National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Sneha Bhatgaonkar
Student ID: x22228136

School of Computing
National College of Ireland

Supervisor: Mohammed Hasanuzzaman

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Sneha Bhatgaonkar		
Student ID:	x22228136		
Programme:	Data Analytics	**Year:**	2025
Module:	Research Project		
Lecturer:	Mohammed Hasanuzzaman		
Submission Due Date:	29/01/2025		
Project Title:	Configuration Manual		
Word Count:	1135		
Page Count:	21		

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Sneha Bhatgaonkar
Date:	29/01/2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Sneha Bhatgaonkar
x22228136

1 Introduction

The configuration manual consists of the necessary steps to build the project. It provides guidelines for running the solution artifact designed to address the research questions presented in research project titled '*Decoding Online Pharmacy Trends: Clustering, Prediction and Business Insights*'. The manual covers environment specification, data sources, data pre-processing activities, model building and model outcome.

2 Environment Specifications

This project is built using 64-bit Windows 10 operating system.

2.1 Hardware

- Operating System: Windows 10 64-bit
- Processor: 11th Gen Intel(R) Core (TM) i5-11300H @ 3.10GHz 2.61 GHz
- RAM: 16GB
- Storage: 512GB
- GPU: NVIDIVA GeForce MX450

2.2 Software

- Microsoft Excel
- Jupyter Notebook
- Visual Studio

In this project following libraries, packages are used for data computation, visualization, cross-validation, classification using machine learning algorithms and model performance evaluation.

Pre-requisite:

1. This project requires transformers, torch -> pip install transformers torch (transformers version 4.42.4 is used in current study)

2. Environment required for spacy Med7: spaCy version >=3.1 and Python >=3.7

Task-specific packages and libraries:

2.2.1 Data computation:

```
import re
import pandas as pd
from ordered_set import OrderedSet
import matplotlib.pyplot as plt
import numpy as np
from wordcloud import WordCloud
import seaborn as sns
```

These imports are used for data-related activities, such as cleaning, creating regular expressions, and visualizing data using seaborn library.

2.2.2 Models for classification and clustering:

```
#Clustering
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

#Split data
from sklearn.model_selection import train_test_split

#Cross validation
from sklearn.model_selection import cross_val_score, StratifiedKFold, GridSearchCV,
RandomizedSearchCV, cross_validate

#Models
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from imblearn.ensemble import EasyEnsembleClassifier

#Sampling
from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTE

#Metrics
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score,classification_report,
roc_auc_score,roc_curve


```

- To split the data (train_test_split).
- To apply techniques like cross-validation (cross_val_score, StratifiedKFold), hyperparameter tuning (GridSearchCV, RandomizedSearchCV), and address class imbalance (RandomOverSampler).
- Perform classification tasks using machine learning models (e.g., RandomForestClassifier, XGBClassifier, EasyEnsembleClassifier).
- Evaluate model performance using various metrics (accuracy_score, f1_score, roc_auc_score, etc.).

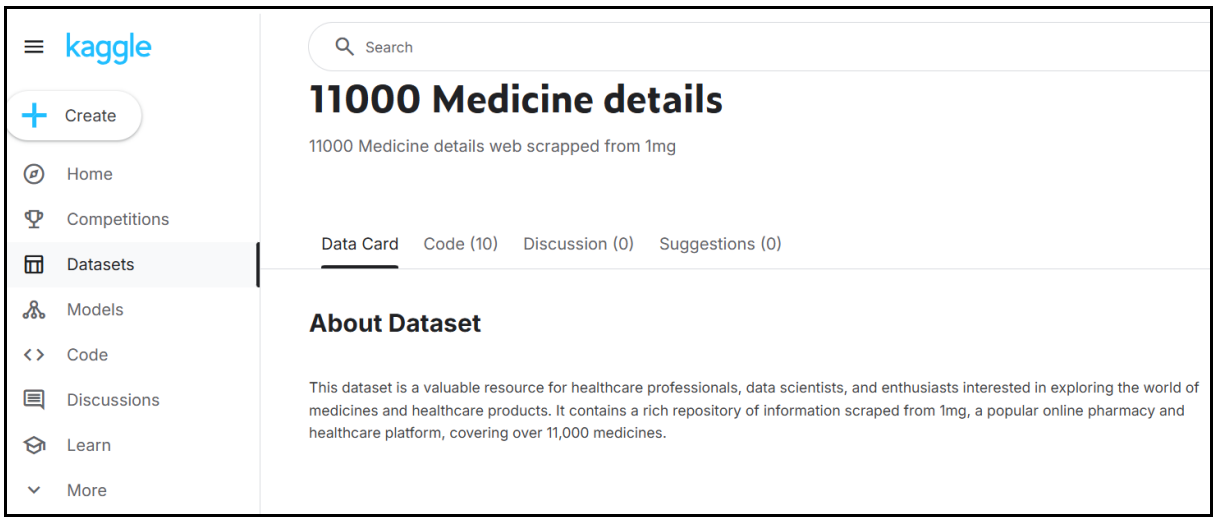
2.2.3 Transformers related imports:

```
#transformers  model, tokenizer imports
from transformers import pipeline,AutoTokenizer,AutoModelForTokenClassification, AutoModel
```

Transformers import is used for model and tokenizer selection to perform tokenization activity.

3 Dataset Overview

Dataset used in this project is a public dataset from Kaggle. It is present at https://www.kaggle.com/datasets/singhnavjot2062001/11000-medicine-details and can be downloaded in csv format. This dataset has 11825 records. And information about Medicine Name, Composition, Uses, Side Effects, Reviews in percentage format.




```
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Medicine Name       11825 non-null  object
 1   Composition         11825 non-null  object
 2   Uses                11825 non-null  object
 3   Side_effects        11825 non-null  object
 4   Image URL           11825 non-null  object
 5   Manufacturer        11825 non-null  object
 6   Excellent Review %  11825 non-null  int64
 7   Average Review %    11825 non-null  int64
 8   Poor Review %       11825 non-null  int64
```

4 Data – Preprocessing

Data cleaning and preprocessing is performed, and details are available in python file named -<1_DataPreprocessing.ipynb>. Each column is analysed and transformed as per need.

