```python
# Importing necessary libraries
import pandas as pd

# Load the dataset from the provided path
file_path = 'loan_data.csv'

# Reading the dataset into a pandas dataframe
loan_data = pd.read_csv(file_path)

# Displaying basic information and first few rows of the dataset to
understand its structure
loan_data.info(), loan_data.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 27 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   loan_amnt             396030 non-null  float64
 1   term                  396030 non-null  object 
 2   int_rate              396030 non-null  float64
 3   installment           396030 non-null  float64
 4   grade                 396030 non-null  object 
 5   sub_grade             396030 non-null  object 
 6   emp_title             373103 non-null  object 
 7   emp_length            377729 non-null  object 
 8   home_ownership        396030 non-null  object 
 9   annual_inc            396030 non-null  float64
 10  verification_status   396030 non-null  object 
 11  issue_d               396030 non-null  object 
 12  loan_status           396030 non-null  object 
 13  purpose               396030 non-null  object 
 14  title                 394275 non-null  object 
 15  dti                   396030 non-null  float64
 16  earliest_cr_line      396030 non-null  object 
 17  open_acc              396030 non-null  float64
 18  pub_rec               396030 non-null  float64
 19  revol_bal             396030 non-null  float64
 20  revol_util            395754 non-null  float64
 21  total_acc             396030 non-null  float64
 22  initial_list_status   396030 non-null  object 
 23  application_type      396030 non-null  object 
 24  mort_acc              358235 non-null  float64
 25  pub_rec_bankruptcies  395495 non-null  float64
 26  address               396030 non-null  object 
dtypes: float64(12), object(15)
memory usage: 81.6+ MB

(None,
    loan_amnt        term  int_rate  installment grade sub_grade  \
```

```
0    10000.0   36 months     11.44       329.48     B        B4
1     8000.0   36 months     11.99       265.68     B        B5
2    15600.0   36 months     10.49       506.97     B        B3
3     7200.0   36 months      6.49       220.65     A        A2
4    24375.0   60 months     17.27       609.33     C        C5

                  emp_title emp_length home_ownership  annual_inc  ...
\
0                 Marketing  10+ years           RENT    117000.0  ...

1           Credit analyst     4 years       MORTGAGE     65000.0  ...

2              Statistician   < 1 year           RENT     43057.0  ...

3           Client Advocate    6 years           RENT     54000.0  ...

4  Destiny Management Inc.    9 years       MORTGAGE     55000.0  ...


   open_acc pub_rec revol_bal revol_util total_acc
initial_list_status  \
0     16.0     0.0   36369.0       41.8      25.0
w
1     17.0     0.0   20131.0       53.3      27.0
f
2     13.0     0.0   11987.0       92.2      26.0
f
3      6.0     0.0    5472.0       21.5      13.0
f
4     13.0     0.0   24584.0       69.8      43.0
f

   application_type  mort_acc  pub_rec_bankruptcies  \
0       INDIVIDUAL       0.0                   0.0
1       INDIVIDUAL       3.0                   0.0
2       INDIVIDUAL       0.0                   0.0
3       INDIVIDUAL       0.0                   0.0
4       INDIVIDUAL       1.0                   0.0

                                            address
0     0174 Michelle Gateway\nMendozaberg, OK 22690
1  1076 Carney Fort Apt. 347\nLoganmouth, SD 05113
2  87025 Mark Dale Apt. 269\nNew Sabrina, WV 05113
3            823 Reid Ford\nDelacruzside, MA 00813
4             679 Luna Roads\nGreggshire, VA 11650

[5 rows x 27 columns])
```

The dataset contains 396,030 entries and 27 columns, including both numerical and categorical features. The dependent variable for classification will be the loan_status column.

