fdxz0mjel

January 23, 2025

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import accuracy_score, classification_report,␣
↪confusion_matrix,mean_squared_error, mean_absolute_error, r2_score
import math
```

[2]:
```
# Load Titanic dataset
data = pd.read_csv(r"C:\Users\91703\OneDrive\Desktop\TITANIC.csv")
```

[3]:
```
data.head()
```

[3]:

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

[4]:
```
data.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
```

[5]:
```
data.describe()
```

[5]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

[6]:
```
data.isnull().sum()
```

[6]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

[8]:
```
data.dropna(subset=['Age','Cabin','Embarked'], inplace=True)
```

[9]:
```
data.isnull().sum()
```

[9]:
```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Cabin          0
Embarked       0
dtype: int64
```

[10]:
```
# Select relevant columns and convert categorical variables
data = data[['Survived', 'Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare']]
data['Sex'] = data['Sex'].map({'male': 0, 'female': 1})
```

[14]:
```
# Handle missing Age values
data['Age'] = pd.to_numeric(data['Age'], errors='coerce')
data['Age'] = data['Age'].fillna(data['Age'].mean())
```

[15]:
```
data.head()
```

[15]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare
1	1	1	1	38.0	1	0	71.2833
3	1	1	1	35.0	1	0	53.1000
6	0	1	0	54.0	0	0	51.8625
10	1	3	1	4.0	1	1	16.7000

```
11          1       1    1  58.0      0      0  26.5500
```

[16]:
```
# Define target variable (y) and feature variables (X)
y = data['Survived']
X = data[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare']]
```

[17]:
```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[18]:
```
# Create and train Random Forest Classifier model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

[18]:
```
RandomForestClassifier(random_state=42)
```

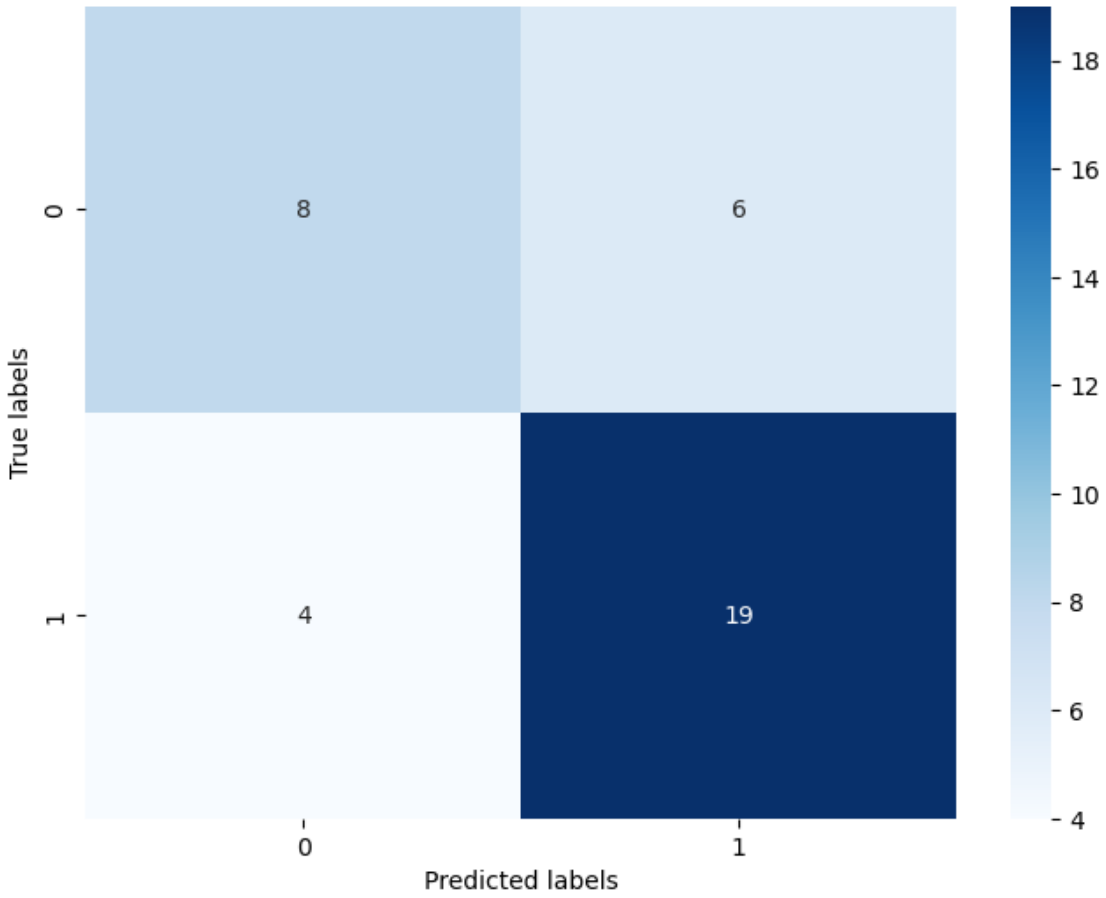
[20]:
```
#Make predictions on test data
y_pred = model.predict(X_test)
print(y_pred)
```

