Natural Language Processing

Assignment-02

Web Scraping with Scrapy - Collecting Urdu Poetry

Introduction of Web-Scrapping :

Following is the provided URL : https://www.rekhta.org

Next it was also mentioned that we were supposed to use a new library , Scrapy to scrap data from the provided URL .

Following is the Link of Scrapy : https://docs.scrapy.org/en/latest/

I have extensively used this documentation since Scrapy was a new library for me but I had some prior knowledge of Web-Scrapping using Selenium .

Following is the Link of Selenium : https://www.selenium.dev/documentation/

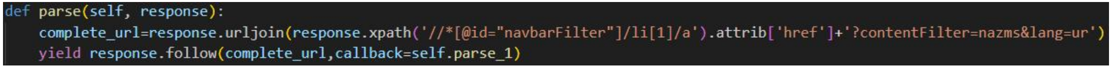
It is assumed that Web-Scrapping is quite a challenging task ask it is necessary to maintain privacy of the scrapped websites since information can be used for good and bad things.
Secondly , the provided website was not in the lists of websites which are not allowed to scrape . Famous websites like Amazon , CostCo. Etc are not allowed to scrape since they have deployed robotic detection which basically identifies the humanoid behaviour when their respective website is accessed and they trigger anti-scrapping measures if non-human behaviour is detected.
Thirdly , not a lot of data was required to scrape so it was extremely safe to scrape data from the provided URL.
We were asked to scrape Urdu Poems of atleaset 25 poets from the website .

Approach For Scrapping Rekhta :

The approach was very simple . The main logic was to navigate between web-pages and to extract the precise information of the poets. What I did was that after navigating to the main page of the website , I navigate to the next page with the following two filters :

Filter - 1 : The page should only contain Nazms(Poems = Nazms)
Filter - 2 : The page should only contain Urdu Poets

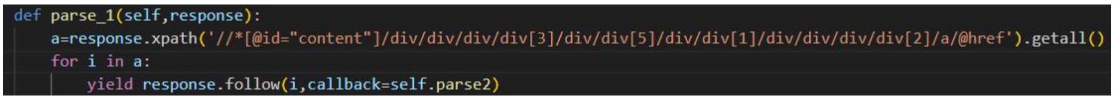
By applying these filters , I navigated to the most important page from where I started my scrapping .
Following is the command used in Scrapy :



```
def parse(self, response):
    complete_url=response.urljoin(response.xpath('//*[@id="navbarFilter"]/li[1]/a').attrib['href']+'?contentFilter=nazms&lang=ur')
    yield response.follow(complete_url,callback=self.parse_1)
```

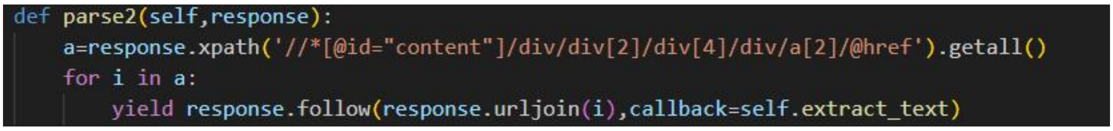
' urljoin' is used to join filters with the base URL

The next Step was to click on each poet avaliable and extract the URL's of all the Poems written by that Poet . Once this is done , do not move back to the previous page but click on each of the extracted link and navigate 1 step ahead to the page where the Poem is written.
Now , from this page , extract the beginning and closing HTML Tags of the Poem ,
Parse these HTML Tags and extract the Poem and save it in a dataframe which was further converted into a csv file .
Following is the command used in Scrapy to extract all links of
Poets :



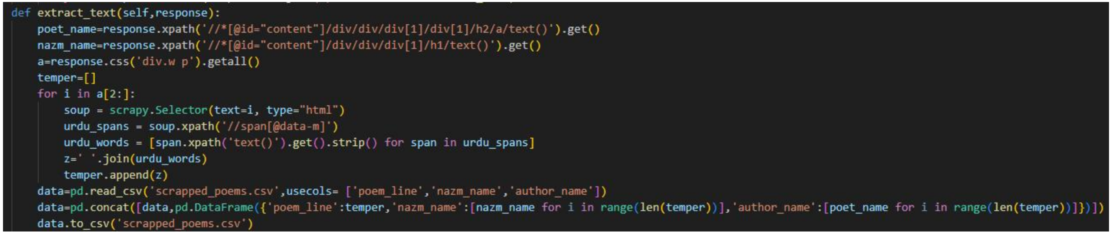
```
def parse_1(self,response):
    a=response.xpath('//*[@id="content"]/div/div/div[3]/div/div[5]/div/div[1]/div/div/div/div[2]/a/@href').getall()
    for i in a:
        yield response.follow(i,callback=self.parse2)
```

Following is the command used in Scrapy to extract all links of Poems of each Poet :



```
def parse2(self,response):
    a=response.xpath('//*[@id="content"]/div/div[2]/div[4]/div/a[2]/@href').getall()
    for i in a:
        yield response.follow(response.urljoin(i),callback=self.extract_text)
```

Following is the command used in Scrapy to extract HTML Tags of Poems :



```
def extract_text(self,response):
    poet_name=response.xpath('//*[@id="content"]/div/div/div[1]/div[1]/h2/a/text()').get()
    nazm_name=response.xpath('//*[@id="content"]/div/div/div[1]/h1/text()').get()
    a=response.css('div.w p').getall()
    temper=[]
    for i in a[2:]:
        soup = scrapy.Selector(text=i, type="html")
        urdu_spans = soup.xpath('//span[@data-m]')
        urdu_words = [span.xpath('text()').get().strip() for span in urdu_spans]
        z=' '.join(urdu_words)
        temper.append(z)
    data=pd.read_csv('scrapped_poems.csv',usecols= ['poem_line','nazm_name','author_name'])
    data=pd.concat([data,pd.DataFrame({'poem_line':temper,'nazm_name':[nazm_name for i in range(len(temper))],'author_name':[poet_name for i in range(len(temper))]})])
    data.to_csv('scrapped_poems.csv')
```

Now I also extract the Poet and Poem name in-order have proper fashion while saving it into the csv file . After each iteration of Poet , I re-read the saved contents of the csv file and extend the existing data with the new one by appending the new data at the bottom of the dataframe and then again converting it back to a csv file thus updating the previous csv file .

The code for scrapping is actually quite small , 31 lines to be precise . I can further shortent down the code but then it would become extremely difficult for anyone else to read , understand , implement and change the code according to their personal liking .

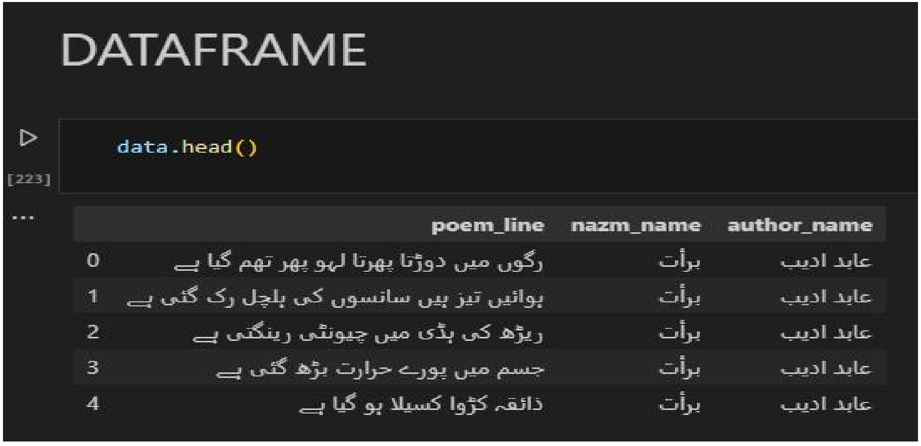
Challenges Faced in Scrapping :

The main challenges faced were selection the correct **XPATH** and the correct **CSS SELECTOR** from the website which were used to not only navigate between pages but also to extract the poems .
The main reason was that Scrapy was quite smilar to Selenium with just differences between the syntax . Moreover , Scrapy is extremely efficient as the runtime of scrapy far smaller than Selenium. So , I would now prefer Scrapy over Selenium wherever a task for Web-Scrapping is given .
Other than this I did not face any challenge since I had prior knowledge about Selenium and how Web-Scrapping works .

