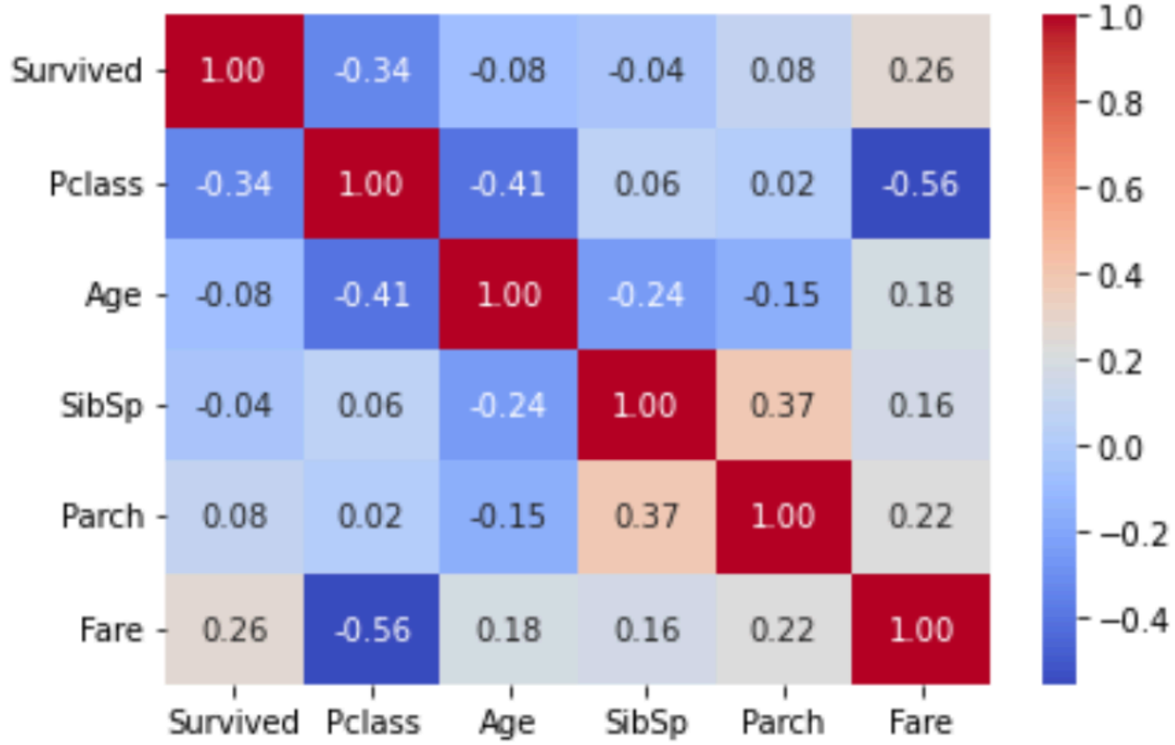
```
% of women who survived: 0.7420382165605095
```

```python
men = train.loc[train.Sex == 'male']["Survived"]
rate_men = sum(men)/len(men)

print("% of men who survived:", rate_men)
```

```
% of men who survived: 0.18890814558058924
```