1. Named Entity Recognition using Med7 and Clinical-AI-Apollo Medical NER:

a. **Med7** is available as part of Spacy, it is transferable clinical natural language processing model. It is used to separate drug name and medicine form from column Medicine name and stored in columns 'trf_Medicine Name' and 'Form_clean' respectively. Below function is used to retrieve entity and label returned by med7 (en_core_med7_lg) stored in excel and analysed the outcome manually.

```
def getAllEntityAndClass(model,document, columnName):
    i = 0
    table= {columnName:[], "Drug":[], "Dosage":[],"Stregth":[], "Form":[], "Duration":[], "Route":[], "Frequency":[] }
    model = model.load()
    for doc in model.pipe(document):
        use=usesList[i]
        entity=list(doc.ents)
        drug = []
        dosage = []
        stregth =[]
        form=[]
        duration=[]
        route=[]
        frequency=[]
        for ent in doc.ents:
            if ent.label_ == 'DRUG':
                drug.append(ent.text)
            elif ent.label_ == 'STRENGTH':
                stregth.append(ent.text)
            elif ent.label_ == 'FORM':
                form.append(ent.text)
            elif ent.label_ == 'ROUTE':
                route.append(ent.text)
            elif ent.label_ == 'DOSAGE':
                dosage.append(ent.text)
            elif ent.label_ == 'FREQUENCY':
                frequency.append(ent.text)
            elif ent.label_ == 'DURATION':
                duration.append(ent.text)
        table[columnName].append(use)
        table["Drug"].append(drug)
        table["Dosage"].append(dosage)
        table["Stregth"].append(stregth)
        table["Form"].append(form)
        table["Duration"].append(duration)
        table["Route"].append(route)
        table["Frequency"].append(frequency)
        i +=1
    return table
```

b. **Clinical-AI-Apollo Medical NER** is used to retrieve entities related to disease names and side effects from the data. Following function is used for tokenization for Uses and Side effects column.

```
def performTokenClassification(data):
    tokenizer = AutoTokenizer.from_pretrained("Clinical-AI-Apollo/Medical-NER")
    model = AutoModelForTokenClassification.from_pretrained("Clinical-AI-Apollo/Medical-NER")
    nlp_tkn_classifier = pipeline("token-classification", model=model, tokenizer=tokenizer)
    return nlp_tkn_classifier(data)
```

Output returned by above method, was used to retrieve entity and words using below method.

```
def getEntityAndClass_combined(result, columnName):
    #j=0
    #columnList = dataFrame[columnName].to_list()
    table= {
           "Entity_DISEASE_DISORDER_"+columnName:[], "Word_DISEASE_DISORDER_"+columnName:[],
           "Entity_DIAGNOSTIC_PROCEDURE_"+columnName:[], "Word_DIAGNOSTIC_PROCEDURE_"+columnName:[],
           "Entity_OTHERS_"+columnName:[], "Word_OTHERS_"+columnName:[]
    }
    for record in result:
        set_word=OrderedSet()
        set_entity=OrderedSet()
        list_entity_SIGN_SYMPTOM=[]
        list_word_SIGN_SYMPTOM=[]

        list_entity_DISEASE_DISORDER=[]
        list_word_DISEASE_DISORDER=[]

        list_entity_DETAILED_DESCRIPTION=[]
        list_word_DETAILED_DESCRIPTION=[]

        list_entity_DIAGNOSTIC_PROCEDURE=[]
        list_word_DIAGNOSTIC_PROCEDURE=[]

        list_entity_OTHERS=[]
        list_word_OTHERS=[]
        #print(len(record))
        for i in range(len(record)):
            #print(record[i]['entity'])
            #print(record[i]['word'])
            if((record[i]['entity']=='B-DISEASE_DISORDER') | (record[i]['entity']=='I-DISEASE_DISORDER') |
              (record[i]['entity']=='B-SIGN_SYMPTOM') | (record[i]['entity']=='I-SIGN_SYMPTOM') |
               (record[i]['entity']=='B-DETAILED_DESCRIPTION') | (record[i]['entity']=='I-DETAILED_DESCRIPTION')|
              (record[i]['entity']=='B-OUTCOME')):
                list_entity_DISEASE_DISORDER.append(record[i]['entity'])
                list_word_DISEASE_DISORDER.append(re.sub("[-_]", "", record[i]['word']))
            elif((record[i]['entity']=='B-DIAGNOSTIC_PROCEDURE') | (record[i]['entity']=='I-DIAGNOSTIC_PROCEDURE')):
                list_entity_DIAGNOSTIC_PROCEDURE.append(record[i]['entity'])
                list_word_DIAGNOSTIC_PROCEDURE.append(re.sub("[-_]", "", record[i]['word']))
            else:
                list_entity_OTHERS.append(record[i]['entity'])
                list_word_OTHERS.append(re.sub("[-_]", "", record[i]['word']))

        #list_entity.append(list(set_entity))
        #list_word.append(list(set_word))
        #value=columnList[j]
        #table[NewColumnName].append(value)

        table["Entity_DISEASE_DISORDER_"+columnName].append(list_entity_DISEASE_DISORDER)
        table["Word_DISEASE_DISORDER_"+columnName].append(list_word_DISEASE_DISORDER)

        table["Entity_DIAGNOSTIC_PROCEDURE_"+columnName].append(list_entity_DIAGNOSTIC_PROCEDURE)
        table["Word_DIAGNOSTIC_PROCEDURE_"+columnName].append(list_word_DIAGNOSTIC_PROCEDURE)

        table["Entity_OTHERS_"+columnName].append(list_entity_OTHERS)
        table["Word_OTHERS_"+columnName].append(list_word_OTHERS)
        #j+=1
    return table
```

Data was analysed and then stored in columns 'disease_clean_lower' and ‘clean_SIGN_SYMPTOM_Side_eff’, after performing subsequent cleaning activities.

In some cases, data was not identified correctly by model, in such scenarios, data was replaced with original text.

2. Manufacturer names were cleaned by removing suffix mentioned as ltd or limited etc.

3. Excellent Review, Average Review and Poor Review columns were combined to form overall score using weighted average technique with values between 1 to 5. Then this column was transformed to ‘review_label’ with two values ‘positive’ (for values >=3) and ‘negative’ (for values <3). ‘review_label’ is target variable.

4. Component column was analysed and separated component and dosages by using following function:

```python
def extract_components_dosages(med_string):
    # Componet column has values with componet name and dosage ,
    # dosage are present brackets for most of the records just after compenet name
    # If record has more than one component , then + sign is seen in most of the records
    #here, we are separating componet and dosage.
    components = med_string.split(' + ')
    result = {}

    # Looping through each component to extract the name and dosage
    for idx, component in enumerate(components):
        # Using  regex to split component and dosage
        match = re.match(r"([a-zA-Z0-9-\s]+)\(([^)]+)\)", component)
        if match:
            name = match.group(1).strip()
            dosage = match.group(2).strip()
            # Adding to result dic
            result[f'Component {idx+1}'] = name
            result[f'Dosage {idx+1}'] = dosage
        else:
            result[f'Component {idx+1}'] = component.strip()
            result[f'Dosage {idx+1}'] = 'N/A'
    return result
```

5 Clustering

Cleaned data is used to perform clustering using KMeans along with TF-IDF and PCA.
Below is code snippet.

```python
def transformTextToVector(dataFrame, columnName):
    vectorizer = TfidfVectorizer()
    X = vectorizer.fit_transform(dataFrame[columnName])
    return X.toarray()
```

```python
def analyze_kmeans(data, kmeans_stop_point):
    wcss =[]
    for i in range (3,kmeans_stop_point):
        kmeans = KMeans(n_clusters=i, init = 'k-means++', random_state=43)
        kmeans.fit(data)
        wcss.append(kmeans.inertia_)
        silhouette_avg = silhouette_score(data, kmeans.labels_)
        print(
            "For n_clusters =",
            i,
            "The average silhouette_score is :",
            silhouette_avg,
        )
```

Below outcome shows with k=7 Silhouette score is 0.653

```
pca = PCA(2)
pca_combined = pca.fit_transform(v_combined)
pca_combined.shape
```

```
(11741, 2)
```

```
analyze_kmeans(pca_combined,11)
```

