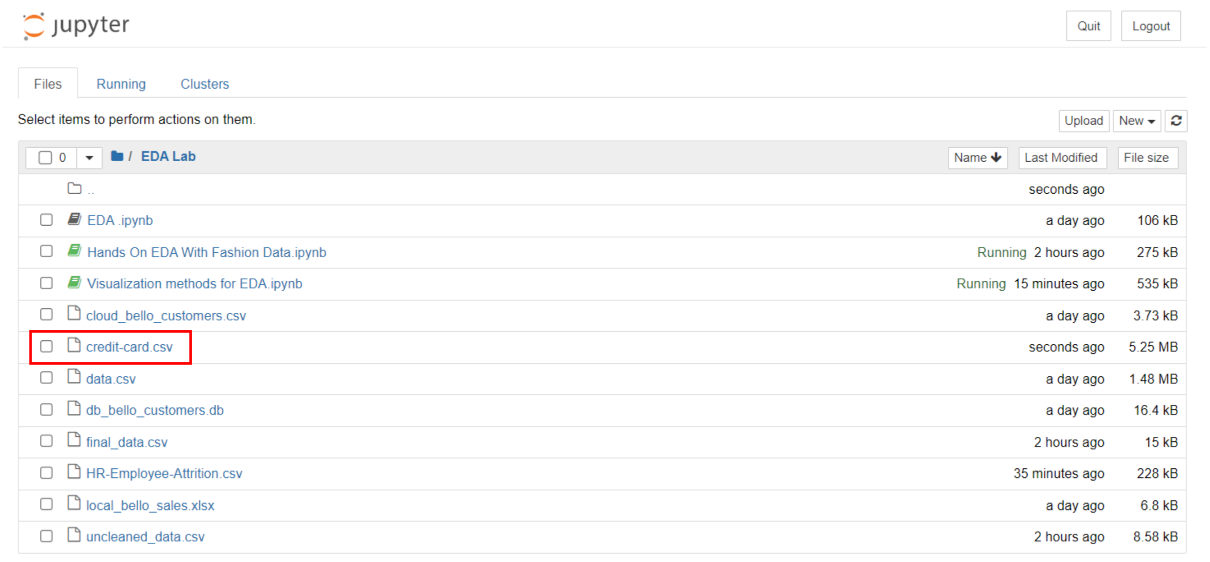
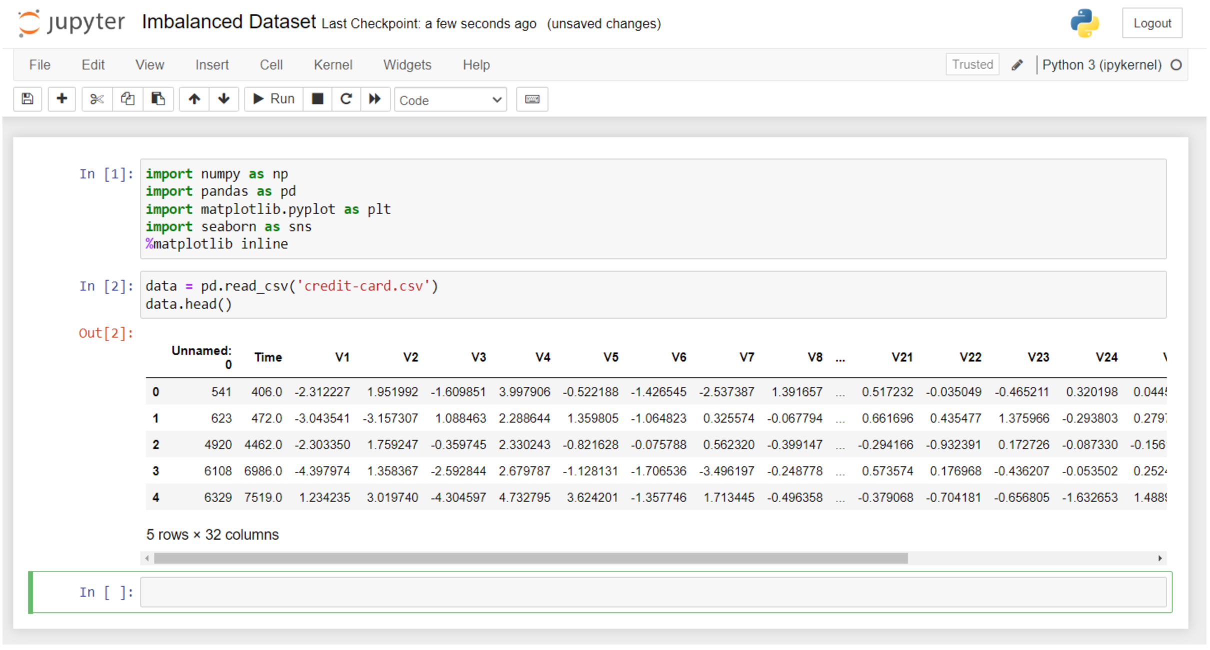
😊 Imbalanced Dataset