VISUALISATION OF DATA/CORRELATION OF DATA

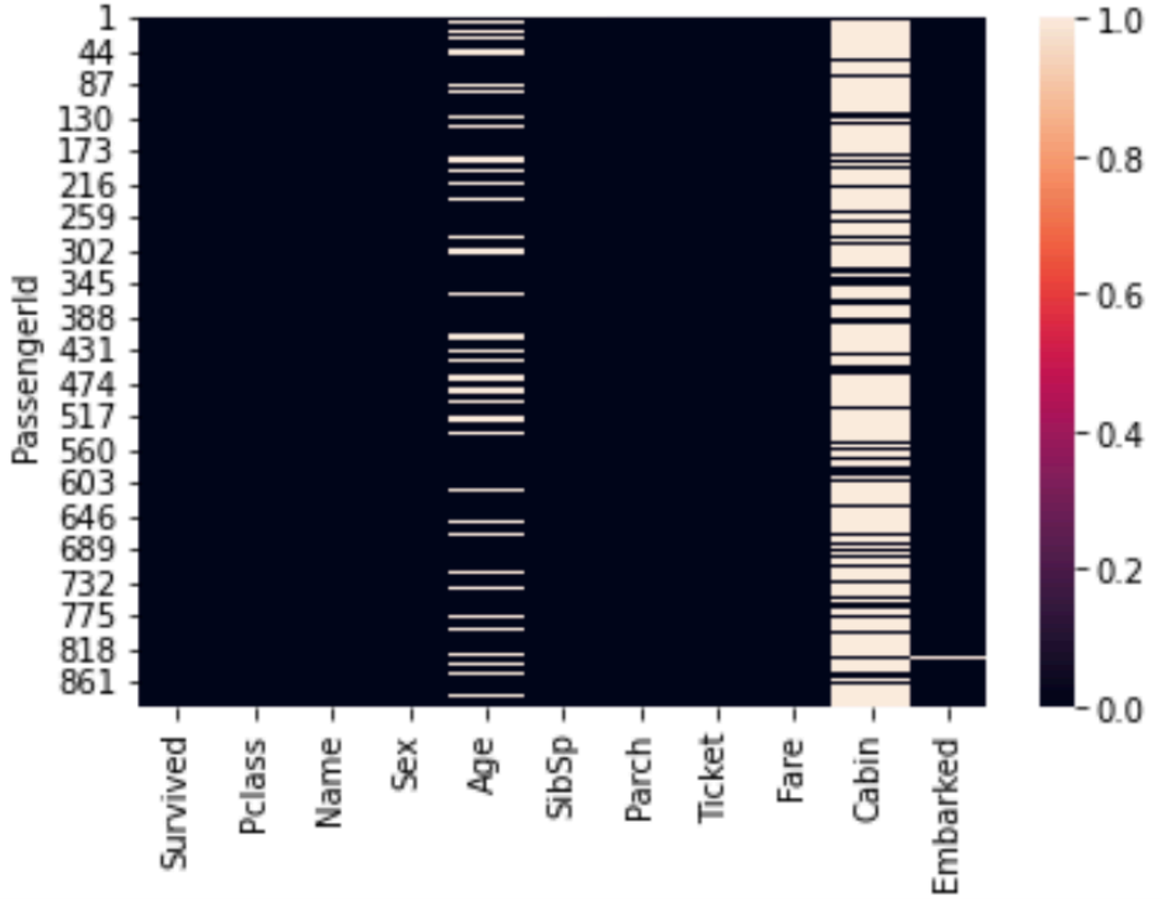
```
relation=titanic_Data.loc[:,['Survived','Pclass','Age','SibSp','Parch','Fare']]
corr=relation.corr()
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f")
plt.show()
#correlation is modest
```



CHECK FOR MISSING DATA

```
1  sns.heatmap(train.isna())
```

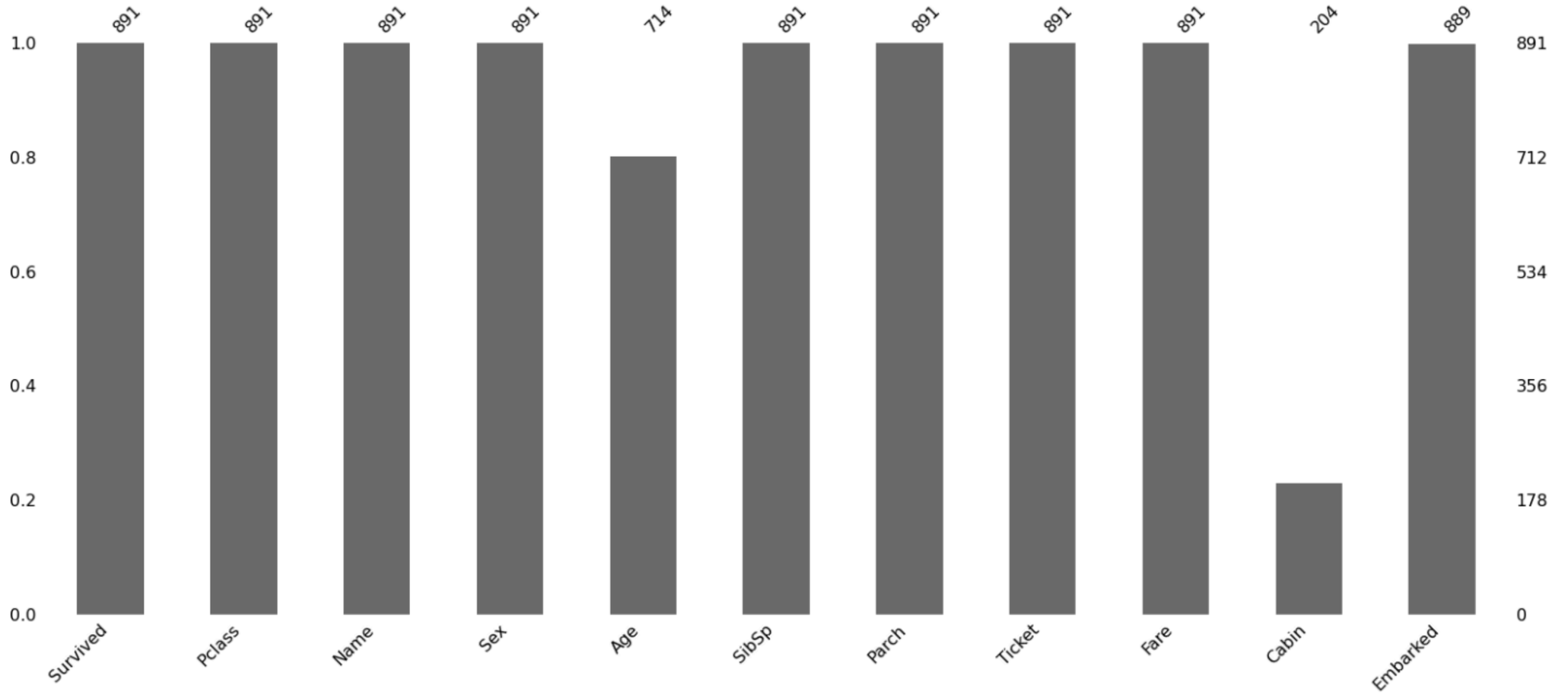
```
<AxesSubplot:ylabel='PassengerId'>
```



