# Practice Supervised Modelling

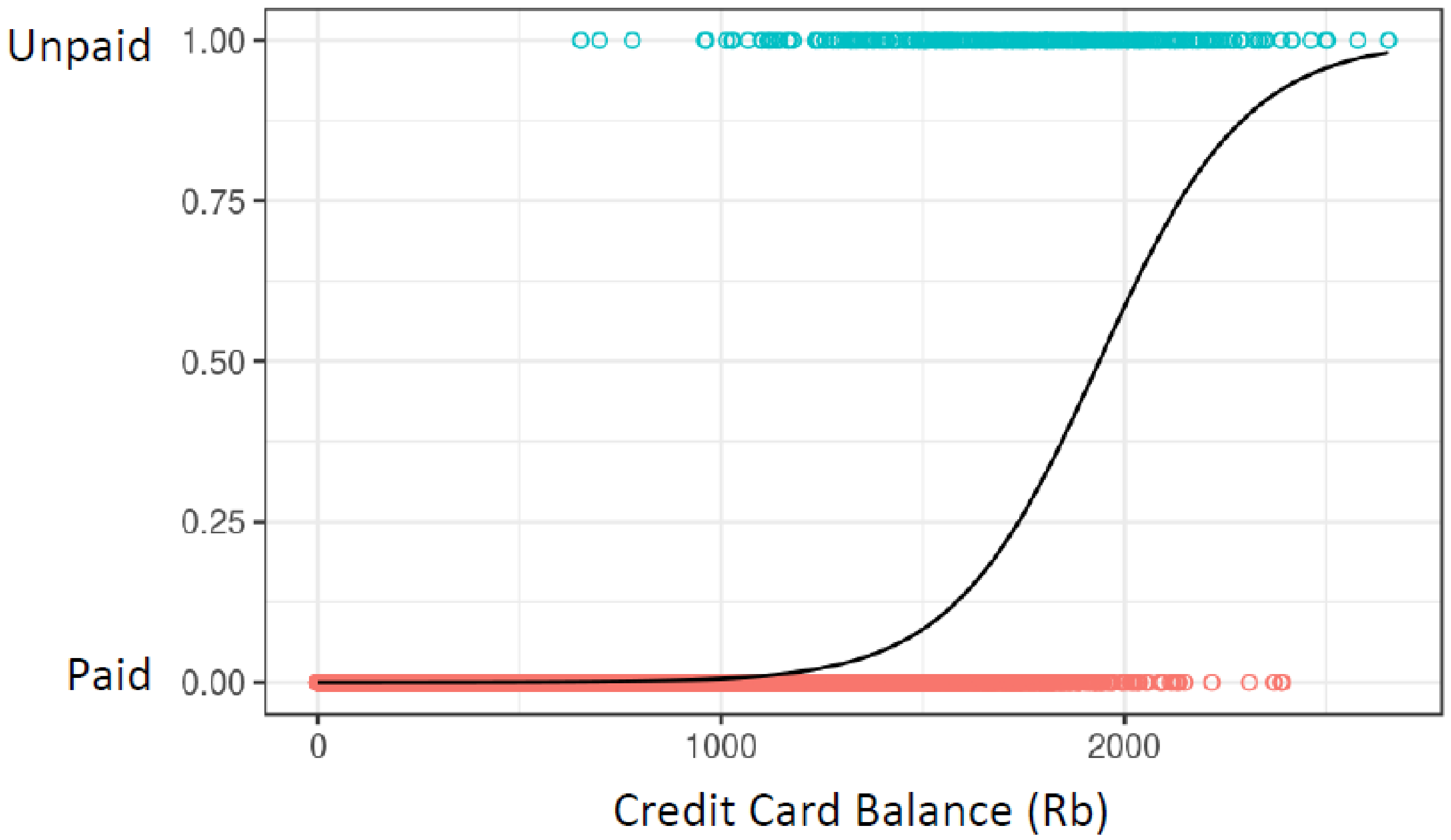
Red & White Consulting Partners LLP

# Table of Content

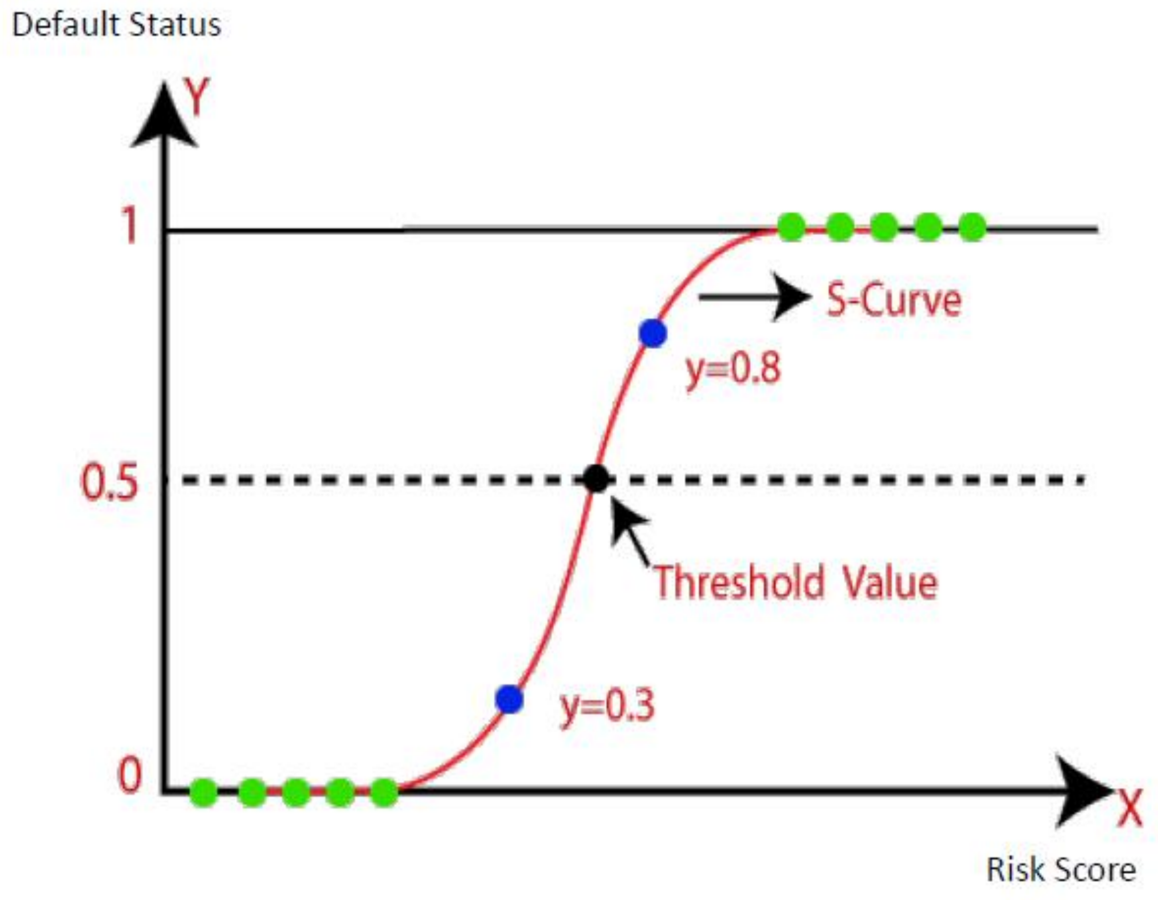
# Supervised Learning: Logistic Regression

## What is Logistic Regression

Logistic Regression models the probability of the response variable belonging to a specific class



## What is Sigmoid Function



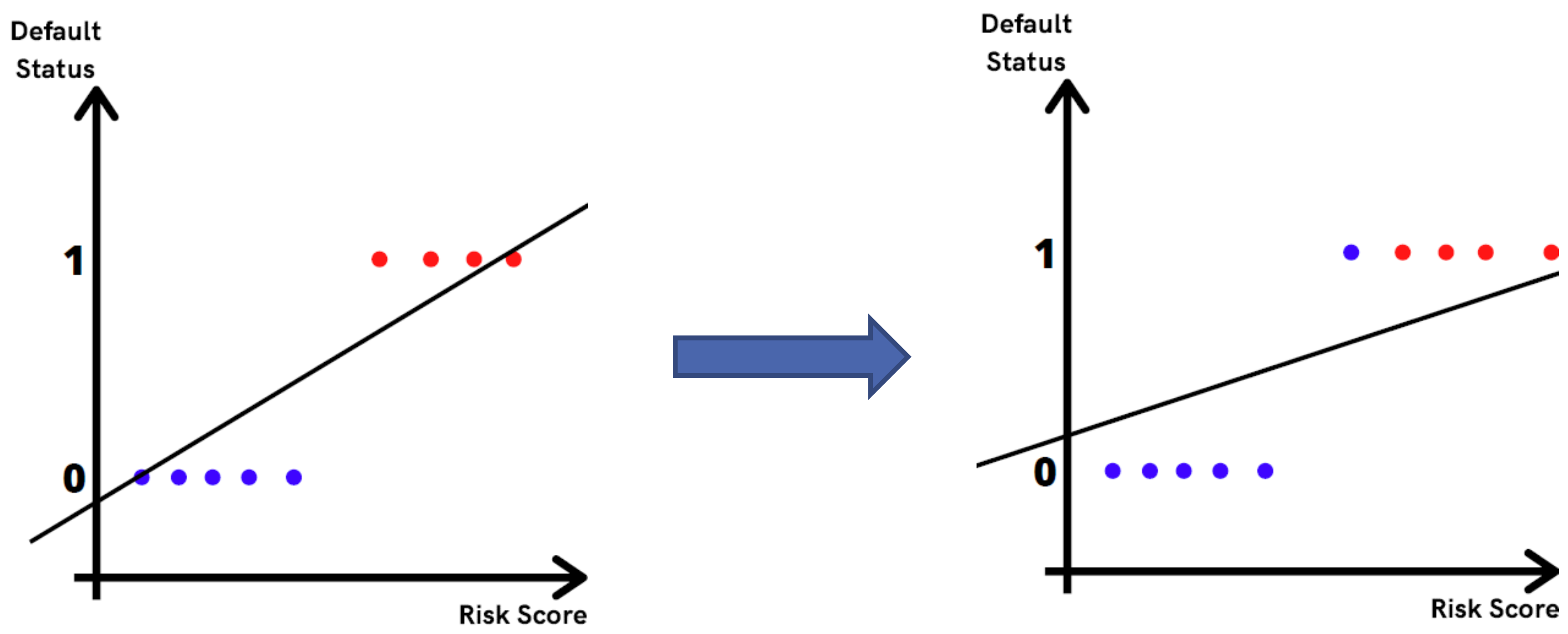
- Sigmoid or Logistic function is an S-shaped curve that is bounded by the interval [0,1]

