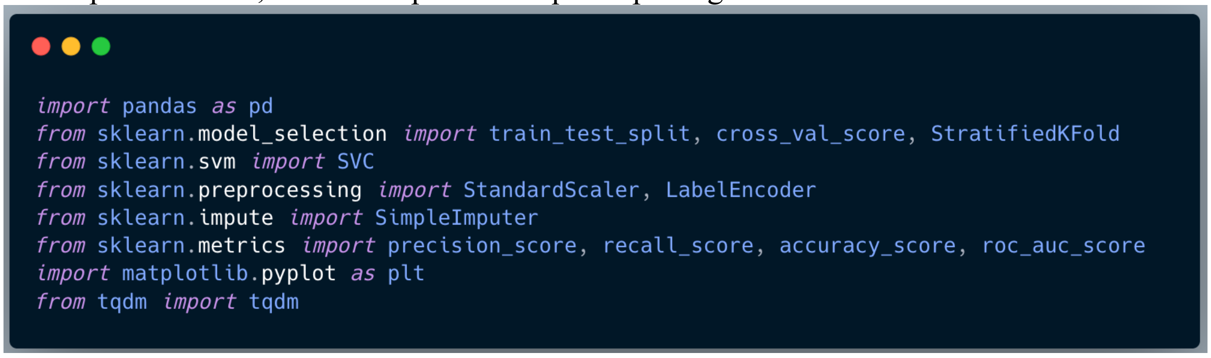
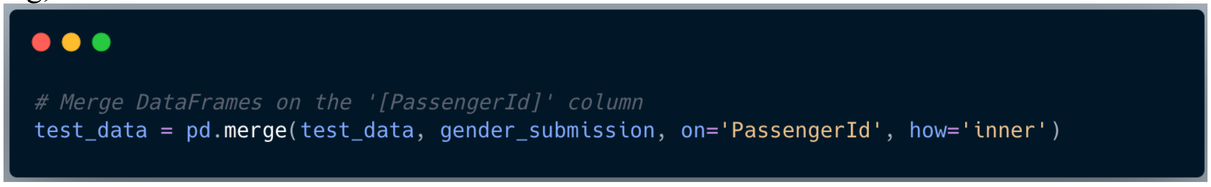
The Titanic dataset is a classic dataset that provides a comprehensive view of the Titanic's passengers. In this exercise, we will fully examine the dataset.
Then we proceed to data preprocessing. Several key preprocessing techniques, including feature normalization and standardization, are used to ensure uniformity in numerical features. In addition, binning is implemented to discretize continuous variables. Categorical feature encoding is used to convert non-numerical variables into a suitable format for machine learning algorithms and increases the interpretability of the model. Missing values management is also an important part of data preprocessing.
The main objective of this exercise is to use Support Vector Machine (SVM) models to predict outcomes. The regularization parameter (C) acts as a hyperparameter on the performance of the model. To evaluate the SVM model, different values of C are tested using 10-fold cross validation and 3-way hold out.
The evaluation criteria include accuracy, recall, precision and AUC. Through detailed analysis, we compare the performance of the model under hyperparameter variations.
For implementation, we first import the required packages.



```python
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score, StratifiedKFold
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.metrics import precision_score, recall_score, accuracy_score, roc_auc_score
import matplotlib.pyplot as plt
from tqdm import tqdm
```

After loading the data, we combine the gender_submission file, which contains the test data tag, with the test data.



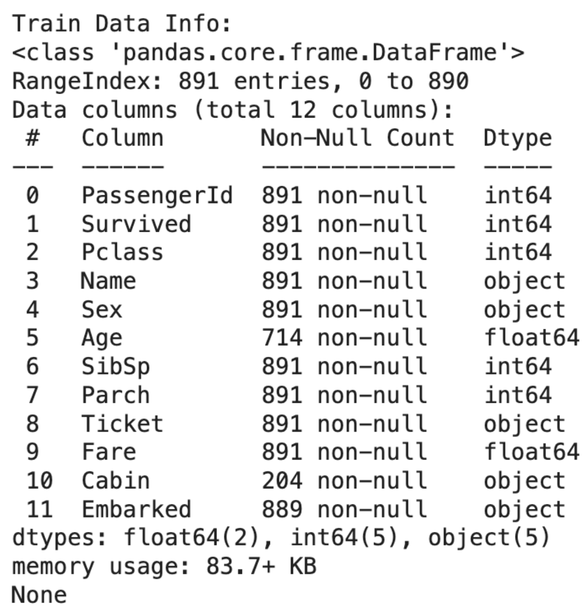
```python
# Merge DataFrames on the '[PassengerId]' column
test_data = pd.merge(test_data, gender_submission, on='PassengerId', how='inner')
```

The data are as follows. The dimensions of the training data are as follows.

```
(891, 12)
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Survived
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q	0
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S	1
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q	0
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S	0
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S	1

The general information of the educational data is as follows



