AIN SHAMS UNIVERSITY

FACULTY OF ENGINEERING

CREDIT HOURS ENG. PROGRAM

Computer engineering and software systems

AIN SHAMS UNIVERSITY

FACULTY OF ENGINEERING

Introduction to machine learning: CSE381

Project Phase 1

Submitted to:

Prof. Mahmoud Khalil

Eng. Omar Elesawy

Submitted by:

Anass Zikry 20P6382

Kerollos Noshy 21P0132

Omar Diab 20P3176

Contents

Dataset pre-processing

The objective variable in Kaggle's Titanic dataset is Survived, which indicates whether a passenger survived (1) or not (0). Each passenger is uniquely recognized by their PassengerId. The dataset contains details about the passenger's name, gender, age, number of siblings/spouses (SibSp) and parents/children (Parch) on board, ticket details, fare, cabin number, and port of embarkation (C = Cherbourg, Q = Queenstown, S = Southampton). It also includes information about the passenger's socioeconomic status, which is indicated by Pclass (1st, 2nd, or 3rd class).

```
1  train_data = pd.read_csv('train.csv', index_col='PassengerId')
2  test_data = pd.read_csv('test.csv', index_col='PassengerId')
```

```
1  train_data.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

Figure 1 Head of data

We start with visualizing the data to look how the data is described and how we will work with it before training the model, as it helps understand the distribution of features, identify

patterns, and gain insights into the relationships between variables, as well as managing missing values and transform categorical values into numerical representations.

```
plt.pie(train_data['Sex'].value_counts(), labels=['male', 'female'],
autopct='%1.1f%%')
plt.title('Percentage of male and female')
plt.show()
```

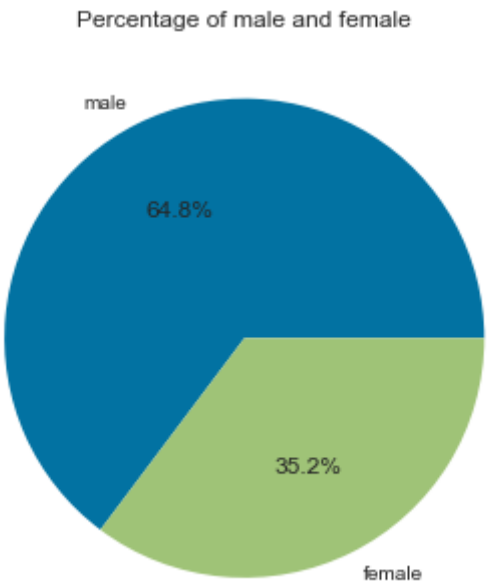


Figure 2 Percentage of male and female

```
plt.pie(train_data[train_data['Sex'] == 'male'].groupby('Survived')
['Name'].count(), labels=['Dead', 'Survived'], autopct='%1.1f%%')
plt.title('Percentage of males who survived')
plt.show()
```

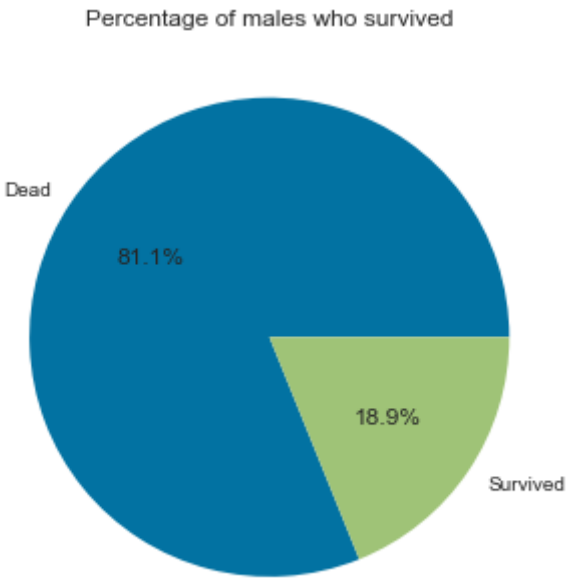


Figure 3 Percentage of survived males

```
plt.pie(train_data[train_data['Sex'] == 'female'].groupby('Survived')
['Name'].count(), labels=['Dead', 'Survived'], autopct='%1.1f%%')
plt.title('Percentage of females who survived')
plt.show()
```

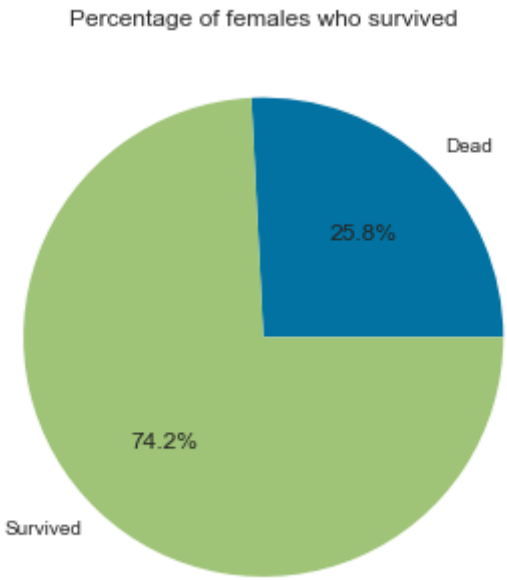


Figure 4 Percentage of survived females

```
plt.figure(figsize=(20,5))
sns.countplot(data= train_data[(train_data['Sex'] == 'male')], x='Ag
e',hue='Survived')
plt.legend(['Dead', 'Survived'])
plt.xticks(rotation=90)
plt.show()
```

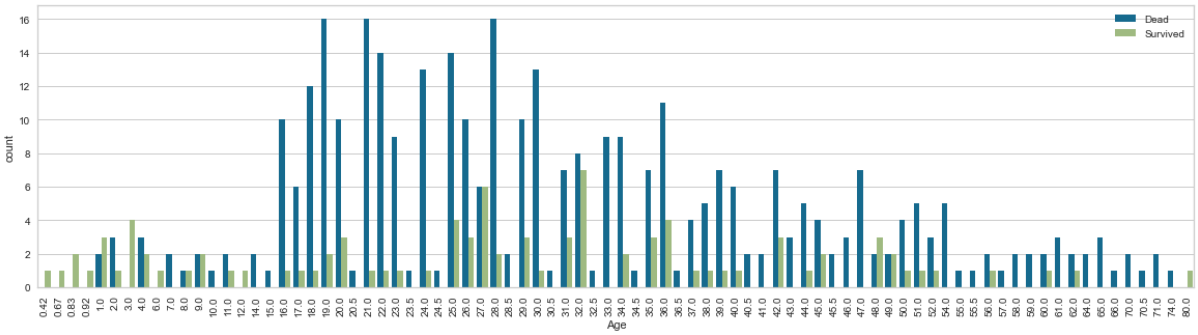


Figure 5 Age count plot

```
1
2  sns.swarmplot(data = train_data, x='Pclass', y='Age', hue='Survived')
3  plt.title('Age Vs. passenger class')
4  plt.tight_layout()
5  plt.show()
```

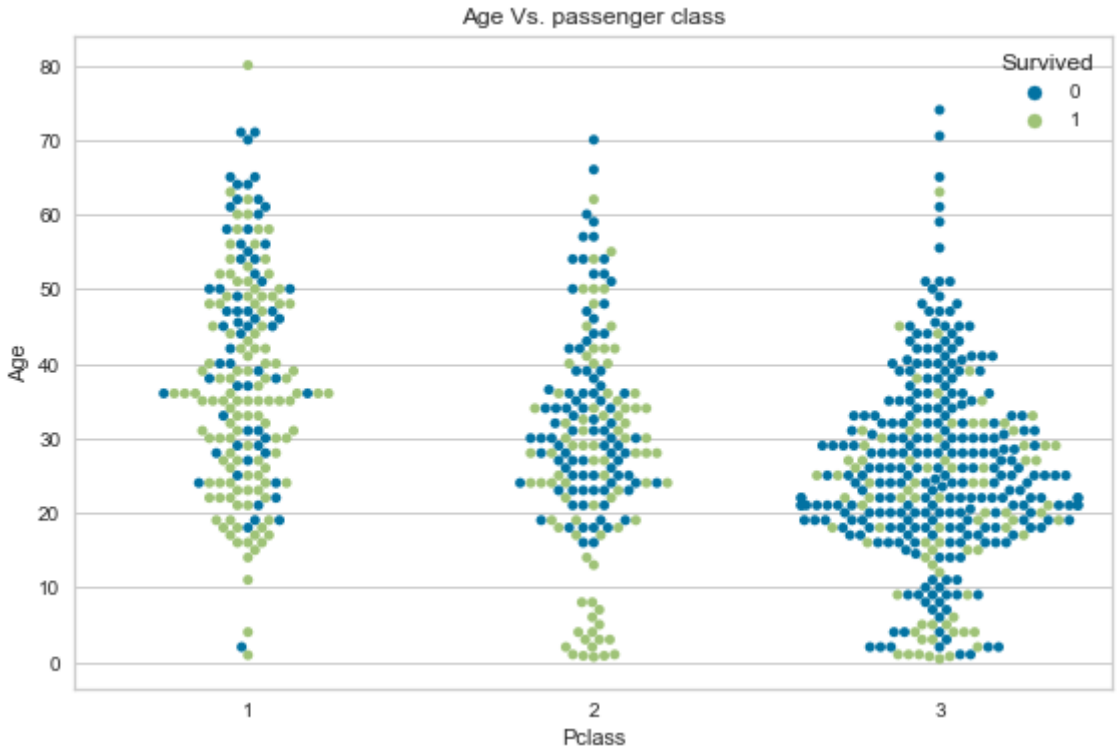


Figure 6 Age vs Passenger class

