Certainly, let's document the code step by step:

**Data Exploration:**

**Reading Data:**

* + **df1**: Training data with columns such as **PassengerId**, **Survived**, **Pclass**, **Name**, **Sex**, **Age**, **SibSp**, **Parch**, **Ticket**, **Fare**, **Cabin**, **Embarked**.
  + **df2**: Test data with similar columns, excluding **Survived**.
  + **df3**: Gender submission data with columns **PassengerId** and **Survived**.

**1. df1 = pd.read\_csv(r"C:\Users\salma elbadry\Desktop\Projects\inter career\task1\train.csv")**

**2. df2 = pd.read\_csv(r"C:\Users\salma elbadry\Desktop\Projects\inter career\task1\test.csv")**

**df3 = pd.read\_csv(r"C:\Users\salma elbadry\Desktop\Projects\inter career\task1\gender\_submission.csv")**

**Data Shape and Info:**

**display the shape and information about the training, test, and submission datasets.**

**print(df1.shape)**

**print(df2.shape)**

**print(df3.shape)**

**df1.info()**

**Handling Missing Data:**

**Check the percentage of missing values in each column.**

**Drop the "Cabin" column and fill missing values in "Age" with the mean, and in "Embarked" with the mode.**

**df1.isna().sum() / len(df1) \* 100**

**df1.drop("Cabin", axis=1, inplace=True)**

**df1["Age"].fillna(df1["Age"].mean(), inplace=True)**

**df1["Embarked"].fillna(df1["Embarked"].mode()[0], inplace=True)**

**Data Visualization:**

**Visualize data using various plots.**

**sns.countplot(data=df1, x="Embarked")**

**A graph with different colored bars

Description automatically generated**

**sns.barplot(data=df1, x="Embarked", y="Survived")**

**A graph of different colored bars

Description automatically generated with medium confidence**

**sns.barplot(data=df1, x="Sex", y="Age")**

A blue and orange rectangular bars

Description automatically generated

**sns.countplot(data=df1, x="Sex")**

**A graph with blue and orange squares

Description automatically generated**

**sns.barplot(data=df1, x="SibSp", y="Survived")**

**A graph with different colored bars

Description automatically generated**

**sns.pairplot(df1)**

**A screenshot of a graph

Description automatically generated**

**Encoding Categorical Features:**

**Encode "Sex" and "Embarked" using label encoding.**

**label\_encoder = LabelEncoder()**

**df1["Sex"] = label\_encoder.fit\_transform(df1["Sex"])**

**df1["Embarked"] = label\_encoder.fit\_transform(df1["Embarked"])**

**Feature Selection:**

**Drop unnecessary columns from the training dataset.**

**columns\_to\_drop = ["PassengerId", "Name", "Ticket"]**

**df1.drop(columns=columns\_to\_drop, inplace=True)**

**Splitting Data:**

**Split the training dataset into features (x) and target (y).**

**Split the data into training and testing sets.**

**x = df1.drop("Survived", axis=1)**

**y = df1["Survived"]**

**x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.20, shuffle=True, random\_state=168)**

**Model Training and Evaluation:**

**SVM Model Initialization and Training:**

**Initialize an SVM classifier and fit it to the training data.**

**best\_svm = SVC()**

**best\_svm.fit(x\_train, y\_train)**

**Model Evaluation Metrics:**

**Calculate accuracy, precision, recall, and F1-score.**

**y\_pred = best\_svm.predict(x\_test)**

**accuracy = accuracy\_score(y\_test, y\_pred)**

**precision = precision\_score(y\_test, y\_pred)**

**recall = recall\_score(y\_test, y\_pred)**

**f1 = f1\_score(y\_test, y\_pred)**

**Data Scaling and Hyperparameter Tuning:**

**Scale the data using StandardScaler.**

**Perform hyperparameter tuning using Grid Search with parameters C, kernel, and gamma.**

**scaler = StandardScaler()**

**X\_train = scaler.fit\_transform(x\_train)**

**X\_test = scaler.transform(x\_test)**

**param\_grid = {**

**'C': [0.1, 1, 10],**

**'kernel': ['linear', 'rbf'],**

**'gamma': ['scale', 'auto']**

**}**

**grid\_search = GridSearchCV(best\_svm, param\_grid, cv=5)**

**grid\_search.fit(x\_train, y\_train)**

**best\_params = grid\_search.best\_params\_**

**Model Training with Best Hyperparameters:**

**Train the model with the best hyperparameters.**

**best\_svm = SVC(\*\*best\_params)**

**best\_svm.fit(x\_train, y\_train)**

**Model Evaluation and Cross-Validation:**

**Evaluate the model on the test set and print metrics.**

**Perform cross-validation to assess model stability.**

**y\_pred = best\_svm.predict(x\_test)**

**accuracy = accuracy\_score(y\_test, y\_pred)**

**precision = precision\_score(y\_test, y\_pred)**

**recall = recall\_score(y\_test, y\_pred)**

**f1 = f1\_score(y\_test, y\_pred)**

**conf\_matrix = confusion\_matrix(y\_test, y\_pred)**

**kf = KFold(n\_splits=5, shuffle=True, random\_state=42)**

**cross\_val\_scores = cross\_val\_score(best\_svm, X\_train, y\_train, cv=kf, scoring='accuracy')**

**Print Evaluation Metrics and Visualize Confusion Matrix:**

**Print evaluation metrics.**

**Visualize the confusion matrix using a heatmap.**

**print(f"Best Hyperparameters: {best\_params}")**

**print(f"Accuracy: {accuracy:.2f}")**

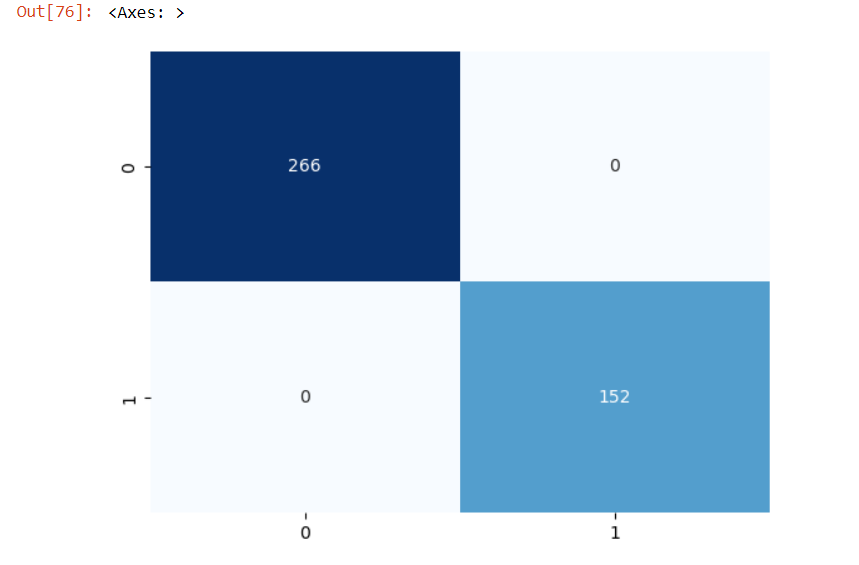
**print(f"Precision: {precision:.2f}")**

**print(f"Recall: {recall:.2f}")**

**print(f"F1-Score: {f1:.2f}")**

**print(f"Cross-Validation Accuracy: {cross\_val\_scores.mean():.2f}")**

**sns.heatmap(conf\_matrix, annot=True, fmt="d", cmap="Blues", cbar=False)**

****