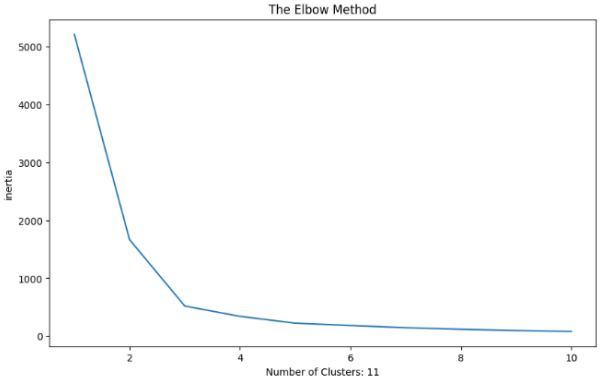
```
C:\Users\MyPC\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
For n_clusters = 3 The average silhouette_score is : 0.7923450784350792
C:\Users\MyPC\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
For n_clusters = 4 The average silhouette_score is : 0.7428857795067821
C:\Users\MyPC\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
For n_clusters = 5 The average silhouette_score is : 0.6358926014530275
C:\Users\MyPC\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
For n_clusters = 6 The average silhouette_score is : 0.6354557359372603
C:\Users\MyPC\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
For n_clusters = 7 The average silhouette_score is : 0.6530339313066416
```

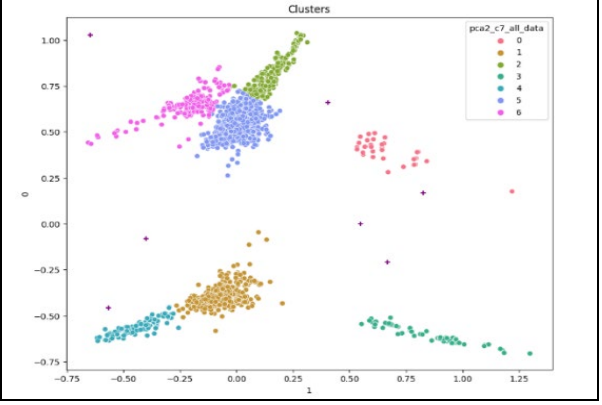
Labels obtained using K-Means are stored in column ‘pca2_c7_all_data’

```
kmeans_pca2_c7 = KMeans(n_clusters=7, init = 'k-means++', random_state=43)
kmeans_pca2_c7.fit(pca_combined)
data['pca2_c7_all_data'] = kmeans_pca2_c7.labels_.tolist()
```

Below is elbow plot and cluster visualization

```
def plotElbowGraph(numberOfClusters, data):
    cs = []
    plt.figure(figsize=(10,6))
    for i in range(1, numberOfClusters):
        kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 43)
        kmeans.fit(data)
        cs.append(kmeans.inertia_)
    plt.plot(range(1, numberOfClusters), cs)
    plt.title('The Elbow Method')
    plt.xlabel('Number of Clusters: {}'.format(numberOfClusters))
    plt.ylabel('inertia')
    plt.show()
```





6 Cluster-wise Data Analysis

Below is data info after performing data cleaning and pre-processing

```
---  ------                          --------------  -----
0    Form_clean                      11741 non-null  object
1    trf_Medicine Name               11741 non-null  object
2    Manufacturer_clean              11741 non-null  object
3    med_weighted_score              11741 non-null  float64
4    review_label                    11741 non-null  object
5    Component 1                     11741 non-null  object
6    Dosage 1                        11741 non-null  object
7    Side_effects_stp                11741 non-null  object
8    disease_clean_lower             11741 non-null  object
9    clean_SIGN_SYMPTOM_Side_eff     11741 non-null  object
10   pca2_c3_all_data                11741 non-null  int64
11   pca2_c7_all_data                11741 non-null  int64
12   gmm_7                           11741 non-null  int64
13   cos_7                           11741 non-null  int64
```

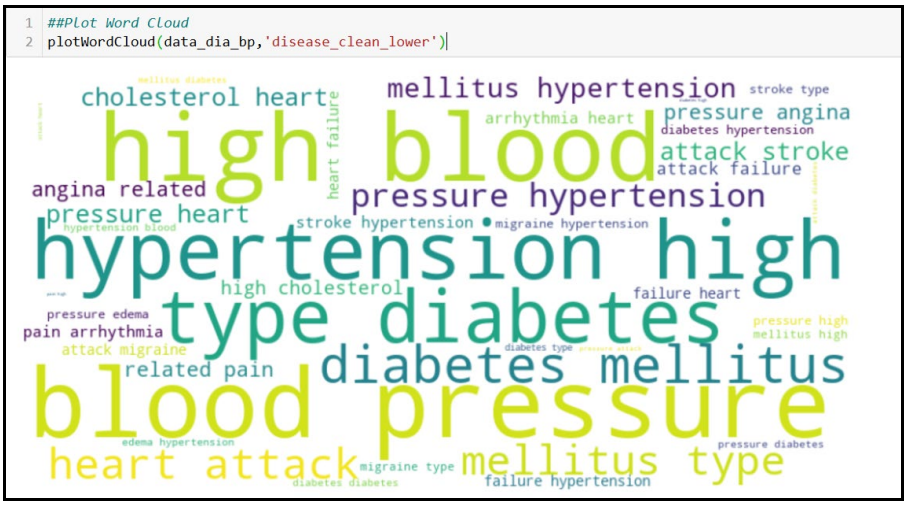
From this we are using following columns:

1. **Form_Clean** -> This contains clean version of various Medicine Forms(Tablet, capsule etc.)
2. **Trf_Medicine Name** -> This contains clean version of Medicine Names
3. **Manufacturer_clean** -> This has information on Manufacturer
4. **Component 1** -> This contains medicine component information
5. **Dosage 1** -> This has information on Dosages
6. **Disease_clean_lower** -> This contains disease names
7. **Clean_SIGN_SYMPTOM_Side_eff** -> This has side-effects related information
8. **Pca2_c7_all data** -> This contains all labels predicted by Kmeans Clustering with n_cluster as 7
9. **Review_label** -> This is target variable with values positive and negative.