```
[0 1 1 0 0 0 1 1 1 1 0 1 0 1 1 0 1 0 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1]
```

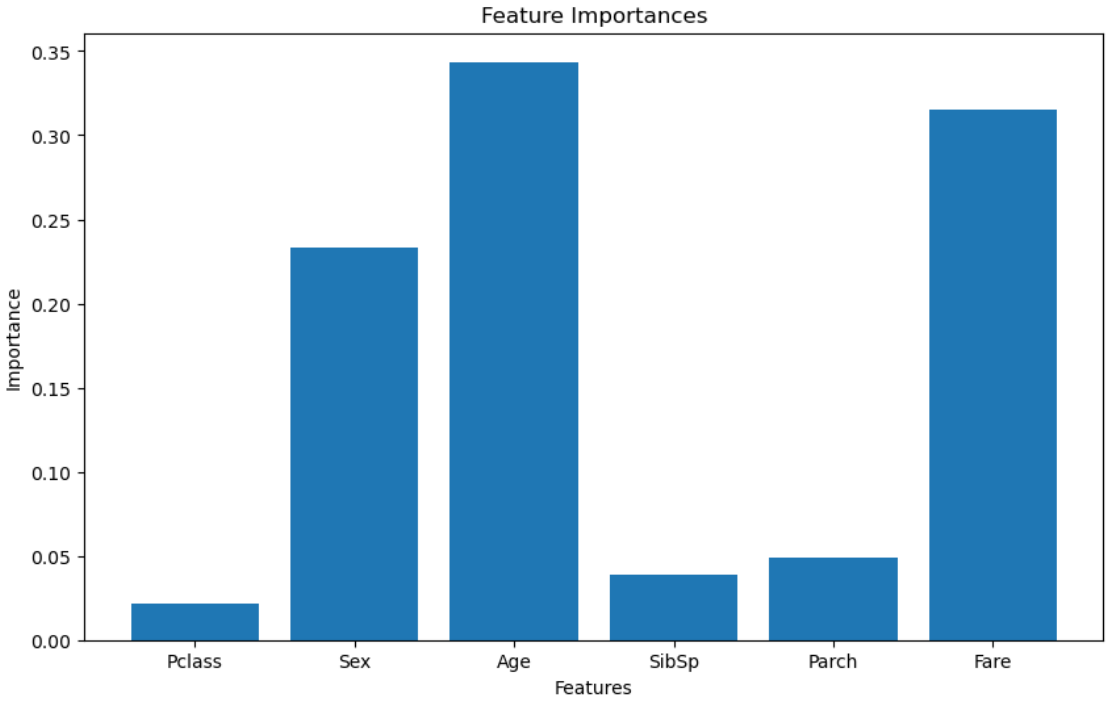
[21]:
```
# Evaluate model performance
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:", classification_report(y_test, y_pred))
print("Confusion Matrix:", confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.7297297297297297
Classification Report:               precision    recall  f1-score   support

           0       0.67      0.57      0.62        14
           1       0.76      0.83      0.79        23

    accuracy                           0.73        37
   macro avg       0.71      0.70      0.70        37
weighted avg       0.72      0.73      0.72        37

Confusion Matrix: [[ 8  6]
 [ 4 19]]
```

[22]:
```
# Plot Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap='Blues')
plt.xlabel("Predicted labels")
plt.ylabel("True labels")
plt.show()
```



[23]:
```
# Plot Feature Importances
feature_importances = model.feature_importances_
plt.figure(figsize=(10, 6))
plt.bar(X.columns, feature_importances)
plt.xlabel("Features")
plt.ylabel("Importance")
plt.title("Feature Importances")
plt.show()
```



[24]:
```
# Define target variable (y) and feature variables (X)
y = data['Age']
X = data[['Pclass', 'Sex', 'SibSp', 'Parch', 'Fare']]
```

[25]:
```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[26]:
```
# Create and train Random Forest Regressor model
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

[26]:
```
RandomForestRegressor(random_state=42)
```

[28]:
```
# Make predictions on test data
y_pred = model.predict(X_test)
print(y_pred)
```

```
[40.745       5.64083333 19.15738095 22.45       36.57333333 46.08
 38.985      36.381      32.27714286 42.79       42.35325    57.49533333
 43.346      45.08166667 50.81       39.67977778 31.266      51.52666667
 28.03219048 26.027      26.35       47.82383333 36.328      11.24916667
 39.21977778 15.255      22.705      39.31566667 38.5407381  42.47833333
 28.14       21.262      45.57133333 36.08       49.33166667 37.04333333
```

37.07733333]

[29]:
```
# Evaluate model performance
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = math.sqrt(mse)
r2 = r2_score(y_test, y_pred)
print("Mean Squared Error:", mse)
print("Mean Absolute Error:", mae)
print("Root Mean Squared Error:", rmse)
print("R-squared:", r2)
```

```
Mean Squared Error: 182.0158632058968
Mean Absolute Error: 11.645574967824965
Root Mean Squared Error: 13.491325479948099
R-squared: 0.3332568698068825
```

[30]:
```
# Plot predicted vs actual values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred)
plt.xlabel("Actual Age")
plt.ylabel("Predicted Age")
plt.show()
```