Poetry Generation in Urdu

Dataset:

Following is how the csv file looked after scrapping all the data :



```
data.head()
```

	poem_line	nazm_name	author_name
0	رگوں میں دوڑتا پھرتا لہو پھر تھم گیا ہے	برأت	عابد ادیب
1	ہوائیں تیز ہیں سانسوں کی ہلچل رک گئی ہے	برأت	عابد ادیب
2	ریڑھ کی ہڈی میں چیونٹی رینگتی ہے	برأت	عابد ادیب
3	جسم میں پورے حرارت بڑھ گئی ہے	برأت	عابد ادیب
4	ذائقہ کڑوا کسیلا ہو گیا ہے	برأت	عابد ادیب

Libraries Used :

I. Spacy
II. Numpy
III. Pandas
IV. Random

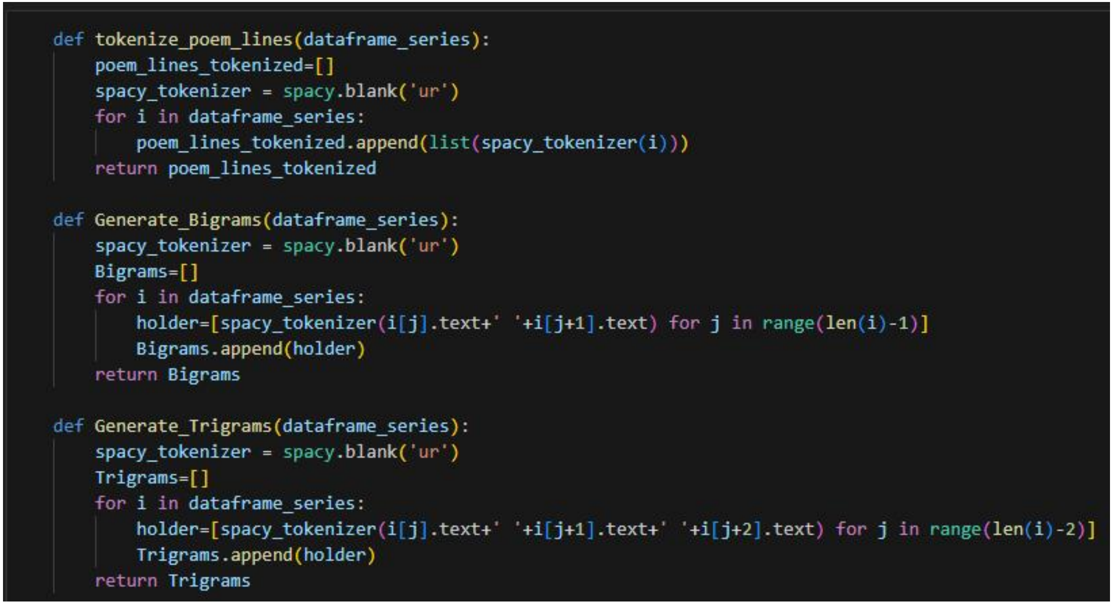
Approach For N-GRAM Model :

The apprroach was extremely simple . Tokenize the poem_lines. Make 2 functions which are the following :

I. Generate Bigrams
II. Generate Trigrams
III. Unigrams were actually tokenized poem_lines with removed stopwords

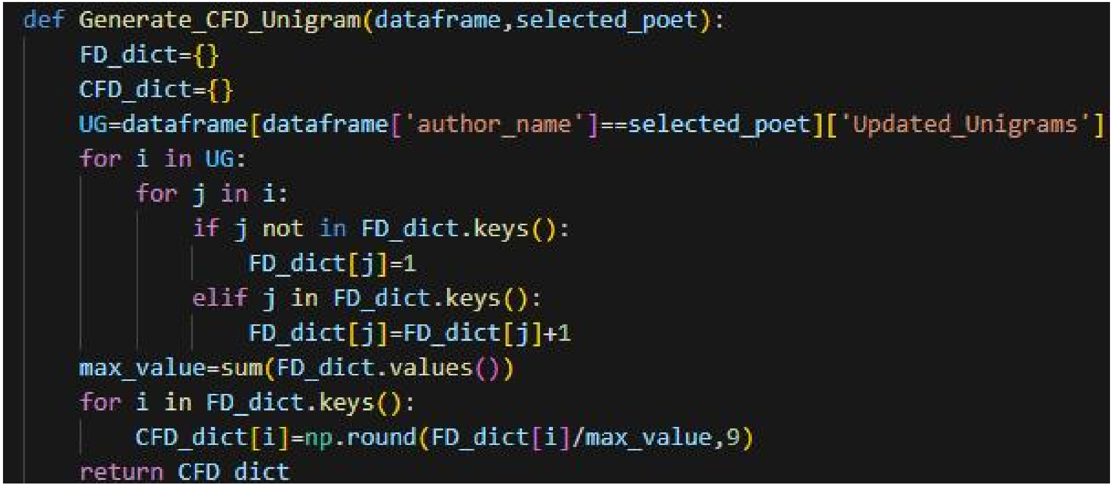
The approach I used to filter-out stopwords from the generated N-grams was that after generating N-grams , I remove all those n-grams from the corpus that begin and end with stopwords . This strategy was used in-order to keep some information but to remove the noise created by the stopwords bigrams .

Following is the code for N-grams :



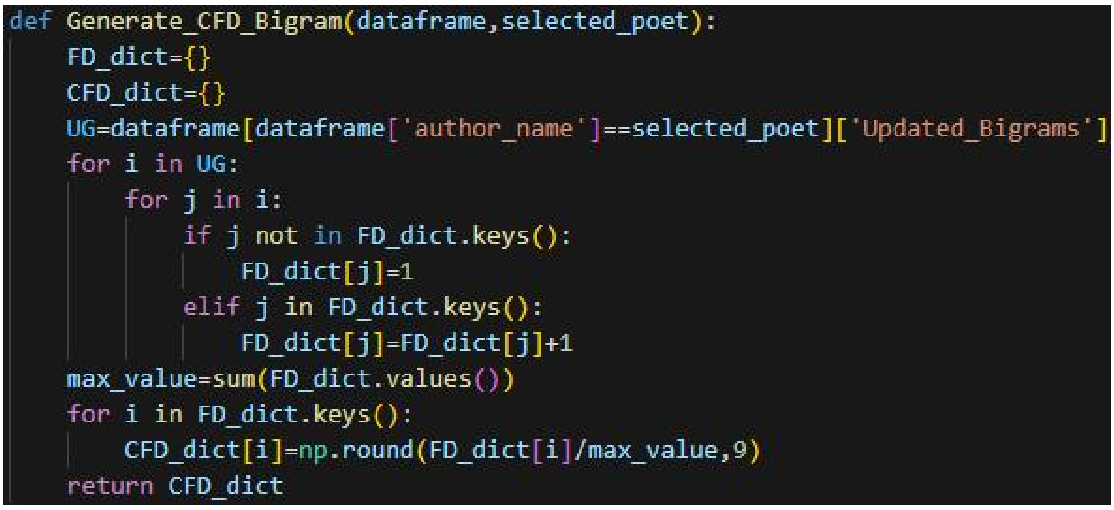
```
def tokenize_poem_lines(dataframe_series):
    poem_lines_tokenized=[]
    spacy_tokenizer = spacy.blank('ur')
    for i in dataframe_series:
        poem_lines_tokenized.append(list(spacy_tokenizer(i)))
    return poem_lines_tokenized

def Generate_Bigrams(dataframe_series):
    spacy_tokenizer = spacy.blank('ur')
    Bigrams=[]
    for i in dataframe_series:
        holder=[spacy_tokenizer(i[j].text+' '+i[j+1].text) for j in range(len(i)-1)]
        Bigrams.append(holder)
    return Bigrams

def Generate_Trigrams(dataframe_series):
    spacy_tokenizer = spacy.blank('ur')
    Trigrams=[]
    for i in dataframe_series:
        holder=[spacy_tokenizer(i[j].text+' '+i[j+1].text+' '+i[j+2].text) for j in range(len(i)-2)]
        Trigrams.append(holder)
    return Trigrams
```

UnigramCummlative Frequency Distribution Code :



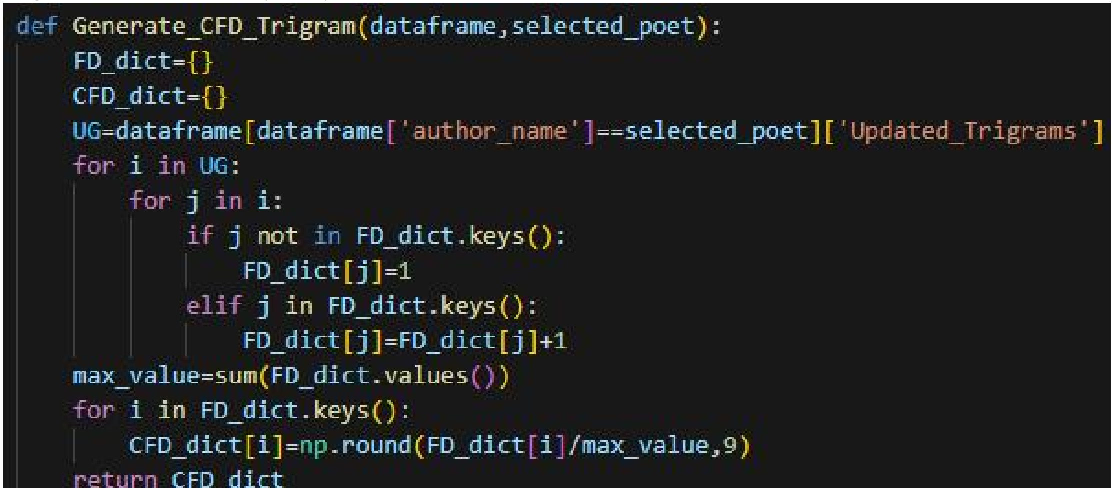
```
def Generate_CFD_Unigram(dataframe,selected_poet):
    FD_dict={}
    CFD_dict={}
    UG=dataframe[dataframe['author_name']==selected_poet]['Updated_Unigrams']
    for i in UG:
        for j in i:
            if j not in FD_dict.keys():
                FD_dict[j]=1
            elif j in FD_dict.keys():
                FD_dict[j]=FD_dict[j]+1
    max_value=sum(FD_dict.values())
    for i in FD_dict.keys():
        CFD_dict[i]=np.round(FD_dict[i]/max_value,9)
    return CFD_dict
```

