SCHOOL OF COMPUTER SCIENCE ENGINEERING AND INFORMATION SYSTEMS

Winter Semester 2023 – 2024

SURVIVAL PREDICTION ON THE TITANIC SHIP

Programme Name and Branch: BTech (IT)

BITE410L – Machine Learning

Slot – A2+TA2

Faculty Name: Dr Valarmathi

Name: Shrinidhi B
Reg no: 21BIT0381

SURVIVAL PREDICTION ON THE TITANIC SHIP

Dataset link: https://www.kaggle.com/competitions/titanic/data

Description:

The Titanic dataset is a classic dataset in data science and machine learning, often used for educational purposes and analysis. It contains information about passengers aboard the RMS Titanic.

1. Survival: Indicates whether a passenger survived or not (0 = No, 1 = Yes).

2. Pclass (Ticket class): Represents the socio-economic status of the passenger, categorized into three classes: 1st (Upper), 2nd (Middle), and 3rd (Lower).

3. Sex: Specifies the gender of the passenger.

4. Age: Represents the age of the passenger in years. It may include fractional values for infants and estimated ages in the form of xx.5.

5. Sibsp: Indicates the number of siblings/spouses aboard the Titanic for a particular passenger.

6. Parch: Represents the number of parents/children aboard the Titanic for a particular passenger.

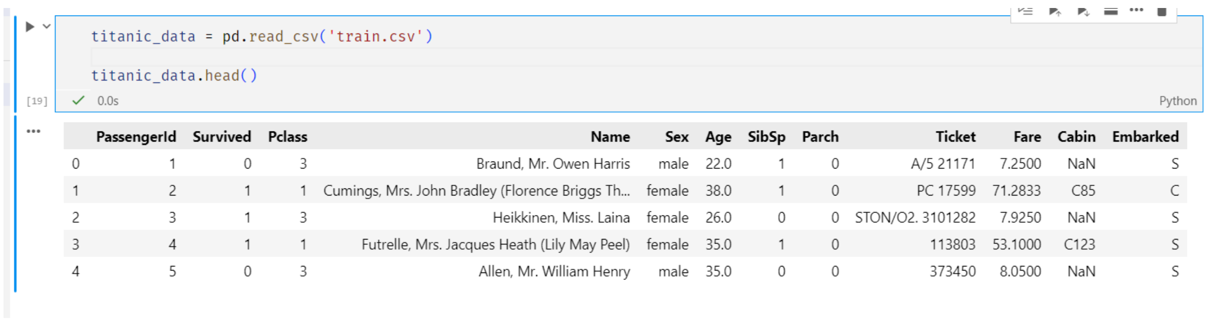
7. Ticket: Specifies the ticket number of the passenger.

8. Fare: Represents the passenger fare.

9. Cabin: Indicates the cabin number of the passenger.

10. Embarked (Port of Embarkation): Specifies the port of embarkation for the passenger, categorized as C (Cherbourg), Q (Queenstown), or S (Southampton).

Sample Data:



```
titanic_data = pd.read_csv('train.csv')

titanic_data.head()
```

[19] 0.0s Python

	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

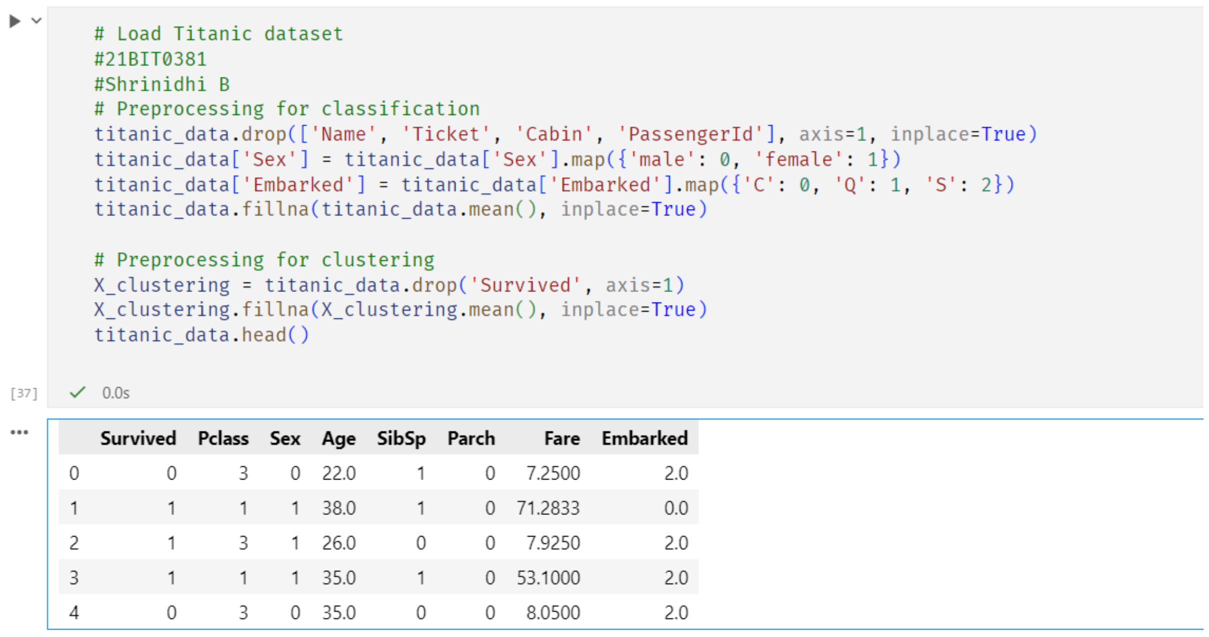
Data Pre-Processing Technique:

