Lab 5: ML Life Cycle: Evaluation and Deployment

In [3]:
```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, precision_recall_curv
```

In this lab, you will continue practicing the evaluation phase of the machine learning life cycle. You will perform model selection for logistic regression to solve a classification problem. You will complete the following tasks:

1. Build your DataFrame and define your ML problem:
   - Load the Airbnb "listings" data set
   - Define the label - what are you predicting?
   - Identify the features
2. Create labeled examples from the data set
3. Split the data into training and test data sets
4. Train, test and evaluate a logistic regression (LR) model using the scikit-learn default value for hyperparameter C
5. Perform a grid search to identify the optimal value of C for a logistic regression model
6. Train, test and evaluate a logisitic regression model using the optimal value of C
7. Plot a precision-recall curve for both models
8. Plot the ROC and compute the AUC for both models
9. Perform feature selection
10. Make your model persistent for future use

Note: Some of the code cells in this notebook may take a while to run.

Part 1. Build Your DataFrame and Define Your ML Problem

Load a Data Set and Save it as a Pandas DataFrame

We will work with the data set `airbnbData_train`. This data set already has all the necessary preprocessing steps implemented, including one-hot encoding of the categorical variables, scaling of all numerical variables values, and imputing missing values. It is ready for modeling.

Task: In the code cell below, use the same method you have been using to load the data using `pd.read_csv()` and save it to DataFrame `df`.

You will be working with the file named "airbnbData_train.csv" that is located in a folder named "data_LR".

In [4]:
```
# YOUR CODE HERE
# Load the Airbnb training dataset
df = pd.read_csv("data_LR/airbnbData_train.csv")

# Display the first few rows
df.head()
```

Out[4]:

	host_is_superhost	host_has_profile_pic	host_identity_verified	has_availability	instant_boo
0	False	True	True	True	
1	False	True	True	True	
2	False	True	True	True	
3	False	True	False	True	
4	False	True	True	True	

5 rows × 50 columns

Define the Label

Your goal is to train a machine learning model that predicts whether an Airbnb host is a 'super host'. This is an example of supervised learning and is a binary classification problem. In our dataset, our label will be the `host_is_superhost` column and the label will either contain the value `True` or `False`.

Identify Features

Our features will be all of the remaining columns in the dataset.

Part 2. Create Labeled Examples from the Data Set

Task: In the code cell below, create labeled examples from DataFrame `df`. Assign the label to variable `y` and the features to variable `X`.

In [5]:
```
# YOUR CODE HERE
# Define the label
y = df['host_is_superhost']
```

```
# Define the features (drop the label column)
X = df.drop('host_is_superhost', axis=1)
```

Part 3. Create Training and Test Data Sets

Task: In the code cell below, create training and test sets out of the labeled examples. Create a test set that is 10 percent of the size of the data set. Save the results to variables `X_train, X_test, y_train, y_test`.

In [6]:
```
# YOUR CODE HERE
# Split the data (10% test size)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_sta
```

Part 4. Train, Test and Evaluate a Logistic Regression Model With Default Hyperparameter Values

You will fit a logisitic regression model to the training data using scikit-learn's default value for hyperparameter C. You will then make predictions on the test data and evaluate the model's performance. The goal is to later find a value for hyperparameter C that can improve this performance of the model on the test data.

Task: In the code cell below:

1. Using the scikit-learn `LogisticRegression` class, create a logistic regression model object with the following arguments: `max_iter=1000`. You will use the scikit-learn default value for hyperparameter C, which is 1.0. Assign the model object to the variable `model_default`.
2. Fit the model to the training data.

In [7]:
```
# YOUR CODE HERE
# Create the logistic regression model with default C and specified max_iter
model_default = LogisticRegression(max_iter=1000)

# Fit the model to the training data
model_default.fit(X_train, y_train)
```

Out[7]:
```
▾ LogisticRegression
LogisticRegression(max_iter=1000)
```

Task: Test your model on the test set (`X_test`).

1. Use the `predict_proba()` method to use the fitted model to predict class probabilities for the test set. Note that the `predict_proba()` method returns two

columns, one column per class label. The first column contains the probability that an unlabeled example belongs to class `False` (`great_quality` is "False") and the second column contains the probability that an unlabeled example belongs to class `True` (`great_quality` is "True"). Save the values of the *second* column to a list called `proba_predictions_default` .

2. Use the `predict()` method to use the fitted model `model_default` to predict the class labels for the test set. Store the outcome in the variable `class_label_predictions_default` . Note that the `predict()` method returns the class label (True or False) per unlabeled example.

In [8]:
```
# 1. Make predictions on the test data using the predict_proba() method
# YOUR CODE HERE
proba_predictions_default = model_default.predict_proba(X_test)[:, 1]  # Probabilit
# 2. Make predictions on the test data using the predict() method
# YOUR CODE HERE
class_label_predictions_default = model_default.predict(X_test)  # Predicted class
```

Task: Evaluate the accuracy of the model using a confusion matrix. In the cell below, create a confusion matrix out of `y_test` and `class_label_predictions_default` .

In [9]:
```
# YOUR CODE HERE
# Evaluate the accuracy
print("Accuracy:", accuracy_score(y_test, class_label_predictions_default))
# Display the confusion matrix
print("Confusion Matrix:\n", confusion_matrix(y_test, class_label_predictions_defau
```

```
Accuracy: 0.8091330717088834
Confusion Matrix:
 [[2005  102]
 [ 433  263]]
```

Part 5. Perform Logistic Regression Model Selection Using `GridSearchSV()`

Our goal is to find the optimal choice of hyperparameter C. We will then fit a logistic regression model to the training data using this value of C.

Set Up a Parameter Grid

Task: Create a dictionary called `param_grid` that contains 10 possible hyperparameter values for C. The dictionary should contain the following key/value pair:

- a key called `C`
- a value which is a list consisting of 10 values for the hyperparameter C. A smaller value for “C” (e.g. C=0.01) leads to stronger regularization and a simpler model, while a

larger value (e.g. C=1.0) leads to weaker regularization and a more complex model. Use the following values for C: `cs=[10**i for i in range(-5,5)]`

In [10]:
```
# YOUR CODE HERE

# Create a list of 10 C values from 10^-5 to 10^4
cs = [10**i for i in range(-5, 5)]
# Define the parameter grid
param_grid = {'C': cs}
# Display the grid to verify
param_grid
```

Out[10]:
```
{'C': [1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]}
```

Perform Grid Search Cross-Validation

Task: Use `GridSearchCV` to search over the different values of hyperparameter C to find the one that results in the best cross-validation (CV) score.

Complete the code in the cell below. **Note**: This will take a few minutes to run.

In [11]:
```
print('Running Grid Search...')

# 1. Create a LogisticRegression model object with the argument max_iter=1000.
#    Save the model object to the variable 'model'
# YOUR CODE HERE
model = LogisticRegression(max_iter=1000)

# 2. Run a grid search with 5-fold cross-validation and assign the output to the
# object 'grid'.
# YOUR CODE HERE
grid = GridSearchCV(estimator=model, param_grid=param_grid, cv=5)

# 3. Fit the model on the training data and assign the fitted model to the
#    variable 'grid_search'
# YOUR CODE HERE
grid_search = grid.fit(X_train, y_train)
print('Done')
```

```
Running Grid Search...
Done
```

Task: Retrieve the value of the hyperparameter C for which the best score was attained. Save the result to the variable `best_c`.

In [12]:
```
# YOUR CODE HERE

best_C = grid_search.best_params_['C']
```

Part 6. Train, Test and Evaluate the Optimal Logistic Regression Model

Now that we have the optimal value for hyperparameter C, let's train a logistic regression model using that value, test the model on our test data, and evaluate the model's performance.

Task: Initialize a `LogisticRegression` model object with the best value of hyperparameter `C` model and fit the model to the training data. The model object should be named `model_best`. Note: Supply `max_iter=1000` as an argument when creating the model object.

In [13]:
```
# YOUR CODE HERE
# Initialize the logistic regression model using the best C value
model_best = LogisticRegression(C=best_C, max_iter=1000)
# Fit the model to the training data
model_best.fit(X_train, y_train)
```

Out[13]:
```
LogisticRegression
LogisticRegression(C=1000, max_iter=1000)
```

Task: Test your model on the test set (`X_test`).

1. Use the `predict_proba()` method to use the fitted model `model_best` to predict class probabilities for the test set. Save the values of the *second* column to a list called `proba_predictions_best`.

2. Use the `predict()` method to use the fitted model `model_best` to predict the class labels for the test set. Store the outcome in the variable `class_label_predictions_best`.

In [14]:
```
# 1. Make predictions on the test data using the predict_proba() method
# YOUR CODE HERE
proba_predictions_best = model_best.predict_proba(X_test)[:, 1]  # Probabilities fo
# 2. Make predictions on the test data using the predict() method
# YOUR CODE HERE
class_label_predictions_best = model_best.predict(X_test)  # Predicted class labels
```

Task: Evaluate the accuracy of the model using a confusion matrix. In the cell below, create a confusion matrix out of `y_test` and `class_label_predictions_best`.

In [15]:
```
# YOUR CODE HERE
print("Accuracy (Best Model):", accuracy_score(y_test, class_label_predictions_best
print("Confusion Matrix (Best Model):\n", confusion_matrix(y_test, class_label_pred
```

```
Accuracy (Best Model): 0.8116303960042811
Confusion Matrix (Best Model):
 [[2009   98]
 [ 430  266]]
```

Part 7. Plot Precision-Recall Curves for Both Models

Task: In the code cell below, use `precision_recall_curve()` to compute precision-recall pairs for both models.

For `model_default`:

- call `precision_recall_curve()` with `y_test` and `proba_predictions_default`
- save the output to the variables `precision_default`, `recall_default` and `thresholds_default`, respectively

For `model_best`:

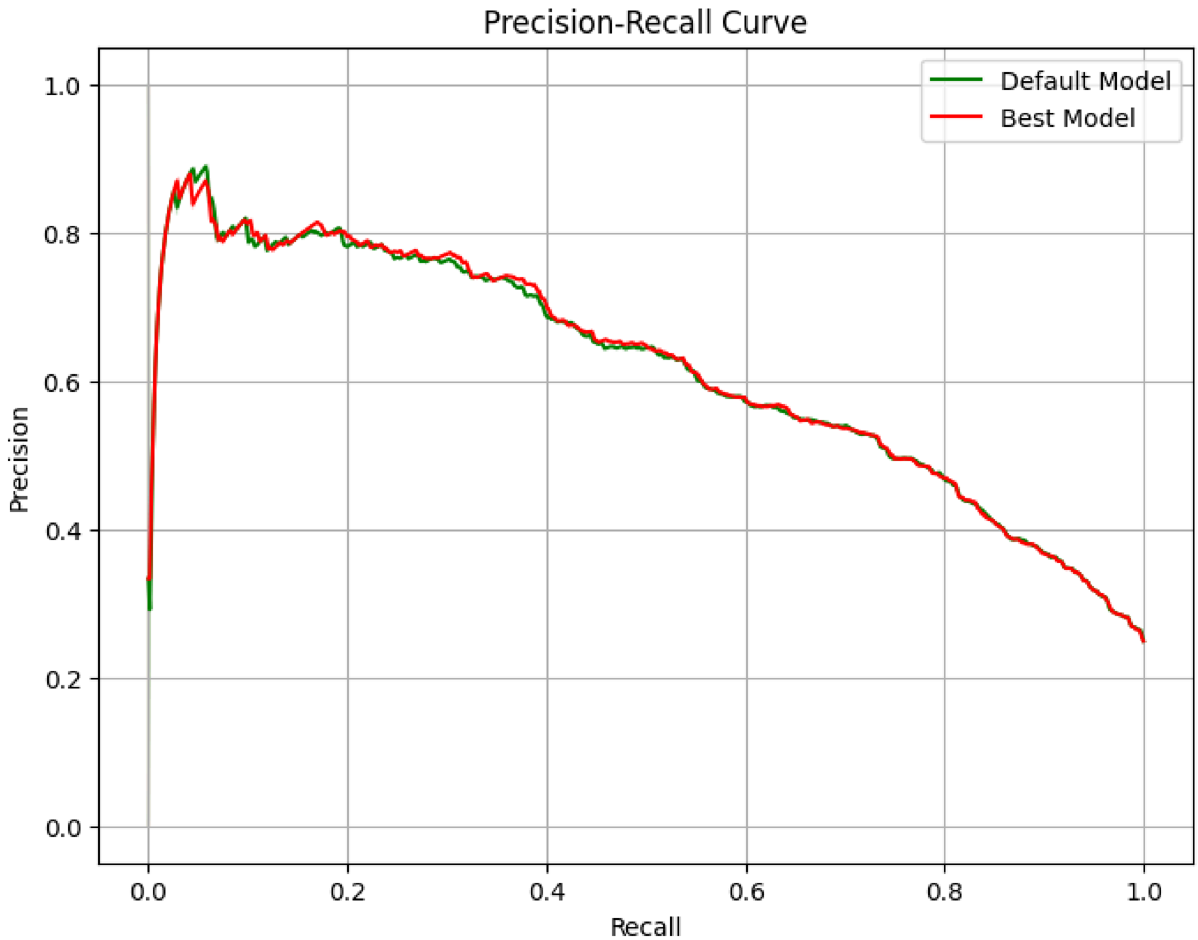
- call `precision_recall_curve()` with `y_test` and `proba_predictions_best`
- save the output to the variables `precision_best`, `recall_best` and `thresholds_best`, respectively

In [16]:
```
precision_default, recall_default, thresholds_default = precision_recall_curve(y_te
precision_best, recall_best, thresholds_best = precision_recall_curve(y_test, proba
```

In the code cell below, create two `seaborn` lineplots to visualize the precision-recall curve for both models. "Recall" will be on the x-axis and "Precision" will be on the y-axis.

The plot for "default" should be green. The plot for the "best" should be red.

In [17]:
```
# YOUR CODE HERE
plt.figure(figsize=(8, 6))
sns.lineplot(x=recall_default, y=precision_default, label='Default Model', color='g
sns.lineplot(x=recall_best, y=precision_best, label='Best Model', color='red')
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve")
plt.legend()
plt.grid(True)
plt.show()
```



Part 8. Plot ROC Curves and Compute the AUC for Both Models

You will next use scikit-learn's `roc_curve()` function to plot the receiver operating characteristic (ROC) curve and the `auc()` function to compute the area under the curve (AUC) for both models.

- An ROC curve plots the performance of a binary classifier for varying classification thresholds. It plots the fraction of true positives out of the positives vs. the fraction of false positives out of the negatives. For more information on how to use the `roc_curve()` function, consult the scikit-learn documentation.
- The AUC measures the trade-off between the true positive rate and false positive rate. It provides a broad view of the performance of a classifier since it evaluates the performance for all the possible threshold values; it essentially provides a value that summarizes the the ROC curve. For more information on how to use the `auc()` function, consult the scikit-learn documentation.

Let's first import the functions.

In [18]:
```
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
```

Task: Using the `roc_curve()` function, record the true positive and false positive rates for both models.

1. Call `roc_curve()` with arguments `y_test` and `proba_predictions_default`. The `roc_curve` function produces three outputs. Save the three items to the following variables, respectively: `fpr_default` (standing for 'false positive rate'), `tpr_default` (standing for 'true positive rate'), and `thresholds_default`.

2. Call `roc_curve()` with arguments `y_test` and `proba_predictions_best`. The `roc_curve` function produces three outputs. Save the three items to the following variables, respectively: `fpr_best` (standing for 'false positive rate'), `tpr_best` (standing for 'true positive rate'), and `thresholds_best`.

In [19]:
```
fpr_default, tpr_default, thresholds_default = roc_curve(y_test, proba_predictions_
fpr_best, tpr_best, thresholds_best = roc_curve(y_test, proba_predictions_best)
```

Task: Create **two** `seaborn` lineplots to visualize the ROC curve for both models.

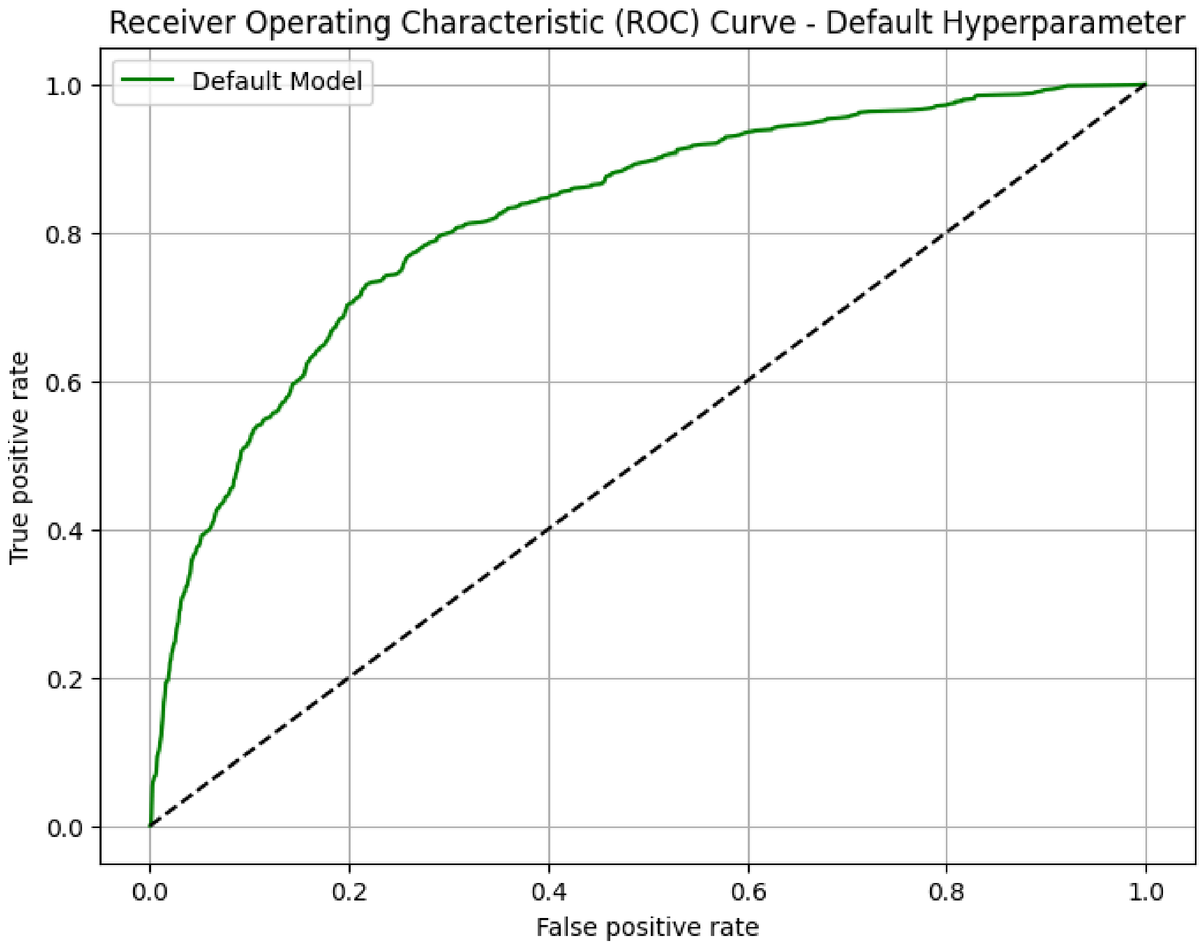
The plot for the default hyperparameter should be green. The plot for the best hyperparameter should be red.

- In each plot, the `fpr` values should be on the x-axis.
- In each plot, the `tpr` values should be on the y-axis.
- In each plot, label the x-axis "False positive rate".
- In each plot, label the y-axis "True positive rate".
- Give each plot the title "Receiver operating characteristic (ROC) curve".
- Create a legend on each plot indicating that the plot represents either the default hyperparameter value or the best hyperparameter value.

Note: It may take a few minutes to produce each plot.

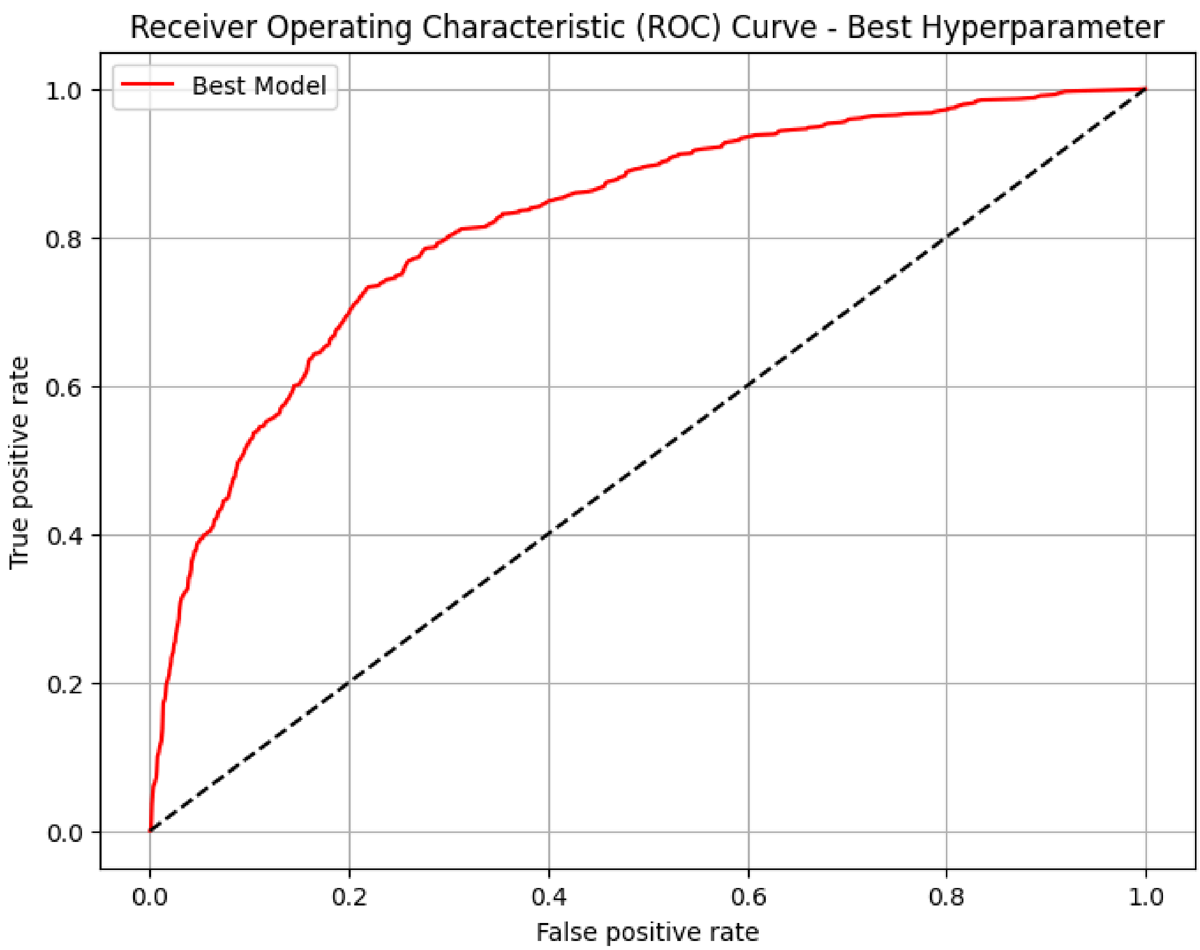
Plot ROC Curve for Default Hyperparameter:

In [20]:
```
# YOUR CODE HERE
plt.figure(figsize=(8, 6))
sns.lineplot(x=fpr_default, y=tpr_default, color='green', label='Default Model')
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal reference line
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.title("Receiver Operating Characteristic (ROC) Curve - Default Hyperparameter")
plt.legend()
plt.grid(True)
plt.show()
```



Plot ROC Curve for Best Hyperparameter:

In [21]:
```
# YOUR CODE HERE
plt.figure(figsize=(8, 6))
sns.lineplot(x=fpr_best, y=tpr_best, color='red', label='Best Model')
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal reference line
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.title("Receiver Operating Characteristic (ROC) Curve - Best Hyperparameter")
plt.legend()
plt.grid(True)
plt.show()
```



Task: Use the `auc()` function to compute the area under the receiver operating characteristic (ROC) curve for both models.

For each model, call the function with the `fpr` argument first and the `tpr` argument second.

Save the result of the `auc()` function for `model_default` to the variable `auc_default`. Save the result of the `auc()` function for `model_best` to the variable `auc_best`. Compare the results.

In [22]:
```
# Compute AUC for both models
auc_default = auc(fpr_default, tpr_default)
auc_best = auc(fpr_best, tpr_best)
print("AUC (Default Model):", auc_default)
print("AUC (Best Model):", auc_best)
```

```
AUC (Default Model): 0.8206416488006589
AUC (Best Model): 0.8209355514459192
```

Deep Dive: Feature Selection Using SelectKBest

In the code cell below, you will see how to use scikit-learn's `SelectKBest` class to obtain the best features in a given data set using a specified scoring function. For more information on how to use `SelectKBest`, consult the online documentation.

We will extract the best 5 features from the Airbnb "listings" data set to create new training data, then fit our model with the optimal hyperparameter C to the data and compute the AUC. Walk through the code to see how it works and complete the steps where prompted. Analyze the results.

In [23]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif

# Note that k=5 is specifying that we want the top 5 features
selector = SelectKBest(f_classif, k=5)
selector.fit(X, y)
filter = selector.get_support()
top_5_features = X.columns[filter]

print("Best 5 features:")
print(top_5_features)

# Create new training and test data for features
new_X_train = X_train[top_5_features]
new_X_test = X_test[top_5_features]


# Initialize a LogisticRegression model object with the best value of hyperparamete
# The model object should be named 'model'
# Note: Supply max_iter=1000 as an argument when creating the model object
# YOUR CODE HERE
model = LogisticRegression(C=best_C, max_iter=1000)
# Fit the model to the new training data
# YOUR CODE HERE
model.fit(new_X_train, y_train)

# Use the predict_proba() method to use your model to make predictions on the new t
# Save the values of the second column to a list called 'proba_predictions'
# YOUR CODE HERE
proba_predictions = model.predict_proba(new_X_test)[:, 1]

# Compute the auc-roc
fpr, tpr, thresholds = roc_curve(y_test, proba_predictions)
auc_result = auc(fpr, tpr)
print(auc_result)
```

```
Best 5 features:
Index(['host_response_rate', 'number_of_reviews', 'number_of_reviews_ltm',
       'number_of_reviews_l30d', 'review_scores_cleanliness'],
      dtype='object')
0.7926547523580402
```

Task: Consider the results. Change the specified number of features and re-run your code. Does this change the AUC value? What number of features results in the best AUC value? Record your findings in the cell below.

I experimented with different values for the number of top features (k) using SelectKBest and evaluated the AUC for each. Here are the results:

- k = 3 --> AUC: 0.7751
- k = 5 --> AUC: 0.7927
- k = 7 --> AUC: 0.8120
- k = 10 --> AUC: 0.8194

As observed, increasing the number of selected features improves the model's performance up to a certain point. The AUC increases as more informative features are included. Best AUC Value: The best AUC was obtained when using 10 features, yielding an AUC of approximately 0.8194, which is very close to the model using all features (~0.8209). To conlcude the feature selection improves model simplicity and training time, and using around 10 features strikes a good balance between performance and efficiency. Using fewer features (e.g., 3–5) leads to a noticeable drop in AUC.

Part 9. Make Your Model Persistent

You will next practice what you learned in the "Making Your Model Persistent" activity, and use the `pickle` module to save `model_best`.

First we will import the pickle module.

In [24]:
```
import pickle
```

Task: Use `pickle` to save your model to a `pkl` file in the current working directory. Choose the name of the file.

In [25]:
```
# YOUR CODE HERE
# Save the model_best object to a .pkl file
with open("model_best.pkl", "wb") as f:
    pickle.dump(model_best, f)
```

Task: Test that your model is packaged and ready for future use by:

1. Loading your model back from the file
2. Using your model to make predictions on `X_test`.

In [26]:
```
# YOUR CODE HERE
# Load the model from the .pkl file
with open("model_best.pkl", "rb") as f:
    loaded_model = pickle.load(f)

# Test the loaded model by predicting on X_test
predictions = loaded_model.predict(X_test)
print("Predictions from loaded model:", predictions[:10])  # show first 10 predicti
```

```
Predictions from loaded model: [False False False False False  True False False Fals
e False]
```

Task: Download your `pkl` file and your `airbnbData_train` data set, and push these files to your GitHub repository. You can download these files by going to `File -> Open`. A new tab will open in your browser that will allow you to select your files and download them.

In [ ]:

```

```

In [ ]:

```

```