In [ ]:
```
!jupyter nbconvert --to html /content/KNN.ipynb
```

In [4]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics._plot.confusion_matrix import ConfusionMatrixDisplay
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
```

In [5]:
```
path = './kaggle/input/Titanic'

train_data = pd.read_csv(f'{path}/train.csv').set_index('PassengerId', drop=True)
test_data = pd.read_csv(f'{path}/test.csv').set_index('PassengerId', drop=True)
```

In [6]:
```
train_data.head()
```

Out[6]:

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

In [7]:
```
test_data.head()
```

Out[7]:

	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
PassengerId										
892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

In [8]:

```
train_data.drop('Ticket', axis=1, inplace=True)
test_data.drop('Ticket', axis=1, inplace=True)
```

In [9]:

```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 891 entries, 1 to 891
Data columns (total 10 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Survived  891 non-null    int64  
 1   Pclass    891 non-null    int64  
 2   Name      891 non-null    object 
 3   Sex       891 non-null    object 
 4   Age       714 non-null    float64
 5   SibSp     891 non-null    int64  
 6   Parch     891 non-null    int64  
 7   Fare      891 non-null    float64
 8   Cabin     204 non-null    object 
 9   Embarked  889 non-null    object 
dtypes: float64(2), int64(4), object(4)
memory usage: 76.6+ KB
```

In [10]:

```
train_data.describe()
```

Out[10]:

	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 [11]:

```
pd.pivot_table(train_data, index='Survived')
```

Out[11]:

	Age	Fare	Parch	Pclass	SibSp
Survived					
0	30.626179	22.117887	0.329690	2.531876	0.553734
1	28.343690	48.395408	0.464912	1.950292	0.473684

In [12]:

```
#Sex valuse to numeric
train_data['Sex'] = train_data['Sex'].apply(lambda x: 1 if x == 'male' else 0 )
test_data['Sex'] = test_data['Sex'].apply(lambda x: 1 if x == 'male' else 0 )
```

In [13]:

```
#Age values to categorical
def replace_age(age):
    if age <= 16:
        return 0
    elif age <= 32:
        return 1
    elif age <= 48:
        return 2
    elif age <= 64:
        return 3
    else:
        return 4

train_data['Age'] = train_data['Age'].apply(lambda x: replace_age(x))
test_data['Age'] = test_data['Age'].apply(lambda x: replace_age(x))
```

In [14]:

```
#Fare values to categorical
def replace_fare(fare):
    if fare <= 7.91:
        return 0
    elif fare <= 14.454:
        return 1
    elif fare <= 31:
        return 2
    else:
        return 3

train_data['Fare'] = train_data['Fare'].apply(lambda x: replace_fare(x))
test_data['Fare'] = test_data['Fare'].apply(lambda x: replace_fare(x))
```

In [15]:
```
#Embarked values to numeric
train_data['Embarked'] = train_data['Embarked'].apply(lambda x: 0 if x == 'S' else
test_data['Embarked'] = test_data['Embarked'].apply(lambda x: 0 if x == 'S' else (
```

In [16]:
```
#Cabin values to categorical
train_data['Cabin'] = train_data['Cabin'].apply(lambda x: 0 if pd.isna(x) else 1)
test_data['Cabin'] = test_data['Cabin'].apply(lambda x: 0 if pd.isna(x) else 1)
```

In [17]:
```
#SibSp to FamilySize
train_data['SibSp'] = train_data['SibSp'].apply(lambda x: x + 1)
test_data['SibSp'] = test_data['SibSp'].apply(lambda x: x + 1)

train_data = train_data.rename(columns={'SibSp': 'FamilySize'})
test_data = test_data.rename(columns={'SibSp': 'FamilySize'})
```

In [18]:
```
#Adding isAlone feature
train_data['isAlone'] = train_data['FamilySize'].apply(lambda x: 1 if x == 1 else (
test_data['isAlone'] = test_data['FamilySize'].apply(lambda x: 1 if x == 1 else 0)
```

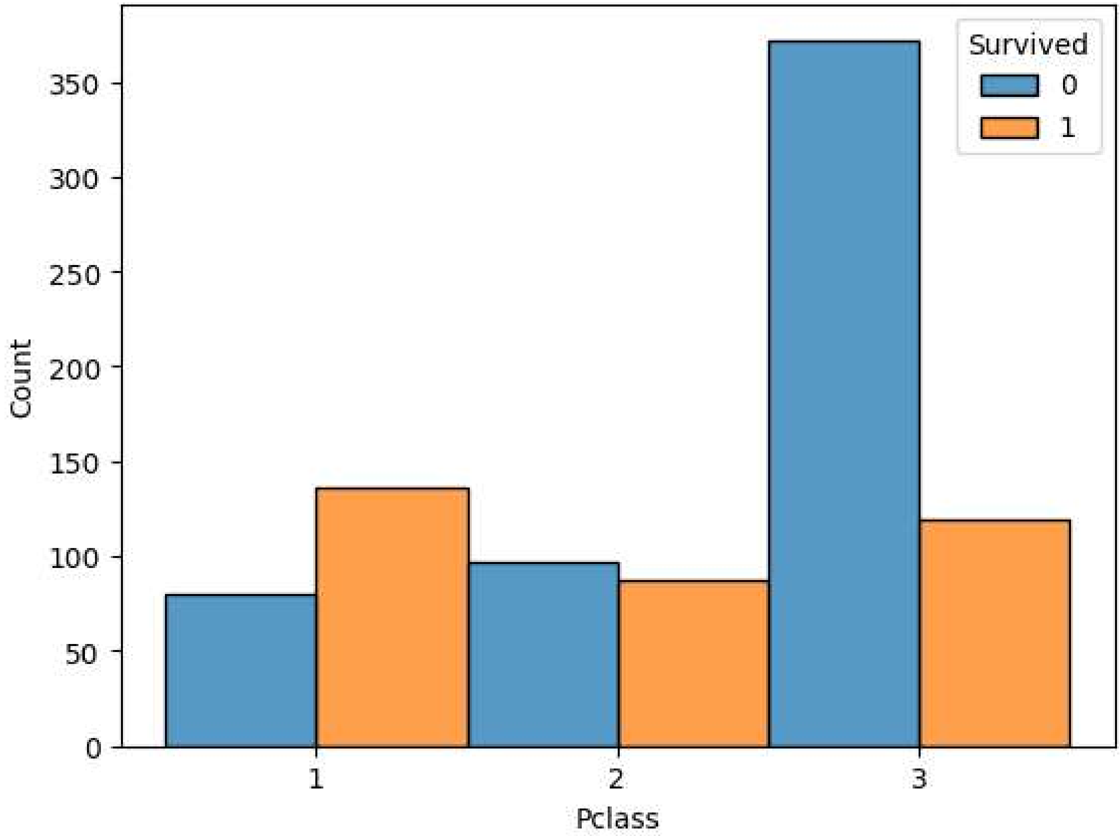
In [19]:
```
#Adding Title feature
def replace_name(name):
    if 'Mr.' in name:
        return 1
    elif 'Master' in name:
        return 2
    elif 'Mrs.' in name:
        return 3
    elif 'Miss' in name:
        return 4
    else:
        return 5

train_data['Title'] = train_data['Name'].apply(lambda x: replace_name(x))
test_data['Title'] = test_data['Name'].apply(lambda x: replace_name(x))

