In [1]:
```
from pandas import read_csv, DataFrame, crosstab
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.colors import Normalize
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, LabelEncoder, Stan
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import Perceptron
from imblearn.pipeline import Pipeline as ImbPipeline
from imblearn.over_sampling import RandomOverSampler, SMOTE, SVMSMOTE, BorderlineSM
import joblib
import json
from os.path import exists
```

In [2]:
```
df = read_csv("IIMK_DSAI_W13_Graded Assignment 13.2_Titanic Data Set.csv")
```

In [3]:
```
print(df.head())
```

```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

In [4]:
```
print(df.tail())
```

```
     PassengerId  Survived  Pclass                                     Name  \
886          887         0       2                    Montvila, Rev. Juozas
887          888         1       1             Graham, Miss. Margaret Edith
888          889         0       3  Johnston, Miss. Catherine Helen "Carrie"
889          890         1       1                    Behr, Mr. Karl Howell
890          891         0       3                      Dooley, Mr. Patrick

        Sex   Age  SibSp  Parch      Ticket   Fare Cabin Embarked
886    male  27.0      0      0      211536  13.00   NaN        S
887  female  19.0      0      0      112053  30.00   B42        S
888  female   NaN      1      2  W./C. 6607  23.45   NaN        S
889    male  26.0      0      0      111369  30.00  C148        C
890    male  32.0      0      0      370376   7.75   NaN        Q
```

In [5]:
```
# Check if there are any duplicate values in dataset
print(sum(df.duplicated()))
```

```
0
```

In [6]:
```
print(df.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
None
```

In [7]:
```
na_counts = DataFrame(df.isna().sum(),columns=["NA Counts"]).reset_index()
na_counts = na_counts.rename(columns={'index': 'Column Name'})
print(na_counts)
```

```
    Column Name  NA Counts
0   PassengerId          0
1      Survived          0
2        Pclass          0
3          Name          0
4           Sex          0
5           Age        177
6         SibSp          0
7         Parch          0
8        Ticket          0
9          Fare          0
10        Cabin        687
11     Embarked          2
```

In [8]:
```
df.describe()
```

Out[8]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.00000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.20420
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.69342
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.00000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.91040
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.45420
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.00000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.32920

In [9]:
```
# Printing unique values in datasets columns
for column in ["Survived","Pclass","Sex","SibSp","Parch","Cabin","Embarked"]:
    print(f"{column}:{df[column].unique()}\n")
```

```
Survived:[0 1]

Pclass:[3 1 2]

Sex:['male' 'female']

SibSp:[1 0 3 4 2 5 8]

Parch:[0 1 2 5 3 4 6]

Cabin:[nan 'C85' 'C123' 'E46' 'G6' 'C103' 'D56' 'A6' 'C23 C25 C27' 'B78' 'D33'
 'B30' 'C52' 'B28' 'C83' 'F33' 'F G73' 'E31' 'A5' 'D10 D12' 'D26' 'C110'
 'B58 B60' 'E101' 'F E69' 'D47' 'B86' 'F2' 'C2' 'E33' 'B19' 'A7' 'C49'
 'F4' 'A32' 'B4' 'B80' 'A31' 'D36' 'D15' 'C93' 'C78' 'D35' 'C87' 'B77'
 'E67' 'B94' 'C125' 'C99' 'C118' 'D7' 'A19' 'B49' 'D' 'C22 C26' 'C106'
 'C65' 'E36' 'C54' 'B57 B59 B63 B66' 'C7' 'E34' 'C32' 'B18' 'C124' 'C91'
 'E40' 'T' 'C128' 'D37' 'B35' 'E50' 'C82' 'B96 B98' 'E10' 'E44' 'A34'
 'C104' 'C111' 'C92' 'E38' 'D21' 'E12' 'E63' 'A14' 'B37' 'C30' 'D20' 'B79'
 'E25' 'D46' 'B73' 'C95' 'B38' 'B39' 'B22' 'C86' 'C70' 'A16' 'C101' 'C68'
 'A10' 'E68' 'B41' 'A20' 'D19' 'D50' 'D9' 'A23' 'B50' 'A26' 'D48' 'E58'
 'C126' 'B71' 'B51 B53 B55' 'D49' 'B5' 'B20' 'F G63' 'C62 C64' 'E24' 'C90'
 'C45' 'E8' 'B101' 'D45' 'C46' 'D30' 'E121' 'D11' 'E77' 'F38' 'B3' 'D6'
 'B82 B84' 'D17' 'A36' 'B102' 'B69' 'E49' 'C47' 'D28' 'E17' 'A24' 'C50'
 'B42' 'C148']

Embarked:['S' 'C' 'Q' nan]
```

In [10]:

```
#Printing unique values in dataset columns
for column in ["Survived","Pclass","Sex","SibSp","Parch","Cabin","Embarked"]:
    print(f"{df[column].value_counts()}{df[column].value_counts(normalize=True)}\n"
```

```
Survived
0    549
1    342
Name: count, dtype: int64Survived
0    0.616162
1    0.383838
Name: proportion, dtype: float64

Pclass
3    491
1    216
2    184
Name: count, dtype: int64Pclass
3    0.551066
1    0.242424
2    0.206510
Name: proportion, dtype: float64

Sex
male      577
female    314
Name: count, dtype: int64Sex
male      0.647587
female    0.352413
Name: proportion, dtype: float64

SibSp
0    608
1    209
2     28
4     18
3     16
8      7
5      5
Name: count, dtype: int64SibSp
0    0.682379
1    0.234568
2    0.031425
4    0.020202
3    0.017957
8    0.007856
5    0.005612
Name: proportion, dtype: float64

Parch
0    678
1    118
2     80
5      5
3      5
4      4
6      1
Name: count, dtype: int64Parch
0    0.760943
1    0.132435
2    0.089787
```

```
5    0.005612
3    0.005612
4    0.004489
6    0.001122
Name: proportion, dtype: float64

Cabin
B96 B98        4
G6             4
C23 C25 C27    4
C22 C26        3
F33            3
              ..
E34            1
C7             1
C54            1
E36            1
C148           1
Name: count, Length: 147, dtype: int64Cabin
B96 B98        0.019608
G6             0.019608
C23 C25 C27    0.019608
C22 C26        0.014706
F33            0.014706
                 ...
E34            0.004902
C7             0.004902
C54            0.004902
E36            0.004902
C148           0.004902
Name: proportion, Length: 147, dtype: float64

Embarked
S    644
C    168
Q     77
Name: count, dtype: int64Embarked
S    0.724409
C    0.188976
Q    0.086614
Name: proportion, dtype: float64
```

In [11]:

```
# Replacing NA in Age with the mean age
mean_Age = df['Age'].mean()
df['Age'] = df['Age'].fillna(mean_Age)
```

In [12]:

```
# As there are several NA in Cabin, for better analysis making another column is_Ca
df['is_Cabin'] = np.where(df['Cabin'].isna(), 0, 1)
df = df.drop('Cabin', axis = 1)
```

In [13]:

```
# Replacing NA in Embarked with the mode
mode_Embarked = df['Embarked'].mode()
df['Embarked'] = df['Embarked'].fillna(mode_Embarked[0])
```

In [14]:
```
# Now checking for NA
na_counts = DataFrame(df.isna().sum(),columns=["NA Counts"]).reset_index()
na_counts = na_counts.rename(columns={'index': 'Column Name'})
print(na_counts)
```

```
   Column Name  NA Counts
0   PassengerId          0
1      Survived          0
2        Pclass          0
3          Name          0
4           Sex          0
5           Age          0
6         SibSp          0
7         Parch          0
8        Ticket          0
9          Fare          0
10     Embarked          0
11     is_Cabin          0
```

In [15]:
```
# Printing unique values in datasets columns
for column in ["Survived","Pclass","Sex","SibSp","Parch","is_Cabin","Embarked"]:
    print(f"{column}:{df[column].unique()}\n")
```

```
Survived:[0 1]

Pclass:[3 1 2]

Sex:['male' 'female']

SibSp:[1 0 3 4 2 5 8]

Parch:[0 1 2 5 3 4 6]

is_Cabin:[0 1]

Embarked:['S' 'C' 'Q']
```

In [16]:
```
#Printing unique values in dataset columns
for column in ["Survived","Pclass","Sex","SibSp","Parch","is_Cabin","Embarked"]:
    print(f"{df[column].value_counts()}{df[column].value_counts(normalize=True)}\n"
```

```
Survived
0    549
1    342
Name: count, dtype: int64Survived
0    0.616162
1    0.383838
Name: proportion, dtype: float64

Pclass
3    491
1    216
2    184
Name: count, dtype: int64Pclass
3    0.551066
1    0.242424
2    0.206510
Name: proportion, dtype: float64

Sex
male      577
female    314
Name: count, dtype: int64Sex
male      0.647587
female    0.352413
Name: proportion, dtype: float64

SibSp
0    608
1    209
2     28
4     18
3     16
8      7
5      5
Name: count, dtype: int64SibSp
0    0.682379
1    0.234568
2    0.031425
4    0.020202
3    0.017957
8    0.007856
5    0.005612
Name: proportion, dtype: float64

Parch
0    678
1    118
2     80
5      5
3      5
4      4
6      1
Name: count, dtype: int64Parch
0    0.760943
1    0.132435
2    0.089787
```

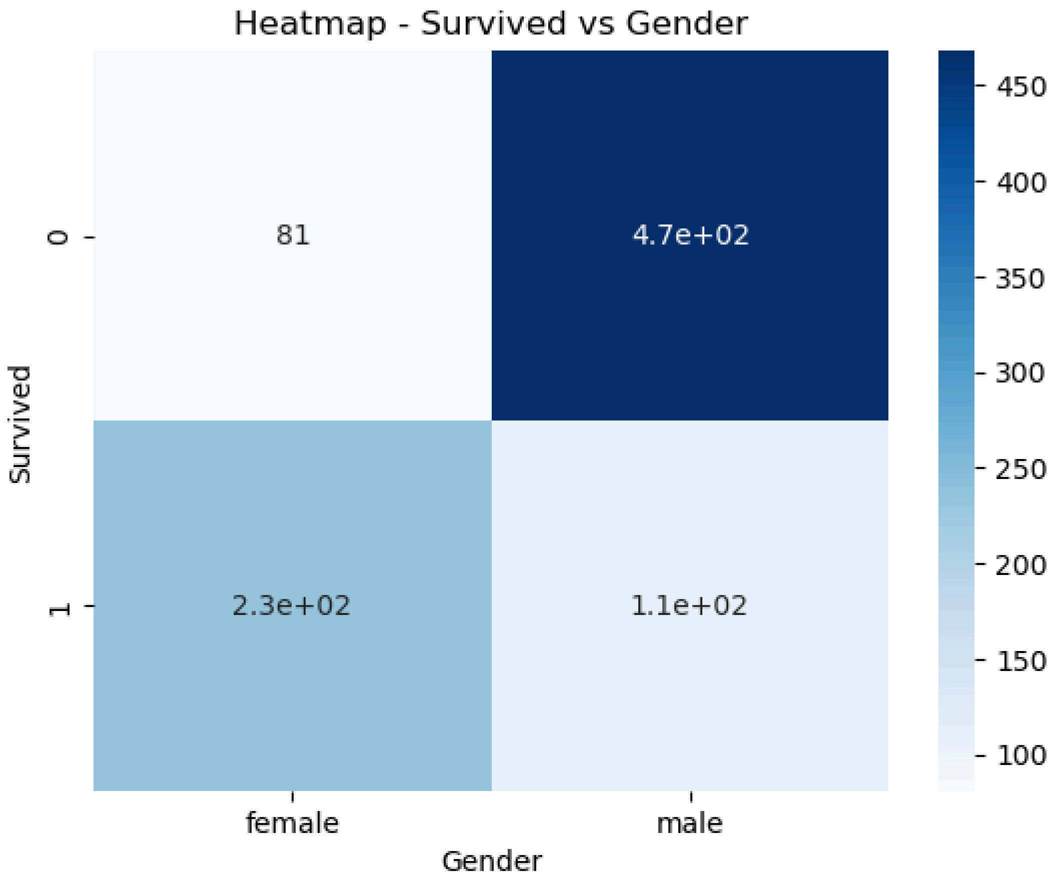
```
5    0.005612
3    0.005612
4    0.004489
6    0.001122
Name: proportion, dtype: float64

is_Cabin
0    687
1    204
Name: count, dtype: int64is_Cabin
0    0.771044
1    0.228956
Name: proportion, dtype: float64

