Medicine Cum Health Plan Recommendation System

Problem Statement: Developing a sophisticated Deep Learning model with the ability to effectively analyze user-supplied health data, encompassing their medical history, clinical interactions, chronic conditions, and prescribed medications. This model will generate a personalized score, which will play a pivotal role in our platform's ability to provide tailored recommendations for healthcare plans or insurance premiums that align with the individual user's needs and health profile.

Colab Notebook: Link to Notebook

Github Repo: https://github.com/hiiimanshusharma/HealthPlanRecommendationSystem

Demonstration Video: Loom Video Recording

Dataset: UCI ML Drug Review dataset (Link to dataset)

TTS Ratio: 80:20

Model: Naive Bayes(Multinomial Naive Bayes), Passive Aggressive Classifier with and without TFIDF vectorization.

Instruction for running code:

```
$ pip install -r requirements.txt

$ streamlit run st_app.py
```

These are the requirements needed to execute the code for the assignment:

1. **Pandas:** A data manipulation library for working with structured data in Python, providing data structures like DataFrames.

2. **itertools**: A module in Python's standard library that provides a collection of fast, memory-efficient tools for working with iterators.

3. **string**: A Python module that provides a collection of common string constants and functions for string manipulation.

4. **NumPy:** A fundamental package for numerical computing in Python, providing support for arrays and mathematical operations on them.

5. **Seaborn:** A data visualization library based on Matplotlib, designed for creating informative and attractive statistical graphics.

6. **Matplotlib**: A popular 2D plotting library for creating static, animated, or interactive visualizations in Python.

7. **BeautifulSoup4:** A library for parsing HTML and XML documents, making it easier to extract data from web pages.

8. **WordCloud**: A library for creating word clouds, which visually represent the frequency of words in a text.

9. **NLTK (Natural Language Toolkit):** A comprehensive library for natural language processing and text analysis tasks.

10. **Scikit-Learn (sklearn):** A machine learning library in Python that provides simple and efficient tools for data analysis and modeling.

11. **Joblib:** A library for lightweight pipelining in Python, particularly useful for efficiently saving and loading machine learning models and data.

These libraries are necessary for running the code, and should be installed in the environment or system before running the code.

```
!pip install pandas, itertools, string, numpy, seaborn, numpy, seaborn,
sklearn, matplotlib, BeautifulSoup4, WordCloud, nltk, joblib
```

Description of Dataset:

The dataset provides patient reviews on specific drugs along with related conditions. Furthermore, reviews are grouped into reports on the three aspects: benefits, side effects and overall comment. Additionally, ratings are available concerning overall satisfaction as well as a 5 step side effect rating and a 5 step effectiveness rating.

```
[9] df.head()
```

	uniqueID	drugName	condition	review	rating	date	usefulCount
0	206461	Valsartan	Left Ventricular Dysfunction	"It has no side effect, I take it in combinati...	9	20-May-12	27
1	95260	Guanfacine	ADHD	"My son is halfway through his fourth week of ...	8	27-Apr-10	192
2	92703	Lybrel	Birth Control	"I used to take another oral contraceptive, wh...	5	14-Dec-09	17
3	138000	Ortho Evra	Birth Control	"This is my first time using any form of birth...	8	3-Nov-15	10
4	35696	Buprenorphine / naloxone	Opiate Dependence	"Suboxone has completely turned my life around...	9	27-Nov-16	37

The pre-processing steps for the data in the assignment are as follows:

1. Importing the necessary libraries:

2. Reading the CSV file into a DataFrame:

3. Counting the values in the 'condition' column:

```
df.condition.value_counts()
```

4. Filtering the DataFrame based on specific conditions:

```
df_train = df[(df['condition']=='Birth Control') |
(df['condition']=='Depression') | (df['condition']=='High
Blood Pressure')|(df['condition']=='Diabetes, Type 2')]
```

. It only includes rows where the 'condition' column matches one of the specified conditions: 'Birth Control', 'Depression', 'High Blood Pressure', or 'Diabetes, Type 2'.