Handling missing values - filling missing numerical columns with the median and categorical columns with mode

```
loan_data['emp_title'].fillna(loan_data['emp_title'].mode()[0],
inplace=True)
loan_data['emp_length'].fillna(loan_data['emp_length'].mode()[0],
inplace=True)
loan_data['title'].fillna(loan_data['title'].mode()[0], inplace=True)
loan_data['revol_util'].fillna(loan_data['revol_util'].median(),
inplace=True)
loan_data['mort_acc'].fillna(loan_data['mort_acc'].median(),
inplace=True)
loan_data['pub_rec_bankruptcies'].fillna(loan_data['pub_rec_bankruptci
es'].median(), inplace=True)
```

Encoding categorical columns using Label Encoding for simplicity (Loan_status and other categories)

```
# List of columns to encode
#cat_columns = ['term', 'grade', 'sub_grade', 'emp_length',
'home_ownership',
               #'verification_status', 'loan_status', 'purpose',
'title', 'initial_list_status',
               #'application_type']
```

Separate categorical and numerical columns based on dtype

```
cat_columns = loan_data.select_dtypes(include=['object',
'category']).columns.tolist()
num_columns = loan_data.select_dtypes(include=['int64',
'float64']).columns.tolist()
cat_columns # displaying columns with categorical values
```

```
['term',
 'grade',
 'sub_grade',
 'emp_title',
 'emp_length',
 'home_ownership',
 'verification_status',
 'issue_d',
 'loan_status',
 'purpose',
 'title',
```

```
 'earliest_cr_line',
 'initial_list_status',
 'application_type',
 'address']
```

```
num_columns # displaying columns with numerical values
```

```
['loan_amnt',
 'int_rate',
 'installment',
 'annual_inc',
 'dti',
 'open_acc',
 'pub_rec',
 'revol_bal',
 'revol_util',
 'total_acc',
 'mort_acc',
 'pub_rec_bankruptcies']
```

Initialize the label encoder

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

# Applying label encoding to each of the categorical columns
for col in cat_columns:
    loan_data[col] = le.fit_transform(loan_data[col])

# Now, the dataset should be preprocessed and ready for feature
selection and modeling
loan_data.info(), loan_data.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 27 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   loan_amnt             396030 non-null  float64
 1   term                  396030 non-null  int64
 2   int_rate              396030 non-null  float64
 3   installment           396030 non-null  float64
 4   grade                 396030 non-null  int64
 5   sub_grade             396030 non-null  int64
 6   emp_title             396030 non-null  int64
 7   emp_length            396030 non-null  int64
 8   home_ownership        396030 non-null  int64
 9   annual_inc            396030 non-null  float64
 10  verification_status   396030 non-null  int64
```

```
 11  issue_d               396030 non-null  int64
 12  loan_status           396030 non-null  int64
 13  purpose               396030 non-null  int64
 14  title                 396030 non-null  int64
 15  dti                   396030 non-null  float64
 16  earliest_cr_line      396030 non-null  int64
 17  open_acc              396030 non-null  float64
 18  pub_rec               396030 non-null  float64
 19  revol_bal             396030 non-null  float64
 20  revol_util            396030 non-null  float64
 21  total_acc             396030 non-null  float64
 22  initial_list_status   396030 non-null  int64
 23  application_type      396030 non-null  int64
 24  mort_acc              396030 non-null  float64
 25  pub_rec_bankruptcies  396030 non-null  float64
 26  address               396030 non-null  int64
dtypes: float64(12), int64(15)
memory usage: 81.6 MB

(None,
    loan_amnt  term  int_rate  installment  grade  sub_grade
emp_title  \
 0    10000.0     0     11.44       329.48      1          8
80956
 1     8000.0     0     11.99       265.68      1          9
33317
 2    15600.0     0     10.49       506.97      1          7
127182
 3     7200.0     0      6.49       220.65      0          1
27760
 4    24375.0     1     17.27       609.33      2         14
38300

    emp_length  home_ownership  annual_inc  ...  open_acc  pub_rec
revol_bal  \
 0           1               5    117000.0  ...      16.0      0.0
36369.0
 1           4               1     65000.0  ...      17.0      0.0
20131.0
 2          10               5     43057.0  ...      13.0      0.0
11987.0
 3           6               5     54000.0  ...       6.0      0.0
5472.0
 4           9               1     55000.0  ...      13.0      0.0
24584.0

    revol_util  total_acc  initial_list_status  application_type
mort_acc  \
 0        41.8       25.0                    1                 1
0.0
```