Here, Cluster based disease are visualized using word cloud.
Below cluster is formed by combining labels 0,3,4 .

```
1 data_dia_bp = data[(data['pca2_c7_all_data']==0) | (data['pca2_c7_all_data']==3) | (data['pca2_c7_all_data']==4)]
```

```
##Plot Word Cloud
plotWordCloud(data_dia_bp,'disease_clean_lower')
```

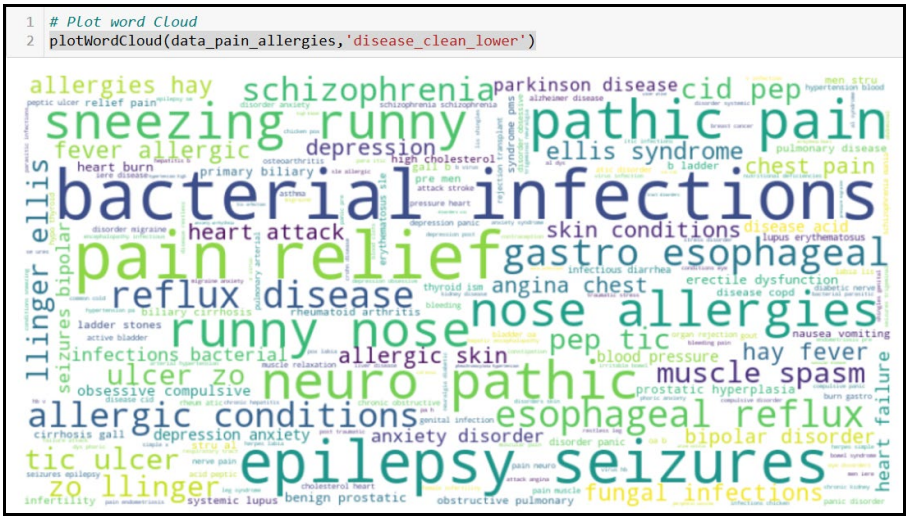


Cluster 1 with Diabetes and hypertension

Below cluster contains label values 1

```
data_pain_allergies = data[(data['pca2_c7_all_data']==1)]
```

```
# Plot word Cloud
plotWordCloud(data_pain_allergies,'disease_clean_lower')
```

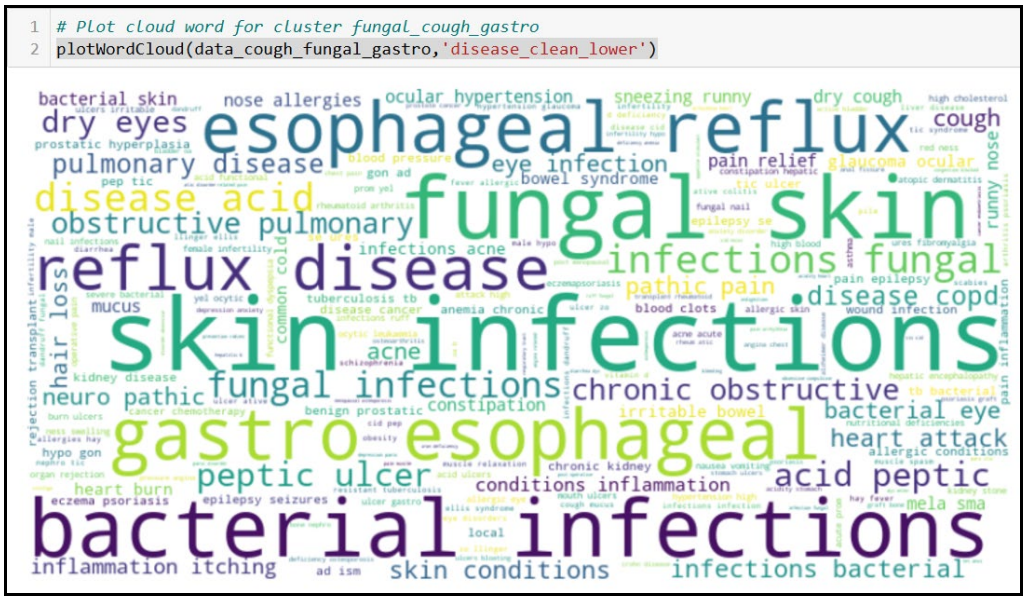


Cluster 2 with pain relief, bacterial infections, allergies

Below cluster is formed by combining label values 2,5,6

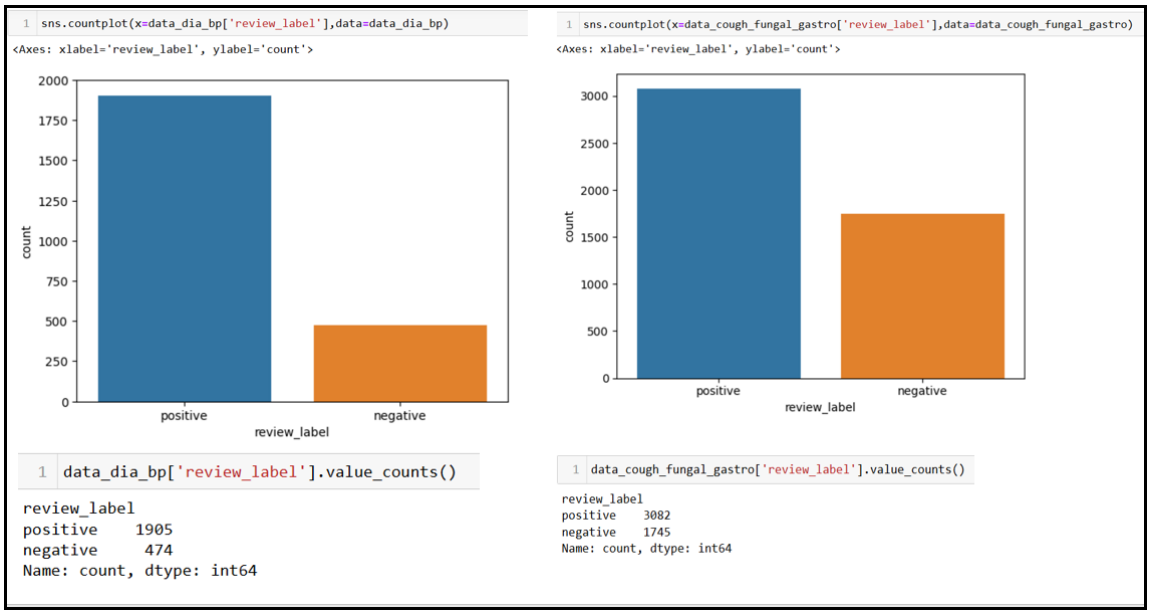
```
data_cough_fungal_gastro = data[(data['pca2_c7_all_data']==2) | (data['pca2_c7_all_data']==5) |
                                (data['pca2_c7_all_data']==6)]
```

```
# Plot cloud word for cluster fungal_cough_gastro
plotWordCloud(data_cough_fungal_gastro,'disease_clean_lower')
```



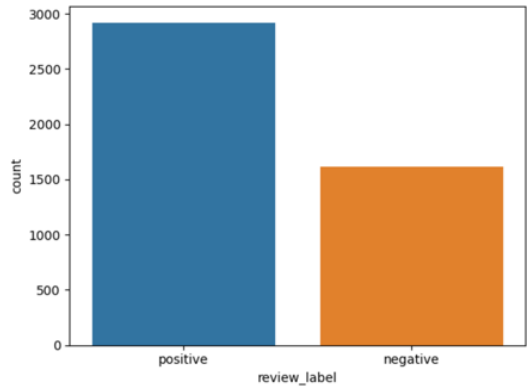
Cluster 3 with fungal skin infections and gastro

Review label distribution per cluster is plotted below:



```
1 sns.countplot(x=data_pain_allergies['review_label'],data=data_pain_allergies)
```