BigramCummlative Frequency Distribution Code :



```
def Generate_CFD_Bigram(dataframe,selected_poet):
    FD_dict={}
    CFD_dict={}
    UG=dataframe[dataframe['author_name']==selected_poet]['Updated_Bigrams']
    for i in UG:
        for j in i:
            if j not in FD_dict.keys():
                FD_dict[j]=1
            elif j in FD_dict.keys():
                FD_dict[j]=FD_dict[j]+1
    max_value=sum(FD_dict.values())
    for i in FD_dict.keys():
        CFD_dict[i]=np.round(FD_dict[i]/max_value,9)
    return CFD_dict
```

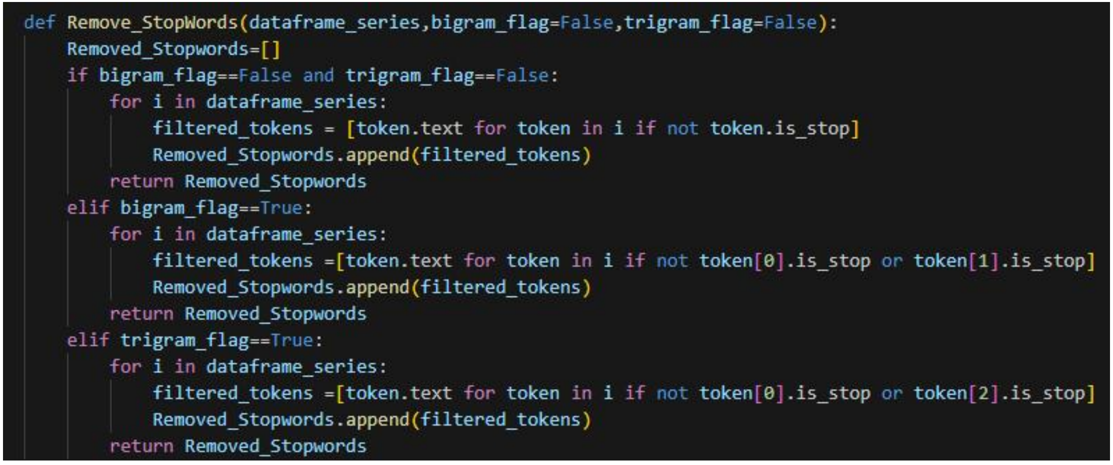
TrigramCummlative Frequency Distribution Code :



```
def Generate_CFD_Trigram(dataframe,selected_poet):
    FD_dict={}
    CFD_dict={}
    UG=dataframe[dataframe['author_name']==selected_poet]['Updated_Trigrams']
    for i in UG:
        for j in i:
            if j not in FD_dict.keys():
                FD_dict[j]=1
            elif j in FD_dict.keys():
                FD_dict[j]=FD_dict[j]+1
    max_value=sum(FD_dict.values())
    for i in FD_dict.keys():
        CFD_dict[i]=np.round(FD_dict[i]/max_value,9)
    return CFD_dict
```

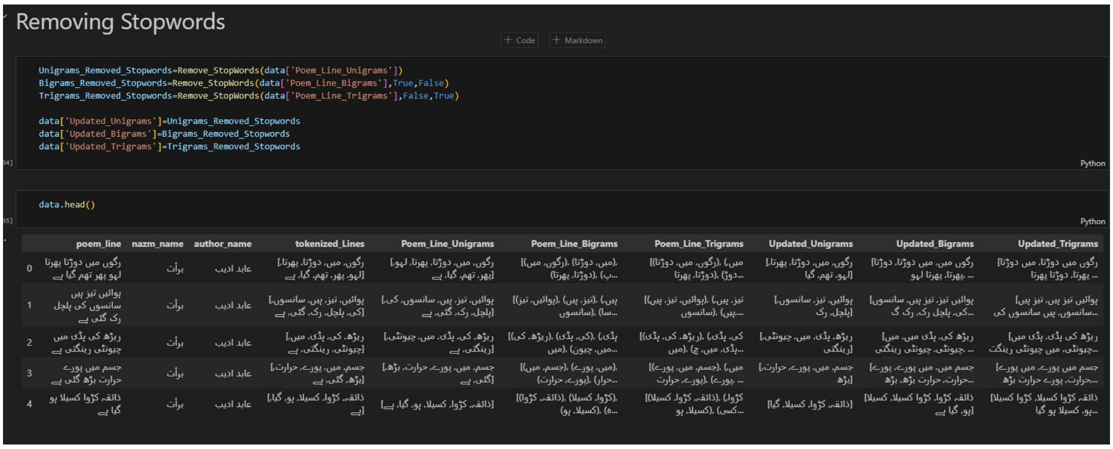
CFD was required to predict the next word based on the previous word

Stopwords Removal Function :



```
def Remove_StopWords(dataframe_series,bigram_flag=False,trigram_flag=False):
    Removed_Stopwords=[]
    if bigram_flag==False and trigram_flag==False:
        for i in dataframe_series:
            filtered_tokens = [token.text for token in i if not token.is_stop]
            Removed_Stopwords.append(filtered_tokens)
        return Removed_Stopwords
    elif bigram_flag==True:
        for i in dataframe_series:
            filtered_tokens =[token.text for token in i if not token[0].is_stop or token[1].is_stop]
            Removed_Stopwords.append(filtered_tokens)
        return Removed_Stopwords
    elif trigram_flag==True:
        for i in dataframe_series:
            filtered_tokens =[token.text for token in i if not token[0].is_stop or token[2].is_stop]
            Removed_Stopwords.append(filtered_tokens)
        return Removed_Stopwords
```

Resultant Dataframe after performing the previous functions :



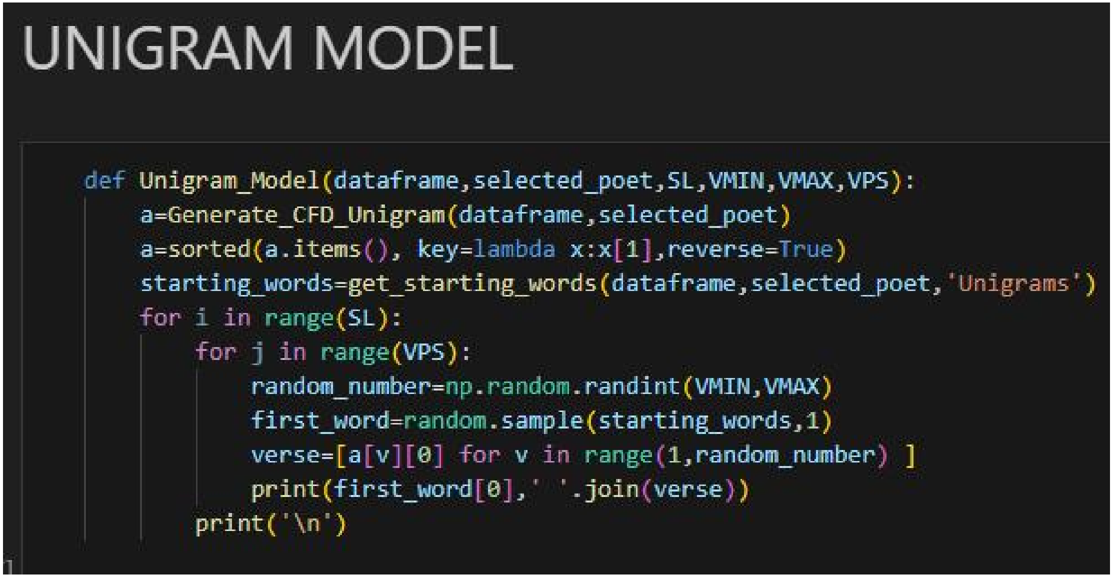
Removing Stopwords

```python
Unigrams_Removed_Stopwords=Remove_StopWords(data['Poem_Line_Unigrams'])
Bigrams_Removed_Stopwords=Remove_StopWords(data['Poem_Line_Bigrams'],True,False)
Trigrams_Removed_Stopwords=Remove_StopWords(data['Poem_Line_Trigrams'],False,True)

data['Updated_Unigrams']=Unigrams_Removed_Stopwords
data['Updated_Bigrams']=Bigrams_Removed_Stopwords
data['Updated_Trigrams']=Trigrams_Removed_Stopwords
```

```python
data.head()
```

