Structured Data Assignment

Data Description:

- **Train.parquet** - Dataset to be used for training
- **Test.parquet** - Dataset to be used for testing

The data consists of three primary columns:

- **Patient-Uid:** A unique alphanumeric identifier assigned to each patient who are taking the medicine.
- **Date:** The specific date when the patient experienced the event.
- **Incident:** This column provides a description of the event that took place on the given date for the patient.

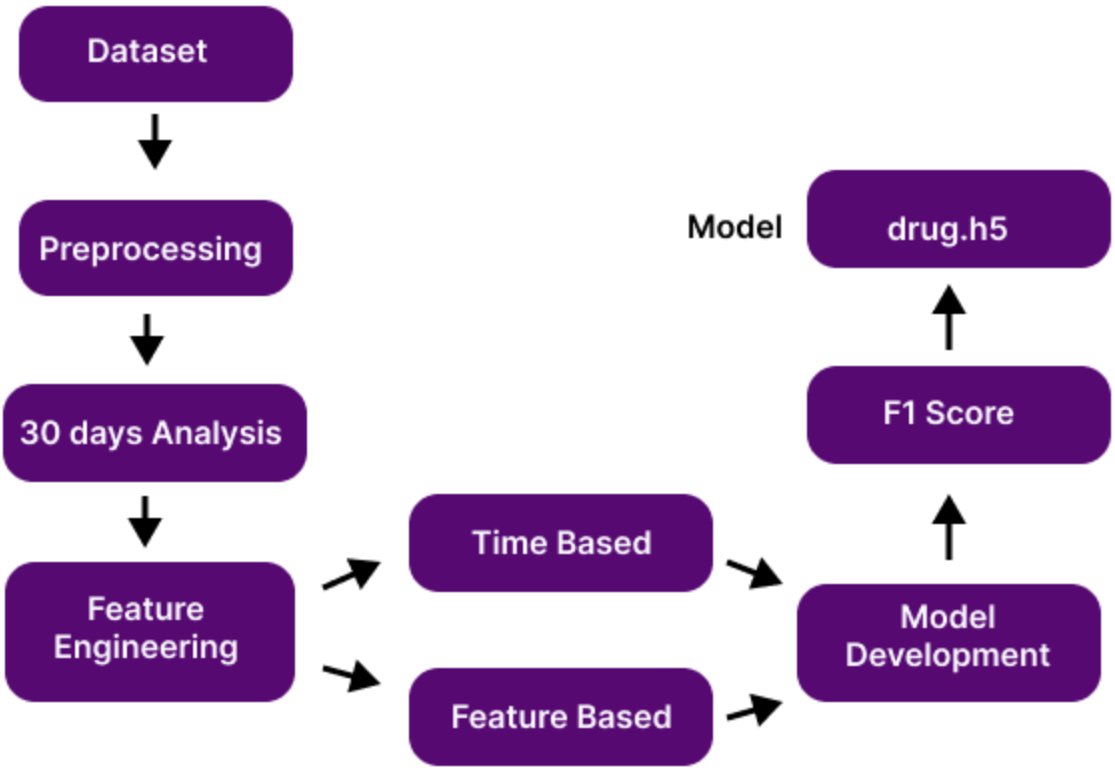
Problem Statement Problem 1:

The development of drugs is critical in providing therapeutic options for patients suffering from chronic and terminal illnesses. “Target Drug”, in particular, is designed to enhance the patient's health and well-being without causing dependence on other medications that could potentially lead to severe and life-threatening side effects. These drugs are specifically tailored to treat a particular disease or condition, offering a more focused and effective approach to treatment, while minimising the risk of harmful reactions. The objective in this assignment is to develop a predictive model which will predict whether a patient will be eligible*** for “Target Drug” or not in next 30 days. Knowing if the patient is eligible or not will help physician treating the patient make informed decision on the which treatments to give. *** - A patient is considered eligible for a particular drug when they have taken their first prescription for that drug. Below table gives an example

Patient-Uid	Date	Incident
a0db1e73-1c7c-11ec-ae39-16262ee38c7f	2015-09-22	DRUG_TYPE_7
a0db1e73-1c7c-11ec-ae39-16262ee38c7f	2018-04-13	SYMPTOM_TYPE_2
a0db1e73-1c7c-11ec-ae39-16262ee38c7f	2018-05-02	DRUG_TYPE_7
a0db1e73-1c7c-11ec-ae39-16262ee38c7f	2018-11-23	TARGET DRUG
a0db1e73-1c7c-11ec-ae39-16262ee38c7f	2018-12-30	TARGET DRUG

In above example, we see that the patient took his first prescription of "Target Drug" on 2018-11-23, so it can be assumed that on this particular day the patient became eligible for "Target Drug". Please follow below steps for developing the model - A. Come up with a positive and negative set for developing the model, here the positive point is the patient who has taken 'Target Drug". Make sure you are also taking into account the time aspect while coming up with a positive & negative set because the aim is to predict 30 days in advance whether a patient is going to be eligible or not. B. Come up with the right kind of feature engineering for developing your model. The features can be frequency-based, time-based etc. If possible can also leverage deep learning techniques C. Evaluate the model on validation set & come up with the right strategy to reduce false positives & false negatives. D. Once you have developed your predictive model, use your model to generate predictions for patients in test.parquet, each patient in test.parquet should be labelled as 1 or 0 using your predictive model and the final generated predictions should be submitted in final_submission.csv E. The evaluation metric for the assignment is F1-Score (candidates with the highest F1 score would be prioritized)

Process Flow:



Pipeline for test dataset:

Implementation

Initial analysis of data:

- Loading the dataset

	Patient-Uid	Date	Incident
0	a0db1e73-1c7c-11ec-ae39-16262ee38c7f	2019-03-09	PRIMARY_DIAGNOSIS
1	a0dc93f2-1c7c-11ec-9cd2-16262ee38c7f	2015-05-16	PRIMARY_DIAGNOSIS
3	a0dc94c6-1c7c-11ec-a3a0-16262ee38c7f	2018-01-30	SYMPTOM_TYPE_0
4	a0dc950b-1c7c-11ec-b6ec-16262ee38c7f	2015-04-22	DRUG_TYPE_0
8	a0dc9543-1c7c-11ec-bb63-16262ee38c7f	2016-06-18	DRUG_TYPE_1

- Calculating the total number of distinct patients and the total number of distinct incident

- Grouping the data by Patient-Uid and Date

	Patient-Uid	Incident
0	a0db1e73-1c7c-11ec-ae39-16262ee38c7f	DRUG_TYPE_7 SYMPTOM_TYPE_2 DRUG_TYPE_7 SYMPTOM...
1	a0dc93f2-1c7c-11ec-9cd2-16262ee38c7f	DRUG_TYPE_0 DRUG_TYPE_2 DRUG_TYPE_0 PRIMARY_DI...
2	a0dc94c6-1c7c-11ec-a3a0-16262ee38c7f	DRUG_TYPE_0 PRIMARY_DIAGNOSIS DRUG_TYPE_7 DRUG...
3	a0dc950b-1c7c-11ec-b6ec-16262ee38c7f	DRUG_TYPE_0 DRUG_TYPE_7 DRUG_TYPE_2 PRIMARY_DI...
4	a0dc9543-1c7c-11ec-bb63-16262ee38c7f	DRUG_TYPE_1 TEST_TYPE_1 SYMPTOM_TYPE_8 DRUG_TY...
...	...	...
27028	a0f0d4c5-1c7c-11ec-bfec-16262ee38c7f	DRUG_TYPE_6 DRUG_TYPE_0 DRUG_TYPE_6 DRUG_TYPE_...
27029	a0f0d4f4-1c7c-11ec-b144-16262ee38c7f	DRUG_TYPE_6 DRUG_TYPE_8 DRUG_TYPE_1 DRUG_TYPE_...
27030	a0f0d523-1c7c-11ec-89d2-16262ee38c7f	DRUG_TYPE_6 DRUG_TYPE_1 DRUG_TYPE_9 DRUG_TYPE_...
27031	a0f0d553-1c7c-11ec-a70a-16262ee38c7f	DRUG_TYPE_9 SYMPTOM_TYPE_7 DRUG_TYPE_2 DRUG_TY...
27032	a0f0d582-1c7c-11ec-a6c1-16262ee38c7f	DRUG_TYPE_6 DRUG_TYPE_1 DRUG_TYPE_6 DRUG_TYPE_...

27033 rows × 2 columns

Positive-set and Negative-set segregation:

We have organized each patient's information into a single row, and their entire history is now sorted by timestamps and stored as features. Additionally, the classes have been segregated and stored in the "**classes**" column **0's** and **1's.**

	Patient-Uid	Incident	classes
0	a0db1e73-1c7c-11ec-ae39-16262ee38c7f	DRUG_TYPE_7 SYMPTOM_TYPE_2 DRUG_TYPE_7 SYMPTOM...	0
1	a0dc93f2-1c7c-11ec-9cd2-16262ee38c7f	DRUG_TYPE_0 DRUG_TYPE_2 DRUG_TYPE_0 PRIMARY_DI...	0
2	a0dc94c6-1c7c-11ec-a3a0-16262ee38c7f	DRUG_TYPE_0 PRIMARY_DIAGNOSIS DRUG_TYPE_7 DRUG...	0
3	a0dc950b-1c7c-11ec-b6ec-16262ee38c7f	DRUG_TYPE_0 DRUG_TYPE_7 DRUG_TYPE_2 PRIMARY_DI...	0
4	a0dc9543-1c7c-11ec-bb63-16262ee38c7f	DRUG_TYPE_1 TEST_TYPE_1 SYMPTOM_TYPE_8 DRUG_TY...	0
...	...	...	...
27028	a0f0d4c5-1c7c-11ec-bfec-16262ee38c7f	DRUG_TYPE_6 DRUG_TYPE_0 DRUG_TYPE_6 DRUG_TYPE_...	1
27029	a0f0d4f4-1c7c-11ec-b144-16262ee38c7f	DRUG_TYPE_6 DRUG_TYPE_8 DRUG_TYPE_1 DRUG_TYPE_...	1
27030	a0f0d523-1c7c-11ec-89d2-16262ee38c7f	DRUG_TYPE_6 DRUG_TYPE_1 DRUG_TYPE_9 DRUG_TYPE_...	1
27031	a0f0d553-1c7c-11ec-a70a-16262ee38c7f	DRUG_TYPE_9 SYMPTOM_TYPE_7 DRUG_TYPE_2 DRUG_TY...	1
27032	a0f0d582-1c7c-11ec-a6c1-16262ee38c7f	DRUG_TYPE_6 DRUG_TYPE_1 DRUG_TYPE_6 DRUG_TYPE_...	1

Analysis for 30 days:

Some patients have already used the target drug, so they are considered part of the positive group. Now, our goal is to predict whether a patient will be eligible for this drug within the next 30 days. To achieve this, we need to exclude or mask out the records of those patients who have already taken the "target_drug." This is because we want to focus on the data leading up to the 30 days before they take the drug, enabling us to train our model effectively to make predictions without prior knowledge of a patient directly using the drug.

- Iterating through the dates of the patient records and checking if he has taken the “TARGET_DRUG”.
- If they have taken it , note down the index of the date(or drug) taken before 30 days of the date of the “TARGET_DRUG”.
- Now all these position indexes are stored in the list called “cutoff_index_list”
- During the analysis we have encountered a patient, who has takent the “TARGET_DRUG” before 30 days itself like with in 6 days of the joining the clinic or atleast when the records are started on their name.
- Finally we also note down those patients-id also to remove them. As their data is of no use and are treated as outliers.

```
print("Patient count who taken TARGET_DRUG :", target_count)

print("Patients who are administered TARGET_DRUG within 30 days of given records itself :", outlier_patients)
```

```
Patient count who taken TARGET_DRUG : 9374
Patients who are administered TARGET_DRUG within 30 days of given records itself : 1
```

Feature Engineering:

- **Time-based features:** These features involve analyzing the time intervals between consecutive tests, drug doses, or symptom records. This is important because these time gaps might provide insights into the emergence of the disease that leads to the use of the TARGET_DRUG.

Time based feature engineering

In [32]:
```
train_df=train_df[train_df['Patient-Uid']!='a0ee64e1-1c7c-11ec-becd-16262ee38c7f']
grouped= train_df.groupby('Patient-Uid')
print(grouped.ngroups)
```

```
27032
```

In [33]:
```
# Define pds as an iterator over your patient data
# Example data structure:
# pds = [(patient_id_1, patient_data_1), (patient_id_2, patient_data_2), ...]

count = 0
Time = []

# Iterate through the records of each patient
for patient_id, patient_data in grouped:
    # List to store the gap history of each patient
    tem = []
    # Loop through the patient history
    for j in range(train_updated_group['cutoff_indices'].iloc[count] - 1):
        # Calculate the difference between consecutive dates in days
        gap = (patient_data["Date"].iloc[j + 1] - patient_data["Date"].iloc[j]).days
        tem.append(gap)

    # Pad the gap vector with zeros to match the max_length
    tem.extend([0] * (max(cutoff_index_list) - len(tem)))

    # Finally, add the patient's gap history to the list
    Time.append(tem)

    count += 1  # Increment the count for the next patient
```

- **Frequency-based features:** These features involve assessing how frequently a specific symptom or drug has been utilized by a patient. This information can be valuable in distinguishing between patients classified as positive (those who have taken the TARGET_DRUG) and negative (those who haven't).

Applying Tfidf vectorizer

In [45]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(ngram_range=(1,4), lowercase=False, max_features=1000)
vectorizer.fit(X_train['Incident_Drug'].values) # fit has to happen only on train data

vectorizer.get_feature_names_out()[:10]
```

Out[45]:
```
array(['DRUG_TYPE_0', 'DRUG_TYPE_0 DRUG_TYPE_0',
       'DRUG_TYPE_0 DRUG_TYPE_0 DRUG_TYPE_0',
       'DRUG_TYPE_0 DRUG_TYPE_0 DRUG_TYPE_0 DRUG_TYPE_0',
       'DRUG_TYPE_0 DRUG_TYPE_0 DRUG_TYPE_0 DRUG_TYPE_1',
       'DRUG_TYPE_0 DRUG_TYPE_0 DRUG_TYPE_0 DRUG_TYPE_2',
       'DRUG_TYPE_0 DRUG_TYPE_0 DRUG_TYPE_0 DRUG_TYPE_6',
       'DRUG_TYPE_0 DRUG_TYPE_0 DRUG_TYPE_0 PRIMARY_DIAGNOSIS',
       'DRUG_TYPE_0 DRUG_TYPE_0 DRUG_TYPE_1',
       'DRUG_TYPE_0 DRUG_TYPE_0 DRUG_TYPE_1 DRUG_TYPE_0'], dtype=object)
```

In [46]:
```
X_train_tfidf = vectorizer.transform(X_train['Incident_Drug'].values)
X_test_tfidf = vectorizer.transform(X_test['Incident_Drug'].values)
```

In [47]:
```
print(X_train_tfidf.shape, X_test_tfidf.shape)
```

```
(21625, 1000) (5407, 1000)
```

Model Building:

LSTMs are designed to overcome some of the limitations of traditional RNNs, which struggle with capturing long-range dependencies in sequential data. LSTMs use a more complex cell structure with gating mechanisms, allowing them to learn and remember information over longer sequences. This makes them highly effective for tasks where the order and timing of events matter.

In our problem, where you are working with medical data and trying to predict patient eligibility for a specific drug, LSTMs can be valuable. You can input patient data over time, such as test results, symptom records, and drug doses, into the LSTM model. The model learns to extract relevant patterns and relationships in this sequential data, helping you make predictions about whether a patient will be eligible for the target drug within a certain time frame, such as 30 days.

```
Model: "model"
_________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
=================================================================================================
 Input_layer_for_LSTM_based  [(None, 1645)]            0         []
 _features (InputLayer)

 Embedding_layer_1 (Embeddi  (None, 1645, 128)         7296      ['Input_layer_for_LSTM_based_f
 ng)                                                             eatures[0][0]']

 LSTM_layer_1 (LSTM)         (None, 1645, 64)          49408     ['Embedding_layer_1[0][0]']

 Input_layer_for_frequency_  [(None, 1000)]            0         []
 based_features (InputLayer
 )

 Flatten_layer_1 (Flatten)   (None, 105280)            0         ['LSTM_layer_1[0][0]']

 Dense_for_tfidf_input (Den  (None, 256)               256256    ['Input_layer_for_frequency_ba
 se)                                                             sed_features[0][0]']

 concatenate (Concatenate)   (None, 105536)            0         ['Flatten_layer_1[0][0]',
                                                                  'Dense_for_tfidf_input[0][0]'
                                                                 ]