```
Train 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
None
```

Data set analysis:
The dataset contains 891 entries (rows) and 12 columns.
Columns contain different data types: (`int64`), (`float64`) and object

- `PassengerId` a unique identifier for each passenger.
- ``Survived" is a binary variable that indicates whether a passenger survived (1) or not (0).
- "Pclass" the class of the ticket, with values of 1, 2, or 3 representing different passenger classes.
- Name of the passenger.
- 'Sex', the gender of the passenger

'Age'- the age of the passenger. There are missing values (NaN) in this column.

- `SibSp' The number of siblings or spouses on the Titanic.
- ``Parch" is the number of parents or children on the Titanic.

ticket`- Ticket number.

- ``Fare" is the fare paid for the ticket.

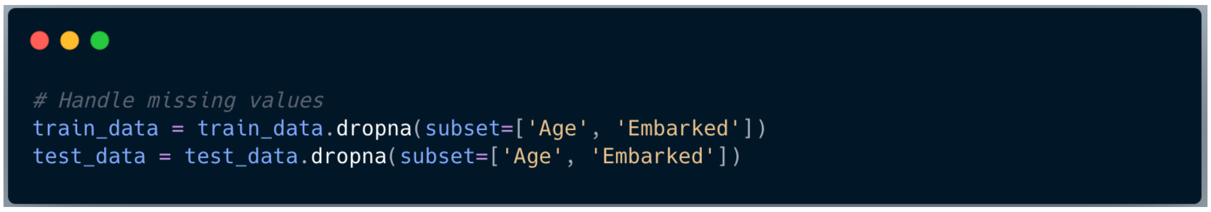
Cabin'- cabin number. There are missing values in this column.

- 'Embarked' boarding port. (C = Cherbourg, Q = Queenstown, S = Southampton). There are missing values in this column.

Missing Values:

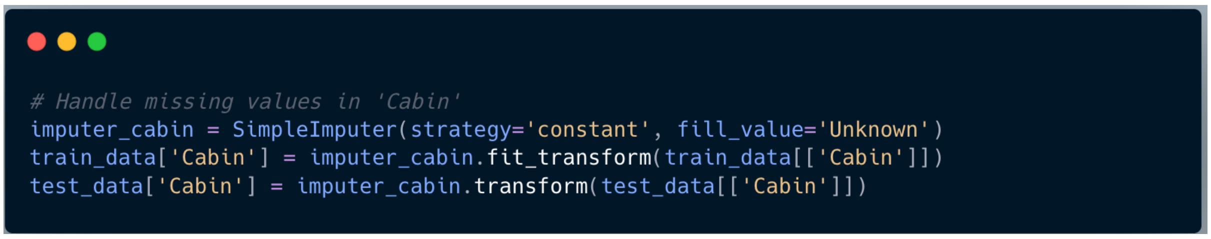
- The "Age" column has missing values, with 714 non-zero entries out of 891.
- The "Cabin" column has a significant number of missing values (more than 75% missing).
- The 'Embarked' column has some missing values.

The dataset contains both numerical and categorical features.
Features like "Sex" and "Embarked" may need to be coded for machine learning models.
Inserting or removing missing values in columns such as "Age" and "Embarked" may be necessary.
In the first step it removes rows from train_data and test_data where there are missing values (NaN) in the columns 'Age' or 'Embarked'. The 'dropna' method is used to remove rows that contain missing values in these particular columns.



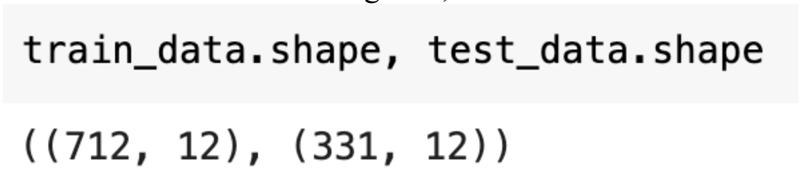
```python
# Handle missing values
train_data = train_data.dropna(subset=['Age', 'Embarked'])
test_data = test_data.dropna(subset=['Age', 'Embarked'])
```

The Cabin column also has many missing values and if we try to use the dropna method, we will lose a lot of data. We can use other methods to handle missing values.
This code uses scikit-learn's "SimpleImputer" to handle missing values in the "Cabin" column of the "train_data" and "test_data" DataFrames. Fills missing values with constant value 'Unknown.'



```python
# Handle missing values in 'Cabin'
imputer_cabin = SimpleImputer(strategy='constant', fill_value='Unknown')
train_data['Cabin'] = imputer_cabin.fit_transform(train_data[['Cabin']])
test_data['Cabin'] = imputer_cabin.transform(test_data[['Cabin']])
```

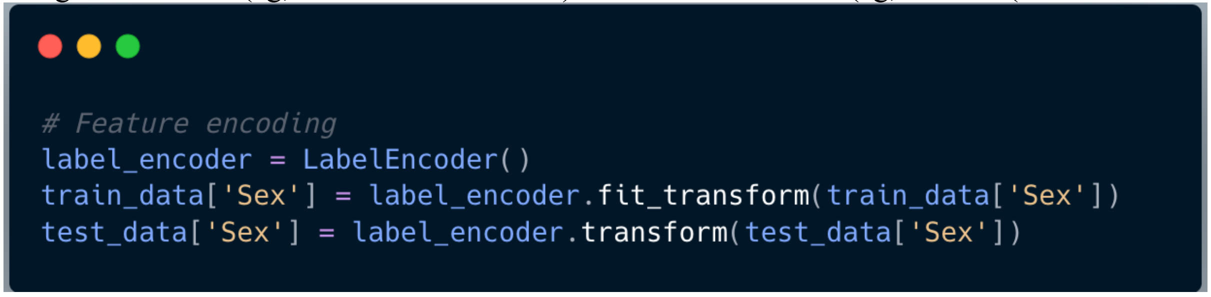
But for our machine learning model, it is better not to consider this column for training due to its many missing values.
After removing the rows that contain missing data, the dimensions of our data are as follows.