CHECK FOR MISSING DATA(TRAIN)

```
1 missing_values_train = msno.bar(train)
2 missing_values_train
```

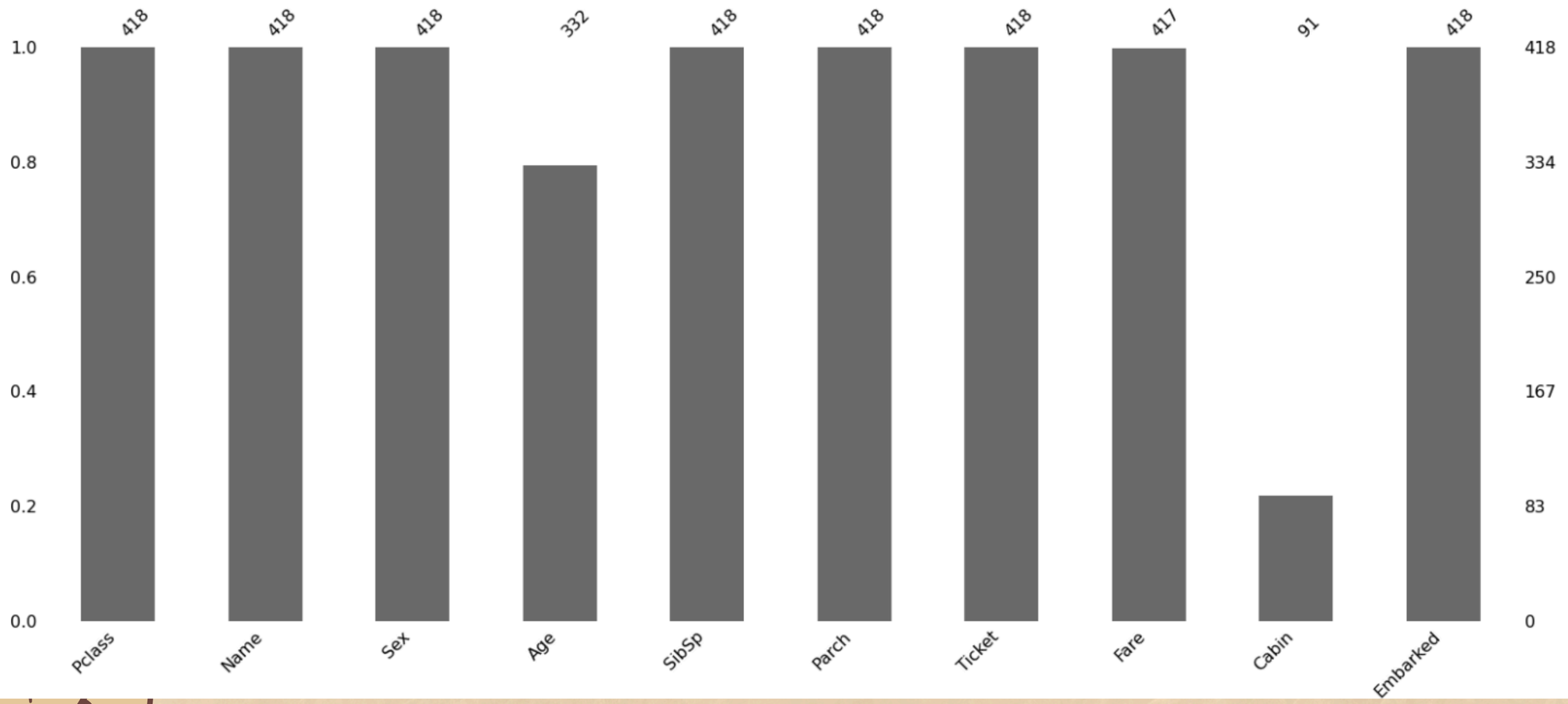
```
<AxesSubplot:>
```



CHECK FOR MISSING DATA(TEST)

```
1 missing_values_test = msno.bar(test)
2 missing_values_test
```

```
<AxesSubplot:>
```