- The sigmoid function gives the probability that Y belongs to a particular class

$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

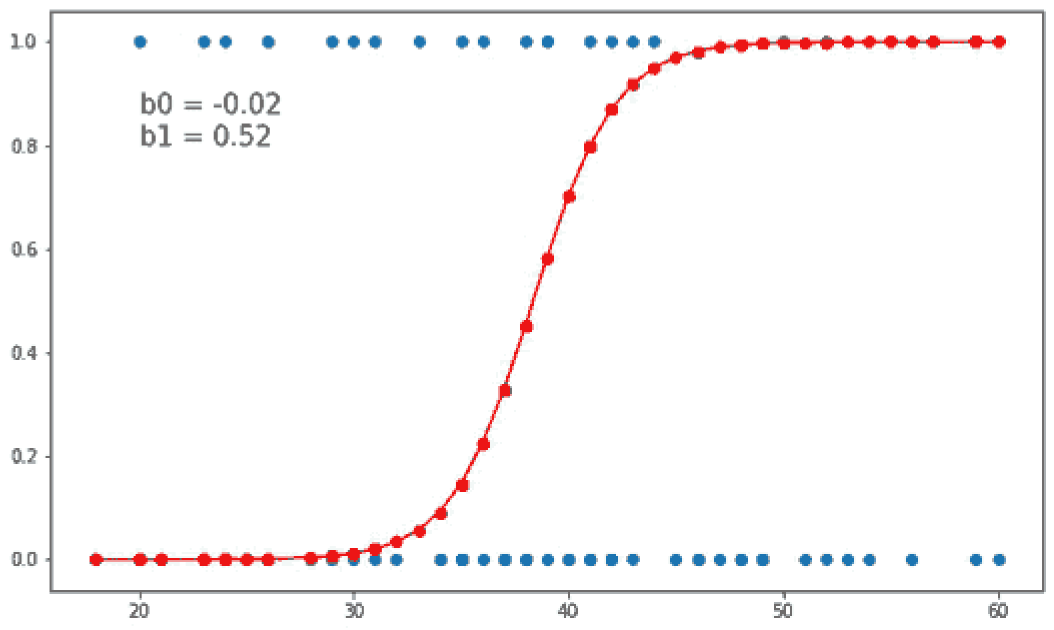
- Whenever p(x) >= 0.5; resulting y = 1
- Whenever p(x) < 0.5; resulting y = 0

## Why not using Linear Regression?



- It seems like we can also use Linear Regression to solve classification problems by assigning a threshold value
- But it gets a problem when the linear regression value is greater than 1 or less than 0 will have no meaning value for the classification
- Also, the model tends to make a misclassification when it gets new data, especially outliers

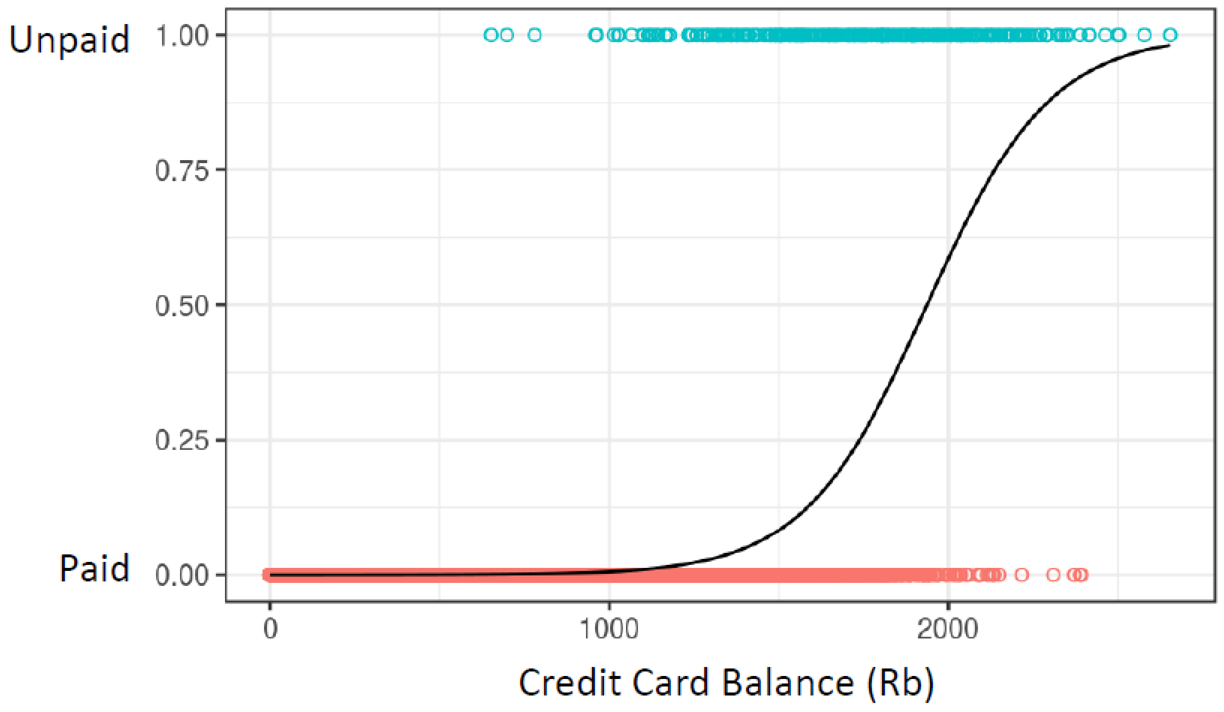
## How Logistic Regression Works using ML?



Machine Learning uses iteration to find the best model for the sigmoid function to suit with the data

## Parameter Vs. Hyperparameter

### Parameter Definition



- A parameter is a variable **estimated from the given data**
- They are required to **make predictions**
- **Not set manually** by the practitioner, meaning it will become the **output**
- Example: coefficients of Logistic Regression

$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

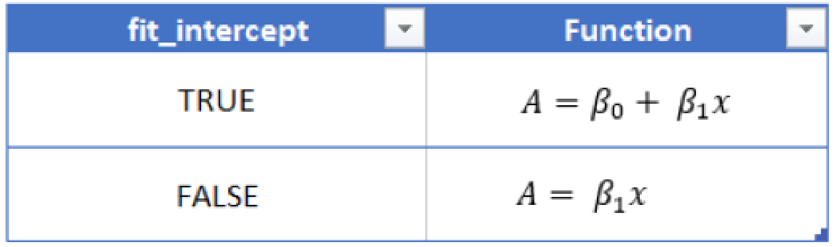
# Parameter Vs. Hyperparameter

**Hyperparameter Definition**

- Hyperparameter is used for **estimating the model parameter**
- They are **specified by the user**
- Tuned for achieving **a specific evaluation metric** such as regularization (decrease probability of overfitting), model adjustment, making the calculation simpler, etc.
- You can learn more about hyperparameter for logistic regression here

Examples:

- **fit_intercept** is a Boolean input that decides whether to include the intercept or not

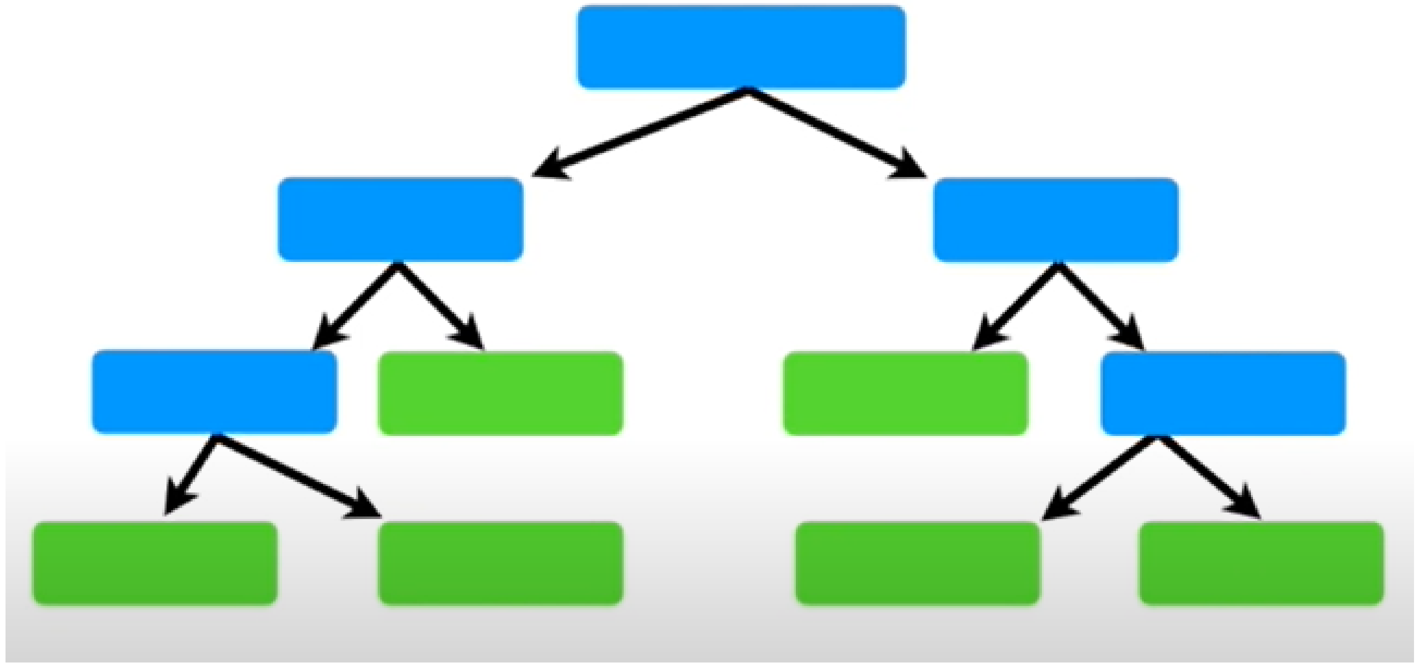


| fit_intercept | Function |
|---|---|
| TRUE | $A = \beta_0 + \beta_1 x$ |
| FALSE | $A = \beta_1 x$ |

- **multi_class** is a string that decides the approach to use for handling multiple classes (‘ovr’ by default, ‘multinomial’, ‘auto’)
- **max_iter** is an integer (100 by default) that defines the maximum number of iterations by the solver during model fitting

