Importing the Necessary Libraries

```
!pip install ctgan
!pip install table_evaluator
!pip install tensorflow
!pip install sdv
!pip install wordcloud
!pip install textwrap
!pip install spacy
!pip install textblob
```

```
from wordcloud import WordCloud
from textwrap import wrap
from nltk.corpus import stopwords
from collections import  Counter
from sklearn.feature_extraction.text import CountVectorizer
from matplotlib.ticker import StrMethodFormatter
from nltk.stem import WordNetLemmatizer,PorterStemmer
from statsmodels.graphics.mosaicplot import mosaic
from sdv.evaluation import evaluate
from ctgan import CTGANSynthesizer
from textblob import TextBlob
from nltk.stem import WordNetLemmatizer,PorterStemmer
from nltk.tokenize import word_tokenize
from wordcloud import WordCloud, STOPWORDS
```

```
import pandas as pd
import numpy as np
import re
import string
import matplotlib.pyplot as plt
import nltk
import seaborn as sns
import en_core_web_sm
import plotly.graph_objects as go
import spacy
```

```
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('stopwords')
```

Importing our Data

```
complaints_dt =
pd.read_csv('C:/Users/ryan/Downloads/consumer_complaints (1).csv')
```

#Data examination

```
complaints_dt.info()

complaints_dt.head()

complaints_dt.drop(['zipcode','date_received','consumer_complaint_narr
ative', 'company_public_response', 'sub_issue',
                    'sub_product', 'company', 'tags',
'consumer_consent_provided', 'date_sent_to_company','complaint_id',
                    'timely_response', 'consumer_disputed?'], axis=1,
inplace=True)
print(complaints_dt.columns)

complaints_dt.info()

complaints_dt.head()
```

Exploratory data analysis

```
fig1 = complaints_dt.groupby('product')
['product'].count().sort_values()
plt.figure(figsize=(15, 8))
fig1.plot(kind='barh')

response_values =
complaints_dt['company_response_to_consumer'].value_counts()
response_labels =
complaints_dt['company_response_to_consumer'].unique().tolist()
fig2= go.Figure(data=[go.Pie(values=response_values,
labels=response_labels, hole=.3)])
fig2.show()

complaints_dt1 = complaints_dt.dropna()

complaints_dt1.info()

complaints_dt1.shape
```

Expand Contractions

```
contractions_dict = { "ain't": "are not","'s":" is","aren't": "are
not",
                     "can't": "cannot","can't've": "cannot have",
                     "'cause": "because","could've": "could
have","couldn't": "could not",
                     "couldn't've": "could not have", "didn't": "did
not","doesn't": "does not",
                     "don't": "do not","hadn't": "had
not","hadn't've": "had not have",
                     "hasn't": "has not","haven't": "have not","he'd":
"he would",
                     "he'd've": "he would have","he'll": "he will",
```

```
"he'll've": "he will have",
                    "how'd": "how did","how'd'y": "how do
you","how'll": "how will",
                    "I'd": "I would", "I'd've": "I would
have","I'll": "I will",
                    "I'll've": "I will have","I'm": "I am","I've": "I
have", "isn't": "is not",
                    "it'd": "it would","it'd've": "it would
have","it'll": "it will",
                    "it'll've": "it will have", "let's": "let
us","ma'am": "madam",
                    "mayn't": "may not","might've": "might
have","mightn't": "might not",
                    "mightn't've": "might not have","must've": "must
have","mustn't": "must not",
                    "mustn't've": "must not have", "needn't": "need
not",
                    "needn't've": "need not have","o'clock": "of the
clock","oughtn't": "ought not",
                    "oughtn't've": "ought not have","shan't": "shall
not","sha'n't": "shall not",
                    "shan't've": "shall not have","she'd": "she
would","she'd've": "she would have",
                    "she'll": "she will", "she'll've": "she will
have","should've": "should have",
                    "shouldn't": "should not", "shouldn't've":
"should not have","so've": "so have",
                    "that'd": "that would","that'd've": "that would
have", "there'd": "there would",
                    "there'd've": "there would have", "they'd": "they
would",
                    "they'd've": "they would have","they'll": "they
will",
                    "they'll've": "they will have", "they're": "they
are","they've": "they have",
                    "to've": "to have","wasn't": "was not","we'd":
"we would",
                    "we'd've": "we would have","we'll": "we
will","we'll've": "we will have",
                    "we're": "we are","we've": "we have", "weren't":
"were not","what'll": "what will",
                    "what'll've": "what will have","what're": "what
are", "what've": "what have",
                    "when've": "when have","where'd": "where did",
"where've": "where have",
                    "who'll": "who will","who'll've": "who will
have","who've": "who have",
                    "why've": "why have","will've": "will
have","won't": "will not",
                    "won't've": "will not have", "would've": "would
```

```
have","wouldn't": "would not",
                      "wouldn't've": "would not have","y'all": "you
all", "y'all'd": "you all would",
                      "y'all'd've": "you all would have","y'all're":
"you all are",
                      "y'all've": "you all have", "you'd": "you
would","you'd've": "you would have",
                      "you'll": "you will","you'll've": "you will
have", "you're": "you are",
                      "you've": "you have"}


contractions_re=re.compile('(%s)' %
'|'.join(contractions_dict.keys()))

def contractions_expansion(text,contractions_dict=contractions_dict):
  def replace(match):
    return contractions_dict[match.group(0)]
  return contractions_re.sub(replace, text)

complaints_dt1['issue']=complaints_dt1['issue'].apply(lambda
x:contractions_expansion(x))
```