```
train_data.shape, test_data.shape
```

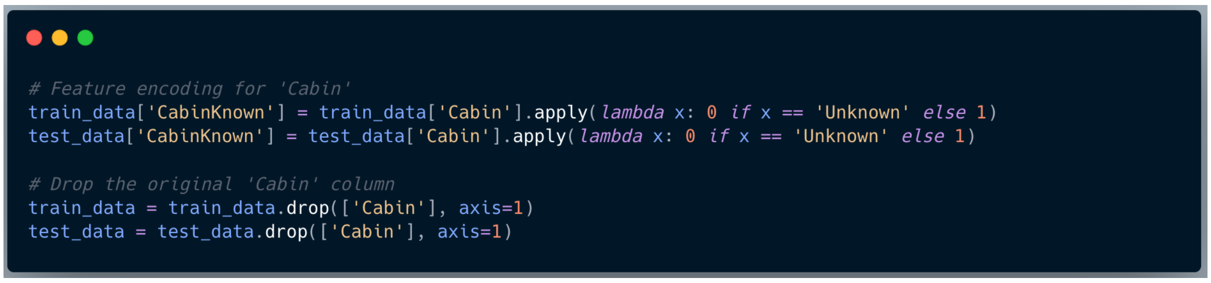
```
((712, 12), (331, 12))
```

We then use the LabelEncoder to encode the "sex" attribute. This encoding replaces categorical values (eg, "male" and "female") with numeric values (eg, 0 and 1.(



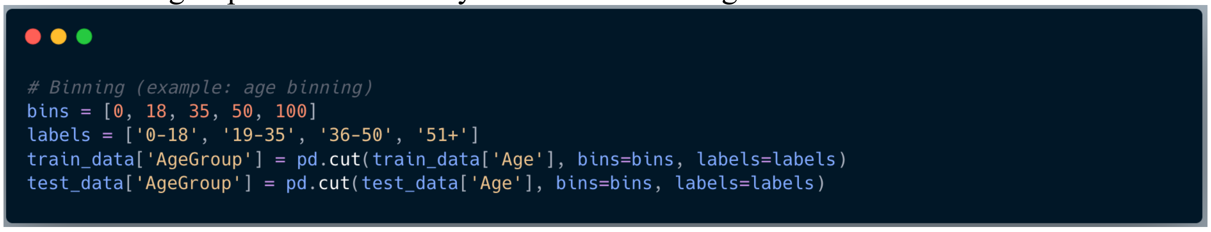
```python
# Feature encoding
label_encoder = LabelEncoder()
train_data['Sex'] = label_encoder.fit_transform(train_data['Sex'])
test_data['Sex'] = label_encoder.transform(test_data['Sex'])
```

This code adds a new binary attribute "CabinKnown" to the data. The "CabinKnown" attribute is created based on the "Cabin" column, assigned 0 if the cabin is labeled as "unkown" and 1 otherwise. After adding this new feature, the main column 'Cabin' is removed from both DataFrames.



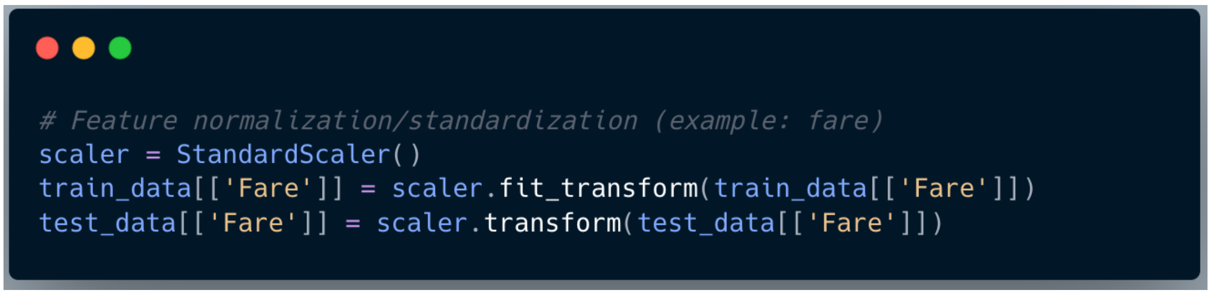
```python
# Feature encoding for 'Cabin'
train_data['CabinKnown'] = train_data['Cabin'].apply(lambda x: 0 if x == 'Unknown' else 1)
test_data['CabinKnown'] = test_data['Cabin'].apply(lambda x: 0 if x == 'Unknown' else 1)