 Dense_layer_after_concat (  (None, 256)               2701747   ['concatenate[0][0]']
 Dense)                                                2

 Dropout_1 (Dropout)         (None, 256)               0         ['Dense_layer_after_concat[0][
                                                                 0]']

 Dense_2 (Dense)             (None, 128)               32896     ['Dropout_1[0][0]']

 Dropout_2 (Dropout)         (None, 128)               0         ['Dense_2[0][0]']

 Dense_3 (Dense)             (None, 64)                8256      ['Dropout_2[0][0]']

 Input_layer_for_Time_based  [(None, 1645)]            0         []
 _features (InputLayer)

 Output_layer_for_binary_cl  (None, 1)                 65        ['Dense_3[0][0]']
 assification (Dense)

=================================================================================================
Total params: 27371649 (104.41 MB)
Trainable params: 27371649 (104.41 MB)
Non-trainable params: 0 (0.00 Byte)
```

Adam (short for Adaptive Moment Estimation) is one such optimizer commonly used for training neural networks, including those based on LSTM (Long Short-Term Memory) cells. Adam is a popular choice for optimization because it combines the benefits of two other widely used optimizers, namely AdaGrad and RMSprop, to offer efficient and adaptive learning rates.

```
opt = tf.keras.optimizers.Adam(learning_rate=0.0001)
model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
# Specify the batch size in model.fit
batch_size = 16
model.fit(train_X, train_y, epochs=10, validation_data = (test_X,test_y),batch_size=16)
```

F1 Score:

The F1 score achieved by our model on the medical data is 80 percent. This F1 score indicates a strong performance in terms of both precision and recall, making it a valuable metric for evaluating the model's effectiveness in a medical context. An F1 score of 80 percent signifies that our model strikes a good balance between correctly identifying positive cases, such as disease diagnoses or treatment eligibility, and minimizing both false positives and false negatives. This is crucial in healthcare applications where accurate predictions are essential for patient well-being and clinical decision-making. Our model's performance is a promising result, but it should be further validated and fine-tuned to ensure its reliability and effectiveness in real-world medical scenarios

In [63]:
```
from sklearn.metrics import f1_score

predictions = model.predict(test_X)


threshold = 0.5
binary_predictions = (predictions > threshold).astype(int)


f1 = f1_score(test_y, binary_predictions)

print("F1 Score:", f1)
```

```
169/169 [==============================] - 41s 238ms/step
F1 Score: 0.8019297775395337
```

Saving the model:

```
model.save("drug.h5")
```

The trained model for the given data is saved.

Pipeline for test dataset:

A typical pipeline for processing a test dataset in a machine learning or deep learning project involves several steps. Here's an outline of the pipeline for a test dataset:

1. **Data Loading**: Load the test dataset into memory. This could be in various formats, such as CSV, JSON, or database queries, depending on your data source.
2. **Data Preprocessing**: Apply the same data preprocessing steps that you applied to the training dataset.
3. **Model Loading**: Use previously saved your trained model **("drug.h5"),** load it into memory using the appropriate library and function (e.g., **load_model** in Keras).
4. **Final generated prediction**

Patient-Uid	label
a0f9e8a9-1c7c-11ec-8d25-16262ee38c7f	0
a0f9e9f9-1c7c-11ec-b565-16262ee38c7f	0
a0f9ea43-1c7c-11ec-aa10-16262ee38c7f	0
a0f9ea7c-1c7c-11ec-af15-16262ee38c7f	0
a0f9eab1-1c7c-11ec-a732-16262ee38c7f	0
a0f9eae4-1c7c-11ec-9bdb-16262ee38c7f	1
a0f9eb1c-1c7c-11ec-a373-16262ee38c7f	0
a0f9eb4d-1c7c-11ec-8c2b-16262ee38c7f	0
a0f9eb80-1c7c-11ec-91c3-16262ee38c7f	0
a0f9ebb1-1c7c-11ec-882f-16262ee38c7f	0
a0f9ebe4-1c7c-11ec-bff2-16262ee38c7f	0
a0f9ec45-1c7c-11ec-9011-16262ee38c7f	1
a0f9ec78-1c7c-11ec-9c94-16262ee38c7f	0
a0f9eca8-1c7c-11ec-80ec-16262ee38c7f	0
a0f9ecdf-1c7c-11ec-8ef1-16262ee38c7f	1