1. For this lab you need to upload a new file to your EDA Lab folder in Jupyter Notebook with the name credit-card.csv as you can see below.



```
jupyter                                                        Quit  Logout

Files   Running   Clusters

Select items to perform actions on them.                  Upload  New ▾  ⟳

☐ 0 ▾  📁 / EDA Lab                    Name ↓   Last Modified   File size
  ..                                                  seconds ago
☐ EDA .ipynb                                          a day ago      106 kB
☐ Hands On EDA With Fashion Data.ipynb       Running  2 hours ago    275 kB
☐ Visualization methods for EDA.ipynb        Running  15 minutes ago 535 kB
☐ cloud_bello_customers.csv                           a day ago      3.73 kB
☐ credit-card.csv                                     seconds ago    5.25 MB
☐ data.csv                                            a day ago      1.48 MB
☐ db_bello_customers.db                               a day ago      16.4 kB
☐ final_data.csv                                      2 hours ago    15 kB
☐ HR-Employee-Attrition.csv                           35 minutes ago 228 kB
☐ local_bello_sales.xlsx                              a day ago      6.8 kB
☐ uncleaned_data.csv                                  2 hours ago    8.58 kB
```

2. After that create a new Python 3 kernel in your Notebook. So, the first thing we need to do is import some important libraries as you can see below then load our CSV file in the notebook then read the data from it.



jupyter Imbalanced Dataset Last Checkpoint: a few seconds ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        %matplotlib inline

In [2]: data = pd.read_csv('credit-card.csv')
        data.head()

Out[2]:
   Unnamed:   Time        V1        V2        V3        V4        V5        V6        V7        V8  ...       V21       V22       V23       V24    V
          0
0       541  406.0 -2.312227  1.951992 -1.609851  3.997906 -0.522188 -1.426545 -2.537387  1.391657  ...  0.517232 -0.035049 -0.465211  0.320198  0.044
1       623  472.0 -3.043541 -3.157307  1.088463  2.288644  1.359805 -1.064823  0.325574 -0.067794  ...  0.661696  0.435477  1.375966 -0.293803  0.279
2      4920 4462.0 -2.303350  1.759247 -0.359745  2.330243 -0.821628 -0.075788  0.562320 -0.399147  ... -0.294166 -0.932391  0.172726 -0.087330 -0.156
3      6108 6986.0 -4.397974  1.358367 -2.592844  2.679787 -1.128131 -1.706536 -3.496197 -0.248778  ...  0.573574  0.176968 -0.436207 -0.053502  0.252
4      6329 7519.0  1.234235  3.019740 -4.304597  4.732795  3.624201 -1.357746  1.713445 -0.496358  ... -0.379068 -0.704181 -0.656805 -1.632653  1.488

5 rows × 32 columns

In [ ]:
```

3. If you look at the dataset, we also have a column labelled **Class**, which indicates whether the credit card has been approved or not. Let's analyze this column to understand its distribution:
   Value Counts: By examining the value counts of the Class column, we find that there are **9,000 rows** with a value of **0**, indicating that these credit cards were not approved.

Conversely, there are **492 rows** with a value of **1**, signifying that these credit cards were approved.

4. This distribution suggests a significant imbalance between the two classes, with a much higher number of unapproved credit cards compared to approved ones. This information is crucial for understanding the dataset and may impact the choice of analysis methods or algorithms used in modeling.