# Drop the original 'Cabin' column
train_data = train_data.drop(['Cabin'], axis=1)
test_data = test_data.drop(['Cabin'], axis=1)
```

This code creates a new category attribute "AgeGroup" based on the "Age" column. It places the ages in specific ranges defined by the list of "bins" and assigns the corresponding labels from the list of "labels" to each bin. This is a method to separate the continuous variable "Age" into discrete groups for further analysis or model training.



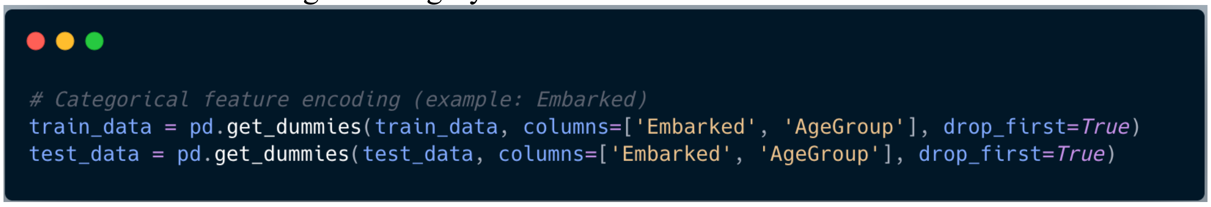
```
# Binning (example: age binning)
bins = [0, 18, 35, 50, 100]
labels = ['0-18', '19-35', '36-50', '51+']
train_data['AgeGroup'] = pd.cut(train_data['Age'], bins=bins, labels=labels)
test_data['AgeGroup'] = pd.cut(test_data['Age'], bins=bins, labels=labels)
```

This code standardizes the "Fare" feature in the data using "StandardScaler" from scikit-learn. Standardization involves converting the data to a mean of 0 and a standard deviation of 1. This process is used to further compare the Fare values and ensure that they are on the same scale.



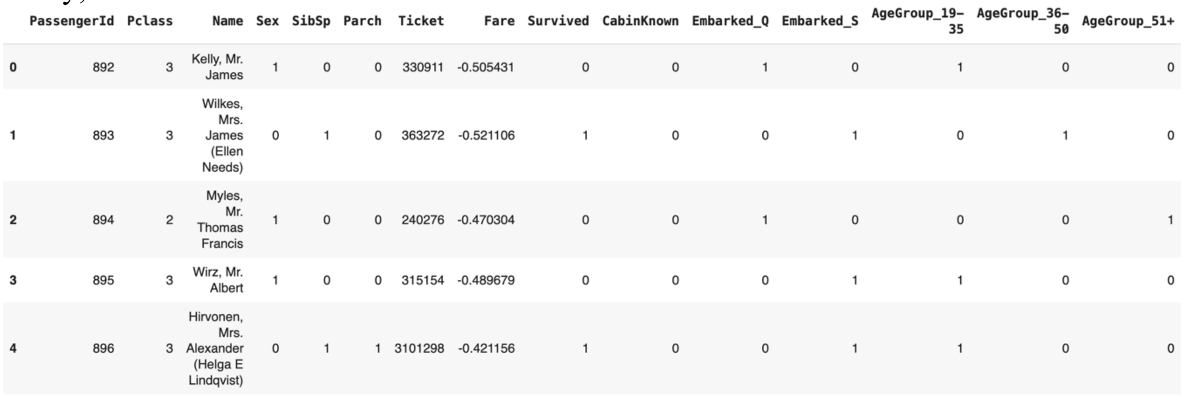
```
# Feature normalization/standardization (example: fare)
scaler = StandardScaler()
train_data[['Fare']] = scaler.fit_transform(train_data[['Fare']])
test_data[['Fare']] = scaler.transform(test_data[['Fare']])
```

This code encodes the categorical features in the 'Embarked' and 'AgeGroup' columns in both the training and test data. The "pd.get_dummies" function is used to convert categorical variables to dummy/index variables. The resulting dummy variables are added to the DataFrames and the original category columns are removed.



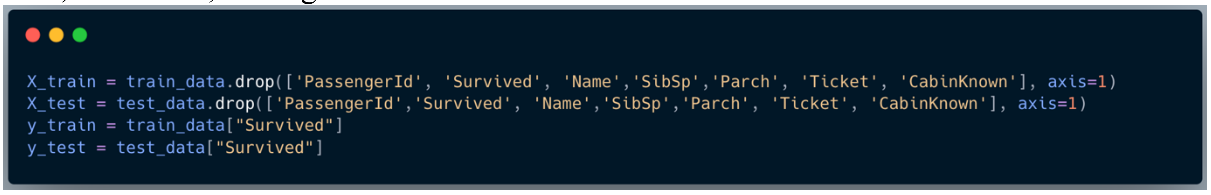
```
# Categorical feature encoding (example: Embarked)
train_data = pd.get_dummies(train_data, columns=['Embarked', 'AgeGroup'], drop_first=True)
test_data = pd.get_dummies(test_data, columns=['Embarked', 'AgeGroup'], drop_first=True)
```

Finally, the data will be as follows.



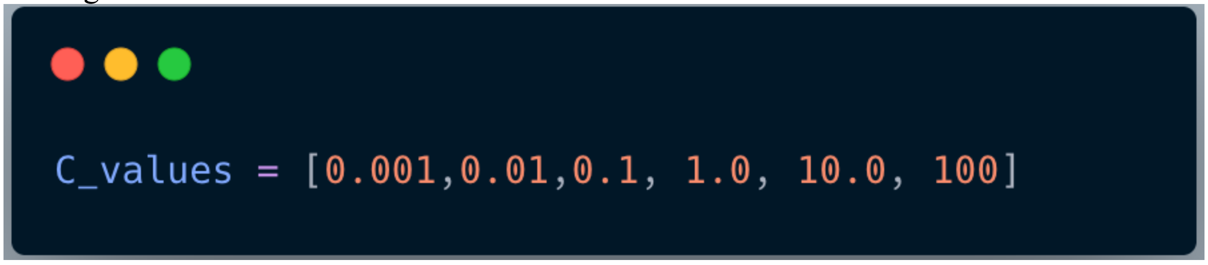
	PassengerId	Pclass	Name	Sex	SibSp	Parch	Ticket	Fare	Survived	CabinKnown	Embarked_Q	Embarked_S	AgeGroup_19-35	AgeGroup_36-50	AgeGroup_51+
0	892	3	Kelly, Mr. James	1	0	0	330911	-0.505431	0	0	1	0	1	0	0
1	893	3	Wilkes, Mrs. James (Ellen Needs)	0	1	0	363272	-0.521106	1	0	0	1	0	1	0
2	894	2	Myles, Mr. Thomas Francis	1	0	0	240276	-0.470304	0	0	1	0	0	0	1
3	895	3	Wirz, Mr. Albert	1	0	0	315154	-0.489679	0	0	0	1	1	0	0
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	0	1	1	3101298	-0.421156	1	0	0	1	1	0	0

Among the features provided to determine the survived label for passengers, Pclass, Sex, Fare, Embarked, and Age data can be useful.



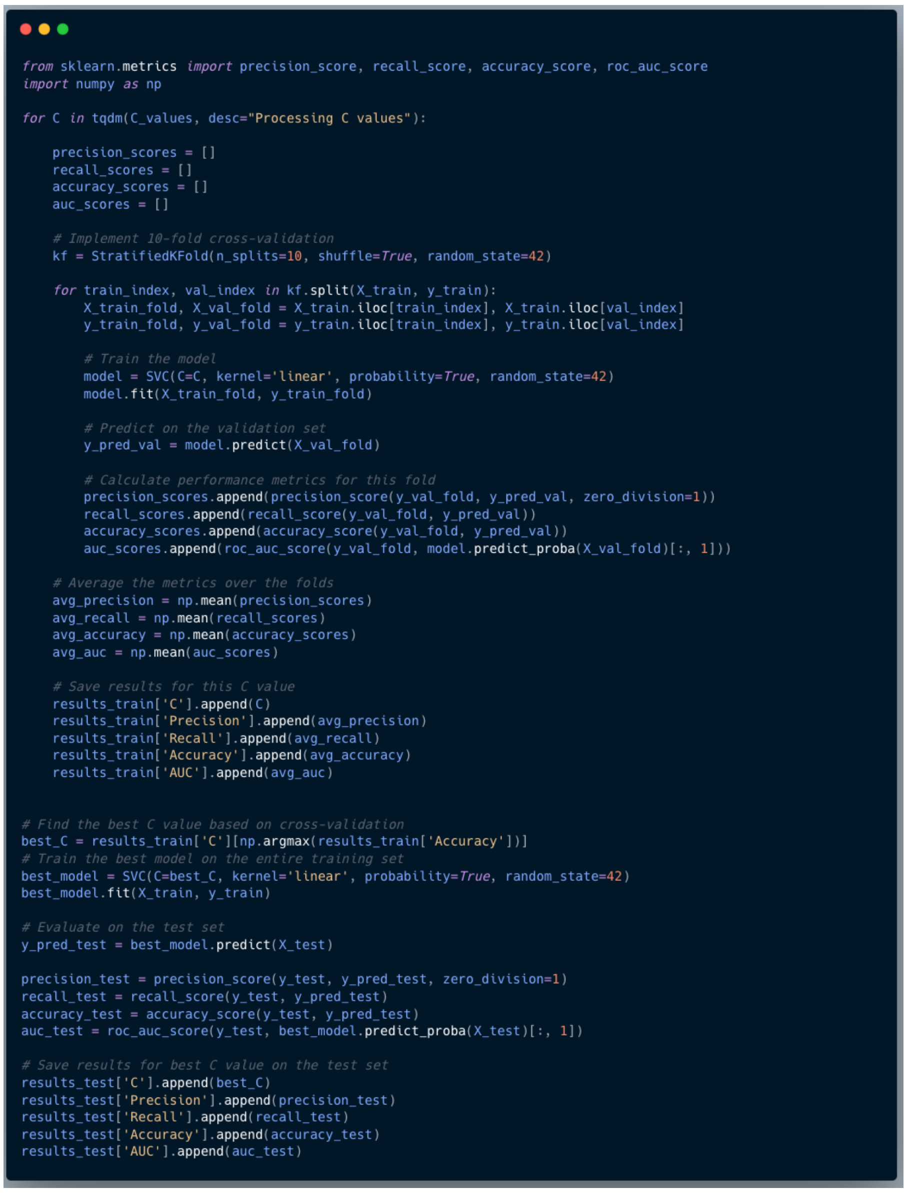
```
X_train = train_data.drop(['PassengerId', 'Survived', 'Name','SibSp','Parch', 'Ticket', 'CabinKnown'], axis=1)
X_test = test_data.drop(['PassengerId','Survived', 'Name','SibSp','Parch', 'Ticket', 'CabinKnown'], axis=1)
y_train = train_data["Survived"]
y_test = test_data["Survived"]
```

The variable "C_values" is a list that contains the different values of the setting parameter "C". These values are often used when training Support Vector Machine (SVM) models. The purpose of providing a range of values is to test different levels of regularization to find an optimal value that balances the fit between training data and generalization to new, unseen data. The values in the list represent different amounts of regularization, and you can adjust them based on the characteristics of your data and the performance of the SVM model during training and validation.



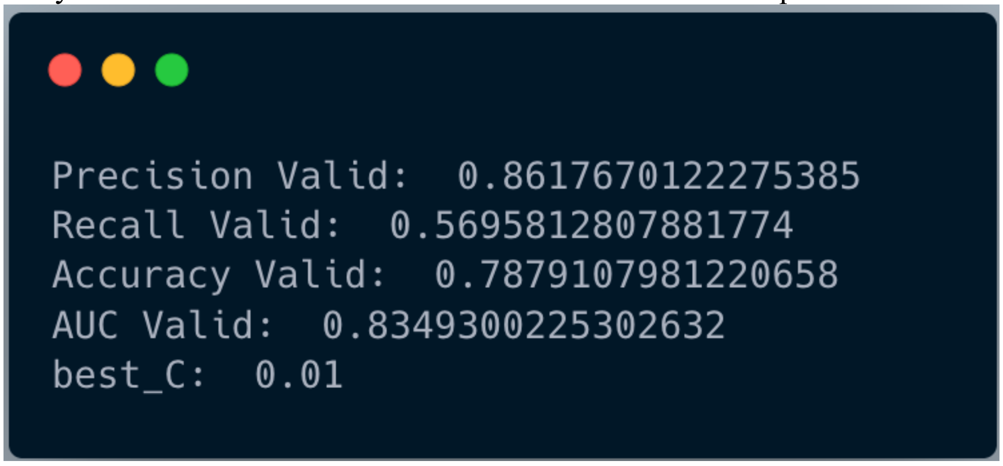
```
C_values = [0.001,0.01,0.1, 1.0, 10.0, 100]
```

This code uses a support vector classifier (SVC) with different regularization parameters (C) to find the best value of C through 10-fold cross-validation. It calculates precision, recall, accuracy, and AUC metrics for each value of C in the training set. The best value of C is chosen based on the highest accuracy. The final model with the best C is trained on the full training set and evaluated on a separate test set, and the results are saved for analysis.



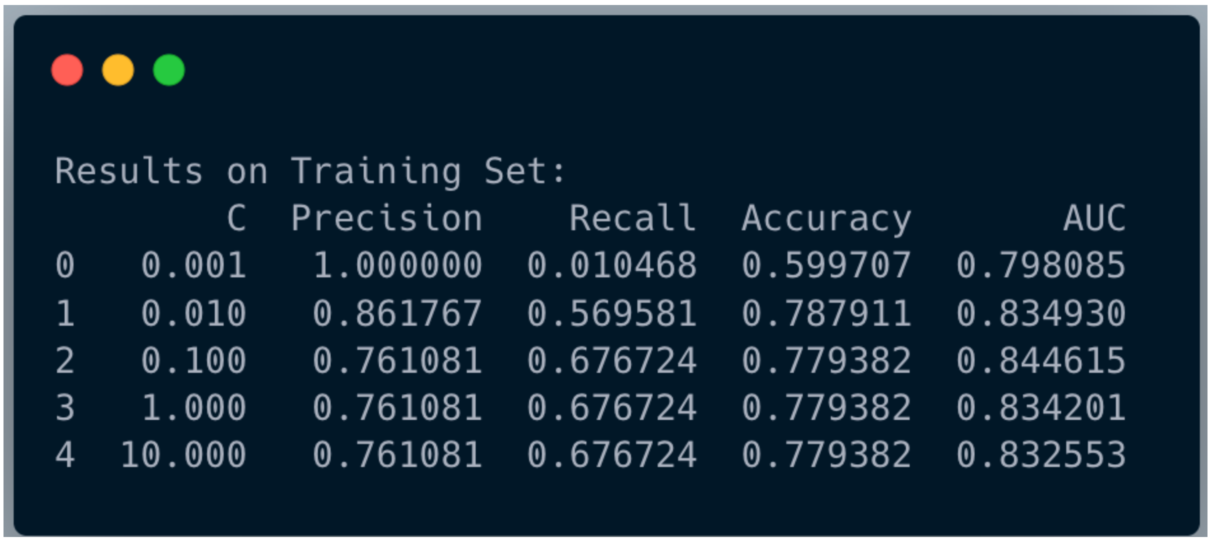
```python
from sklearn.metrics import precision_score, recall_score, accuracy_score, roc_auc_score
import numpy as np