train_data.drop('Name', axis=1, inplace=True)
test_data.drop('Name', axis=1, inplace=True)
```

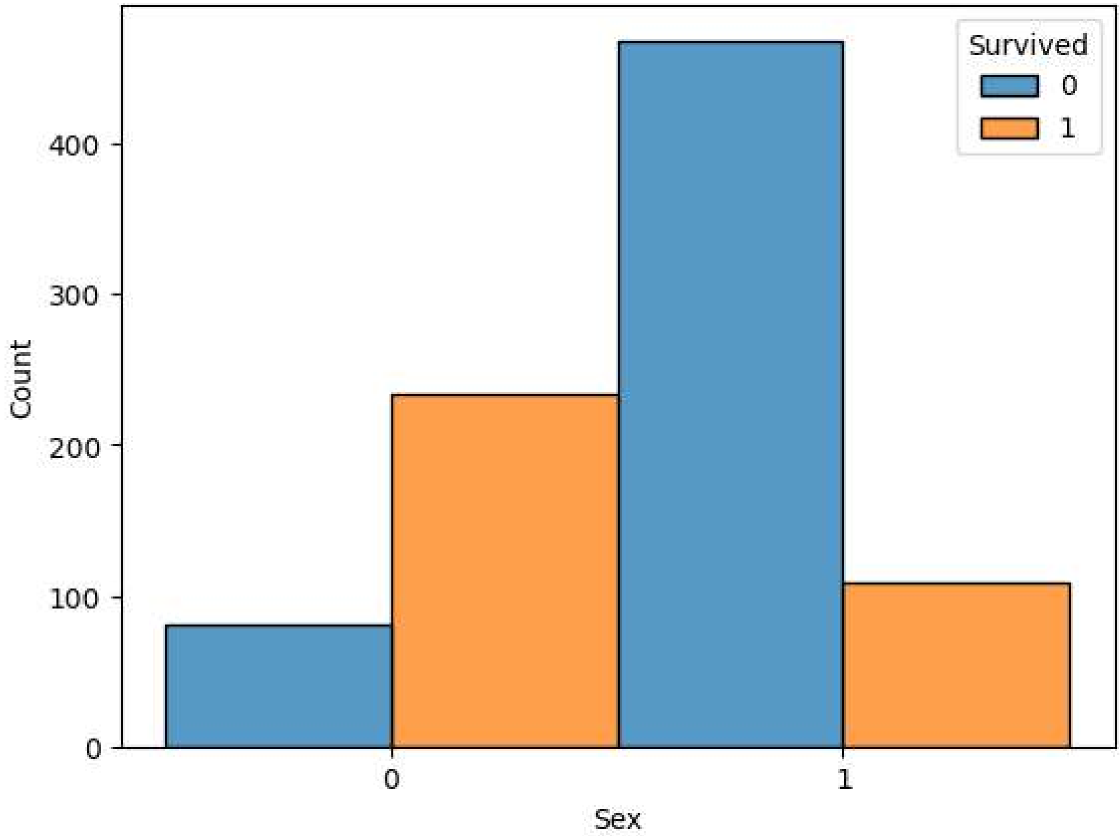
In [20]:
```
train_data.head()
```

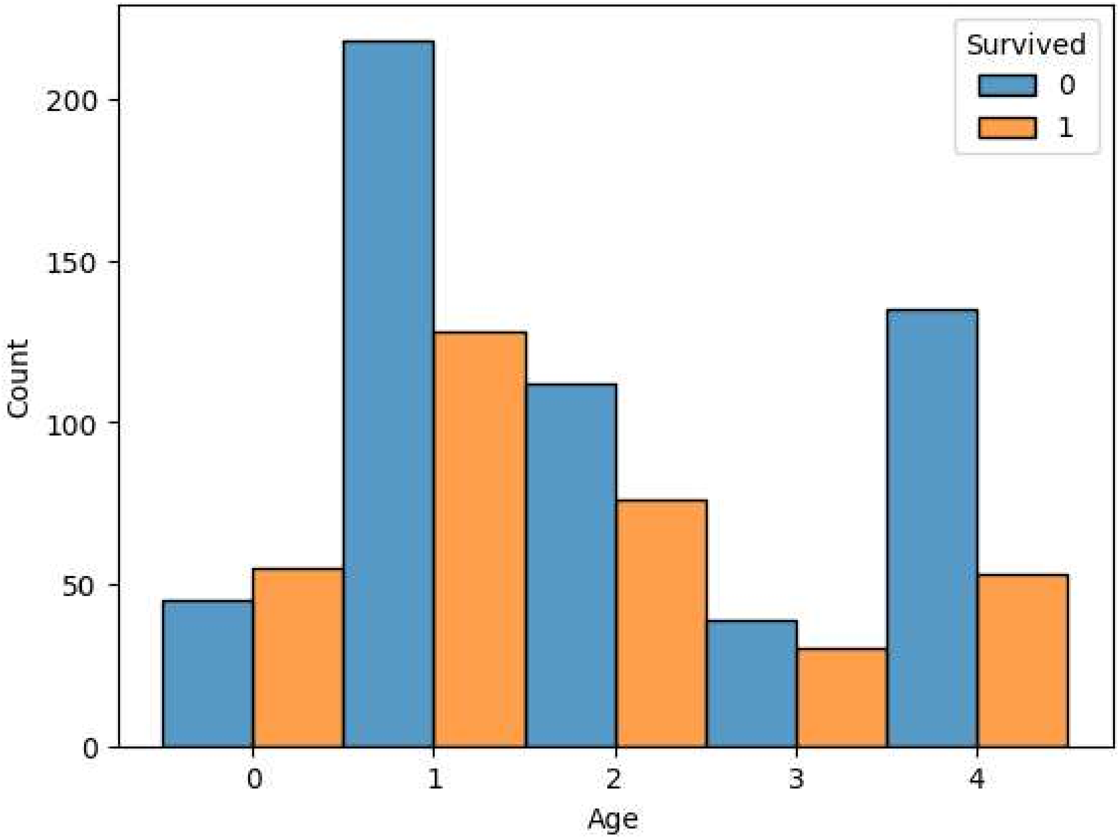
Out[20]:

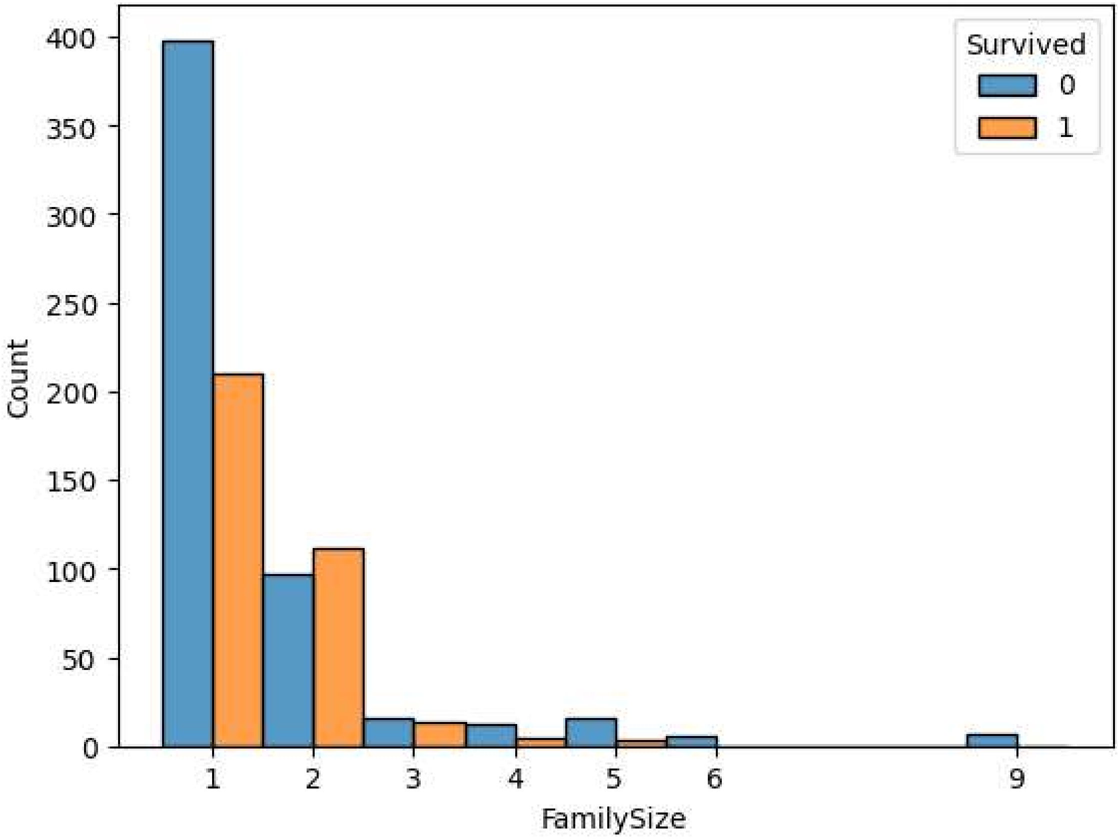
	Survived	Pclass	Sex	Age	FamilySize	Parch	Fare	Cabin	Embarked	isAlone	Titl
PassengerId											
1	0	3	1	1	2	0	0	0	0	0	
2	1	1	0	2	2	0	3	1	1	0	
3	1	3	0	1	1	0	1	0	0	1	
4	1	1	0	2	2	0	3	1	0	0	
5	0	3	1	2	1	0	1	0	0	1	

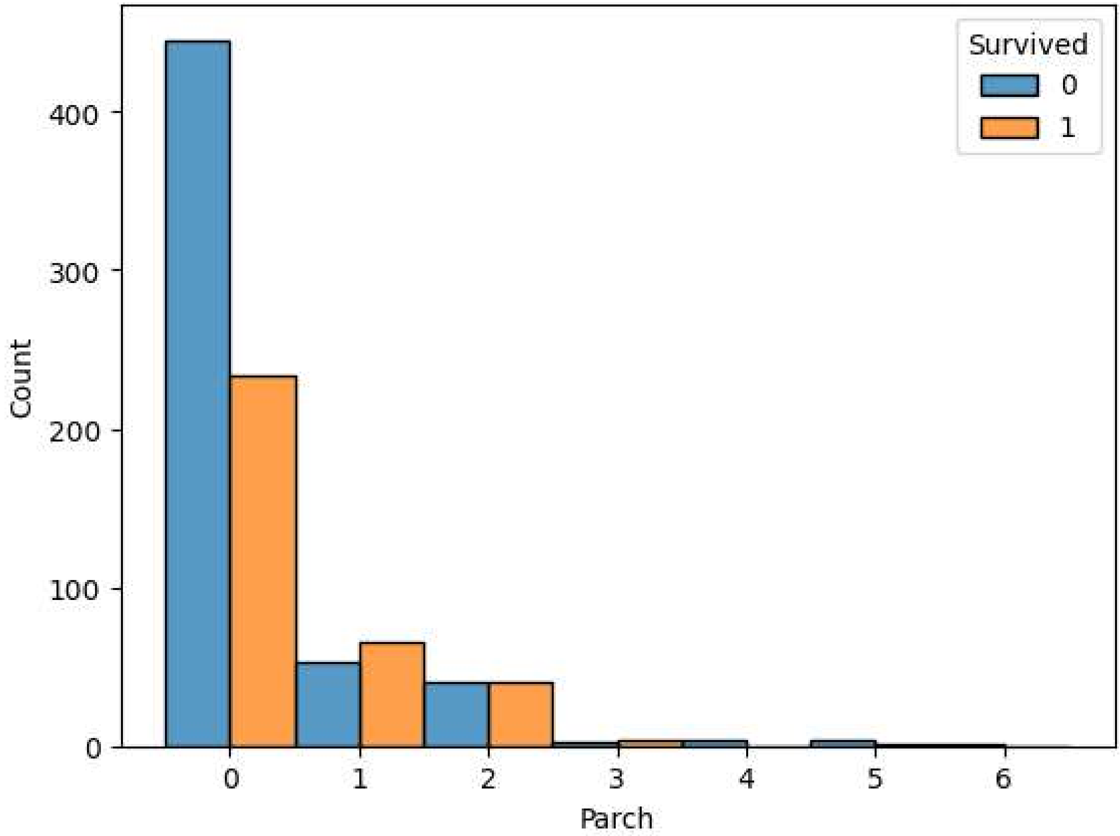
In [21]:
```
for col in train_data.columns[1:]:
    sns.histplot(data=train_data, x=col, hue='Survived', discrete=True, multiple="
    plt.xticks(train_data[col].unique())
    plt.show()
```

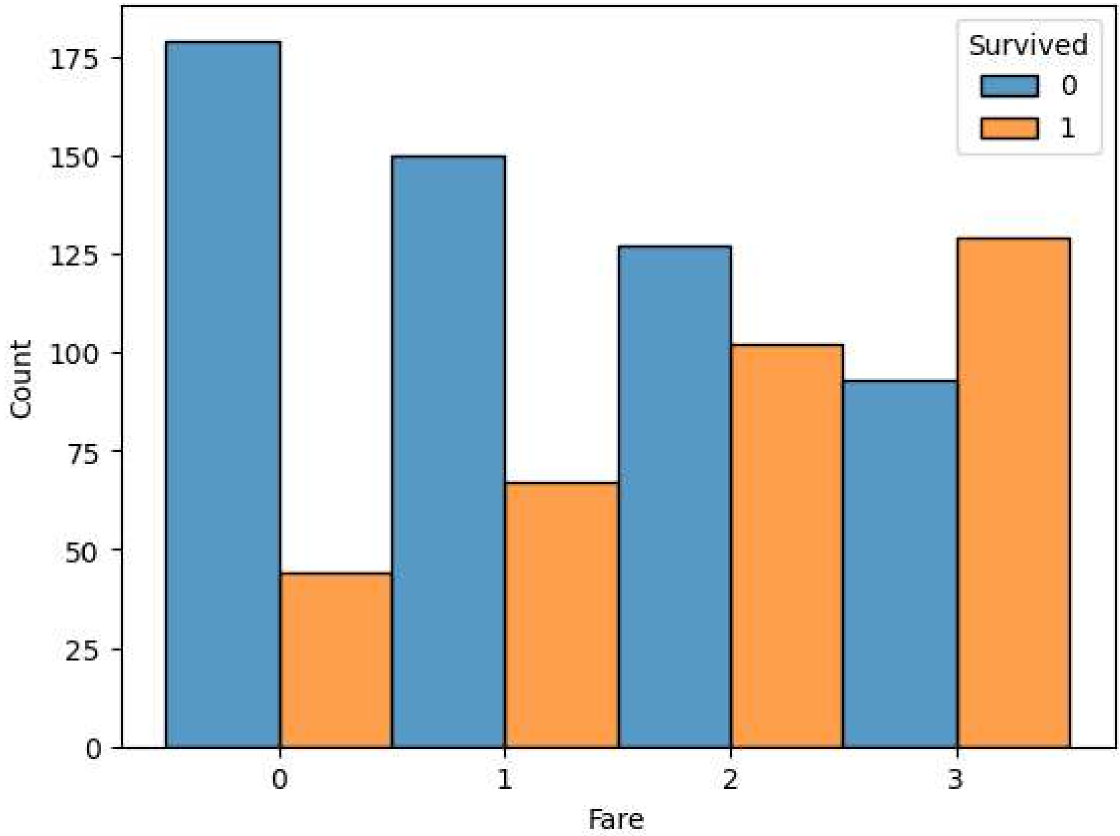

Survived
0
1
Count
0
50
100
150
200
250
300
350
1
2
3
Pclass

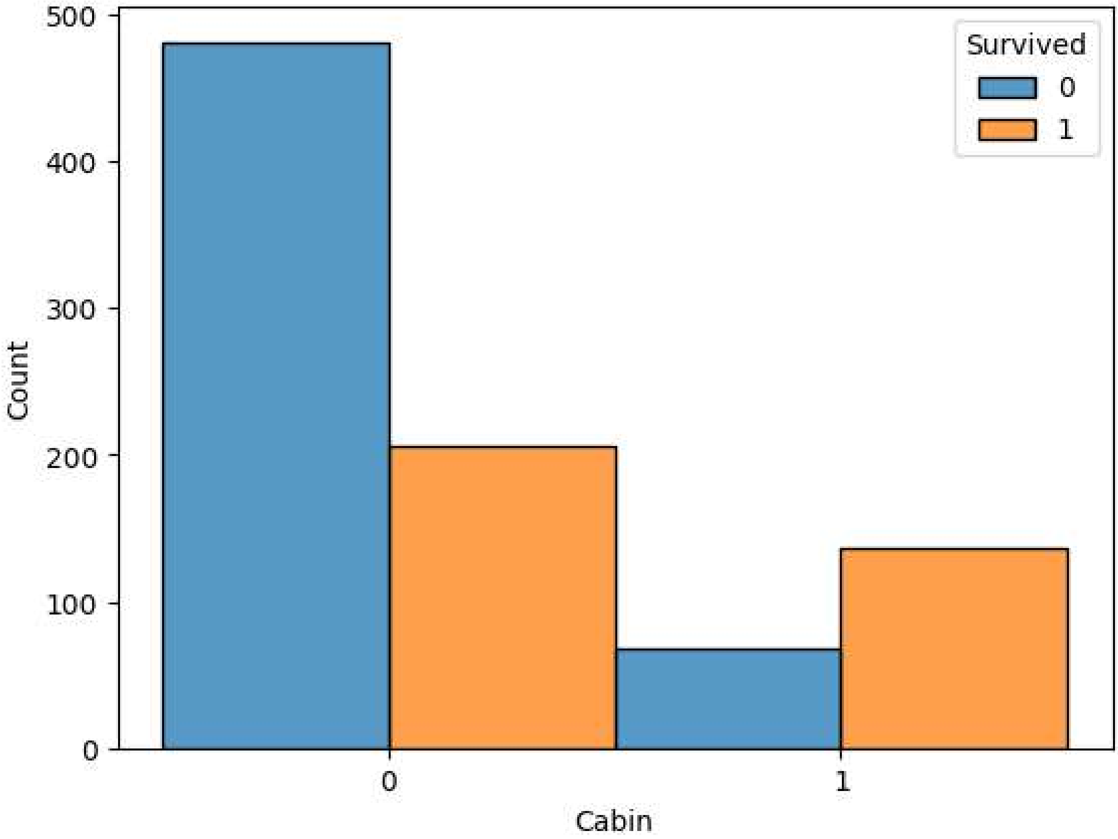

Survived
0
1
Count
0
100
200
300
400
0
1
Sex

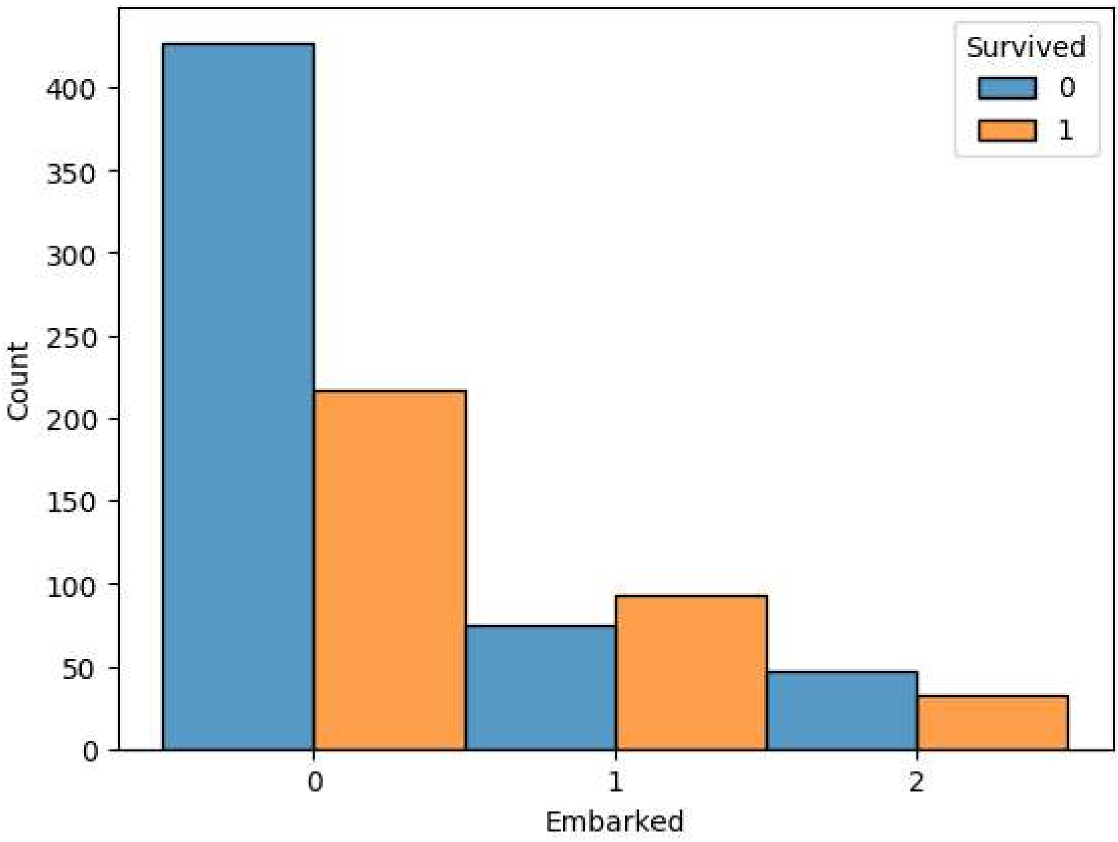

Survived
0
1
Count
Age
0
50
100
150
200
0
1
2
3
4

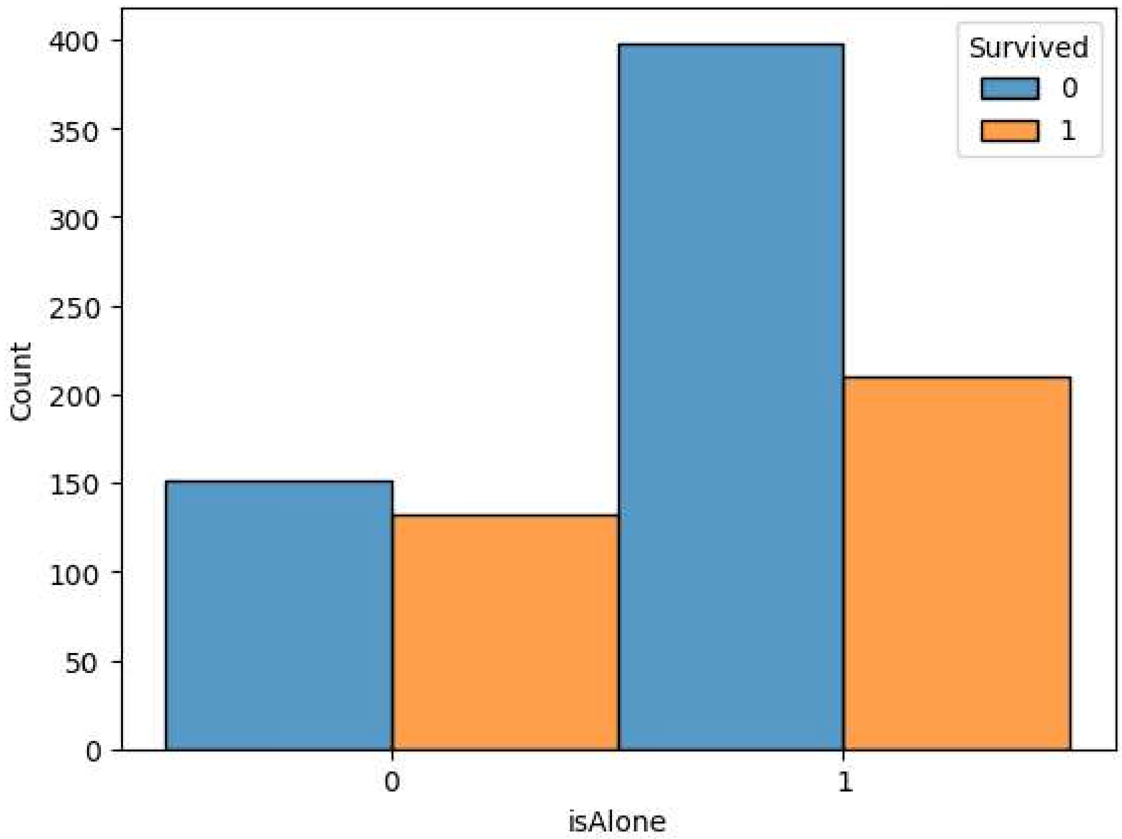

Survived
0
1