Embarked
S    646
C    168
Q     77
Name: count, dtype: int64Embarked
S    0.725028
C    0.188552
Q    0.086420
Name: proportion, dtype: float64
```

Exploratory Data Analysis

As we know that females, elderly and children were preferred to be saved via lifeboats let us check the survival on basis of these two parameters. Then let us analyze it in terms of Pclass to check if there was a preference to save 1st class passengers.

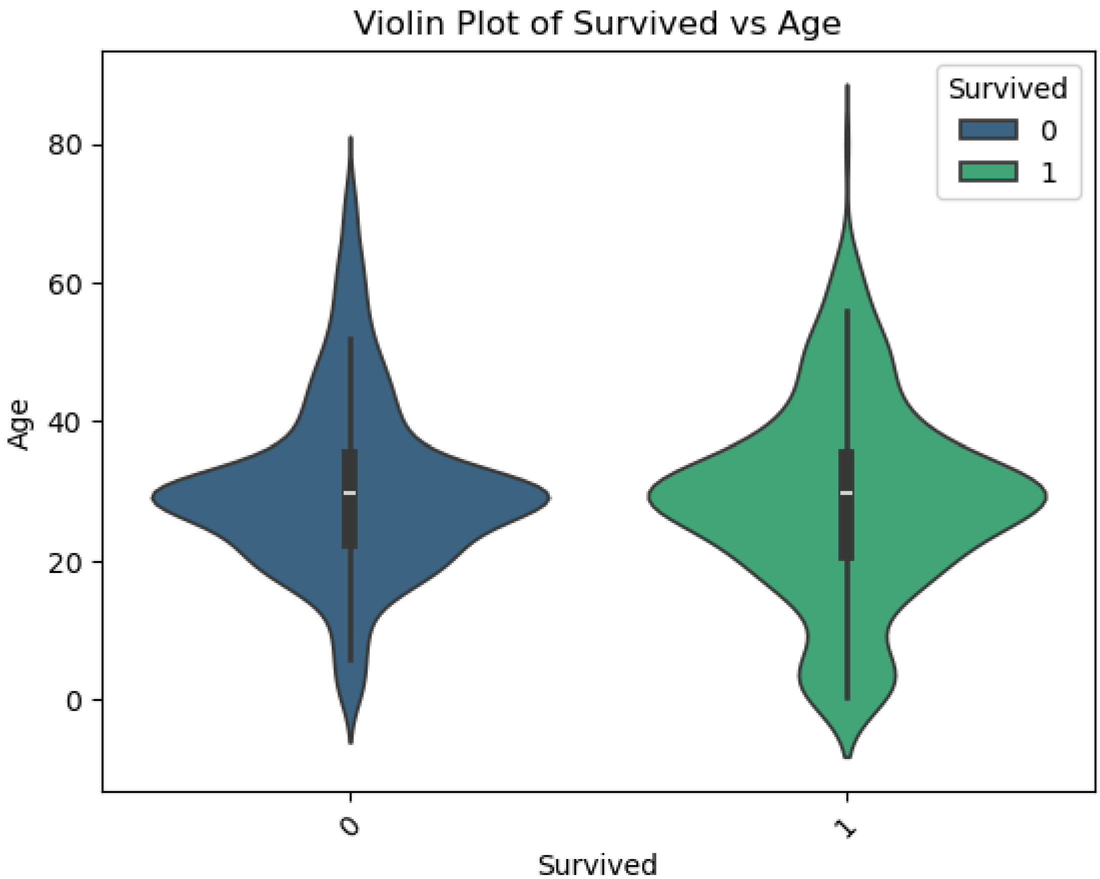
In [17]:

```python
# Create a heatmap directly from the DataFrame
sns.heatmap(crosstab(df['Survived'], df['Sex']), cmap='Blues', annot=True)  # Adjus

# Customize the plot
plt.xlabel('Gender')
plt.ylabel('Survived')
plt.title('Heatmap - Survived vs Gender')
plt.show()
```



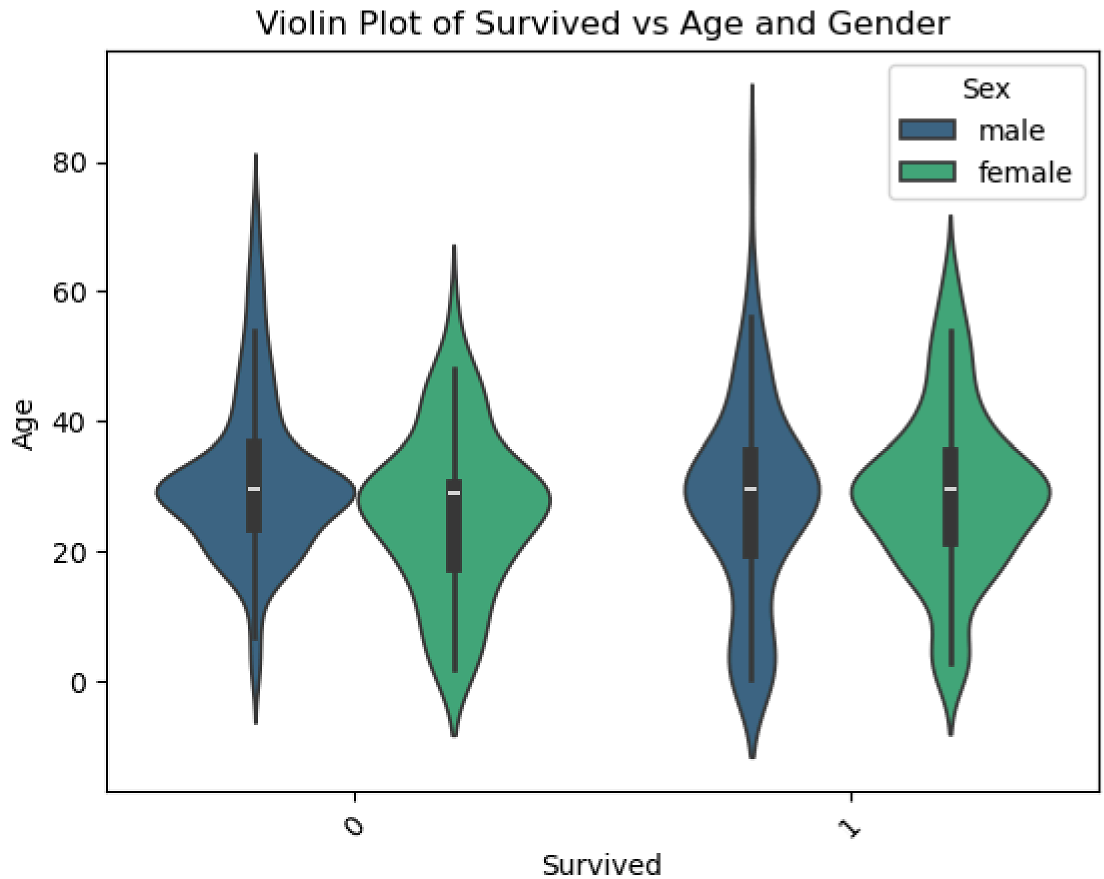
In [18]:

```
# Create the violin plot
sns.violinplot(x = "Survived",
               y = "Age",
               data = df,
               hue = "Survived",
               palette = "viridis")