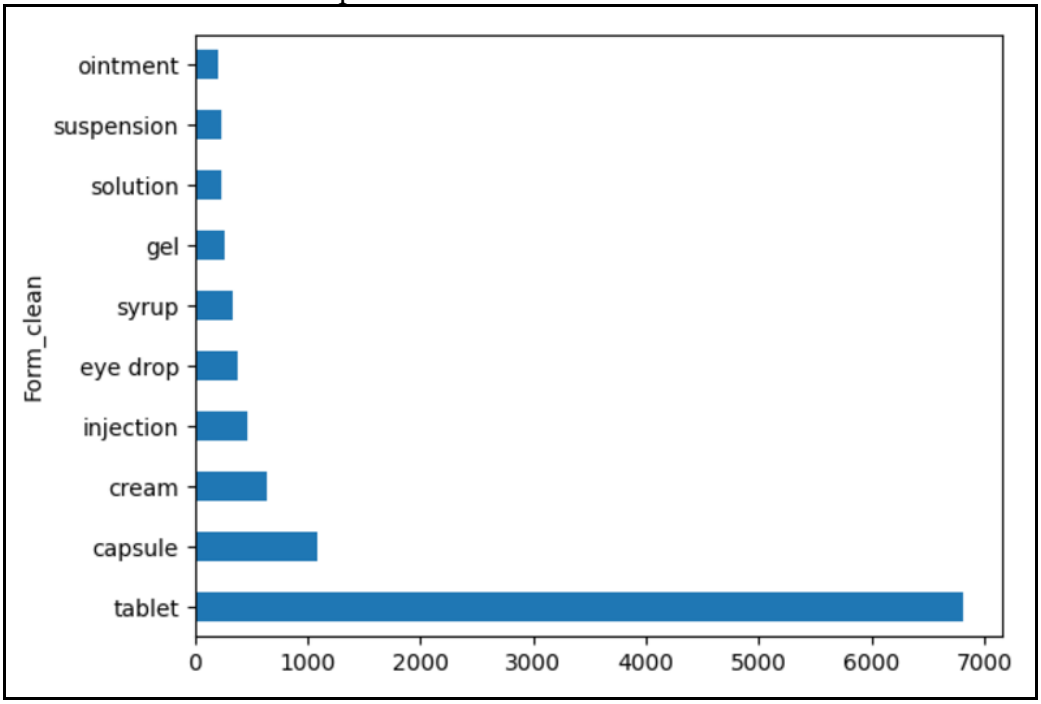
```
<Axes: xlabel='review_label', ylabel='count'>
```




```
1 data_pain_allergies['review_label'].value_counts()
```

```
review_label
positive    2921
negative    1614
Name: count, dtype: int64
```

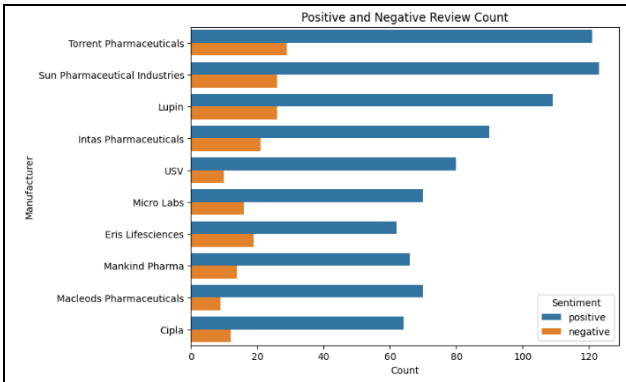
Below is distribution of top 10 medicine forms:



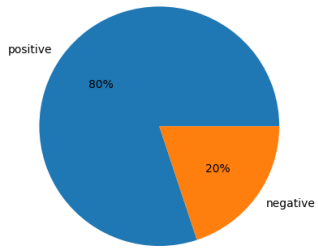
Below is function used to analyse manufacturer distribution based on the reviews:

```python
def analyze_ManufactureData_BasedOnReviews(dataframe):
    lst_manf = []
    lst_pos = []
    lst_neg = []
    data_manufacture_with_reviews = pd.DataFrame()
    data_manufacture_with_reviews['Manufacturer_Name']=''
    data_manufacture_with_reviews['positive']=''
    data_manufacture_with_reviews['negative']=''

    # Get Unique Manufactures:
    uniqueManufactures= dataframe['Manufacturer_clean'].value_counts()

    #Iterate and store positive negative count:
    for x in uniqueManufactures.index :
        review_per_man = dataframe[dataframe['Manufacturer_clean'] == x]['review_label'].value_counts()
        lst_manf.append(x)
        if(len(review_per_man.index)==2):
            lst_pos.append(review_per_man['positive'])
            lst_neg.append(review_per_man['negative'])
        else:
            if (review_per_man.index == 'positive'):
                lst_pos.append(review_per_man['positive'])
                lst_neg.append(0)
            else:
                lst_neg.append(review_per_man['negative'])
                lst_pos.append(0)

    data_manufacture_with_reviews['Manufacturer_Name'] = lst_manf
    data_manufacture_with_reviews['positive']= lst_pos
    data_manufacture_with_reviews['negative']=lst_neg
    return data_manufacture_with_reviews
```

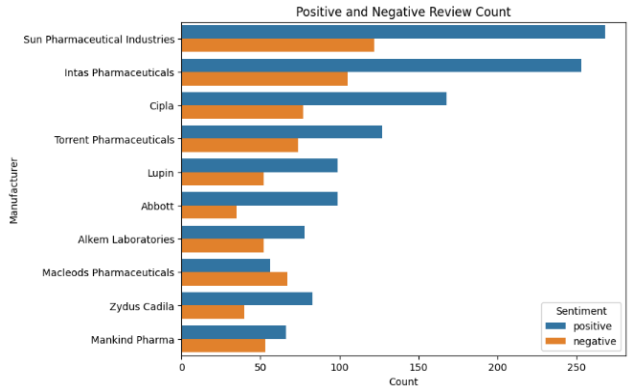
```python
def plotBarChart(data):
    df_melted = pd.melt(data, id_vars=['Manufacturer_Name'], value_vars=['positive', 'negative'],
                        var_name='Sentiment', value_name='Count')

    plt.figure(figsize=(8, 6))
    sns.barplot(data=df_melted, y='Manufacturer_Name', x='Count', hue='Sentiment')

    plt.title('Positive and Negative Review Count')
    plt.xlabel('Count')
    plt.ylabel('Manufacturer')
    plt.legend(title='Sentiment')
```



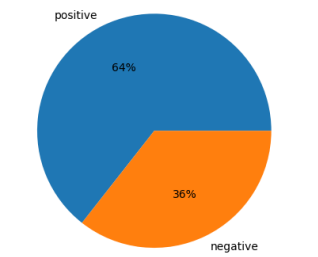
Cluster 1 – Manufacturer performance



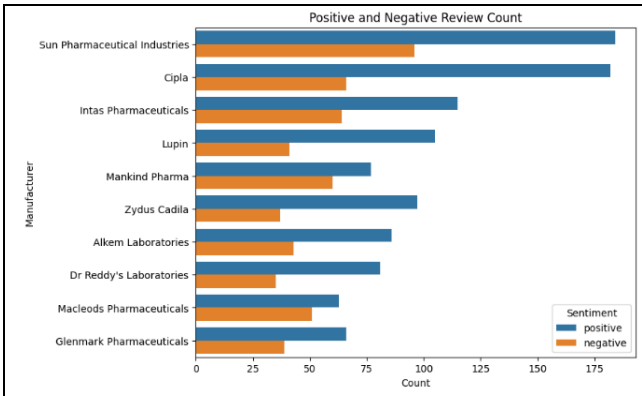
Cluster 1 – Review distribution



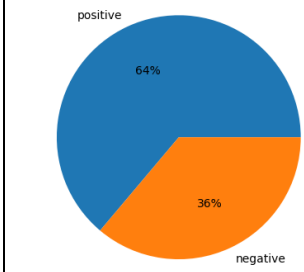
Cluster 2 – Manufacturer performance



Cluster 2 – Review distribution



Cluster 3 – Manufacturer performance



Cluster 3 – Review distribution

7 Embeddings and Classification Models

Classification was performed using Random Forest Classifier, XGBoost, Easy Ensemble classifier with vectorizations – BioBERT, Apollo-AI and BioFormer. Below is code snippet.

