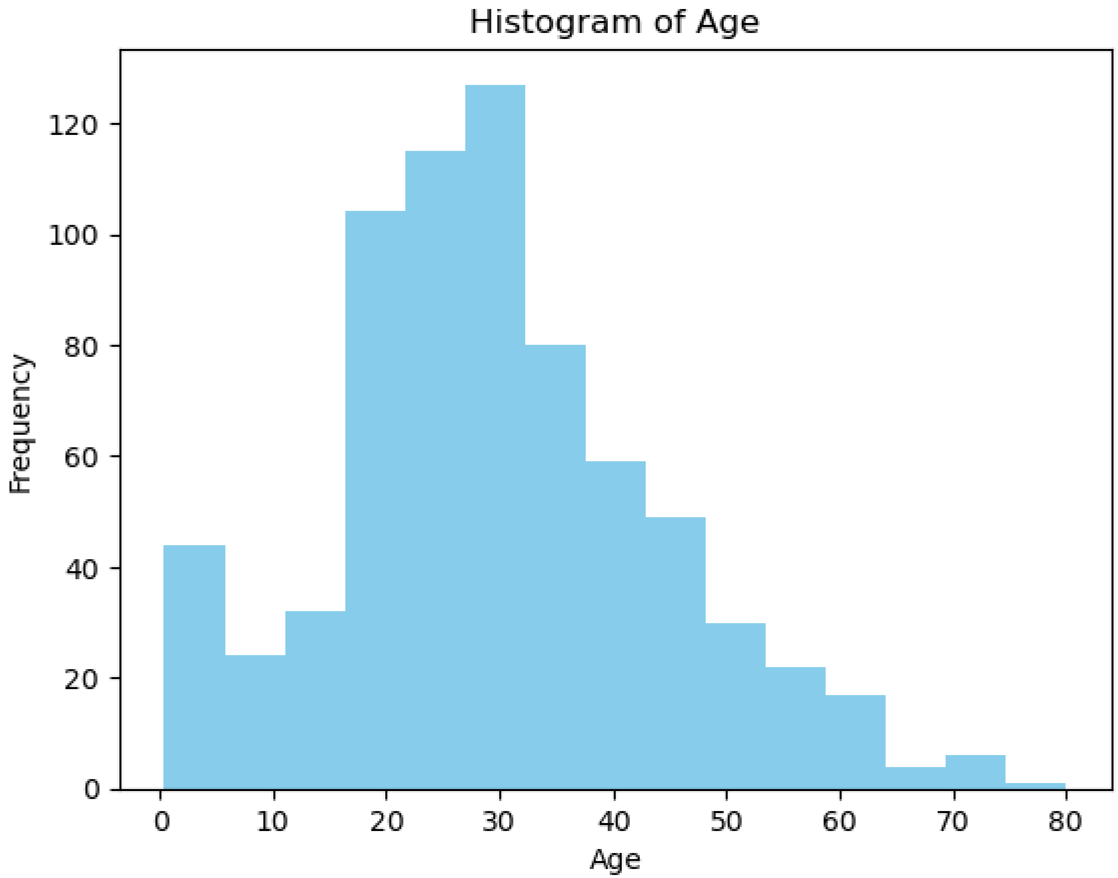
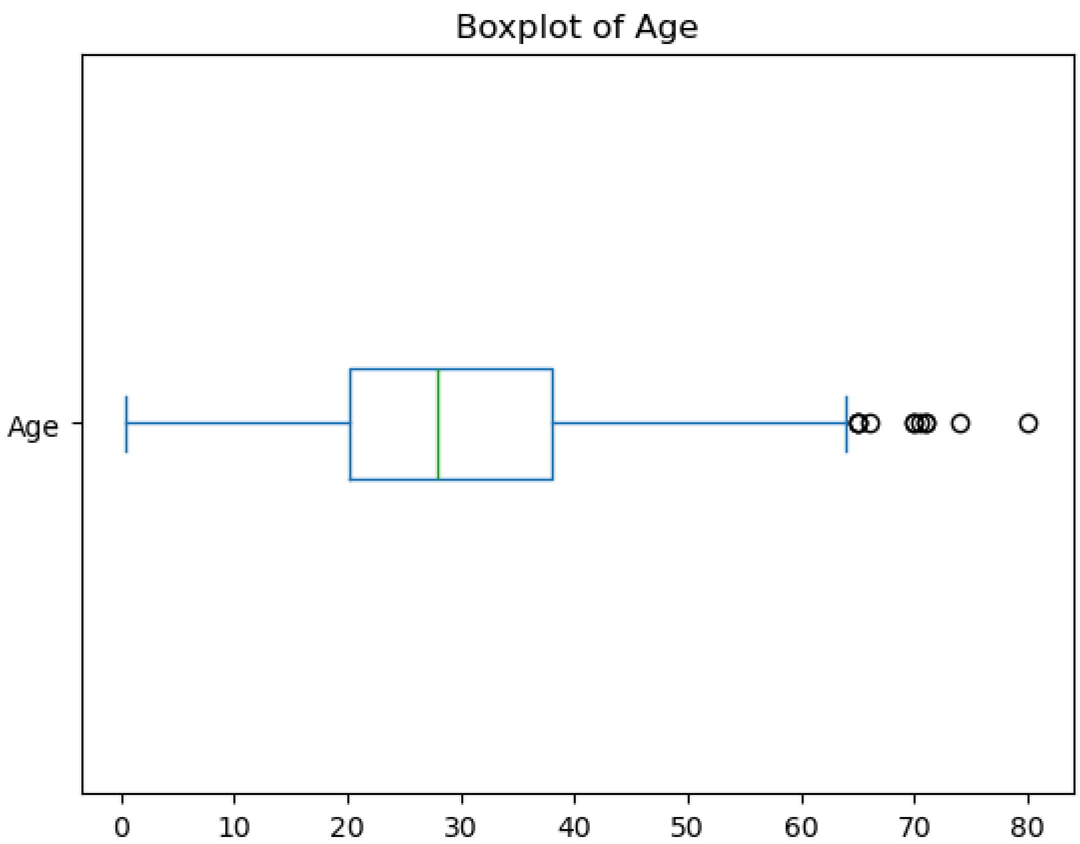
In [19]:
```
#1
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('titanic.csv')
print("Columns:", df.columns.tolist())

# Numerical analysis for 'Age'
num_col = 'Age'
data = df[num_col].dropna()
print("Mean:  ", data.mean())
print("Median:", data.median())
print("Mode:  ", data.mode().iat[0])
print("Std:   ", data.std())
print("Var:   ", data.var())
print("Range: ", data.max() - data.min())

# Histogram (no grid lines)
data.hist(bins=15, color='skyblue')
plt.title("Histogram of Age")
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.grid(False)  # Removes the grid lines
plt.show()

# Boxplot
data.plot.box(vert=False)
plt.title("Boxplot of Age")
plt.show()

# Outlier detection
Q1 = data.quantile(0.25)
Q3 = data.quantile(0.75)
IQR = Q3 - Q1