DATA CLEANING: IMPUTE MISSING VALUES IN AGE BASED PCLASS

```
imputer = SimpleImputer(strategy='mean')

data = [train, test]
for dataset in data:
    dataset['Age'] = imputer.fit_transform(dataset['Age'].values.reshape(-1, 1))
    dataset.dropna(axis=0,subset='Embarked',inplace=True)
    dataset['Fare'] = (dataset['Fare'].fillna(0)).astype(int)
```

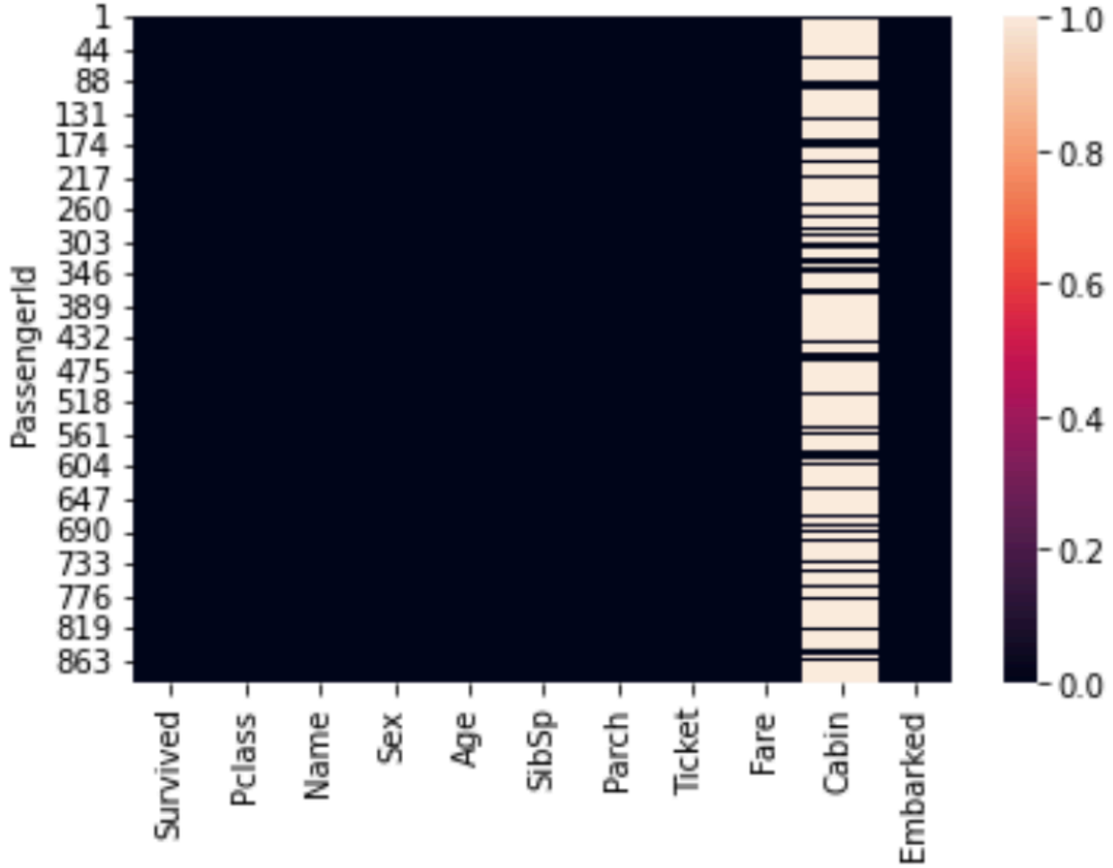
RESULT

```
1 train.isna().sum()
```

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

```
1 sns.heatmap(train.isna())
2
```

```
<AxesSubplot:ylabel='PassengerId'>
```



FEATURE ENGENEERING

```python
for dataset in data:
    dataset['isAlone'] = (dataset['SibSp'] + dataset['Parch'] == 0).astype(int)
    dataset['isFamily'] = (dataset['SibSp'] + dataset['Parch'] > 0).astype(int)
```

```python
for dataset in data:

    dataset['FamilySize'] = dataset['SibSp'] + dataset['Parch'] + 1

    dataset['FarePerPerson'] = (dataset['Fare'] / dataset['FamilySize']).astype(int)
```

```python
for dataset in data:
     dataset['AgeGroup'] = pd.cut(dataset['Age'], bins=[0, 12, 18, 60, np.inf],
                                  labels=['Child', 'Teenager', 'Adult', 'Senior'])
```