	poem_line	nazm_name	author_name	tokenized_Lines	Poem_Line_Unigrams	Poem_Line_Bigrams	Poem_Line_Trigrams	Updated_Unigrams	Updated_Bigrams	Updated_Trigrams
0	رگوں میں دوڑتا پھرتا لہو پھر تھم گیا ہے	برأت	عابد ادیب	[رگوں، میں، دوڑتا، پھرتا، لہو، پھر، تھم، گیا، ہے]	[رگوں، میں، دوڑتا، پھرتا، لہو، پھر، تھم، گیا، ہے]	[(رگوں، میں)، (میں، دوڑتا)، (دوڑتا، پھرتا)، (پ...	[(رگوں، میں، دوڑتا)، (میں، دوڑتا، پھرتا)، (دوڑ...	[رگوں، میں، دوڑتا، پھرتا، لہو، تھم، گیا]	[رگوں میں، میں دوڑتا، دوڑتا پھرتا، پھرتا لہو، ...	[رگوں میں دوڑتا، میں دوڑتا پھرتا، دوڑتا پھرتا ...
1	ہوائیں تیز ہیں سانسوں کی ہلچل رک گئی ہے	برأت	عابد ادیب	[ہوائیں، تیز، ہیں، سانسوں، کی، ہلچل، رک، گئی، ہے]	[ہوائیں، تیز، ہیں، سانسوں، کی، ہلچل، رک، گئی، ہے]	[(ہوائیں، تیز)، (تیز، ہیں)، (ہیں، سانسوں)، (سا...	[(ہوائیں، تیز، ہیں)، (تیز، ہیں، سانسوں)، (ہیں،...	[ہوائیں، تیز، سانسوں، ہلچل، رک]	[ہوائیں تیز، تیز ہیں، سانسوں کی، ہلچل رک، رک گ...	[ہوائیں تیز ہیں، تیز ہیں سانسوں، ہیں سانسوں کی...
2	ریڑھ کی ہڈی میں چیونٹی رینگتی ہے	برأت	عابد ادیب	[ریڑھ، کی، ہڈی، میں، چیونٹی، رینگتی، ہے]	[ریڑھ، کی، ہڈی، میں، چیونٹی، رینگتی، ہے]	[(ریڑھ، کی)، (کی، ہڈی)، (ہڈی، میں)، (میں، چیون...	[(ریڑھ، کی، ہڈی)، (کی، ہڈی، میں)، (ہڈی، میں، چ...	[ریڑھ، ہڈی، میں، چیونٹی، رینگتی]	[ریڑھ کی، ہڈی میں، میں چیونٹی، چیونٹی رینگتی، ...	[ریڑھ کی ہڈی، ہڈی میں چیونٹی، میں چیونٹی رینگت...
3	جسم میں پورے حرارت بڑھ گئی ہے	برأت	عابد ادیب	[جسم، میں، پورے، حرارت، بڑھ، گئی، ہے]	[جسم، میں، پورے، حرارت، بڑھ، گئی، ہے]	[(جسم، میں)، (میں، پورے)، (پورے، حرارت)، (حرار...	[(جسم، میں، پورے)، (میں، پورے، حرارت)، (پورے، ...	[جسم، میں، پورے، حرارت، بڑھ]	[جسم میں، میں پورے، پورے حرارت، حرارت بڑھ، بڑھ...	[جسم میں پورے، میں پورے حرارت، پورے حرارت بڑھ،...
4	ذائقہ کڑوا کسیلا ہو گیا ہے	برأت	عابد ادیب	[ذائقہ، کڑوا، کسیلا، ہو، گیا، ہے]	[ذائقہ، کڑوا، کسیلا، ہو، گیا، ہے]	[(ذائقہ، کڑوا)، (کڑوا، کسیلا)، (کسیلا، ہو)، (ہ...	[(ذائقہ، کڑوا، کسیلا)، (کڑوا، کسیلا، ہو)، (کسی...	[ذائقہ، کڑوا، کسیلا، گیا]	[ذائقہ کڑوا، کڑوا کسیلا، کسیلا ہو، ہو گیا، گیا ہے]	[ذائقہ کڑوا کسیلا، کڑوا کسیلا ہو، کسیلا ہو گیا...

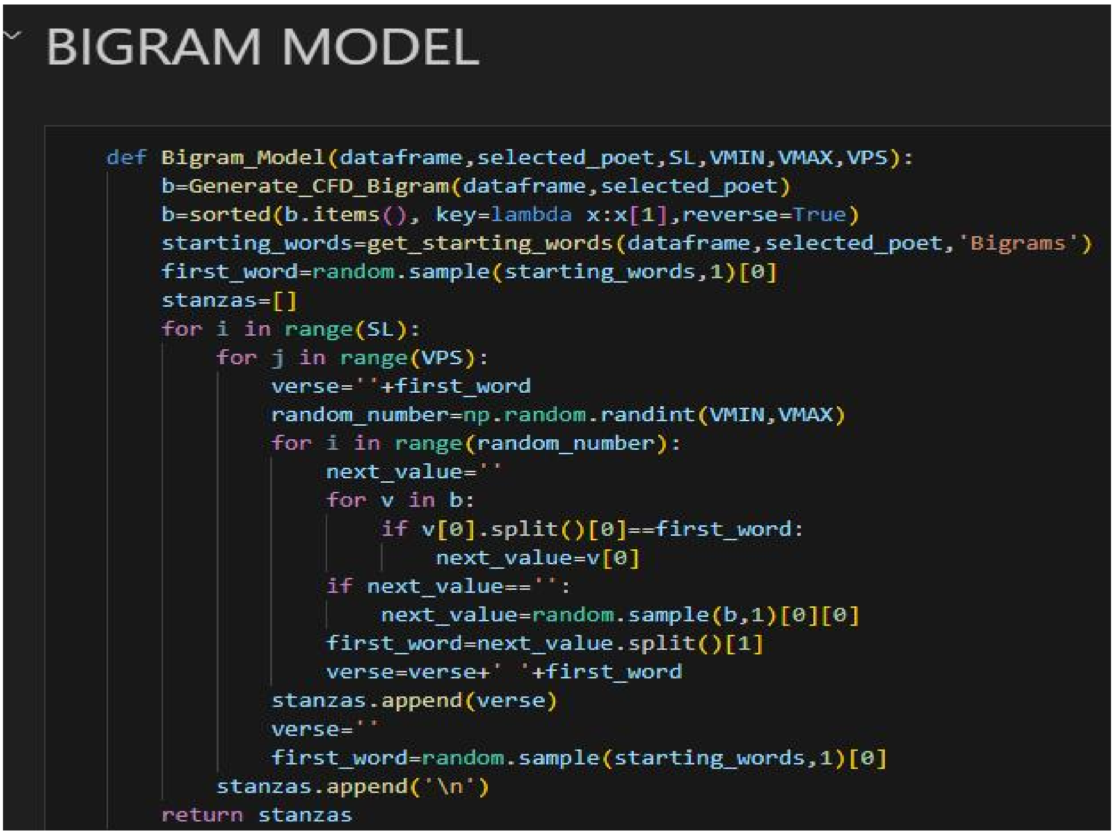
Unigram Model :



UNIGRAM MODEL

```python
def Unigram_Model(dataframe,selected_poet,SL,VMIN,VMAX,VPS):
    a=Generate_CFD_Unigram(dataframe,selected_poet)
    a=sorted(a.items(), key=lambda x:x[1],reverse=True)
    starting_words=get_starting_words(dataframe,selected_poet,'Unigrams')
    for i in range(SL):
        for j in range(VPS):
            random_number=np.random.randint(VMIN,VMAX)
            first_word=random.sample(starting_words,1)
            verse=[a[v][0] for v in range(1,random_number) ]
            print(first_word[0],' '.join(verse))
        print('\n')
```

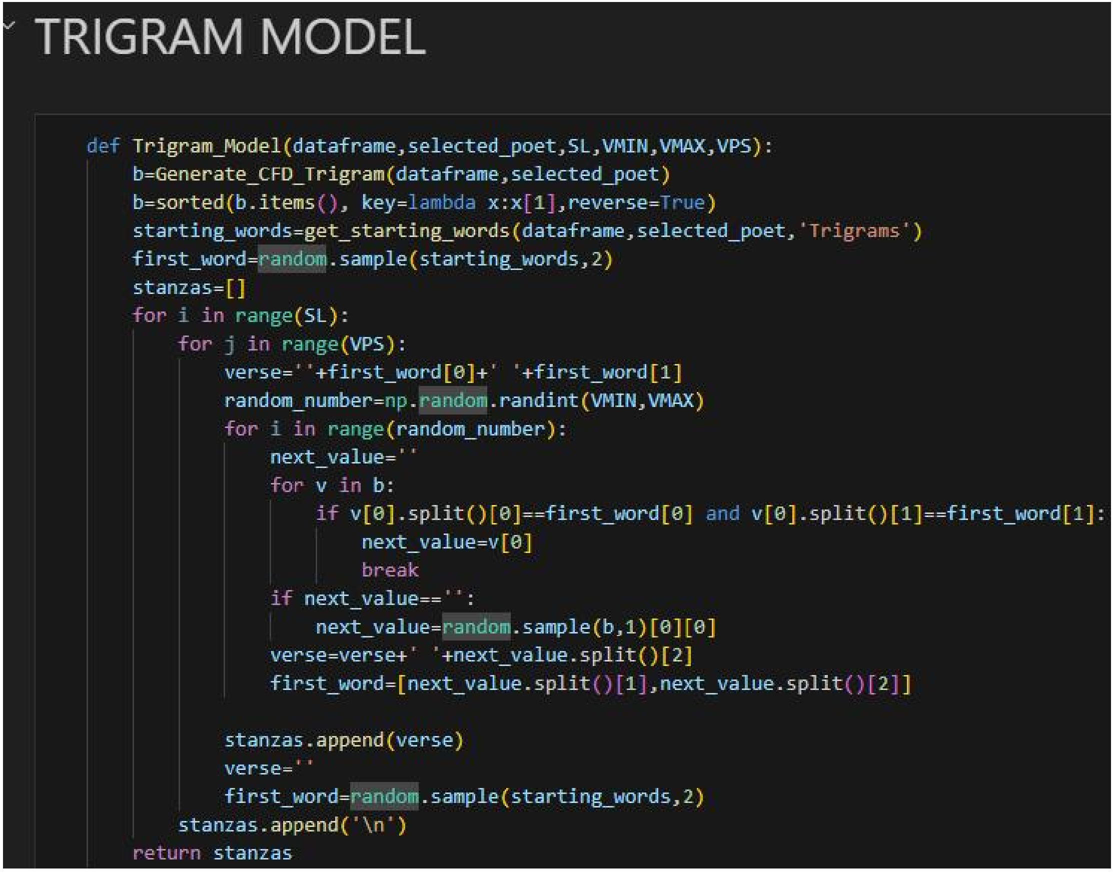
Bigram Model :