```
sns.swarmplot(data = train_data, x='Sex', y='Age', hue='Survived')
plt.legend(['Dead', 'Survived'])
plt.title('Age Vs. gender')
plt.tight_layout()
plt.show()
```

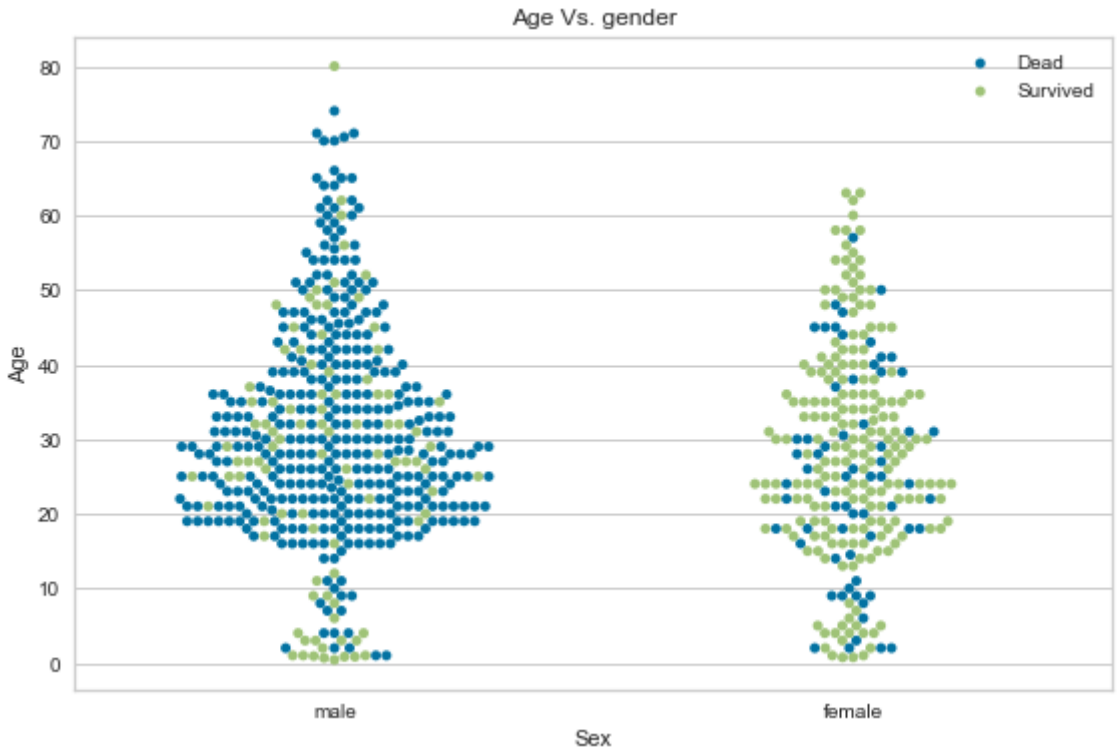


Figure 7 Age vs Gender

```
sns.countplot(data= train_data, x='Sex',hue='Survived')
plt.title('Number of males and females who survived or died')
plt.legend(['Dead', 'Survived'])
plt.show()
```

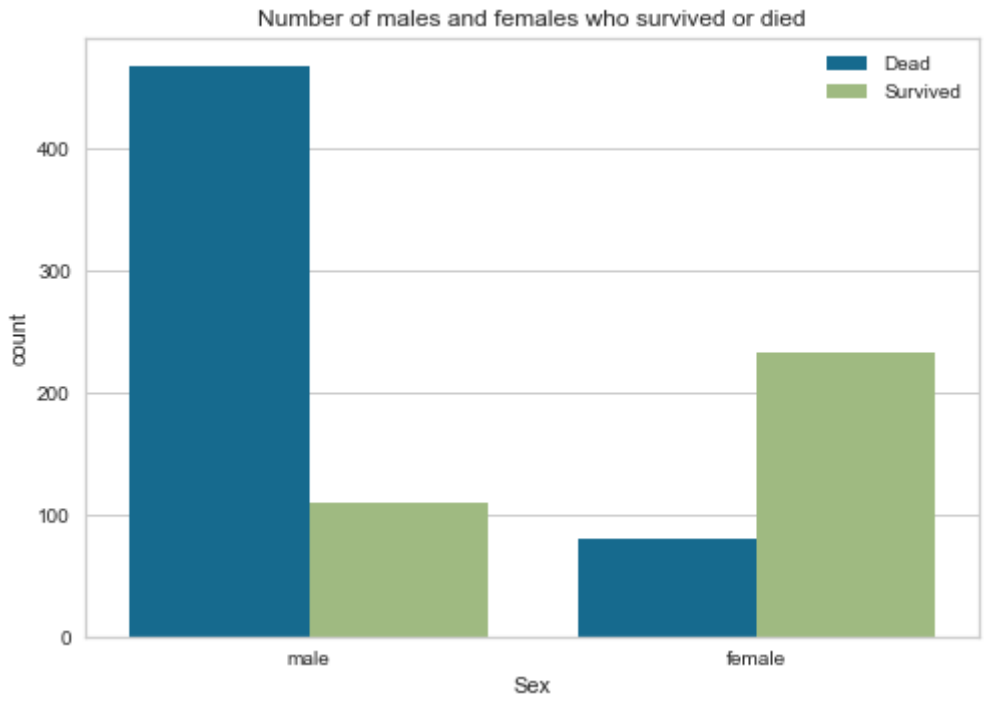


Figure 8 Survived males vs females

```
sns.countplot(data= train_data[train_data['Sex']=='female'], x='Pclass',hue='Survived')
plt.title('Survival females Vs. class')
plt.legend(['Dead', 'Survived'])
plt.show()
```

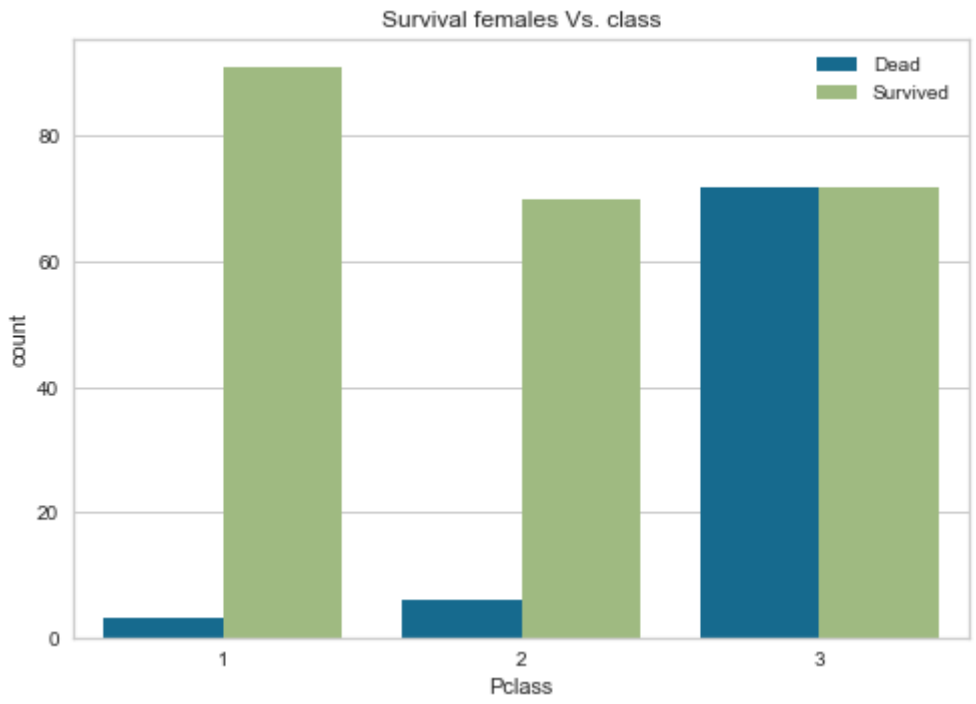


Figure 9 Survived females vs Passenger class

```
sns.countplot(data= train_data[train_data['Sex']=='male'], x='Pclass',hue='Survived')
plt.title('Survival males Vs. class')
plt.legend(['Dead', 'Survived'])
plt.show()
```

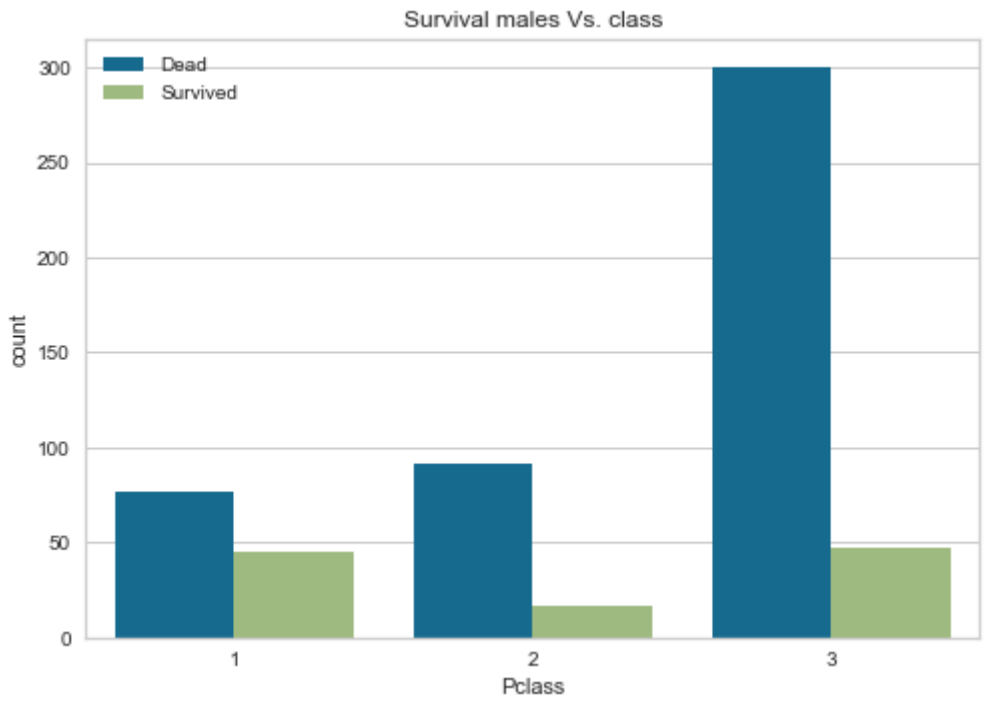


Figure 10 Survived males vs Passenger class