FEATURE CREATION/DROPPING COLUMNS

```python
for dataset in data:
    # Extract Title from the Name column
    dataset['Title'] = dataset['Name'].str.extract(' ([A-Za-z]+)\.', expand=False)

    # Consolidate rare titles into a single 'Rare' category
    rare_titles = ['Lady', 'Countess', 'Capt', 'Col', 'Don', 'Dr', 'Major', 'Rev', 'Sir', 'Jonkheer', 'Dona']
    dataset['Title'] = dataset['Title'].replace(rare_titles, 'Rare')

    # Replace 'Mlle' and 'Ms' with 'Miss', and 'Mme' with 'Mrs'
    dataset['Title'] = dataset['Title'].replace({'Mlle': 'Miss', 'Ms': 'Miss', 'Mme': 'Mrs'})

```

```python
columns_to_drop = ['Name', 'Ticket', 'Cabin']

for dataset in data:
    dataset.drop(columns=columns_to_drop, inplace=True)
```

CREATING DUMMY VARIABLES

```
#gender=pd.get_dummies(train_data['Sex'],drop_first=True)
#train_data['Gender']=gender
#embarked = pd.get_dummies(train_data['Embarked'], drop_first = True)
#train_data = pd.concat([train_data, embarked], axis='columns')
#train_data.drop(['Name','Ticket','Embarked'],axis=1,inplace=True)
```

PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Gender	Q	S
1	0	3	male	22.0	1	0	7.2500	1	0	1
2	1	1	female	38.0	1	0	71.2833	0	0	0
3	1	3	female	26.0	0	0	7.9250	0	0	1
4	1	1	female	35.0	1	0	53.1000	0	0	1
5	0	3	male	35.0	0	0	8.0500	1	0	1

CREATING NEW FEATURES

```python
def category_mapping(data):

    mapping = {
    'Sex': {'male': 0, 'female': 1},
    'Embarked': {'S': 0, 'C': 1, 'Q': 2},
    'Title': {'Mr': 1, 'Miss': 2, 'Mrs': 3, 'Master': 4, 'Rare': 5},
    'AgeGroup': {'Child': 1, 'Teenager': 2, 'Adult': 3, 'Senior': 4}
    }

    for df in data:
        for feature, value in mapping.items():
            df[feature] = df[feature].map(value)

    return data

data = category_mapping(data)
```

RESULT

```
1  train.head()
```

PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	isAlone	isFamily	FamilySize	FarePerPerson	AgeGroup	Title
1	0	3	0	22.0	1	0	7	0	0	1	2	3	3	1
2	1	1	1	38.0	1	0	71	1	0	1	2	35	3	3
3	1	3	1	26.0	0	0	7	0	1	0	1	7	3	2
4	1	1	1	35.0	1	0	53	0	0	1	2	26	3	3
5	0	3	0	35.0	0	0	8	0	1	0	1	8	3	1

```
1  scaler = StandardScaler()
2  for i in range(len(data)):
3      data[i][:] = scaler.fit_transform(data[i])
```

BUILDING CLASSIFIERS

```
X = train.drop(columns='Survived')
y = train['Survived'].astype(int)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
classifiers = [RandomForestClassifier, DecisionTreeClassifier, SVC, LogisticRegression, KNeighborsClassifier]
classifiers_names = ['RandomForestClassifier', 'DecisionTreeClassifier', 'SVC', 'LogisticRegression',
                     'KNeighborsClassifier']
classifiers_scores = []

for classifier, name in zip(classifiers, classifiers_names):
    kfold = KFold(n_splits=10, random_state=25, shuffle=True)
    model = classifier()
    cv_score = cross_val_score(model, X, y, cv=kfold, scoring='accuracy')
    classifiers_scores.append(cv_score.mean())

# create a dataframe of classifiers and their scores
classifiers_df = pd.DataFrame({'Classifier': classifiers_names, 'Accuracy score': classifiers_scores})
classifiers_df
```

OUTPUT

Out[38]:

	Classifier	Accuracy score
0	RandomForestClassifier	0.813228
1	DecisionTreeClassifier	0.785112
2	SVC	0.823417
3	LogisticRegression	0.811032
4	KNeighborsClassifier	0.802017