```
 1        53.3       27.0                       0                 1
3.0
 2        92.2       26.0                       0                 1
0.0
 3        21.5       13.0                       0                 1
0.0
 4        69.8       43.0                       0                 1
1.0

   pub_rec_bankruptcies  address
 0                  0.0     6206
 1                  0.0    38135
 2                  0.0   307942
 3                  0.0   291181
 4                  0.0   240127

 [5 rows x 27 columns])
```

The preprocessing step is complete, and the dataset is now ready for feature selection and modeling. Here's what we have done:

```
Missing values were filled with either the mode (for categorical
columns) or the median (for numerical columns).
Categorical columns were label-encoded to numerical values for
compatibility with machine learning algorithms.
```

Importing necessary libraries for model building and evaluation

```python
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
```

Defining independent variables (X) and dependent variable (y)

```python
X = loan_data.drop(columns=['loan_status', 'emp_title', 'issue_d',
'earliest_cr_line', 'address'])  # Dropping non-relevant columns
y = loan_data['loan_status']
```

We'll build a Decision Tree Classifier using the loan_status as the target variable and the remaining columns as predictors.

Splitting the data into training (80%) and testing (20%) sets

```python
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

Initializing and training the Decision Tree Classifier

```
tree_clf = DecisionTreeClassifier(random_state=42)
tree_clf.fit(X_train, y_train)

DecisionTreeClassifier(random_state=42)
```

Making predictions on the test set

```
y_pred = tree_clf.predict(X_test)
```

Evaluating model performance

```
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
```

Getting feature importance from the trained model

```
feature_importances = tree_clf.feature_importances_

accuracy

0.7041890765851072

conf_matrix

array([[ 4592, 10985],
       [12445, 51184]])

feature_importances

array([0.04507793, 0.0049767 , 0.0548494 , 0.07461699, 0.00327091,
       0.08698289, 0.04161787, 0.01281924, 0.08427486, 0.01700044,
       0.02013734, 0.05144465, 0.11388186, 0.05735229, 0.00999808,
       0.10329507, 0.09884682, 0.07401184, 0.01049358, 0.00025082,
       0.02847054, 0.00632988])

X.columns

Index(['loan_amnt', 'term', 'int_rate', 'installment', 'grade',
'sub_grade',
       'emp_length', 'home_ownership', 'annual_inc',
'verification_status',
       'purpose', 'title', 'dti', 'open_acc', 'pub_rec', 'revol_bal',
       'revol_util', 'total_acc', 'initial_list_status',
'application_type',
       'mort_acc', 'pub_rec_bankruptcies'],
      dtype='object')

# Create a DataFrame for feature importances
feature_importance= pd.DataFrame({
    'Feature': X.columns,
```

```python
    'Importance': feature_importances
})
feature_importance
```

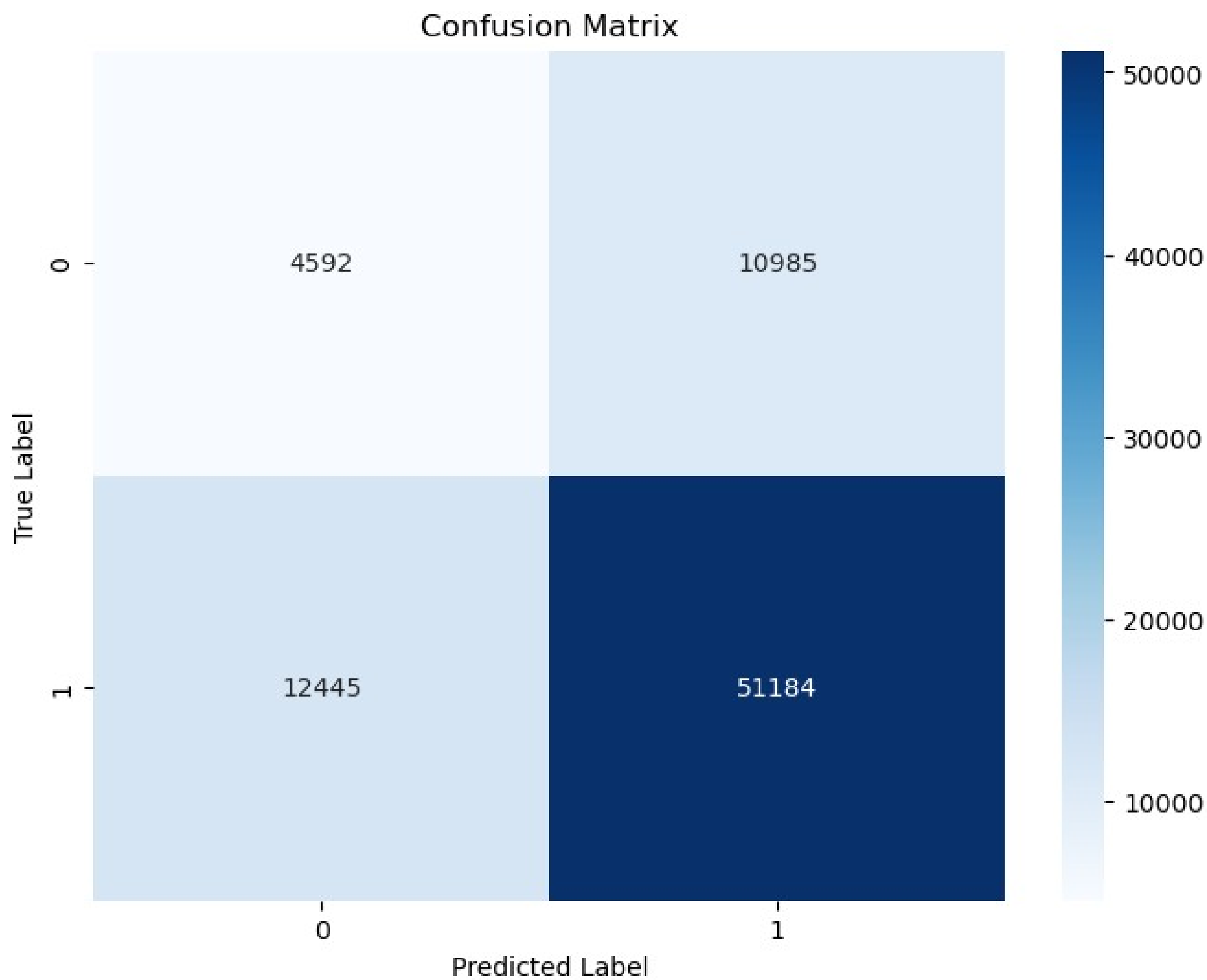
	Feature	Importance
0	loan_amnt	0.045078
1	term	0.004977
2	int_rate	0.054849
3	installment	0.074617
4	grade	0.003271
5	sub_grade	0.086983
6	emp_length	0.041618
7	home_ownership	0.012819
8	annual_inc	0.084275
9	verification_status	0.017000
10	purpose	0.020137
11	title	0.051445
12	dti	0.113882
13	open_acc	0.057352
14	pub_rec	0.009998
15	revol_bal	0.103295
16	revol_util	0.098847
17	total_acc	0.074012
18	initial_list_status	0.010494
19	application_type	0.000251
20	mort_acc	0.028471
21	pub_rec_bankruptcies	0.006330

```python
# Sort the DataFrame by importance in descending order
feature_importance = feature_importance.sort_values(by='Importance',
ascending=False).reset_index(drop=True)
feature_importance
```

	Feature	Importance
0	dti	0.113882
1	revol_bal	0.103295
2	revol_util	0.098847
3	sub_grade	0.086983
4	annual_inc	0.084275
5	installment	0.074617
6	total_acc	0.074012
7	open_acc	0.057352
8	int_rate	0.054849
9	title	0.051445
10	loan_amnt	0.045078
11	emp_length	0.041618
12	mort_acc	0.028471
13	purpose	0.020137
14	verification_status	0.017000
15	home_ownership	0.012819

16	initial_list_status	0.010494
17	pub_rec	0.009998
18	pub_rec_bankruptcies	0.006330
19	term	0.004977
20	grade	0.003271
21	application_type	0.000251

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues")
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```



Results from Decision Tree Classification:

Model Accuracy: The Decision Tree model achieved an accuracy of 70.42% on the test data, which indicates that the model is correctly predicting the loan_status around 70% of the time.

Confusion Matrix: The confusion matrix indicates the number of correct and incorrect predictions for each class:

```
True Positives (TP): 51,184
True Negatives (TN): 4,592
False Positives (FP): 10,985
False Negatives (FN): 12,445
```

Feature Importance: The Decision Tree model also provides the importance of each feature in determining the loan status.

These features have the most influence in predicting loan status

Tune this model or try other machine learning algorithms to improve the results?

To tune the Decision Tree model, we'll focus on optimizing key hyperparameters to improve performance. The main hyperparameters to tune for Decision Trees are:

```
max_depth: Maximum depth of the tree (controls overfitting).
min_samples_split: Minimum number of samples required to split an
internal node.
min_samples_leaf: Minimum number of samples required to be at a leaf
node.
criterion: The function to measure the quality of a split (e.g.,
"gini" or "entropy").
```

We will use Grid Search Cross-Validation to explore different values of these hyperparameters and find the optimal combination.

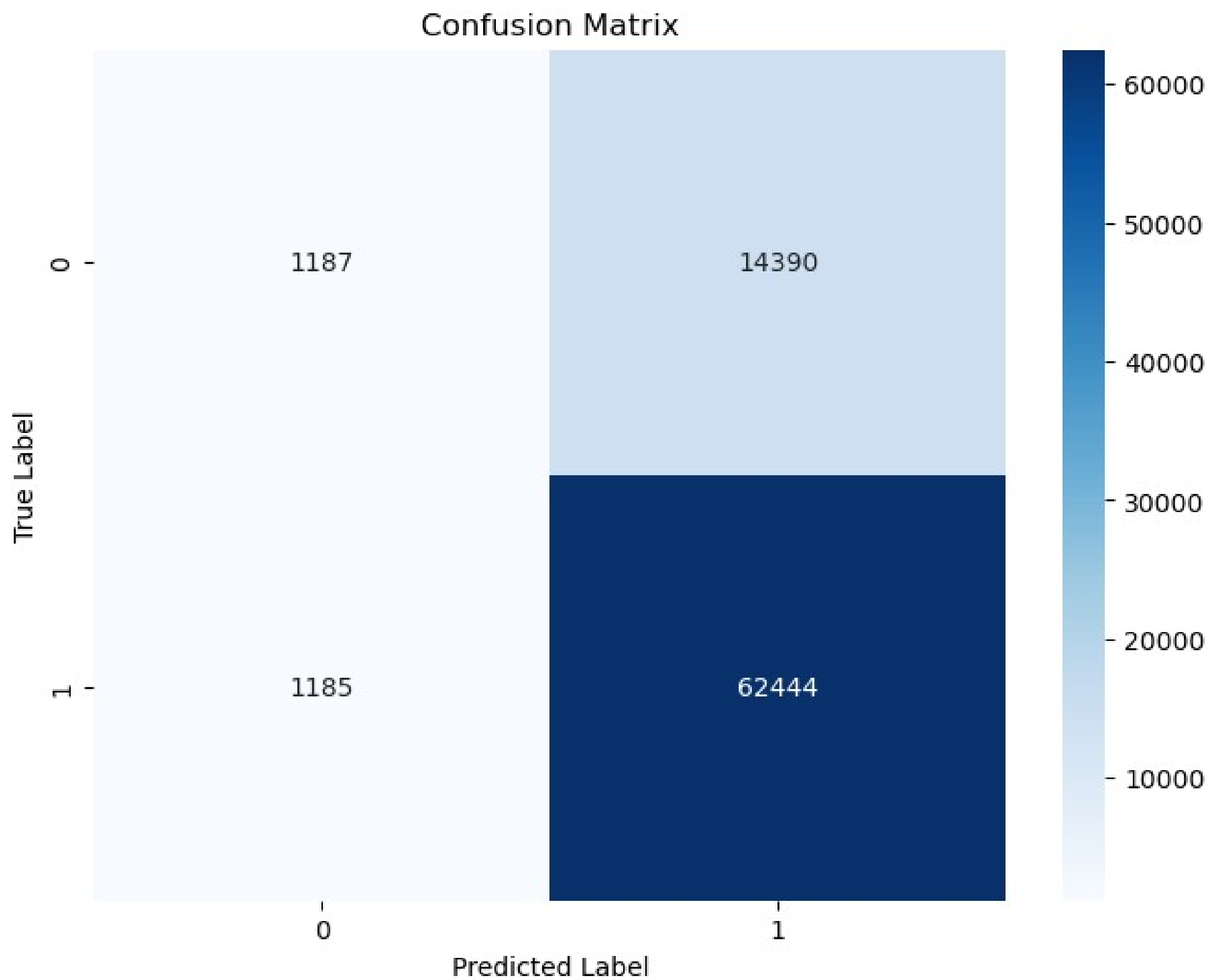
```
# Importing the necessary library for hyperparameter tuning
from sklearn.model_selection import GridSearchCV

# Setting up the hyperparameters for tuning
param_grid = {
    'max_depth': [10, 20, 30, None],
    'min_samples_split': [2, 10, 20],
    'min_samples_leaf': [1, 5, 10],
    'criterion': ['gini', 'entropy']
}

# Setting up the GridSearchCV to find the best parameters
grid_search = GridSearchCV(estimator=tree_clf, param_grid=param_grid,
```

```
                           cv=3, scoring='accuracy', n_jobs=-1,
verbose=1)

# Fitting the grid search to the data
grid_search.fit(X_train, y_train)

# Get the best parameters from the grid search
best_params = grid_search.best_params_

# Train a new Decision Tree with the best parameters
best_tree_clf = grid_search.best_estimator_

# Evaluating the optimized model
y_pred_tuned = best_tree_clf.predict(X_test)
tuned_accuracy = accuracy_score(y_test, y_pred_tuned)
tuned_conf_matrix = confusion_matrix(y_test, y_pred_tuned)

best_params, tuned_accuracy, tuned_conf_matrix
```

```
Fitting 3 folds for each of 72 candidates, totalling 216 fits
```

```
({'criterion': 'entropy',
  'max_depth': 10,
  'min_samples_leaf': 5,
  'min_samples_split': 20},
 0.8033608565007702,
 array([[ 1187, 14390],
        [ 1185, 62444]]))
```