```
sns.countplot(data= train_data, x='Pclass',hue='Survived')
plt.title('Survival Vs. class')
plt.legend(['Dead', 'Survived'])
plt.show()
```

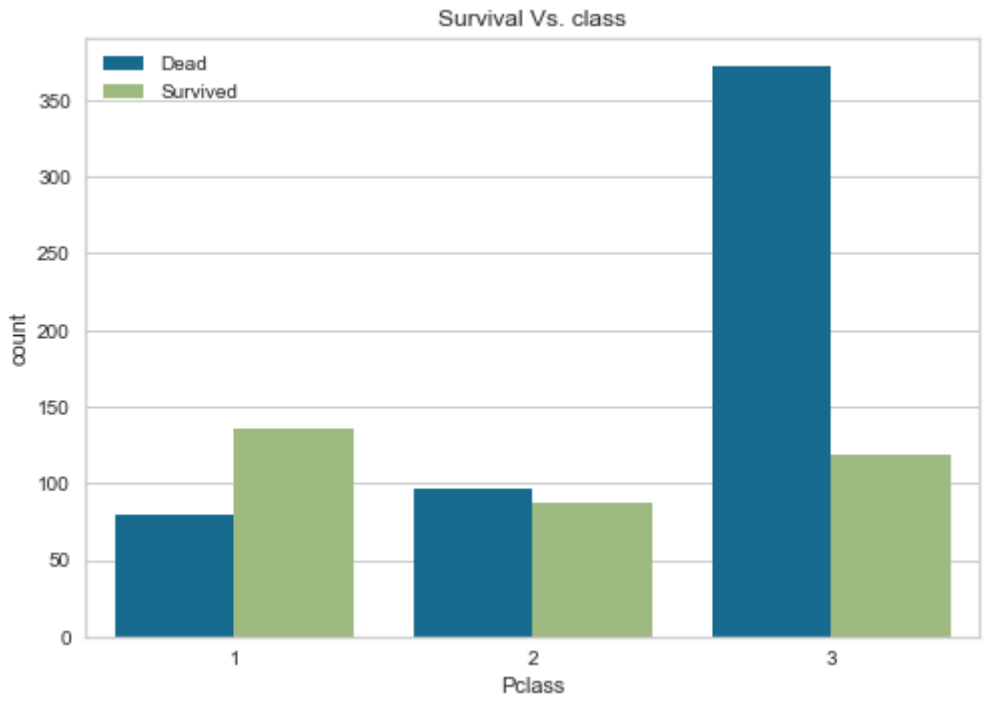


Figure 11 Survival vs Passenger class

```
sns.countplot(data= train_data, x='SibSp',hue='Survived')
plt.title('Survival Vs. number of siblings')
plt.legend(['Dead', 'Survived'])
plt.show()
```

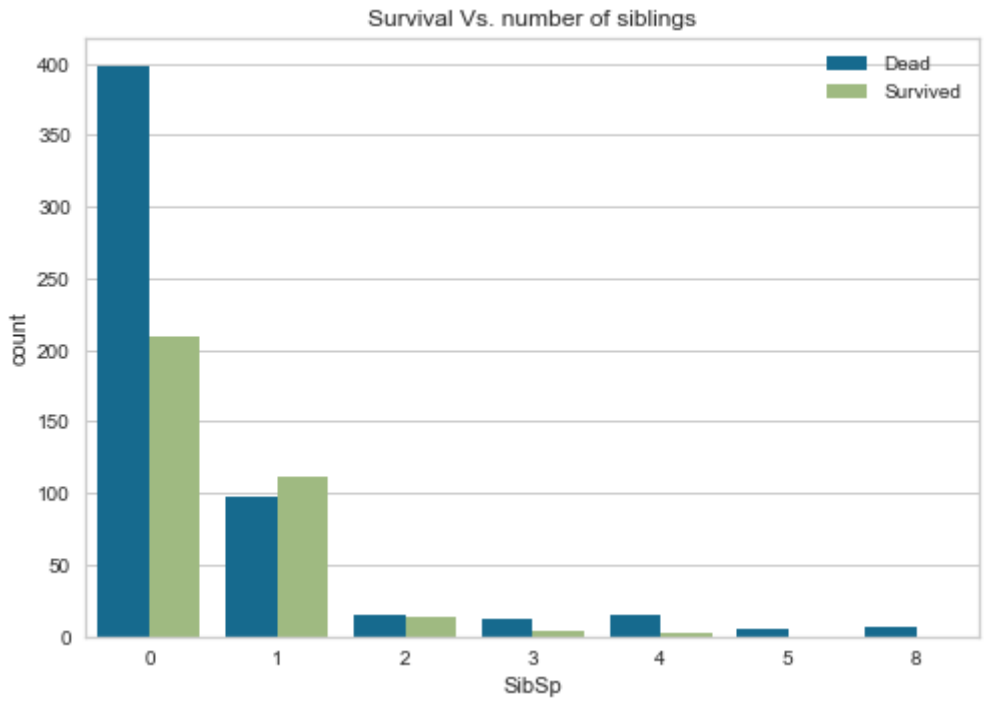


Figure 12 Survival vs Number of siblings

```
sns.countplot(data= train_data, x='Parch',hue='Survived')
plt.title('Survival Vs. number of parents')
plt.legend(['Dead', 'Survived'])
plt.show()
```

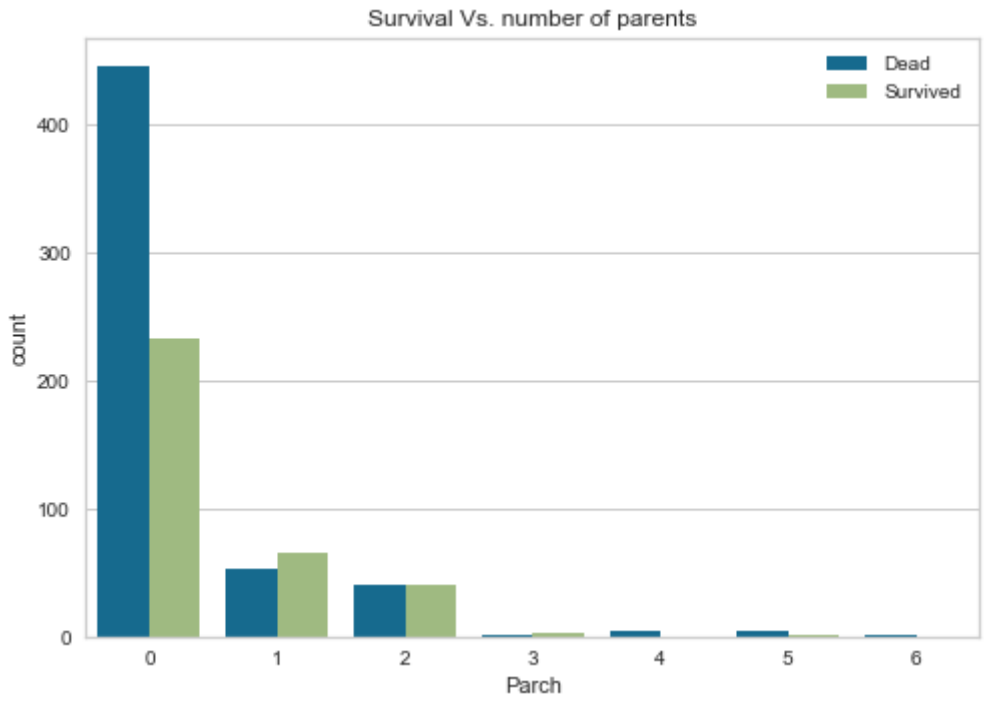


Figure 13 Survival vs Number of parents

```
sns.countplot(data= train_data, x='Embarked',hue='Survived')
plt.title('Survival Vs. Embarked')
plt.legend(['Dead', 'Survived'])
plt.show()
```

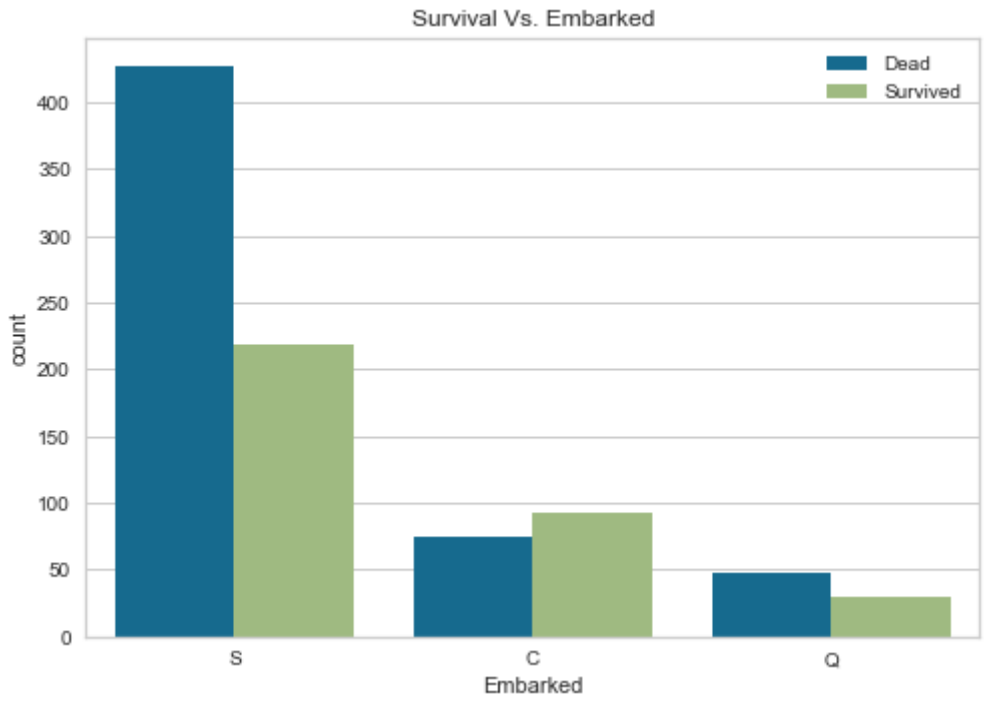


Figure 14 Survival vs Embarked

We generated a new feature from the name of each passenger by extracting the title of the passenger and adding it to a column Title in the dataset.