Preprocessing for Classification:

1. **Dropping Columns**: The code drops the columns 'Name', 'Ticket', 'Cabin', and 'PassengerId' using the drop() method along the columns axis (axis=1). These columns are not used for classification and are removed to simplify the dataset.
2. **Mapping Categorical Variables to Numerical Values**:
   - **Sex Mapping**: The 'Sex' column is mapped from categorical values ('male' and 'female') to numerical values (0 and 1) using the map() method.
   - **Embarked Mapping**: The 'Embarked' column is mapped from categorical values ('C', 'Q', 'S') to numerical values (0, 1, 2) using the map() method.
3. **Handling Missing Values**:
   - Missing values are filled with the mean of each respective column using the fillna() method with the mean value of the Data Frame.

Preprocessing for Clustering:

1. **Dropping Target Column**:
   - The 'Survived' column, which represents the target variable for classification, is dropped from the dataset using the drop() method along the columns axis (axis=1). This column is not relevant for clustering, as clustering is an unsupervised learning technique.
2. **Handling Missing Values**:
   - Similar to the classification preprocessing, missing values in the remaining columns are filled with the mean of each respective column using the fillna() method with the mean value of the DataFrame.



```
# Load Titanic dataset
#21BIT0381
#Shrinidhi B
# Preprocessing for classification
titanic_data.drop(['Name', 'Ticket', 'Cabin', 'PassengerId'], axis=1, inplace=True)
titanic_data['Sex'] = titanic_data['Sex'].map({'male': 0, 'female': 1})
titanic_data['Embarked'] = titanic_data['Embarked'].map({'C': 0, 'Q': 1, 'S': 2})
titanic_data.fillna(titanic_data.mean(), inplace=True)

# Preprocessing for clustering
X_clustering = titanic_data.drop('Survived', axis=1)
X_clustering.fillna(X_clustering.mean(), inplace=True)
titanic_data.head()
```

[37] ✓ 0.0s

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	0	22.0	1	0	7.2500	2.0
1	1	1	1	38.0	1	0	71.2833	0.0
2	1	3	1	26.0	0	0	7.9250	2.0
3	1	1	1	35.0	1	0	53.1000	2.0
4	0	3	0	35.0	0	0	8.0500	2.0

XGBoost Model:

1. Model Training:

The code fits an XGBoost model (`xgb_model`) to the training data (`X_train`, `y_train`) using the `fit()` method. This step involves the model learning the patterns in the training data to make predictions.

2. Cross-Validation:

Cross-validation is performed using the `cross_val_score()` function with 5-fold cross-validation (`cv=5`). This step helps estimate the performance of the model on unseen data and assess its generalization ability. The cross-validation scores are printed to evaluate the model's performance.

Hyperparameter Tuning using GridSearchCV:

1. Defining Hyperparameter Grid:

A dictionary `param_grid` is defined, specifying the hyperparameters and their respective values to be searched during grid search. Here, `n_estimators` (number of trees) and `max_depth` (maximum depth of each tree) are the hyperparameters to be tuned.

2. Grid Search:

Grid search is performed using `GridSearchCV` to find the best combination of hyperparameters that maximize model performance. The grid search is conducted with 5-fold cross-validation (`cv=5`). The best model obtained from grid search (`best_xgb_model`) is printed.

3. Model Evaluation on Test Data:

The performance of the best XGBoost model is evaluated on the test data (`X_test`, `y_test`) using the `score()` method, which computes the accuracy of the model on the test data. The test accuracy is printed to assess the final model performance.