# Supervised Learning: Random Forest

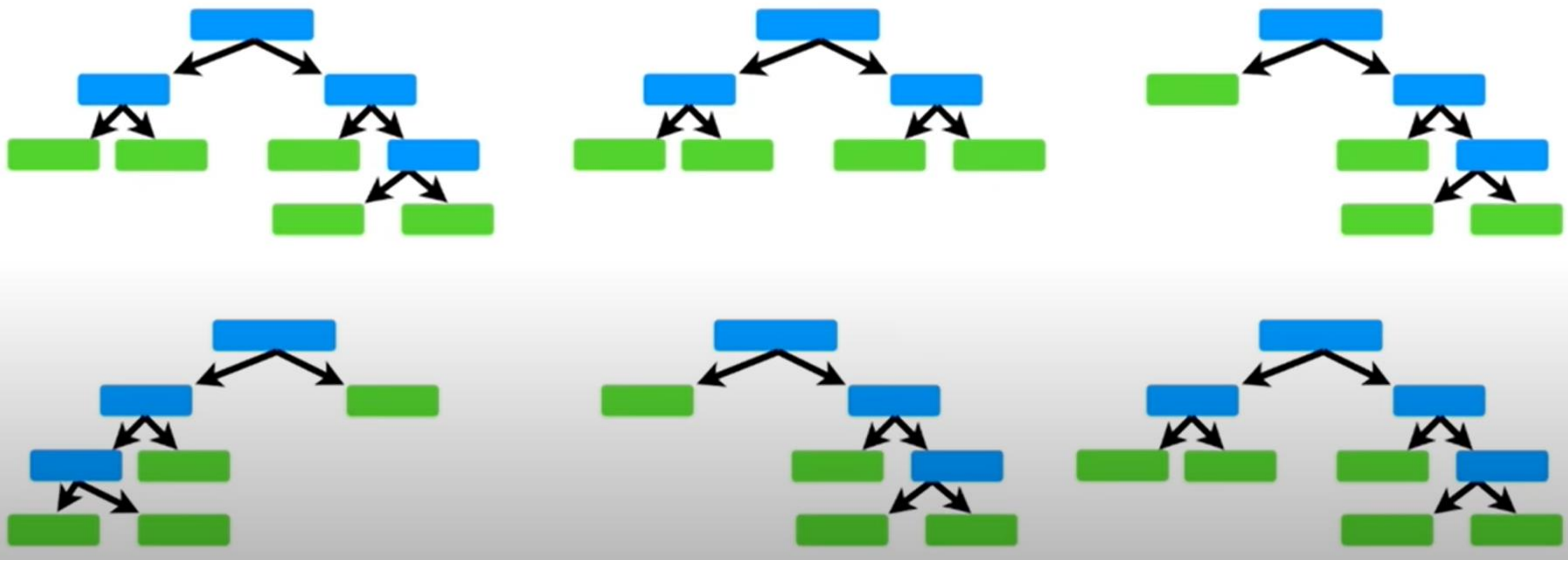
## Why Decision Tree Rarely Used for Classification



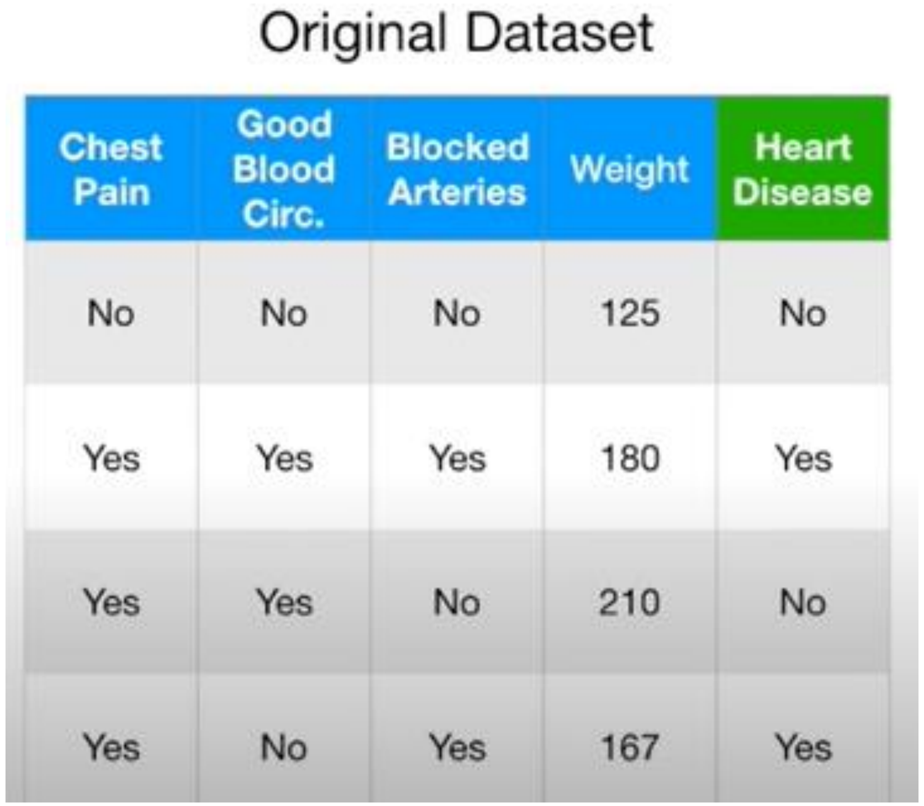
- Decision Trees are easy to build, easy to use, and easy to interpret, but it has a bad **inaccuracy**
- This is due to Decision Trees is **inflexible** when it comes to **classifying new samples**

## Why Decision Tree Rarely Used for Classification

Thus, we are using **Random Forest** by combining simple decision trees with flexibility resulting in a vast improvement in accuracy

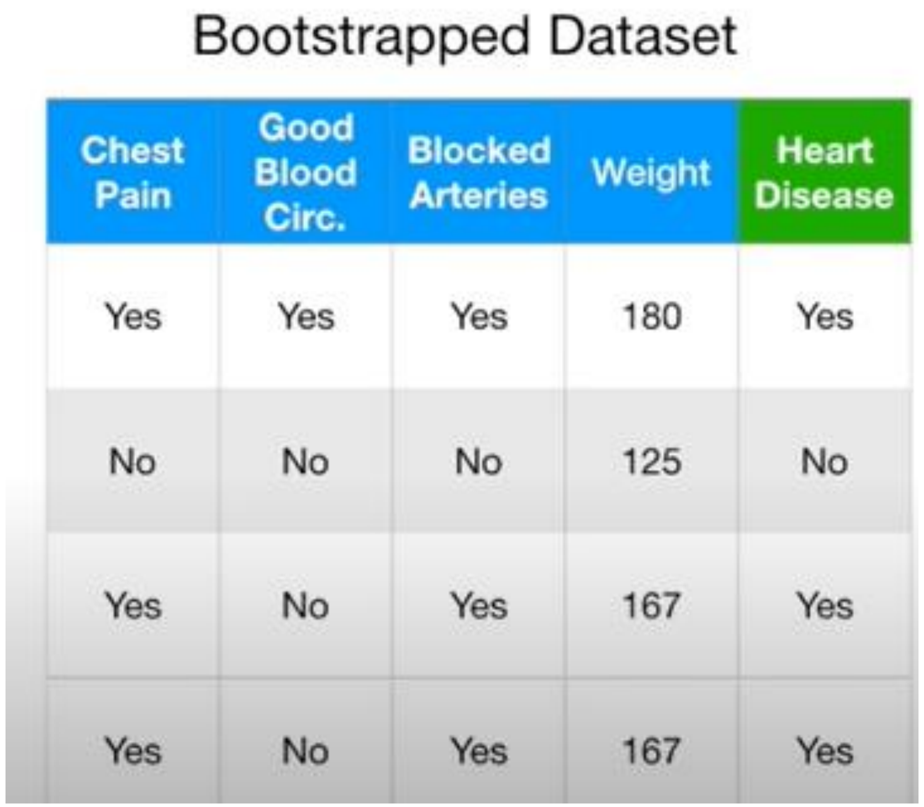


## How Random Forest Works



Original Dataset

| Chest Pain | Good Blood Circ. | Blocked Arteries | Weight | Heart Disease |
|---|---|---|---|---|
| No | No | No | 125 | No |
| Yes | Yes | Yes | 180 | Yes |
| Yes | Yes | No | 210 | No |
| Yes | No | Yes | 167 | Yes |



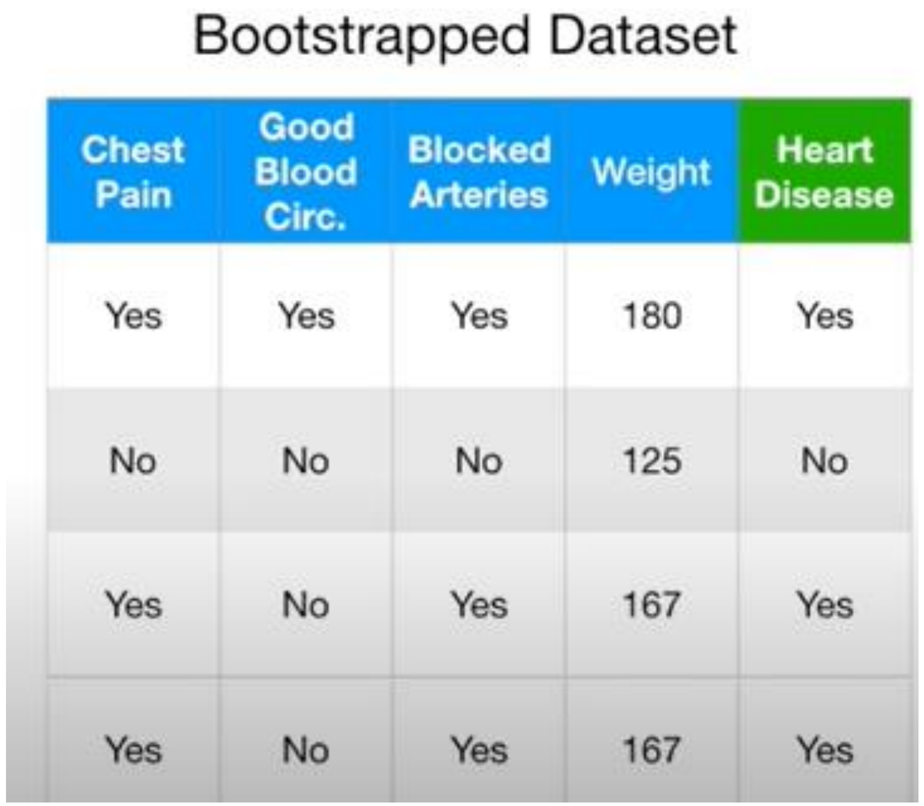
Bootstrapped Dataset

| Chest Pain | Good Blood Circ. | Blocked Arteries | Weight | Heart Disease |
|---|---|---|---|---|
| Yes | Yes | Yes | 180 | Yes |
| No | No | No | 125 | No |
| Yes | No | Yes | 167 | Yes |
| Yes | No | Yes | 167 | Yes |