for C in tqdm(C_values, desc="Processing C values"):

    precision_scores = []
    recall_scores = []
    accuracy_scores = []
    auc_scores = []

    # Implement 10-fold cross-validation
    kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

    for train_index, val_index in kf.split(X_train, y_train):
        X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.iloc[val_index]
        y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index]

        # Train the model
        model = SVC(C=C, kernel='linear', probability=True, random_state=42)
        model.fit(X_train_fold, y_train_fold)

        # Predict on the validation set
        y_pred_val = model.predict(X_val_fold)

        # Calculate performance metrics for this fold
        precision_scores.append(precision_score(y_val_fold, y_pred_val, zero_division=1))
        recall_scores.append(recall_score(y_val_fold, y_pred_val))
        accuracy_scores.append(accuracy_score(y_val_fold, y_pred_val))
        auc_scores.append(roc_auc_score(y_val_fold, model.predict_proba(X_val_fold)[:, 1]))

    # Average the metrics over the folds
    avg_precision = np.mean(precision_scores)
    avg_recall = np.mean(recall_scores)
    avg_accuracy = np.mean(accuracy_scores)
    avg_auc = np.mean(auc_scores)

    # Save results for this C value
    results_train['C'].append(C)
    results_train['Precision'].append(avg_precision)
    results_train['Recall'].append(avg_recall)
    results_train['Accuracy'].append(avg_accuracy)
    results_train['AUC'].append(avg_auc)


