Predict the survival of passengers based on features.

- Features: age, fare, sex, and others.
- Classes: 2 (Survived or Not Survived).

Logistic Regression

In [1]:
```
#Import necessary libraries

import pandas as pd
import numpy as np
import mglearn
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

In [2]:
```
#Load the dataset

url = "https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv"
titanic = pd.read_csv(url)
```

In [3]:
```
#Preprocess the data

titanic = titanic.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1)
titanic['Age'].fillna(titanic['Age'].mean(), inplace=True)
titanic['Embarked'].fillna(titanic['Embarked'].mode()[0], inplace=True)
```

In [4]:
```
titanic.head()
```

Out[4]:

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

In [5]:
```
#Convert categorical variables into dummy/indicator variables

titanic = pd.get_dummies(titanic, columns=['Sex', 'Embarked'], drop_first=True)
```

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

Out[6]:

	Survived	Pclass	Age	SibSp	Parch	Fare	Sex_male	Embarked_Q	Embarked_S
0	0	3	22.0	1	0	7.2500	1	0	1
1	1	1	38.0	1	0	71.2833	0	0	0
2	1	3	26.0	0	0	7.9250	0	0	1
3	1	1	35.0	1	0	53.1000	0	0	1
4	0	3	35.0	0	0	8.0500	1	0	1

In [7]:

```
#Define features (X) and target (y)

X = titanic.drop('Survived', axis=1)
y = titanic['Survived']
```

In [8]:

```
#Split data into training and test sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4
```

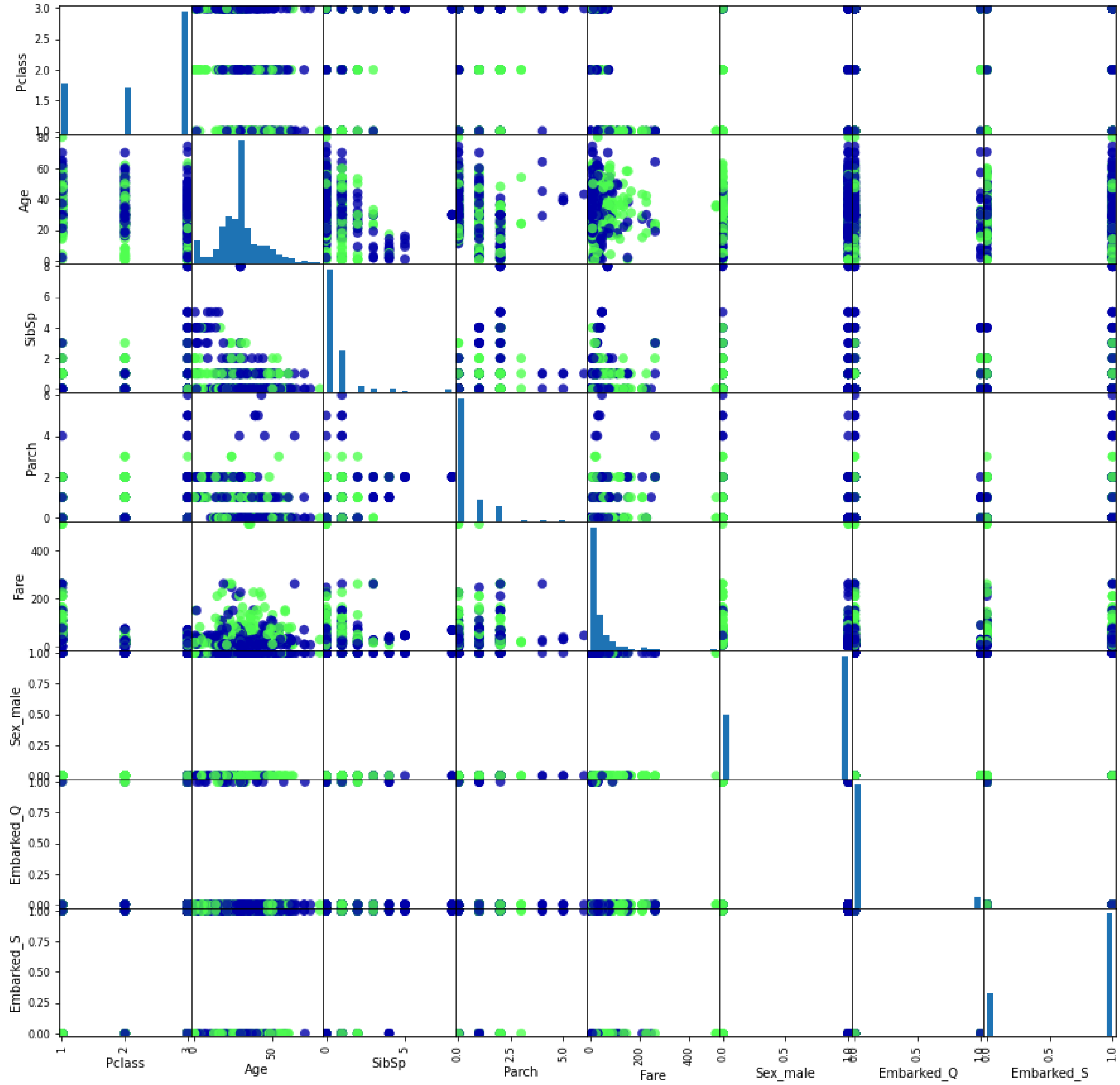
In [9]:

```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(712, 8)
(712,)
(179, 8)
(179,)
```

In [10]:

```
grr = scatter_matrix(X_train,
                     c=y_train,
                     figsize=(15,15),
                     marker='o',
                     s=60,
                     hist_kwds={'bins':20},
                     alpha=0.8,
                     cmap=mglearn.cm3)
```



In [11]:
```python
#Train the Logistic Regression model

logreg = LogisticRegression(solver = 'liblinear')
logreg.fit(X_train, y_train)
```

Out[11]:
```
LogisticRegression(solver='liblinear')
```

In [12]:
```python
#Make predictions on the test set

y_pred = logreg.predict(X_test)
```

In [13]:
```python
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
```

In [14]:
```
print(f"Accuracy: {accuracy}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(class_report)
```

```
Accuracy: 0.7821229050279329
Confusion Matrix:
[[89 16]
 [23 51]]
Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.85      0.82       105
           1       0.76      0.69      0.72        74

    accuracy                           0.78       179
   macro avg       0.78      0.77      0.77       179
weighted avg       0.78      0.78      0.78       179
```

Logistic Regression (K-Fold)

In [15]:
```
from sklearn.model_selection import cross_val_score, KFold
```

In [16]:
```
#Perform K-Fold Cross-Validation

kf = KFold(n_splits=5, shuffle=True, random_state=42)  # 5 folds
cv_scores = cross_val_score(logreg, X, y, cv=kf, scoring='accuracy')
```

In [17]:
```
#Print the results of cross-validation

formatted_scores = [f"{score:.3f}" for score in cv_scores]
print(f"Cross-Validation Accuracy Scores: {formatted_scores}")

print(f"Mean Cross-Validation Accuracy: {cv_scores.mean():.3f}")
print(f"Standard Deviation of CV Accuracy: {cv_scores.std():.3f}")
```

```
Cross-Validation Accuracy Scores: ['0.782', '0.787', '0.848', '0.764', '0.815']
Mean Cross-Validation Accuracy: 0.799
Standard Deviation of CV Accuracy: 0.029
```

In [18]:
```
logreg.fit(X, y)
y_pred = logreg.predict(X)
```

In [19]:
```
accuracy = accuracy_score(y, y_pred)
conf_matrix = confusion_matrix(y, y_pred)
class_report = classification_report(y, y_pred)
```

In [20]:
```
print(f"\nAccuracy on the whole dataset: {accuracy}")
print("Confusion Matrix:")
print(conf_matrix)
```

```
print("Classification Report:")
print(class_report)
```

```
Accuracy on the whole dataset: 0.8024691358024691
Confusion Matrix:
[[479  70]
 [106 236]]
Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.87      0.84       549
           1       0.77      0.69      0.73       342

    accuracy                           0.80       891
   macro avg       0.80      0.78      0.79       891
weighted avg       0.80      0.80      0.80       891
```

Logistic Regression (K-Fold)

In [21]:

```
from sklearn.model_selection import GridSearchCV
```

In [22]:

```
#Define the parameter grid

param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
    'penalty': ['l1', 'l2', 'elasticnet', 'none'],
    'solver': ['liblinear', 'saga']
}
```

In [23]:

```
#Use GridSearchCV without K-Fold cross-validation

grid_search = GridSearchCV(logreg, param_grid, scoring='accuracy', verbose=1, n_jobs=-1
grid_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:918: UserW
arning: One or more of the test scores are non-finite: [0.66710332 0.66570472 0.67971043
0.68530484        nan        nan
        nan 0.68109918 0.66704422 0.66704422 0.7106274  0.68109918
        nan        nan        nan 0.68109918 0.7850586  0.68250763
 0.7892938  0.68109918        nan        nan        nan 0.68109918
 0.79489806 0.68109918 0.79629666 0.68109918        nan        nan
        nan 0.68109918 0.78925441 0.68109918 0.79207131 0.68109918
        nan        nan        nan 0.68109918 0.78925441 0.68109918
 0.78925441 0.68109918        nan        nan        nan 0.68109918]
  warnings.warn(
```

Out[23]:

```
GridSearchCV(estimator=LogisticRegression(solver='liblinear'), n_jobs=-1,
             param_grid={'C': [0.001, 0.01, 0.1, 1, 10, 100],
                         'penalty': ['l1', 'l2', 'elasticnet', 'none'],
                         'solver': ['liblinear', 'saga']},
             scoring='accuracy', verbose=1)
```

In [24]:

```
#Get the best parameters and best score from the grid search

best_params = grid_search.best_params_
best_score = grid_search.best_score_
```

```
print(f"Best Parameters from Grid Search: {best_params}")
print(f"Best Training Accuracy: {best_score:.3f}")
```

```
Best Parameters from Grid Search: {'C': 1, 'penalty': 'l2', 'solver': 'liblinear'}
Best Training Accuracy: 0.796
```

In [25]:

```
#Evaluate on the test set

y_pred = grid_search.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
```

In [26]:

```
print(f"\nAccuracy on the test set: {test_accuracy:.3f}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(class_report)
```

```
Accuracy on the test set: 0.782
Confusion Matrix:
[[89 16]
 [23 51]]
Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.85      0.82       105
           1       0.76      0.69      0.72        74

    accuracy                           0.78       179
   macro avg       0.78      0.77      0.77       179
weighted avg       0.78      0.78      0.78       179
```

Logistic Regression (CV & K-Fold)

In [27]:

```
#Define the parameter grid

param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
    'penalty': ['l1', 'l2', 'elasticnet', 'none'],
    'solver': ['liblinear', 'saga']
}
```

In [28]:

```
#Set up K-Fold cross-validation and GridSearchCV

kf = KFold(n_splits=5, shuffle=True, random_state=42)
```

In [29]:

```
#Use GridSearchCV to perform the grid search

grid_search = GridSearchCV(logreg, param_grid, cv=kf, scoring='accuracy', verbose=1, n_
grid_search.fit(X, y)
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:918: UserW
arning: One or more of the test scores are non-finite: [0.66220576 0.65883498 0.6868872
0.68351641        nan        nan
        nan 0.6868872  0.67004582 0.66555772 0.71267968 0.6868872
        nan        nan        nan 0.6868872  0.79124976 0.68576988
 0.79687402 0.6868872         nan        nan        nan 0.6868872
 0.79573787 0.6868872  0.79912121 0.6868872         nan        nan
        nan 0.6868872  0.79908982 0.6868872  0.80471408 0.6868872
        nan        nan        nan 0.6868872  0.80021342 0.6868872
 0.80021342 0.6868872         nan        nan        nan 0.6868872 ]
  warnings.warn(
```

Out[29]:
```
GridSearchCV(cv=KFold(n_splits=5, random_state=42, shuffle=True),
             estimator=LogisticRegression(solver='liblinear'), n_jobs=-1,
             param_grid={'C': [0.001, 0.01, 0.1, 1, 10, 100],
                         'penalty': ['l1', 'l2', 'elasticnet', 'none'],
                         'solver': ['liblinear', 'saga']},
             scoring='accuracy', verbose=1)
```

In [30]:
```
#Get the best parameters and best score from the grid search

best_params = grid_search.best_params_
best_score = grid_search.best_score_

print(f"Best Parameters from Grid Search: {best_params}")
print(f"Best Cross-Validation Accuracy: {best_score:.3f}")
```

```
Best Parameters from Grid Search: {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}
Best Cross-Validation Accuracy: 0.805
```

In [31]:
```
#Train the model with the best parameters

best_logreg = grid_search.best_estimator_
y_pred = best_logreg.predict(X)
```

In [32]:
```
#Evaluate the model on the entire dataset

accuracy = accuracy_score(y, y_pred)
conf_matrix = confusion_matrix(y, y_pred)
class_report = classification_report(y, y_pred)
```

In [33]:
```
print(f"\nAccuracy on the entire dataset: {accuracy:.3f}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(class_report)
```

```
Accuracy on the entire dataset: 0.799
Confusion Matrix:
[[473  76]
 [103 239]]
Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.86      0.84       549
           1       0.76      0.70      0.73       342

    accuracy                           0.80       891
   macro avg       0.79      0.78      0.78       891
```

```
weighted avg       0.80      0.80      0.80       891
```

KNN

In [34]:
```python
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
```

In [35]:
```python
knn.fit(X_train, y_train)
```

Out[35]:
```
KNeighborsClassifier(n_neighbors=3)
```

In [36]:
```python
y_pred = knn.predict(X_test)
```

In [38]:
```python
print("Test set score: {:.2f}".format(knn.score(X_test, y_test)))

accuracy = accuracy_score(y_test, y_pred)
print("Model accuracy: {:.2f}" .format(accuracy))
```

```
Test set score: 0.71
Model accuracy: 0.71
```