Count
FamilySize
0
50
100
150
200
250
300
350
400
1
2
3
4
5
6
9

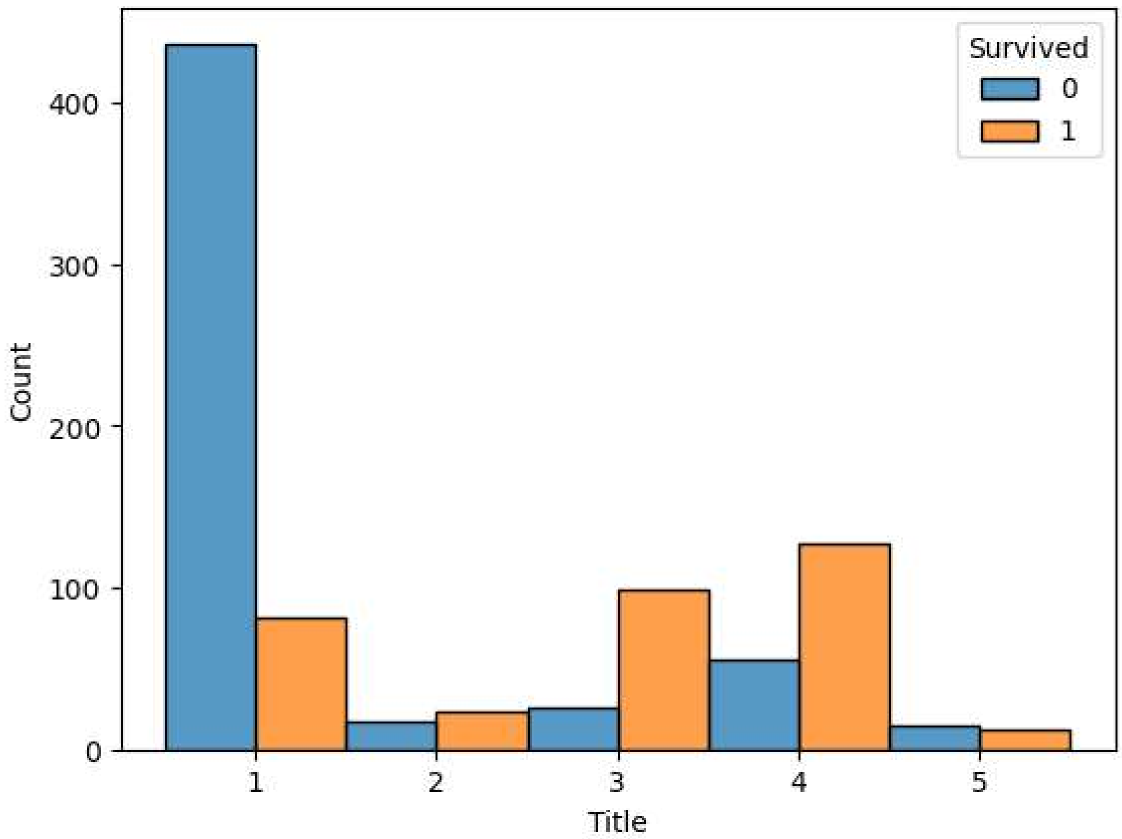

Survived
0
1
400
300
200
100
0
Count
0
1
2
3
4
5
6
Parch


Survived
0
1
175
150
125
100
75
50
25
0
Count
0
1
2
3
Fare


Survived
0
1
Count
500
400
300
200
100
0
0
1
Cabin

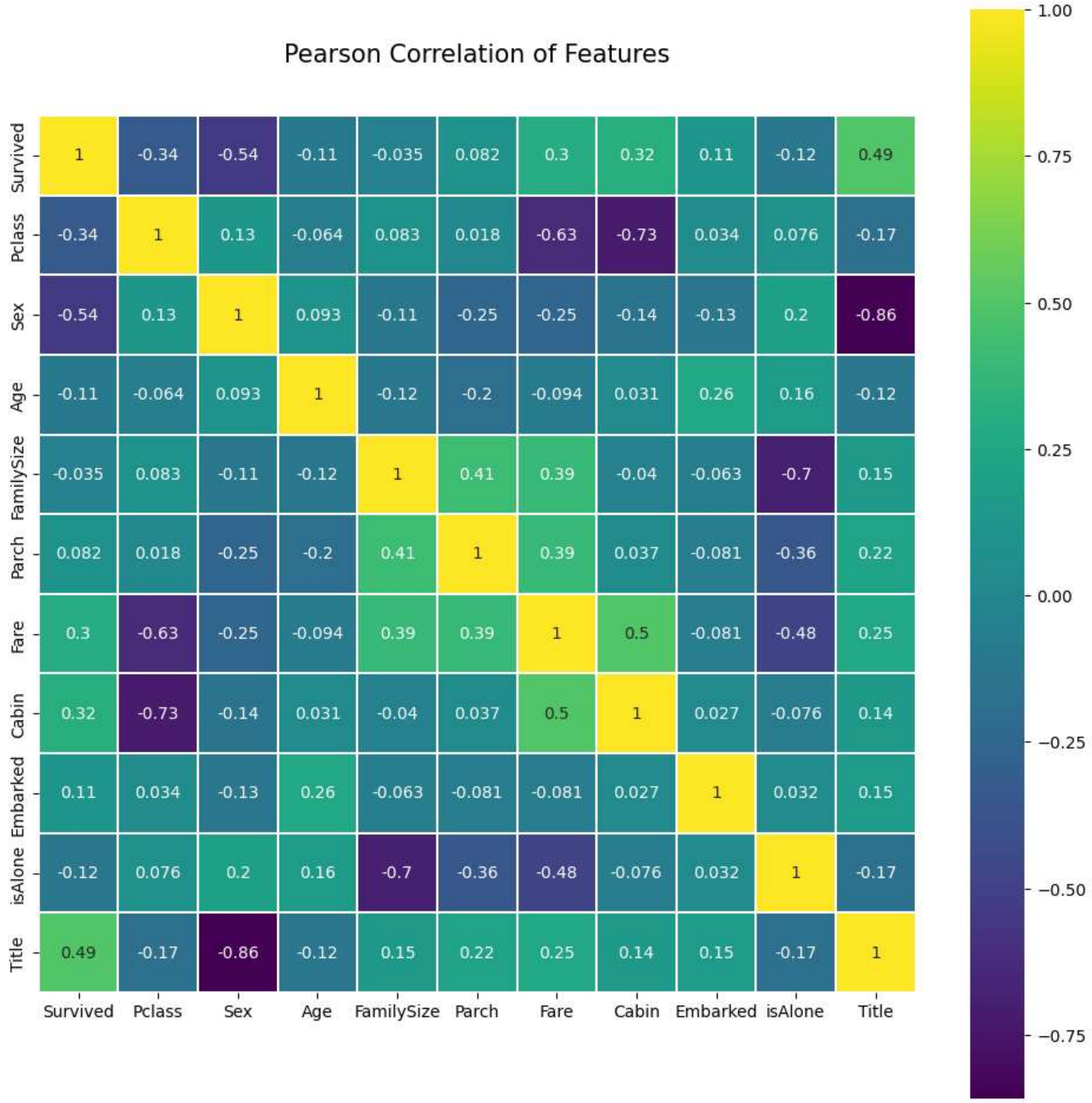

Survived
0
1
Count
400
350
300
250
200
150
100
50
0
0
1
2
Embarked





In [22]:
```
colormap = plt.cm.viridis
plt.figure(figsize=(12,12))
plt.title('Pearson Correlation of Features', y=1.05, size=15)