- Imagine that you have the dataset on the left

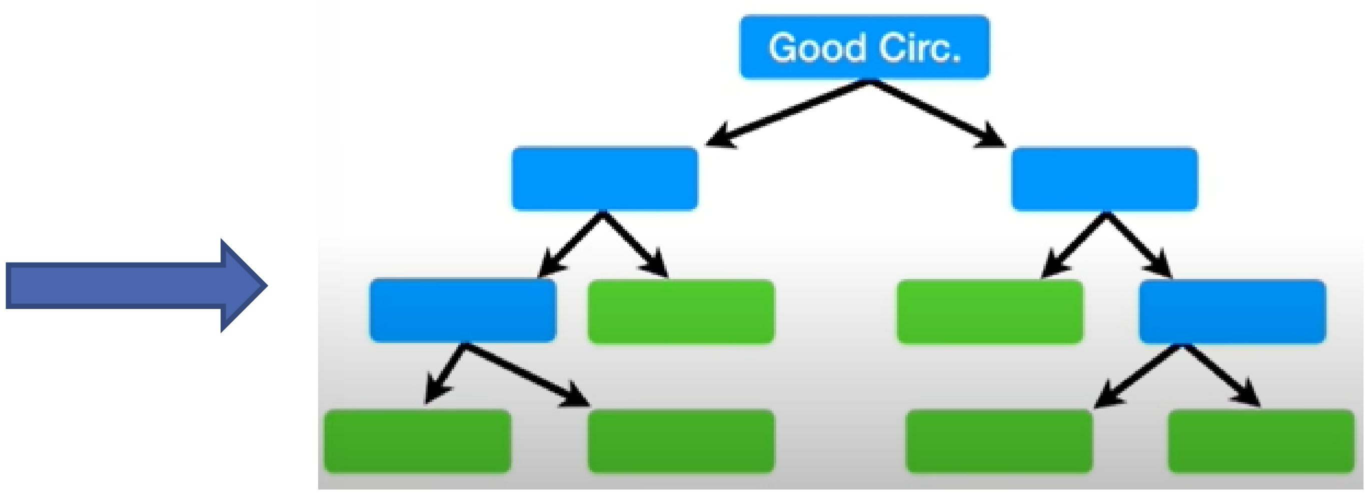
- Create a “bootstrapped” dataset by randomly select samples from original (it can be picked more than once)

## How Random Forest Works



Bootstrapped Dataset

| Chest Pain | Good Blood Circ. | Blocked Arteries | Weight | Heart Disease |
|---|---|---|---|---|
| Yes | Yes | Yes | 180 | Yes |
| No | No | No | 125 | No |
| Yes | No | Yes | 167 | Yes |
| Yes | No | Yes | 167 | Yes |

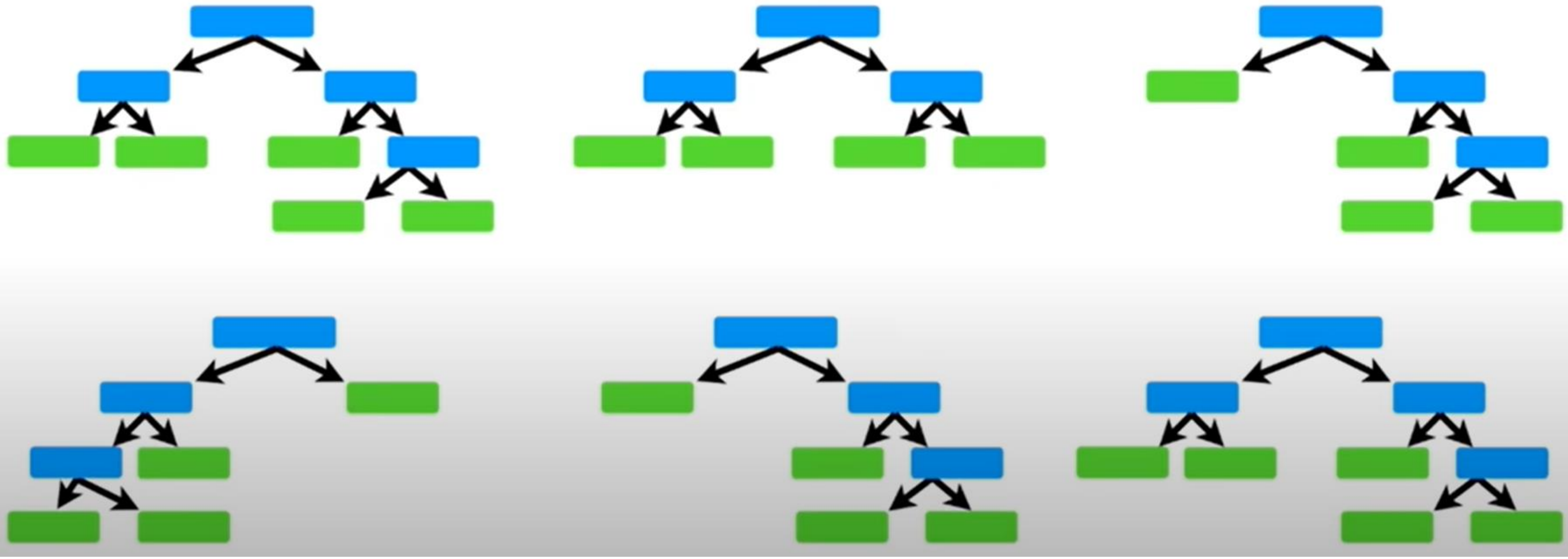


- Then, create a decision tree using the bootstrapped dataset, but only use a random subset of variables at each step

- The number of step is also limited in order to make a simpler decision tree

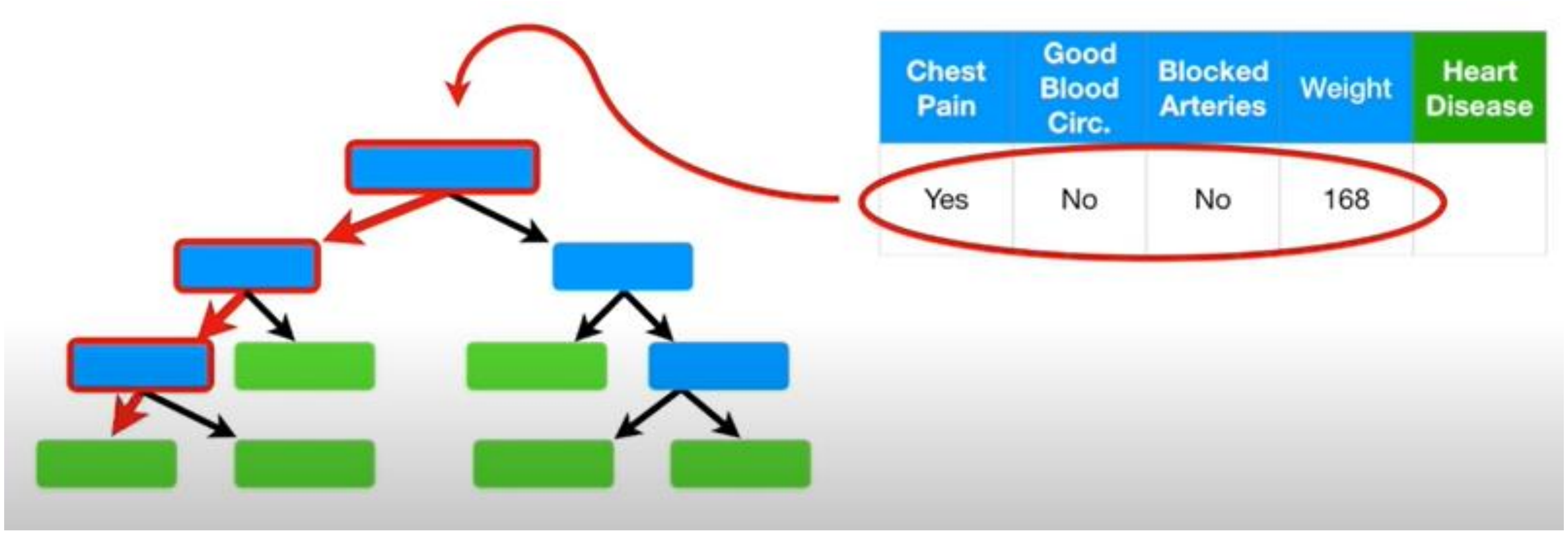
## How Random Forest Works

- Now, repeat the step from the beginning to make multiple decision trees (in default 100 times)



## How Random Forest Works

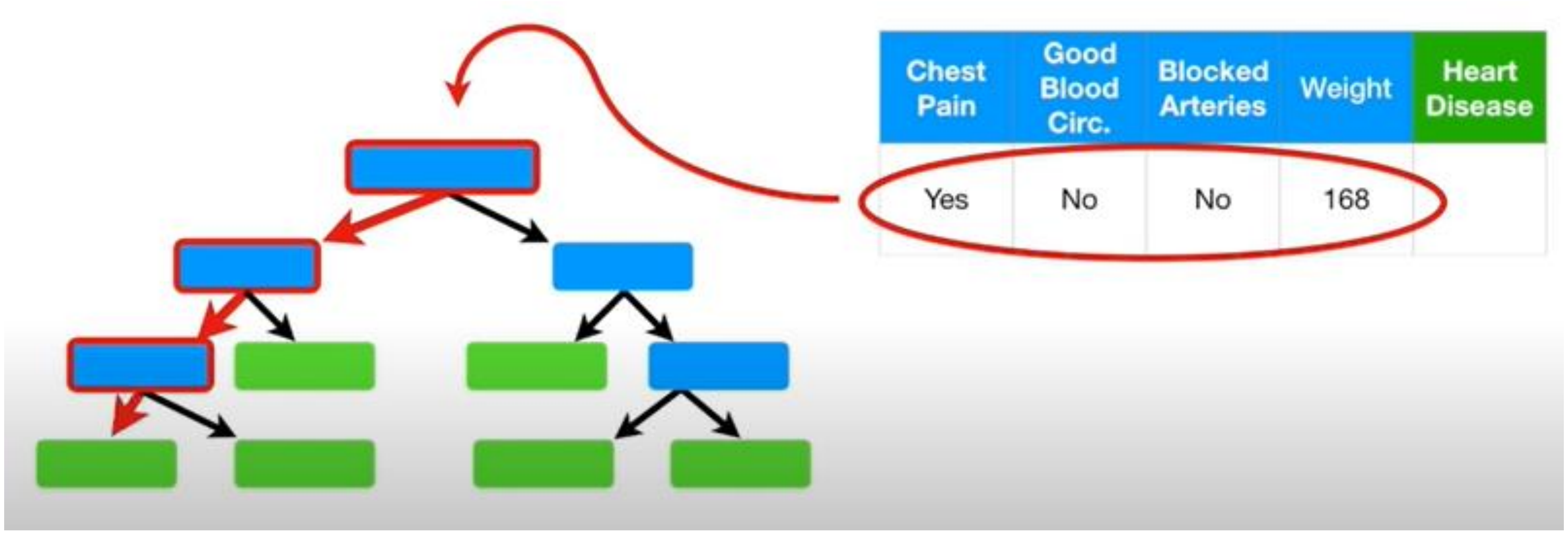
- Now back to the original data to make the predictions by running based on the first decision tree



| Chest Pain | Good Blood Circ. | Blocked Arteries | Weight | Heart Disease |
|---|---|---|---|---|
| Yes | No | No | 168 | |

