

**Structured Data Assignment**

**Thadimudupula Sahith**

**GUVI**

**001**

## Introduction

The dataset in question contains a comprehensive collection of electronic health records belonging to patients who have been diagnosed with a specific disease. These health records comprise a detailed log of every aspect of the patients' medical history including all diagnoses symptoms prescribed drug treatments and medical tests that they have undergone. Each row represents a healthcare record/medical event for a patient and it includes a timestamp for each entry/event thereby allowing for a chronological view of the patient's medical history.

The Data has mainly three columns

1. **Patient-Uid** - Unique Alphanumeric Identifier for a patient
2. **Date** - Date when patient encountered the event.
3. **Incident** - This column describes which event occurred on the day.

## Problem Statement

The objective is to construct a prognostic model capable of ascertaining the potential eligibility of a patient for the "Target Drug" within a forthcoming period of 50 days. The provision of this information is of utmost importance for clinicians to make well-informed decisions regarding treatment alternatives.

## Problem

The primary challenge is to a binary classification undertaking, wherein the objective is to categorize patients into two distinct groups: those who are deemed eligible (1) or ineligible (0) for the "Target Drug" based on their medical records.

### **Objective**

The purpose of this task is to develop a reliable predictive model that, within the next half-century, can determine whether or not a particular patient is qualified to receive the "Target Drug." The purpose of this model is to provide medical professionals with a tool that will assist them in making better educated treatment decisions.

### **Potential Applications of Problem Statement**

The developed model may have uses in the medical and pharmaceutical fields in the future. It can be utilized to: Assist medical professionals in the process of developing individualized treatment regimens for patients.

Improve the production and distribution of drugs by optimizing their strategies.

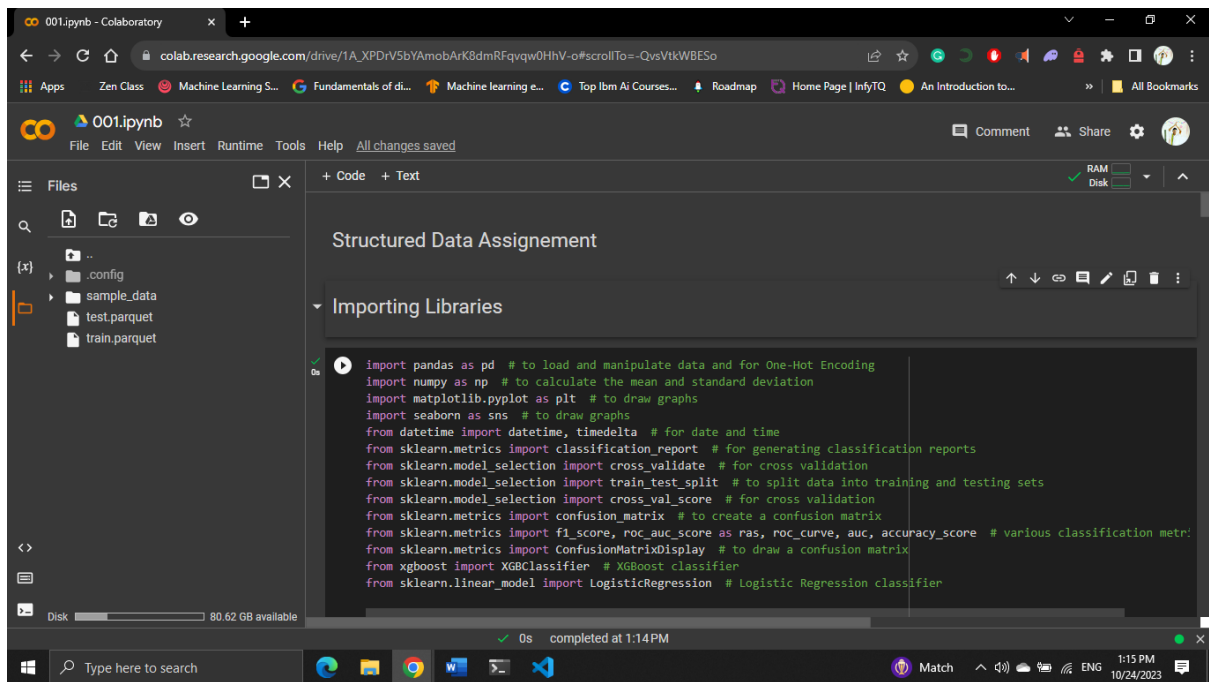
Enhance the quality of care provided to patients and the sector as a whole.

Reduce the risk of adverse reactions to medications.

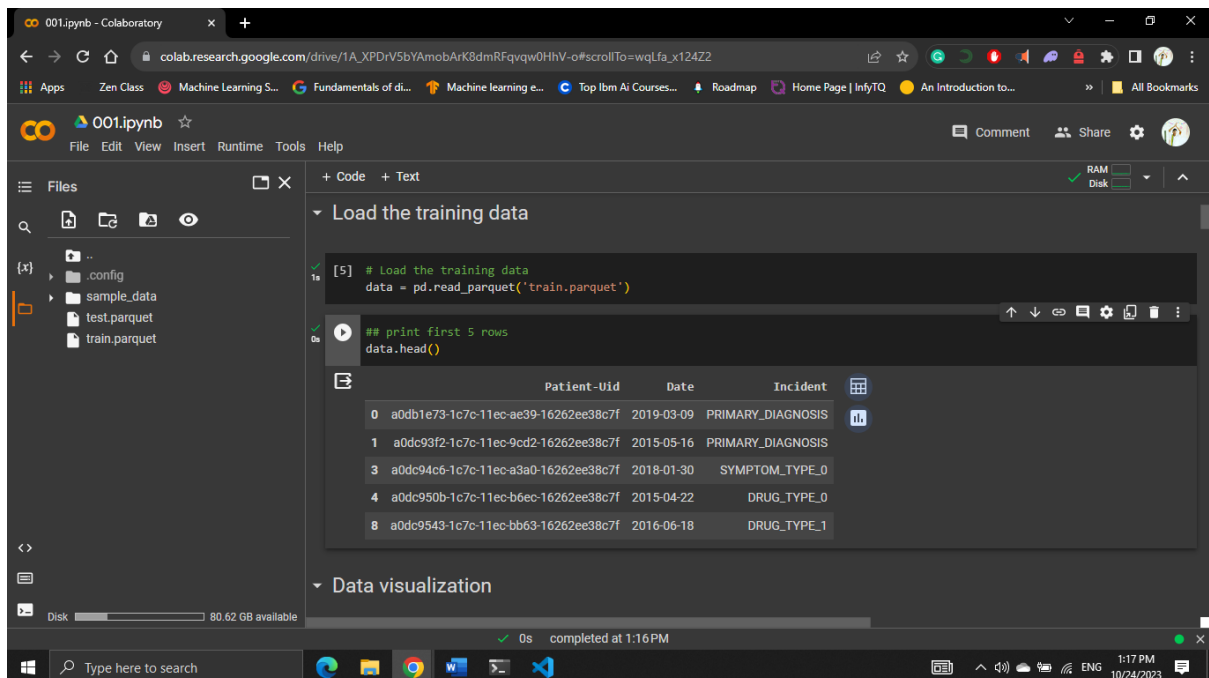
### **Type of Machine Learning Problem**

The task at hand pertains to a binary classification scenario whereby the objective is to make predictions regarding the eligibility of a patient for the "Target Drug" during a span of 30 days. These predictions are based on an analysis of the patient's medical history. The outcome variable is represented by two distinct classes: 1 denotes eligibility for the drug, while 0 signifies ineligibility. Additionally this problem may involve time-series analysis and feature engineering techniques to incorporate temporal aspects into the predictive model.