```
In [3]: data['Class'].value_counts()

Out[3]: Class
        0    9000
        1     492
        Name: count, dtype: int64
```

5. To address imbalanced data, one common approach is to perform **undersampling** and **oversampling**.

- **Undersampling**: This technique involves reducing the number of instances in the majority class (in this case, the unapproved credit cards) to balance the class distribution. While this can help reduce bias in the model, it may lead to the loss of potentially valuable information.
- **Oversampling**: This method involves increasing the number of instances in the minority class (approved credit cards) to create a more balanced dataset. This can be done by duplicating existing instances or generating synthetic examples (e.g., using techniques like SMOTE - Synthetic Minority Over-sampling Technique).
  Both undersampling and oversampling aim to create a balanced dataset, which can lead to improved model performance and more reliable predictions, especially for the minority class.

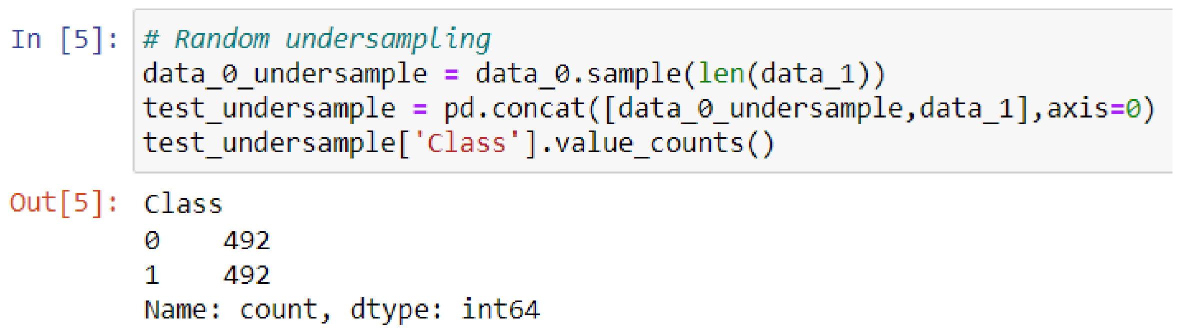
6. Using the command below we have separated the records which belong to class 0 and 1.

7. In this case, I will randomly select 92 records from the majority class (class zero) to match the minority class (class one). This process will help balance the dataset, allowing us to have an equal representation of both classes, which can improve the performance of our model.
8. By doing so, we aim to ensure that the model does not become biased towards the majority class, thereby enhancing its ability to accurately predict instances of the minority class.