outliers = data[(data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5 * IQR))]
print("Number of outliers:", len(outliers))

# Categorical analysis for 'Sex'
cat_col = 'Sex'
```

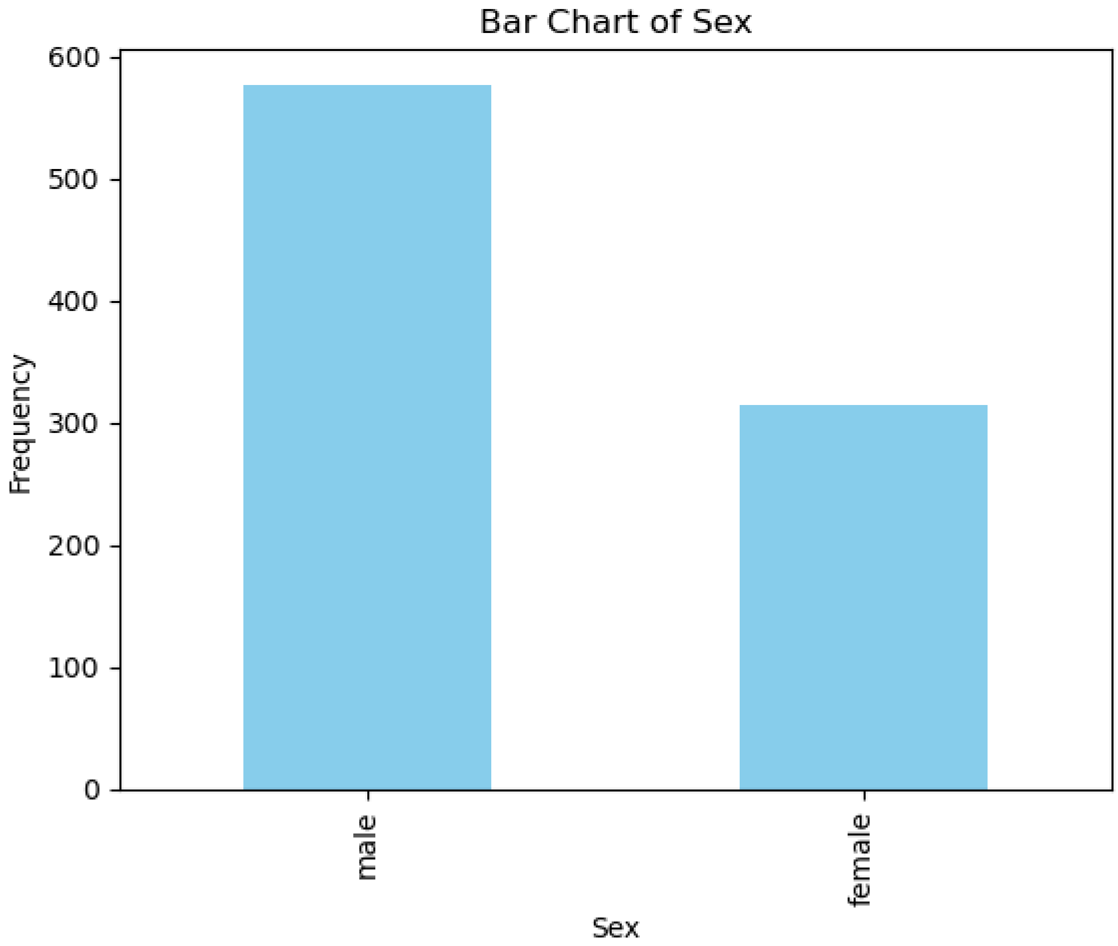
```python
cat_data = df[cat_col].value_counts()

# Bar chart
cat_data.plot(kind='bar', color='skyblue')
plt.title(f"Bar Chart of {cat_col}")
plt.xlabel(cat_col)
plt.ylabel("Frequency")
plt.show()

# Pie chart
cat_data.plot(kind='pie', autopct='%1.1f%%', startangle=90)
plt.title(f"Pie Chart of {cat_col}")
plt.ylabel("")  # Hide y-label
plt.axis('equal')  # Keeps pie chart circular
plt.show()
```

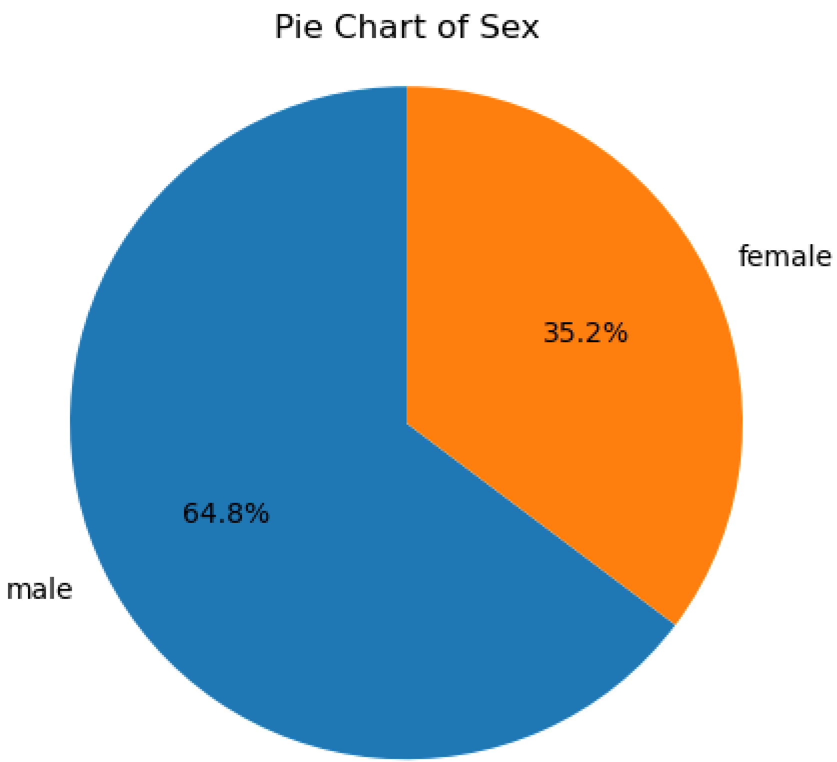
```
Columns: ['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked']
Mean:   29.69911764705882
Median: 28.0
Mode:   24.0
Std:    14.526497332334044
Var:    211.0191247463081
Range:  79.58
```