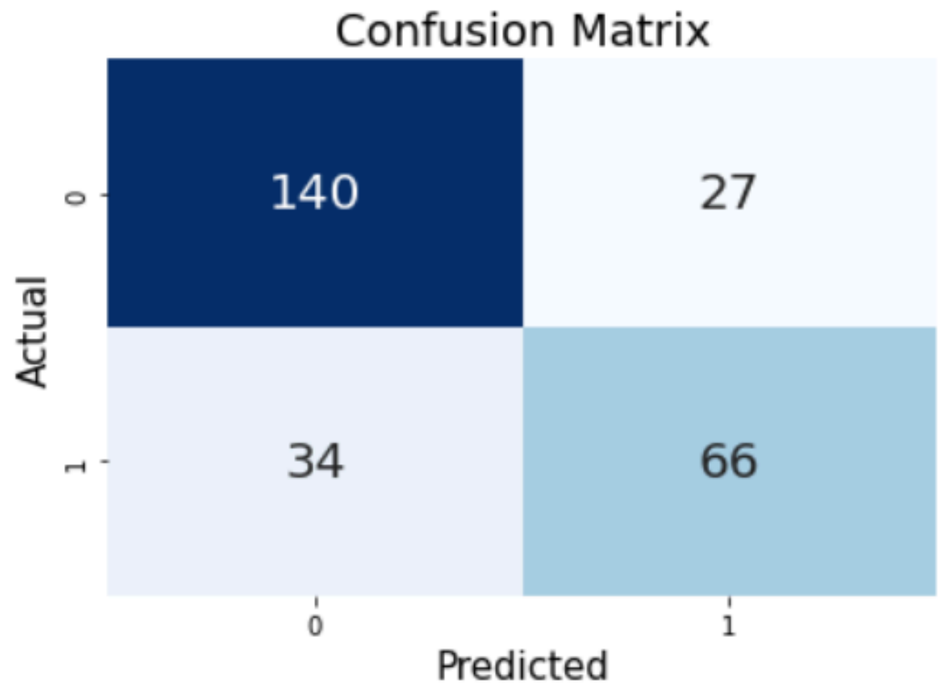
BUILDING A MODEL

```
for classifier, name in zip(classifiers, classifiers_names):
    model = classifier()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    report = classification_report(y_test, y_pred)
    matrix = confusion_matrix(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)

    print("\033[1mName:\033[0m {}:".format(name))
    print(f"Classification Report:\n{report}\n")

    plt.figure(figsize=(8, 6))
    sns.heatmap(matrix, annot=True, fmt='d', cmap='Blues', cbar=False, annot_kws={'size': 20})
    plt.xlabel('Predicted', size=15)
    plt.ylabel('Actual', size=15)
    plt.title('Confusion Matrix', size=18)
    plt.show()

    print(f"Accuracy Score: {accuracy}\n")
```

RANDOM FOREST CLASSIFIER

Name: RandomForestClassifier:
Classification Report:

	precision	recall	f1-score	support
0	0.80	0.84	0.82	167
1	0.71	0.66	0.68	100
accuracy			0.77	267
macro avg	0.76	0.75	0.75	267
weighted avg	0.77	0.77	0.77	267

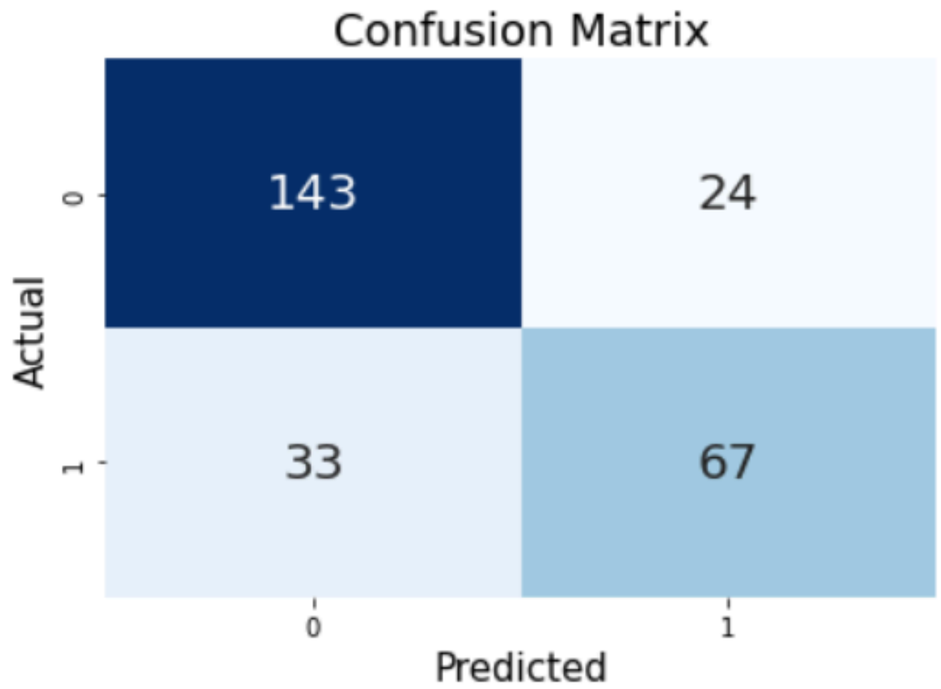


Accuracy Score: 0.7715355805243446

DECISION TREE CLASSIFIER

Name: DecisionTreeClassifier:
Classification Report:

	precision	recall	f1-score	support
0	0.81	0.86	0.83	167
1	0.74	0.67	0.70	100
accuracy			0.79	267
macro avg	0.77	0.76	0.77	267
weighted avg	0.78	0.79	0.78	267