```python
def extract_title(name:str):
    return name.split(',')[1].split('.')[0]
train_data['Name'].apply(lambda x:extract_title(x))
titles = {}
for n in train_data['Name']:
    if n.split(',')[1].split('.')[0] not in titles.keys():
        titles[n.split(',')[1].split('.')[0]] = 1
    else:
        titles[n.split(',')[1].split('.')[0]]+=1
for k, v in titles.items():
    if v < 20:
        titles['other'] += v
def extract_title(name:str):
    if name.split(',')[1].split('.')[0] not in new_titles.keys():
        return 'other'
    else:
        return name.split(',')[1].split('.')[0]
train_data['Title'] = train_data['Name'].apply(lambda x: extract_titl
e(x))
train_data['Title'].value_counts()
```

```
Mr        517
Miss      182
Mrs       125
Master     40
other      27
```

Figure 15 Titles count

We looked at the Fare which was right skewed so we normalized its values using log, then we dropped the Ticket and the Cabin columns, as for the Ticket it was the number of ticket for each passenger that did not indicate a certain information and for the Cabin there was a lot of missing values (77% of the values are missing), then we convert the categorical values into numerical representations for the Sex, Embarked, and Title columns.

```
1  sns.histplot(data=train_data['Fare'], kde=True)
```

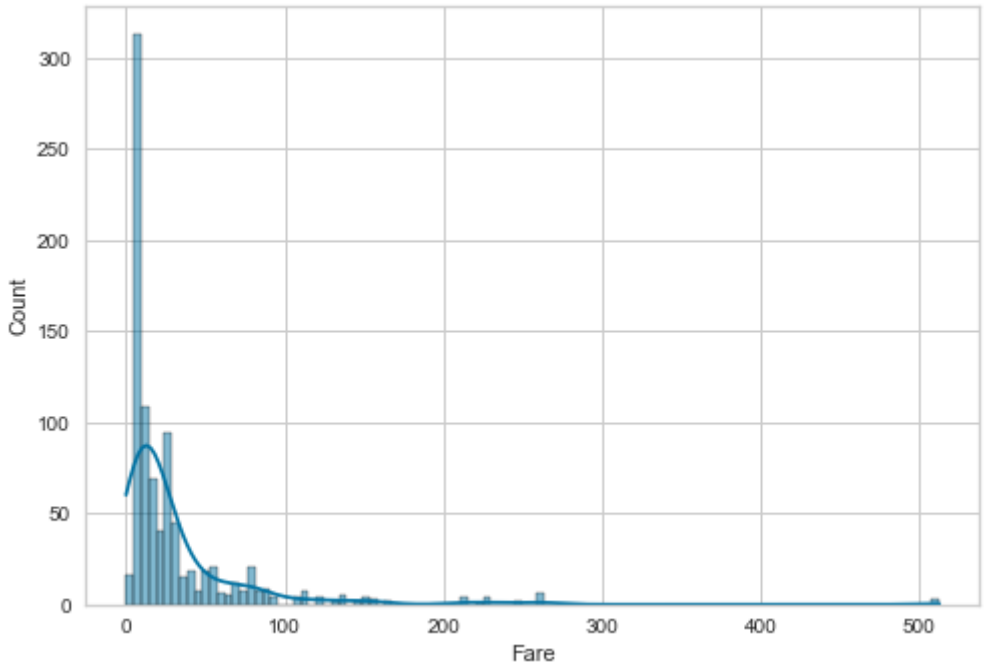


Figure 16 Fare before normalization

```
1  sns.histplot(np.log1p(train_data['Fare']), kde=True)
```

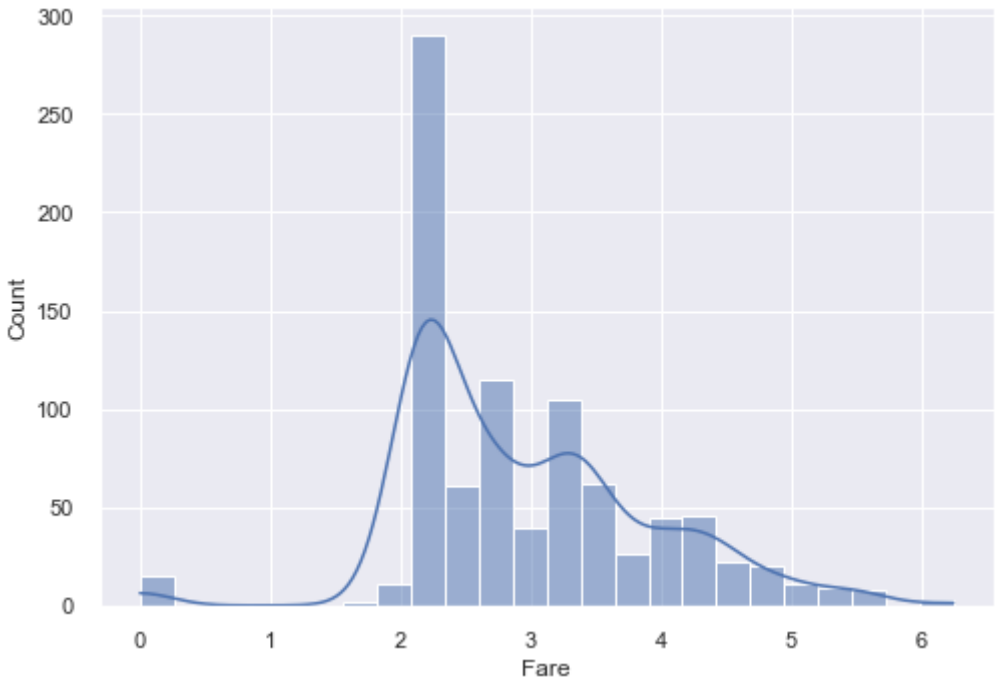


Figure 17 Fare after normalization

Filling missing data for the Age column using KNN imputer which is a method for imputing missing values in a dataset based on the values of its neighbors as not to change the original distribution of data, while using another method such as the median will change the original distribution.

```
imputer = KNNImputer(n_neighbors=8)
X_imputed = imputer.fit_transform(train_data)
X_imputed = pd.DataFrame(X_imputed, columns=train_data.columns)
```

```
X_imputed = pd.concat((X_imputed, pd.get_dummies(X_imputed['Title'])), axis=1)
X_imputed.columns = pd.Index(['Survived', 'Pclass', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked',
       'Title', 'female', 'male', 'Mr', 'Miss', 'Mrs', 'Master', 'other'])
X_imputed.drop('Title', axis=1, inplace=True)
X_imputed
```

```
plt.figure(figsize=(16, 9))
plt.subplot(2, 2, 1).set_title('Age Before Imputer missing values')
sns.histplot(train_data['Age'], kde=True, bins=40)

plt.subplot(2, 2, 2).set_title('Age after Imputer missing values using median')
sns.histplot(train_data['Age'].fillna(train_data['Age'].median()), kde=True, bins=40)

plt.subplot(2, 2, 3).set_title('Age Before Imputer missing values')
sns.histplot(train_data['Age'], kde=True, bins=40)

plt.subplot(2, 2, 4).set_title('Age after Imputer missing values using KNN Imputer')
sns.histplot(X_imputed['Age'], kde=True, bins=40)

plt.tight_layout()
plt.show()
```

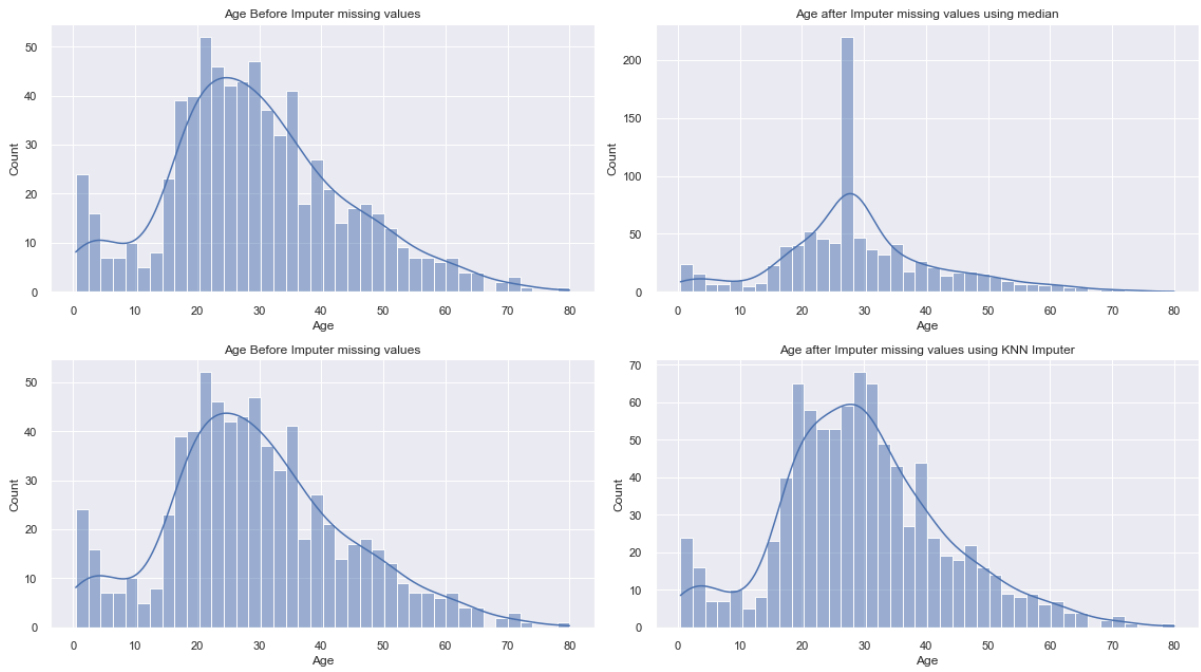


Figure 18 Age before/after filling missing data

The relationships between the features can be visualized through a heatmap displaying the correlation between the features. Identifying strong correlations, can be useful for feature selection and understanding how variables are related in your dataset.

