Table of Contents

[1. Introduction 2](#_Toc156518878)

[2. Data Exploration 3](#_Toc156518879)

[3. Data Visualization 6](#_Toc156518880)

[4. Data Preparation 9](#_Toc156518881)

[5. Model Building with Hyperparameter Tuning 13](#_Toc156518882)

[6. Model Evaluation by Performance Metrics 16](#_Toc156518883)

[References 21](#_Toc156518884)

# 1. Introduction

In the contemporary era, the advancements in technology have facilitated widespread access to rapid transportation, enabling more and more customers to traverse the globe conveniently. Among all means of available transportations, airplane emerge as the most common choice. With numerous airline companies competing for the market share, retaining the existing customers becomes a priority task. To address this, they should proactively enhance the potential sources of dissatisfaction and give offers to potentially dissatisfied customers. Building a machine learning algorithm could help identify customers with a high likelihood of dissatisfaction based on their flight characteristics. This study aims to build a logistic regression model to assist airline company in predicting the customer dissatisfaction. According to Boateng & Abaye (2019), logistic regression is commonly the preferred statistic for situations where predicting the occurrence of a binary outcome is based on one or more independent variables. Given that our study’s target variable dissatisfaction can be transformed to binary format (1: dissatisfaction, 0: satisfaction), logistic regression is a fitting choice as our model building algorithm. Table 1.1 below shows the information of all columns in the dataset:

**Table 1.1**

*Dataset Metadata*

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Remark |
| Id | Integer | Customer’s unique identifier |
| Gender | String | Male or Female |
| Customer Type | String | Loyal Customer or disloyal Customer |
| Age | Integer | Customer’s age |
| Type of Travel | String | Business Travel or Personal Travel |
| Class | String | Business, Eco, Eco Plus |
| Flight Distance | Integer | Distance of flight |
| Inflight wifi service | Integer | Level of service: 0 to 5, 0 means no wifi service |
| Departure/Arrival time convenient | Integer | Level of convenience: 0 to 5, 0 means unsatisfactory; 5 means perfect satisfactory |
| Ease of Online Booking | Integer | Same as above |
| Gate location | Integer | Same as above |
| Food and drink | Integer | Same as above |
| Online boarding | Integer | Same as above |
| Seat comfort | Integer | Same as above |
| Inflight entertainment | Integer | Same as above |
| On-board service | Integer | Same as above |
| Leg room service | Integer | Same as above |
| Baggage handling | Integer | Same as above |
| Check-in service | Integer | Same as above |
| Inflight service | Integer | Same as above |
| Cleanliness | Integer | Same as above |
| Departure Delay in Minutes | Integer | Number of minutes delayed when departure |
| Arrival Delay in Minutes | Integer | Number of minutes delayed when arrival |
| Satisfaction | String | Satisfaction, neutral or dissatisfaction |

# 2. Data Exploration

This study utilizes Visual Studio Code and a third-party extension Jupyter Notebook to perform data extraction, data exploration, data wrangling, data transformation and model building with hyperparameters tuning. The first step is to extract data from two csv files using Pandas library. The “train\_test.csv” file is used for building and testing model before deployment while the “deploy\_validate.csv” file is used for validation when model is deployed. We extract them to “traintest” and “validate” dataframes respectively. As shown in Figure 2.1, the size of “traintest” dataframe is (103904, 25), meaning it has 103904 rows and 25 columns.

**Figure 2.1**

*Data Import Code Snippet*

A screenshot of a computer program

Description automatically generated

Next, we dropped irrelevant columns “Unnamed: 0” and “id” as they do not contribute to target variable “satisfaction” in model building. After this, we checked the information of dataframe as shown in Figure 2.2. The “Gender”, “Cutomer Type”, “Type of Travel”, “Class”, “Satisfaction” columns are categorical columns. They are of string datatype, but in Pandas dataframe, they are categorized as object datatype. The other columns are of numeric datatype as they contain only integer (int64) or float values. Among these numeric columns, #6 to #19 columns are categorical columns, because the numbers inside these columns are ordinal, indicating the convenience level from 0 to 5.