```
In [4]: data_0 = data[data['Class']==0] # majority
        data_1 = data[data['Class']==1] # minority
```

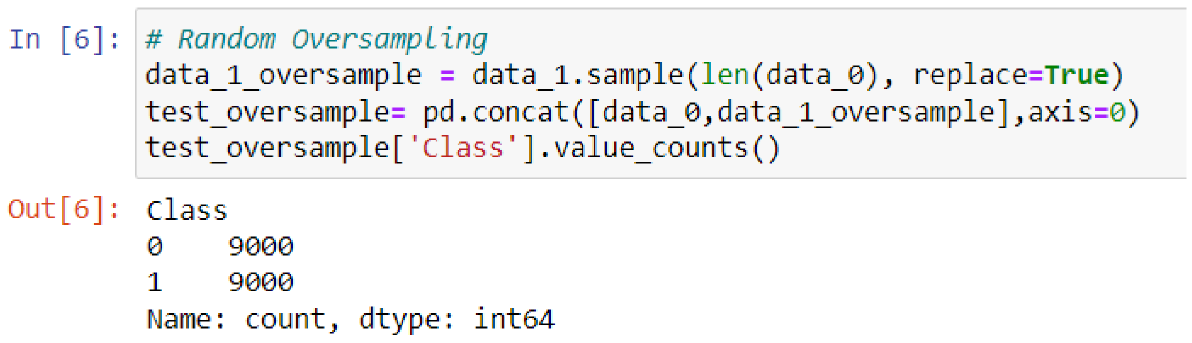
9. Now, if you check the value counts, you'll see that both classes have an equal number of rows. This demonstrates how random undersampling can effectively balance the records for both classes, ensuring they are equally represented. Balancing the dataset

like this is crucial for training models that can perform well on both classes, minimizing bias towards the majority class.



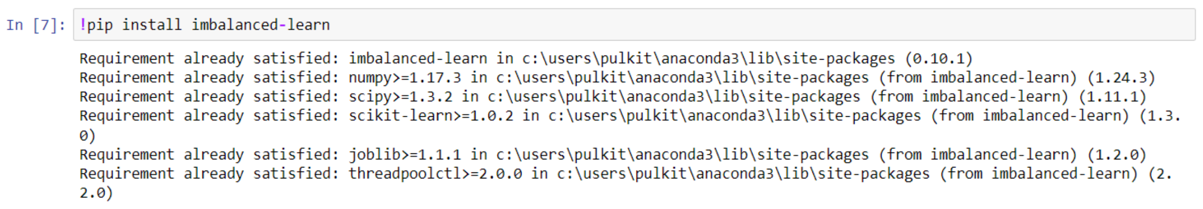
```
In [5]: # Random undersampling
        data_0_undersample = data_0.sample(len(data_1))
        test_undersample = pd.concat([data_0_undersample,data_1],axis=0)
        test_undersample['Class'].value_counts()

Out[5]: Class
        0    492
        1    492
        Name: count, dtype: int64
```

10. Conversely, if I want to perform oversampling, I can do this by sampling from the minority class until it matches the number of records in the majority class. This technique helps to create a balanced dataset by increasing the representation of the minority class, which can improve the performance of machine learning models by providing them with more examples to learn from.

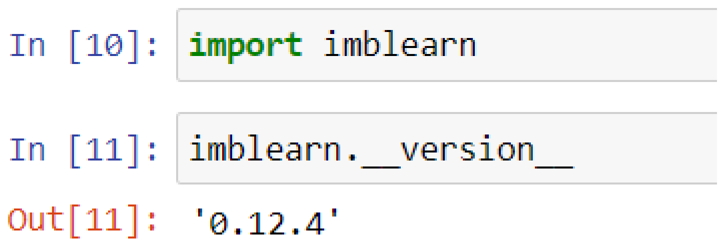


```
In [6]: # Random Oversampling
        data_1_oversample = data_1.sample(len(data_0), replace=True)
        test_oversample= pd.concat([data_0,data_1_oversample],axis=0)
        test_oversample['Class'].value_counts()

Out[6]: Class
        0    9000
        1    9000
        Name: count, dtype: int64
```

11. Now we'll be using the imbalanced-learn library, first, we need to install it on our Notebook.
12. After that we imported it and check for its version.



```
In [7]: !pip install imbalanced-learn

        Requirement already satisfied: imbalanced-learn in c:\users\pulkit\anaconda3\lib\site-packages (0.10.1)
        Requirement already satisfied: numpy>=1.17.3 in c:\users\pulkit\anaconda3\lib\site-packages (from imbalanced-learn) (1.24.3)
        Requirement already satisfied: scipy>=1.3.2 in c:\users\pulkit\anaconda3\lib\site-packages (from imbalanced-learn) (1.11.1)
        Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\pulkit\anaconda3\lib\site-packages (from imbalanced-learn) (1.3.
        0)
        Requirement already satisfied: joblib>=1.1.1 in c:\users\pulkit\anaconda3\lib\site-packages (from imbalanced-learn) (1.2.0)
        Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\pulkit\anaconda3\lib\site-packages (from imbalanced-learn) (2.
        2.0)
```



```
In [10]: import imblearn

In [11]: imblearn.__version__

Out[11]: '0.12.4'
```

13. The code is using a technique called **random undersampling** to balance the dataset. It randomly reduces the number of samples from the majority class (non-target) so that

both the majority and minority classes (target) have a similar number of samples, making the dataset more balanced for better model training.

14. The fit method in RandomUnderSampler analyzes the input data to learn how the undersampling should be applied. Specifically, it looks at the distribution of the target class ('Class' in this case) and determines how many samples from the majority class need to be removed so that both classes are balanced. However, the fit method itself does not modify the data; it just prepares the undersampler with the necessary information for future transformations like fit_resample.