## How Random Forest Works

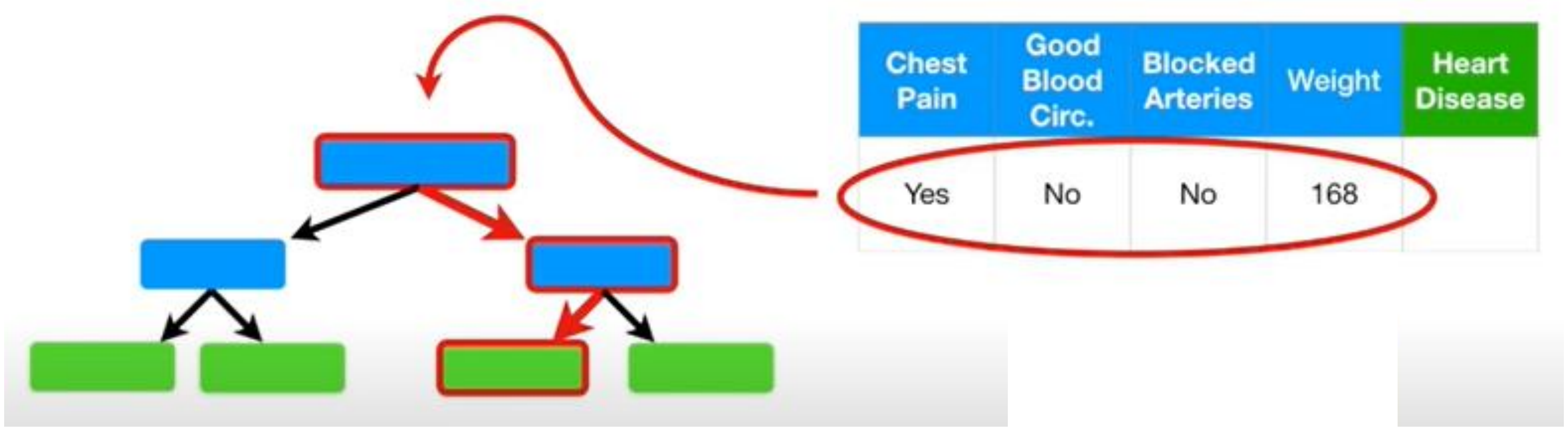
- Now back to the original data to make the predictions by running based on the first decision tree



| Chest Pain | Good Blood Circ. | Blocked Arteries | Weight | Heart Disease |
|---|---|---|---|---|
| Yes | No | No | 168 | |

## How Random Forest Works

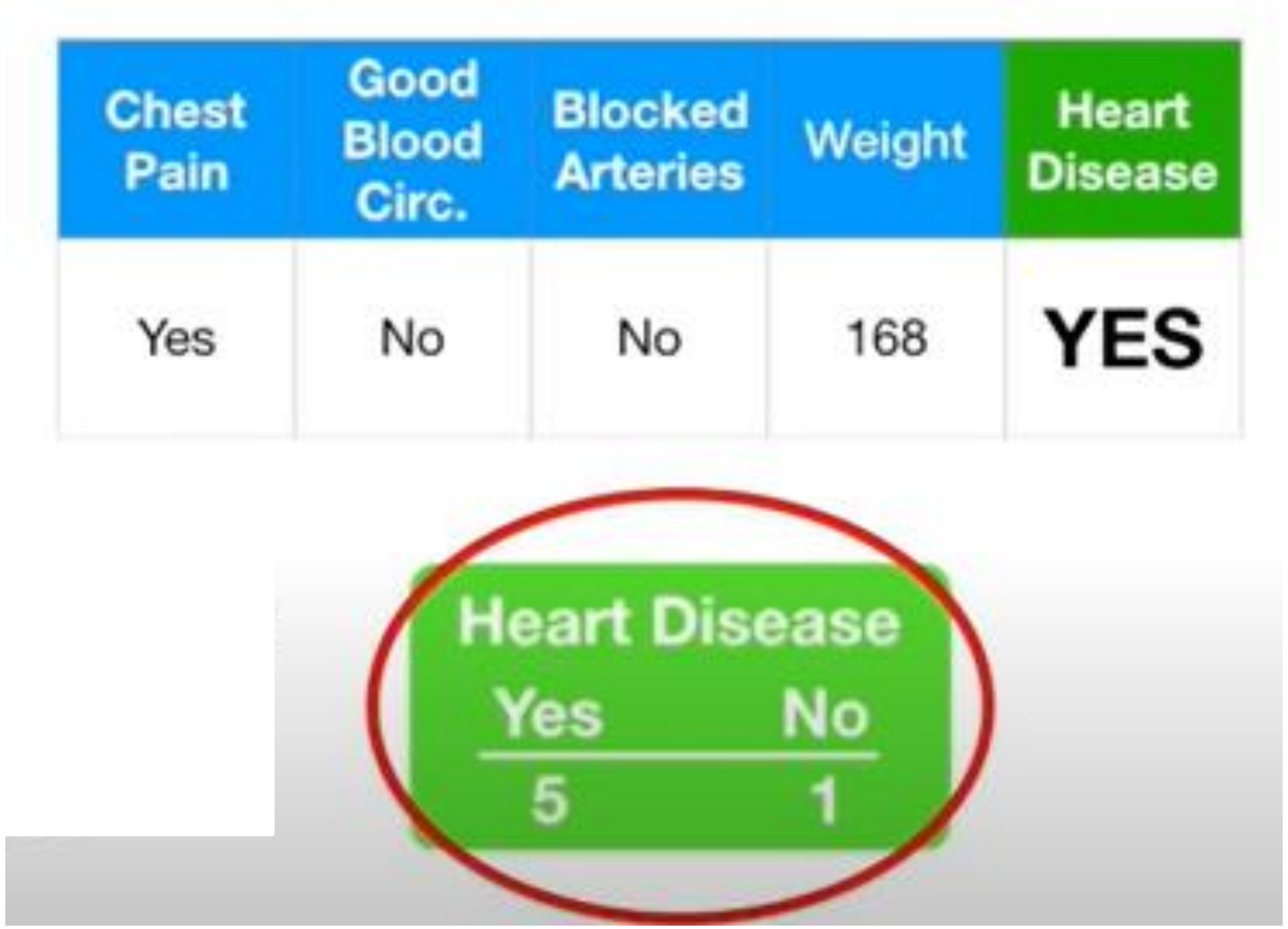
- Then, create another prediction from the second tree and so on



| Chest Pain | Good Blood Circ. | Blocked Arteries | Weight | Heart Disease |
|---|---|---|---|---|
| Yes | No | No | 168 | |

## How Random Forest Works

- It will predict as “YES” when it received the most “YES” votes from all decision trees



| Chest Pain | Good Blood Circ. | Blocked Arteries | Weight | Heart Disease |
|---|---|---|---|---|
| Yes | No | No | 168 | YES |