KMeans Clustering:

1. Initializing KMeans Algorithm:

KMeans clustering algorithm is initialized with `n_clusters=3`, specifying the number of clusters to be formed. The `random_state` parameter ensures reproducibility of results.

2. Fitting KMeans to Data:

The KMeans algorithm is fitted to the preprocessed data `X_clustering` using the `fit()` method. This step assigns each data point to one of the clusters based on their similarity.

3. Adding Cluster Labels:

The cluster labels obtained from KMeans clustering are added as a new column `'Cluster'` to the original Titanic dataset (`titanic_data`). Each row in the dataset is now associated with a cluster label.

4. Displaying Clustered Dataset:

The first few rows of the Titanic dataset with the added cluster labels are displayed using the `head()` method to visually inspect the clustering results.

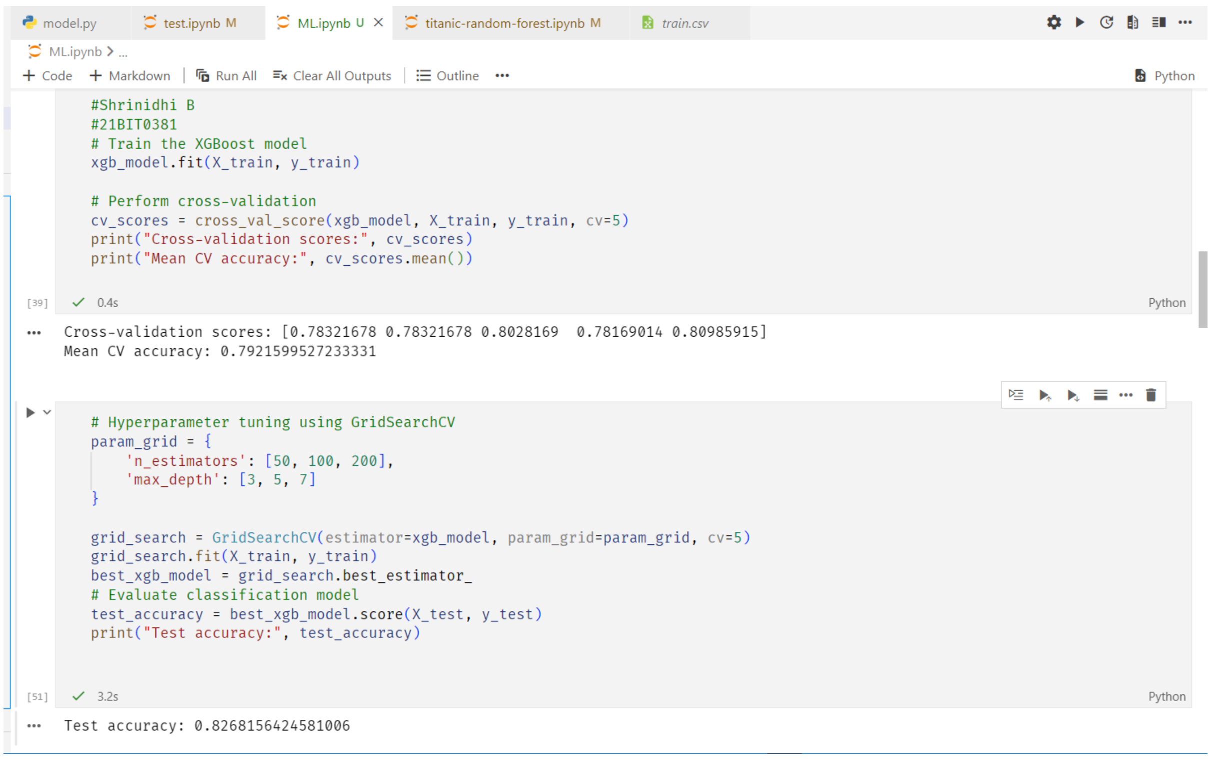
Classification:

Code:

```
# Initialize XGBoost classifier
#Shrinidhi B
#21BIT0381
xgb_model = XGBClassifier(random_state=42)

# Split the data for classification
X_classification = titanic_data.drop('Survived', axis=1)
y_classification = titanic_data['Survived']
X_train, X_test, y_train, y_test = train_test_split(X_classification, y_classification, test_size=0.2,
random_state=42)

# Evaluate classification model
test_accuracy = best_xgb_model.score(X_test, y_test)
print("Test accuracy:", test_accuracy)
```



```
#Shrinidhi B
#21BIT0381
# Train the XGBoost model
xgb_model.fit(X_train, y_train)

# Perform cross-validation
cv_scores = cross_val_score(xgb_model, X_train, y_train, cv=5)
print("Cross-validation scores:", cv_scores)
print("Mean CV accuracy:", cv_scores.mean())
```