**Figure 2.2**

*Drop Columns & Dataframe Information Code Snippet*

A screenshot of a computer program

Description automatically generated

In Figure 2.3, we replaced the spaces in column names with underscores to ease the access of columns. In addition, we transformed target variable “satisfaction” to be binary values: 1 as “neutral or dissatisfied”; 0 as “satisfied”. We transform “neutral or dissatisfied” as 1 because our goal is to predict the dissatisfaction, so dissatisfaction should be a positive case.

**Figure 2.3**

*Rename & Transform Columns Code Snippet*

A screen shot of a computer code

Description automatically generated

In Figure 2.4, we checked null values in “traintest” and “validate” datasets and found out there are missing values in “Arrival\_Delay\_in\_Minutes”. According to the experience from Zhang (2016), for numeric variables, he proposed imputing them with the mean, while for categorical variables, the mode will be used. Thus, we will impute the missing value of numeric variable “Arrival\_Delay\_in\_Minutes” with the mean value.

**Figure 2.4**

*Impute Mean Value Code Snippet*

A screen shot of a computer

Description automatically generated

# 3. Data Visualization

Following the imputation of missing values, we proceeded with data visualization using the matplotlib and Seaborn libraries. In Figure 3.1, we created the first graph, a barplot, to illustrate the distribution of the target variable “satisfaction”. From the graph we can see the discrepancy between 0 and 1 classes is approximately 15000 cases, which is deemed insubstantial. Thus, we can conclude that two classes of the target variable “satisfaction” shows a balanced distribution.

**Figure 3.1**

*Distribution of Satisfaction Column Code Snippet & Visualization*

A screenshot of a computer screen

Description automatically generated

The graphs shown in Figure 3.2 illustrates these relationships: satisfaction against gender and satisfaction against customer type. From the left graph, we can conclude that the distribution of both genders in two satisfaction levels is balanced. Besides, for both male and female customers, the number of dissatisfied customers is higher than the number of satisfied customers. The right graph showcases loyal customers occupy significantly larger proportion of all customers than disloyal customers. Moreover, the proportion of dissatisfaction in disloyal customers is higher than the one in loyal customers.

**Figure 3.2**

*Satisfaction Column against Gender/Customer Type Code Snippet & Visualization*

A screenshot of a computer screen

Description automatically generated

Furthermore, we explored more complex relationship among Class, Online Boarding Convenient Level, Departure/Arrival Time Convenient Level, and satisfaction. The graph in Figure 3.3 shows three graphs which illustrates Online Boarding Convenient Level against Departure Arrival Time Convenient Level, grouped by Class. From these graphs, we concluded that for Eco Plus Class, there are more dissatisfied customers than satisfied customers when the Departure/Arrival Time Convenient Level is the lowest (0), despite Online Boarding level is acceptable (2.5). In other words, the customers who take Eco Plus Class are sensitive to inconvenient departure time and arrival time, and they are more likely to be dissatisfied.

**Figure 3.3**

*Online\_boarding against Departure/Arrival\_time\_convenient Code Snippet & Visualization*

A screenshot of a graph

Description automatically generated

Lastly, we delved into relationship between Class and Departure/Arrival Delay Minutes. In Figure 3.4, we built two catplots (Category plots), one is for Class and Departure Delay Minutes, and another is for Class and Arrival Delay Minutes. The graphs generated by the code resembled each other in Figure 3.5. From these two graphs, we can conclude that for personal travel, especially in Eco and Eco Plus classes, when the arrival or departure delay in minutes is high, the number of dissatisfied customers is much higher than the number of satisfied customers. This implies the customers taking Eco and Eco Plus classes for personal travel are sensitive to high delay time for both departure and arrival.

**Figure 3.4**

*Class against Departure/Arrival Delay Minutes Code Snippet*



**Figure 3.5**

*Class against Departure/Arrival Delay Minutes Visualization*

A screenshot of a graph

Description automatically generated

# 4. Data Preparation

Before building model, we encoded categorical variables for both “traintest” and “validate” dataframes using LabelEncoder from sklearn library as shown in Figure 4.1. After that, we checked outliers in the “traintest” dataframe using interquartile range method in Figure 4.2, where we printed out interquartile range for each column. In Figure 4.3, we removed these outliers detected and there are 61197 remaining rows of data.