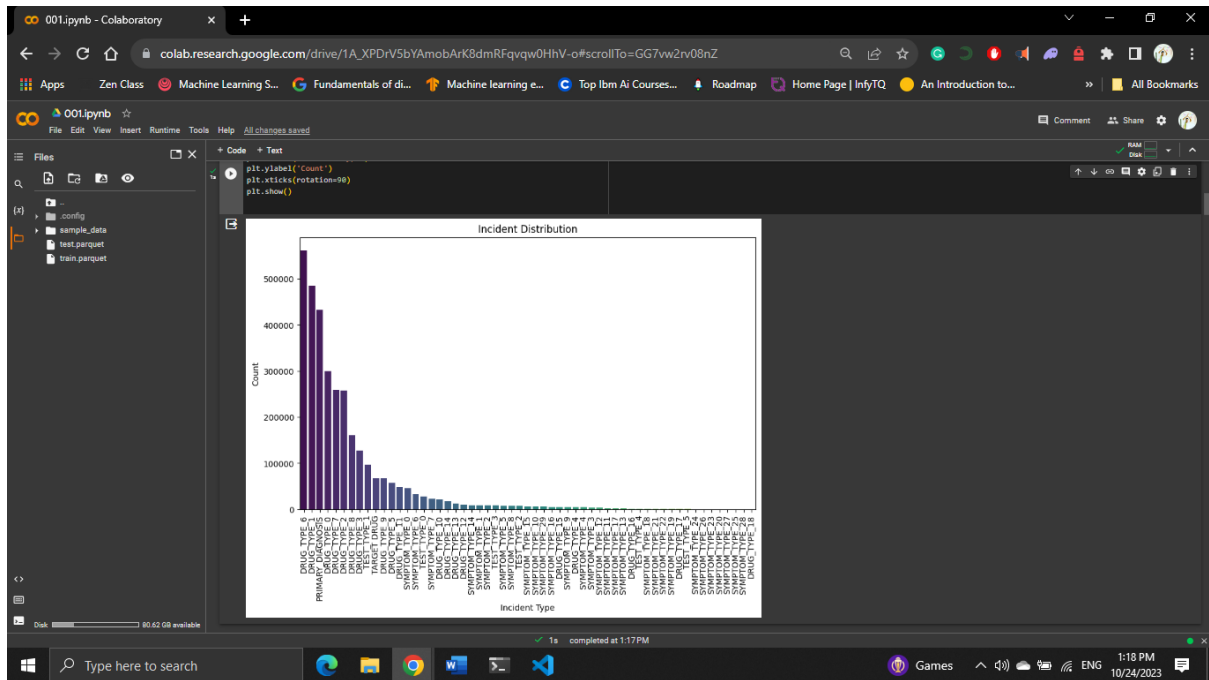
### **Importing Libraries**



## Load the training data

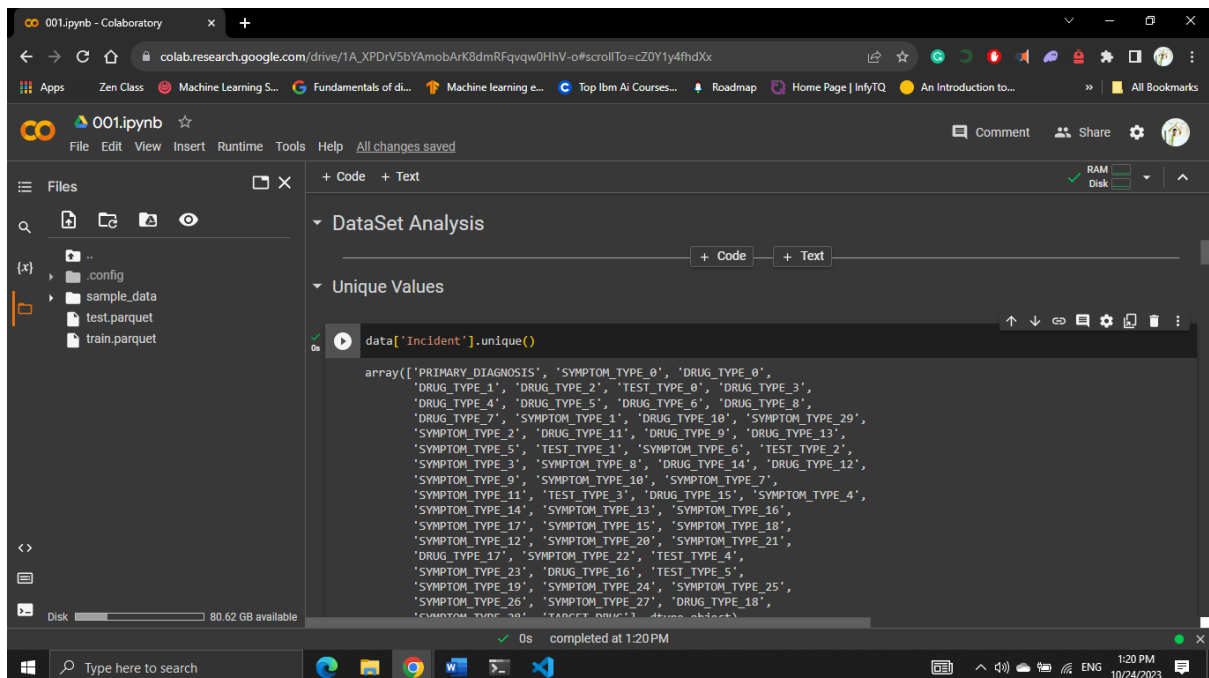


## Data visualization



## Dataset Analysis

### Unique Values in Incident column



Based on the aforementioned findings, it can be concluded that all values are distinct, indicating the absence of any erroneous data such as spelling mistakes or inconsistencies in capitalization.

```
[9] len(data)
```

```
3220868
```

```
[10] # Checking size of dataset
print("Dataset size : ", data.shape)
```

```
Dataset size : (3220868, 3)
```

```
#Basic statistical analysis of dataset
data.describe()
```

	Patient-Uid	Date	Incident
count	3220868	3220868	3220868
unique	27033	1977	57

0s completed at 1:39 PM

```
#Basic statistical analysis of dataset
data.describe()
```

	Patient-Uid	Date	Incident
count	3220868	3220868	3220868
unique	27033	1977	57
top	a0ddfd2c-1c7c-11ec-876d-16262ee38c7f	2019-05-21 00:00:00	DRUG_TYPE_6
freq	1645	3678	561934
first	NaN	2015-04-07 00:00:00	NaN
last	NaN	2020-09-03 00:00:00	NaN

Find and Remove Missing Data

Checking Data Type

```
#dtypes tells us "data types" for each column
```

0s completed at 1:39 PM

## Data Summary

**Total Rows:** 3,220,868

## Columns

**Patient-Uid:** Unique alphanumeric identifier for patients

**Date:** Date when the event occurred

**Incident:** Describes the type of event that occurred (e.g., primary diagnosis, symptom type, drug type)

### **Unique Incidents**

There are a total of 57 unique types of incidents.

### **Most Frequent Incidents**

The most frequent incident is **DRUG\_TYPE\_6** with 549,616 occurrences, followed by **DRUG\_TYPE\_1** with 484,565 occurrences, and **PRIMARY\_DIAGNOSIS** with 424,879 occurrences.

### **Occurrences of "Target Drug"**

There are 67,218 occurrences of the "TARGET DRUG" incident.

### **Data Range**

The data spans from April 7, 2015, to September 3, 2020.

### **Number of Duplicates**

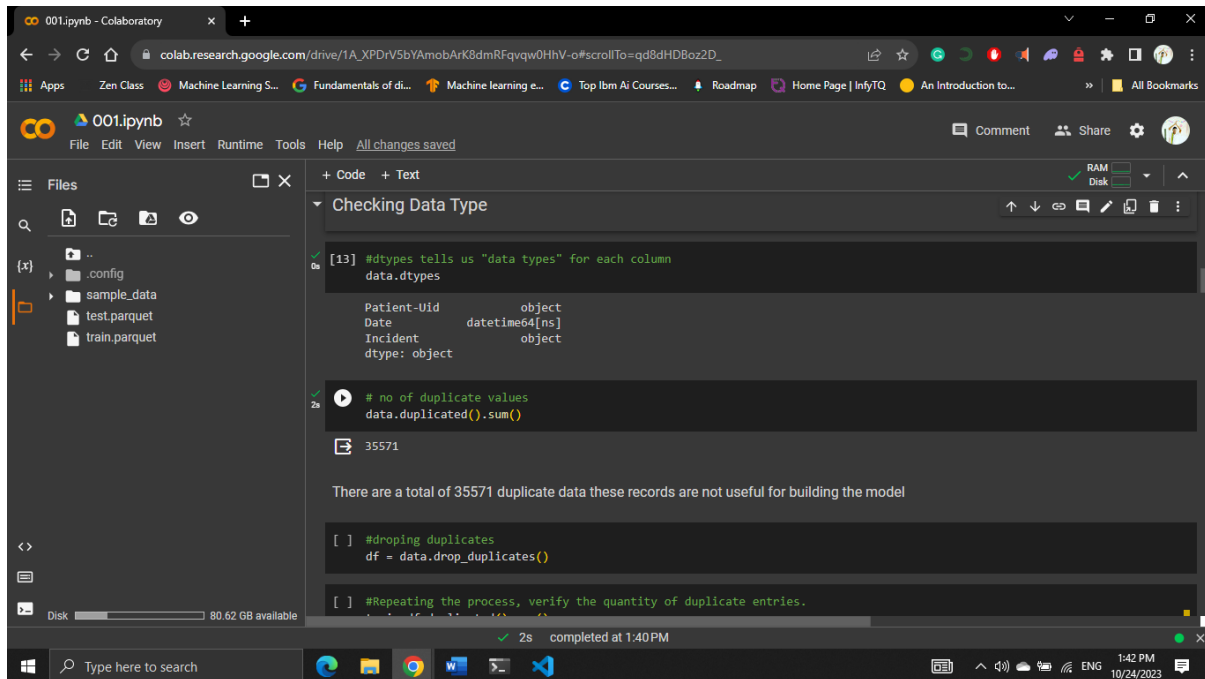
There are 35,571 duplicate rows in the dataset.