sns.heatmap(train_data.corr(),linewidths=0.1,vmax=1.0,
square=True, cmap=colormap, linecolor='white', annot=True)
```

Out[22]:
```
<Axes: title={'center': 'Pearson Correlation of Features'}>
```



In [23]:
```
train_data.loc[:, ['Title', 'Survived']].groupby('Title').mean()
```

Out[23]:

Title	Survived
1	0.156673
2	0.575000
3	0.792000
4	0.697802
5	0.444444

In [24]:
```
train_data.loc[:, ['Sex', 'Survived']].groupby('Sex').count()
```

Out[24]:

Sex	Survived
0	314
1	577

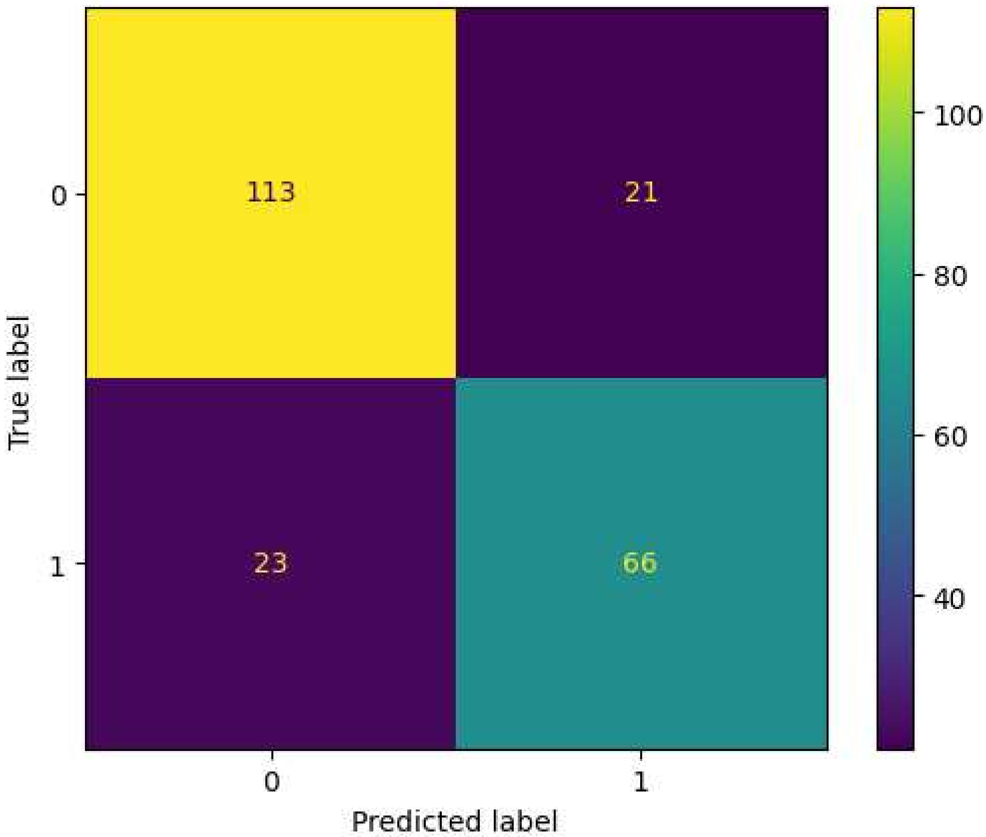
In [25]:

```
labels = train_data.loc[:, 'Survived']
train_data.drop('Survived', axis=1, inplace=True)

X_train, X_test, y_train, y_test = train_test_split(train_data, labels, test_size=(
```

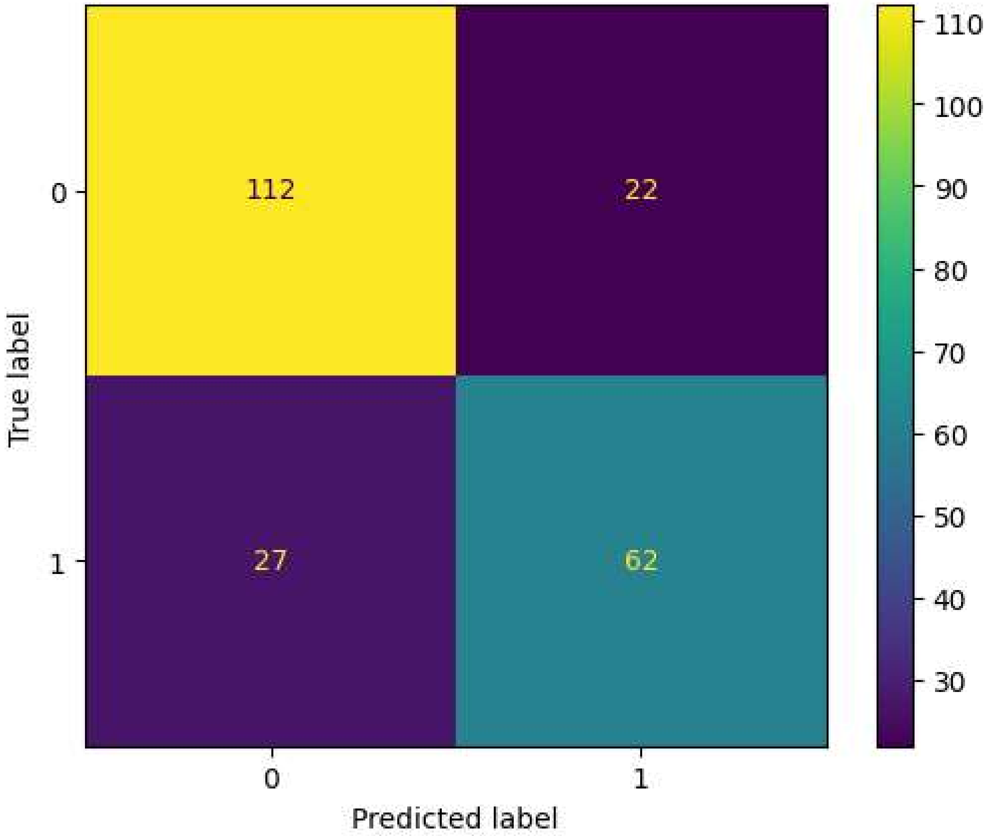
In [26]:

```
classifiers = [LogisticRegression(),
               KNeighborsClassifier(),
               DecisionTreeClassifier(),
               RandomForestClassifier(),
               SVC()]

for classifier in classifiers:
    clf = classifier
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)

    print(f"Accuracy for {clf.__class__.__name__}: {accuracy}")

    cm = confusion_matrix(y_test, y_pred, labels=clf.classes_)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_
    disp.plot()
    plt.show()
```

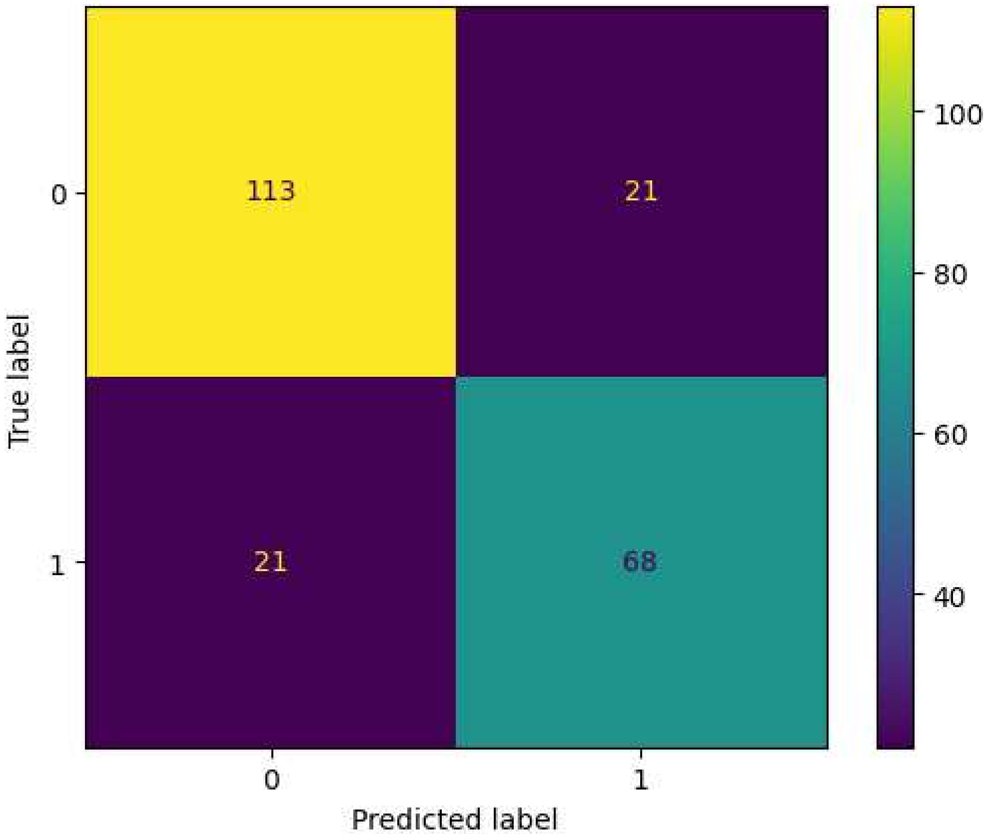
```
Accuracy for LogisticRegression: 0.8026905829596412
```



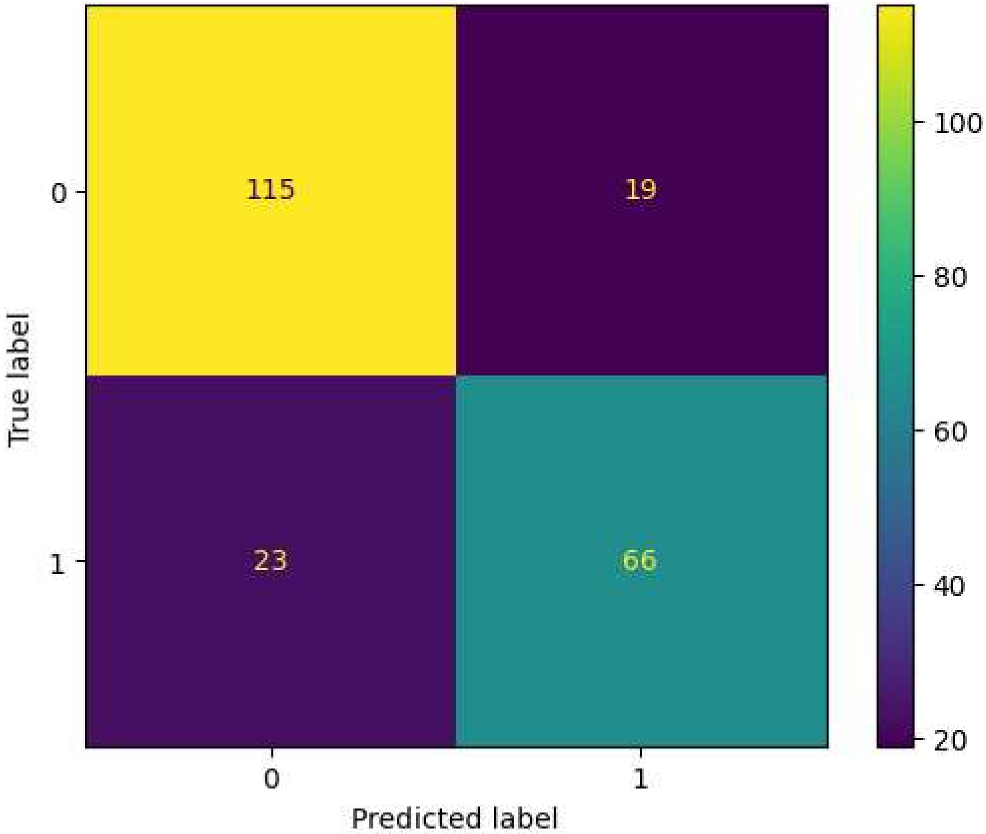