Heart Disease
Yes
No
5
1

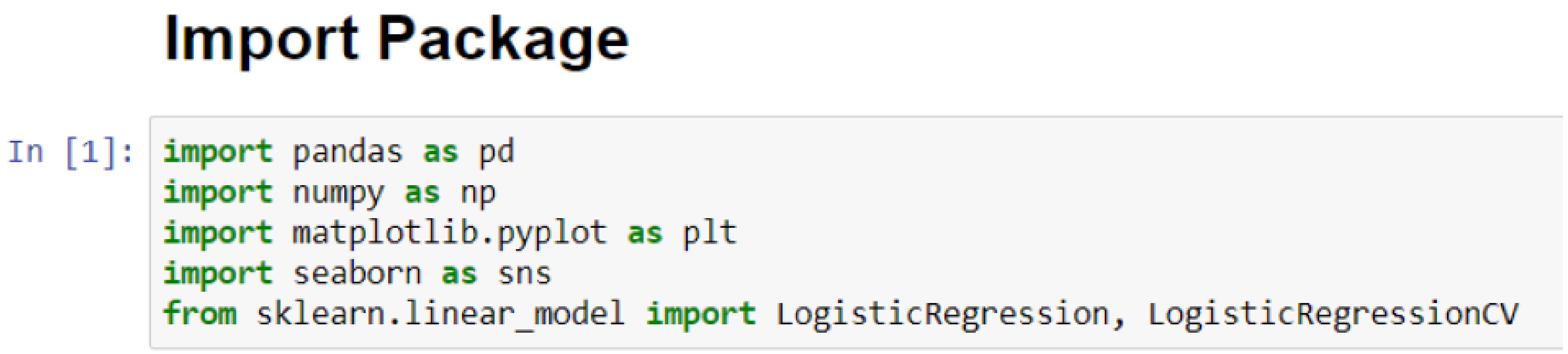
## Coffee Break

*10:00 - 10:15*

## Technical Details

## Import Package

- Import Package Required for Creating Logistic Regression

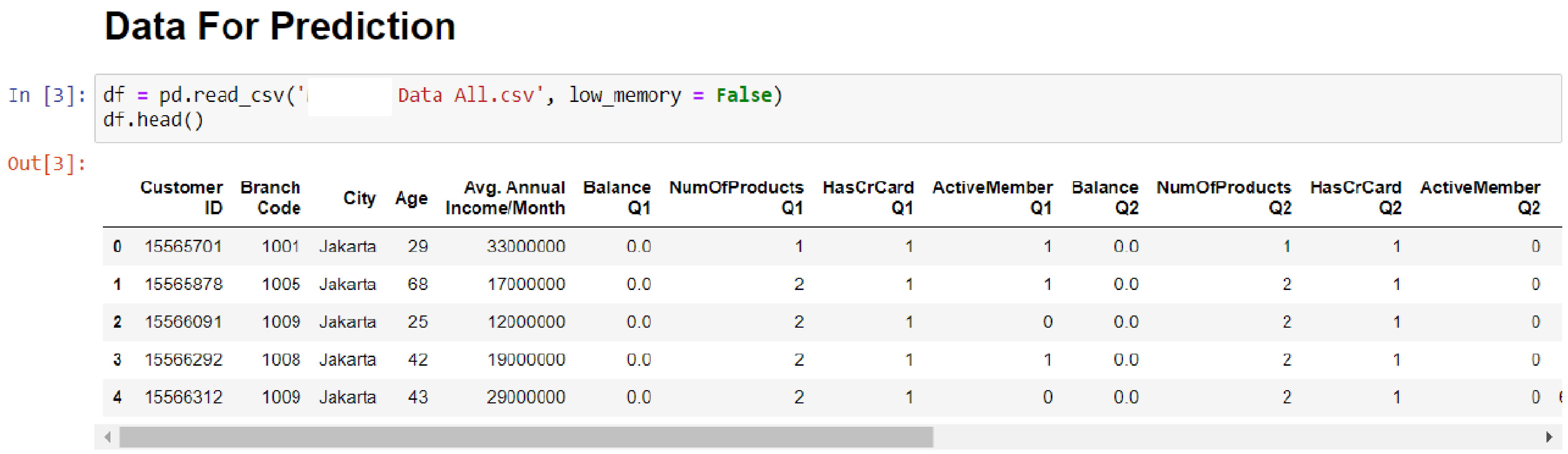


### Import Package

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
```

# Read Train Data

- Import Data Required for Creating Predictive Model.



## Data For Prediction

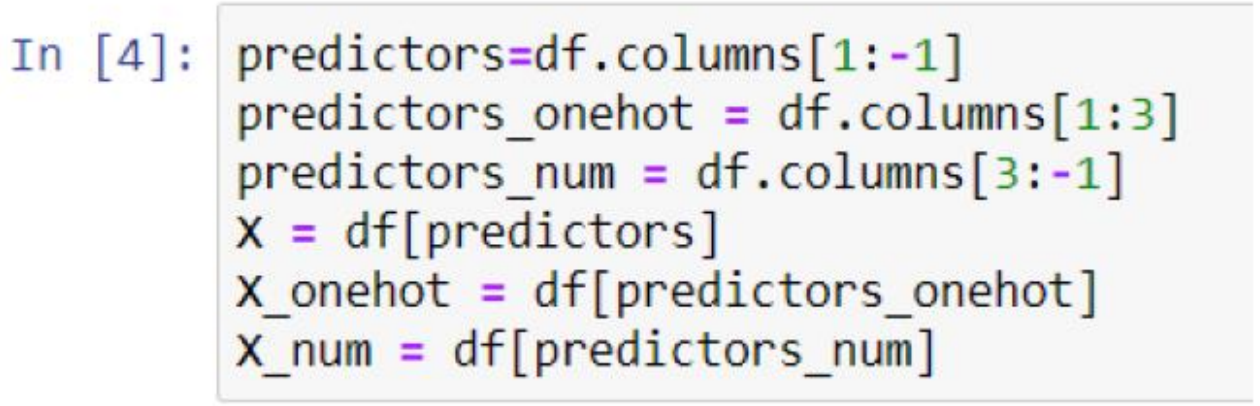
```
In [3]: df = pd.read_csv('[redacted]Data All.csv', low_memory = False)
        df.head()
```

Out[3]:

| | Customer ID | Branch Code | City | Age | Avg. Annual Income/Month | Balance Q1 | NumOfProducts Q1 | HasCrCard Q1 | ActiveMember Q1 | Balance Q2 | NumOfProducts Q2 | HasCrCard Q2 | ActiveMember Q2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 15565701 | 1001 | Jakarta | 29 | 33000000 | 0.0 | 1 | 1 | 1 | 0.0 | 1 | 1 | 0 |
| **1** | 15565878 | 1005 | Jakarta | 68 | 17000000 | 0.0 | 2 | 1 | 1 | 0.0 | 2 | 1 | 0 |
| **2** | 15566091 | 1009 | Jakarta | 25 | 12000000 | 0.0 | 2 | 1 | 0 | 0.0 | 2 | 1 | 0 |
| **3** | 15566292 | 1008 | Jakarta | 42 | 19000000 | 0.0 | 2 | 1 | 1 | 0.0 | 2 | 1 | 0 |
| **4** | 15566312 | 1009 | Jakarta | 43 | 29000000 | 0.0 | 2 | 1 | 0 | 0.0 | 2 | 1 | 0 |

## Data Separation

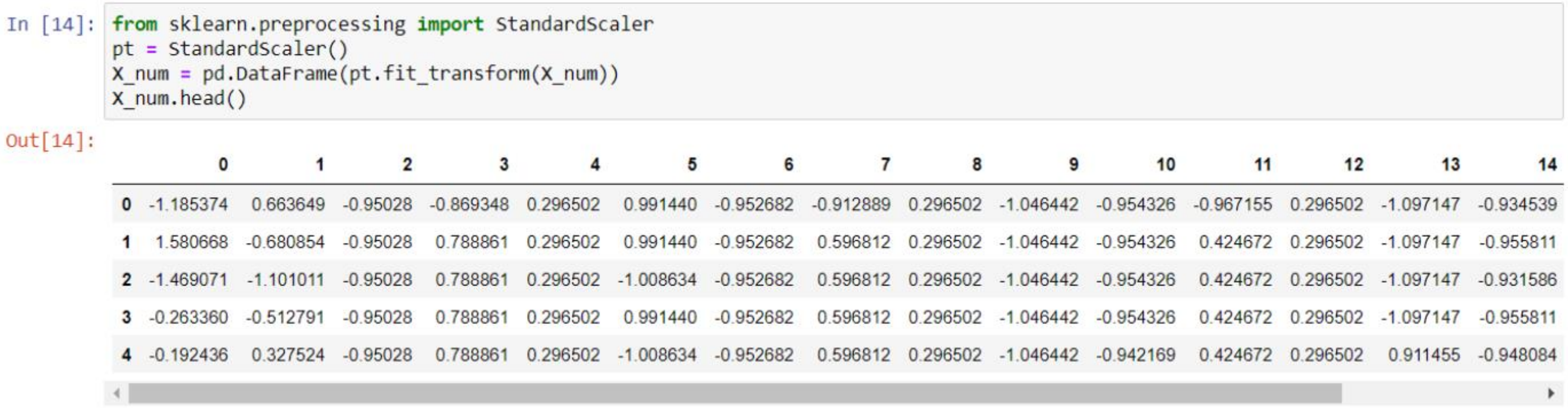
- Separate which data will be considered inside the model
- Separate between text data and numerical data as well