### **Value Counts for Each Incident Type**

**DRUG\_TYPE\_6** occurs most frequently with 549,616 instances followed by **DRUG\_TYPE\_1** with 484,565 instances, and **PRIMARY\_DIAGNOSIS** with 424,879 instances.

There's a wide range of incident types including various drug types symptom types and test types.

## Find and Remove Missing Data



The screenshot shows a Jupyter Notebook interface with the following code and output:

```
[13] #dtypes tells us "data types" for each column
data.dtypes

Patient-Uid      object
Date             datetime64[ns]
Incident         object
dtype: object
```

2s

```
# no of duplicate values
data.duplicated().sum()
```

35571

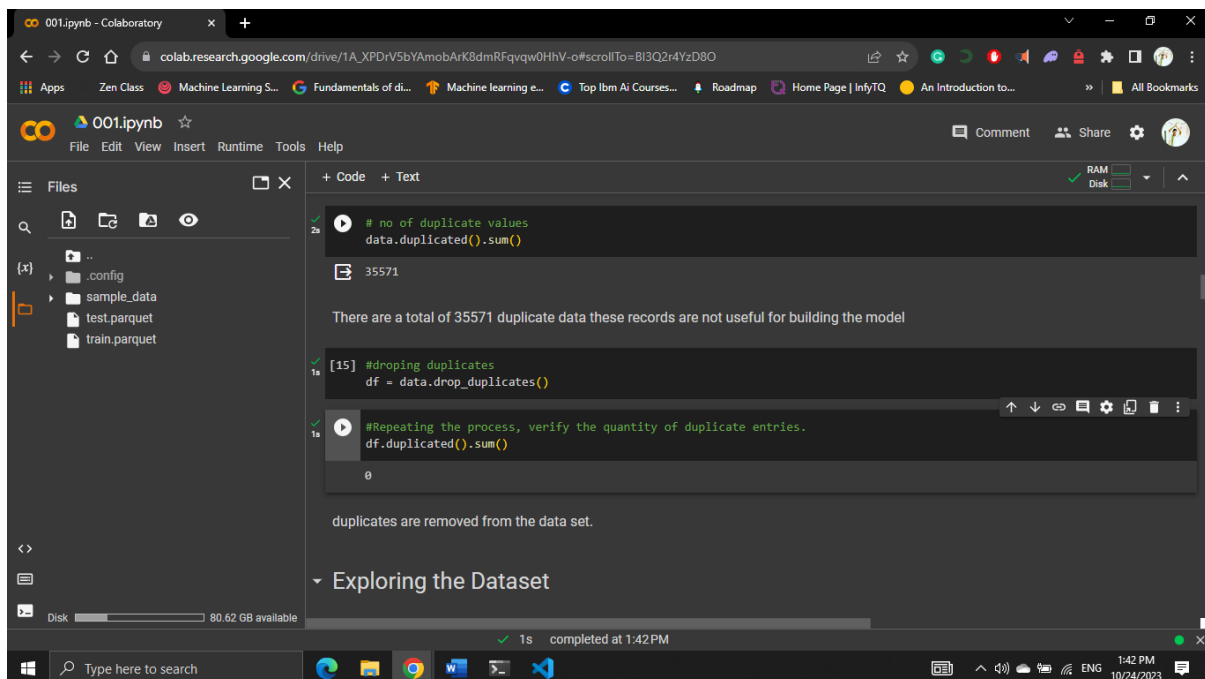
There are a total of 35571 duplicate data these records are not useful for building the model

```
[ ] #dropping duplicates
df = data.drop_duplicates()
```

```
[ ] #Repeating the process, verify the quantity of duplicate entries.
```

2s completed at 1:40 PM

**Duplicates Are Removed from The Data Set.**



The screenshot shows a Jupyter Notebook interface with the following code and output:

```
# no of duplicate values
data.duplicated().sum()
```

35571

There are a total of 35571 duplicate data these records are not useful for building the model

```
[15] #dropping duplicates
df = data.drop_duplicates()
```

```
#Repeating the process, verify the quantity of duplicate entries.
df.duplicated().sum()
```

0

duplicates are removed from the data set.

Exploring the Dataset

1s completed at 1:42 PM

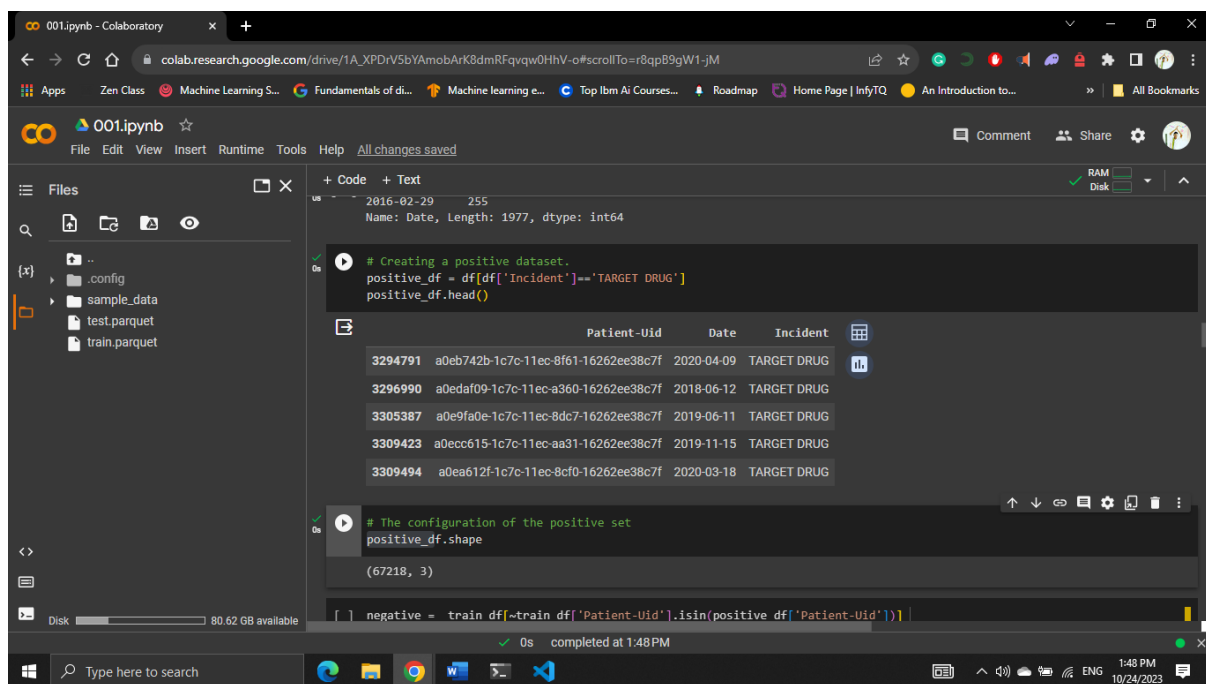
**Positive Dataset.**



`positive_df`: This is a new DataFrame that will store the positive examples. These examples represent instances where the incident is 'TARGET DRUG' meaning that the patient has taken the "Target Drug".

`df[df['Incident']=='TARGET DRUG']`: This line of code filters the `df` DataFrame to select only the rows where the value in the 'Incident' column is 'TARGET DRUG'. In other words it selects all the records where the incident is the "Target Drug".

`.head()`: This function is used to display the first few rows of the DataFrame. By default it displays the first 5 rows. This is useful for quickly inspecting the data. show the first 5 rows.



```

001.ipynb - Colaboratory
colab.research.google.com/drive/1A_XPDtV5bYAmobArk3dmRFqvw0HhV-c#scrollTo=r8qp89gW1-jM
001.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Files
  .config
  sample_data
  test.parquet
  train.parquet
Code
  2016-02-29 255
  Name: Date, Length: 1977, dtype: int64
  # Creating a positive dataset.
  positive_df = df[df['Incident']=='TARGET DRUG']
  positive_df.head()
  Patient-Uid Date Incident
  3294791 a0eb742b-1c7c-11ec-8f61-16262ee38c7f 2020-04-09 TARGET DRUG
  3296990 a0edaf09-1c7c-11ec-a360-16262ee38c7f 2018-06-12 TARGET DRUG
  3305387 a0e9fa0e-1c7c-11ec-8dc7-16262ee38c7f 2019-06-11 TARGET DRUG
  3309423 a0ecc615-1c7c-11ec-aa31-16262ee38c7f 2019-11-15 TARGET DRUG
  3309494 a0ea612f-1c7c-11ec-8cf0-16262ee38c7f 2020-03-18 TARGET DRUG
  # The configuration of the positive set
  positive_df.shape
  (67218, 3)
  negative = train_df[~train_df['Patient-Uid'].isin(positive_df['Patient-Uid'])]
  completed at 1:48 PM
  1:48 PM 10/24/2023

```

## Negative Dataset

`negative = df[~df['Patient-Uid'].isin(positive_df['Patient-Uid'])]`:

`df['Patient-Uid'].isin(positive_df['Patient-Uid'])` checks if the 'Patient-Uid' values in `df` are present in the 'Patient-Uid' values of `positive_df`.