[39] ✓ 0.4s

```
Cross-validation scores: [0.78321678 0.78321678 0.8028169  0.78169014 0.80985915]
Mean CV accuracy: 0.7921599527233331
```

```
# Hyperparameter tuning using GridSearchCV
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7]
}

grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid, cv=5)
grid_search.fit(X_train, y_train)
best_xgb_model = grid_search.best_estimator_
# Evaluate classification model
test_accuracy = best_xgb_model.score(X_test, y_test)
print("Test accuracy:", test_accuracy)
```

[51] ✓ 3.2s

```
Test accuracy: 0.8268156424581006
```

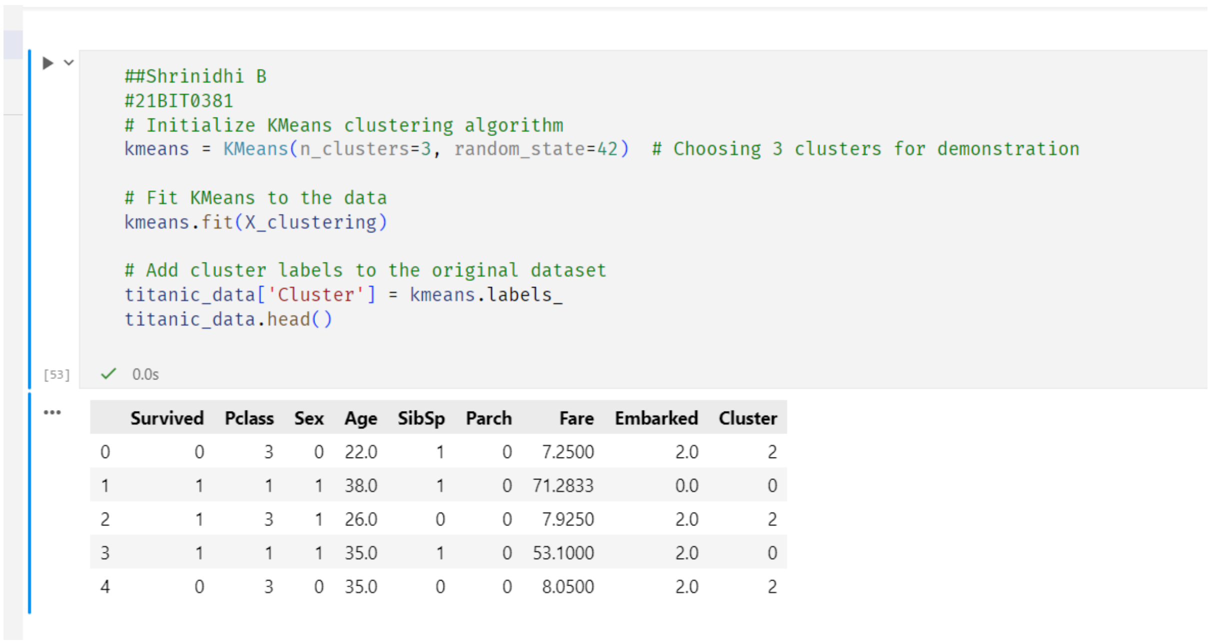
Clustering:

Code:

```
##Shrinidhi B
#21BIT0381
# Initialize KMeans clustering algorithm
kmeans = KMeans(n_clusters=3, random_state=42)  # Choosing 3 clusters for demonstration

# Fit KMeans to the data
kmeans.fit(X_clustering)

# Add cluster labels to the original dataset
titanic_data['Cluster'] = kmeans.labels_
titanic_data.head()
```



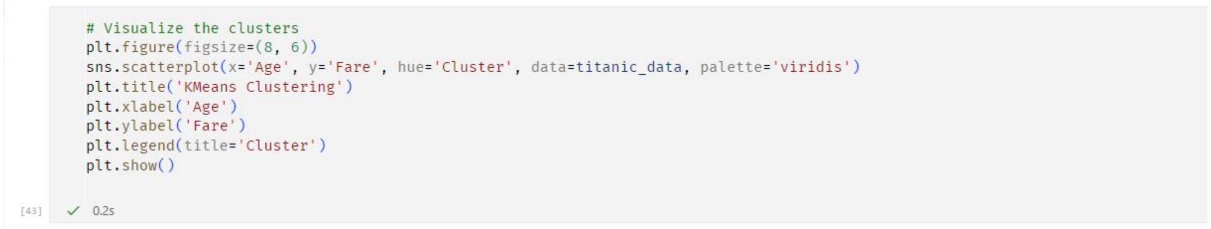
```
##Shrinidhi B
#21BIT0381
# Initialize KMeans clustering algorithm
kmeans = KMeans(n_clusters=3, random_state=42)  # Choosing 3 clusters for demonstration

# Fit KMeans to the data
kmeans.fit(X_clustering)

# Add cluster labels to the original dataset
titanic_data['Cluster'] = kmeans.labels_
titanic_data.head()
```