**Figure 4.1**

*Label Encoding for Categorical Variables Code Snippet*

**A screen shot of a computer program

Description automatically generated**

**Figure 4.2**

*Outliers Detection Code Snippet*

**A screenshot of a computer program

Description automatically generated**

**Figure 4.3**

*Outliers Removal Code Snippet*

**A screen shot of a computer code

Description automatically generated**

Subsequently, we performed feature selection by using chi-squared method and a correlation heatmap. The code and result for feature selection based on chi-squared score are shown in Figure 4.4. According to the result, there are 10 columns being selected, from “Type\_of\_Travel” to “Cleanliness” Columns. Apart from chi-squared method, we plan to select more variables based on the correlation heatmap. We visualized the correlation between the target variable “satisfaction” and other columns using a heatmap. The code is shown in Figure 4.5 and the heatmap is demonstrated in Figure 4.6. According to the heatmap, all 10 columns selected based on chi-squared method have slight to moderate correlation to “satisfaction”. Other than these 10 columns, we selected two more columns based on the heatmap correlation scores, which are “Checkin\_service” and “Inflight\_service”.

**Figure 4.4**

*Feature Selection by Chi-squared Method Code Snippet and Result*

**A screenshot of a computer program

Description automatically generated**

**Figure 4.5**

*Correlation Heatmap Code Snippet* **A black background with colorful text

Description automatically generated**

**Figure 4.6**

*Correlation Heatmap*

**A screenshot of a graph

Description automatically generated**

After selecting features for model building, we separated independent variables to variable “X” and target variable to “y”. Following this, we split train and test datasets into “X\_train, X\_test, y\_train, y\_test” variables by using “train\_test\_split” function. As shown in Figure 4.7, the test size is 0.2, meaning we allocated 20% of original dataset to be test data and the left 80% data to be train data. Finally, we scaled independent variables stored in both train and test as shown in Figure 4.8. Scaling ensures that each feature contributes almost equally to the model's learning and regularization is applied fairly across all features, preventing certain features from being unfairly penalized simply due to their scale.