7.1.1 Bio-BERT embedding:

```
tokenizer = AutoTokenizer.from_pretrained("dmis-lab/biobert-v1.1")
model = AutoModel.from_pretrained("dmis-lab/biobert-v1.1")
```

```
def get_embeddings_mean(texts):
    inputs = tokenizer(texts, return_tensors="pt", padding=True, truncation=True)
    with torch.no_grad():
        outputs = model(**inputs, output_hidden_states=True)
        last_hidden_state = outputs.last_hidden_state.mean(dim=1).squeeze().numpy()
    return last_hidden_state
```

7.1.2 Bio-Former -16L:

```
tokenizer = AutoTokenizer.from_pretrained("bioformers/bioformer-16L")
model = AutoModel.from_pretrained("bioformers/bioformer-16L")
```

```
# Function to convert text into embeddings
def get_embeddings(text_list):
    inputs = tokenizer(text_list, return_tensors='pt', padding=True, truncation=True, max_length=512)
    with torch.no_grad():
        outputs = model(**inputs)
    return outputs.last_hidden_state[:, 0, :].numpy()
```

7.1.3 Clinical-AI-Apollo:

```
tokenizer = AutoTokenizer.from_pretrained("Clinical-AI-Apollo/Medical-NER")
model = AutoModel.from_pretrained("Clinical-AI-Apollo/Medical-NER")
```

```
# Function to convert text into embeddings
def get_embeddings(text_list):
    inputs = tokenizer(text_list, return_tensors='pt', padding=True, truncation=True, max_length=512)
    with torch.no_grad():
        outputs = model(**inputs)
    return outputs.last_hidden_state[:, 0, :].numpy()
```

7.1.4 Train-Test Split:

```
X_1 = data_dia_bp[['trf_Medicine Name','Form_clean','Manufacturer_clean' , 'disease_clean_lower',
          'Component 1','Dosage 1','clean_SIGN_SYMPTOM_Side_eff' ]]
y_1 = data_dia_bp['review_label']
```

```
X_train_1, X_test_1, y_train_1, y_test_1 = train_test_split(X_1, y_1, test_size=0.2, stratify=y_1, random_state=42)
```

```
print(X_train_1.shape, X_test_1.shape)
```

```
(1903, 7) (476, 7)
```

7.1.5 Vectorization for data:

a. Domain specific vectorization:

```
X_train_uses_1 = get_embeddings(X_train_1['disease_clean_lower'].tolist())
X_test_uses_1 = get_embeddings(X_test_1['disease_clean_lower'].tolist())
print(X_train_uses_1.shape)
print(X_test_uses_1.shape)
#15Mins
```

```
(1903, 768)
(476, 768)
```

```
X_train_side_effects_1 = get_embeddings(X_train_1['clean_SIGN_SYMPTOM_Side_eff'].tolist())
X_test_side_effects_1 = get_embeddings(X_test_1['clean_SIGN_SYMPTOM_Side_eff'].tolist())
print(X_train_side_effects_1.shape)
print(X_test_side_effects_1.shape)
#10 mins
```

```
(1903, 768)
(476, 768)
```

```
X_train_component1_1 = get_embeddings(X_train_1['Component 1'].tolist())
X_test_component1_1 = get_embeddings(X_test_1['Component 1'].tolist())
print(X_train_component1_1.shape)
print(X_test_component1_1.shape)
```

```
(1903, 768)
(476, 768)
```

b. Generic Vectorization:

```
vectorizer = TfidfVectorizer()
X_train_Manufacturer_tf_1 = vectorizer.fit_transform(X_train_1['Manufacturer_clean']).toarray()
X_test_Manufacturer_tf_1 = vectorizer.transform(X_test_1['Manufacturer_clean']).toarray()
print(X_train_Manufacturer_tf_1.shape)
print(X_test_Manufacturer_tf_1.shape)
```

```
(1903, 177)
(476, 177)
```

```

```

```
X_train_MedicineName_tf_1 = vectorizer.fit_transform(X_train_1['trf_Medicine Name']).toarray()
X_test_MedicineName_tf_1 = vectorizer.transform(X_test_1['trf_Medicine Name']).toarray()
print(X_train_MedicineName_tf_1.shape)
print(X_test_MedicineName_tf_1.shape)
```

```
(1903, 925)
(476, 925)
```

```
X_train_Form_tf_1 = vectorizer.fit_transform(X_train_1['Form_clean']).toarray()
X_test_Form_tf_1 = vectorizer.transform(X_test_1['Form_clean']).toarray()
print(X_train_Form_tf_1.shape)
print(X_test_Form_tf_1.shape)
```

```
(1903, 16)
(476, 16)
```

```
X_train_Dosage1_tf_1 = vectorizer.fit_transform(X_train_1['Dosage 1']).toarray()
X_test_Dosage1_tf_1 = vectorizer.transform(X_test_1['Dosage 1']).toarray()
print(X_train_Dosage1_tf_1.shape)
print(X_test_Dosage1_tf_1.shape)
```

```
(1903, 52)
(476, 52)
```

Combine data to create X_train and X_test:

```
X_train_combined_1 = np.concatenate((X_train_MedicineName_tf_1,  X_train_Form_tf_1,
                                     X_train_Manufacturer_tf_1, X_train_uses_1,
                                     X_train_component1_1,X_train_Dosage1_tf_1,
                                     X_train_side_effects_1), axis=1)

X_test_combined_1 = np.concatenate((X_test_MedicineName_tf_1,  X_test_Form_tf_1,
                                    X_test_Manufacturer_tf_1, X_test_uses_1,
                                    X_test_component1_1,X_test_Dosage1_tf_1,
                                    X_test_side_effects_1), axis=1)
```

7.1.6 Various functions:

Cross validation is calculated using following function

```

def crossValidation(model, X_train, y_train, cv):
    scoring = ['accuracy', 'precision', 'recall', 'f1', 'roc_auc']
    cv_score = cross_validate(model, X_train,y_train, scoring = scoring, cv = cv, return_train_score = True, error_score='ra
    df_score = pd.DataFrame(cv_score)
    return df_score.mean()
```