```
plt.figure(figsize=(12, 10))
sns.heatmap(X_imputed.corr(), annot=True)
plt.show()
```

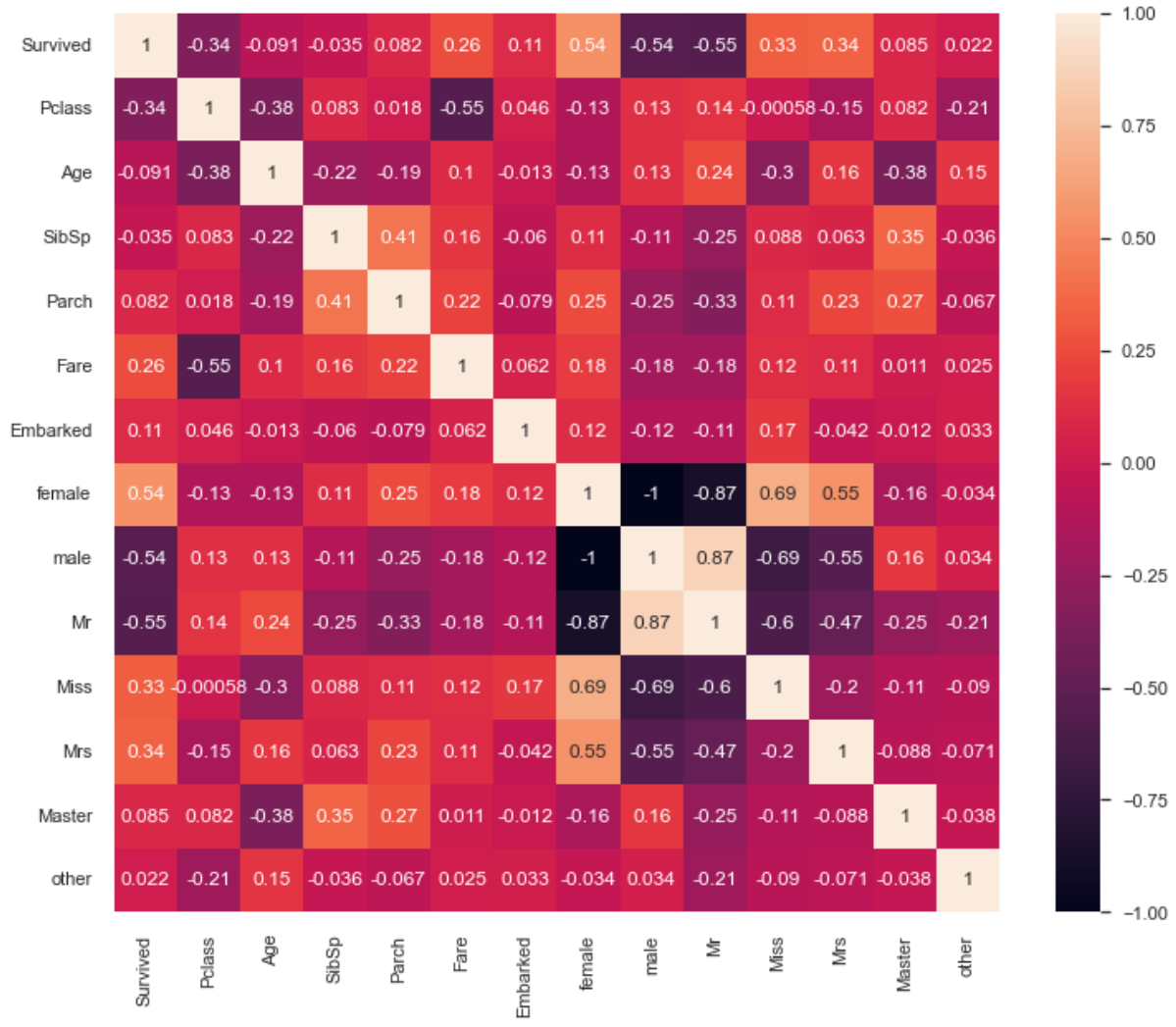


Figure 19 Correlation heatmap

Scaling the values to make a similar scale for all features is important for algorithms that are sensitive to the magnitude of inputs or that are distance based such as k-nearest neighbor, so we scaled the values of Age and Fare using Min Max scaler.

```
X_train_scaled = X_imputed.copy()
scaler = MinMaxScaler()
X_train_scaled[['Age', 'Fare']] = scaler.fit_transform(X_imputed[['Age', 'Fare']])
X_train_scaled
```

	Survived	Pclass	Age	SibSp	Parch	Fare	Embarked	female	male	Mr	Miss	Mrs	Master	other
0	0.0	3.0	0.271174	1.0	0.0	0.014151	0.0	0.0	1.0	1	0	0	0	0
1	1.0	1.0	0.472229	1.0	0.0	0.139136	1.0	1.0	0.0	0	0	1	0	0
2	1.0	3.0	0.321438	0.0	0.0	0.015469	0.0	1.0	0.0	0	1	0	0	0
3	1.0	1.0	0.434531	1.0	0.0	0.103644	0.0	1.0	0.0	0	0	1	0	0
4	0.0	3.0	0.434531	0.0	0.0	0.015713	0.0	0.0	1.0	1	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	0.0	2.0	0.334004	0.0	0.0	0.025374	0.0	0.0	1.0	0	0	0	0	1
887	1.0	1.0	0.233476	0.0	0.0	0.058556	0.0	1.0	0.0	0	1	0	0	0
888	0.0	3.0	0.293164	1.0	2.0	0.045771	0.0	1.0	0.0	0	1	0	0	0
889	1.0	1.0	0.321438	0.0	0.0	0.058556	1.0	0.0	1.0	1	0	0	0	0
890	0.0	3.0	0.396833	0.0	0.0	0.015127	2.0	0.0	1.0	1	0	0	0	0

Figure 20 Final data form

At last, we split the pre-processed data into 3 parts train data, validation data, and test data on a ratio of 70%,15%,15% respectively, then start implementing the model and train it.

```
#split the data into train validation and test
X_train, X_temp, y_train, y_temp = train_test_split(X_imputed.drop
(['Survived', 'other'], axis = 1), X_imputed['Survived'],train_size=
0.7, test_size=0.3,shuffle=False)

X_valid, X_test, y_valid, y_test = train_test_split(X_temp, y_temp,tr
ain_size=0.5, test_size=0.5, shuffle=False)
```

Support Vector Machines

We implement the model using the linear vector classification of the support vector machines with a regularization parameter of 0.1 which was picked after trying different values for this parameter and selected the best value, then fit the model to the train data and validate it on the validation data, then calculate its accuracy and evaluate the model's performance using a confusion matrix to display the correct and incorrect predictions against the actual values.

```python
svm = LinearSVC(C=0.1, dual=False)

svm.fit(X_train, y_train)

print(f"Validation Accuracy: {svm.score(X_valid, y_valid):.2f}")

print(f"Test Accuracy: {svm.score(X_test, y_test):.2f}")

```

```python
svm_score = [[], []]
for i in range(1, 11):
    svm = LinearSVC(C=i/10, dual=False)
    svm.fit(X_train, y_train)
    svm_score[0].append(svm.score(X_valid, y_valid))
    svm_score[1].append(svm.score(X_test, y_test))

svm_scores = pd.DataFrame(svm_score,index=['validation', 'test']).T
svm_scores
```

```python
visualizer = ConfusionMatrix(svm, classes=['Dead', 'Survived'], cmap='YlGnBu')

visualizer.fit(X_train, y_train)        # Fit the visualizer and the model
visualizer.score(X_test, y_test)        # Evaluate the model on the test data
visualizer.show()
```

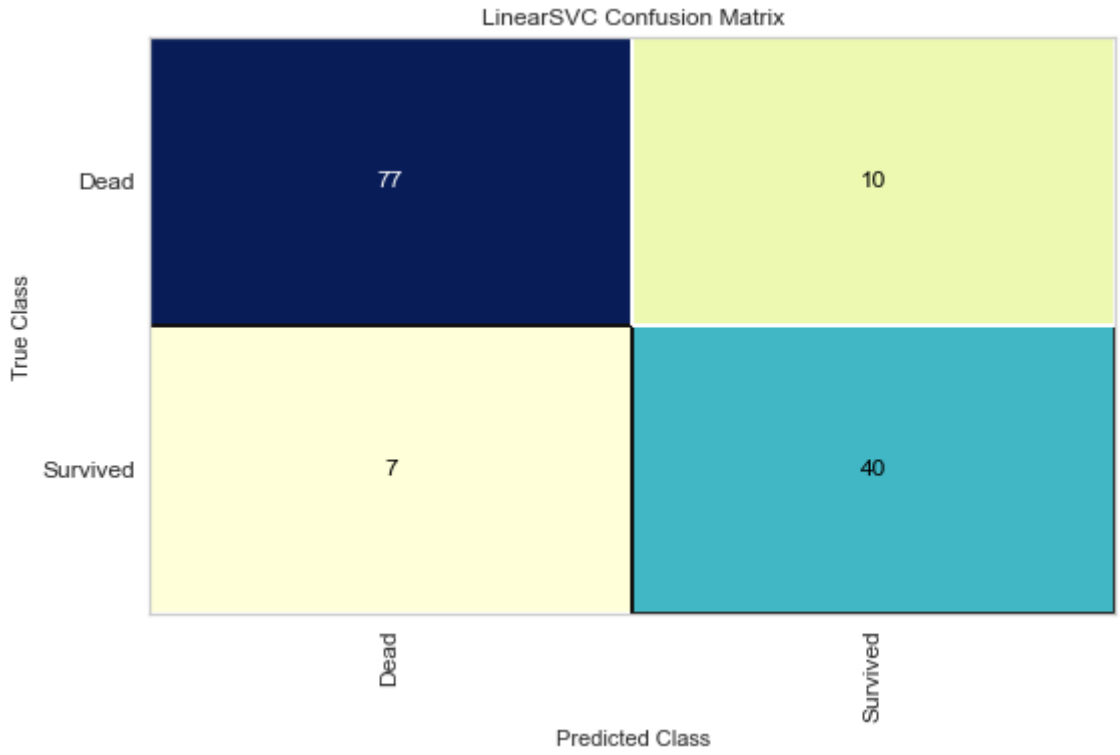


Figure 21 SVC confusion matrix

Evaluating the model's performance using a variety of performance measures, including ROC/AUC curves, precision, recall, and f1-score.

```
visualizer = ClassificationReport(svm, classes=['Dead', 'Survived'], cmap='YlGnBu')
visualizer.fit(X_train, y_train)        # Fit the visualizer and the model
print(visualizer.score(X_test, y_test))       # Evaluate the model on the test data
visualizer.show()                       # Finalize and show the figure
```

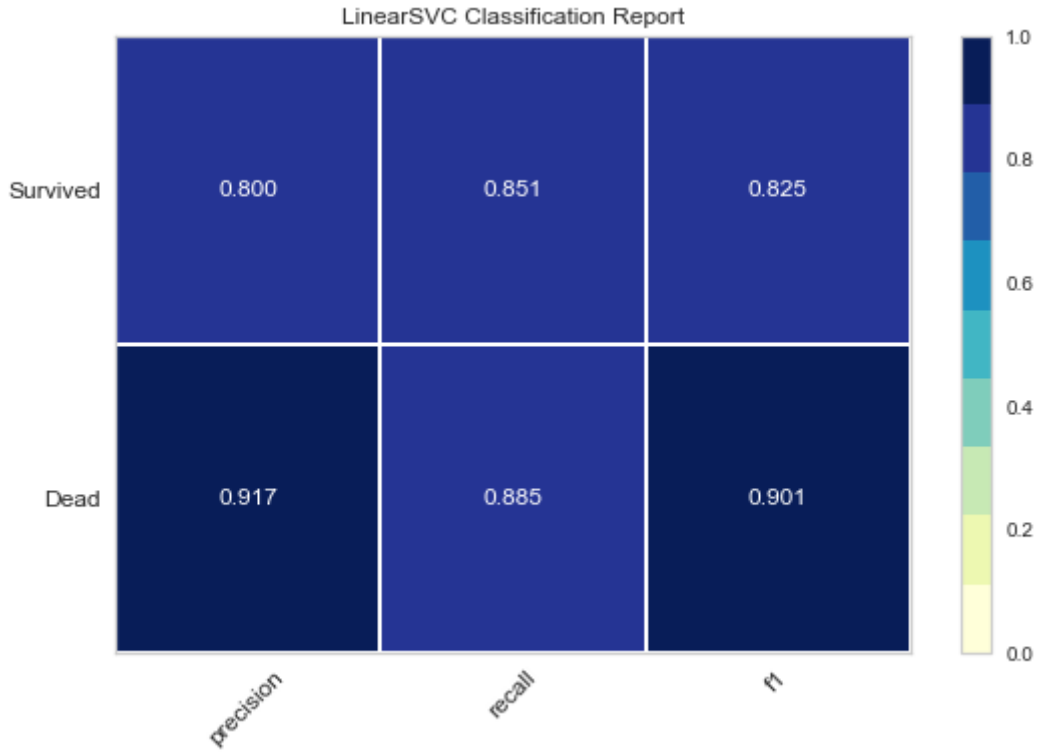


Figure 22 SVC performance metrics

```
visualizer = ROCAUC(LinearSVC(C=0.1, dual=False), classes=['Dead', 'Survived'], cmap='YlGnBu',binary=True)

visualizer.fit(X_train, y_train)        # Fit the training data to the visualizer
visualizer.score(X_test, y_test)        # Evaluate the model on the test data
visualizer.show()
```

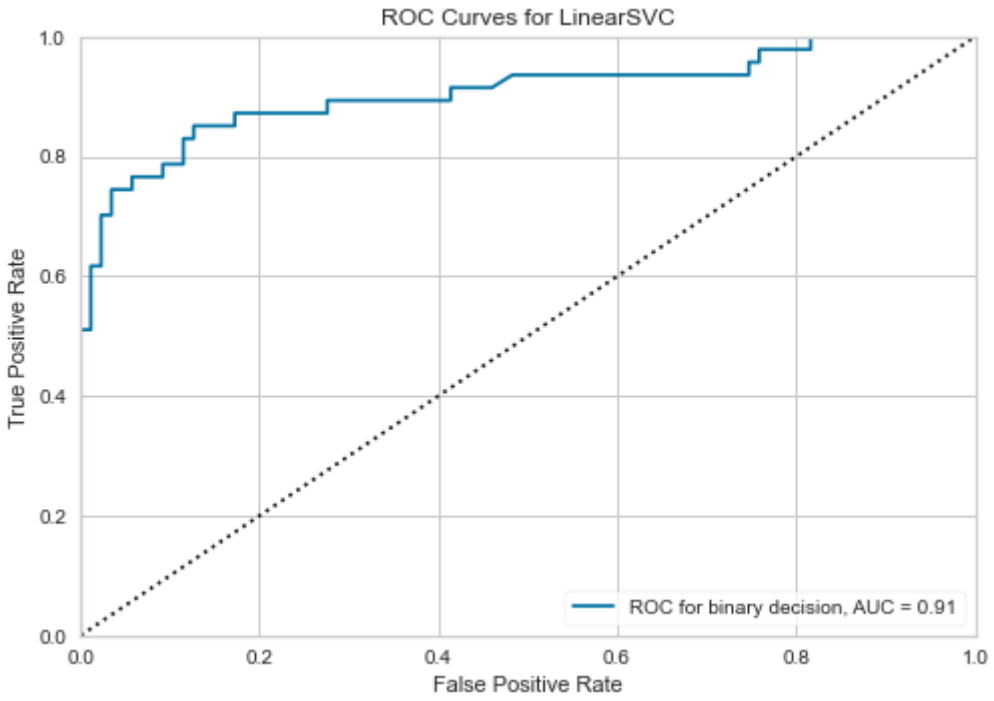


Figure 23 SVC ROC/AUC

Bayes Classifier

We implemented the model using the gaussian naïve bayes classifier, then fitted to the train data and viewed the confusion matrix for a general look on the model's performance.

```
gnb = GaussianNB()

gnb.fit(X_train, y_train)

print(f"Validation Accuracy: {gnb.score(X_valid, y_valid):.2f}")

print(f"Test Accuracy: {gnb.score(X_test, y_test):.2f}")
```

```
visualizer = ConfusionMatrix(gnb, classes=['Dead', 'Survived'], cmap='YlGnBu')

visualizer.fit(X_train, y_train)        # Fit the visualizer and the model
visualizer.score(X_test, y_test)        # Evaluate the model on the test data
visualizer.show()
```

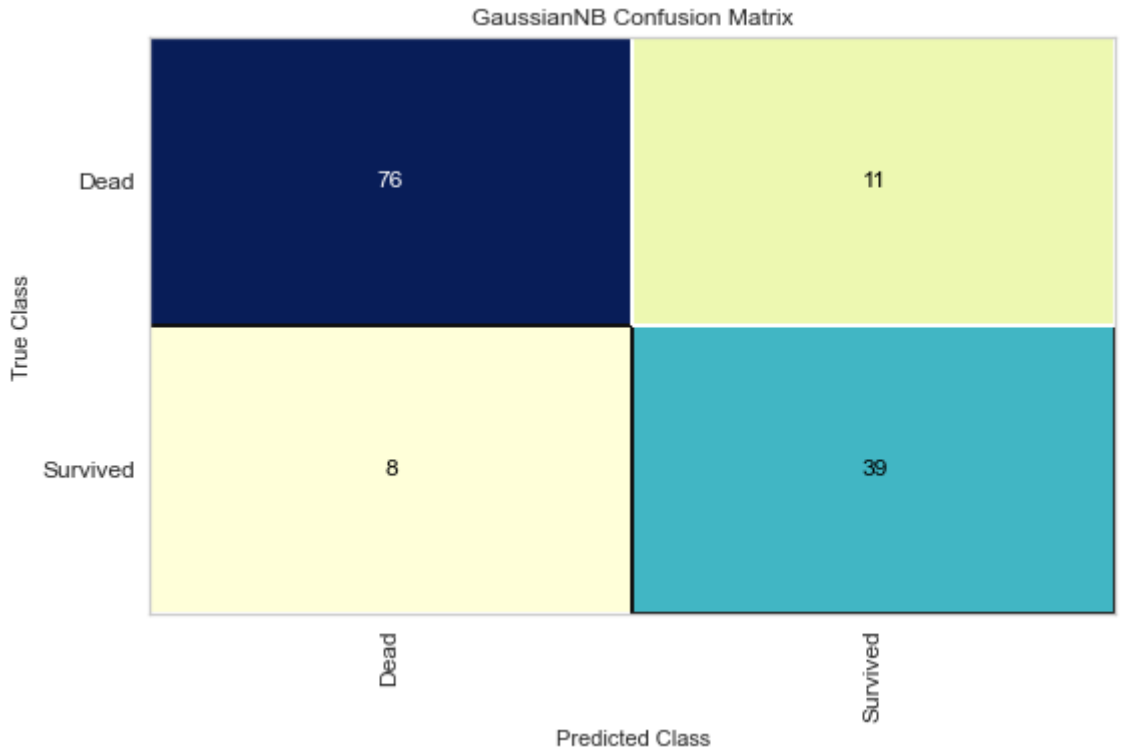


Figure 24 Bayes confusion matrix

Evaluating the model's performance using a variety of performance measures, including ROC/AUC curves, precision, recall, and f1-score.