# Find the best C value based on cross-validation
best_C = results_train['C'][np.argmax(results_train['Accuracy'])]
# Train the best model on the entire training set
best_model = SVC(C=best_C, kernel='linear', probability=True, random_state=42)
best_model.fit(X_train, y_train)

# Evaluate on the test set
y_pred_test = best_model.predict(X_test)

precision_test = precision_score(y_test, y_pred_test, zero_division=1)
recall_test = recall_score(y_test, y_pred_test)
accuracy_test = accuracy_score(y_test, y_pred_test)
auc_test = roc_auc_score(y_test, best_model.predict_proba(X_test)[:, 1])

# Save results for best C value on the test set
results_test['C'].append(best_C)
results_test['Precision'].append(precision_test)
results_test['Recall'].append(recall_test)
results_test['Accuracy'].append(accuracy_test)
results_test['AUC'].append(auc_test)
```

The accuracy obtained for the best model for the validation data is equal to:



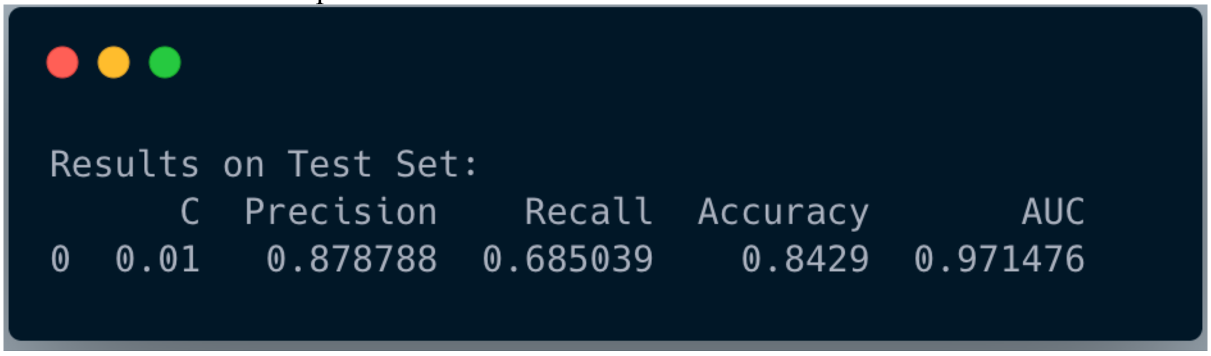
```
Precision Valid:  0.8617670122275385
Recall Valid:  0.5695812807881774
Accuracy Valid:  0.7879107981220658
AUC Valid:  0.8349300225302632
best_C:  0.01
```

The obtained values for hyperparameters C for 10-fold averaged are:

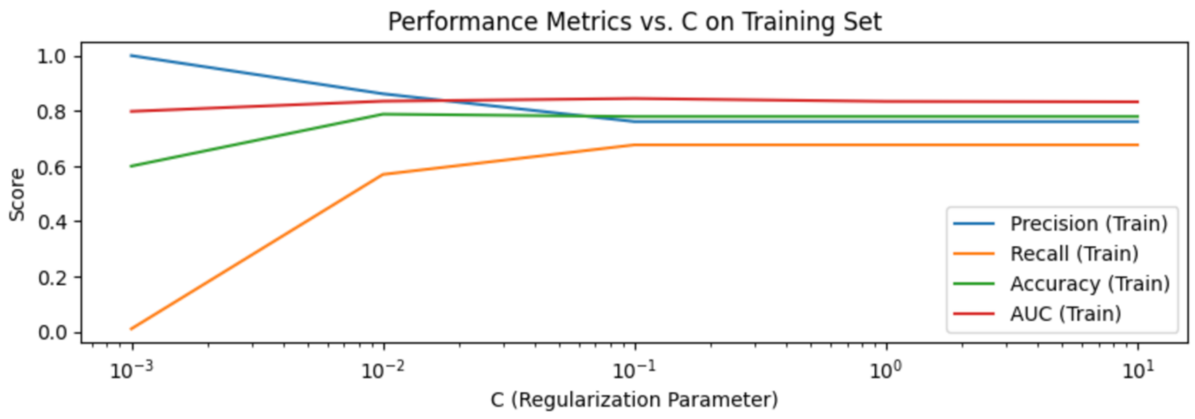