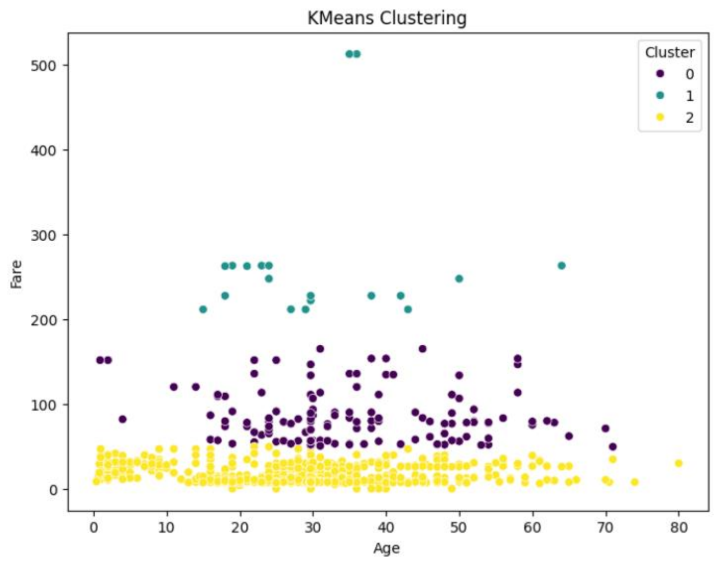
[53] ✓ 0.0s

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Cluster
0	0	3	0	22.0	1	0	7.2500	2.0	2
1	1	1	1	38.0	1	0	71.2833	0.0	0
2	1	3	1	26.0	0	0	7.9250	2.0	2
3	1	1	1	35.0	1	0	53.1000	2.0	0
4	0	3	0	35.0	0	0	8.0500	2.0	2



```
# Visualize the clusters
plt.figure(figsize=(8, 6))
sns.scatterplot(x='Age', y='Fare', hue='Cluster', data=titanic_data, palette='viridis')
plt.title('KMeans Clustering')
plt.xlabel('Age')
plt.ylabel('Fare')
plt.legend(title='Cluster')
plt.show()
```

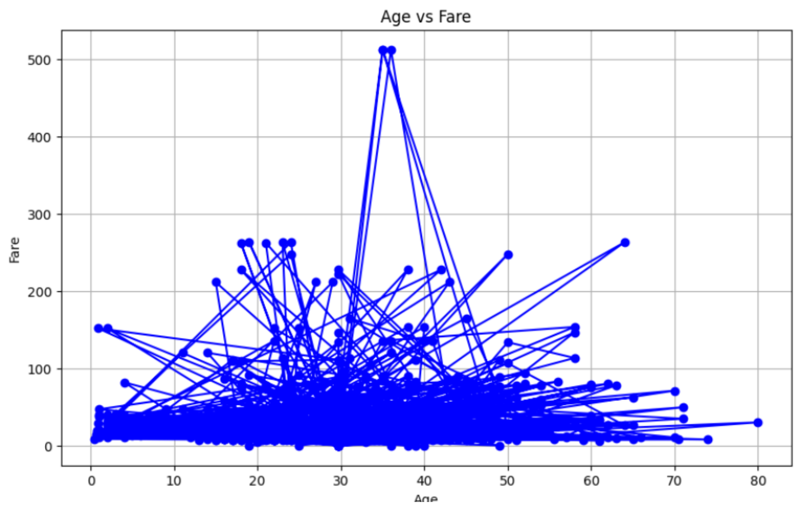
[43] ✓ 0.2s



Data visualization techniques:

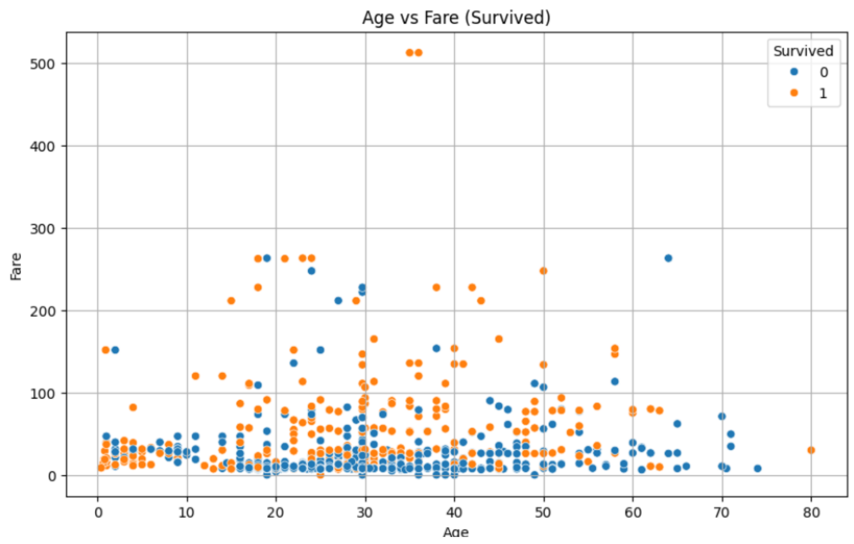
```
# Data Visualization Techniques
#Shrinidhi B
#21BIT0381
# Line Graph
plt.figure(figsize=(10, 6))
plt.plot(titanic_data['Age'], titanic_data['Fare'], marker='o', linestyle='-', color='b')
plt.title('Age vs Fare')
plt.xlabel('Age')
plt.ylabel('Fare')
plt.grid(True)
plt.show()
```

[44] ✓ 0.1s



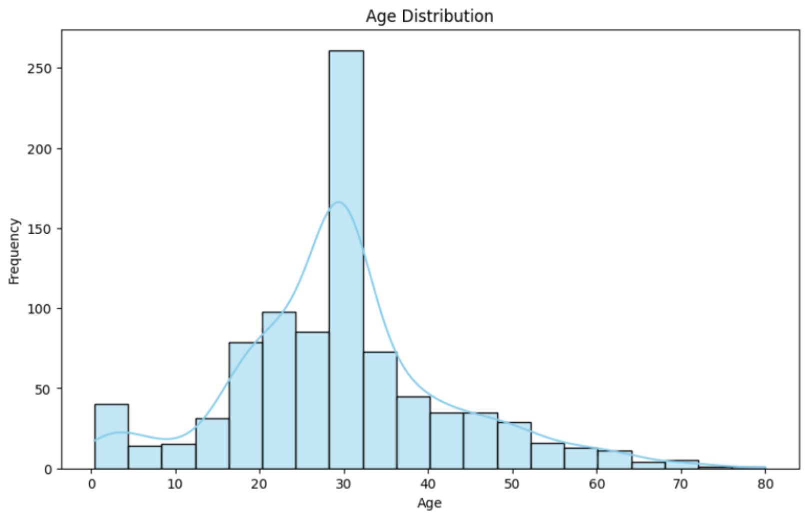
```
# Scatter Plot
#Shrinidhi B
#21BIT0381
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Age', y='Fare', data=titanic_data, hue='Survived')
plt.title('Age vs Fare (Survived)')
plt.xlabel('Age')
plt.ylabel('Fare')
plt.legend(title='Survived', loc='upper right')
plt.grid(True)
plt.show()
```