BIGRAM MODEL

```
def Bigram_Model(dataframe,selected_poet,SL,VMIN,VMAX,VPS):
    b=Generate_CFD_Bigram(dataframe,selected_poet)
    b=sorted(b.items(), key=lambda x:x[1],reverse=True)
    starting_words=get_starting_words(dataframe,selected_poet,'Bigrams')
    first_word=random.sample(starting_words,1)[0]
    stanzas=[]
    for i in range(SL):
        for j in range(VPS):
            verse=''+first_word
            random_number=np.random.randint(VMIN,VMAX)
            for i in range(random_number):
                next_value=''
                for v in b:
                    if v[0].split()[0]==first_word:
                        next_value=v[0]
                if next_value=='':
                    next_value=random.sample(b,1)[0][0]
                first_word=next_value.split()[1]
                verse=verse+' '+first_word
            stanzas.append(verse)
            verse=''
            first_word=random.sample(starting_words,1)[0]
        stanzas.append('\n')
    return stanzas
```

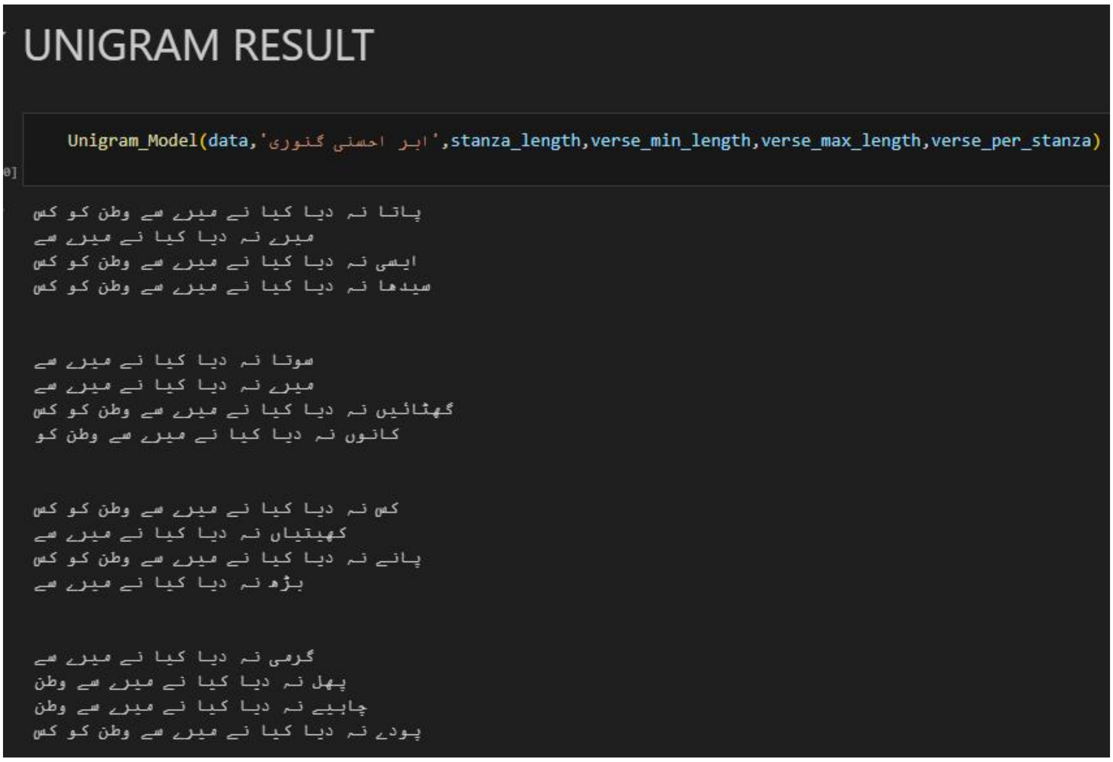
Trigram Model :



TRIGRAM MODEL

```
def Trigram_Model(dataframe,selected_poet,SL,VMIN,VMAX,VPS):
    b=Generate_CFD_Trigram(dataframe,selected_poet)
    b=sorted(b.items(), key=lambda x:x[1],reverse=True)
    starting_words=get_starting_words(dataframe,selected_poet,'Trigrams')
    first_word=random.sample(starting_words,2)
    stanzas=[]
    for i in range(SL):
        for j in range(VPS):
            verse=''+first_word[0]+' '+first_word[1]
            random_number=np.random.randint(VMIN,VMAX)
            for i in range(random_number):
                next_value=''
                for v in b:
                    if v[0].split()[0]==first_word[0] and v[0].split()[1]==first_word[1]:
                        next_value=v[0]
                        break
                if next_value=='':
                    next_value=random.sample(b,1)[0][0]
                verse=verse+' '+next_value.split()[2]
                first_word=[next_value.split()[1],next_value.split()[2]]

            stanzas.append(verse)
            verse=''
            first_word=random.sample(starting_words,2)
        stanzas.append('\n')
    return stanzas
```

Poetry Generation using Unigram:

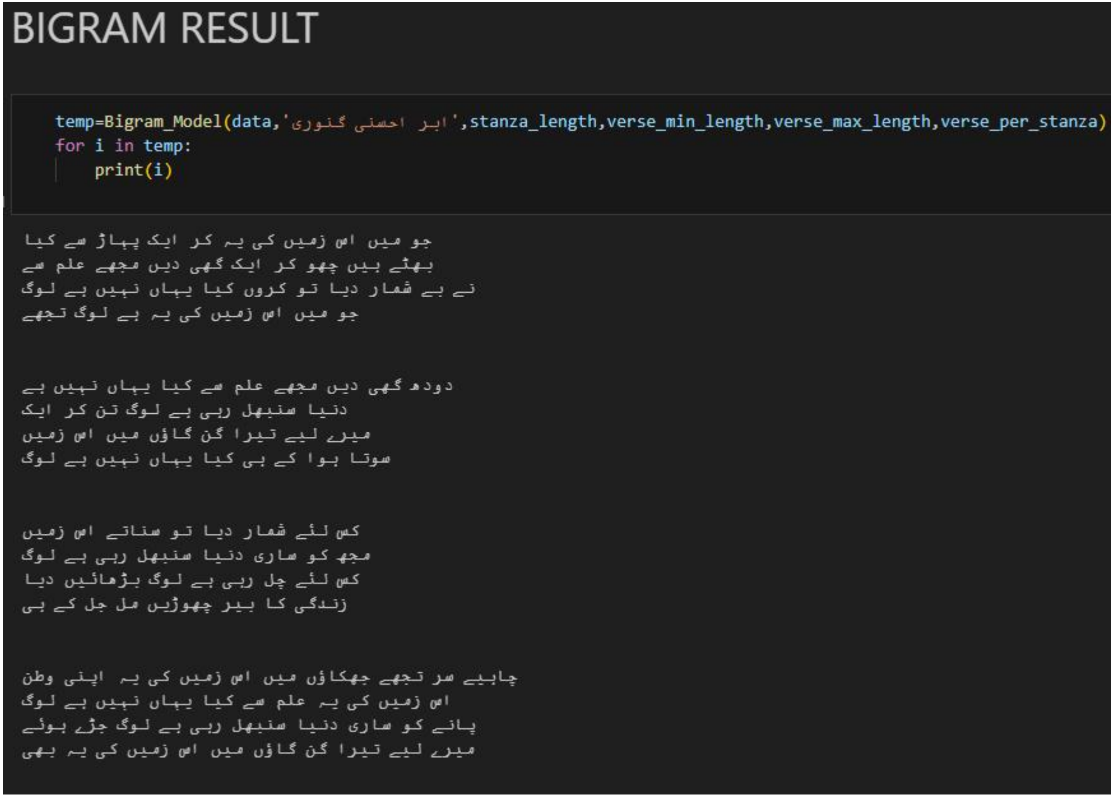


UNIGRAM RESULT

```
Unigram_Model(data,'ابر احسنی گنوری',stanza_length,verse_min_length,verse_max_length,verse_per_stanza)
```

```
پاتا نہ دیا کیا نے میرے سے وطن کو کس
میرے نہ دیا کیا نے میرے سے
ایسی نہ دیا کیا نے میرے سے وطن کو کس
سیدھا نہ دیا کیا نے میرے سے وطن کو کس


سوتا نہ دیا کیا نے میرے سے
میرے نہ دیا کیا نے میرے سے
گھٹائیں نہ دیا کیا نے میرے سے وطن کو کس
کانوں نہ دیا کیا نے میرے سے وطن کو


کس نہ دیا کیا نے میرے سے وطن کو کس
کھیتیاں نہ دیا کیا نے میرے سے
پانے نہ دیا کیا نے میرے سے وطن کو کس
بڑھ نہ دیا کیا نے میرے سے


گرمی نہ دیا کیا نے میرے سے
پھل نہ دیا کیا نے میرے سے وطن
چاہیے نہ دیا کیا نے میرے سے وطن
پودے نہ دیا کیا نے میرے سے وطن کو کس
```