**Figure 4.7**

*Data Splitting Code Snippet*

**A computer screen shot of a program code

Description automatically generated**

**Figure 4.8**

*Scaling Independent Variables Code Snippet*

**A screen shot of a computer program

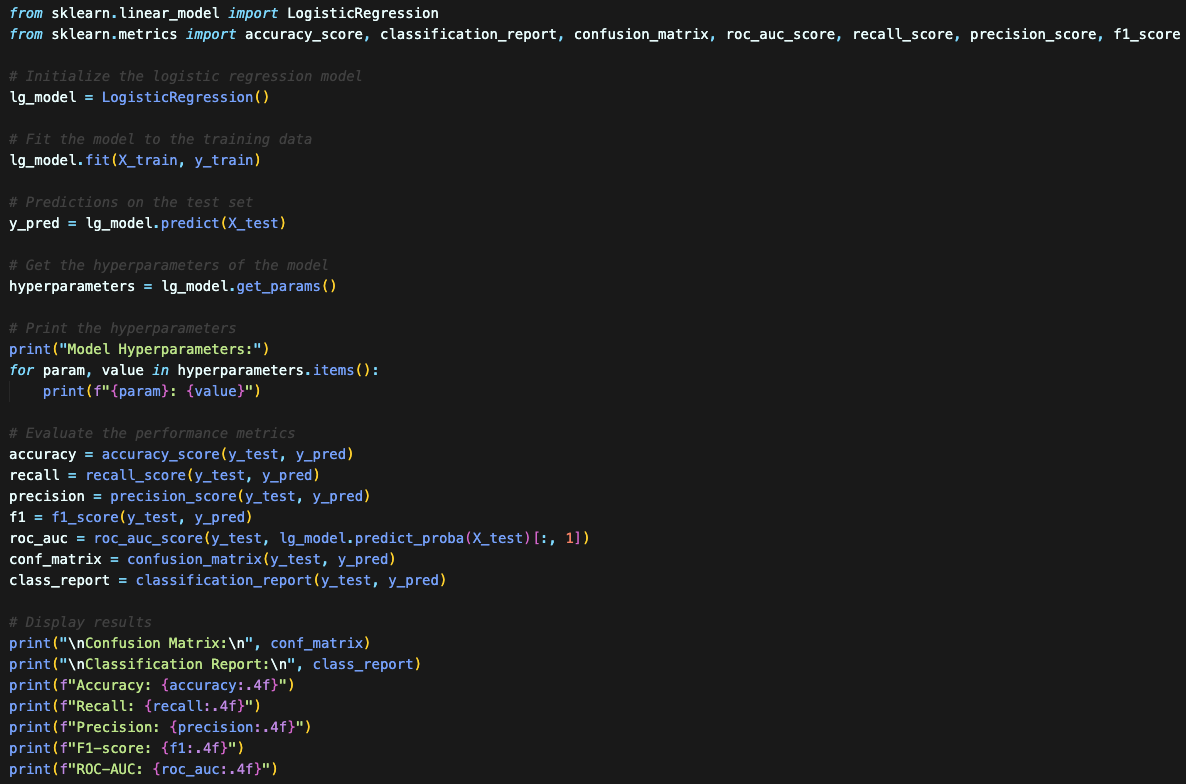
Description automatically generated**

# 5. Model Building with Hyperparameter Tuning

We built a default logistic regression model and other logistic regression models with hyperparameter tuning once data preparation and data preparation were finished. The below Figure 5.1 shows the code for building the default logistic regression model. At the beginning, we imported logistic regression model and performance metrics from “scikitlearn” library. Next, we created a logistic regression model instance “lg\_model” and we fitted this model with training data, “X\_train” and “y\_train”. Then we used “y\_pred” to store the prediction made by the trained model on test data “y\_test”. Following that, we retrieved the hyperparameters of this model so that we could compare with the other models. Finally, we retained the performance metrics and print them out at the last section.

**Figure 5.1**

*Model Building with Default Hyperparameters Code Snippet*



In addition, we created 10 more models by tuning hyperparameters. As shown in Figure 5.2, we imported “RepeatedStratifiedKFold” and “GridSearchCV” from “sklearn.model\_selection” library. After that, we created a new logistic regression model instance to train with. Next, we specified the hyperparameters that needed to be tuned for the logistic regression model. We tuned three kinds of hyperparameters, which are solves, penalty and c values. We included “newton-cg”, “lbfgs” or “liblinear” as the solvers of logistic regression. Solvers are different optimization algorithms for logistic regression to calculate optimal weights. The full name of first solver “newton-cg” is Newton conjugate gradient algorithm. It is an optimization algorithm based on Newton’s method to find the minimum of a function and it is suitable for small to medium-sized datasets, which aligns with the small size of our train data (approximately 50k rows). The second solver “lbfgs” is “Limited-memory Broyden-Fletcher-Goldfarb-Shanno” algorithm for optimization. The last solver “liblinear” is a solver that uses a linear approach and is a good choice for binary classification problems (in this case, dissatisfaction prediction). For penalty, it refers to the type of regularization applied to the logistic regression model. Here, we chose “L2” Ridge regularization because our train data has multiple features (13 in total) and most of them contribute to the prediction. The final hyperparameter is “c\_values”. They are inverse of regularization strength; smaller values of c mean stronger regularization and bigger values of c indicate weaker regularization. We provided 5 values from 100 to 0.01 so that each c-value will be tested.

After setting values for the solvers, penalty, and c-values, we put them in a Python dictionary called “param-grid. The “GridSearchCV” will search through the combinations of solvers, penalty, and c-values, trying each combination to find the best set of hyperparameters. The next line of code creates a cross-validation strategy into the variable “cv” using “RepeatedStratifiedKFold”. Cross-validation is a technique used to assess how well a model will generalize to an independent dataset. “RepeatedStratifiedKFold” here ensures the data is split into 10 folds, and the process is repeated 3 times with random split of data enabled. Following this, a “GridSearchCV” instance “grid\_search” is created with our predefined logistic regression model “logreg\_model”, the hyperparameters dictionary “param\_grid”, and the cross-validation strategy “cv”. We set the scoring to “recall” so that the grid search will search the model with best hyperparameters based on their recall score. The reason we selected recall to be the scoring metric is it aligns with our business goal: the early detection of possible dissatisfaction. Recall represents the proportion of actual dissatisfied customers that the model correctly identifies. If the recall is high, it means the model captures as many dissatisfied customers as possible.

Once the “grid\_search” instance was initialized, we fitted the grid search to the training data. We stored best hyperparameters of logistic regression model in “best\_params”, and we store the best model in “best\_model” variable. Finally, we use best model to predict on test data “X\_test” and store the results in variable “y\_pred”. In Figure 5.3, the rest of the code will retrieve the performance metrics of this best model and print them out at the end.