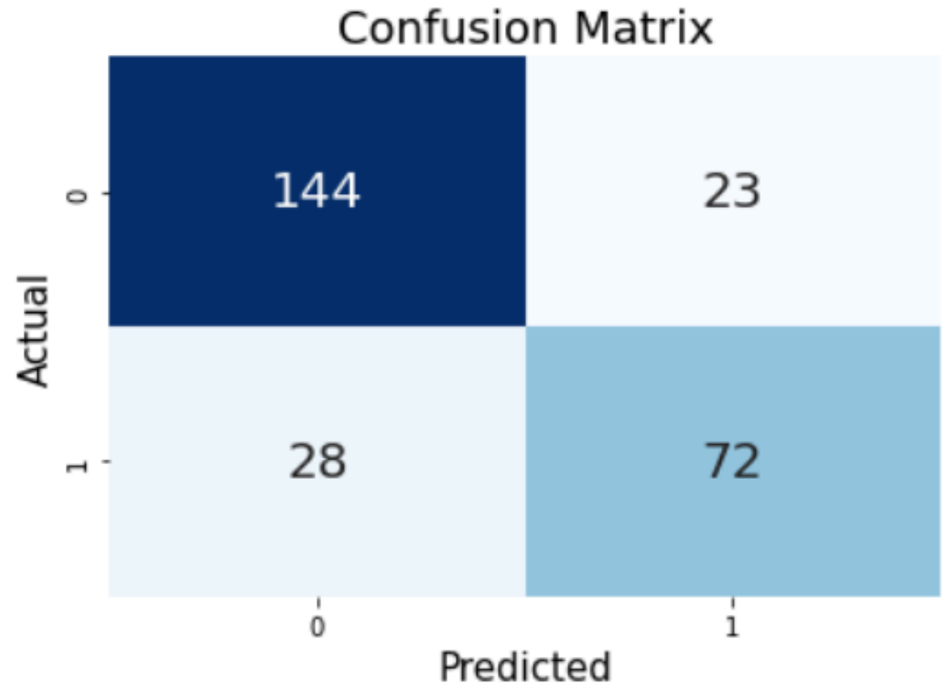
Accuracy Score: 0.7865168539325843

SVC

Name: SVC:

Classification Report:

	precision	recall	f1-score	support
0	0.84	0.86	0.85	167
1	0.76	0.72	0.74	100
accuracy			0.81	267
macro avg	0.80	0.79	0.79	267
weighted avg	0.81	0.81	0.81	267

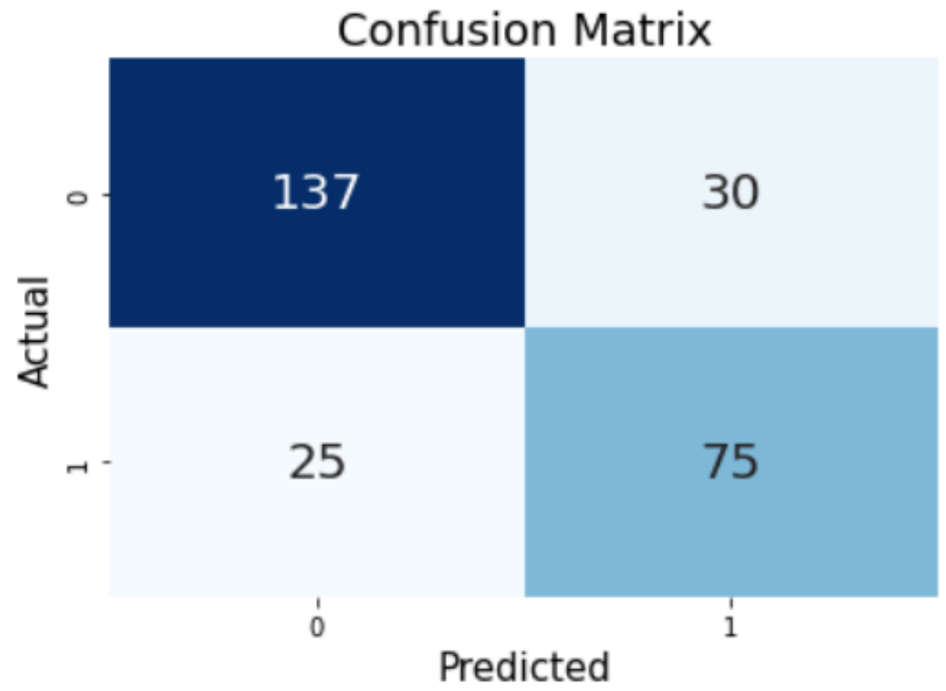


Accuracy Score: 0.8089887640449438

LOGISTIC REGRESSION

Name: LogisticRegression:
Classification Report:

	precision	recall	f1-score	support
0	0.85	0.82	0.83	167
1	0.71	0.75	0.73	100
accuracy			0.79	267
macro avg	0.78	0.79	0.78	267
weighted avg	0.80	0.79	0.79	267