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8, 6))
sns.heatmap(tuned_conf_matrix, annot=True, fmt="d", cmap="Blues")
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```



Components Explained GridSearchCV:

This is the main class from Scikit-learn used to perform grid search with cross-validation. estimator:

estimator=tree_clf: This specifies the machine learning model you want to optimize. In this case, tree_clf is presumably a Decision Tree Classifier (or any other model). This model will be trained with different combinations of hyperparameters as defined in param_grid. param_grid:

param_grid=param_grid: This parameter defines a dictionary where the keys are the hyperparameters of the model, and the values are lists of settings to be tested.

cv:

cv=3: This specifies the number of cross-validation folds. Here, the dataset will be split into 3 parts (folds), and the model will be trained on 2 parts and validated on 1 part in each iteration. This process will repeat for all combinations of parameters, allowing for a more robust evaluation of model performance. scoring:

scoring='accuracy': This specifies the metric to evaluate the performance of the model during cross-validation. In this case, the model will be evaluated based on its accuracy, which is the proportion of correctly predicted instances to the total instances. n_jobs:

n_jobs=-1: This parameter controls the number of jobs to run in parallel. Setting n_jobs=-1 means that all available cores will be used, which can significantly speed up the search process. verbose:

verbose=1: This controls the level of detail displayed during the fitting process. A value of 1 means that progress messages will be printed, allowing you to monitor how the grid search is progressing. Higher values (e.g., 2) provide more detailed messages. Summary of GridSearchCV Process Exhaustive Search: GridSearchCV will perform an exhaustive search over all combinations of the parameters specified in param_grid.

Cross-Validation: For each combination of parameters, the model will be evaluated using cross-validation. The average accuracy score across all folds will be calculated.

Best Parameters: After evaluating all combinations, GridSearchCV will identify the combination of hyperparameters that resulted in the highest average accuracy.

Final Model: The model is then trained using the best-found parameters on the entire training dataset.