```
Accuracy for KNeighborsClassifier: 0.7802690582959642
```



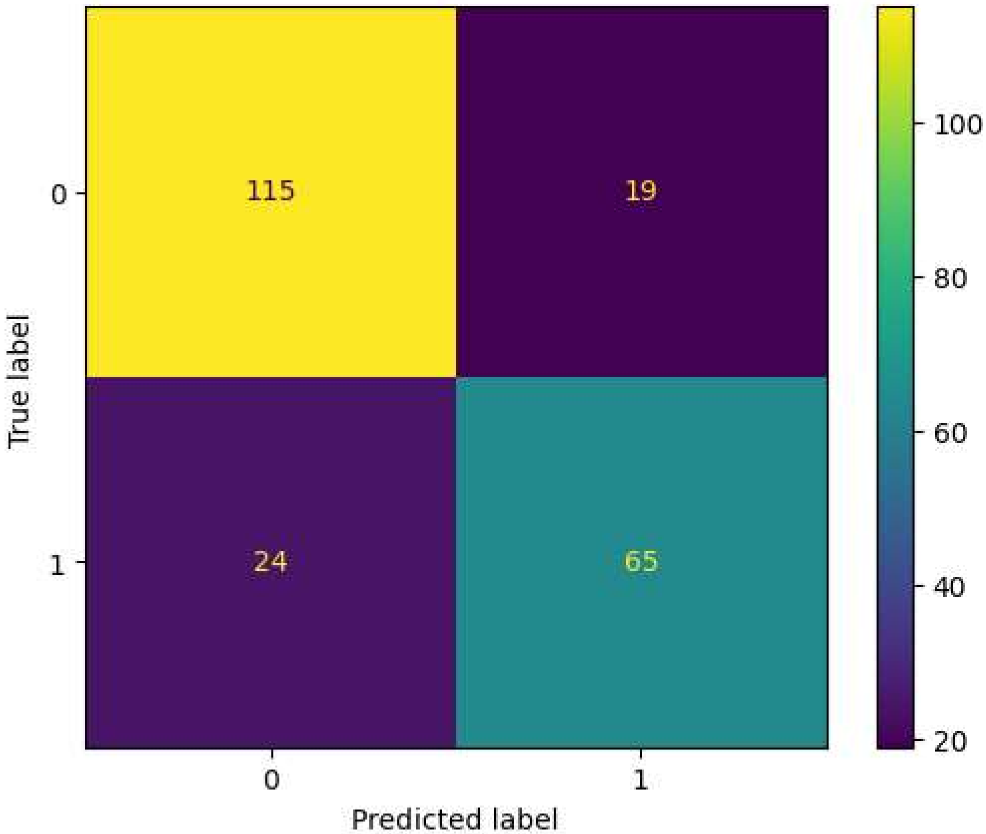

```
Accuracy for DecisionTreeClassifier: 0.8116591928251121
```




```
Accuracy for RandomForestClassifier: 0.8116591928251121
```




```
Accuracy for SVC: 0.8071748878923767
```



In [27]:
```
classifiers = [LogisticRegression(),
               KNeighborsClassifier(),
               DecisionTreeClassifier(),
               RandomForestClassifier(),
               SVC()]

param_grid_lr = {'max_iter': [100, 250, 500, 1000, 2000],
               'penalty': ['l1', 'l2'],
```

```
              'solver': ['liblinear']}

param_grid_knn = {'n_neighbors': [3,5,7,9],
                  'weights': ['uniform', 'distance'],
                  'algorithm': ['auto', 'ball_tree', 'kd_tree'],
                  'p': [1, 2],
                  'metric': ['euclidean', 'manhattan']}

param_grid_dc = {'criterion': ['gini', 'entropy'],
                 'max_depth': range(1, 11)}

param_grid_rf = {'criterion': ['gini', 'entropy'],
                 'max_depth': range(2, 6)}

param_grid_svc = [{'kernel': ['rbf'],
                   'gamma': [.1, .5, 1, 2, 5, 10],
                   'C': [.1, 1, 10, 100, 1000]},
                  {'kernel': ['linear'],
                   'C': [.1, 1, 10, 100, 1000]},
                  {'kernel': ['poly'],
                   'degree': [2, 3, 4, 5],
                   'C':[.1, 1, 10, 100, 1000]}]

grid_params = [param_grid_lr, param_grid_knn, param_grid_dc, param_grid_rf, param_g

for idx, classifier in enumerate(classifiers):
    clf = classifier

    clf_gs = GridSearchCV(clf, param_grid=grid_params[idx], cv=10, verbose=True, n_
    clf_gs.fit(X_train, y_train)

    print(f'Classifier: {classifier.__class__.__name__}')
    print(f'Best score: {clf_gs.best_score_}')
    print(f'Best parameters: {clf_gs.best_params_}')
    print('-'*60)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
Classifier: LogisticRegression
Best score: 0.7905246494798732
Best parameters: {'max_iter': 100, 'penalty': 'l2', 'solver': 'liblinear'}
------------------------------------------------------------
Fitting 10 folds for each of 96 candidates, totalling 960 fits
Classifier: KNeighborsClassifier
Best score: 0.8235187697874264
Best parameters: {'algorithm': 'ball_tree', 'metric': 'manhattan', 'n_neighbors':
7, 'p': 1, 'weights': 'distance'}
------------------------------------------------------------
Fitting 10 folds for each of 20 candidates, totalling 200 fits
Classifier: DecisionTreeClassifier
Best score: 0.8099276345545002
Best parameters: {'criterion': 'entropy', 'max_depth': 6}
------------------------------------------------------------
Fitting 10 folds for each of 8 candidates, totalling 80 fits
Classifier: RandomForestClassifier
Best score: 0.8264812302125735
Best parameters: {'criterion': 'gini', 'max_depth': 5}
------------------------------------------------------------
Fitting 10 folds for each of 55 candidates, totalling 550 fits
Classifier: SVC
Best score: 0.8248982360922659
Best parameters: {'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}
------------------------------------------------------------
```

In [28]:
```
#Best clf
clf = RandomForestClassifier(criterion='gini', max_depth=5)
clf.fit(train_data, labels)
```

Out[28]:
```
▾ RandomForestClassifier
RandomForestClassifier(max_depth=5)
```

In [29]:
```
submission_pred = clf.predict(test_data)
```

In [30]:
```
df = pd.DataFrame({'PassengerId': test_data.index.values,
                   'Survived': submission_pred})
```

In [31]:
```
df.to_csv('submission.csv', index=False)
```