**Figure 5.2**

*Model Building with Hyperparameters Tuning Code Snippet*

A screen shot of a computer program

Description automatically generated

**Figure 5.3**

Model Building with Hyperparameters Tuning Results Code Snippet

A screen shot of a computer program

Description automatically generated

# 6. Model Evaluation by Performance Metrics

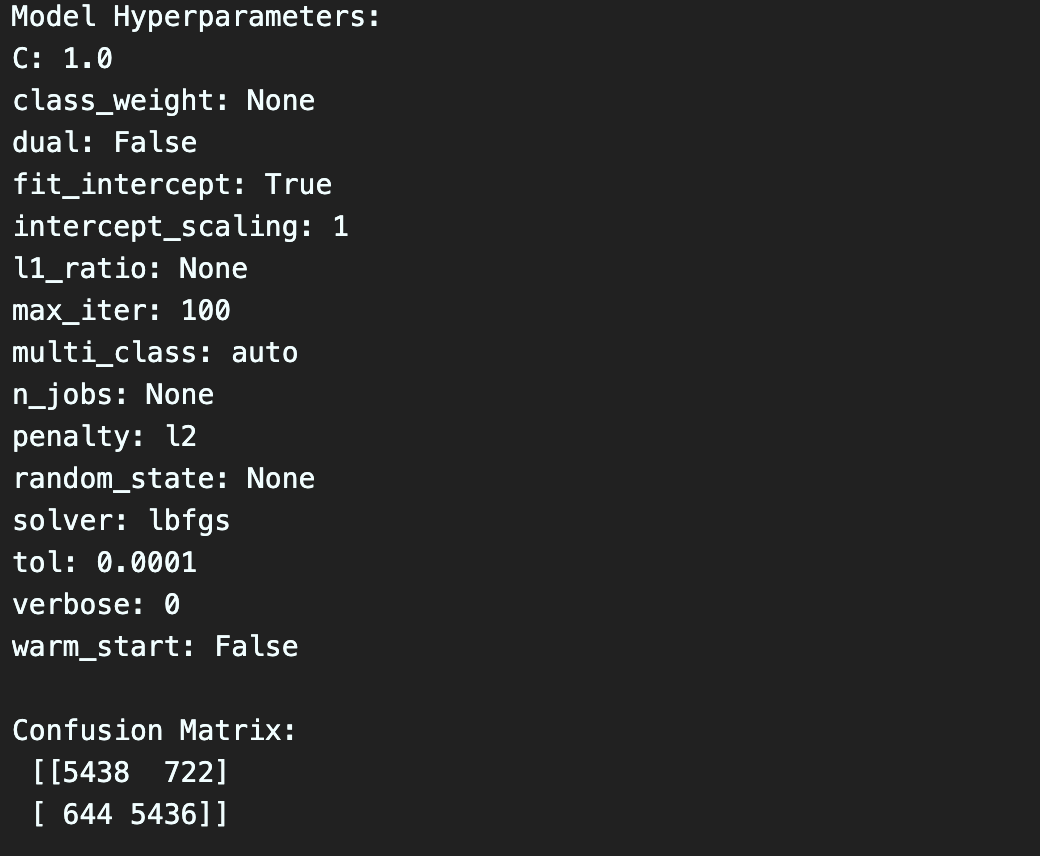
Since the logistic regression model is a classification model, we will evaluate all models created based on their accuracy, precision, recall, f1-score, and AUC-ROC (Area under the Receiver Operating Characteristic). Accuracy is the proportion of correctly classified instances (including true positives and true negatives) among the total number of instances. A high accuracy indicates a good overall performance. Precision is the proportion of true positive predictions among the total number of positive predictions (including true positives and false negatives). Recall is the proportion of true positive predictions among the total number of actual positive instances. F1- score is the harmonic mean of precision and recall. Lastly, AUC-ROC is a metric that evaluates model’s ability to discriminate between positive and negative instances. The true positive instance in this study is the dissatisfaction and the true negative instance is satisfaction. Among all these performance metrics, we priorities recall and AUC-ROC scores. Recall is crucial because we do not wish to miss any positive cases (dissatisfaction), which leads to high cost - losing customers. The AUC-ROC score is important as well because higher AUC-ROC score indicates better ability of discriminating the positive (dissatisfaction) and negative instances (satisfaction).