[45] ✓ 0.2s



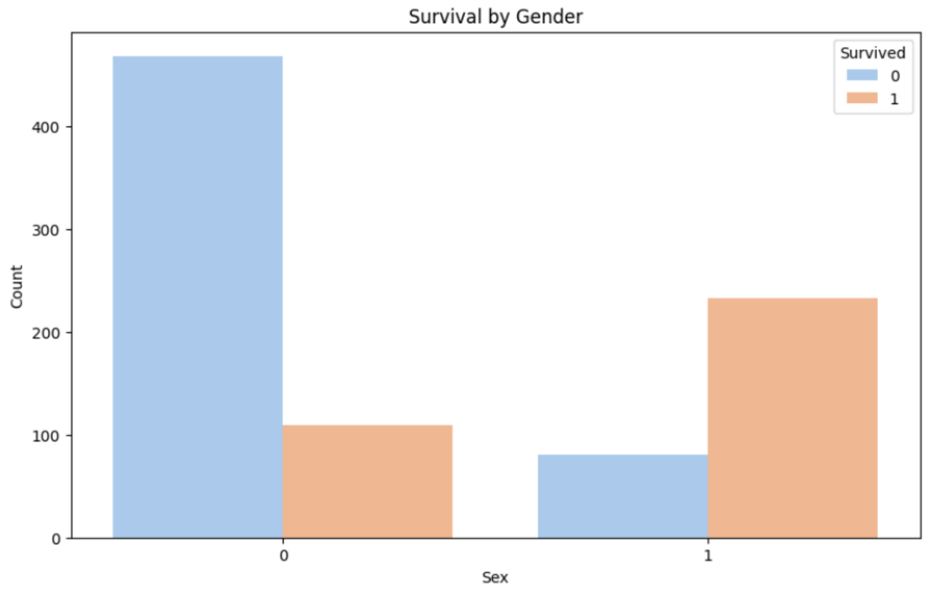
```
#Shrinidhi B
#21BIT0381
# Bar Chart for Numerical Variable
plt.figure(figsize=(10, 6))
sns.histplot(titanic_data['Age'], bins=20, kde=True, color='skyblue')
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```

[46] ✓ 0.1s



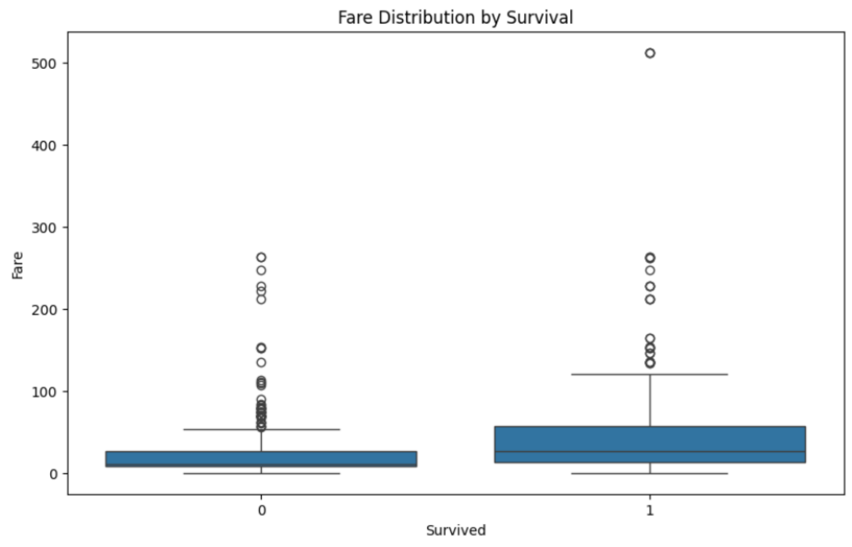
```
#Shrinidhi B
#21BIT0381
# Bar Chart for Categorical Variable
plt.figure(figsize=(10, 6))
sns.countplot(x='Sex', hue='Survived', data=titanic_data, palette='pastel')
plt.title('Survival by Gender')
plt.xlabel('Sex')
plt.ylabel('Count')
plt.legend(title='Survived', loc='upper right')
plt.show()
```

[47] ✓ 0.2s



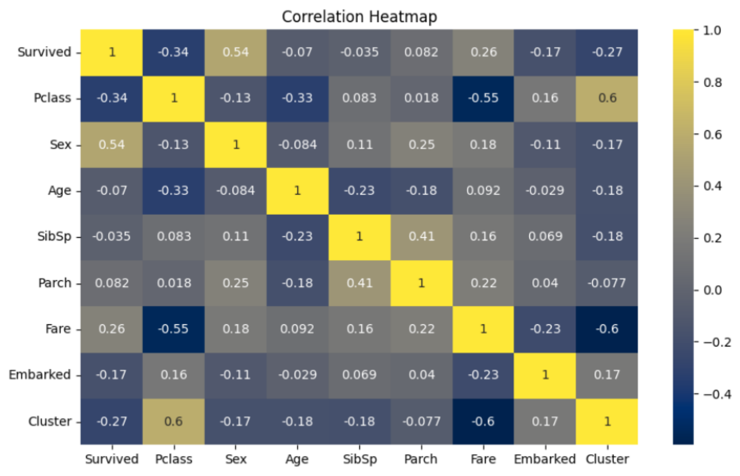
```python
#Shrinidhi B
#21BIT0381
# Boxplot
plt.figure(figsize=(10, 6))
sns.boxplot(x='Survived', y='Fare', data=titanic_data)
plt.title('Fare Distribution by Survival')
plt.xlabel('Survived')
plt.ylabel('Fare')
plt.show()
```

[48] ✓ 0.1s

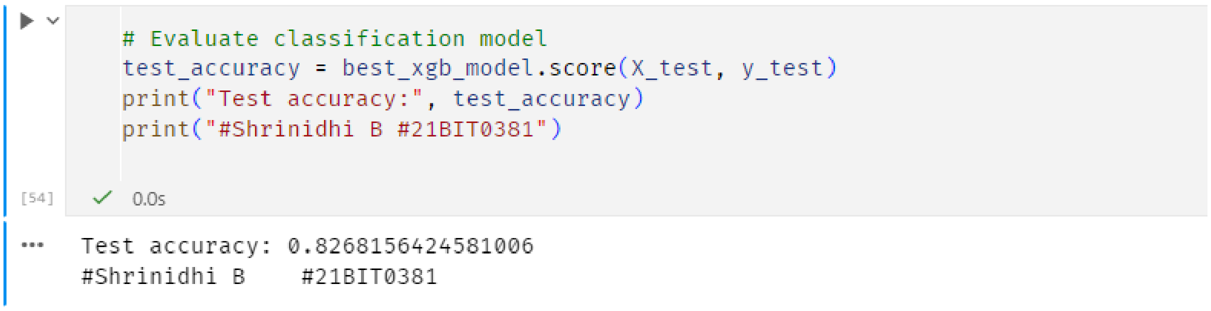