# Customize the plot
plt.xlabel("Survived")
plt.ylabel("Age")
plt.title("Violin Plot of Survived vs Age")
plt.xticks(rotation=45)
plt.show()
```



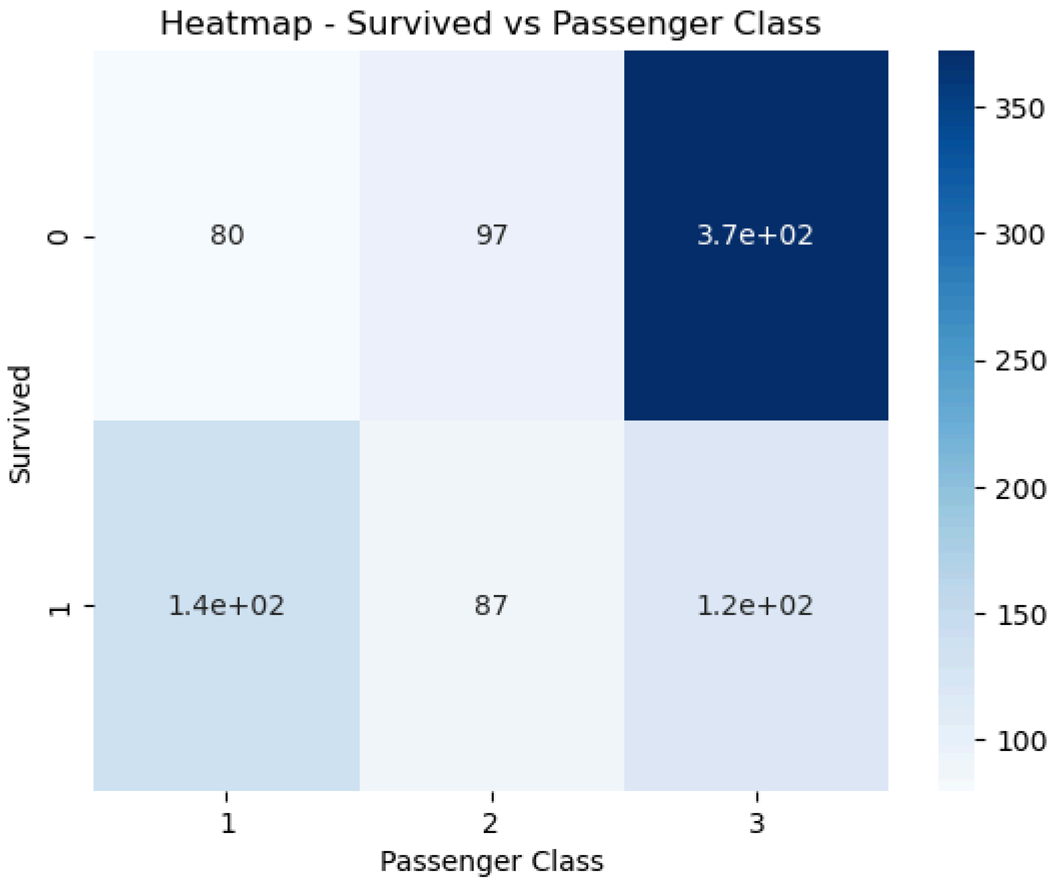
In [19]:

```
# Create the violin plot
sns.violinplot(x = "Survived",
               y = "Age",
               data = df,
               hue = "Sex",
               palette = "viridis")

# Customize the plot
plt.xlabel("Survived")
plt.ylabel("Age")
plt.title("Violin Plot of Survived vs Age and Gender")
plt.xticks(rotation=45)
plt.show()
```

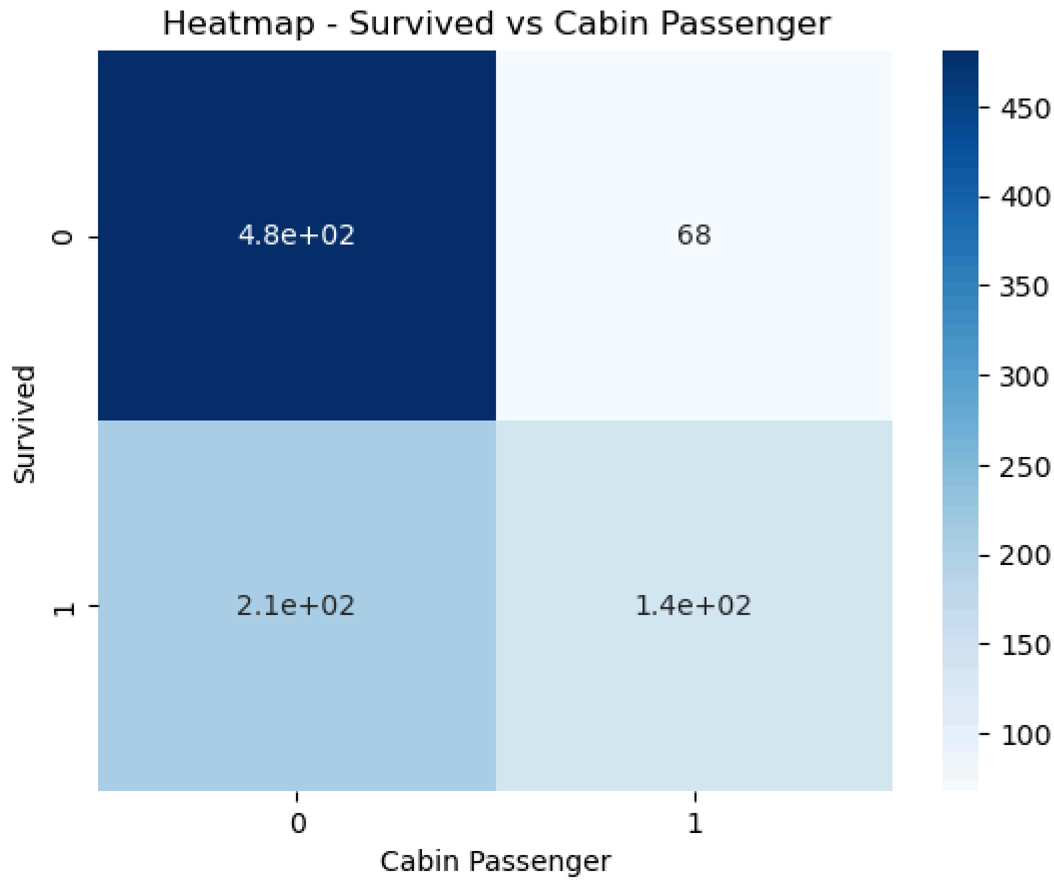


In [20]:

```
# Create a heatmap directly from the DataFrame
sns.heatmap(crosstab(df['Survived'], df['Pclass']), cmap='Blues', annot=True)  # Ad

# Customize the plot
plt.xlabel('Passenger Class')
plt.ylabel('Survived')
plt.title('Heatmap - Survived vs Passenger Class')
plt.show()
```



In [21]:
```python
# Create a heatmap directly from the DataFrame
sns.heatmap(crosstab(df['Survived'], df['is_Cabin']), cmap='Blues', annot=True)  #

# Customize the plot
plt.xlabel('Cabin Passenger')
plt.ylabel('Survived')
plt.title('Heatmap - Survived vs Cabin Passenger')
plt.show()
```



We see a clear indication that female passengers are more likely to survive. Children and the Elderly were also more likely to survive especially in males. Passengers in 1st Class are more likely to survive than passengers in 2nd Class and 3rd Class. Passengers in Cabin are also more likely to survive than passengers not having cabin tickets.

Splitting the Training and Testing Data Set

In [22]:
```
# The only features we will use for further modelling - SibSp, Pclass, Sex, Age, Pa
X,y = df.drop(["PassengerId","Name","Ticket","Survived"],axis=1), df["Survived"]

# Split data into training and testing sets (default test_size=0.2) # Through trial
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta
```

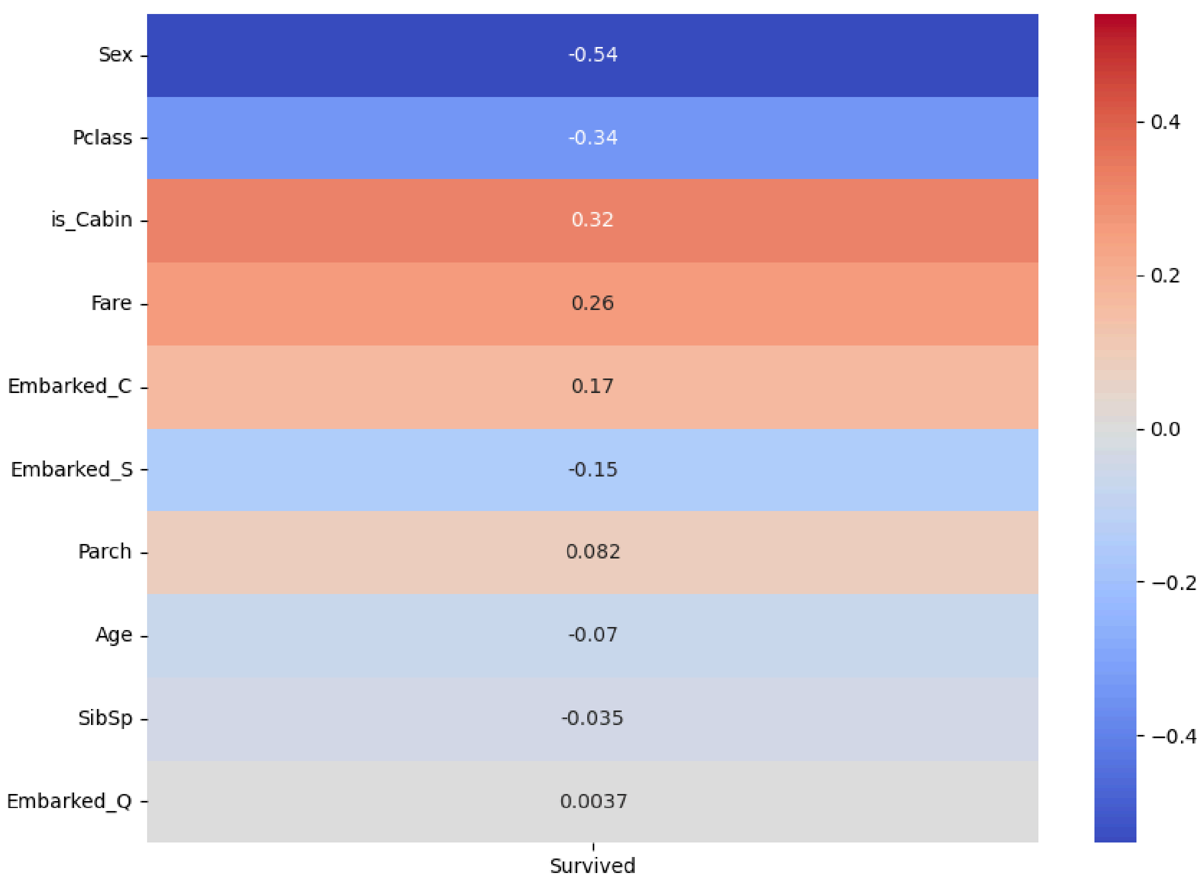
In [23]:
```
print("Training Data Shape")
print(X_train.shape)
print("Testing Data Shape")
print(X_test.shape)
```

```
Training Data Shape
(623, 8)
Testing Data Shape
(268, 8)
```

In [24]:
```
# Define column names
ordinal_cols = ['Sex']
```

```
onehot_cols = ['Embarked']
numerical_cols = [col for col in X.columns if col not in ordinal_cols + onehot_cols

# Define the ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
        ('ordinal', OrdinalEncoder(), ordinal_cols),
        ('onehot', OneHotEncoder(), onehot_cols),
        ('num', 'passthrough', numerical_cols)
    ]
)
```

In [25]:
```
transformed_df = preprocessor.fit_transform(df.drop(["PassengerId","Name","Ticket"]
transformed_df = DataFrame(transformed_df, columns=['Sex', 'Embarked_C', 'Embarked_
transformed_df["Survived"] = df["Survived"].astype(float)
transformed_df["Embarked_S"] = transformed_df.apply(lambda row: 1.0 if row['Embarke

corr = transformed_df.corr()
norm = Normalize(vmin=-0.54, vmax=0.54)
plt.figure(figsize=(10, 7.5))
sns.heatmap(corr["Survived"].to_frame().sort_values(by="Survived", key=lambda x: x.
            annot=True, cmap = "coolwarm",norm = norm)
plt.show()
```



As per the above heatmap we can analyze the feature importance and correlation with the target variable Survived. By this analysis, we find the maximum importance is of gender followed by Passenger Class, whether the passenger has a cabin ticket and fare. The other features have relatively lesser importance due to its low correlation with the target variable.

KNN

In [26]:

```
# If model already run from the existing model or else define the model
if exists ('knn_model.joblib'):
    print("Loading from file")
    knn_loaded = joblib.load('knn_model.joblib')
    y_pred_knn = knn_loaded.predict(X_test)
    y_pred_proba_knn = knn_loaded.predict_proba(X_test)
    with open('knn_results.json', 'r') as json_file:
        results = json.load(json_file)
        optimal_params = results['optimal_params']
        optimal_accuracy = results['optimal_accuracy']
        print(f'Best parameters: {optimal_params}')
        print(f'Best cross-validation accuracy: {optimal_accuracy:.5f}')
else:
    # Define the pipeline
    pipeline = Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('scaler', StandardScaler()),  # Apply scaling to all columns after preproc
        ('knn', KNeighborsClassifier())
    ])

    # Define the parameter grid for GridSearchCV
    param_grid = {
        'knn__n_neighbors': [3, 5, 7, 9, 11],
        'knn__weights': ['uniform', 'distance'],
        'knn__metric': ['euclidean', 'manhattan', 'minkowski', 'canberra', 'braycur
        'knn__p': [1.5, 2.5]
    }

    # Define the GridSearchCV
    grid_search = GridSearchCV(pipeline, param_grid, cv=5, n_jobs = -1)

    # Perform GridSearchCV to find the best parameters and fit the model
    grid_search.fit(X_train, y_train)

    # Update the pipeline with the best estimator
    optimal_estimator = grid_search.best_estimator_

    y_pred_knn = optimal_estimator.predict(X_test)
    y_pred_proba_knn = optimal_estimator.predict_proba(X_test)

    # Print the best parameters and cross-validation accuracy
    print(f'Best parameters: {(optimal_params:= grid_search.best_params_)}')
    print(f'Best cross-validation accuracy: {(optimal_accuracy:= grid_search.best_s

    # Writing it in json file
    results = {'optimal_params': optimal_params, 'optimal_accuracy': optimal_accura
    with open('knn_results.json', 'w') as json_file:
        json.dump(results, json_file)

    # Save the pipeline
```

```
    joblib.dump(optimal_estimator, 'knn_model.joblib')
    print("Model trained and saved to disk.")
```

```
Loading from file
Best parameters: {'knn__metric': 'euclidean', 'knn__n_neighbors': 9, 'knn__p': 1.5,
'knn__weights': 'uniform'}
Best cross-validation accuracy: 0.81218
```

Model Evaluation

In [27]:
```
accuracy_knn = accuracy_score(y_test, y_pred_knn)
print(f"The accuracy of KNN model is {accuracy_knn:5f}")
```

```
The accuracy of KNN model is 0.850746
```

In [28]:
```
print(confusion_matrix(y_test, y_pred_knn))
```

```
[[166  12]
 [ 28  62]]
```

In [29]:
```
print(classification_report(y_test,y_pred_knn))
```

```
              precision    recall  f1-score   support

           0       0.86      0.93      0.89       178
           1       0.84      0.69      0.76        90

    accuracy                           0.85       268
   macro avg       0.85      0.81      0.82       268
weighted avg       0.85      0.85      0.85       268
```

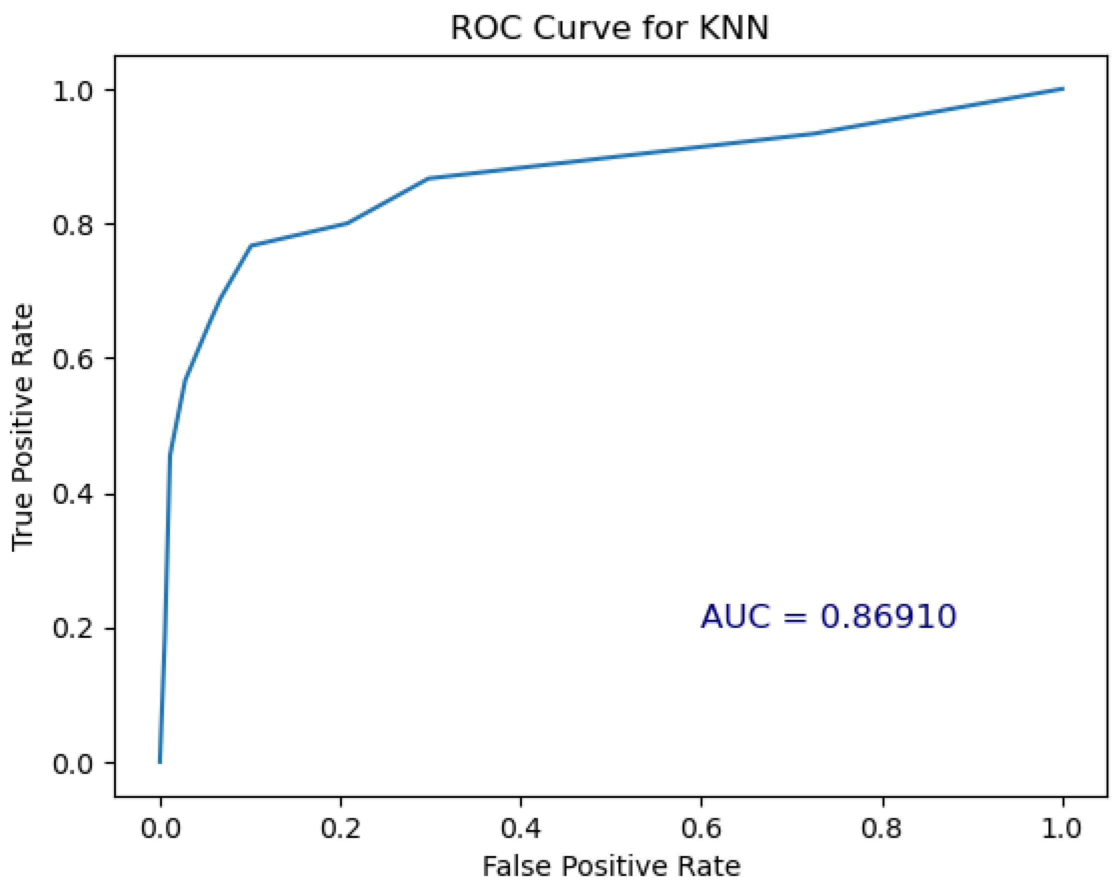
In [30]:
```
f1_knn = f1_score(y_test,y_pred_knn)
print(f"The f1 score of KNN model is {f1_knn:5f}")
```

```
The f1 score of KNN model is 0.756098
```

In [31]:
```
roc_auc_knn = roc_auc_score(y_test, y_pred_proba_knn[:,1])
print(f"The ROC-AUC score of KNN model is {roc_auc_knn:5f}")
```

```
The ROC-AUC score of KNN model is 0.869101
```

In [32]:
```
fpr_knn, tpr_knn, _ = roc_curve(y_test, y_pred_proba_knn[:, 1])
plt.plot(fpr_knn, tpr_knn, label='KNN (AUC = {:.5f})'.format(roc_auc_knn))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for KNN')
plt.text(0.6, 0.2, f'AUC = {roc_auc_knn:.5f}', fontsize=12, color='navy')
plt.show()
```



Gaussian Naive Bayes

In [33]:

```
# If model already run from the existing model or else define the model
if exists ('ppp_model.joblib'):
    print("Loading from file")
    ppp_loaded = joblib.load('ppp_model.joblib')
    y_pred_ppp = ppp_loaded.predict(X_test)
    y_pred_proba_ppp = ppp_loaded.decision_function(X_test)
    with open('ppp_results.json', 'r') as json_file:
        results = json.load(json_file)
        optimal_params = results['optimal_params']
        optimal_accuracy = results['optimal_accuracy']
        print(f'Best parameters: {optimal_params}')
        print(f'Best cross-validation accuracy: {optimal_accuracy:.5f}')
else:
    # Define the pipeline
    pipeline = ImbPipeline(steps=[
        ('preprocessor', preprocessor),
        ('scaler', StandardScaler()),  # Apply scaling to all columns after preproc
        ('smote', BorderlineSMOTE(random_state=21)),
        ('perceptron', Perceptron(random_state = 4))
    ])

    # Define the parameter grid for GridSearchCV
    param_grid = {
        'perceptron__max_iter': [50, 100],          # Maximum number of iterations
```

```
        'perceptron__eta0': [0.1, 0.01, 0.001, 0.0001],          # Initial learnin
    }

    # Define the GridSearchCV
    grid_search = GridSearchCV(pipeline, param_grid, cv=5)

    # Perform GridSearchCV to find the best parameters and fit the model
    grid_search.fit(X_train, y_train)

    # Update the pipeline with the best estimator
    optimal_estimator = grid_search.best_estimator_

    y_pred_ppp = optimal_estimator.predict(X_test)
    y_pred_proba_ppp = optimal_estimator.decision_function(X_test)

    # Print the best parameters and cross-validation accuracy
    print(f'Best parameters: {(optimal_params:= grid_search.best_params_)}')
    print(f'Best cross-validation accuracy: {(optimal_accuracy:= grid_search.best_s

    # Writing it in json file
    results = {'optimal_params': optimal_params, 'optimal_accuracy': optimal_accura
    with open('ppp_results.json', 'w') as json_file:
        json.dump(results, json_file)

    # Save the pipeline
    joblib.dump(optimal_estimator, 'ppp_model.joblib')
    print("Model trained and saved to disk.")
```

```
Loading from file
Best parameters: {'perceptron__eta0': 0.01, 'perceptron__max_iter': 50}
Best cross-validation accuracy: 0.74315
```

Model Evaluation

In [34]:
```
accuracy_ppp = accuracy_score(y_test, y_pred_ppp)
print(f"The accuracy of Perceptron model is {accuracy_ppp:5f}")
```

```
The accuracy of Perceptron model is 0.805970
```

In [35]:
```
print(confusion_matrix(y_test, y_pred_ppp))
```

```
[[148  30]
 [ 22  68]]
```

In [36]:
```
print(classification_report(y_test,y_pred_ppp))
```

```
              precision    recall  f1-score   support

           0       0.87      0.83      0.85       178
           1       0.69      0.76      0.72        90

    accuracy                           0.81       268
   macro avg       0.78      0.79      0.79       268
weighted avg       0.81      0.81      0.81       268
```

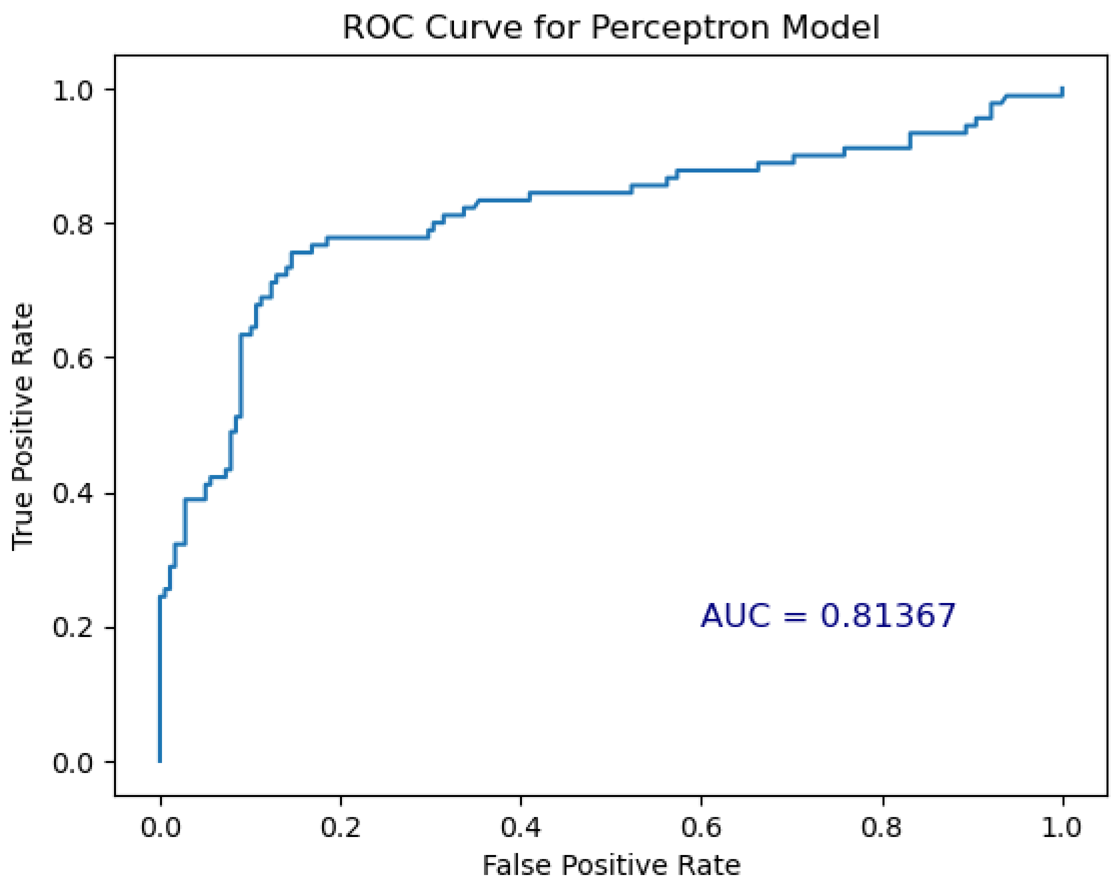
In [37]:
```python
f1_ppp = f1_score(y_test,y_pred_ppp)
print(f"The f1 score of Perceptron model is {f1_ppp:5f}")
```

```
The f1 score of Perceptron model is 0.723404
```

In [38]:
```python
roc_auc_ppp = roc_auc_score(y_test, y_pred_proba_ppp)
print(f"The ROC-AUC score of Perceptron model is {roc_auc_ppp:5f}")
```

```
The ROC-AUC score of Perceptron model is 0.813670
```

In [39]:
```python
fpr_ppp, tpr_ppp, _ = roc_curve(y_test, y_pred_proba_ppp)
plt.plot(fpr_ppp, tpr_ppp, label='Perceptron (AUC = {:.5f})'.format(roc_auc_ppp))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Perceptron Model')
plt.text(0.6, 0.2, f'AUC = {roc_auc_ppp:.5f}', fontsize=12, color='navy')
plt.show()
```



Custom Perceptron

In [40]:
```python
class Perceptron1(object):
    #eta : float, Learning rate (between 0.0 and 1.0)
    #n_iter : int, Passes over the training dataset.
    #random_state : int, Random number generator seed for random weight
      #initialization.

    def __init__(self, eta=0.01, n_iter=50, random_state=1):
```

```python
        self.eta = eta
        self.n_iter = n_iter
        self.random_state = random_state

    def fit(self, X, y):
        rgen = np.random.RandomState(self.random_state)
        self.w_ = rgen.normal(loc=0.0, scale=0.01, size=1 + X.shape[1])
        self.errors_ = []

        for _ in range(self.n_iter):
            errors = 0
            for xi, target in zip(X, y):
                update = self.eta * (target - self.predict(xi))
                self.w_[1:] += update * xi
                self.w_[0] += update
                errors += int(update != 0.0)
            self.errors_.append(errors)
        return self

    def net_input(self, X):
        #Calculate net input
        return np.dot(X, self.w_[1:]) + self.w_[0]

    def predict(self, X):
        #Return class label after unit step
        return np.where(self.net_input(X) >= 0.0, 1, 0)

    def predict_proba(self, X):
        # Apply sigmoid function to output probabilities
        return self.sigmoid(self.net_input(X))

    def sigmoid(self, z):
        return 1.0 / (1.0 + np.exp(-z))
```

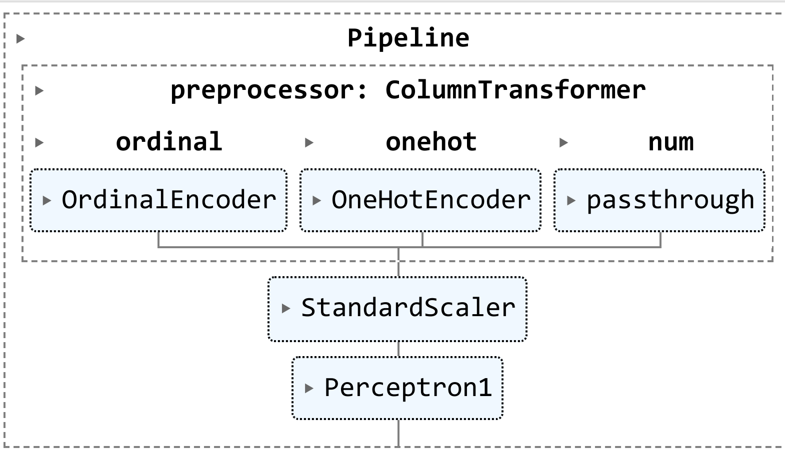
In [41]:

```python
pipeline1 = ImbPipeline(steps=[
    ('preprocessor', preprocessor),
    ('scaler', StandardScaler()),  # Apply scaling to all columns after preprocessi
    ('perceptron', Perceptron1(eta = 0.001, n_iter = 100, random_state = 10))
])
```

In [42]:

```python
pipeline1.fit(X_train, y_train)
```

Out[42]:



Model Evaluation

In [44]:
```
y_pred_ppp1 = pipeline1.predict(X_test)
```

In [45]:
```
accuracy_ppp1 = accuracy_score(y_test, y_pred_ppp1)
print(f"The accuracy of Custom Perceptron model is {accuracy_ppp1:5f}")
```

```
The accuracy of Custom Perceptron model is 0.824627
```

In [46]:
```
print(confusion_matrix(y_test, y_pred_ppp1))
```

```
[[153  25]
 [ 22  68]]
```

In [47]:
```
print(classification_report(y_test,y_pred_ppp1))
```

```
              precision    recall  f1-score   support

           0       0.87      0.86      0.87       178
           1       0.73      0.76      0.74        90

    accuracy                           0.82       268
   macro avg       0.80      0.81      0.81       268
weighted avg       0.83      0.82      0.83       268
```

In [48]:
```
f1_ppp1 = f1_score(y_test,y_pred_ppp1)
print(f"The f1 score of Custom Perceptron model is {f1_ppp1:5f}")
```

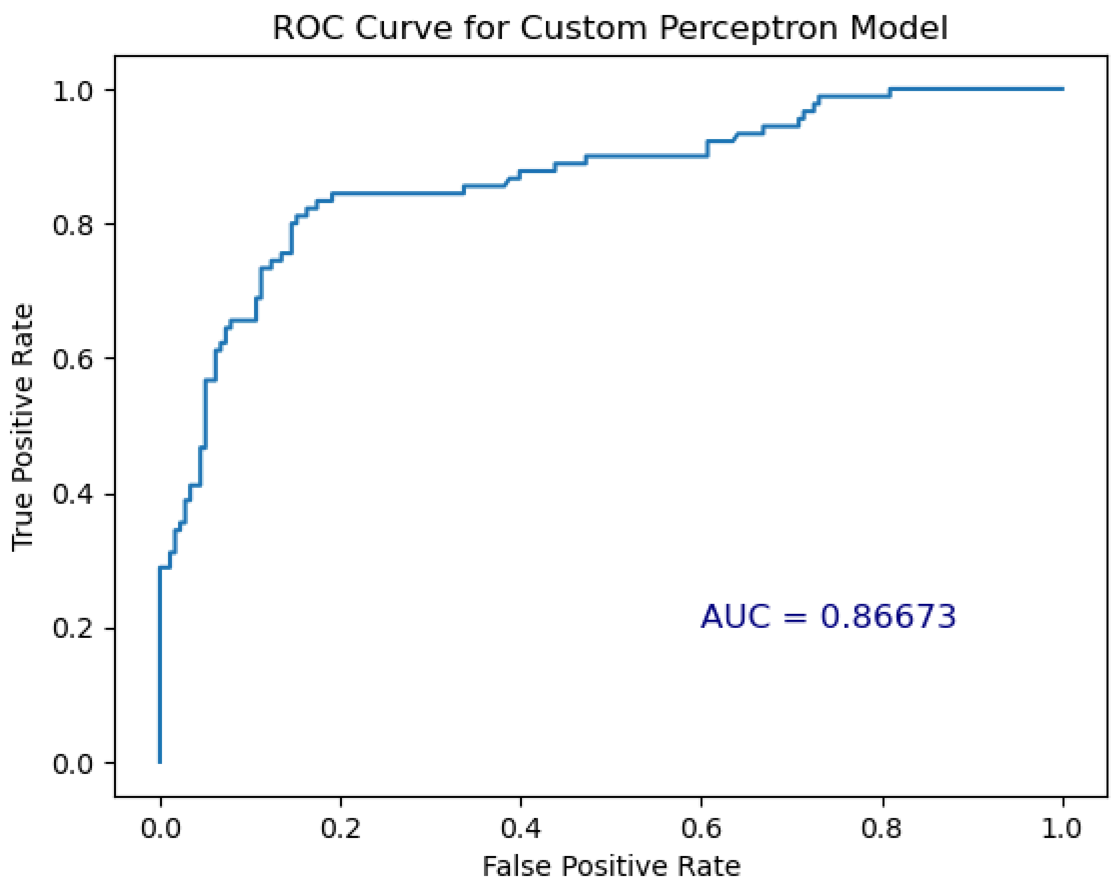
```
The f1 score of Custom Perceptron model is 0.743169
```

In [49]:
```
y_pred_proba_ppp1 = pipeline1.predict_proba(X_test)
```

In [50]:
```
roc_auc_ppp1 = roc_auc_score(y_test, y_pred_proba_ppp1)
print(f"The ROC-AUC score of Custom Perceptron model is {roc_auc_ppp1:5f}")
```

```
The ROC-AUC score of Custom Perceptron model is 0.866729
```

In [51]:
```
fpr_ppp1, tpr_ppp1, _ = roc_curve(y_test, y_pred_proba_ppp1)
plt.plot(fpr_ppp1, tpr_ppp1, label='Custom Perceptron (AUC = {:.5f})'.format(roc_au
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Custom Perceptron Model')
plt.text(0.6, 0.2, f'AUC = {roc_auc_ppp1:.5f}', fontsize=12, color='navy')
plt.show()
```



Comparision of Models

In [52]:
```
metrics = DataFrame({"KNN":[accuracy_knn, f1_knn, roc_auc_knn],
                     "Perceptron":[accuracy_ppp, f1_ppp, roc_auc_ppp],
                     "Custom Perceptron":[accuracy_ppp1, f1_ppp1, roc_auc_ppp1],},
                    index = ["Accuracy", "F1 Score", "ROC AUC Score"])
metrics = metrics.rename_axis('Metrics')
display(metrics)
```

	KNN	Perceptron	Custom Perceptron
Metrics			
Accuracy	0.850746	0.805970	0.824627
F1 Score	0.756098	0.723404	0.743169
ROC AUC Score	0.869101	0.813670	0.866729

In [53]:
```
plt.plot(fpr_knn, tpr_knn, label='KNN (AUC = {:.5f})'.format(roc_auc_knn), color =
plt.plot(fpr_ppp, tpr_ppp, label='Perceptron (AUC = {:.5f})'.format(roc_auc_ppp),co
plt.plot(fpr_ppp1, tpr_ppp1, label='Custom Perceptron (AUC = {:.5f})'.format(roc_au
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Comparision of ROC Curve for KNN, Perceptron and Custom Perceptron')
```

```
plt.legend(loc='lower right')
plt.show()
```