To check classification model output with confusion metrics following function is used.

```python
def getModelOutcome(classifier,X_train,y_train ,X_test,y_test):
    model = classifier

    # Train the model
    model.fit(X_train,y_train)

    # Predict on test set
    y_pred = model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    f1score = f1_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_pred)
    print('accuracy::: ', accuracy)
    print('precision::: ', precision)
    print('F1 score::: ', f1score)
    print('Recall::: ', recall)
    print('ROC_AUC::: ', roc_auc)

    # Print detailed classification report
    print(classification_report(y_test, y_pred))
    cm = confusion_matrix(y_test, y_pred, labels=model.classes_)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                                  display_labels=model.classes_)
    disp.plot()
```

To calculate optimum threshold following function is used:

```python
def getOptimumThreshold(model, X_train, y_train,X_test, y_test):
    step_factor = 0.05
    threshold_value = 0.1
    roc_score=0
    model.fit(X_train, y_train)
    predicted_proba = model.predict_proba(X_test)
    while threshold_value <=0.8:
        temp_thresh = threshold_value
        predicted = (predicted_proba [:,1] >= temp_thresh).astype('int')
        print('Threshold',temp_thresh,'--',roc_auc_score(y_test, predicted))
        if roc_score<roc_auc_score(y_test, predicted
            roc_score = roc_auc_score(y_test, predicted)
            thrsh_score = threshold_value
        threshold_value = threshold_value + step_factor
    print('---Optimum Threshold ---',thrsh_score,'--ROC--',roc_score)
```

To predict the class, as per specific threshold value, following function is used:

```python
def predict_with_threshold(threshold_value, model,X_train, y_train,X_test, y_test):
    model.fit(X_train, y_train)
    y_pred_th = (model.predict_proba(X_test)[:, 1] >= threshold_value)
    accuracy = accuracy_score(y_test, y_pred_th)
    precision = precision_score(y_test, y_pred_th)
    f1score = f1_score(y_test, y_pred_th)
    recall = recall_score(y_test, y_pred_th)
    roc_auc = roc_auc_score(y_test, y_pred_th)
    print('accuracy::: ', accuracy)
    print('precision::: ', precision)
    print('F1 score::: ', f1score)
    print('Recall::: ', recall)
    print('ROC_AUC::: ', roc_auc)

    # Print detailed classification report
    print(classification_report(y_test, y_pred_th))
    cm = confusion_matrix(y_test, y_pred_th, labels=model.classes_)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                                  display_labels=model.classes_)
    disp.plot()
```

8 Results

Random Forest, XGBoost, Easy Ensemble Classifier are used to perform classification using three Vectorization techniques on all three clusters. When results are compared, it is seen that Easy Ensemble with XGBoost estimator performed well compared to Random Forest and XGBoost.

Model configuration used with Easy Ensemble for each cluster and each vectorization are presented below:

8.1 Model Outcomes using BioBERT with Easy Ensemble with XGBoost estimator

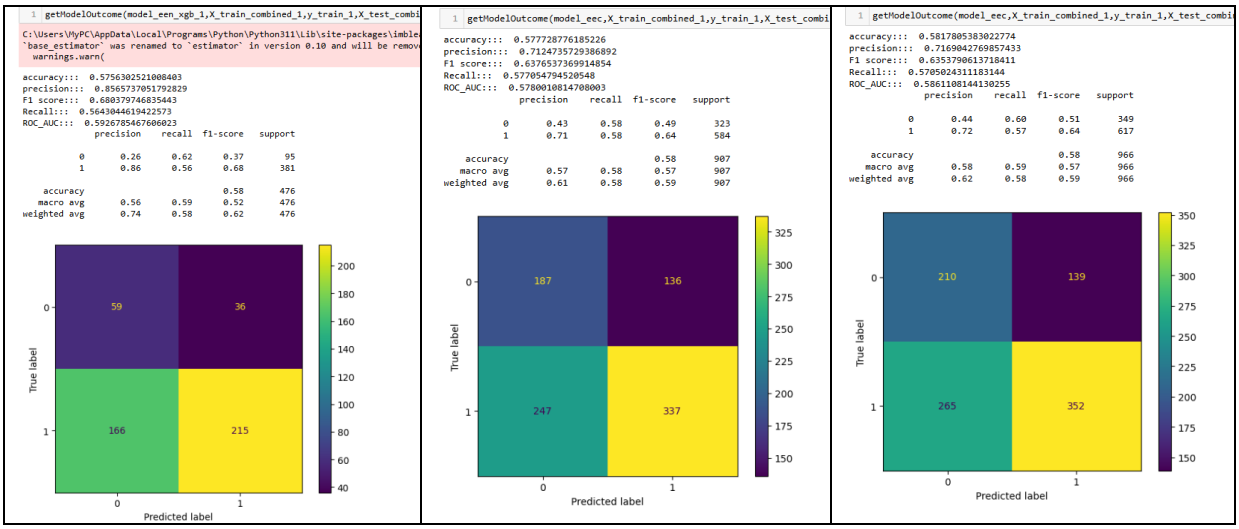
a. Model configuration for Cluster 1 present in 3_Classification_BioBERT-0_3_4_bp_dia.ipynb

```python
import xgboost
model_een_xgb = EasyEnsembleClassifier(n_estimators=10, random_state=43,
                                       estimator=xgboost.XGBClassifier(objective="binary:logistic",random_state=43))
```

b. Model configuration for Cluster 2 and Cluster 3 present in 3_Classification_BioBERT-1_alrg_pain.ipynb, 3_Classification_BioBERT-2_5_6.ipynb respectively

```python
from imblearn.ensemble import EasyEnsembleClassifier
model_eec = EasyEnsembleClassifier(n_estimators=10)
cv_model_eec=crossValidation(model_eec, X_train_combined_1, y_train_1, kf)
cv_model_eec
```

c. Evaluation metrics



8.2 Model Outcomes using BioFormer with Easy Ensemble with XGBoost estimator

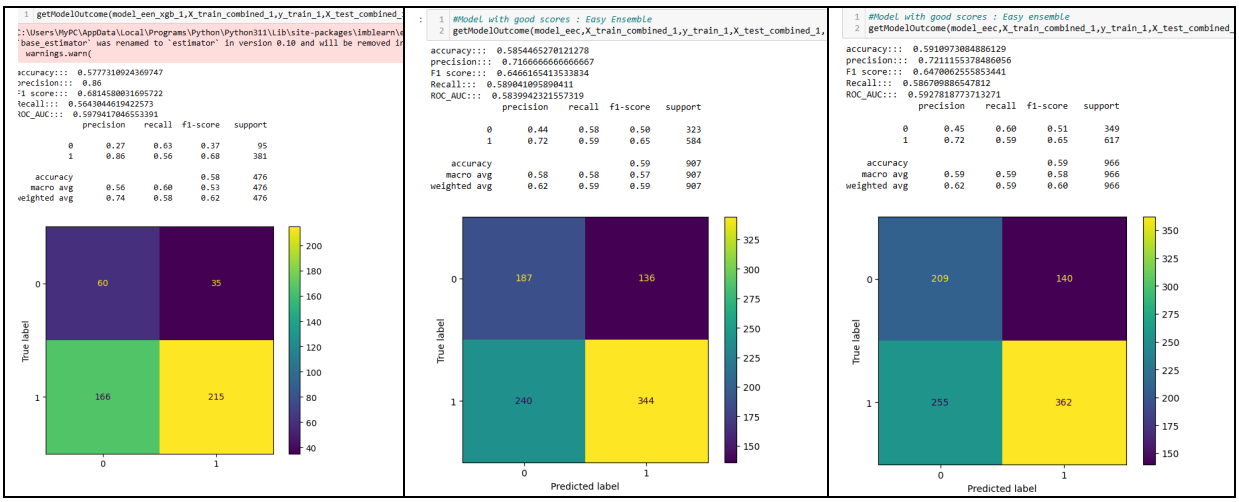
a. Model configuration for Cluster 1 present in 3_Classification_BioFormer-0_3_4-bp_dia.ipynb

```
model_een_xgb_1 = EasyEnsembleClassifier(n_estimators=10, random_state=43, base_estimator=xgboost.XGBClassifier
                                         (objective="binary:logistic",random_state=43, max_delta_step=1,))
```