The result shown in Figure 6.1 are the hyperparameters and confusion matrix for the default model that we built in Figure 5.1. The key hyperparameters are 1.0 for C value, L2 for penalty and “lbfgs” optimization algorithm. Based on the confusion matrix, there are 5438 true positive cases, 722 false positive cases, 644 false negative cases and 5436 true negative cases.

In Figure 6.2, we can tell the accuracy of the default model is 88.84%, recall is 89.41%, Precision is 88.28%, F1-score is 88.84%, and ROC-AUC is 0.9439.

**Figure 6.1**

*Default Model Hyperparameters & Confusion Matrix*



**Figure 6.2**

*Default Model Performance Metrics*

*A screenshot of a computer screen

Description automatically generated*

The process of model building with hyperparameter is shown in Figure 6.3. It illustrates all 10 models created by grid search cross-validation with different combination of hyperparameters. In Figure 6.4, it shows the best model with its hyperparameters, which are 100 for C value, L2 for penalty, and “newton-cg” optimization algorithm. From the confusion matrix, we can tell this best model recognized 5438 true positive cases, 722 false positive cases, 644 false negative cases and 5436 true negative cases. The accuracy of the best model is 88.84%, recall is 89.41%, Precision is 88.28%, F1-score is 88.84%, and ROC-AUC is 0.9440. Apart from ROC-AUC and hyperparameters, the performance metrics of the best model are identical to the default model. Nevertheless, the ROC-AOC score of the best model is 0.0001 higher than the default model, and the higher C value (100) of the best model indicates less regularization strength to the train data. Therefore, we will choose the best model with hyperparameter combination of {c:100, penalty: l2, solver: newton-cg} as the final model. According to Hendricks (2022), the range of industry norms for performance metrics is 70% to 90%. Any performance metric higher than 70% is considered realistic and above threshold level. As the accuracy of the model is 88.84%, the recall is 89.41%, the ROC-AUC is 94.4%, we can conclude the best model has satisfactory performance, and it is ready to be deployed.