Poetry Generation using Bigrams :

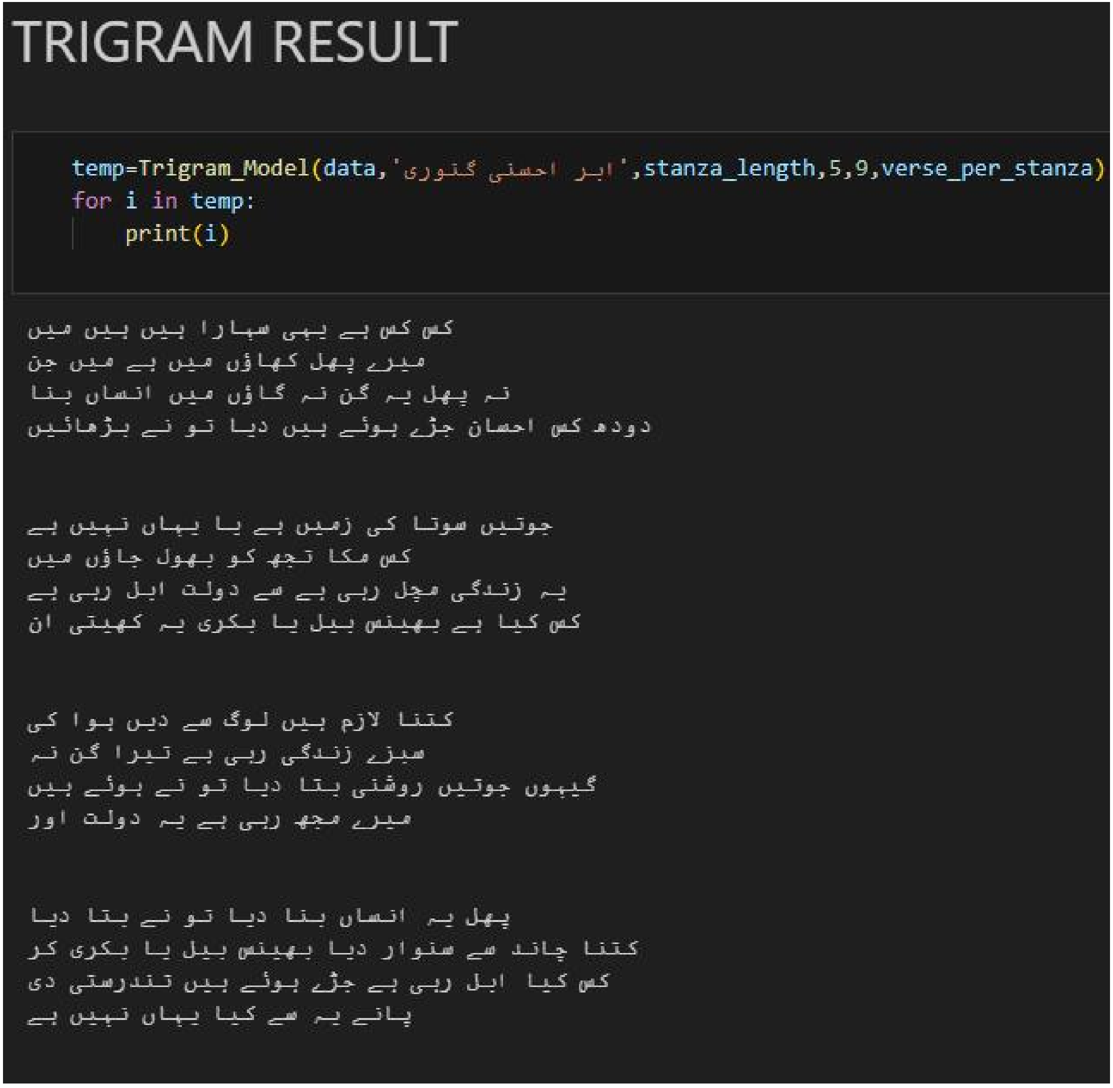


BIGRAM RESULT

```
temp=Bigram_Model(data,'ابر احسنی گنوری',stanza_length,verse_min_length,verse_max_length,verse_per_stanza)
for i in temp:
    print(i)
```

```
جو میں اس زمیں کی یہ کر ایک پہاڑ سے کیا
بہٹے ہیں چھو کر ایک گھی دیں مجھے علم سے
نے بے شمار دیا تو کروں کیا یہاں نہیں ہے لوگ
جو میں اس زمیں کی یہ ہے لوگ تجھے


دودھ گھی دیں مجھے علم سے کیا یہاں نہیں ہے
دنیا سنبھل رہی ہے لوگ تن کر ایک
میرے لیے تیرا گن گاؤں میں اس زمیں
سوتا ہوا کے ہی کیا یہاں نہیں ہے لوگ


کس لئے شمار دیا تو سناتے اس زمیں
مجھ کو ساری دنیا سنبھل رہی ہے لوگ
کس لئے چل رہی ہے لوگ بڑھائیں دیا
زندگی کا بیر چھوڑیں مل جل کے ہی


چاہیے سر تجھے جھکاؤں میں اس زمیں کی یہ اپنی وطن
اس زمیں کی یہ علم سے کیا یہاں نہیں ہے لوگ
پانے کو ساری دنیا سنبھل رہی ہے لوگ جڑے ہوئے
میرے لیے تیرا گن گاؤں میں اس زمیں کی یہ بھی
```

Poetry Genration using Trigrams :



TRIGRAM RESULT

```
    temp=Trigram_Model(data,'ابر احسنی گنوری',stanza_length,5,9,verse_per_stanza)
    for i in temp:
        print(i)
```

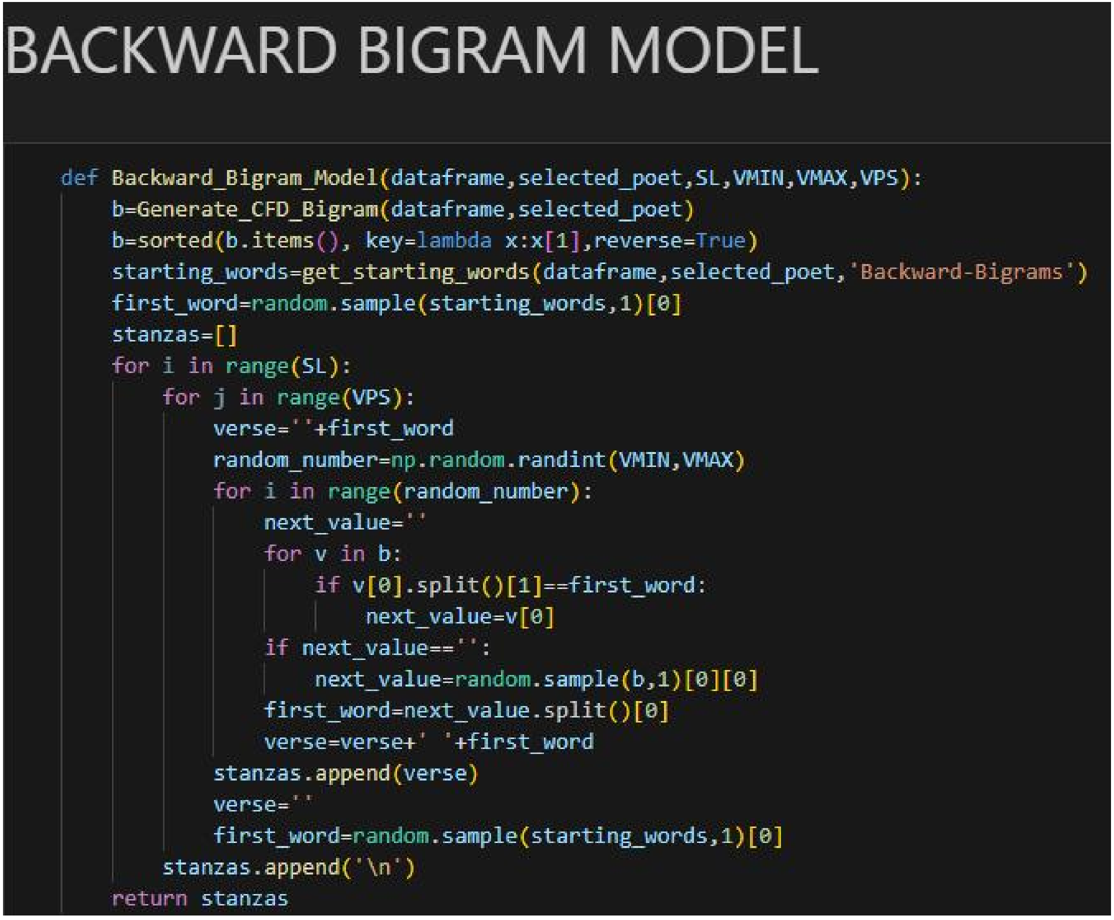
```
کس کس ہے یہی سہارا ہیں ہیں میں
میرے پھل کھاؤں میں ہے میں جن
نہ پھل یہ گن نہ گاؤں میں انسان بنا
دودھ کس احسان جڑے ہوئے ہیں دیا تو نے بڑھائیں


جوتیں سوتا کی زمیں ہے یا یہاں نہیں ہے
کس مکا تجھ کو بھول جاؤں میں
یہ زندگی مچل رہی ہے سے دولت ابل رہی ہے
کس کیا ہے بھینس بیل یا بکری یہ کھیتی ان


کتنا لازم ہیں لوگ سے دیں ہوا کی
سبزے زندگی رہی ہے تیرا گن نہ
گیہوں جوتیں روشنی بتا دیا تو نے بوئے ہیں
میرے مجھ رہی ہے یہ دولت اور


پھل یہ انسان بنا دیا تو نے بتا دیا
کتنا چاند سے سنوار دیا بھینس بیل یا بکری کر
کس کیا ابل رہی ہے جڑے ہوئے ہیں تندرستی دی
پانے یہ سے کیا یہاں نہیں ہے
```