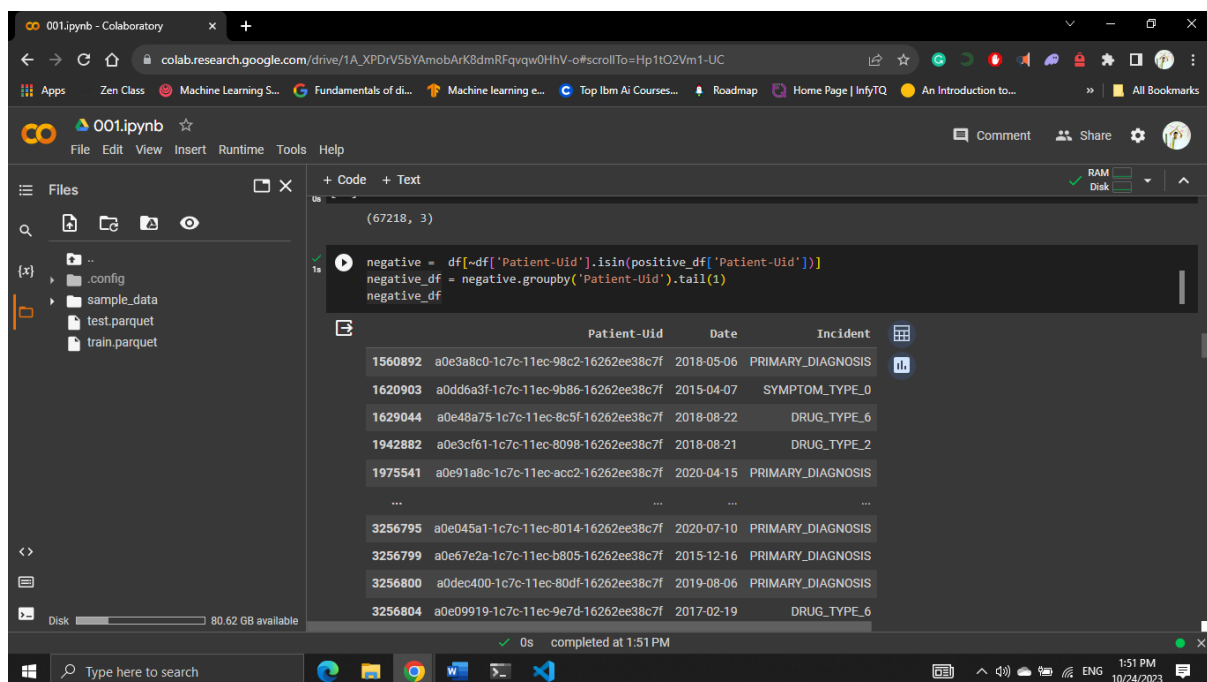
$\sim$  is a logical negation operator so `~df['Patient-Uid'].isin(positive_df['Patient-Uid'])` will return **True** for rows where 'Patient-Uid' is not in **positive\_df**.

This filters the DataFrame **df** to select only the rows where the 'Patient-Uid' is not in the positive set. These are the negative examples (Starmer, 2022).

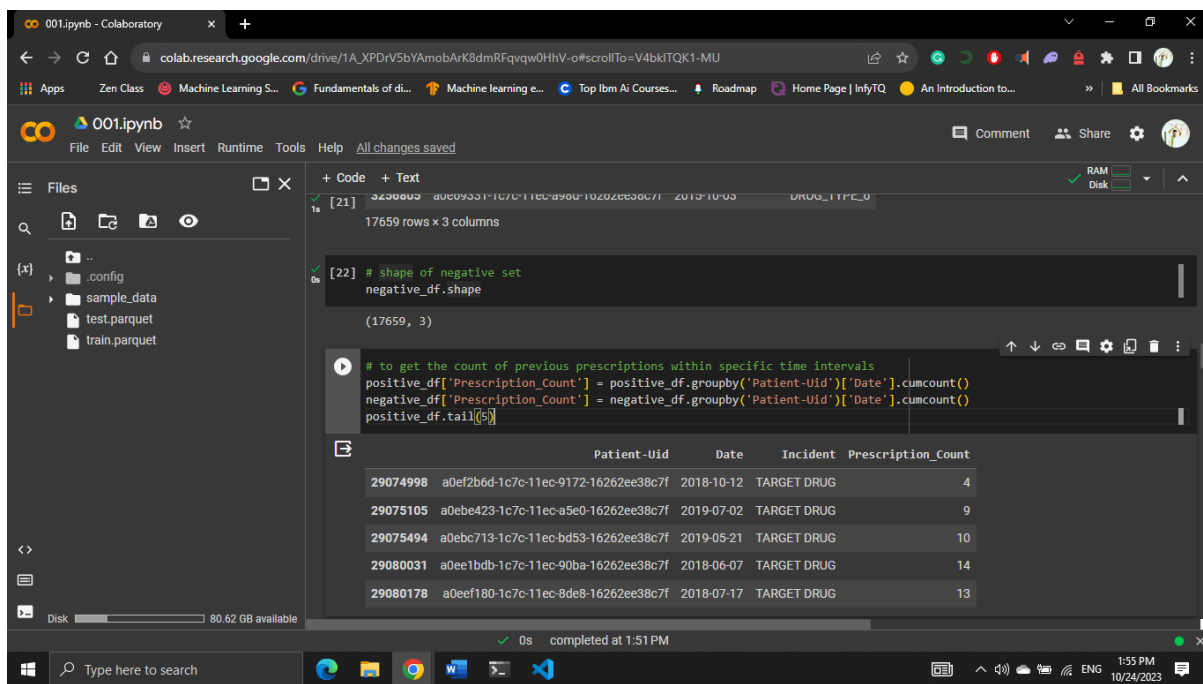
```
negative_df = negative.groupby('Patient-Uid').tail(1)
```

`.groupby('Patient-Uid')` groups the Data Frame by the unique 'Patient-Uid' values.

`.tail(1)` selects the last row from each group. This is assuming that the data is sorted by time so the last entry for each patient is the most recent.



To get the count of previous prescriptions within specific time intervals



The screenshot shows a Google Colab notebook with the following code and output:

```
[21] 3230003 a0e9531-1c7c-11ec-a90f-16262ee38c7f 2018-10-05 TARGET DRUG
17659 rows x 3 columns
```

```
[22] # shape of negative set
negative_df.shape

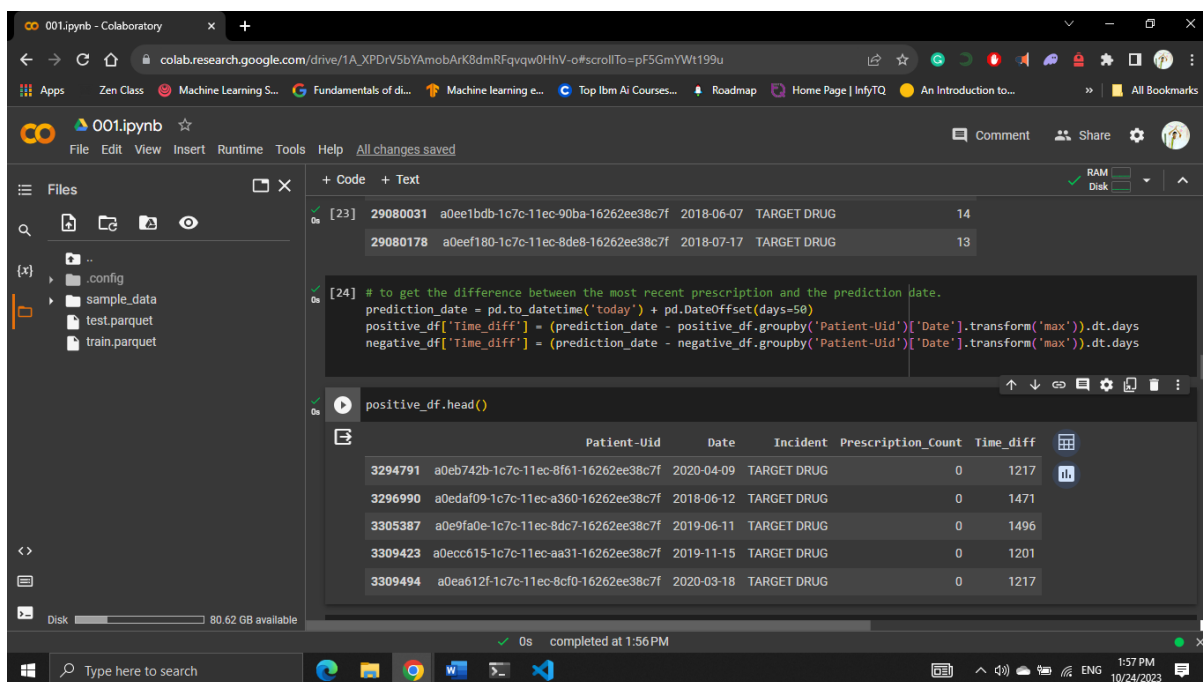
(17659, 3)
```

```
# to get the count of previous prescriptions within specific time intervals
positive_df['Prescription_Count'] = positive_df.groupby('Patient-Uid')['Date'].cumcount()
negative_df['Prescription_Count'] = negative_df.groupby('Patient-Uid')['Date'].cumcount()
positive_df.tail(5)
```

	Patient-Uid	Date	Incident	Prescription_Count
29074998	a0ef2b6d-1c7c-11ec-9172-16262ee38c7f	2018-10-12	TARGET DRUG	4
29075105	a0ebe423-1c7c-11ec-a5e0-16262ee38c7f	2019-07-02	TARGET DRUG	9
29075494	a0ebc713-1c7c-11ec-bd53-16262ee38c7f	2019-05-21	TARGET DRUG	10
29080031	a0ee1bdb-1c7c-11ec-90ba-16262ee38c7f	2018-06-07	TARGET DRUG	14
29080178	a0eef180-1c7c-11ec-8de8-16262ee38c7f	2018-07-17	TARGET DRUG	13

0s completed at 1:51 PM

To get the difference between the most recent prescription and the prediction date.