b. Model configuration for Cluster 2 and Cluster 3 present in 3_Classification_BioFormer-1_alrg_pain.ipynb, 3_Classification_BioFormer-2_5_6.ipynb respectively

```
from imblearn.ensemble import EasyEnsembleClassifier
model_eec = EasyEnsembleClassifier(n_estimators=10)
cv_model_eec=crossValidation(model_eec, X_train_combined_1, y_train_1, kf)
cv_model_eec
```

c. Evaluation metrics



8.3 Model Outcomes using Clinical-AI-Apollo with Easy Ensemble with XGBoost estimator

a. Model configuration for Cluster 1 present in 3_Classification_ApolloAI-0_3_4-bp_dia.ipynb

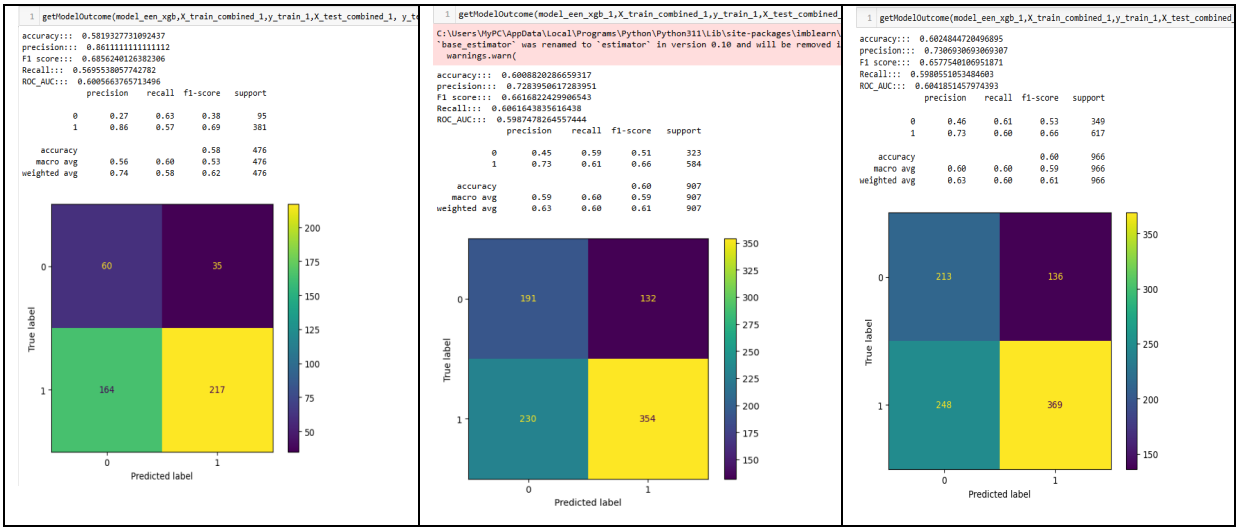
```
import xgboost
model_een_xgb = EasyEnsembleClassifier(n_estimators=10, random_state=43,
                                       estimator=xgboost.XGBClassifier(objective="binary:logistic",random_state=43))
```

b. Model configuration for Cluster 2 and Cluster 3 present in 3_Classification_ApolloAI-1_alrg_pain.ipynb, 3_Classification_ApolloAI--2_5_6.ipynb

```
#Model with good scores : Easy Ensemble XGBoost
model_een_xgb_1 = EasyEnsembleClassifier(n_estimators=10, random_state=43, base_estimator=xgboost.XGBClassifier
                                         (objective="binary:logistic",random_state=43, max_delta_step=1,))
```

```
#Model With Best score
model_een_xgb_1 = EasyEnsembleClassifier(n_estimators=10, random_state=43, estimator=xgboost.XGBClassifier
                                         (objective="binary:logistic",random_state=43, max_delta_step=1,))
```

c. Evaluation metrics



8.4 Other model results:

Random Forest, XGBoost model results are available in following files and can be accessed by referring file name to check classification based on vectorization.

9 Artifacts Details:

File name along with short description of file contents is mentioned below.

1. **Data Pre-Processing:**

   - 1_DataPreprocessing.ipynb -> This file has implementation of data processing activities
   - med7_demo.py -> This file contains function to perform entity extraction using Med7

2. **Clustering:**

   - 2_Clustering.ipynb -> This file includes steps performed to create clusters using K-means, GMM
   - 2_DataAnalysis_Clustering.ipynb -> In this file, Data is analysed cluster wise.

3. **Classification with BioBERT vectorization:**

   - 3_Classification_BioBERT-0_3_4_bp_dia.ipynb -> Classification and evaluation using three models for Cluster 1 using BioBERT

   - 3_Classification_BioBERT-1_alrg_pain.ipynb -> Classification and evaluation using three models for Cluster 2 using BioBERT

   - 3_Classification_BioBERT-2_5_6.ipynb -> Classification and evaluation using three models for Cluster 3 using BioBERT

4. **Classification with BioFormer vectorization:**

   - 3_Classification_BioFormer-0_3_4-bp_dia.ipynb -> Classification and evaluation using three models for Cluster 1 using BioFormer

   - 3_Classification_BioFormer-1_alrg_pain.ipynb -> Classification and evaluation using three models for Cluster 2 using BioFormer

   - 3_Classification_BioFormer-2_5_6.ipynb-> Classification and evaluation using three models for Cluster 3 using BioFormer

5. **Classification with Clinical-AI-Apollo vectorization:**

   - 3_Classification_ApolloAI-0_3_4-bp_dia.ipynb -> Classification and evaluation using three models for Cluster 1 using Clinical-AI_Apollo

   - 3_Classification_ApolloAI-1_alrg_pain.ipynb -> Classification and evaluation using three models for Cluster 2 using Clinical-AI_Apollo

   - 3_Classification_ApolloAI--2_5_6.ipynb -> Classification and evaluation using three models for Cluster 3 using Clinical-AI_Apollo

6. **Dataset:**

- **Medicine_Details.csv**: Original Dataset
- **Entity_pairings_Medicine_Name_2_0.csv:** Data after using Med7
- **data_updated_review_lable_1.csv**: Data after Data-Preprocessing
- **data_updated_review_lable_2.csv:** Data after clustering
- **data_updated_review_lable.csv**: Final Dataset used for classification

References

Singh N. (2024) '11000 Medicine Details', Available at: https://www.kaggle.com/datasets/singhnavjot2062001/11000-medicine-details