```
Results on Training Set:
        C  Precision    Recall  Accuracy       AUC
0   0.001   1.000000  0.010468  0.599707  0.798085
1   0.010   0.861767  0.569581  0.787911  0.834930
2   0.100   0.761081  0.676724  0.779382  0.844615
3   1.000   0.761081  0.676724  0.779382  0.834201
4  10.000   0.761081  0.676724  0.779382  0.832553
```

and for the test data is equal to:



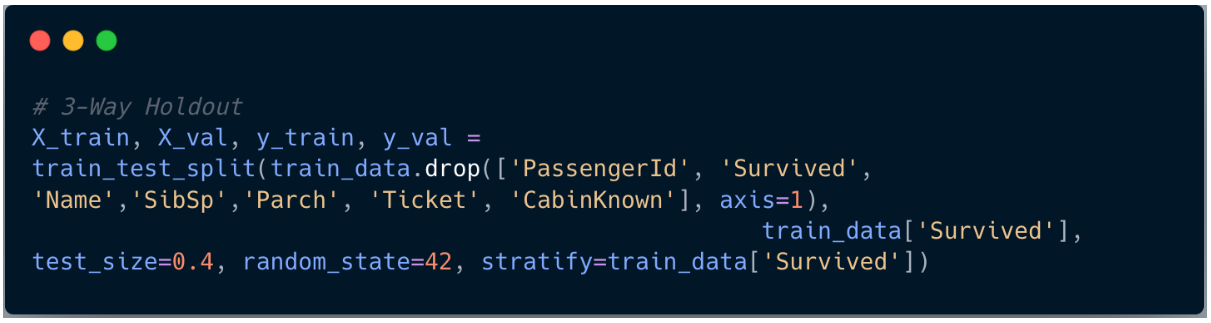
```
Results on Test Set:
      C  Precision    Recall  Accuracy       AUC
0  0.01   0.878788  0.685039    0.8429  0.971476
```



The above graph evaluates different metrics for each C parameter, which is the result of averaging the values obtained for 10-fold.

For the 3 way hold-out method:

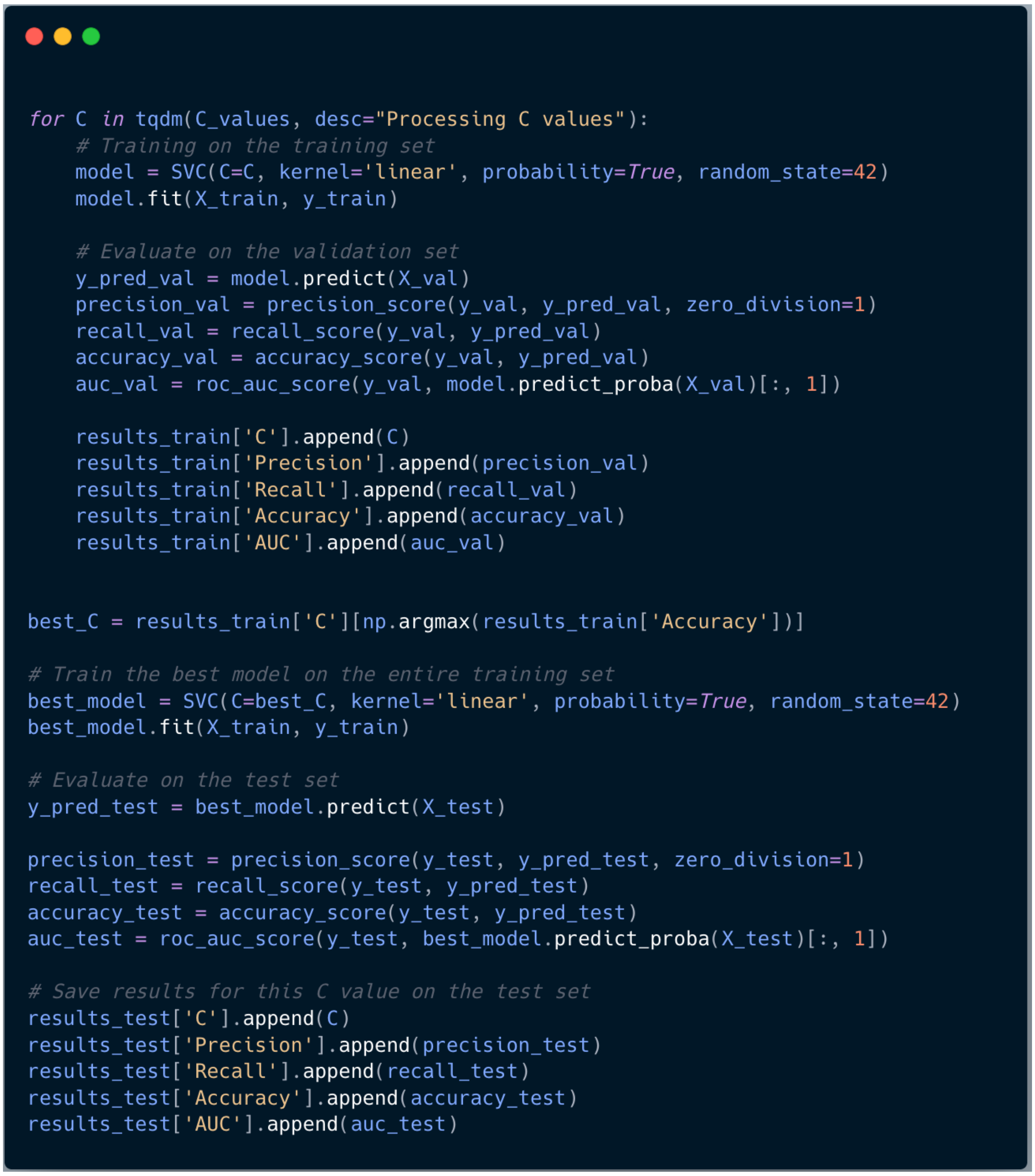
This code uses the "train_test_split" function to create a 3-way validation set. It divides the original training data into training (60%) and validation (40%) sets, excluding the specified columns ("PassengerId", "Survived", "Name", "SibSp", "Parch", "Ticket", " Splits CabinKnown'.') from properties. The partition is sorted by the 'Survived' column to preserve the class distribution.



```
# 3-Way Holdout
X_train, X_val, y_train, y_val =
train_test_split(train_data.drop(['PassengerId', 'Survived',
'Name','SibSp','Parch', 'Ticket', 'CabinKnown'], axis=1),
                                                    train_data['Survived'],
test_size=0.4, random_state=42, stratify=train_data['Survived'])
```

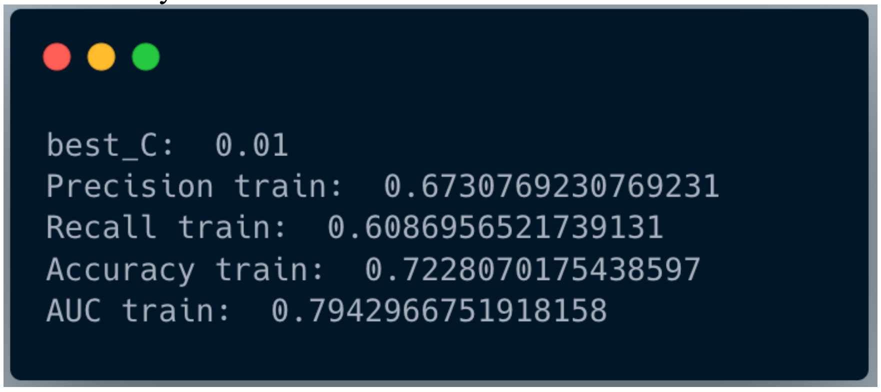
The code is iterated through different values of the tuning parameter (`C') using a support vector classifier (SVC) with a linear kernel. It trains the model on the training set ('X_train', 'y_train') and evaluates it on the validation set ('X_val', 'y_val'). The precision, recall, precision, and AUC scores for each value of "C" in the validation set are recorded in "train_results".

After iterating through all values of "C", it identifies the value of "C" that maximizes the accuracy in the validation set ("best_C"). It then trains the best model using this "C" value on the entire training set and evaluates its performance on the test set ("X_test", "y_test"). The scores for accuracy, recall, precision and AUC in the test set are recorded in "results_test".



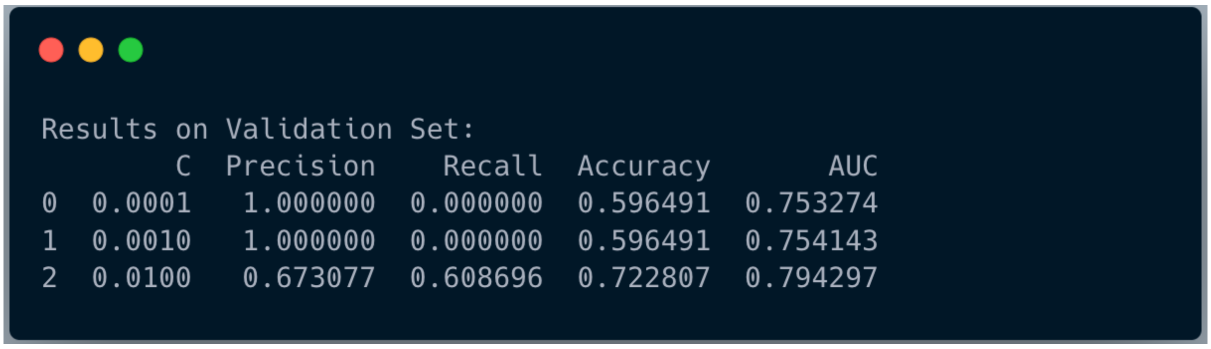
```python
for C in tqdm(C_values, desc="Processing C values"):
    # Training on the training set
    model = SVC(C=C, kernel='linear', probability=True, random_state=42)
    model.fit(X_train, y_train)