The screenshot shows a Google Colab notebook with the following code and output:

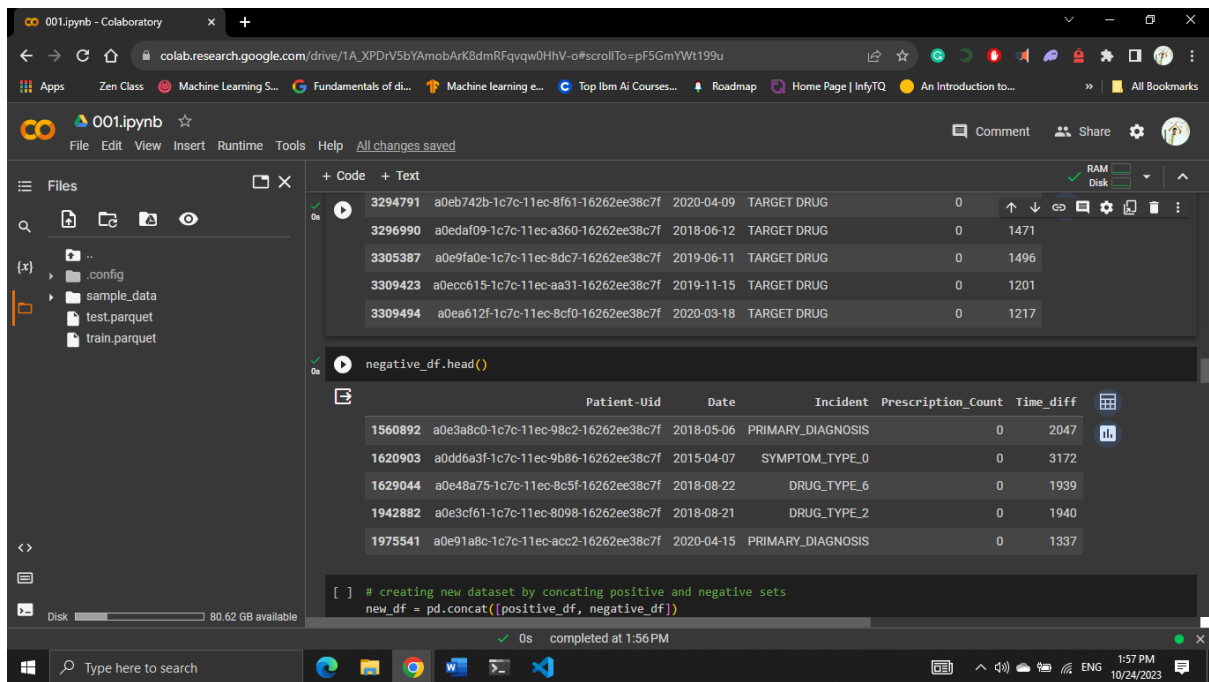
```
[23] 29080031 a0ee1bdb-1c7c-11ec-90ba-16262ee38c7f 2018-06-07 TARGET DRUG 14
29080178 a0eef180-1c7c-11ec-8de8-16262ee38c7f 2018-07-17 TARGET DRUG 13
```

```
[24] # to get the difference between the most recent prescription and the prediction date.
prediction_date = pd.to_datetime('today') + pd.DateOffset(days=50)
positive_df['Time_diff'] = (prediction_date - positive_df.groupby('Patient-Uid')['Date'].transform('max')).dt.days
negative_df['Time_diff'] = (prediction_date - negative_df.groupby('Patient-Uid')['Date'].transform('max')).dt.days
```

```
positive_df.head()
```

	Patient-Uid	Date	Incident	Prescription_Count	Time_diff
3294791	a0eb742b-1c7c-11ec-8f61-16262ee38c7f	2020-04-09	TARGET DRUG	0	1217
3296990	a0edaf09-1c7c-11ec-a360-16262ee38c7f	2018-06-12	TARGET DRUG	0	1471
3305387	a0e9fa0e-1c7c-11ec-8dc7-16262ee38c7f	2019-06-11	TARGET DRUG	0	1496
3309423	a0ecc615-1c7c-11ec-aa31-16262ee38c7f	2019-11-15	TARGET DRUG	0	1201
3309494	a0ea612f-1c7c-11ec-8cf0-16262ee38c7f	2020-03-18	TARGET DRUG	0	1217

0s completed at 1:56 PM



The screenshot shows a Google Colab notebook with a file explorer on the left containing folders like '.config', 'sample\_data', and files 'test.parquet' and 'train.parquet'. The main area displays a table of data with columns: Patient-Uid, Date, Incident, Prescription\_Count, and Time\_diff. Below the table, a code cell is shown with the following code:

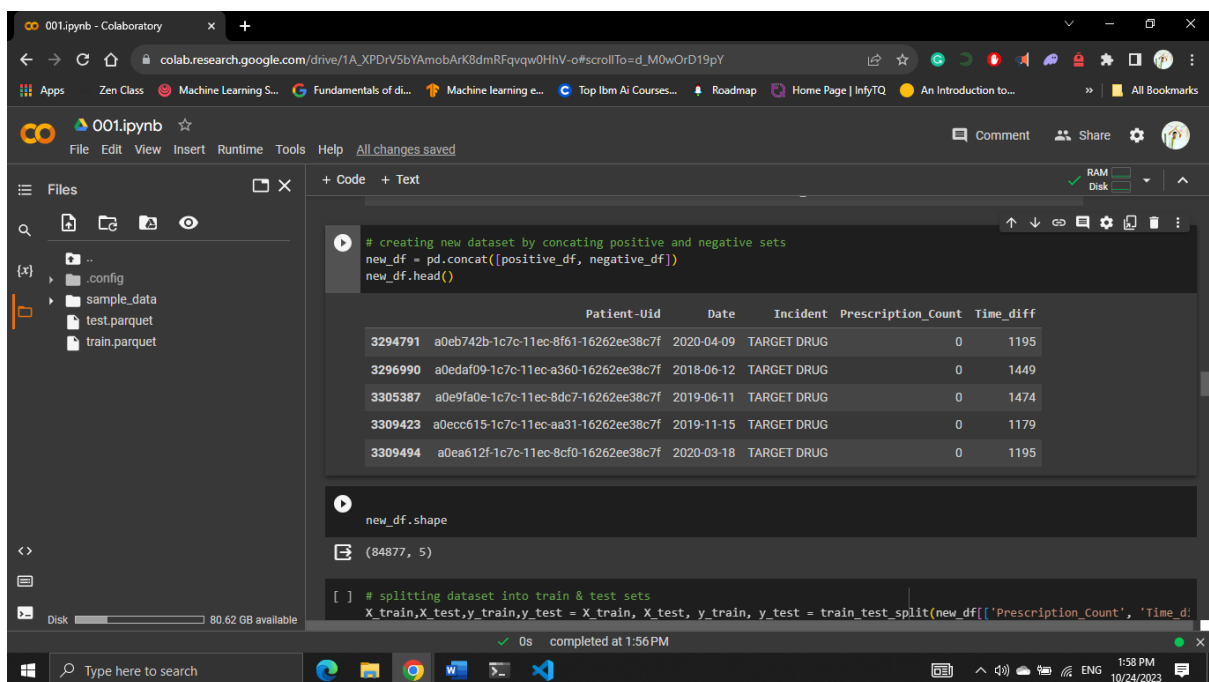
```
negative_df.head()

[ ] # creating new dataset by concatenating positive and negative sets
new_df = pd.concat([positive_df, negative_df])
```

The table data is as follows:

Patient-Uid	Date	Incident	Prescription_Count	Time_diff
3294791	2020-04-09	TARGET DRUG	0	1195
3296990	2018-06-12	TARGET DRUG	0	1449
3305387	2019-06-11	TARGET DRUG	0	1474
3309423	2019-11-15	TARGET DRUG	0	1179
3309494	2020-03-18	TARGET DRUG	0	1195

creating new dataset by concating positive and negative sets



The screenshot shows the same Google Colab notebook. The code cell now includes the following code:

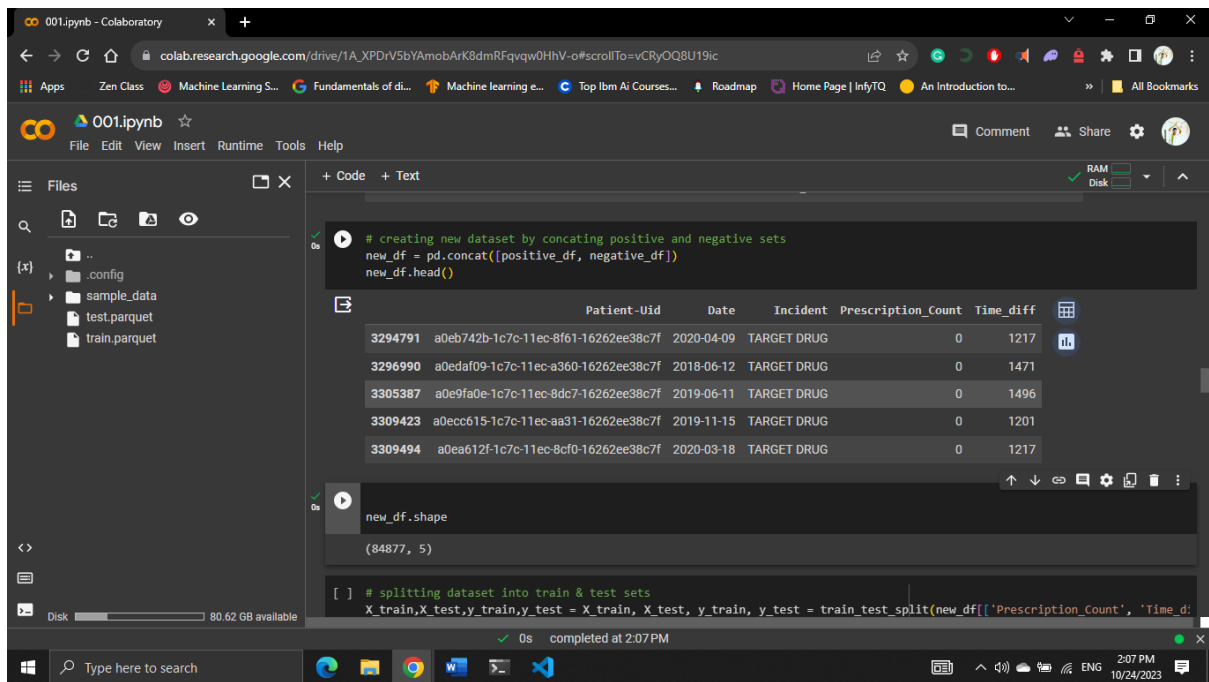
```
# creating new dataset by concatenating positive and negative sets
new_df = pd.concat([positive_df, negative_df])
new_df.head()

new_df.shape

[ ] # splitting dataset into train & test sets
X_train,X_test,y_train,y_test = train_test_split(new_df[['Prescription_Count', 'Time_d
```

The table data is as follows:

Patient-Uid	Date	Incident	Prescription_Count	Time_diff
3294791	2020-04-09	TARGET DRUG	0	1195
3296990	2018-06-12	TARGET DRUG	0	1449
3305387	2019-06-11	TARGET DRUG	0	1474
3309423	2019-11-15	TARGET DRUG	0	1179
3309494	2020-03-18	TARGET DRUG	0	1195



```
# creating new dataset by concatenating positive and negative sets
new_df = pd.concat([positive_df, negative_df])
new_df.head()
```

	Patient-Uid	Date	Incident	Prescription_Count	Time_diff
3294791	a0eb742b-1c7c-11ec-8f61-16262ee38c7f	2020-04-09	TARGET DRUG	0	1217
3296990	a0edaf09-1c7c-11ec-a360-16262ee38c7f	2018-06-12	TARGET DRUG	0	1471
3305387	a0e9fa0e-1c7c-11ec-8dc7-16262ee38c7f	2019-06-11	TARGET DRUG	0	1496
3309423	a0ecc615-1c7c-11ec-aa31-16262ee38c7f	2019-11-15	TARGET DRUG	0	1201
3309494	a0ea612f-1c7c-11ec-8cf0-16262ee38c7f	2020-03-18	TARGET DRUG	0	1217

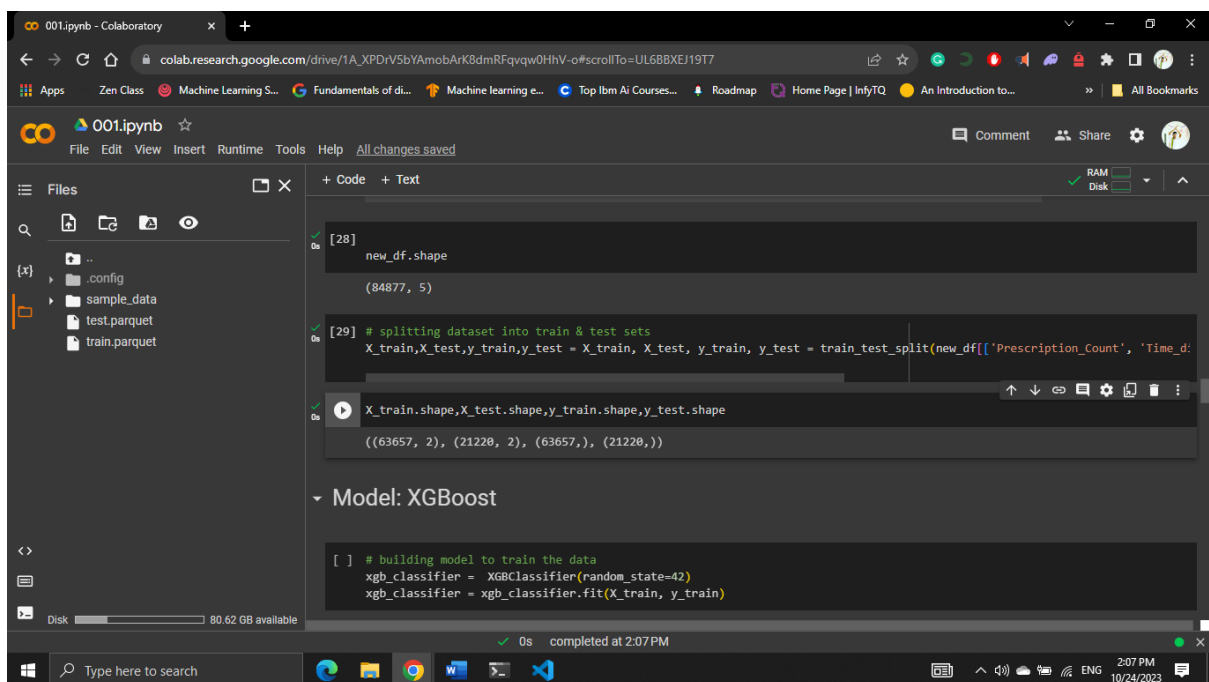
```
new_df.shape
```

```
(84877, 5)
```

```
[ ] # splitting dataset into train & test sets
X_train,X_test,y_train,y_test = X_train, X_test, y_train, y_test = train_test_split(new_df[['Prescription_Count', 'Time_d
```