```
visualizer = ClassificationReport(gnb, classes=['Dead', 'Survived'], cmap='YlGnBu')
visualizer.fit(X_train, y_train)        # Fit the visualizer and the model
print(visualizer.score(X_test, y_test))       # Evaluate the model on the test data
visualizer.show()                       # Finalize and show the figure
```

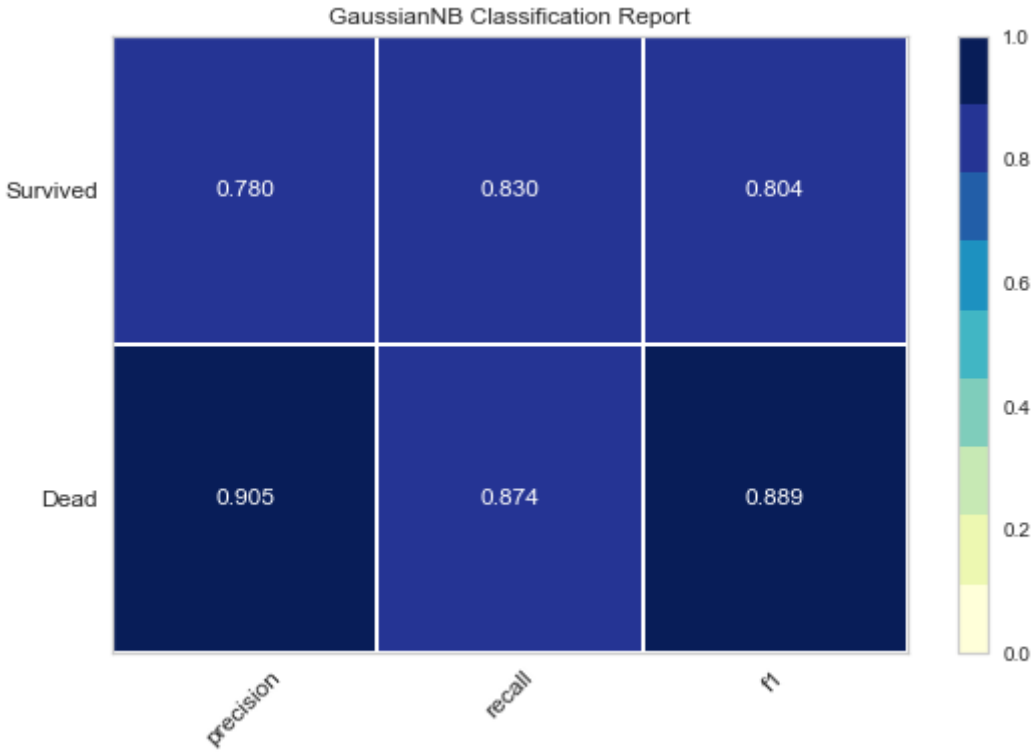


Figure 25 Bayes performance metrics

```
visualizer = ROCAUC(LinearSVC(C=0.1, dual=False), classes=['Dead', 'Survived'], cmap='YlGnBu',binary=True)

visualizer.fit(X_train, y_train)        # Fit the training data to the visualizer
visualizer.score(X_test, y_test)        # Evaluate the model on the test data
visualizer.show()
```

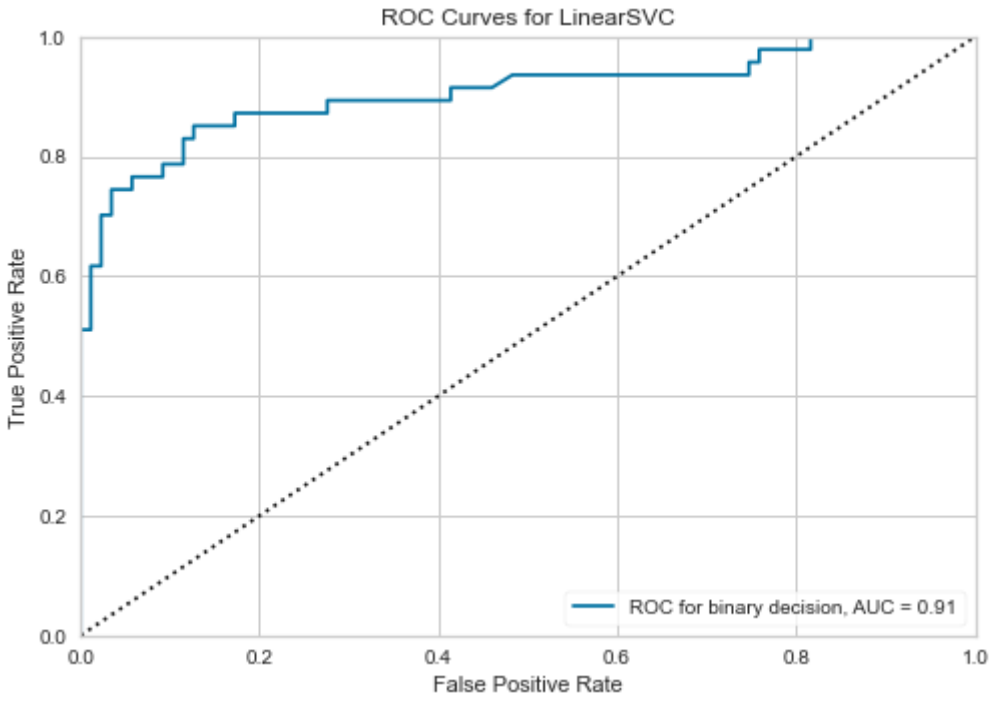


Figure 26 Bayes ROC/AUC

K-Nearest Neighbors

The k-nearest neighbors classifier was used to create the model. The number of neighbors was chosen by experimenting with various values and picking the optimal one that produced high accuracy. The Manhattan distance was also used because it performs better when there are data outliers.

```
knn_score = [[], []]
for i in range(2, 15, 1):

    knn = KNeighborsClassifier(n_neighbors=i, p=1)
    knn.fit(X_train, y_train)
    knn_score[0].append(knn.score(X_valid, y_valid))
    knn_score[1].append(knn.score(X_test, y_test))

knn_score = pd.DataFrame(knn_score,index=['validation', 'test']).T
knn_score
```

```python
knn = KNeighborsClassifier(n_neighbors=8, p=1)
knn.fit(X_train, y_train)

knn.score(X_test, y_test)
```

Using confusion matrix to look upon the performance of the model.

```python
visualizer = ConfusionMatrix(knn, classes=['Dead', 'Survived'], cmap='YlGnBu')

visualizer.fit(X_train, y_train)        # Fit the visualizer and the model
visualizer.score(X_test, y_test)        # Evaluate the model on the test data
visualizer.show()
```

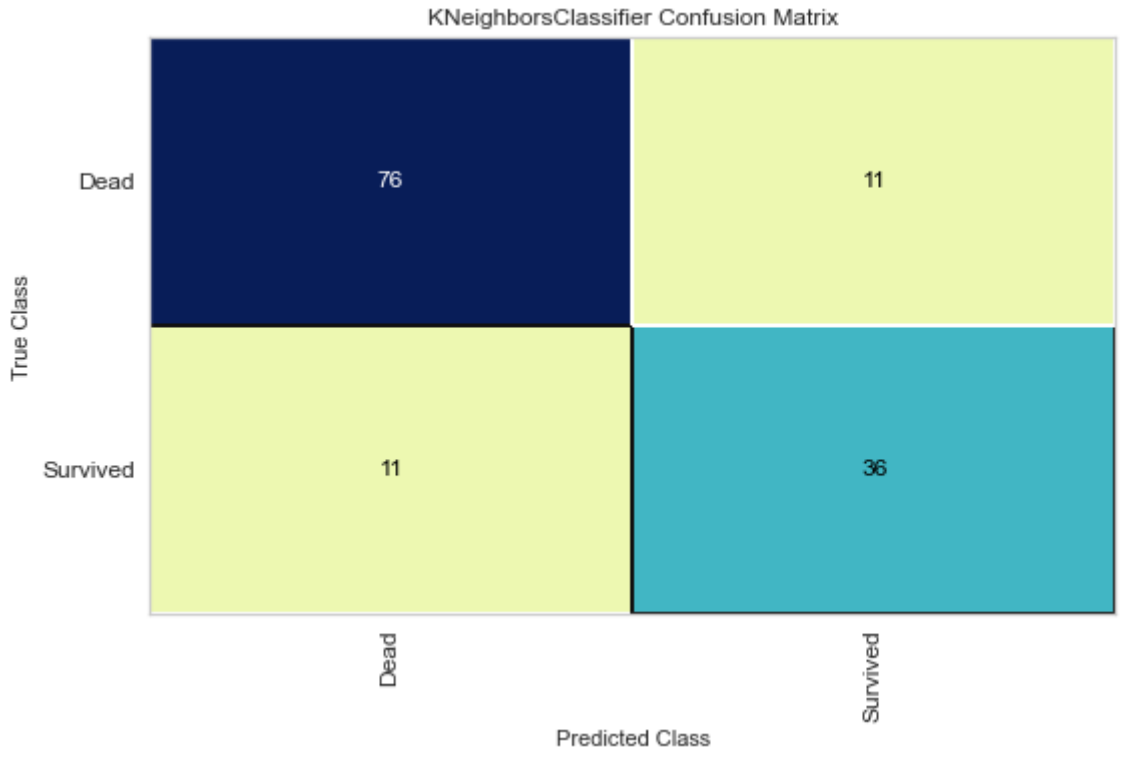


Figure 27 KNN confusion matrix

Evaluating the model's performance using a variety of performance measures, including ROC/AUC curves, precision, recall, and f1-score.