```python
#Shrinidhi B
#21BIT0381
plt.figure(figsize=(10, 6))
sns.heatmap(titanic_data.corr(), annot=True, cmap='cividis')
plt.title('Correlation Heatmap')
plt.show()
```

[49] ✓ 0.4s

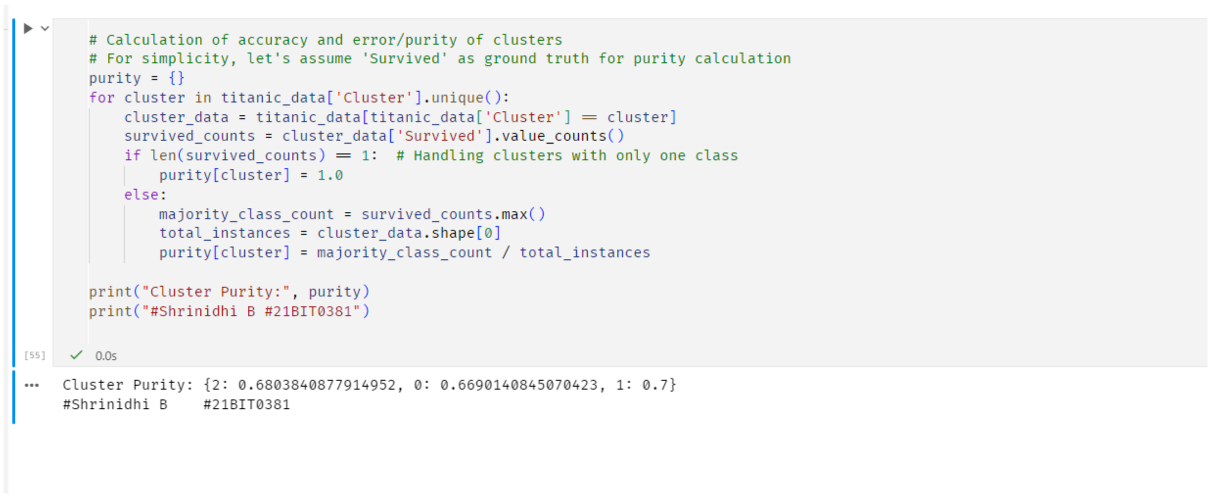


Calculation of accuracy and error / Purity of cluster:



```python
# Evaluate classification model
test_accuracy = best_xgb_model.score(X_test, y_test)
print("Test accuracy:", test_accuracy)
print("#Shrinidhi B #21BIT0381")
```

```
Test accuracy: 0.8268156424581006
#Shrinidhi B    #21BIT0381
```



```python
# Calculation of accuracy and error/purity of clusters
# For simplicity, let's assume 'Survived' as ground truth for purity calculation
purity = {}
for cluster in titanic_data['Cluster'].unique():
    cluster_data = titanic_data[titanic_data['Cluster'] == cluster]
    survived_counts = cluster_data['Survived'].value_counts()
    if len(survived_counts) == 1:  # Handling clusters with only one class
        purity[cluster] = 1.0
    else:
        majority_class_count = survived_counts.max()
        total_instances = cluster_data.shape[0]
        purity[cluster] = majority_class_count / total_instances

print("Cluster Purity:", purity)
print("#Shrinidhi B #21BIT0381")
```

```
Cluster Purity: {2: 0.6803840877914952, 0: 0.6690140845070423, 1: 0.7}
#Shrinidhi B    #21BIT0381
```