```
In [4]: predictors=df.columns[1:-1]
        predictors_onehot = df.columns[1:3]
        predictors_num = df.columns[3:-1]
        X = df[predictors]
        X_onehot = df[predictors_onehot]
        X_num = df[predictors_num]
```

# Standard Scaler

- Use Standard Scaler for Numerical Data



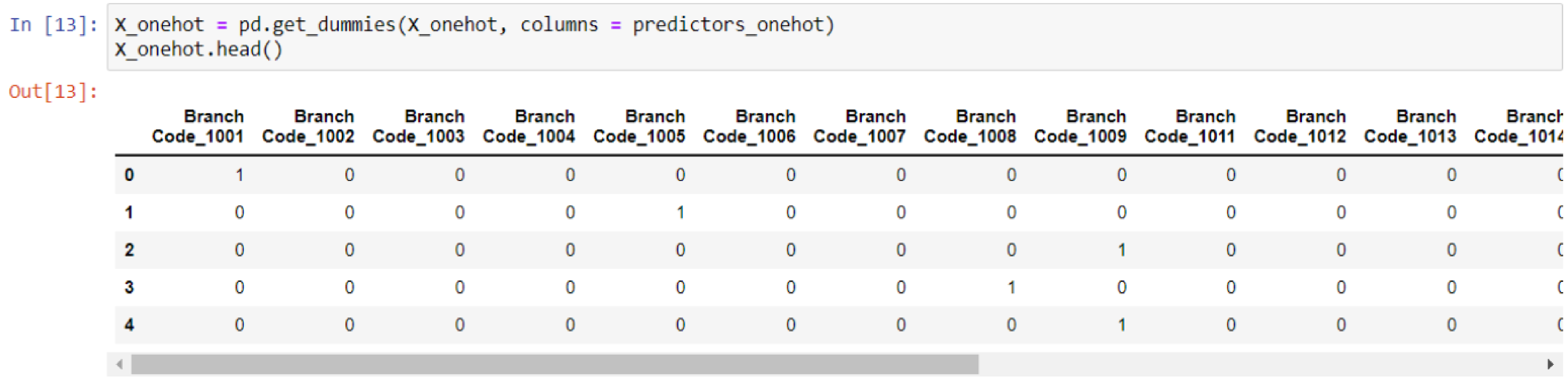
In [14]:
```
from sklearn.preprocessing import StandardScaler
pt = StandardScaler()
X_num = pd.DataFrame(pt.fit_transform(X_num))
X_num.head()
```

Out[14]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | -1.185374 | 0.663649 | -0.95028 | -0.869348 | 0.296502 | 0.991440 | -0.952682 | -0.912889 | 0.296502 | -1.046442 | -0.954326 | -0.967155 | 0.296502 | -1.097147 | -0.934539 |
| 1 | 1.580668 | -0.680854 | -0.95028 | 0.788861 | 0.296502 | 0.991440 | -0.952682 | 0.596812 | 0.296502 | -1.046442 | -0.954326 | 0.424672 | 0.296502 | -1.097147 | -0.955811 |
| 2 | -1.469071 | -1.101011 | -0.95028 | 0.788861 | 0.296502 | -1.008634 | -0.952682 | 0.596812 | 0.296502 | -1.046442 | -0.954326 | 0.424672 | 0.296502 | -1.097147 | -0.931586 |
| 3 | -0.263360 | -0.512791 | -0.95028 | 0.788861 | 0.296502 | 0.991440 | -0.952682 | 0.596812 | 0.296502 | -1.046442 | -0.954326 | 0.424672 | 0.296502 | -1.097147 | -0.955811 |
| 4 | -0.192436 | 0.327524 | -0.95028 | 0.788861 | 0.296502 | -1.008634 | -0.952682 | 0.596812 | 0.296502 | -1.046442 | -0.942169 | 0.424672 | 0.296502 | 0.911455 | -0.948084 |

# One-Hot Encoder

- Use One-Hot Encoder for Categorical Data



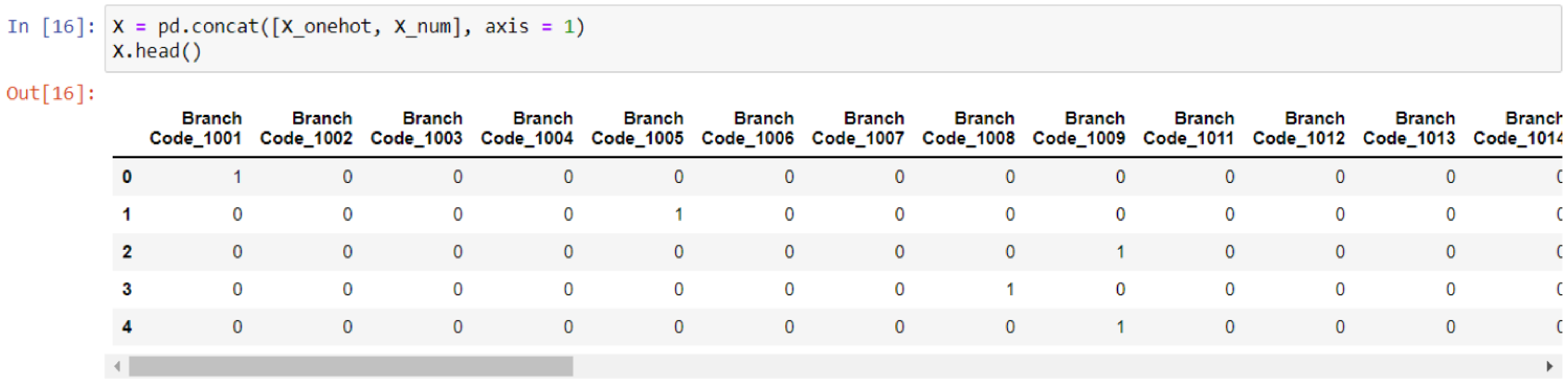
```
In [13]: X_onehot = pd.get_dummies(X_onehot, columns = predictors_onehot)
         X_onehot.head()
```

Out[13]:

| | Branch Code_1001 | Branch Code_1002 | Branch Code_1003 | Branch Code_1004 | Branch Code_1005 | Branch Code_1006 | Branch Code_1007 | Branch Code_1008 | Branch Code_1009 | Branch Code_1011 | Branch Code_1012 | Branch Code_1013 | Branc Code_101 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| **3** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| **4** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

# Combine Numerical & Categorical Data

- Combine both Numerical & Categorical Data



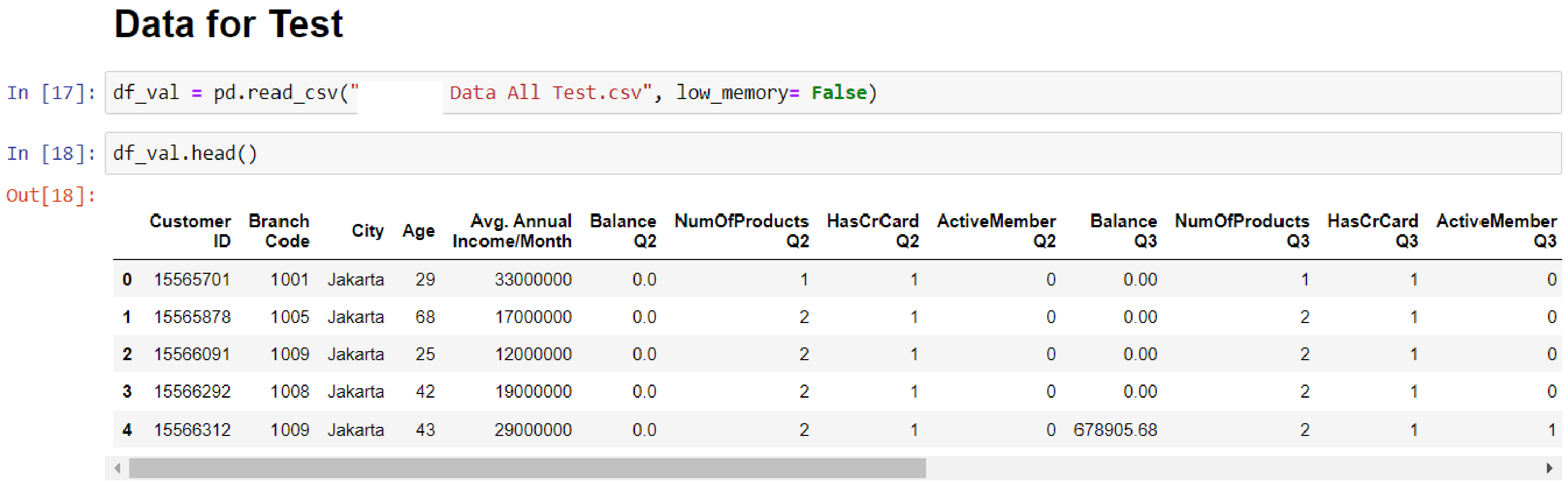
```
In [16]: X = pd.concat([X_onehot, X_num], axis = 1)
         X.head()
```

Out[16]:

| | Branch Code_1001 | Branch Code_1002 | Branch Code_1003 | Branch Code_1004 | Branch Code_1005 | Branch Code_1006 | Branch Code_1007 | Branch Code_1008 | Branch Code_1009 | Branch Code_1011 | Branch Code_1012 | Branch Code_1013 | Branch Code_101 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ( |
| **1** | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ( |
| **2** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | ( |
| **3** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | ( |
| **4** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | ( |

## Data Test

- Do with the Test Data As Well



### Data for Test

```
In [17]: df_val = pd.read_csv("[redacted]Data All Test.csv", low_memory= False)
```

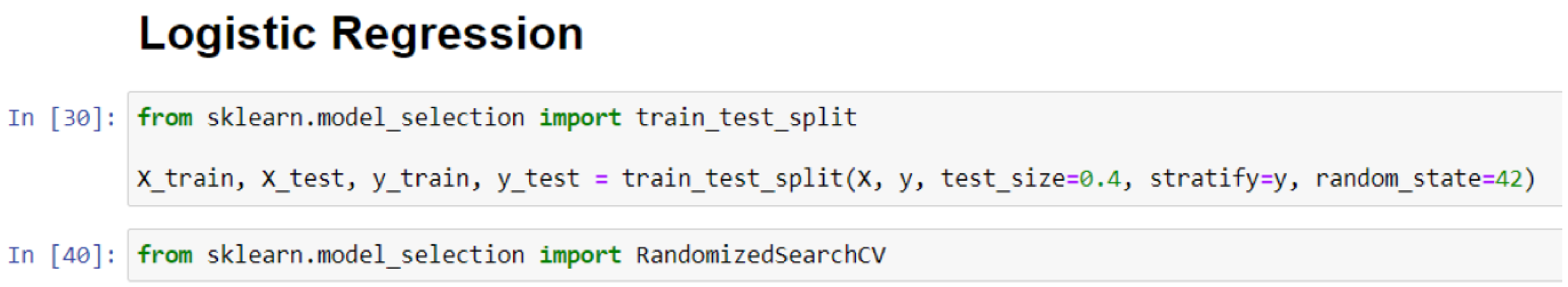
```
In [18]: df_val.head()
```

Out[18]:

| | Customer ID | Branch Code | City | Age | Avg. Annual Income/Month | Balance Q2 | NumOfProducts Q2 | HasCrCard Q2 | ActiveMember Q2 | Balance Q3 | NumOfProducts Q3 | HasCrCard Q3 | ActiveMember Q3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 15565701 | 1001 | Jakarta | 29 | 33000000 | 0.0 | 1 | 1 | 0 | 0.00 | 1 | 1 | 0 |
| **1** | 15565878 | 1005 | Jakarta | 68 | 17000000 | 0.0 | 2 | 1 | 0 | 0.00 | 2 | 1 | 0 |
| **2** | 15566091 | 1009 | Jakarta | 25 | 12000000 | 0.0 | 2 | 1 | 0 | 0.00 | 2 | 1 | 0 |
| **3** | 15566292 | 1008 | Jakarta | 42 | 19000000 | 0.0 | 2 | 1 | 0 | 0.00 | 2 | 1 | 0 |
| **4** | 15566312 | 1009 | Jakarta | 43 | 29000000 | 0.0 | 2 | 1 | 0 | 678905.68 | 2 | 1 | 1 |

# Import Package for Logistic Regression

- Import package for Logistic Regression

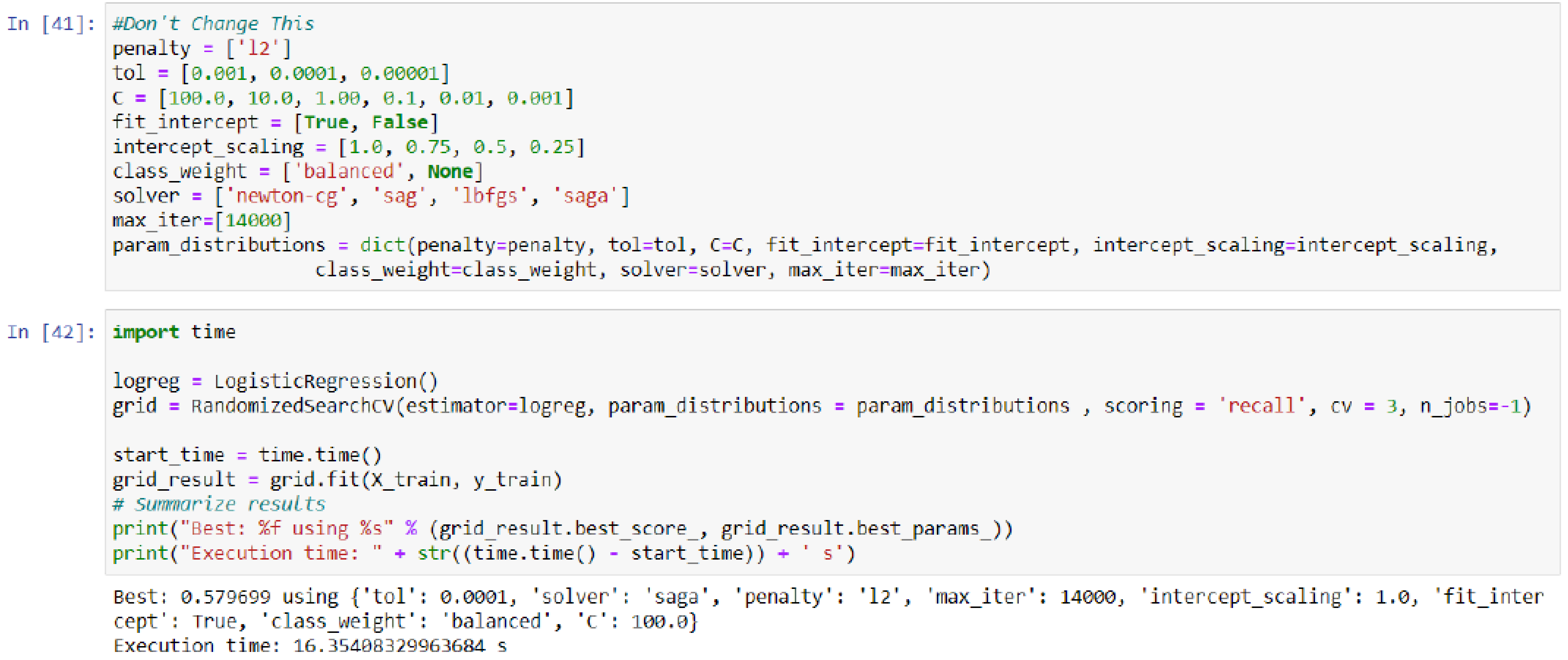


## Logistic Regression