```
In [12]: from imblearn.under_sampling import RandomUnderSampler
         rus = RandomUnderSampler(random_state=40, replacement=True)
         rus.fit(data.drop(columns='Class'),data['Class'])

Out[12]:                 RandomUnderSampler
         RandomUnderSampler(random_state=40, replacement=True)
```

15. After applying the random undersampling technique, I checked the size of the feature set (X), which has 984 samples and 31 features. Now, I want to take a look at the target variable (Y). When I check Y, I see that there are 490 samples for each class. This shows that the random undersampling worked, balancing the dataset evenly.

```
In [13]: X,y = rus.fit_resample(data.drop(columns='Class'),data['Class'])

In [14]: X.shape

Out[14]: (984, 31)

In [15]: y.value_counts()

Out[15]: Class
         0    492
         1    492
         Name: count, dtype: int64
```

16. Below we have applied the over sampler instead of under sampler.

```
In [16]: from imblearn.over_sampling import RandomOverSampler
         ros = RandomOverSampler(random_state=42)
         X,y = ros.fit_resample(data.drop(columns='Class'),data['Class'])
         X.shape

Out[16]: (18000, 31)

In [17]: y.value_counts()

Out[17]: Class
         1    9000
         0    9000
         Name: count, dtype: int64
```

17. In this way, we can use either a manual method or a library like imbalanced-learn to perform undersampling or oversampling. The manual method involves writing custom code to adjust the class distributions, while using a library simplifies the process by providing built-in functions and tools to handle these tasks efficiently.
18. The **SMOTE technique** (Synthetic Minority Oversampling Technique) is a method used to generate synthetic data for the minority class in an imbalanced dataset. Here's a breakdown of how it works:
    - **Random Selection**: It randomly selects a data point from the minority class.
    - **Finding Neighbors**: It calculates the kkk nearest neighbors of that selected data point.
    - **Generating Synthetic Points**: New synthetic data points are created between the chosen point and its neighbors. This helps to increase the representation of the minority class without simply duplicating existing points.
    - Overall, the SMOTE algorithm effectively helps balance the dataset by generating new, similar samples for the minority class, improving the model's ability to learn from this class.
19. The SMOTE algorithm operates in four simple steps:
    - **Select the Minority Class**: The algorithm starts by identifying the minority class from the dataset that needs to be balanced.
    - **Find kkk Nearest Neighbors**: After selecting the minority class, it finds the kkk nearest neighbors for each data point in that class. The value of kkk is a parameter that you need to specify during the initialization of the SMOTE algorithm.
    - **Generate Synthetic Samples**: Using the selected data point and its nearest neighbors, the algorithm creates new synthetic data points by interpolating between them.
    - **Add Synthetic Samples to Dataset**: Finally, the synthetic samples are added to the original dataset, increasing the representation of the minority class.
20. This process helps improve the balance of the dataset and can enhance the performance of machine learning models trained on it.

```
In [18]: from imblearn.over_sampling import SMOTE
         smote = SMOTE()
         X,y = smote.fit_resample(data.drop(columns='Class'),data['Class'])

In [19]: type(y)

Out[19]: pandas.core.series.Series

In [20]: y.value_counts()

Out[20]: Class
         1    9000
         0    9000
         Name: count, dtype: int64
```

21. The **NearMiss** technique is a method used in machine learning to handle class imbalance by undersampling the majority class. It is part of the imbalanced-learn library, which provides several strategies for resampling datasets.

```
In [21]: from imblearn.under_sampling import NearMiss
         nm = NearMiss()
         X,y = nm.fit_resample(data.drop(columns='Class'),data['Class'])
         y.value_counts()

Out[21]: Class
         0    492
         1    492
         Name: count, dtype: int64
```