Lowercase the issues

```
complaints_dt1['cleaned']=complaints_dt1['issue'].apply(lambda x:
x.lower())
```

Remove Punctuations

```
complaints_dt1['cleaned']=complaints_dt1['cleaned'].apply(lambda x:
re.sub('[%s]' % re.escape(string.punctuation), '', x))

complaints_dt1.head()
```

Text looks after cleaning

```
for index,text in enumerate(complaints_dt1['cleaned'][35:40]):
  print('Review %d:\n'%(index+1),text)

stop=set(stopwords.words('english'))

def top_stopwords_barchart(text):
    stop=set(stopwords.words('english'))

    new= text.str.split()
    new=new.values.tolist()
    corpus=[word for i in new for word in i]
    from collections import defaultdict
```

```
    dic=defaultdict(int)
    for word in corpus:
        if word in stop:
            dic[word]+=1

    top=sorted(dic.items(), key=lambda x:x[1],reverse=True)[:10]
    x,y=zip(*top)
    plt.bar(x,y)

top_stopwords_barchart(complaints_dt1['cleaned'])

def top_non_stopwords_barchart(text):
    stop=set(stopwords.words('english'))

    new= text.str.split()
    new=new.values.tolist()
    corpus=[word for i in new for word in i]

    counter=Counter(corpus)
    most=counter.most_common()
    x, y=[], []
    for word,count in most[:40]:
        if (word not in stop):
            x.append(word)
            y.append(count)

    sns.barplot(x=y,y=x)

top_non_stopwords_barchart(complaints_dt1['cleaned'])

def top_ngrams_barchart(text, n=2):
    stop=set(stopwords.words('english'))

    new= text.str.split()
    new=new.values.tolist()
    corpus=[word for i in new for word in i]

    def _get_top_ngram(corpus, n=None):
        vec = CountVectorizer(ngram_range=(n, n)).fit(corpus)
        bag_of_words = vec.transform(corpus)
        sum_words = bag_of_words.sum(axis=0)
        words_freq = [(word, sum_words[0, idx])
                      for word, idx in vec.vocabulary_.items()]
        words_freq =sorted(words_freq, key = lambda x: x[1],
reverse=True)
        return words_freq[:10]

    top_n_bigrams=_get_top_ngram(text,n)[:10]
    x,y=map(list,zip(*top_n_bigrams))
    sns.barplot(x=y,y=x)
```

```
top_ngrams_barchart(complaints_dt1['cleaned'],2)

top_ngrams_barchart(complaints_dt1['cleaned'],3)
```

Preparing Text Data for sentiment analysis

Stopwords Removal & Lemmatization

```
nlp = spacy.load('en_core_web_sm',disable=['parser', 'ner'])

complaints_dt1['lemmatized']=complaints_dt1['cleaned'].apply(lambda x:
' '.join([token.lemma_ for token in list(nlp(x))

if (token.is_stop==False)]))
```

Creating Document Term Matrix

```
complaints_dt1_grouped=complaints_dt1[['product','lemmatized']].groupb
y(by='product').agg(lambda x:' '.join(x))
complaints_dt1_grouped.head()


from sklearn.feature_extraction.text import CountVectorizer
cv=CountVectorizer(analyzer='word')
data=cv.fit_transform(complaints_dt1_grouped['lemmatized'])
complaints_dt1_dtm = pd.DataFrame(data.toarray(),
columns=cv.get_feature_names())
complaints_dt1_dtm.index=complaints_dt1_grouped.index
complaints_dt1_dtm.head(3)
```