```
In [30]: from sklearn.model_selection import train_test_split

         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, stratify=y, random_state=42)

In [40]: from sklearn.model_selection import RandomizedSearchCV
```

## Building Logistic Regression Model

- Calculate the data in Machine Learning using Logistic Regression

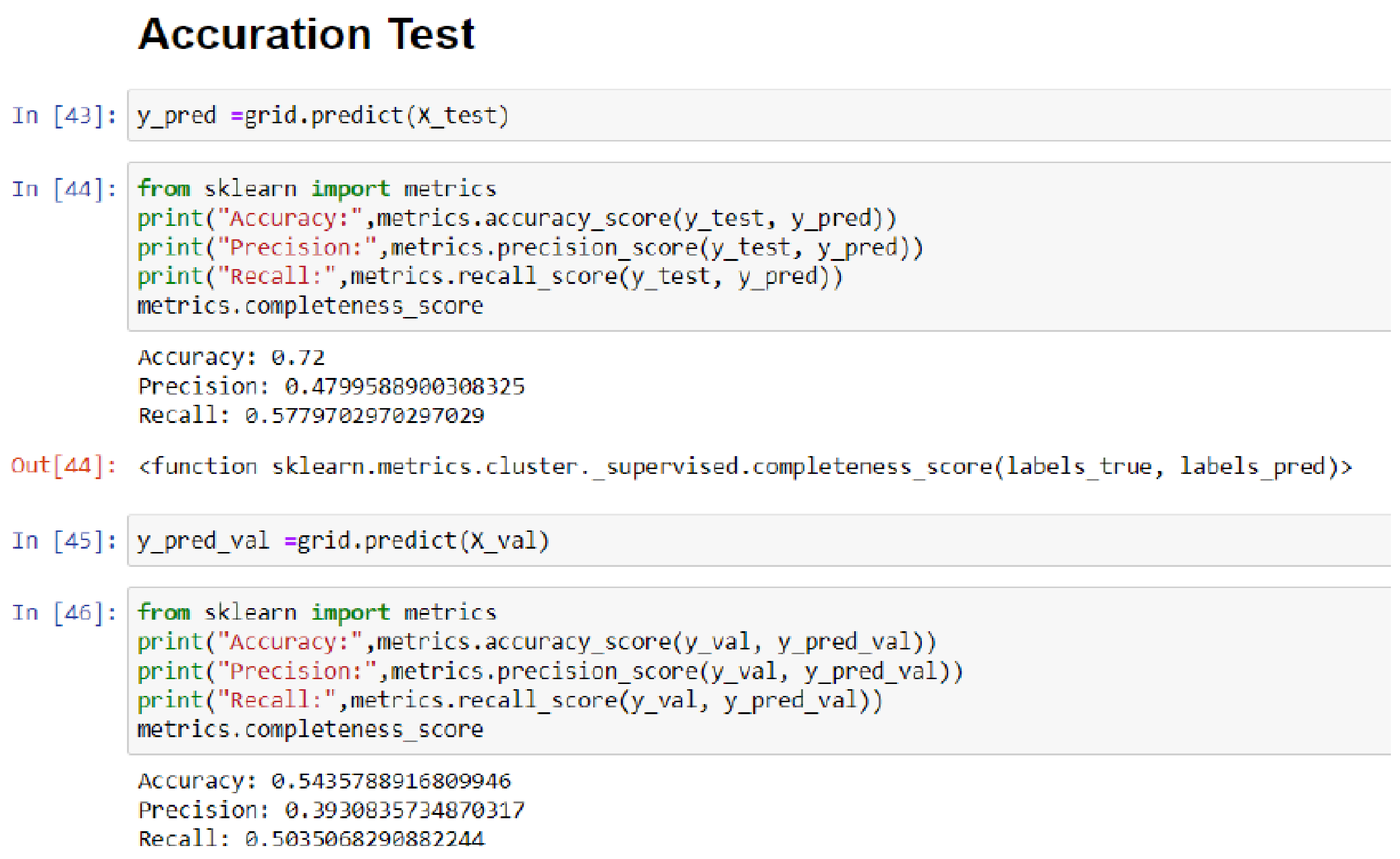


```
In [41]: #Don't Change This
         penalty = ['l2']
         tol = [0.001, 0.0001, 0.00001]
         C = [100.0, 10.0, 1.00, 0.1, 0.01, 0.001]
         fit_intercept = [True, False]
         intercept_scaling = [1.0, 0.75, 0.5, 0.25]
         class_weight = ['balanced', None]
         solver = ['newton-cg', 'sag', 'lbfgs', 'saga']
         max_iter=[14000]
         param_distributions = dict(penalty=penalty, tol=tol, C=C, fit_intercept=fit_intercept, intercept_scaling=intercept_scaling,
                                    class_weight=class_weight, solver=solver, max_iter=max_iter)
```

```
In [42]: import time

         logreg = LogisticRegression()
         grid = RandomizedSearchCV(estimator=logreg, param_distributions = param_distributions , scoring = 'recall', cv = 3, n_jobs=-1)

         start_time = time.time()
         grid_result = grid.fit(X_train, y_train)
         # Summarize results
         print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
         print("Execution time: " + str((time.time() - start_time)) + ' s')
```

```
Best: 0.579699 using {'tol': 0.0001, 'solver': 'saga', 'penalty': 'l2', 'max_iter': 14000, 'intercept_scaling': 1.0, 'fit_inter
cept': True, 'class_weight': 'balanced', 'C': 100.0}
Execution time: 16.35408329963684 s
```

## Calculate the Evaluation Model



### Accuration Test

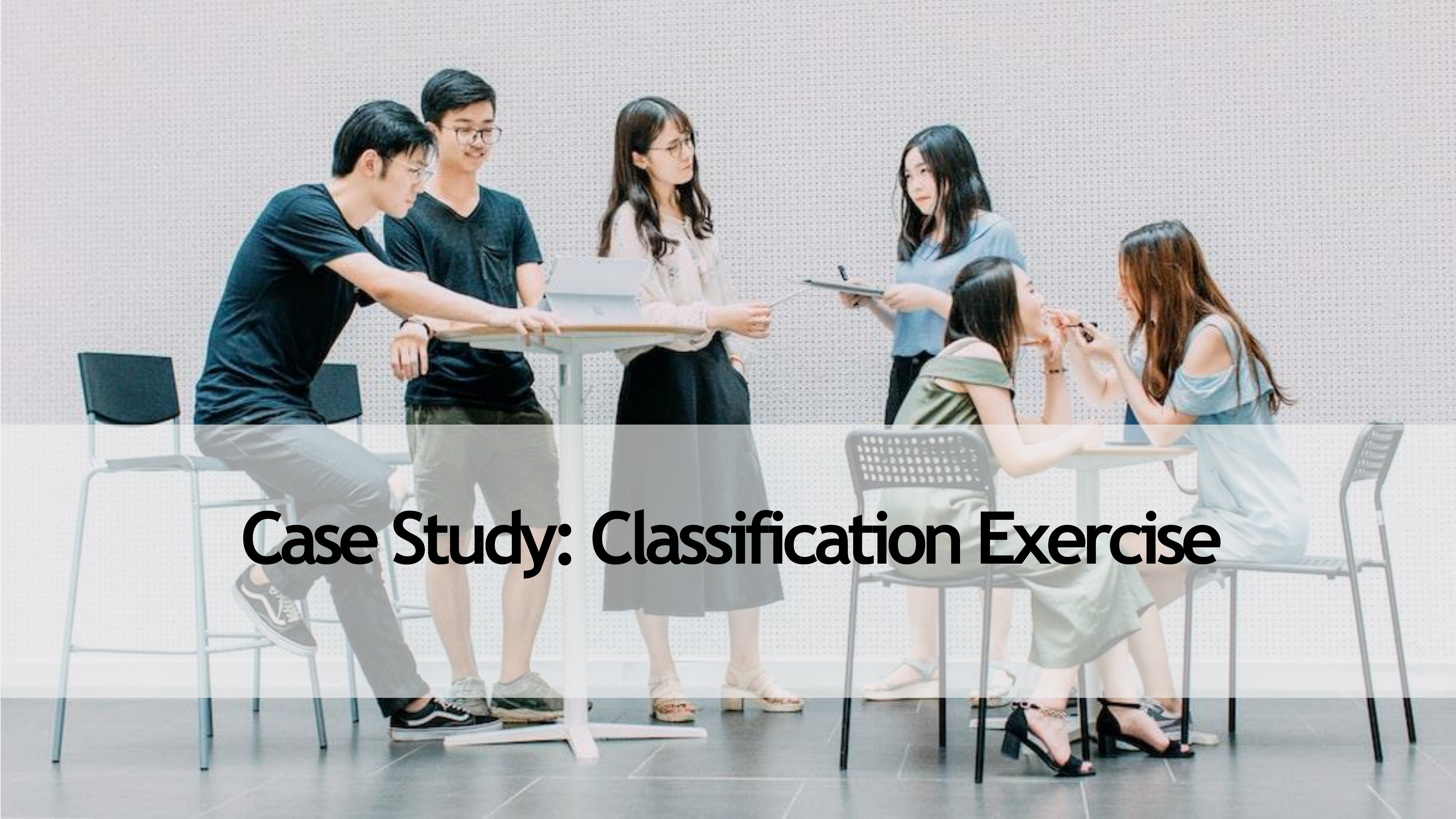
```
In [43]: y_pred =grid.predict(X_test)

In [44]: from sklearn import metrics
         print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
         print("Precision:",metrics.precision_score(y_test, y_pred))
         print("Recall:",metrics.recall_score(y_test, y_pred))
         metrics.completeness_score

         Accuracy: 0.72
         Precision: 0.4799588900308325
         Recall: 0.5779702970297029

Out[44]: <function sklearn.metrics.cluster._supervised.completeness_score(labels_true, labels_pred)>

In [45]: y_pred_val =grid.predict(X_val)

In [46]: from sklearn import metrics
         print("Accuracy:",metrics.accuracy_score(y_val, y_pred_val))
         print("Precision:",metrics.precision_score(y_val, y_pred_val))
         print("Recall:",metrics.recall_score(y_val, y_pred_val))
         metrics.completeness_score

         Accuracy: 0.5435788916809946
         Precision: 0.3930835734870317
         Recall: 0.5035068290882244
```

- Calculate our results for accuracy, precision, and recall.

Which one we want to consider? Why?

## Ishoma

*12:00 - 13:00*



## Case Study: Classification Exercise

## Let's Share!

# Q & A

Thank you