completed at 2:07 PM

splitting dataset into train & test sets



```
[28]
new_df.shape
(84877, 5)
```

```
[29] # splitting dataset into train & test sets
X_train,X_test,y_train,y_test = X_train, X_test, y_train, y_test = train_test_split(new_df[['Prescription_Count', 'Time_d
```

```
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

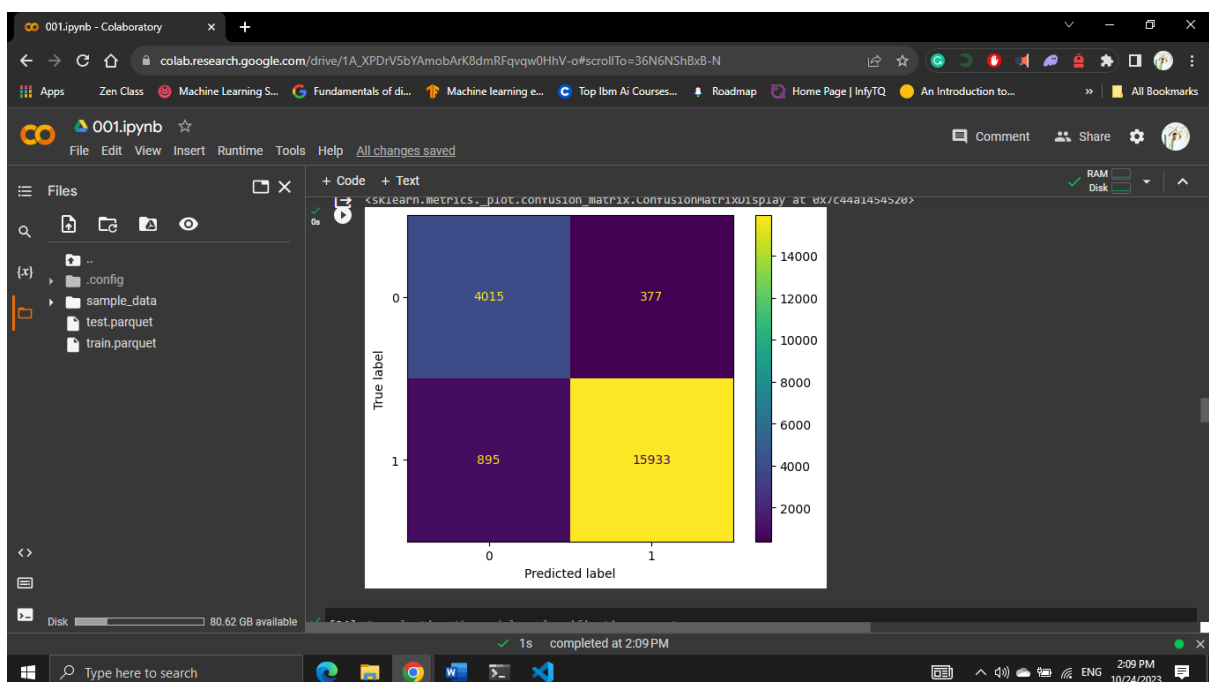
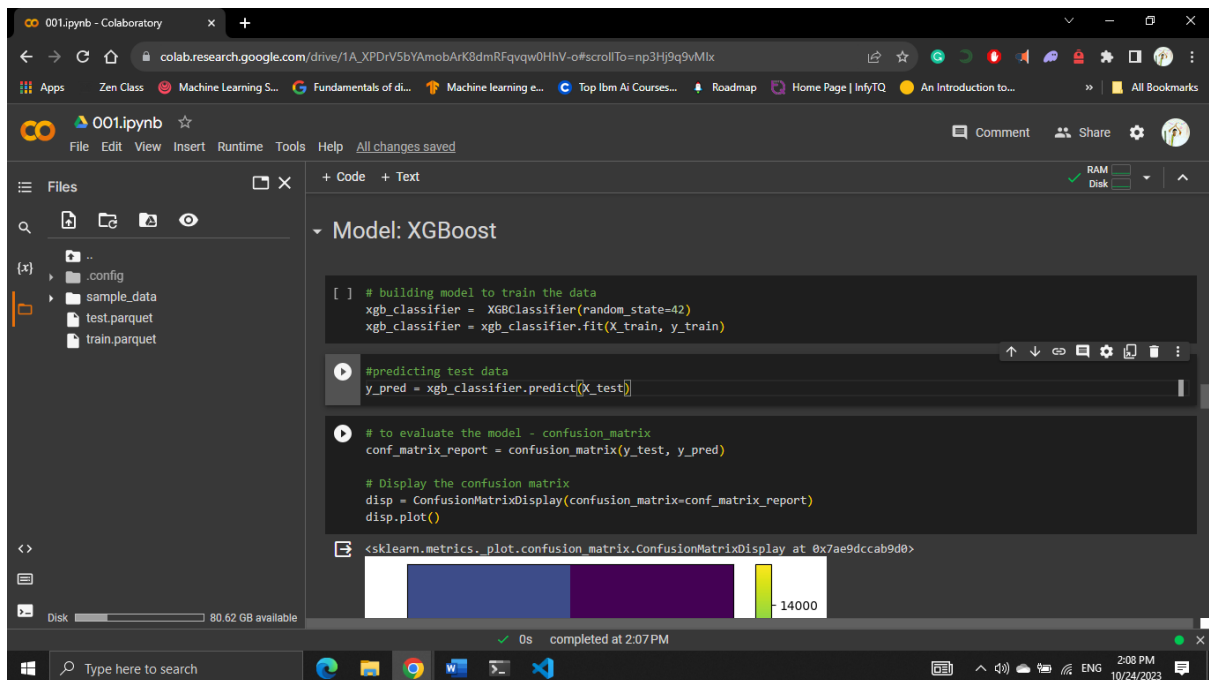
```
((63657, 2), (21220, 2), (63657,), (21220,))
```

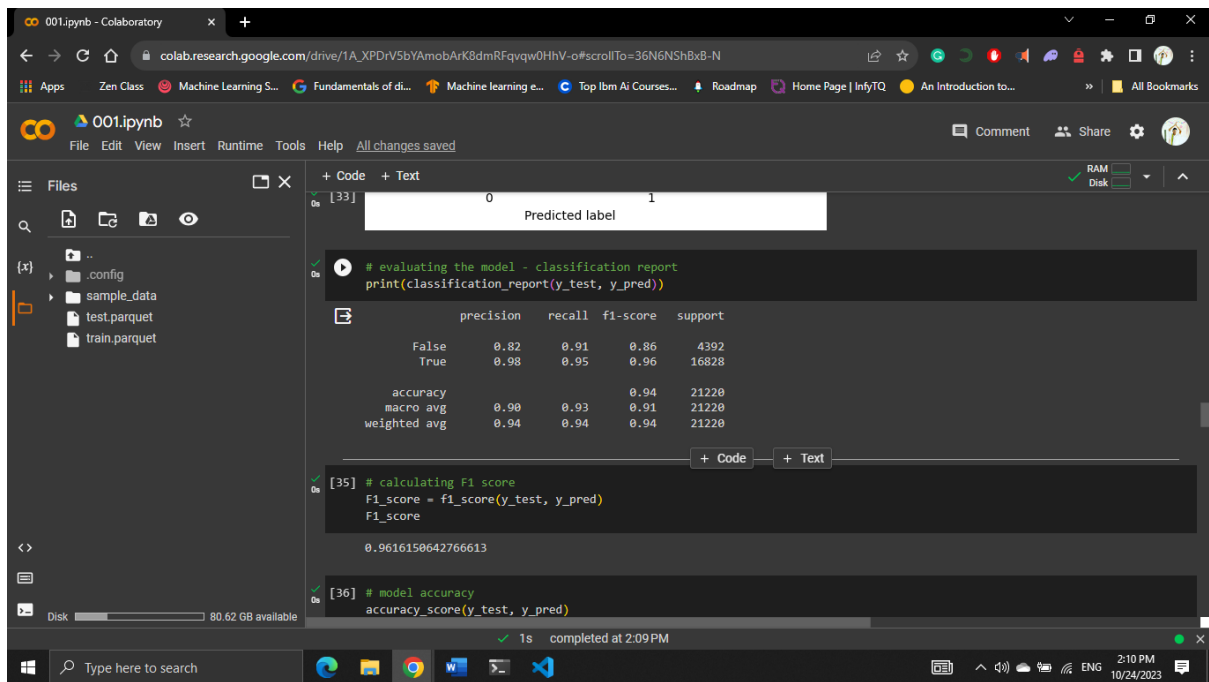
Model: XGBoost

```
[ ] # building model to train the data
xgb_classifier = XGBClassifier(random_state=42)
xgb_classifier = xgb_classifier.fit(X_train, y_train)
```

completed at 2:07 PM

## Model: XGBoost





The screenshot shows a Jupyter Notebook in a web browser. The left sidebar displays a file explorer with a directory structure: `..`, `.config`, `sample_data`, `test.parquet`, and `train.parquet`. The main area contains three code cells. The first cell, labeled [33], shows a predicted label array: `0` and `1`. The second cell, labeled [34], evaluates the model using `classification_report`, displaying a table of metrics. The third cell, labeled [35], calculates the F1 score, resulting in `0.9616150642766613`. The fourth cell, labeled [36], calculates the model accuracy, resulting in `0.9400565504241282`. The bottom status bar indicates the notebook is completed at 2:09 PM.