```
visualizer = ClassificationReport(knn, classes=['Dead', 'Survived'], cmap='YlGnBu')
visualizer.fit(X_train, y_train)        # Fit the visualizer and the model
print(visualizer.score(X_test, y_test))       # Evaluate the model on the test data
visualizer.show()                       # Finalize and show the figure
```

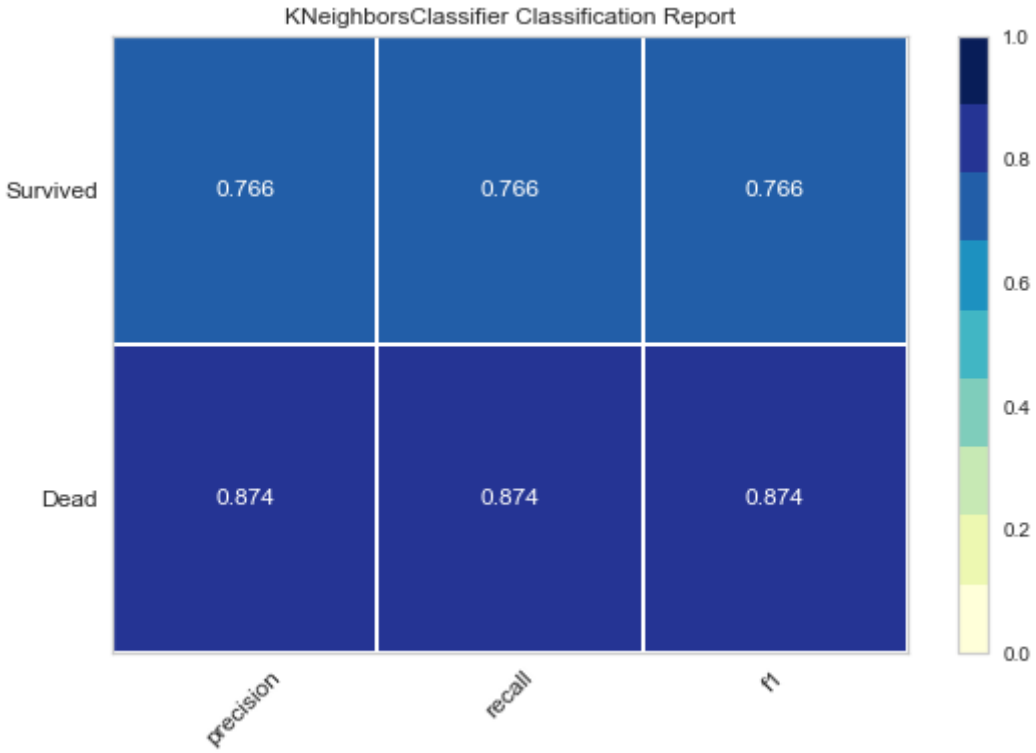


Figure 28 KNN performance metrics

```
visualizer = ROCAUC(knn, classes=['Dead', 'Survived'], cmap='YlGnBu',binary=True)

visualizer.fit(X_train, y_train)        # Fit the training data to the visualizer
visualizer.score(X_test, y_test)        # Evaluate the model on the test data
visualizer.show()
```

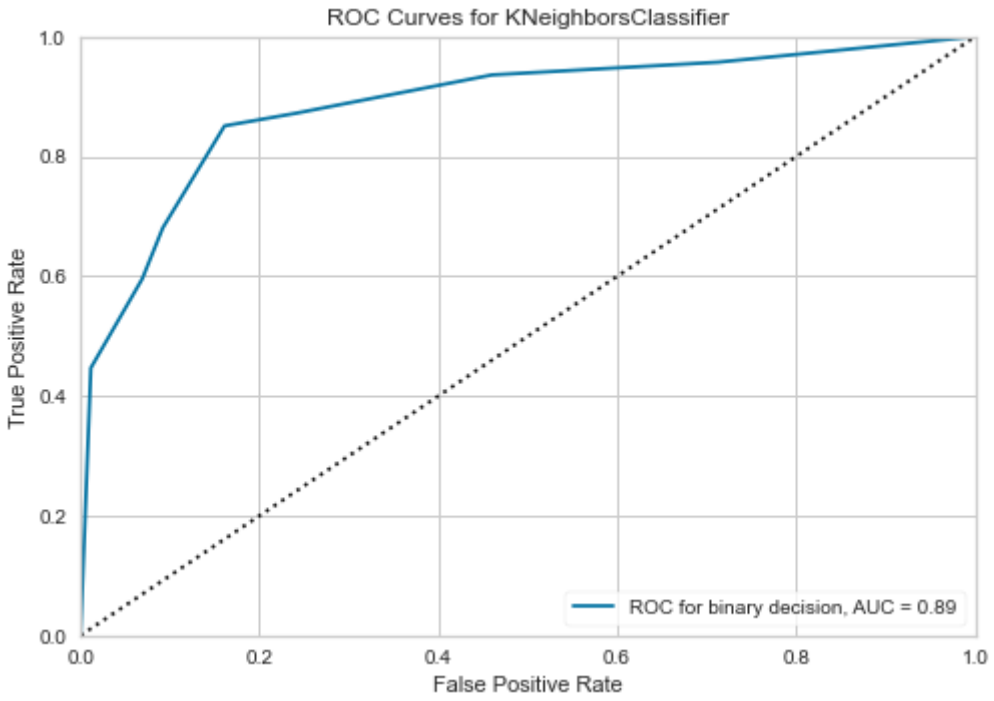


Figure 29 KNN ROC/AUC

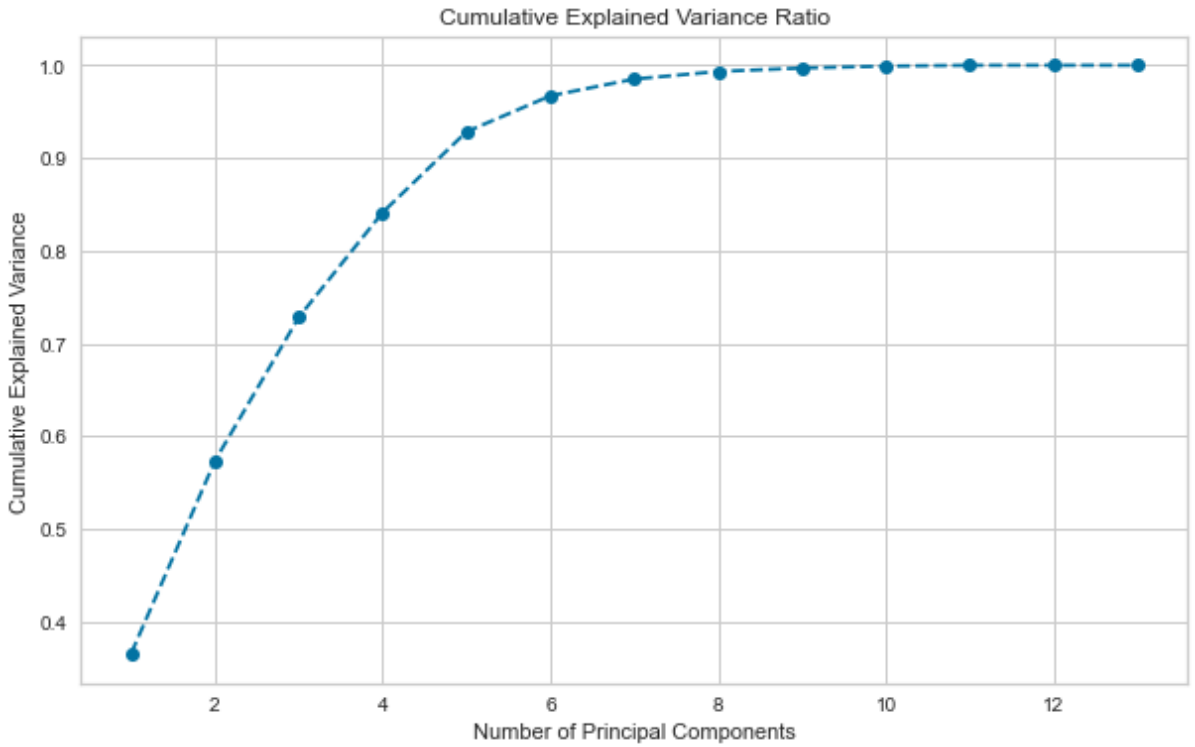
PCA

We applied PCA to the scaled data to reduce its dimensionality and find the best number of components to increase the efficiency of the model and reduce the required processing time and complexity of the model.

```
pca = PCA()
X_pca = pca.fit_transform(X_train_scaled.drop(['Survived'], axis = 1))

# Visualize explained variance ratio
explained_variance_ratio = pca.explained_variance_ratio_
cumulative_variance_ratio = np.cumsum(explained_variance_ratio)

plt.figure(figsize=(10, 6))
plt.plot(range(1, len(explained_variance_ratio) + 1), cumulative_variance_ratio, marker='o', linestyle='--')
plt.title('Cumulative Explained Variance Ratio')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance')
plt.show()
```




```python
pca = PCA(n_components=6)
X_pca = pca.fit_transform(X_train_scaled.drop(['Survived'], axis = 1))
```

```python
svm = LinearSVC(C=0.1, dual=False)

svm.fit(X_train, y_train)

print(f"Validation Accuracy: {svm.score(X_valid, y_valid):.2f}")

print(f"Test Accuracy: {svm.score(X_test, y_test):.2f}")
```

```
Validation Accuracy: 0.77
Test Accuracy: 0.81
```

```
gnb = GaussianNB()

gnb.fit(X_train, y_train)

print(f"Validation Accuracy: {gnb.score(X_valid, y_valid):.2f}")

print(f"Test Accuracy: {gnb.score(X_test, y_test):.2f}")
```

```
Validation Accuracy: 0.78
Test Accuracy: 0.82
```

```
knn = KNeighborsClassifier(n_neighbors=8, p=1)
knn.fit(X_train, y_train)

print(f"Validation Accuracy: {knn.score(X_valid, y_valid):.2f}")

print(f"Test Accuracy: {knn.score(X_test, y_test):.2f}")
```

```
Validation Accuracy: 0.80
Test Accuracy: 0.78
```

The PCA reduced the dimensionality of the data but as a result the accuracy decreased because some features have been dropped.

Conclusion

	SVM	Bayes	KNN
Precision	0.8585	0.8425	0.8275
Recall	0.868	0.852	0.8305
F1	0.863	0.8465	0.829
ROC/AUC	0.91	0.91	0.89

Even though there isn't much of a performance difference between the SVM and other models, the SVM model turned out to be the superior choice based on the performance measures mentioned.

In conclusion, the particular needs and goals of the given work at hand should serve as a guide when choosing the best model. In order to make sure the model selected fits the objectives and limitations of the issue domain, careful evaluation of the advantages and disadvantages of each model is essential.

The git repo for the code is:

https://github.com/OmarMDiab/Titanic-ML_Project