    # Evaluate on the validation set
    y_pred_val = model.predict(X_val)
    precision_val = precision_score(y_val, y_pred_val, zero_division=1)
    recall_val = recall_score(y_val, y_pred_val)
    accuracy_val = accuracy_score(y_val, y_pred_val)
    auc_val = roc_auc_score(y_val, model.predict_proba(X_val)[:, 1])

    results_train['C'].append(C)
    results_train['Precision'].append(precision_val)
    results_train['Recall'].append(recall_val)
    results_train['Accuracy'].append(accuracy_val)
    results_train['AUC'].append(auc_val)


best_C = results_train['C'][np.argmax(results_train['Accuracy'])]

# Train the best model on the entire training set
best_model = SVC(C=best_C, kernel='linear', probability=True, random_state=42)
best_model.fit(X_train, y_train)

# Evaluate on the test set
y_pred_test = best_model.predict(X_test)

precision_test = precision_score(y_test, y_pred_test, zero_division=1)
recall_test = recall_score(y_test, y_pred_test)
accuracy_test = accuracy_score(y_test, y_pred_test)
auc_test = roc_auc_score(y_test, best_model.predict_proba(X_test)[:, 1])

# Save results for this C value on the test set
results_test['C'].append(C)
results_test['Precision'].append(precision_test)
results_test['Recall'].append(recall_test)
results_test['Accuracy'].append(accuracy_test)
results_test['AUC'].append(auc_test)
```

The accuracy obtained for the best model for the validation data is equal to:



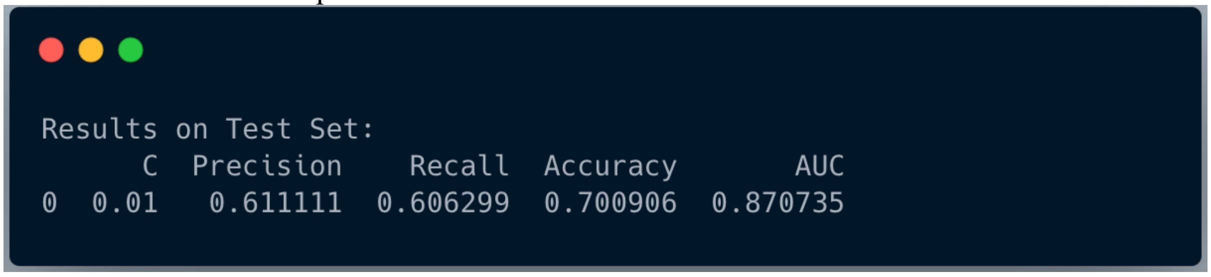
```
best_C:  0.01
Precision train:  0.6730769230769231
Recall train:  0.6086956521739131
Accuracy train:  0.7228070175438597
AUC train:  0.7942966751918158
```

The obtained values for C hyperparameters for 3 way hold-out are equal to:

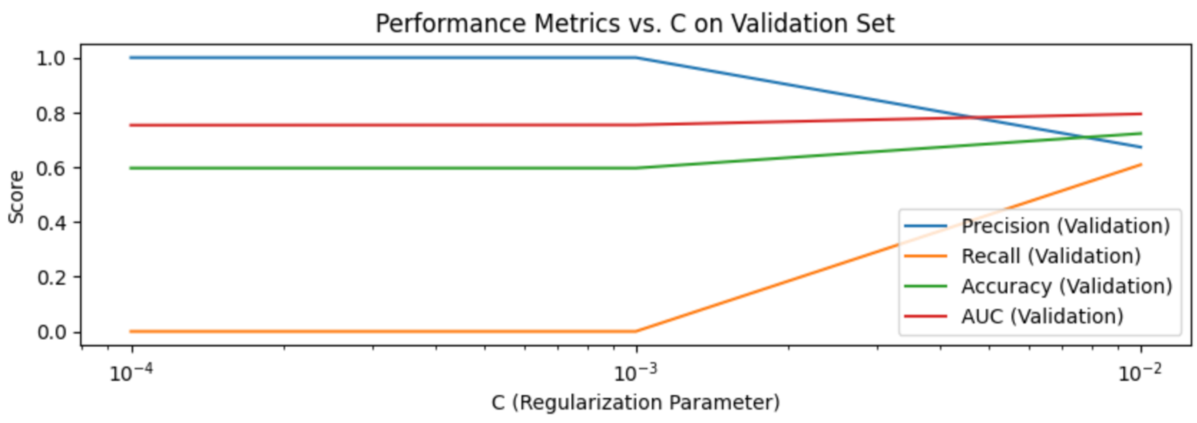


```
Results on Validation Set:
        C  Precision    Recall  Accuracy       AUC
0  0.0001   1.000000  0.000000  0.596491  0.753274
1  0.0010   1.000000  0.000000  0.596491  0.754143
2  0.0100   0.673077  0.608696  0.722807  0.794297
```

and for the test data is equal to:



```
Results on Test Set:
      C  Precision    Recall  Accuracy       AUC
0  0.01   0.611111  0.606299  0.700906  0.870735
```



The above graph evaluates different metrics for each C parameter, which is the result of the values obtained for the 3-way hold out method.

The accuracy on the test data is more than the training data, and according to the analysis, it can be concluded that the test data is easier than the training data.