```
[33] 0 1
      Predicted label

[34] # evaluating the model - classification report
      print(classification_report(y_test, y_pred))

      precision recall f1-score support

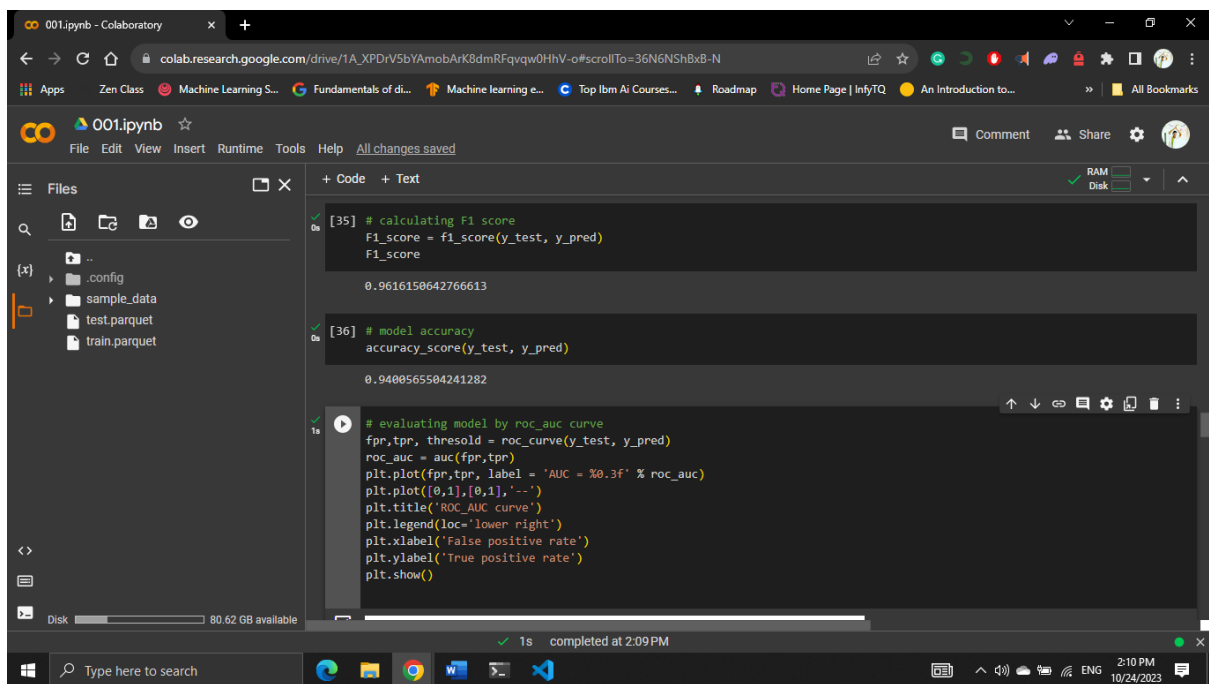
      False    0.82    0.01    0.86    4392
       True    0.98    0.95    0.96   16828

      accuracy    0.90    0.93    0.94   21220
    macro avg    0.90    0.93    0.91   21220
    weighted avg    0.94    0.94    0.94   21220

[35] # calculating F1 score
      F1_score = f1_score(y_test, y_pred)
      F1_score

      0.9616150642766613

[36] # model accuracy
      accuracy_score(y_test, y_pred)
```



The screenshot shows the same Jupyter Notebook interface, but with an additional code cell. The first cell, labeled [35], calculates the F1 score, resulting in `0.9616150642766613`. The second cell, labeled [36], calculates the model accuracy, resulting in `0.9400565504241282`. The third cell, labeled [37], evaluates the model by plotting an ROC curve. The code includes `roc_curve`, `auc`, and `plt` to generate the plot. The bottom status bar indicates the notebook is completed at 2:09 PM.

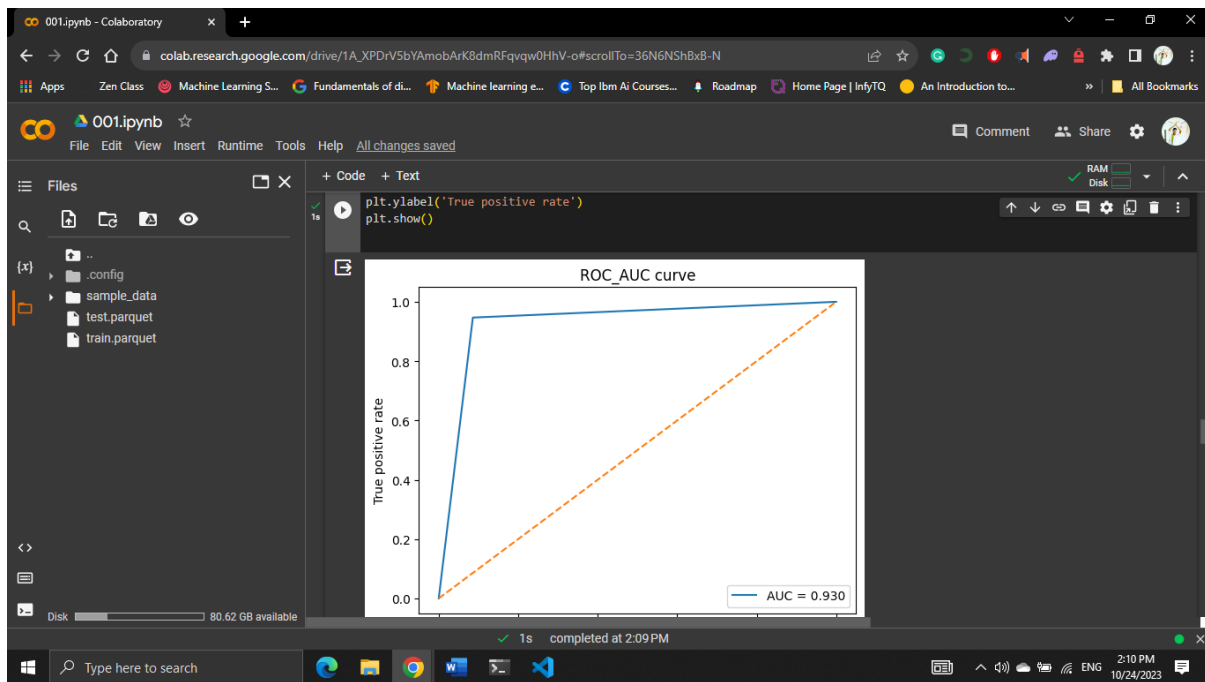
```
[35] # calculating F1 score
      F1_score = f1_score(y_test, y_pred)
      F1_score

      0.9616150642766613

[36] # model accuracy
      accuracy_score(y_test, y_pred)

      0.9400565504241282

[37] # evaluating model by roc_auc curve
      fpr,tpr, threshold = roc_curve(y_test, y_pred)
      roc_auc = auc(fpr,tpr)
      plt.plot(fpr,tpr, label = 'AUC = %0.3f' % roc_auc)
      plt.plot([0,1],[0,1], '--')
      plt.title('ROC_AUC curve')
      plt.legend(loc='lower right')
      plt.xlabel('False positive rate')
      plt.ylabel('True positive rate')
      plt.show()
```



### Building and Training the Model

The XGBoost Classifier is utilized to build a predictive model. This classifier is known for its effectiveness in various machine learning tasks.

**xgb\_classifier = XGBClassifier(random\_state=42)** XGBoost Classifier with a specific random seed.

**xgb\_classifier = xgb\_classifier.fit(X\_train, y\_train)** trains the model on the training data (X\_train and y\_train).

The model is using the XGBoost Classifier a algorithm known for its effectiveness in a wide range of machine learning tasks.

### Confusion Matrix Evaluation

The resulting confusion matrix is displayed as:

```
[[ 4015, 377], [ 895, 15933]]
```



The confusion matrix provides model's predictions. It consists of four values: true negatives (4015) false positives (377) false negatives (895) and true positives (15933).

### **Classification Report**

The classification report provides a concise overview of the performance metrics of the model, encompassing precision recall and F1-score for both the True and False classes. Additionally it offers assistance by providing the support which refers to the frequency of each class in the exam set.

The classification report provides a comprehensive overview of the model's performance by presenting metrics like as precision recall and F1-score for each individual class. The precision recall and F1-score for the 'False' class are 0.82 0.91 and 0.86 respectively. In the 'True' class the precision is 0.98 the recall is 0.95 and the F1-score is 0.96.

### **Calculating F1 Score**

The F1 score is a composite measure that combines precision and recall. The metric gives a singular numerical representation denoting the efficacy of the model. The F1 score denoted as `F1_score` is computed by applying the `f1_score` function on the true labels (`y_test`) and predicted labels (`y_pred`). The F1 score as determined through calculations is roughly 0.962.

The F1 score a metric that combines precision and recall is estimated to be around 0.962. This observation suggests that the model exhibits excellent performance in achieving a balance between precision and recall.

### **The Area Under the Curve**

In this particular instance the area under the receiver operating characteristic curve (AUC) is calculated to be 0.930. This value indicates that the model has a robust capacity to distinguish between patients who meet the criteria for receiving the "Target Drug" and those who do not (Starmer, 2022).

The computed value for the Area Under the Receiver Operating Characteristic Curve (AUC) is 0.930. The obtained score demonstrates that the model has a significant degree of discriminative capability efficiently discerning between patients who meet the criteria for receiving the 'Target Drug' and those who do not.

### **Model Accuracy**

The measure of model accuracy quantifies the proportion of right predictions relative to the total number of predictions made. The function `accuracy_score(y_test, y_pred)` computes the accuracy by comparing the genuine labels (`y_test`) with the predicted labels (`y_pred`).

The accuracy of the model was determined to be roughly 0.940 indicating that it accurately predicts outcomes in approximately 94% of cases. The model has a notable level of accuracy estimated at around 94% indicating its ability to accurately forecast patient eligibility for the 'Target Drug' in approximately 94% of instances.

**References**

Starmer, J. (2022). The Statquest illustrated guide to machine learning!!!: master the concepts, one full-color picture at a time, from the basics all the way to neural networks. BAM!.