Histogram of Age
Frequency
Age
0
20
40
60
80
100
120
0
10
20
30
40
50
60
70
80




```
Number of outliers: 11
```

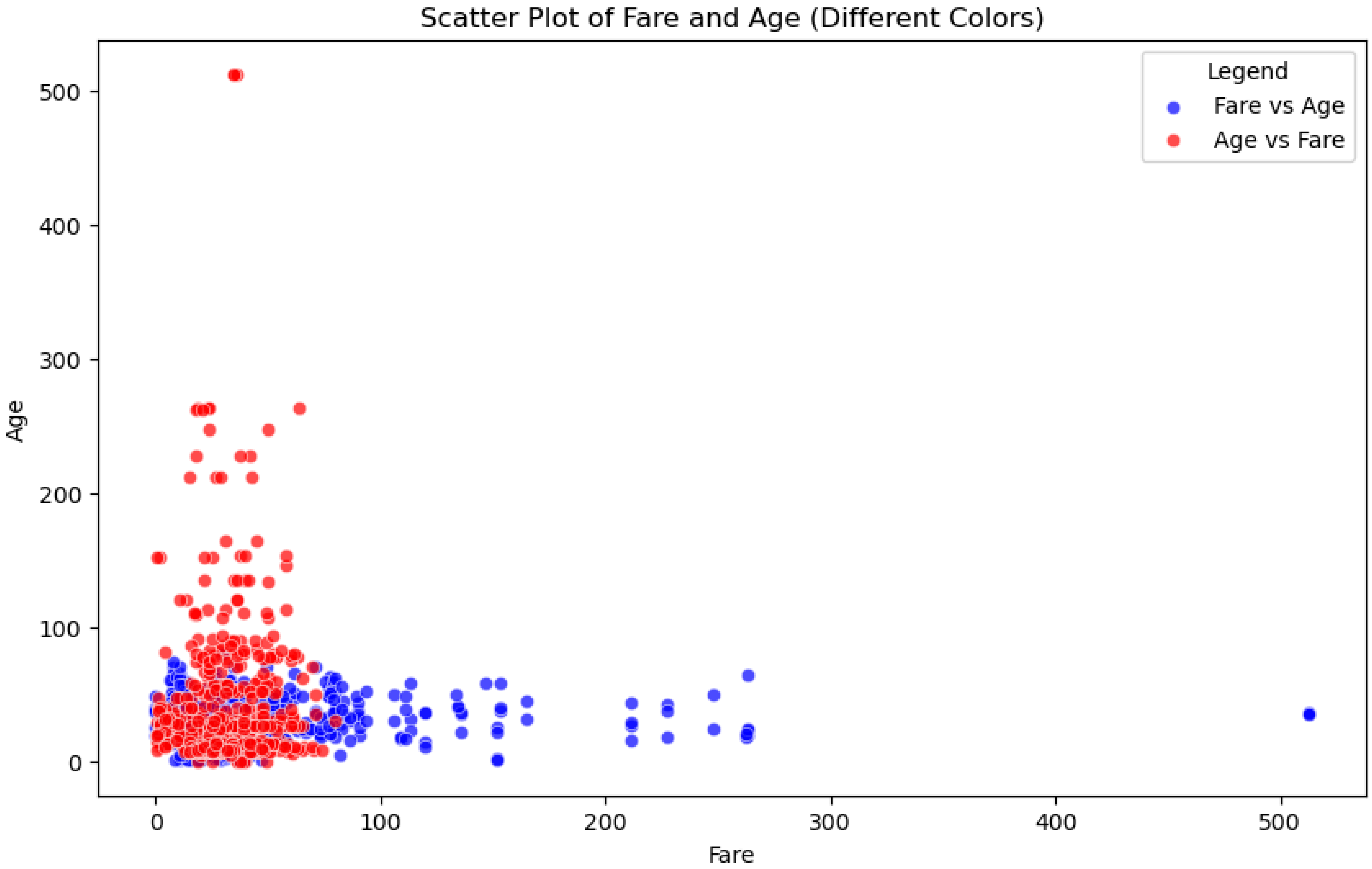

Bar Chart of Sex
600
500
400
300
200
100
0
Frequency
male
female
Sex



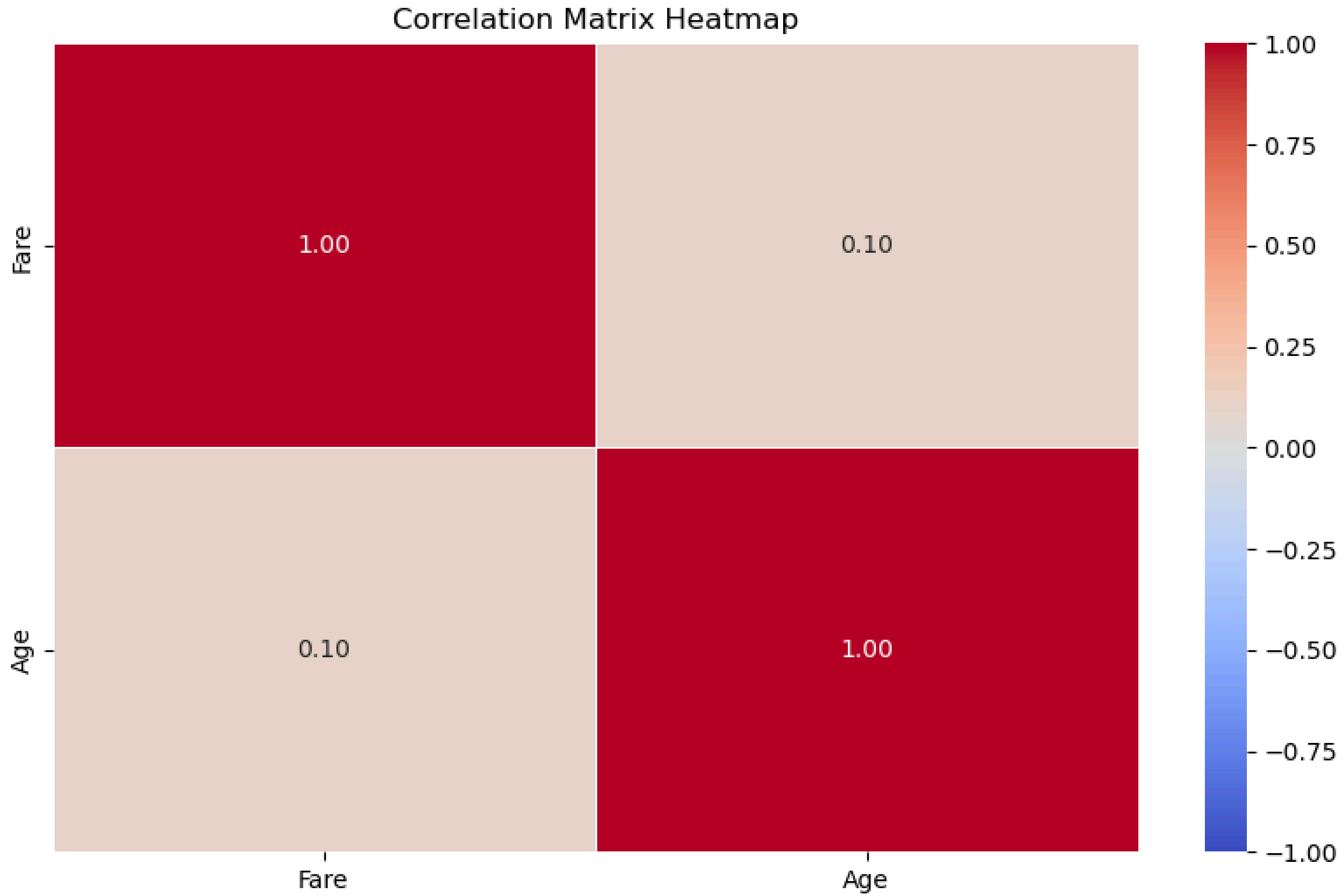
In [31]:

```
#2
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import pearsonr

# Load dataset and select numerical columns for analysis
df = pd.read_csv('titanic.csv')
x_col, y_col = 'Fare', 'Age'
data = df[[x_col, y_col]].dropna()

# Scatter plot with two colors
plt.figure(figsize=(10, 6))
sns.scatterplot(x=data[x_col], y=data[y_col], color='blue', label=f'{x_col} vs {y_col}', alpha=0.7)
sns.scatterplot(x=data[y_col], y=data[x_col], color='red', label=f'{y_col} vs {x_col}', alpha=0.7)
```

```
plt.title(f"Scatter Plot of {x_col} and {y_col} (Different Colors)")
plt.xlabel(x_col)