Word clouds for each product

```
def wordcloud_generation(data,title):
  wc = WordCloud(width=400,
height=330).generate_from_frequencies(data)
  plt.figure(figsize=(8,8))
  plt.imshow(wc, interpolation='bilinear')
  plt.axis("off")
  plt.title('\n'.join(wrap(title,60)),fontsize=15)
  plt.show()

complaints_dt1_dtm=complaints_dt1_dtm.transpose()

for index,product in enumerate(complaints_dt1_dtm.columns):
```

```
wordcloud_generation(complaints_dt1_dtm[product].sort_values(ascending
=False),product)

complaints_dt1['polarity']=complaints_dt1['lemmatized'].apply(lambda
x:TextBlob(x).sentiment.polarity)

product_polarity_sorted=pd.DataFrame(complaints_dt1.groupby('product')
['polarity'].mean().sort_values(ascending=True))

plt.figure(figsize=(15,8))
plt.xlabel('Polarity')
plt.ylabel('Products')
plt.title('Polarity of Different Product issues')
polarity_graph=plt.barh(np.arange(len(product_polarity_sorted.index)),
product_polarity_sorted['polarity'],color='red')


for bar,product in zip(polarity_graph,product_polarity_sorted.index):
  plt.text(0.01,bar.get_y()
+bar.get_width(),'{}'.format(product),fontsize=14,color='black')

for bar,polarity in
zip(polarity_graph,product_polarity_sorted['polarity']):
  plt.text(bar.get_width(),bar.get_y()
+bar.get_width(),'%.3f'%polarity,va='center',fontsize=12,color='black'
)

plt.yticks([])
plt.show()

complaints_cat = ['product', 'issue', 'state',
'submitted_via','company_response_to_consumer']

complaints_dt1.head()

complaints_dt1.drop(['cleaned','lemmatized','polarity'], axis=1,
inplace=True)
print(complaints_dt1.columns)
```

Original data subset

```
complaints_sample=
complaints_dt1.sample(frac=0.1).reset_index(drop=True)

complaints_sample.head()
```

Original data sample visualization

```
x1 = complaints_sample.groupby('product')
['product'].count().sort_values()
```

```
plt.figure(figsize=(15, 8))
plt.title('product values frequencies', fontsize=20)
x1.plot(kind='bar')

def Wordcloud_plot(text):
    nltk.download('stopwords')
    stop=set(stopwords.words('english'))

    def _preprocess_text(text):
        corpus=[]
        stem=PorterStemmer()
        lem=WordNetLemmatizer()
        for news in text:
            words=[w for w in word_tokenize(news) if (w not in stop)]

            words=[lem.lemmatize(w) for w in words if len(w)>2]

            corpus.append(words)
        return corpus

    corpus=_preprocess_text(text)

    wordcloud = WordCloud(
        background_color='white',
        stopwords=set(STOPWORDS),
        max_words=100,
        max_font_size=30,
        scale=3,
        random_state=1)

    wordcloud=wordcloud.generate(str(corpus))

    fig = plt.figure(1, figsize=(12, 12))
    plt.axis('off')

    plt.imshow(wordcloud)
    plt.show()

Wordcloud_plot(complaints_sample['issue'])

values = complaints_sample['submitted_via'].value_counts()
labels =  complaints_sample['submitted_via'].unique().tolist()
fig= go.Figure(data=[go.Pie(values=values, labels=labels, hole=.3)])
fig.show()

x = complaints_sample.groupby('company_response_to_consumer')
['company_response_to_consumer'].count().sort_values()
plt.figure(figsize=(15, 8))
plt.title('company_response_to_consumer values frequencies',
fontsize=20)
x.plot(kind='barh')
```

Model training

```
ctgan = CTGANSynthesizer(verbose=True)
ctgan.fit(complaints_sample, complaints_cat, epochs = 100)
```

Synthetic data sample

```
samples = ctgan.sample(55000)

print(samples.head())
```

Synthetic data sample visualization

```
x1 = samples.groupby('product')['product'].count().sort_values()
plt.figure(figsize=(15, 8))
plt.title('product values frequencies', fontsize=20)
x1.plot(kind='bar')

def Wordcloud_plot(text):
    nltk.download('stopwords')
    stop=set(stopwords.words('english'))

    def _preprocess_text(text):
        corpus=[]
        stem=PorterStemmer()
        lem=WordNetLemmatizer()
        for news in text:
            words=[w for w in word_tokenize(news) if (w not in stop)]

            words=[lem.lemmatize(w) for w in words if len(w)>2]

            corpus.append(words)
        return corpus

    corpus=_preprocess_text(text)

    wordcloud = WordCloud(
        background_color='white',
        stopwords=set(STOPWORDS),
        max_words=100,
        max_font_size=30,
        scale=3,
        random_state=1)

    wordcloud=wordcloud.generate(str(corpus))

    fig = plt.figure(1, figsize=(12, 12))
    plt.axis('off')
```

```
    plt.imshow(wordcloud)
    plt.show()

Wordcloud_plot(samples['issue'])

values = samples['submitted_via'].value_counts()
labels =  samples['submitted_via'].unique().tolist()
fig= go.Figure(data=[go.Pie(values=values, labels=labels, hole=.3)])
fig.show()

x = samples.groupby('company_response_to_consumer')
['company_response_to_consumer'].count().sort_values()
plt.figure(figsize=(15, 8))
plt.title('company_response_to_consumer values frequencies',
fontsize=20)
x.plot(kind='barh')
```

Orignal and sythetic data sample evaluation

```
evaluate(samples, complaints_sample)
```