5. Resetting the index of the filtered DataFrame:

```
df_train = df_train.reset_index(drop=True)
```

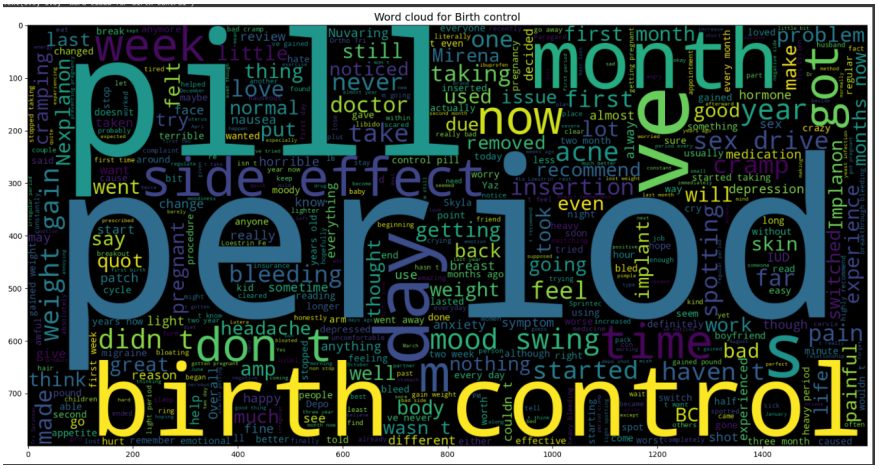
These preprocessing steps involve reading a CSV file, exploring the counts of unique values in a specific column, filtering the DataFrame based on certain conditions, and resetting the index of the filtered DataFrame to ensure it starts from 0 and is continuous.

Exploratory Data Analysis(EDA) :

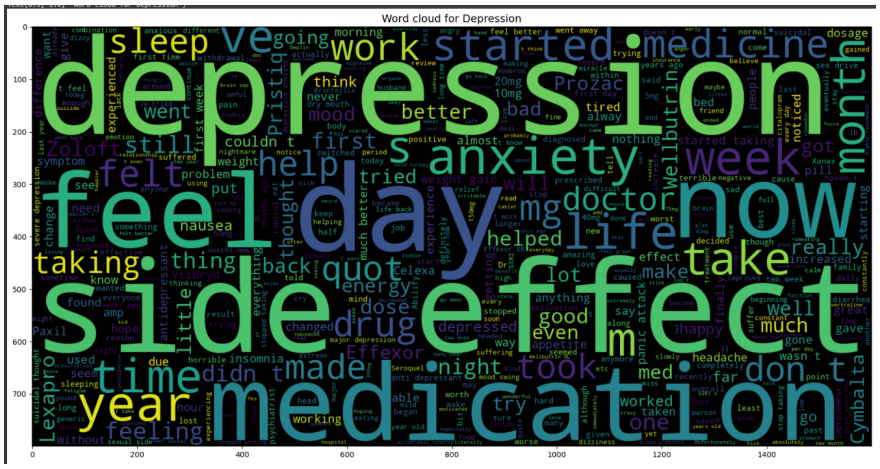
1. Segregating DataFrames:
   - Four new DataFrames (`X_birth`, `X_dep`, `X_bp`, `X_diab`) are created by filtering rows

in the original DataFrame `X` based on specific conditions ('Birth Control', 'Depression', 'High Blood Pressure', 'Diabetes, Type 2').

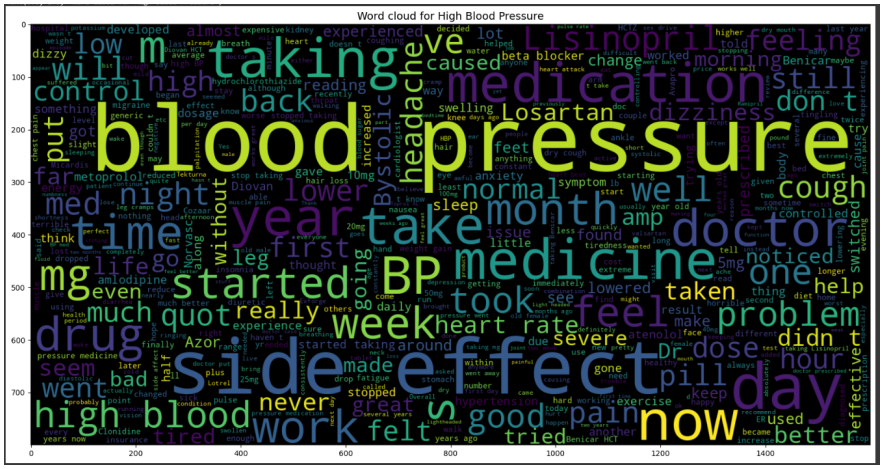
2. Creating Word Clouds:
   - For each medical condition, a word cloud is generated using the `WordCloud` library.
   - For Birth control



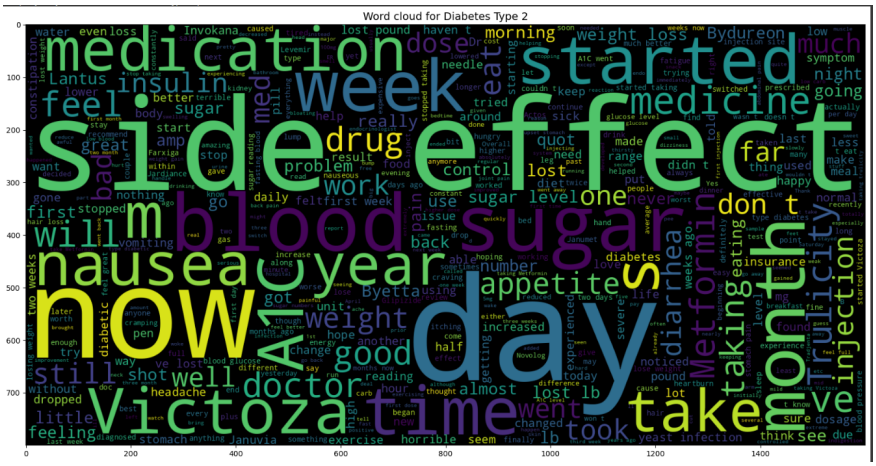
   - For Depression



- For Blood Pressure



- For Diabetes type2



Data Pre-processing:

Preprocessed raw text review, typically as part of text data preparation for natural language processing (NLP) or machine learning tasks. It applies a series of text cleaning and transformation steps to make the review text more suitable for analysis or modeling. Here's the breakdown of its purposes:

1. Delete HTML: Remove HTML tags from the raw text, as reviews might contain HTML formatting that is irrelevant for text analysis.

2. Make a space: Replace any characters that are not letters with spaces. This effectively splits the text into words, removing special characters and punctuation.

3. Lowercase letters: Convert all letters to lowercase. This ensures that the text is case-insensitive and helps in standardizing the text.

4. Tokenization: Split the text into individual words. This step essentially converts the processed text into a list of words.

5. Stopwords: Remove common stopwords from the list of words. Stopwords are often excluded because they don't carry much meaningful information and can be noise in text analysis.

6. Lemmatization: Reduce words to their base or root form through lemmatization. This step helps in reducing word variation and making the text more consistent (e.g., "running" becomes "run").

7. Space-join words: Join the lemmatized words back together into a single string, separated by spaces. This creates a clean, preprocessed version of the review text.

```
def review_to_words(raw_review):

    # 1. Delete HTML

    review_text = BeautifulSoup(raw_review, 'html.parser').get_text()

    # 2. Make a space

    letters_only = re.sub('[^a-zA-Z]', ' ', review_text)

    # 3. lower letters

    words = letters_only.lower().split()

    # 5. Stopwords

    meaningful_words = [w for w in words if not w in stop]

    # 6. lemmatization

    lemmitize_words = [lemmatizer.lemmatize(w) for w in meaningful_words]

    # 7. space join words

    return( ' '.join(lemmitize_words))
```

Feature Extraction:

Text classification using a Bag of Words (BoW) approach with scikit-learn. Let me break down the code and explain each part briefly:

1. `X_feat` and `y`: These variables are created to store the feature (in this case, text data) and target labels (in this case, the 'condition' labels)

2. `X_train`, `X_test`, `y_train`, `y_test`: These variables are used to split the dataset into training and testing sets. The `train_test_split` function from scikit-learn is employed for this purpose. The dataset is split in such a way that 80% of the data is used for training (`X_train` and `y_train`), and 20% is reserved for testing (`X_test` and `y_test`). The `stratify=y` argument ensures that the target labels are distributed evenly between the training and testing sets to maintain the same class distribution as the original dataset. `random_state` is set to 0 for reproducibility.

3. `CountVectorizer`: This is a feature extraction technique used to convert text data into numerical features. It transforms the text into a matrix of word frequencies. The `stop_words='english'` argument specifies that common English stop words (e.g., "the," "and," "is") should be ignored during the vectorization process.

4. `count_train` and `count_test`: These variables store the transformed BoW representations of the training and testing text data, respectively. `count_train` is obtained by fitting the `CountVectorizer` on `X_train` and then transforming `X_train`. `count_test` is obtained by transforming `X_test` using the same fitted `CountVectorizer`.

```
X_feat=X['review_clean']
y=X['condition'] X_train, X_test, y_train, y_test =
train_test_split(X_feat,        y,stratify=y,test_size=0.2,
random_state=0)
# bag of words
count_vectorizer = CountVectorizer(stop_words='english')
count_train = count_vectorizer.fit_transform(X_train)
count_test = count_vectorizer.transform(X_test)
```

Machine Learning Model:

Multinomial Naive Bias:

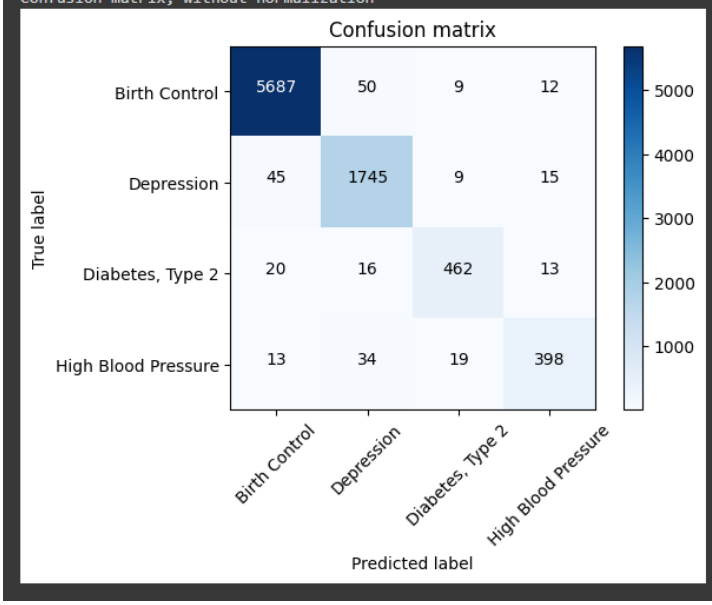
Multinomial Naive Bayes algorithm is a probabilistic learning method that is mostly used

in Natural Language Processing (NLP). The algorithm is based on the Bayes theorem and predicts the tag of a text such as a piece of email or newspaper article. It calculates the probability of each tag for a given sample and then gives the tag with the highest probability as output.

```
mnb = MultinomialNB()
mnb.fit(count_train, y_train)
pred = mnb.predict(count_test)
score = metrics.accuracy_score(y_test, pred)
print("accuracy:   %0.3f" % score)

cm = metrics.confusion_matrix(y_test, pred, labels=['Birth Control', 'Depression','Diabetes, Type 2','High Blood Pressure'])
plot_confusion_matrix(cm, classes=['Birth Control', 'Depression','Diabetes, Type 2','High Blood Pressure'])
```

```
accuracy:   0.970
Confusion matrix, without normalization
```



accuracy : 97%

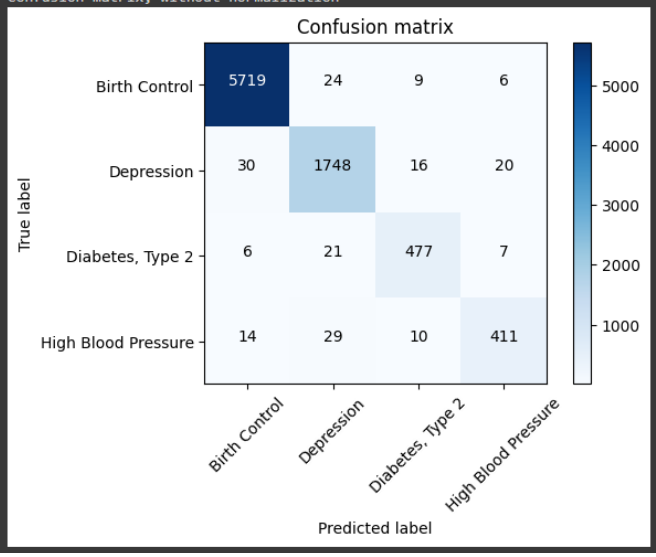
Passive Aggressive classification:

Passive-Aggressive algorithms are generally used for large-scale learning. It is one of the online-learning algorithms. In online machine learning algorithms, the input data comes in sequential order and the machine learning model is updated sequentially, as opposed to conventional batch learning, where the entire training dataset is used at once.

```
from sklearn.linear_model import PassiveAggressiveClassifier,LogisticRegression

passive = PassiveAggressiveClassifier()
passive.fit(count_train, y_train)
pred = passive.predict(count_test)
score = metrics.accuracy_score(y_test, pred)
print("accuracy:   %0.3f" % score)
cm = metrics.confusion_matrix(y_test, pred, labels=['Birth Control', 'Depression','Diabetes, Type 2','High Blood Pressure'])
plot_confusion_matrix(cm, classes=['Birth Control', 'Depression','Diabetes, Type 2','High Blood Pressure'])
```

```
accuracy:   0.978
Confusion matrix, without normalization
```



accuracy : 97.8%

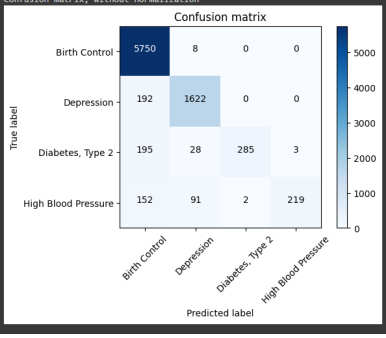
TF(Term Frequency)-IDF(Inverse Document Frequency):

TF-IDF stands for Term Frequency Inverse Document Frequency of records. It can be defined as the calculation of how relevant a word in a series or corpus is to a text. The meaning increases proportionally to the number of times in the text a word appears but is compensated by the word frequency in the corpus (data-set).

Naive Bayes With TF IDF :

```
mnb_tf = MultinomialNB()
mnb_tf.fit(tfidf_train_2, y_train)
pred = mnb_tf.predict(tfidf_test_2)
score = metrics.accuracy_score(y_test, pred)
print("accuracy:   %0.3f" % score)
cm = metrics.confusion_matrix(y_test, pred, labels=['Birth Control', 'Depression','Diabetes, Type 2','High Blood Pressure'])
plot_confusion_matrix(cm, classes=['Birth Control', 'Depression','Diabetes, Type 2','High Blood Pressure'])
```

```
accuracy:   0.921
Confusion matrix, without normalization
```



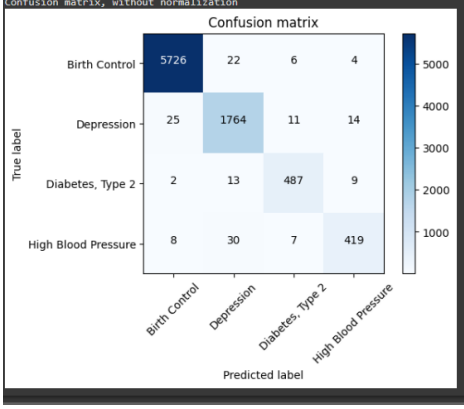
accuracy : 92.1%

Passive Aggressive classification With TF IDF :

```
tfidf_vectorizer = TfidfVectorizer(stop_words='english', max_df=0.8)
tfidf_train = tfidf_vectorizer.fit_transform(X_train)
tfidf_test = tfidf_vectorizer.transform(X_test)

pass_tf = PassiveAggressiveClassifier()
pass_tf.fit(tfidf_train, y_train)
pred = pass_tf.predict(tfidf_test)
score = metrics.accuracy_score(y_test, pred)
print("accuracy:   %0.3f" % score)
cm = metrics.confusion_matrix(y_test, pred, labels=['Birth Control', 'Depression','Diabetes, Type 2','High Blood Pressure'])
plot_confusion_matrix(cm, classes=['Birth Control', 'Depression','Diabetes, Type 2','High Blood Pressure'])
```

```
accuracy:   0.982
Confusion matrix, without normalization
```



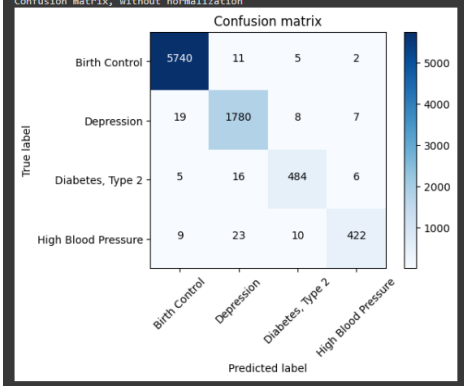
accuracy : 98.2%

Passive Aggressive classification With TF IDF Bigrams :

```
tfidf_vectorizer2 = TfidfVectorizer(stop_words='english', max_df=0.8, ngram_range=(1,2))
tfidf_train_2 = tfidf_vectorizer2.fit_transform(X_train)
tfidf_test_2 = tfidf_vectorizer2.transform(X_test)

pass_tf = PassiveAggressiveClassifier()
pass_tf.fit(tfidf_train_2, y_train)
pred = pass_tf.predict(tfidf_test_2)
score = metrics.accuracy_score(y_test, pred)
print("accuracy:   %0.3f" % score)
cm = metrics.confusion_matrix(y_test, pred, labels=['Birth Control', 'Depression','Diabetes, Type 2','High Blood Pressure'])
plot_confusion_matrix(cm, classes=['Birth Control', 'Depression','Diabetes, Type 2','High Blood Pressure'])
```

```
accuracy:   0.986
Confusion matrix, without normalization
```



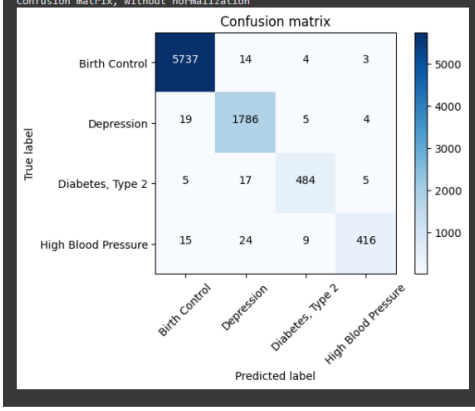
accuracy : 98.6%

Passive Aggressive classification With TF IDF Trigrams :

```
tfidf_vectorizer3 = TfidfVectorizer(stop_words='english', max_df=0.8, ngram_range=(1,3))
tfidf_train_3 = tfidf_vectorizer3.fit_transform(X_train)
tfidf_test_3 = tfidf_vectorizer3.transform(X_test)

pass_tf = PassiveAggressiveClassifier()
pass_tf.fit(tfidf_train_3, y_train)
pred = pass_tf.predict(tfidf_test_3)
score = metrics.accuracy_score(y_test, pred)
print("accuracy:   %0.3f" % score)
cm = metrics.confusion_matrix(y_test, pred, labels=['Birth Control', 'Depression','Diabetes, Type 2','High Blood Pressure'])
plot_confusion_matrix(cm, classes=['Birth Control', 'Depression','Diabetes, Type 2','High Blood Pressure'])
```