Comparison of Results

Best Result : Trigram Model
Average Result : Bigram Model
Worst Result : Unigram Model

The reason is that if we talk about long term dependency , trigrams are always going to have the upper hand as they are predicting the next word based on the previous 2 words . Then further down the column , Bigrams predict the next word based on previous 1 word so they can adhere to the conext of the sentence to some extent. The worst case is of Unigrams . It just predicts a word based on the probabillity it ocurred in the document. It will never produce good result .

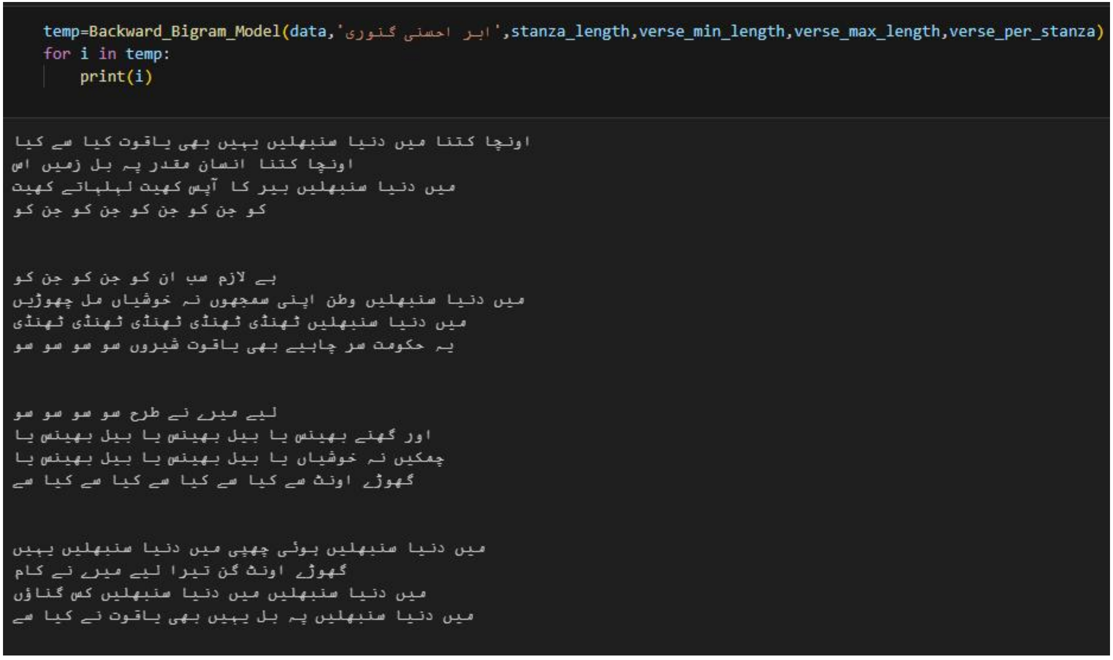
Backward Bigram Model :



BACKWARD BIGRAM MODEL

```
def Backward_Bigram_Model(dataframe,selected_poet,SL,VMIN,VMAX,VPS):
    b=Generate_CFD_Bigram(dataframe,selected_poet)
    b=sorted(b.items(), key=lambda x:x[1],reverse=True)
    starting_words=get_starting_words(dataframe,selected_poet,'Backward-Bigrams')
    first_word=random.sample(starting_words,1)[0]
    stanzas=[]
    for i in range(SL):
        for j in range(VPS):
            verse=''+first_word
            random_number=np.random.randint(VMIN,VMAX)
            for i in range(random_number):
                next_value=''
                for v in b:
                    if v[0].split()[1]==first_word:
                        next_value=v[0]
                if next_value=='':
                    next_value=random.sample(b,1)[0][0]
                first_word=next_value.split()[0]
                verse=verse+' '+first_word
            stanzas.append(verse)
            verse=''
            first_word=random.sample(starting_words,1)[0]
        stanzas.append('\n')
    return stanzas
```

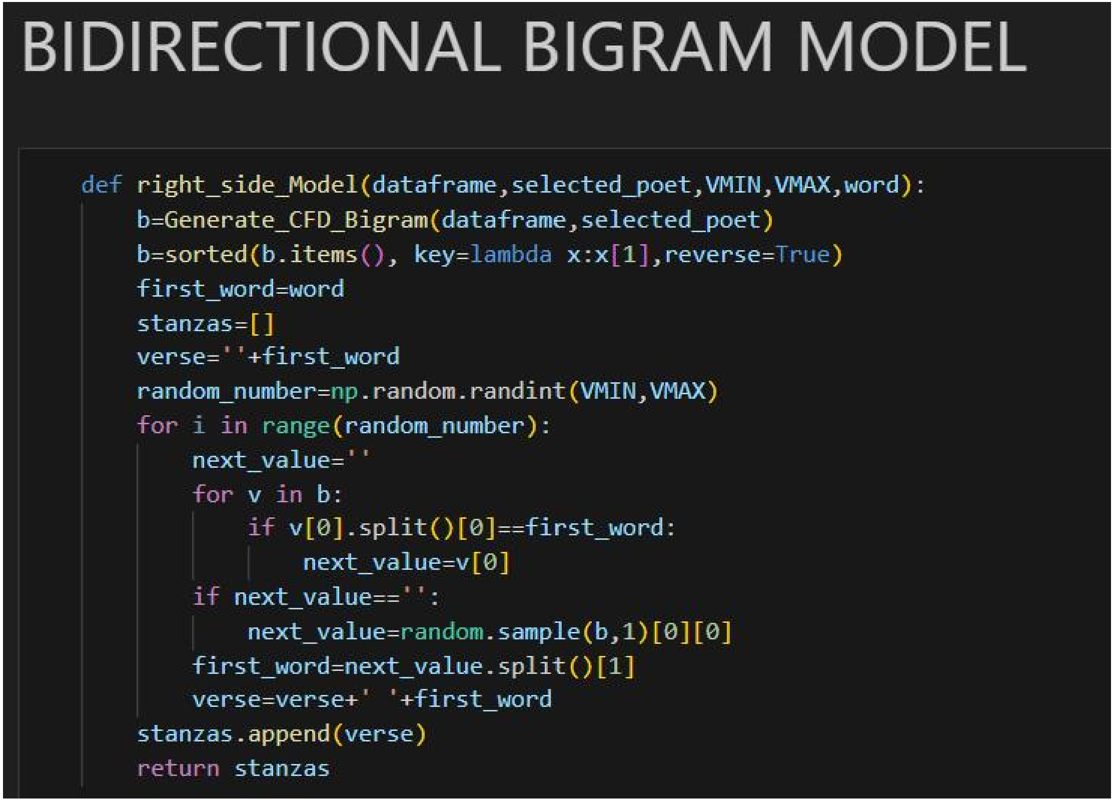
Poetry Generation using Backward Bigram Model:



```
temp=Backward_Bigram_Model(data,'ابر احسنی گنوری',stanza_length,verse_min_length,verse_max_length,verse_per_stanza)
for i in temp:
    print(i)
```