plt.ylabel(y_col)
plt.legend(title='Legend')
plt.show()

# Pearson correlation
corr, _ = pearsonr(data[x_col], data[y_col])
print(f"Pearson Correlation Coefficient: {corr:.3f}")

# Covariance and correlation matrices for the selected columns
print("Covariance Matrix:\n", data.cov())
print("\nCorrelation Matrix:\n", data.corr())

# Correlation heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(data.corr(), annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5, vmin=-1, vmax=1)
plt.title("Correlation Matrix Heatmap")
plt.show()
```

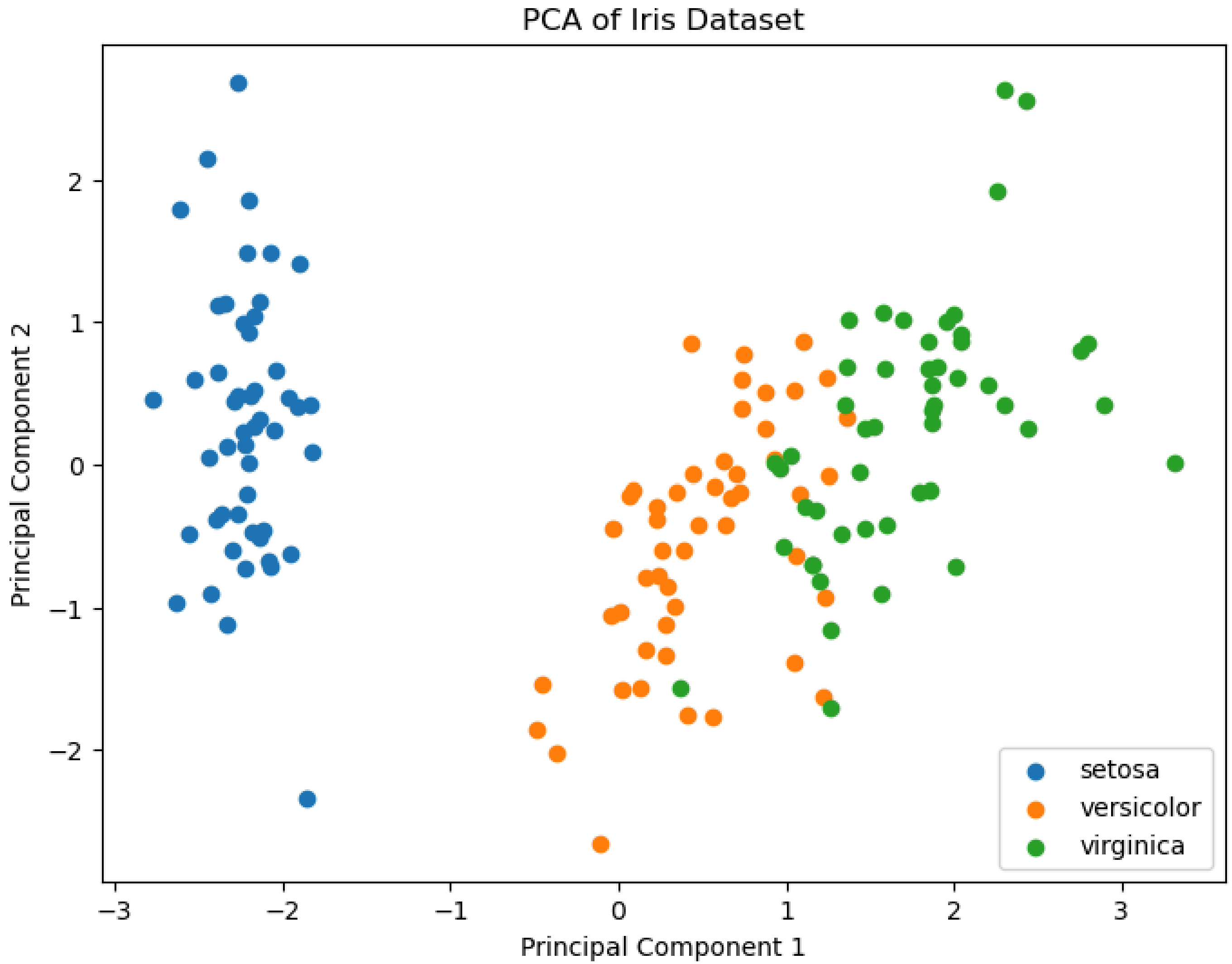

Scatter Plot of Fare and Age (Different Colors)
Legend
Fare vs Age
Age vs Fare
Age
0
100
200
300
400
500
Fare
0
100
200
300
400
500

```
Pearson Correlation Coefficient: 0.096
Covariance Matrix:
             Fare         Age
Fare  2800.41310   73.849030