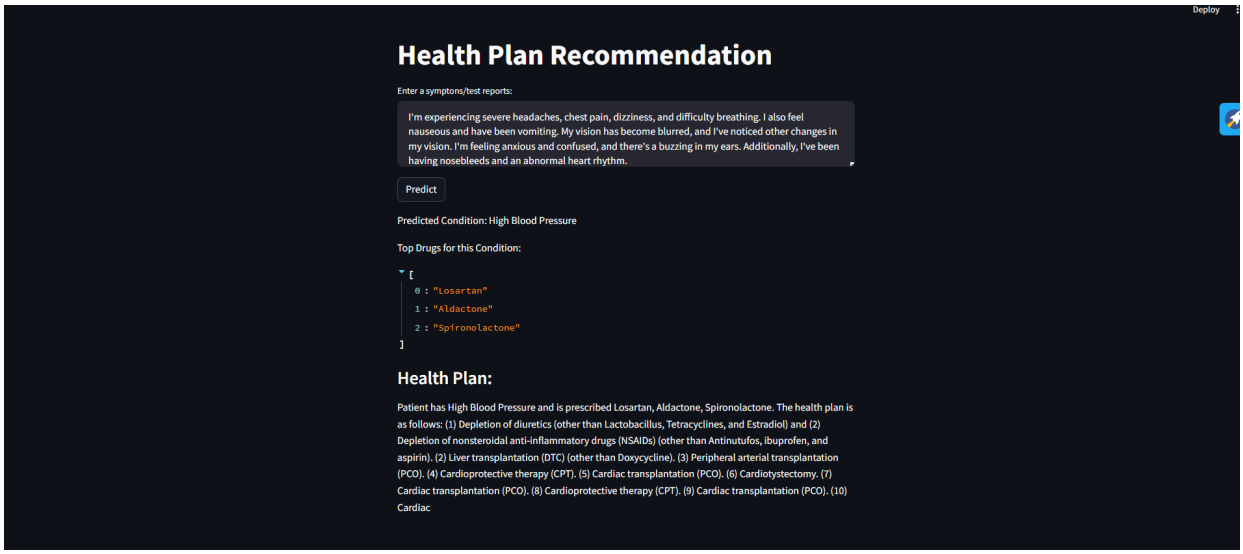
```
accuracy:   0.985
Confusion matrix, without normalization
```



accuracy : 98.5%

Deployment :

I've deployed a **Streamlit web application** for health plan recommendations, and it's been quite an exciting project. With this app, users can simply input their symptoms or test reports, and with the click of a button, it predicts their medical condition based on the provided text. The application then suggests the top drugs for that condition,based on trained model **tfidfvectorizer.pkl, passmodel.pkl** considering factors like high ratings and usefulness counts from a dataset, trained in **Medicine_Recommendation_System.ipynb** file using **Passive Agression Classifier** model with **TFIDF vectorization**. What's really fascinating is that it doesn't stop there – it also generates a personalized health plan using a GPT-2 language model. This tool has the potential to offer valuable health insights and recommendations to users, making it a useful addition to the healthcare tech landscape.



Insights gained :

1. Data integration and preprocessing are critical for handling user-supplied health data.
2. Text analysis and NLP are essential for medical condition prediction.
3. Building recommendation systems requires thoughtful algorithms and data.
4. Deploying models in a user-friendly web app is essential for accessibility.
5. Personalization with advanced AI models like GPT-2 enhances user engagement.
6. Deepened domain knowledge in healthcare and its intricacies.
7. A user-centric approach is vital for delivering valuable recommendations.
8. Collaboration across disciplines is necessary for success.
9. Ethical considerations and data privacy regulations are paramount.
10. The importance of continuous learning in the ever-evolving healthcare tech field.