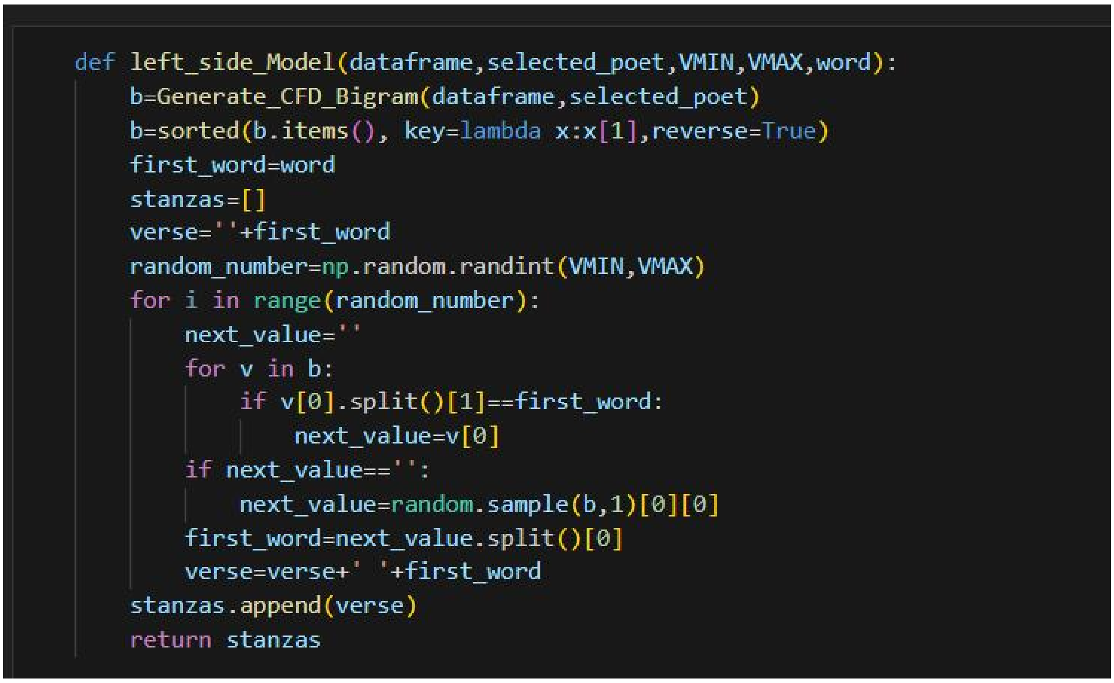
```
اونچا کتنا میں دنیا سنبھلیں یہیں بھی یاقوت کیا سے کیا
اونچا کتنا انسان مقدر پہ بل زمیں اس
میں دنیا سنبھلیں بیر کا آپس کھیت لہلہاتے کھیت
کو جن کو جن کو جن کو جن کو


بے لازم سب ان کو جن کو جن کو
میں دنیا سنبھلیں وطن اپنی سمجھوں نہ خوشیاں مل چھوڑیں
میں دنیا سنبھلیں ٹھنڈی ٹھنڈی ٹھنڈی ٹھنڈی ٹھنڈی
یہ حکومت سر چاہیے بھی یاقوت شیروں سو سو سو سو


لیے میرے نے طرح سو سو سو سو
اور گھنے بھینس یا بیل بھینس یا بیل بھینس یا
چمکیں نہ خوشیاں یا بیل بھینس یا بیل بھینس یا
گھوڑے اونٹ سے کیا سے کیا سے کیا سے


میں دنیا سنبھلیں ہوئی چھپی میں دنیا سنبھلیں یہیں
گھوڑے اونٹ گن تیرا لیے میرے نے کام
میں دنیا سنبھلیں میں دنیا سنبھلیں کس گناؤں
میں دنیا سنبھلیں پہ بل یہیں بھی یاقوت نے کیا سے
```

Bidirectional Bigram Model :

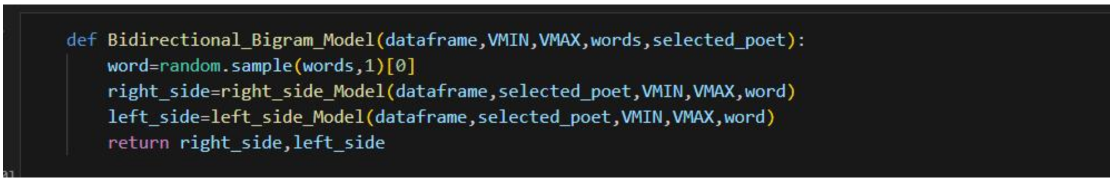


BIDIRECTIONAL BIGRAM MODEL

```
def right_side_Model(dataframe,selected_poet,VMIN,VMAX,word):
    b=Generate_CFD_Bigram(dataframe,selected_poet)
    b=sorted(b.items(), key=lambda x:x[1],reverse=True)
    first_word=word
    stanzas=[]
    verse=''+first_word
    random_number=np.random.randint(VMIN,VMAX)
    for i in range(random_number):
        next_value=''
        for v in b:
            if v[0].split()[0]==first_word:
                next_value=v[0]
        if next_value=='':
            next_value=random.sample(b,1)[0][0]
        first_word=next_value.split()[1]
        verse=verse+' '+first_word
    stanzas.append(verse)
    return stanzas
```

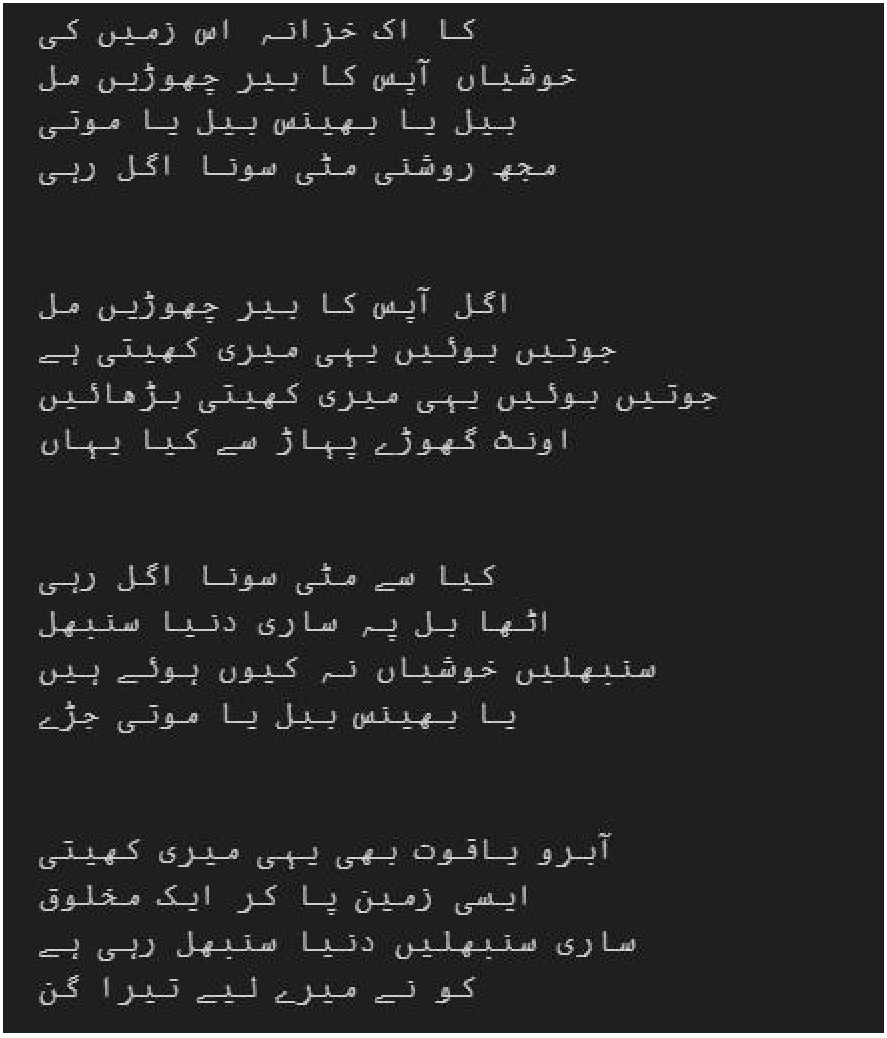


```
def left_side_Model(dataframe,selected_poet,VMIN,VMAX,word):
    b=Generate_CFD_Bigram(dataframe,selected_poet)
    b=sorted(b.items(), key=lambda x:x[1],reverse=True)
    first_word=word
    stanzas=[]
    verse=''+first_word
    random_number=np.random.randint(VMIN,VMAX)
    for i in range(random_number):
        next_value=''
        for v in b:
            if v[0].split()[1]==first_word:
                next_value=v[0]
        if next_value=='':
            next_value=random.sample(b,1)[0][0]
        first_word=next_value.split()[0]
        verse=verse+' '+first_word
    stanzas.append(verse)
    return stanzas
```



```
def Bidirectional_Bigram_Model(dataframe,VMIN,VMAX,words,selected_poet):
    word=random.sample(words,1)[0]
    right_side=right_side_Model(dataframe,selected_poet,VMIN,VMAX,word)
    left_side=left_side_Model(dataframe,selected_poet,VMIN,VMAX,word)
    return right_side,left_side
```

Poetry Generation using Bidirectional Bigram Model



```
کا اک خزاتہ اس زمین کی
خوشیاں آیس کا بیر چھوڑیں مل
بیل یا بھینس بیل یا موتی
مجھ روشنی مٹی سونا اگل رہی


اگل آیس کا بیر چھوڑیں مل
جوتیں بوئیں یہی میری کھیتی ہے
جوتیں بوئیں یہی میری کھیتی بڑھائیں
اونٹ گھوڑے پہاڑ سے کیا یہاں


کیا سے مٹی سونا اگل رہی
اٹھا بل یہ ساری دتیا ستبھل
ستبھلیں خوشیاں نہ کیوں ہوئے ہیں
یا بھینس بیل یا موتی جڑے


آبرو یاقوت بھی یہی میری کھیتی
ایسی زمین پا کر ایک مخلوق
ساری ستبھلیں دتیا ستبھل رہی ہے
کو نے میرے لیے تیرا گن
```

Introduced Optimization

The optimization I introduced was that CFD is being calculated at runtime rahter then pre-compiling it and then using the CFD table . Since it was going to take a lot of space , I on the other hand ask the user to enter the specific poet name and based on that poet and his/her specific poems , Unigrams , Bigrams , Trigrams and their repective CFD’s are calculated . By performing this I hav reduced the Space Complexity of the code since Already the dataframe was occupying quite some space .
No difficulty was faced since everything was explained quite good in the documentation .

THE END