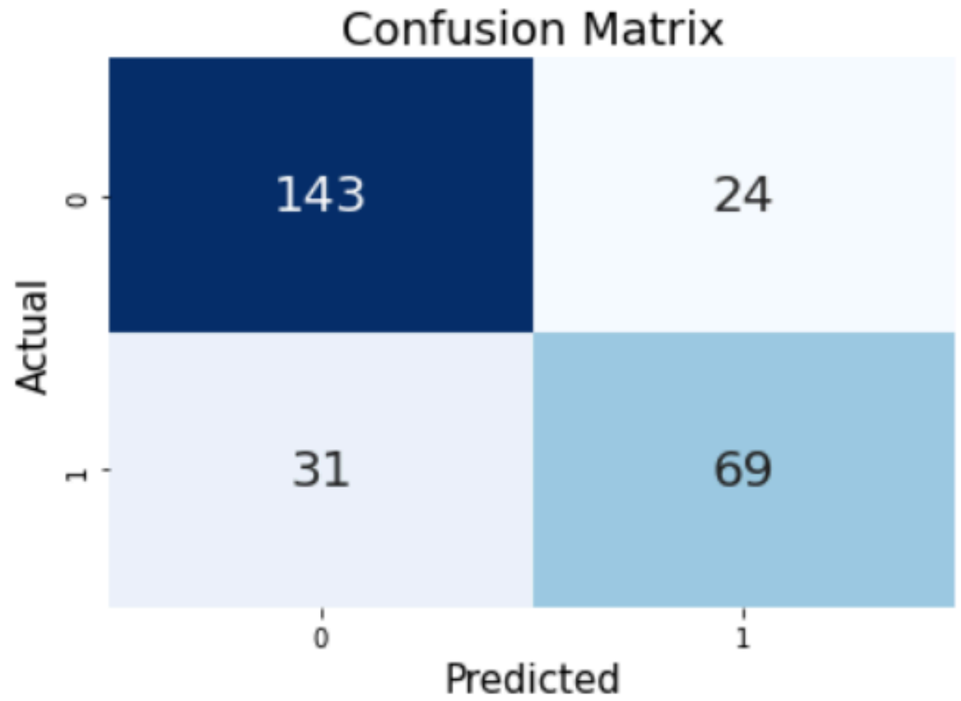


Accuracy Score: 0.7940074906367042

KNEIGHBORS CLASSIFIER

Name: KNeighborsClassifier:
Classification Report:

	precision	recall	f1-score	support
0	0.82	0.86	0.84	167
1	0.74	0.69	0.72	100
accuracy			0.79	267
macro avg	0.78	0.77	0.78	267
weighted avg	0.79	0.79	0.79	267



Accuracy Score: 0.7940074906367042

PREDICTIONS SUBMISSION

```
rf = RandomForestClassifier(n_estimators = 100)
rf.fit(X,y)

test_predictions = rf.predict(test)
```

```
test_predictions = pd.DataFrame({'Survived' : test_predictions})
```

```
test_id = pd.read_csv("titanic_test.csv")['PassengerId']
```

```
submission_df = pd.concat([test_id, test_predictions], axis = 1)
```

```
submission_df.to_csv('random_forest.csv', index=False)
```

PREDICTIONS SUBMISSION

```
svm = SVC()
svm.fit(X,y)

test_predictions = svm.predict(test)
```

```
test_predictions = pd.DataFrame({'Survived' : test_predictions})
```

```
test_id = pd.read_csv("titanic_test.csv")['PassengerId']
```

```
submission_df = pd.concat([test_id, test_predictions], axis = 1)
```

```
submission_df.to_csv('SVC.csv', index=False)
```

PREDICTIONS SUBMISSION

```
clf = LogisticRegression()
clf.fit(X,y)

test_predictions = clf.predict(test)
```

```
test_predictions = pd.DataFrame({'Survived' : test_predictions}
```

```
test_id = pd.read_csv("titanic_test.csv")['PassengerId']
```

```
submission_df = pd.concat([test_id, test_predictions], axis = 1
```

```
submission_df.to_csv('logistic_regression.csv', index=False)
```

RESULTS

Submissions

All | Successful | Errors

Recent

Submission and Description	Public Score
logistic_regression.csv Complete · now	0.78708
SVC.csv Complete · 20s ago	0.78229
random_forest.csv Complete · 1m ago	0.7488