**Figure 6.3**

Models Created Based on Grid Search Cross-Validation

A screenshot of a computer

Description automatically generated

**Figure 6.4**

*Best Model Hyperparameters & Performance Metrics*

A screenshot of a computer

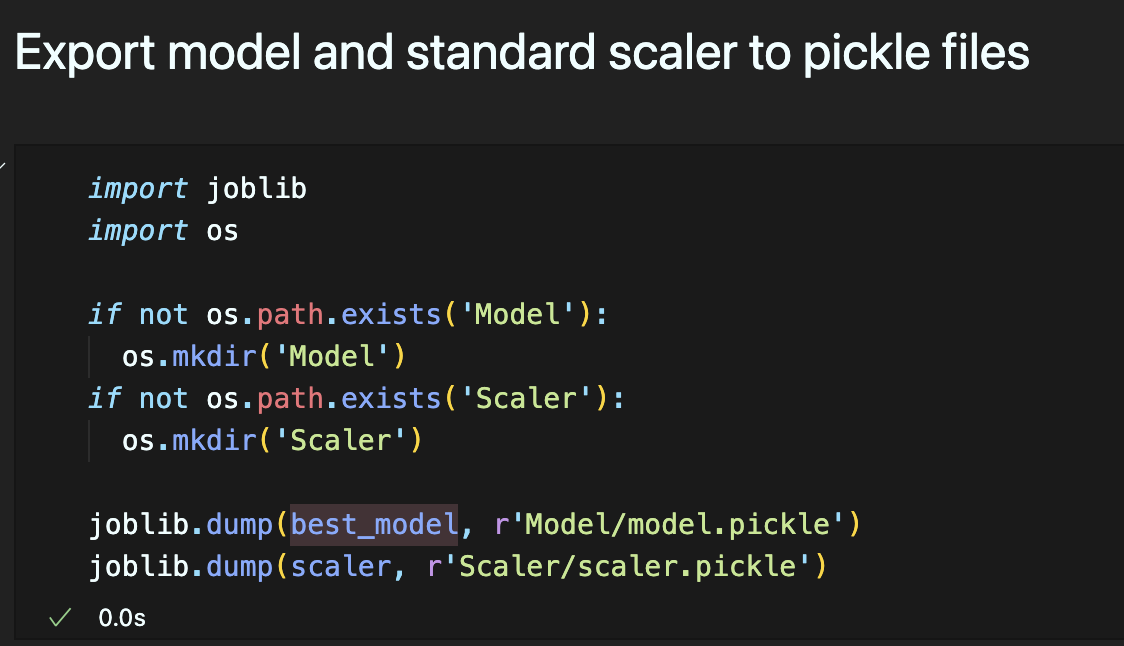
Description automatically generated

**7. Model Deployment**

To deploy this best model selected by grid search cross-validation, we exported the it and standard scaler from previous sections to “pickle” files by using “joblib” and “os” libraries from Python. In addition, we used the model to predict the outcome on the edited validation dataset to evaluate its performance on newly unseen data. In Figurre 7.1, we exported best model to the file “model.pickle” and the scaler to the file “scaler.pickle”. Next, we imported them again in Figure 7.2 to test the model performance on the validation dataset. The model imported is named “deploy\_model” and the standard scaler is stored in “deploy\_scaler” as shown in Figure 7.4. In Figure 7.2, we exported the modified validation dataset to “validate\_editted.csv” file, and then we imported it in Figure 7.3 to store in the “deploy\_validate” dataframe.

**Figure 7.1**

*Export Model and Scaler Code Snippet*



**Figure 7.2**

*Export Modified Validation Dataset Code Snippet*

**A black background with white text

Description automatically generated**

**Figure 7.3**

*Import Edited Dataset Code Snippet*

**A screen shot of a computer code

Description automatically generated**

**Figure 7.4**

*Import Model and Scaler Code Snippet*

*A screen shot of a computer code

Description automatically generated*

After importing all necessary instances, the feature selection and scaling were done on the validation dataframe as shown in Figure 7.5. The selected features were the 13 columns that we chose during the data splitting phase in Section 4. All columns that participate in prediction are stored in “prediction\_columns” dataframe. The scaling is applied to ensure all selected features have the same range of data in order to reduce bias.

**Figure 7.5**

*Feature Selection & Scaling on Validation Dataframe Code Snippet*

A screen shot of a computer program

Description automatically generated

Finally, we predicted the target variable “satisfaction” by using “deploy\_model” on the prediction columns. The results are stored in “output” variable, which are added as the last column “Prediction” in the validation dataset as shown in Figure 7.6. The “satisfaction” column is the actual satisfaction, while the “Prediction” column next to it is contains the predicted result. We could easily compare actual values and predictions as they align side by side. Please note that for the binary values in “satisfaction” and “prediction” columns, 1 means dissatisfaction; 0 means satisfaction since the goal is to predict the dissatisfaction.

**Figure 7.6**

*Predict Outcome Code Snippet*

A screenshot of a computer

Description automatically generated

The performance of this model can be accessed by recall, accuracy, and AUC-ROC scores. As shown in Figure 7.7, we imported these performance metrics from “sklearn.metrics” library and store each score in different variables. At last, we printed them out to see the model performance. From Figure 7.8, we could tell that the performance of the best model on the validation dataset is: 82.79% for accuracy, 79.81% for recall, 83.11% for ROC-AUC. Each performance metric has decreased comparing to the ones of test dataset. The reason of this phenomenon might be the overfitting to train data of the best model, so that it fails to generalize unseen data. However, since all these performance metrics are above 70%, it is still considered as a decent deployed model that has a satisfactory performance.

**Figure 7.7**

*Deployed Model Performance Metrics on Validation Data*

A black background with white text

Description automatically generated

# References

Boateng, E. Y., & Abaye, D. A. (2019). A Review of the Logistic Regression Model with Emphasis on Medical Research. *Journal of Data Analysis and Information Processing*, *07*(04), 190–207. <https://doi.org/10.4236/jdaip.2019.74012>

Hendricks, R. (2022, November 13). *What is a good accuracy score in Machine Learning?* Deepchecks. <https://deepchecks.com/question/what-is-a-good-accuracy-score-in-machine-learning/#:~:text=Industry%20standards%20are%20between%2070,various%20businesses%20and%20sectors'%20needs>.

Zhang, Z. (2016). Missing data imputation: focusing on single imputation. *PubMed*, *4*(1), 9. <https://doi.org/10.3978/j.issn.2305-5839.2015.12.38>