Age     73.84903  211.019125

Correlation Matrix:
           Fare       Age
Fare  1.000000  0.096067
Age   0.096067  1.000000
```



In [37]:
```
#3
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Load the Iris dataset
```

```python
iris = datasets.load_iris()
X = iris.data  # 4 features
y = iris.target

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA to reduce dimensions from 4 to 2
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Plot the PCA-transformed data
plt.figure(figsize=(8, 6))
for i, target_name in enumerate(iris.target_names):
    plt.scatter(X_pca[y == i, 0], X_pca[y == i, 1], label=target_name)
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA of Iris Dataset')
plt.legend()
plt.show()
```



In [38]:
```
#4
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
```

```python
from sklearn.metrics import accuracy_score, f1_score

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split dataset into training (80%) and testing (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Function to evaluate k-NN with different k values
def evaluate_knn(k_values, weighted=False):
    accuracies = []
    f1_scores = []

    for k in k_values:
        weight_type = "distance" if weighted else "uniform"

        # Create k-NN model
        knn = KNeighborsClassifier(n_neighbors=k, weights=weight_type)
        knn.fit(X_train, y_train)

        # Predict on test set
        y_pred = knn.predict(X_test)

        # Compute accuracy & F1-score
        acc = accuracy_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred, average="weighted")

        accuracies.append(acc)
        f1_scores.append(f1)

        print(f"k={k}, Weighted={weighted}, Accuracy={acc:.4f}, F1-score={f1:.4f}")

    return accuracies, f1_scores

# Test k-NN for k=1,3,5 without weights
k_values = [1, 3, 5]
print("\n### k-NN without distance-based weighting ###")
evaluate_knn(k_values, weighted=False)
```

```python
# Test k-NN for k=1,3,5 with distance-based weighting (1/d^2)
print("\n### k-NN with distance-based weighting ###")
evaluate_knn(k_values, weighted=True)
```

```
### k-NN without distance-based weighting ###
k=1, Weighted=False, Accuracy=1.0000, F1-score=1.0000
k=3, Weighted=False, Accuracy=1.0000, F1-score=1.0000
k=5, Weighted=False, Accuracy=1.0000, F1-score=1.0000

### k-NN with distance-based weighting ###
k=1, Weighted=True, Accuracy=1.0000, F1-score=1.0000
k=3, Weighted=True, Accuracy=1.0000, F1-score=1.0000
k=5, Weighted=True, Accuracy=1.0000, F1-score=1.0000
```

Out[38]:
```
([1.0, 1.0, 1.0], [1.0, 1.0, 1.0])
```

In [ ]: