Mahesh Code

by A B

Submission date: 09-Jul-2023 11:13AM (UTC+0100)
Submission ID: 210000931
File name: Mahesh_Code.pdf (1.59M)
Word count: 3725
Character count: 22696

1. Importing and Installing Program Dependencies

1.1 Importing the libraries

In [1]:

```
import pandas as pd
import numpy as np
import re
import spacy
import unicodedata
import nltk
import joblib
import utils as utils

from sklearn.linear_model import LinearRegression
from sklearn import tree

import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn import metrics

from sklearn import ensemble
from sklearn import naive_bayes
from sklearn.svm import LinearSVC

from sklearn.metrics import confusion_matrix, classification_report, accuracy_score


from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from wordcloud import WordCloud, STOPWORDS

import warnings
warnings.filterwarnings('ignore')
```

1.2 Installing the toolkit for processing natural language

In [2]:

```
#nltk.download('punkt')
#nltk.download('averaged_perceptron_tagger')
#nltk.download('maxent_ne_chunker')
#nltk.download('words')
```

2. Loading data

In [3]:

```
data = pd.read_csv(r'flipkart_com-ecommerce_sample.csv')
data.head(5)
```

Out[3]:

	uniq_id	crawl_timestamp	product_url
0	c2d766ca982eca8304150849735ffef9	2016-03-25 22:59:23 +0000	http://www.flipkart.com/alisha-solid-women-s-c...
1	7f7036a6d550aaa89d34c77bd39a5e48	2016-03-25 22:59:23 +0000	http://www.flipkart.com/fabhomedecor-fabric-do...
2	f449ec65dcbc041b6ae5e6a32717d01b	2016-03-25 22:59:23 +0000	http://www.flipkart.com/aw-bellies/p/itmeh4grg...
3	0973b37acd0c664e3de26e97e5571454	2016-03-25 22:59:23 +0000	http://www.flipkart.com/alisha-solid-women-s-c...
4	bc940ea42ee6bef5ac7cea3fb5cfbee7	2016-03-25 22:59:23 +0000	http://www.flipkart.com/sicons-all-purpose-arn...

3. Data preprocessing

3.1 Information of the dataset

In [4]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20002 entries, 0 to 20001
Data columns (total 15 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   uniq_id                  20000 non-null  object 
 1   crawl_timestamp          20000 non-null  object 
 2   product_url              20000 non-null  object 
 3   product_name             20000 non-null  object 
 4   product_category_tree    20000 non-null  object 
 5   pid                      20000 non-null  object 
 6   retail_price             19922 non-null  float64
 7   discounted_price         19922 non-null  float64
 8   image                    19997 non-null  object 
 9   is_FK_Advantage_product  20000 non-null  object 
 10  description              19998 non-null  object 
 11  product_rating           20000 non-null  object 
 12  overall_rating           20000 non-null  object 
 13  brand                    14136 non-null  object 
 14  product_specifications   19986 non-null  object 
dtypes: float64(2), object(13)
memory usage: 2.3+ MB
```

3.2 Checking for NULL values

In [5]:

```
data.isna().sum()
```

Out[5]:

```
uniq_id                    2
crawl_timestamp            2
product_url                2
product_name               2
product_category_tree      2
pid                        2
retail_price              80
discounted_price          80
image                      5
is_FK_Advantage_product    2
description                4
product_rating             2
overall_rating             2
brand                   5866
product_specifications    16
dtype: int64
```

In [6]:

```
data = data.dropna()
```

In [7]:

```
data.isna().sum()
```

Out[7]:

```
uniq_id                   0
crawl_timestamp           0
product_url               0
product_name              0
product_category_tree     0
pid                       0
retail_price              0
discounted_price          0
image                     0
is_FK_Advantage_product   0
description               0
product_rating            0
overall_rating            0
brand                     0
product_specifications    0
dtype: int64
```

3.3 Checking for the data types

In [8]:

```
data.dtypes
```

Out[8]:

```
uniq_id                   object
crawl_timestamp           object
product_url               object
product_name              object
product_category_tree     object
pid                       object
retail_price             float64
discounted_price         float64
image                     object
is_FK_Advantage_product   object
description               object
product_rating            object
overall_rating            object
brand                     object
product_specifications    object
dtype: object
```

4. Text data processing

4.1 Stopwords

In [9]:

```
stopwords = stopwords.words('english')
unnecessary_stopwords = {'no', 'nor', 'not', 'ain', 'aren', "aren't", 'couldn', 'what',
                                      'whom',
                                      'why', 'how', "couldn't", 'didn', "didn't", 'doesn
                                      'hasn',
                                      "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma'
                                      "mustn't", 'needn', "needn't", 'shan', "shan't", '
                                      "wasn't",
                                      'weren', "weren't", 'won', "won't", 'wouldn', "wou

stopwords = [i for i in stopwords if i not in unnecessary_stopwords]
lemma = WordNetLemmatizer()
nouns = ['NNP', 'NNPS']
nlp = spacy.load('en_core_web_sm')
labelencoder = LabelEncoder()
```

4.1 Label extraction

Spliting the "product_category_tree" column usiung ">>" as the separator to add a column namely "Labels" in the dataset

In [10]:

```
def extract_labels(data, top=5):
    data['Labels'] = data['product_category_tree'].apply(lambda x: x.split('>>')[0][2:].
    top_items = list(data.groupby('Labels').count().sort_values(by='uniq_id',ascending=F
    data = data[data['Labels'].isin(top_items)]
    return data

data = extract_labels(data, top=10)
data.Labels.value_counts()
```

Out[10]:

```
Jewellery                    3521
Clothing                     3129
Mobiles & Accessories        1096
Automotive                   1010
Home Decor & Festive Needs    859
Home Furnishing               698
Computers                     572
Baby Care                     455
Tools & Hardware              387
Footwear                      375
Name: Labels, dtype: int64
```

4.2 Cleaning text data

In [11]:

```
def text_cleaner(text, remove_stopwords=True, lemmatize=True):
    #removing mailing addresses
    text = re.sub('\S*@\S*\s?', '', text)
    # Remove new line characters
    text = re.sub('\s+', ' ', text)
    # Remove distracting single quotes
    text = re.sub("\'", '', text)

    # cleaning the puntuations and numbers
    text = re.sub('[^a-zA-Z]', ' ', text)

    # removing single characters
    text = re.sub('\s+[a-zA-Z]\s+^I', ' ', text)
    #removing accented words
    text = unicodedata.normalize('NFKD', text).encode('ascii', 'ignore').decode('utf-8',

    # removing multiple spaces
    text = re.sub(r'\s+', ' ', text)
    text = re.sub(r'^\s*|\s\s*', ' ', text).strip()
    text = text.lower()

    if not remove_stopwords and not lemmatize:
        return text

    # removing unncecessay stopwords
    if remove_stopwords:
        text = word_tokenize(text)
        text = " ".join([word for word in text if word not in stopwords])
        # Word lemmatization
        if lemmatize:
            text = nlp(text)
            lemmatized_text = []
            for word in text:
                if word.lemma_.isalpha():
                    if word.lemma_ != '-PRON-':
                        lemmatized_text.append(word.lemma_.lower())
                        text = " ".join([word.lower() for word in lemmatized_text])

                        return text
```

4.3 Featuring Scaling

In [12]:

```
def ct_vectorize(x_train, x_test, analyzer='word', token_pattern=r'\w{1,}', max_features
    if analyzer == 'word':
        cvector = CountVectorizer(analyzer=analyzer,
                                  token_pattern=token_pattern,
                                  max_features=max_features,
                                  ngram_range=ngram_range)
    else:
        cvector = CountVectorizer(analyzer=analyzer,
                                      max_features=max_features,
                                      ngram_range=ngram_range)
    cvector.fit(x_train)

    # transform the training and validation data using count vectorizer object
    count_xtrain = cvector.transform(x_train)
    count_xvalid = cvector.transform(x_test)

    return count_xtrain, count_xvalid
```

In [13]:

```
def tfidf_vectorize(x_train, x_test, analyzer='word', token_pattern=r'\w{1,}', max_featu
    if analyzer == 'word':
        tfidfvector = TfidfVectorizer(analyzer=analyzer,
                                      token_pattern=token_pattern,
                                      max_features=max_features,
                                      ngram_range=ngram_range)
    else:
        tfidfvector = TfidfVectorizer(analyzer=analyzer,
                                          max_features=max_features,
                                          ngram_range=ngram_range)
    tfidfvector.fit(x_train)

    xtrain_tfidf = tfidfvector.transform(x_train)
    xvalid_tfidf = tfidfvector.transform(x_test)

    return xtrain_tfidf, xvalid_tfidf
```

4.4 Preparing data for model training

In [14]:

```
def data_preparation(data, test_size=0.3, remove_stopwords=True, lemmatize=True):
    data = data.sample(frac=1).reset_index(drop=True)
    des = data['description'].apply(lambda x: text_cleaner(str(x), remove_stopwords=remo
    Lablels = data['Labels'].values.tolist()
    Lablels = labelencoder.fit_transform(Lablels)
    x_train, x_test, y_train, y_test = train_test_split(des,
                                                        Lablels,
                                                        test_size=test_size,
                                                        stratify=Lablels)
    return x_train, x_test, y_train, y_test
```

4.5 Train-Test Split (70%-30%)

In [15]:

```
x_train, x_test, y_train, y_test = data_preparation(data, test_size=0.3, remove_stopword

print("Total training examples: ", len(x_train))
print("Total test examples: ", len(x_test))
```

```
Total training examples:  8471
Total test examples:  3631
```

4.6 Feature generation

In [16]:

```
# Word frequency based representation (only unigrams)
xtrain_count, xvalid_count = ct_vectorize(x_train, x_test, analyzer='word', token_patter
                                          max_features=10000, ngram_range=(1,1))

# Word frequency based representation (unigrams and bigrams)
xtrain_count_freq1, xvalid_count_freq1 = ct_vectorize(x_train, x_test, analyzer='word',
                                                      max_features=10000, ngram_range=(1

xtrain_count_freq2, xvalid_count_freq2 = ct_vectorize(x_train, x_test, analyzer='word',
                                                      max_features=10000, ngram_range=(1


xtrain_tfidf, xvalid_tfidf = tfidf_vectorize(x_train, x_test, analyzer='word', token_pat
                                             max_features=10000, ngram_range=(1,1))


xtrain_tfidf_freq1, xvalid_tfidf_freq1 = tfidf_vectorize(x_train, x_test, analyzer='word
                                                         max_features=10000, ngram_range


xtrain_tfidf_freq2, xvalid_tfidf_freq2 = tfidf_vectorize(x_train, x_test, analyzer='word
                                                         max_features=10000, ngram_range

xtrain_tfidf_freq2_str, xvalid_tfidf_freq2_str = tfidf_vectorize(x_train, x_test, analyz
                                                                 max_features=10000, ngr
print("\nFeatures generated!")
```

```
Features generated!
```

5. Statistical Analysis

In [17]:

```
d = data
```

In [18]:

```
d[['uniq_id']] = d[['uniq_id']].apply(lambda col: pd.Categorical(col).codes)
d[['product_url']] = d[['product_url']].apply(lambda col: pd.Categorical(col).codes)
d[['product_name']] = d[['product_name']].apply(lambda col: pd.Categorical(col).codes)
d[['product_category_tree']] = d[['product_category_tree']].apply(lambda col: pd.Categor
d[['pid']] = d[['pid']].apply(lambda col: pd.Categorical(col).codes)
d[['image']] = d[['image']].apply(lambda col: pd.Categorical(col).codes)
d[['is_FK_Advantage_product']] = d[['is_FK_Advantage_product']].apply(lambda col: pd.Cat
d[['description']] = d[['description']].apply(lambda col: pd.Categorical(col).codes)
d[['product_rating']] = d[['product_rating']].apply(lambda col: pd.Categorical(col).code
d[['overall_rating']] = d[['overall_rating']].apply(lambda col: pd.Categorical(col).code
d[['brand']] = d[['brand']].apply(lambda col: pd.Categorical(col).codes)
d[['product_specifications']] = d[['product_specifications']].apply(lambda col: pd.Categ
```

5.1 Description of data

In [19]:

```
d.describe()
```

Out[19]:

	uniq_id	product_url	product_name	product_category_tree	pid	
count	12102.000000	12102.000000	12102.000000	12102.000000	12102.000000	12
mean	6050.500000	6050.500000	3807.754751	2060.016278	6049.042555	3
std	3493.690813	3493.690813	2045.671997	1038.791261	3493.106665	8
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	3025.250000	3025.250000	2085.000000	1246.000000	3024.250000	
50%	6050.500000	6050.500000	3735.000000	2469.500000	6048.500000	1
75%	9075.750000	9075.750000	5606.750000	2941.000000	9073.750000	2
max	12101.000000	12101.000000	7252.000000	3347.000000	12099.000000	116

5.2 Correlation

In [20]:

```
d.corr().abs()
```

Out[20]:

	uniq_id	product_url	product_name	product_category_tree	
uniq_id	1.000000	0.000734	0.002891	0.007983	0.00
product_url	0.000734	1.000000	0.934848	0.097472	0.02
product_name	0.002891	0.934848	1.000000	0.107389	0.02
product_category_tree	0.007983	0.097472	0.107389	1.000000	0.20
pid	0.007281	0.020175	0.025211	0.203795	1.00
retail_price	0.020501	0.041806	0.052476	0.169690	0.14
discounted_price	0.020467	0.027527	0.038332	0.189493	0.15
image	0.007130	0.021541	0.018844	0.050115	0.45
is_FK_Advantage_product	0.005163	0.035840	0.030950	0.061636	0.04
description	0.009680	0.663922	0.683888	0.024494	0.05
product_rating	0.000503	0.013191	0.010182	0.013947	0.01
overall_rating	0.000503	0.013191	0.010182	0.013947	0.01
brand	0.000681	0.814515	0.879655	0.088948	0.06
product_specifications	0.002864	0.081707	0.095616	0.029792	0.30

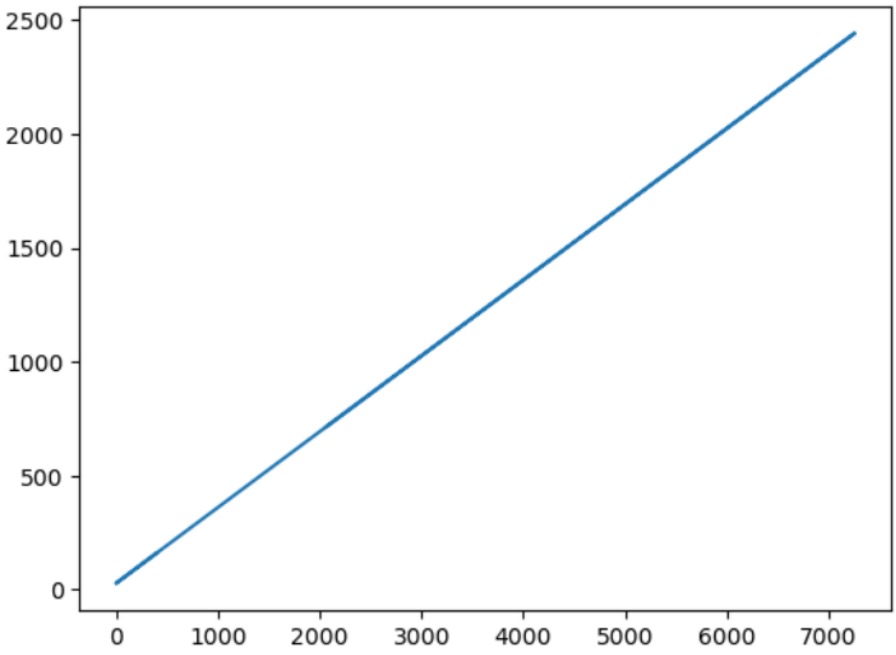
In [21]:

```
x = np.array(d['product_name']).reshape(-1, 1)
y = np.array(d['brand']).reshape(-1, 1)

linearmodel = LinearRegression().fit(x, y)

print("\n==================================")
print('1. Score:\n', linearmodel.score(x, y))
print("\n==================================")
print('2. Model Coefficient:\n', linearmodel.coef_)
print("\n==================================")
print('3. Model Intercept:\n',linearmodel.intercept_)
print("\n==================================")
print('4. Predictions made:\n',linearmodel.predict(x))
print("\n==================================")

plt.plot(x, linearmodel.predict(x))
```

```
==================================
1. Score:
 0.7737920418324518

==================================
2. Model Coefficient:
 [[0.33269957]]

==================================
3. Model Intercept:
 [28.37367674]

==================================
4. Predictions made:
 [[159.79000543]
 [ 78.27861169]
 [159.79000543]
 ...
 [714.73288202]
 [714.73288202]
 [714.73288202]]

==================================
```

Out[21]:

```
[<matplotlib.lines.Line2D at 0x195e643ff10>]
```


2500
2000
1500
1000
500
0
0
1000
2000
3000
4000
5000
6000
7000

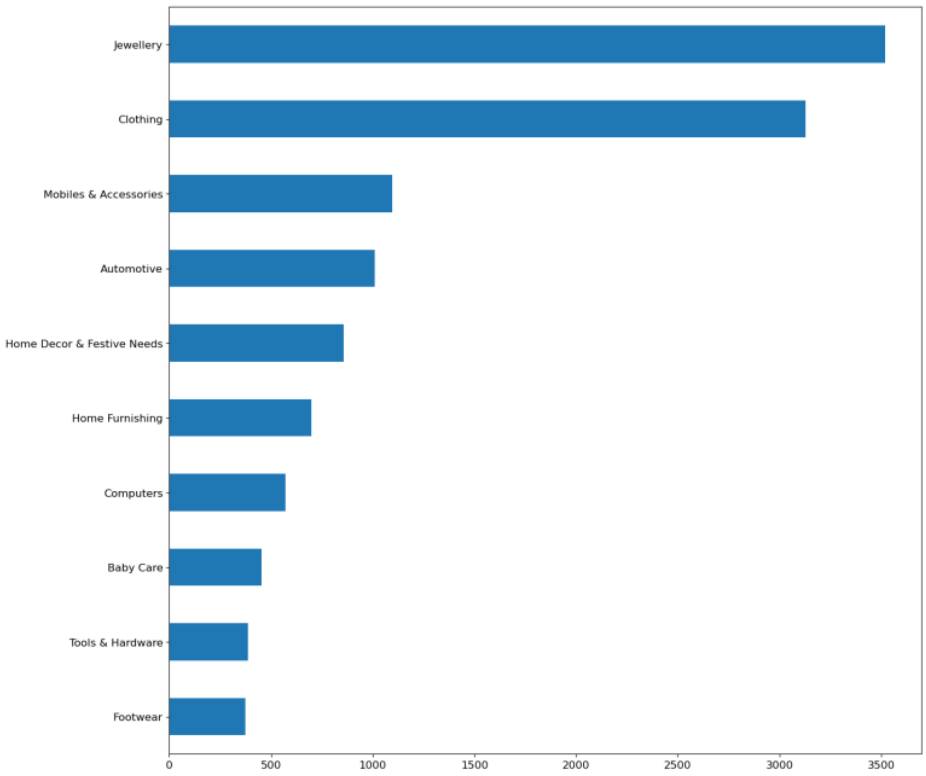
6.1 Visualizing the frequencies of the occurences of the labeled products in the dataset

In [22]:

```
plt.figure(figsize=(15,15))
data['Labels'].value_counts().sort_values(ascending=True).plot(kind='barh')
plt.yticks(fontsize=12)
plt.xticks(fontsize=12)
```

Out[22]:

```
(array([   0.,  500., 1000., 1500., 2000., 2500., 3000., 3500., 4000.]),
 [Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, '')])
```



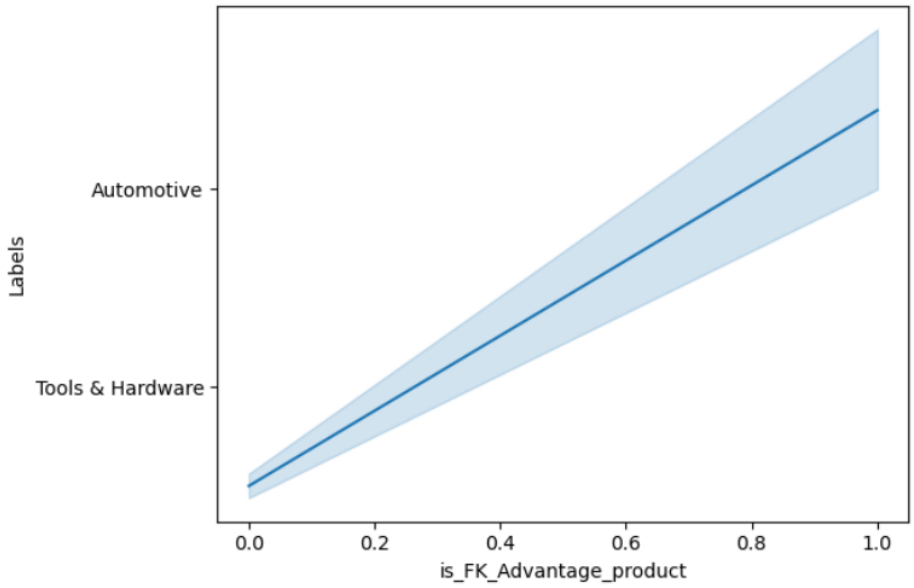
6.2 Lineplot represeanting the "is_FK_Advantage_product" and "Labels" entities

In [23]:

```
sns.lineplot(data['is_FK_Advantage_product'], data['Labels'])
```

Out[23]:

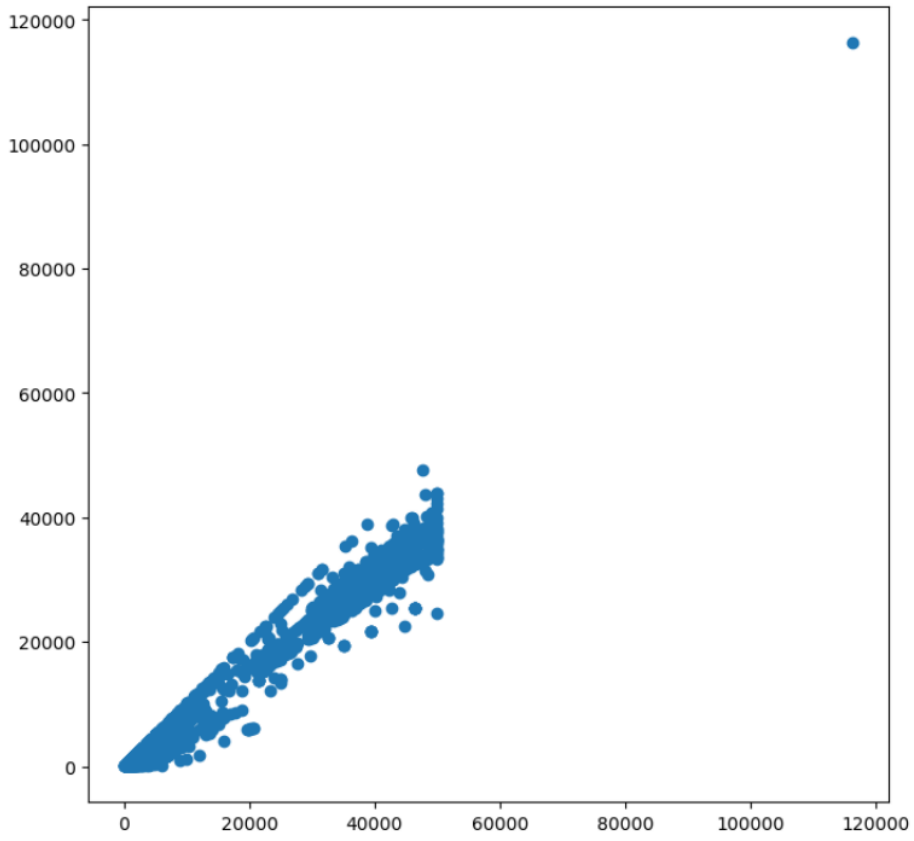
```
<AxesSubplot:xlabel='is_FK_Advantage_product', ylabel='Labels'>
```



6.3 Scatterplot for "retail_price" and "discounted_price"

In [24]:

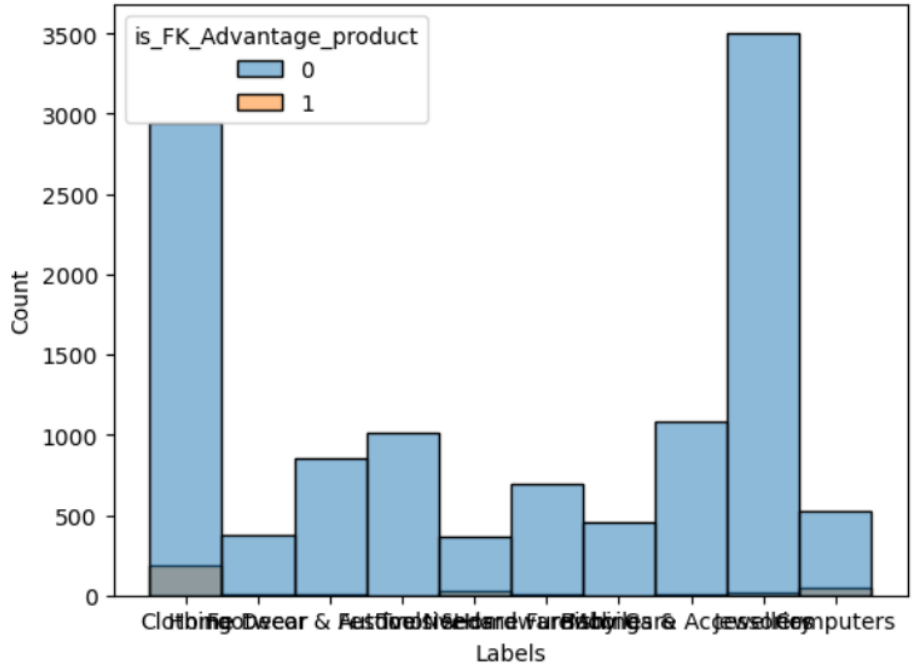
```
plt.figure(figsize=(8,8))
plt.scatter(data['retail_price'],data['discounted_price'], cmap='rainbow')
plt.show()
```



6.4 Histogram for comparing the "Labels" on the basis of "is_FK_Advantage_product"

In [25]:

```
sns.histplot(x='Labels', data=data, kde=False, hue='is_FK_Advantage_product')
plt.show()
```



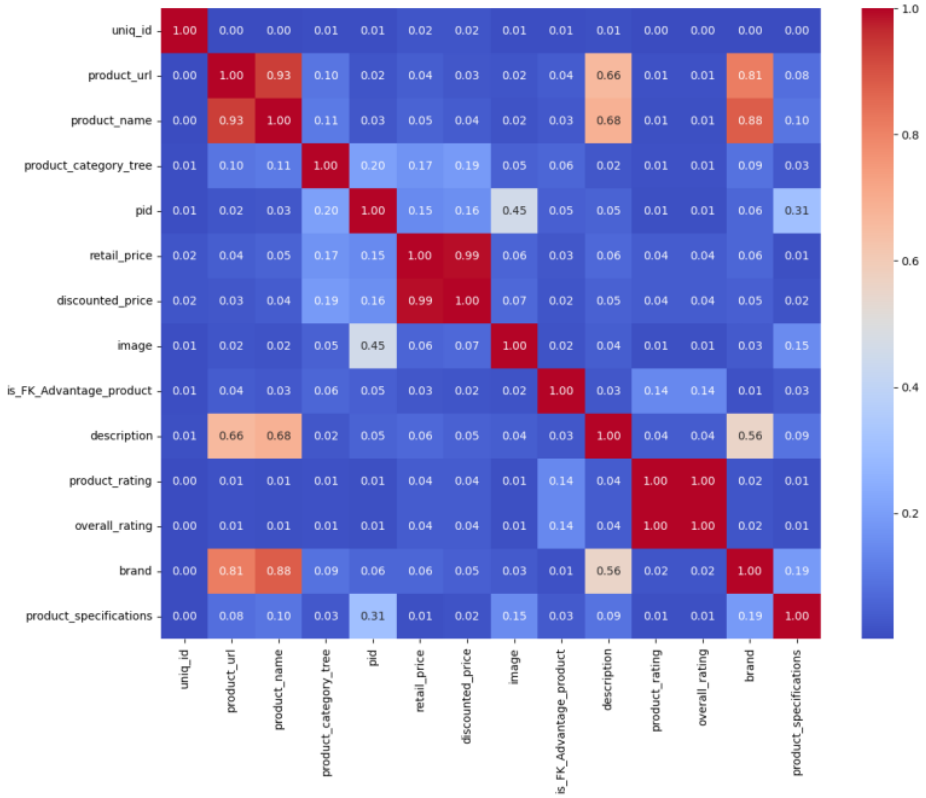
6.5 Heatmap

In [26]:

```
plt.figure(figsize=(13,10))
sns.heatmap(d.corr().abs(),annot=True, fmt = ".2f", cmap = "coolwarm")
```

Out[26]:

<AxesSubplot:>



7. Model training in Machine Learning

7.1 Defining the driver function

In [27]:

```
def train_models(model, x_train, x_test, y_train, y_test):
    model.fit(x_train, y_train)
    pred = model.predict(x_test)
    acc = metrics.accuracy_score(pred, y_test)
    confmat = confusion_matrix(y_test, pred)
    return model, acc, confmat
```

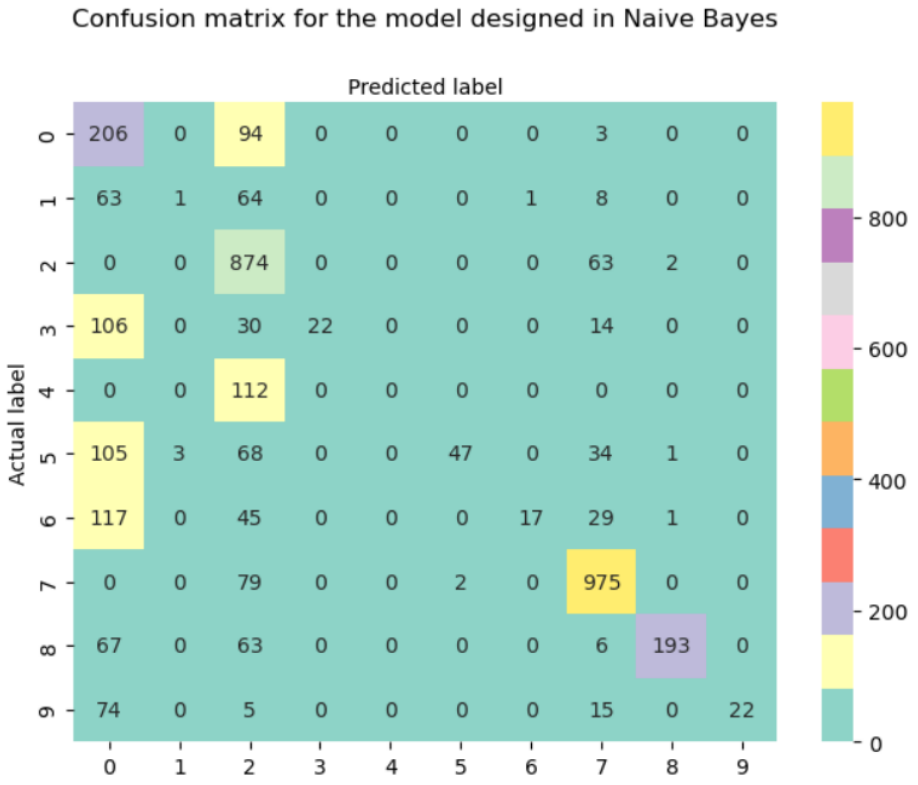
7.2 Naive Bayes

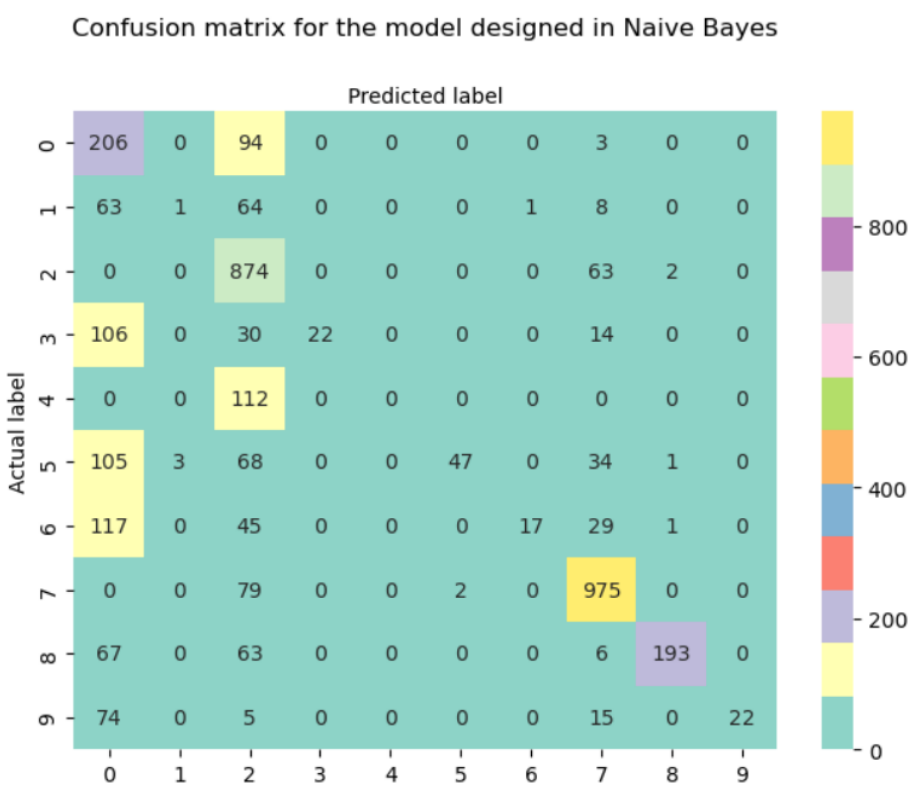
In [28]:

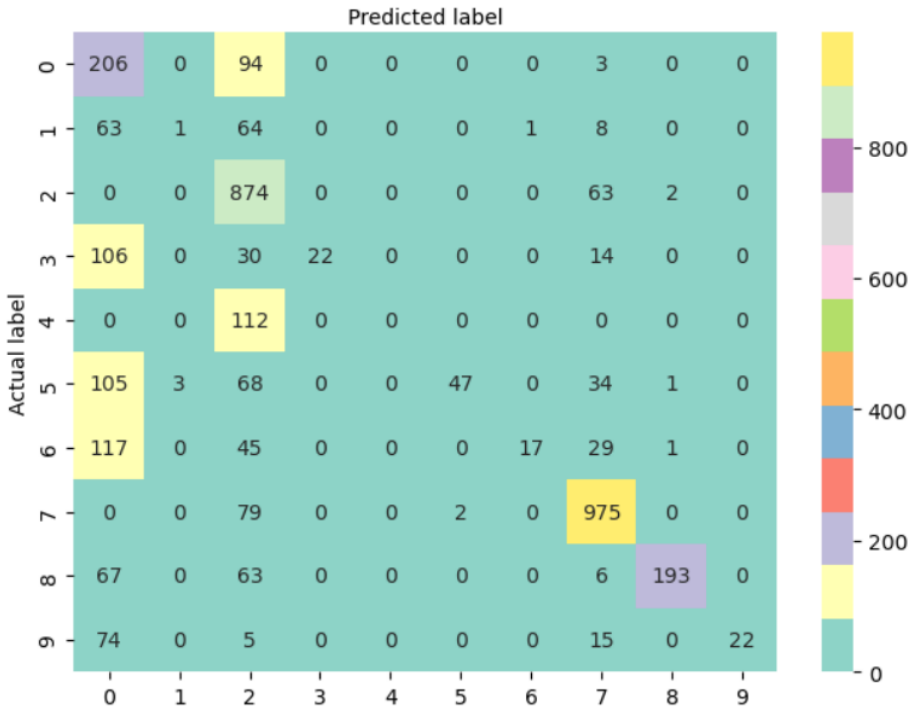
```
print("\n ================================== Evaluation on Multinomial Naive Bayes ==
# Model trained using the "Count Vectors"
NBmod1, NBacc1, NBconfmat1 = train_models(naive_bayes.MultinomialNB(),
                            xtrain_count, xvalid_count,
                            y_train, y_test)
print("Naive Bayes, Count Vectors: ", NBacc1)
classes=[0,1]
def plot_confusion_matrix(confmat, classes):
    fig, ax = plt.subplots()
    ticks = np.arange(len(classes))
    plt.xticks(ticks, classes)
    plt.yticks(ticks, classes)
    sns.heatmap(pd.DataFrame(confmat), annot=True, cmap="Set3", fmt='g')
    ax.xaxis.set_label_position("top")
    plt.tight_layout()
    plt.title('Confusion matrix for the model designed in Naive Bayes', y=1.1)
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')

# Model training and evaluation
def train_and_evaluate_model(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    accuracy = model.score(X_test, y_test)
    y_pred = model.predict(X_test)
    confmat = confusion_matrix(y_test, y_pred)
    return model, accuracy, confmat

# Model 1: Naive Bayes with unigram using "Count Vectors"
NBmod1, NBacc1, NBconfmat1 = train_and_evaluate_model(naive_bayes.MultinomialNB(),
                                                      xtrain_count, xvalid_count,
                                                      y_train, y_test)
print("Naive Bayes, unigram Count Vectors:", NBacc1)
plot_confusion_matrix(NBconfmat1, [0, 1])

# Model 2: Naive Bayes with bigram using "Count Vectors"
NBmod2, NBacc2, NBconfmat2 = train_and_evaluate_model(naive_bayes.MultinomialNB(),
                                                      xtrain_count_freq1, xvalid_count_f
                                                      y_train, y_test)
print("Naive Bayes, bigram Count Vectors:", NBacc2)
plot_confusion_matrix(NBconfmat2, [0, 1])

# Model 3: Naive Bayes with ngram using "Count Vectors"
NBmod3, NBacc3, NBconfmat3 = train_and_evaluate_model(naive_bayes.MultinomialNB(),
                                                      xtrain_count_freq2, xvalid_count_f
                                                      y_train, y_test)
print("Naive Bayes, ngram Count Vectors:", NBacc3)
plot_confusion_matrix(NBconfmat3, [0, 1])

# Model 4: Naive Bayes with Word Level using "TF IDF Vectors"
NBmod4, NBacc4, NBconfmat4 = train_and_evaluate_model(naive_bayes.MultinomialNB(),
                                                      xtrain_tfidf, xvalid_tfidf,
                                                      y_train, y_test)
print("Naive Bayes, WordLevel TF-IDF:", NBacc4)
plot_confusion_matrix(NBconfmat4, [0, 1])

# Model 5: Naive Bayes with bigram Word Level using "TF IDF Vectors"
NBmod5, NBacc5, NBconfmat5 = train_and_evaluate_model(naive_bayes.MultinomialNB(),
                                                      xtrain_tfidf_freq1, xvalid_tfidf_f
                                                      y_train, y_test)
```

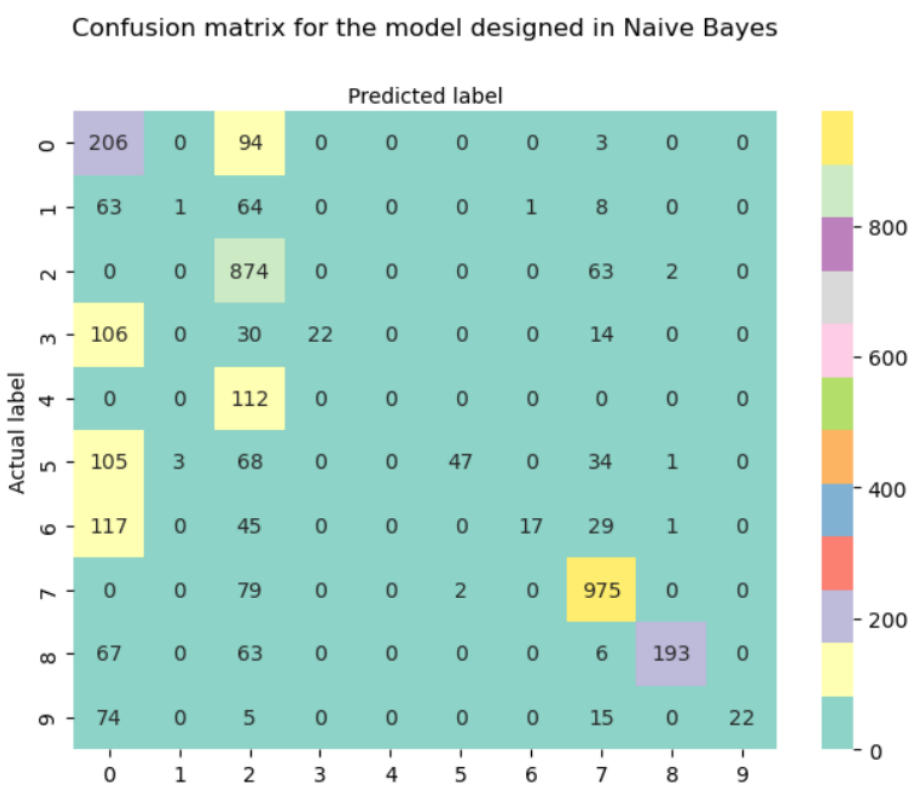
```
print("Naive Bayes, WordLevel bigram TF-IDF:", NBacc5)
plot_confusion_matrix(NBconfmat5, [0, 1])

# Model 6: Naive Bayes with ngram Word Level using "TF IDF Vectors"
NBmod6, NBacc6, NBconfmat6 = train_and_evaluate_model(naive_bayes.MultinomialNB(),
                                                      xtrain_tfidf_freq2, xvalid_tfidf_f
                                                      y_train, y_test)
print("Naive Bayes, WordLevel ngram TF-IDF:", NBacc6)
plot_confusion_matrix(NBconfmat6, [0, 1])

# Model 7: Naive Bayes with Character Level using "TF IDF Vectors"
NBmod7, NBacc7, NBconfmat7 = train_and_evaluate_model(naive_bayes.MultinomialNB(),
                                                      xtrain_tfidf_freq2_str, xvalid_tfi
                                                      y_train, y_test)
print("Naive Bayes, CharLevel ngram TF-IDF:", NBacc7)
plot_confusion_matrix(NBconfmat7, [0, 1])
```

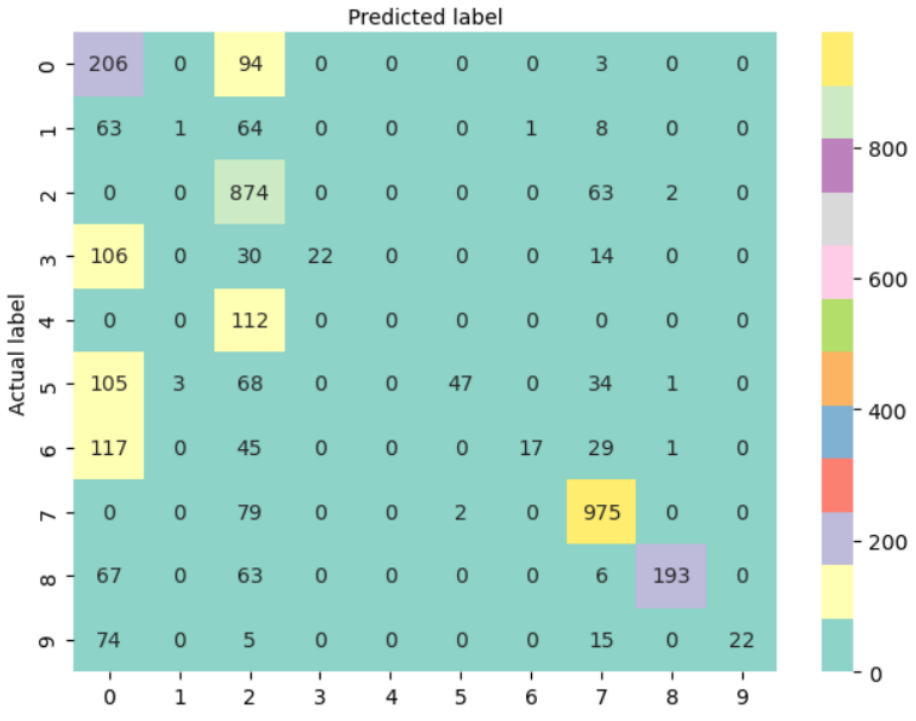
```
 ==================================== Evaluation on Multinomial Naive Ba
yes ====================================

Naive Bayes, Count Vectors:  0.6491324703938309
Naive Bayes, unigram Count Vectors: 0.6491324703938309
Naive Bayes, bigram Count Vectors: 0.6491324703938309
Naive Bayes, ngram Count Vectors: 0.6491324703938309
Naive Bayes, WordLevel TF-IDF: 0.6491324703938309
Naive Bayes, WordLevel bigram TF-IDF: 0.6491324703938309
Naive Bayes, WordLevel ngram TF-IDF: 0.6491324703938309
Naive Bayes, CharLevel ngram TF-IDF: 0.6174607546130543
```



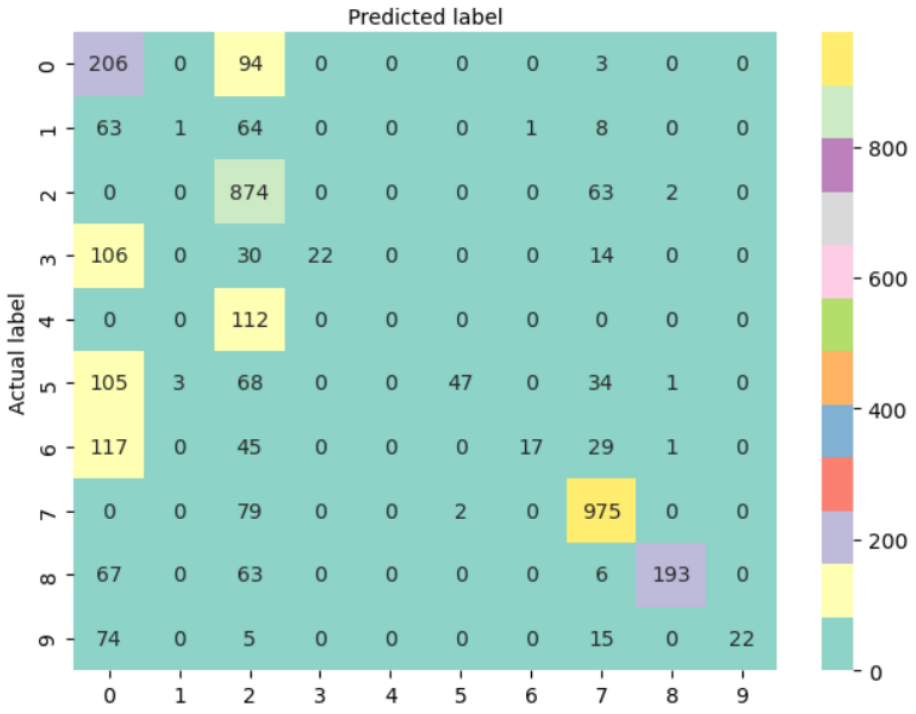

Confusion matrix for the model designed in Naive Bayes
Predicted label
Actual label
206 0 94 0 0 0 0 3 0 0
63 1 64 0 0 0 1 8 0 0
0 0 874 0 0 0 0 63 2 0
106 0 30 22 0 0 0 14 0 0
0 0 112 0 0 0 0 0 0 0
105 3 68 0 0 47 0 34 1 0
117 0 45 0 0 0 17 29 1 0
0 0 79 0 0 2 0 975 0 0
67 0 63 0 0 0 0 6 193 0
74 0 5 0 0 0 0 15 0 22
0 1 2 3 4 5 6 7 8 9
800
600
400
200
0


Confusion matrix for the model designed in Naive Bayes
Predicted label
Actual label
206 0 94 0 0 0 0 3 0 0
63 1 64 0 0 0 1 8 0 0
0 0 874 0 0 0 0 63 2 0
106 0 30 22 0 0 0 14 0 0
0 0 112 0 0 0 0 0 0 0
105 3 68 0 0 47 0 34 1 0
117 0 45 0 0 0 17 29 1 0
0 0 79 0 0 2 0 975 0 0
67 0 63 0 0 0 0 6 193 0
74 0 5 0 0 0 0 15 0 22
0 1 2 3 4 5 6 7 8 9
800
600
400
200
0

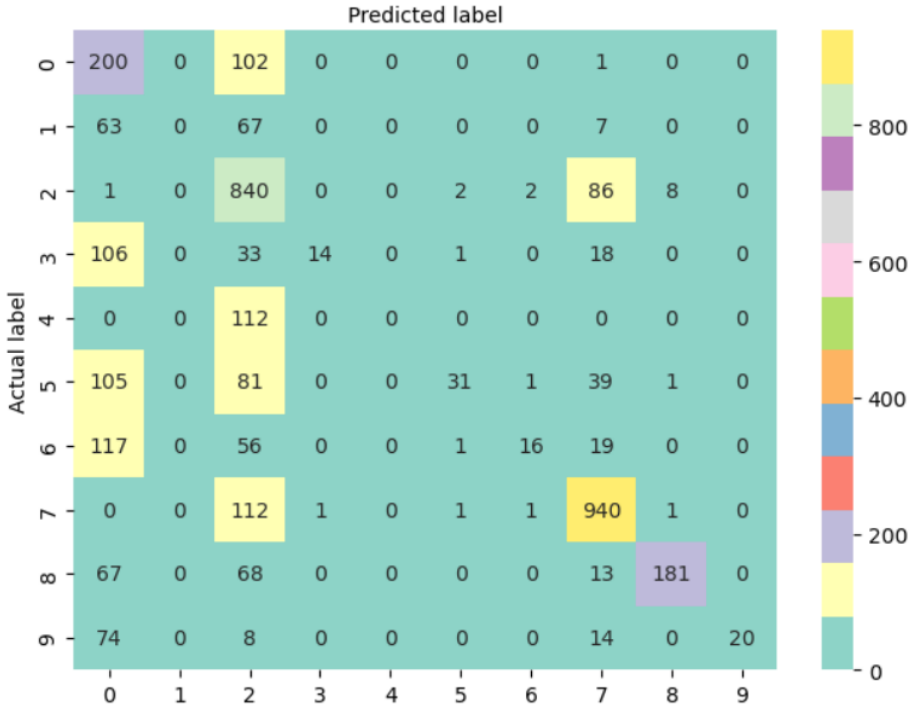

Confusion matrix for the model designed in Naive Bayes
Predicted label
Actual label
206 0 94 0 0 0 0 3 0 0
63 1 64 0 0 0 1 8 0 0
0 0 874 0 0 0 0 63 2 0
106 0 30 22 0 0 0 14 0 0
0 0 112 0 0 0 0 0 0 0
105 3 68 0 0 47 0 34 1 0
117 0 45 0 0 0 17 29 1 0
0 0 79 0 0 2 0 975 0 0
67 0 63 0 0 0 0 6 193 0
74 0 5 0 0 0 0 15 0 22
0 1 2 3 4 5 6 7 8 9
800
600
400
200
0


Confusion matrix for the model designed in Naive Bayes
Predicted label
Actual label
206 0 94 0 0 0 0 3 0 0
63 1 64 0 0 0 1 8 0 0
0 0 874 0 0 0 0 63 2 0
106 0 30 22 0 0 0 14 0 0
0 0 112 0 0 0 0 0 0 0
105 3 68 0 0 47 0 34 1 0
117 0 45 0 0 0 17 29 1 0
0 0 79 0 0 2 0 975 0 0
67 0 63 0 0 0 0 6 193 0
74 0 5 0 0 0 0 15 0 22
0 1 2 3 4 5 6 7 8 9
800
600
400
200
0

Confusion matrix for the model designed in Naive Bayes



Confusion matrix for the model designed in Naive Bayes



7.3 Random Forest Classifier

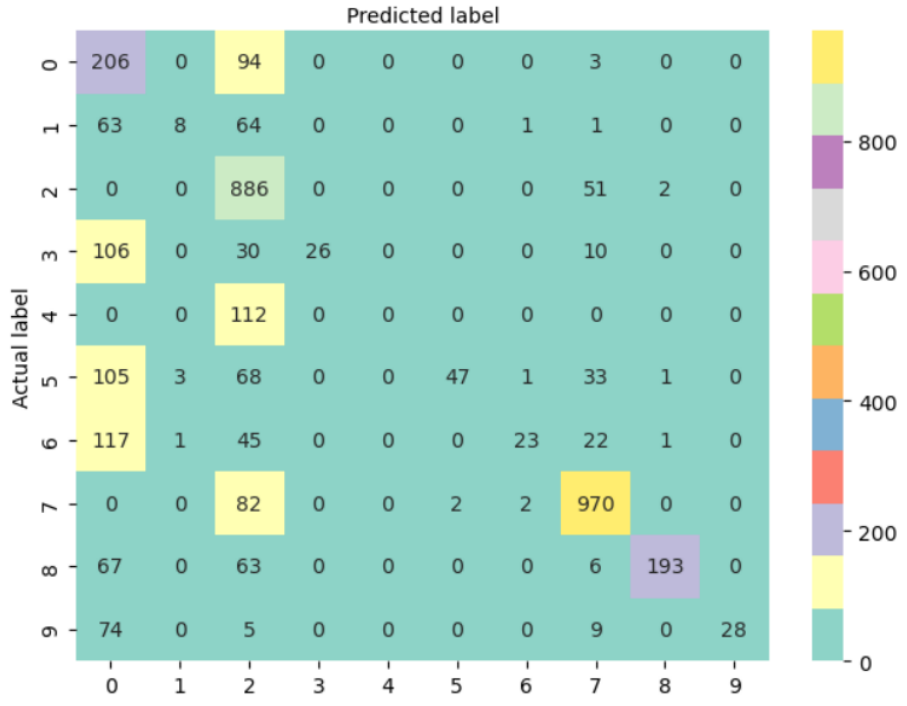
In [29]:

```
print("\n ================================== Random Forest Classifier ==============
# Model trained using the "Count Vectors"
RFCmod1, RFCacc1, RFCconfmat1 = train_models(ensemble.RandomForestClassifier(n_estimator
                            xtrain_count, xvalid_count,
                            y_train, y_test)
print("Random Forest Classifier, Count Vectors: ", RFCacc1)
classes=[0,1]
def plot_confusion_matrix(confmat, classes, model_name):
    fig, ax = plt.subplots()
    ticks = np.arange(len(classes))
    plt.xticks(ticks, classes)
    plt.yticks(ticks, classes)
    sns.heatmap(pd.DataFrame(confmat), annot=True, cmap="Set3", fmt='g')
    ax.xaxis.set_label_position("top")
    plt.tight_layout()
    plt.title(f'Confusion matrix for the model designed in {model_name}', y=1.1)
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')

# Model training and evaluation
def train_and_evaluate_model(model, X_train, X_test, y_train, y_test, model_name):
    model.fit(X_train, y_train)
    accuracy = model.score(X_test, y_test)
    y_pred = model.predict(X_test)
    confmat = confusion_matrix(y_test, y_pred)
    return model, accuracy, confmat

# Model 1: Random Forest Classifier with unigram using "Count Vectors"
RFCmod1, RFCacc1, RFCconfmat1 = train_and_evaluate_model(ensemble.RandomForestClassifier
                                                         xtrain_count, xvalid_count,
                                                         y_train, y_test, "Random Fores
print("Random Forest Classifier, unigram Count Vectors:", RFCacc1)
plot_confusion_matrix(RFCconfmat1, [0, 1], "Random Forest Classifier")

# Model 2: Random Forest Classifier with bigram using "Count Vectors"
RFCmod2, RFCacc2, RFCconfmat2 = train_and_evaluate_model(ensemble.RandomForestClassifier
                                                         xtrain_count_freq1, xvalid_cou
                                                         y_train, y_test, "Random Fores
print("Random Forest Classifier, bigram Count Vectors:", RFCacc2)
plot_confusion_matrix(RFCconfmat2, [0, 1], "Random Forest Classifier")

# Model 3: Random Forest Classifier with ngram using "Count Vectors"
RFCmod3, RFCacc3, RFCconfmat3 = train_and_evaluate_model(ensemble.RandomForestClassifier
                                                         xtrain_count_freq2, xvalid_cou
                                                         y_train, y_test, "Random Fores
print("Random Forest Classifier, ngram Count Vectors:", RFCacc3)
plot_confusion_matrix(RFCconfmat3, [0, 1], "Random Forest Classifier")

# Model 4: Random Forest Classifier with Word Level using "TF IDF Vectors"
RFCmod4, RFCacc4, RFCconfmat4 = train_and_evaluate_model(ensemble.RandomForestClassifier
                                                         xtrain_tfidf, xvalid_tfidf,
                                                         y_train, y_test, "Random Fores
print("Random Forest Classifier, WordLevel TF-IDF:", RFCacc4)
plot_confusion_matrix(RFCconfmat4, [0, 1], "Random Forest Classifier")

# Model 5: Random Forest Classifier with bigram Word Level using "TF IDF Vectors"
RFCmod5, RFCacc5, RFCconfmat5 = train_and_evaluate_model(ensemble.RandomForestClassifier
                                                         xtrain_tfidf_freq1, xvalid_tfi
                                                         y_train, y_test, "Random Fores
```

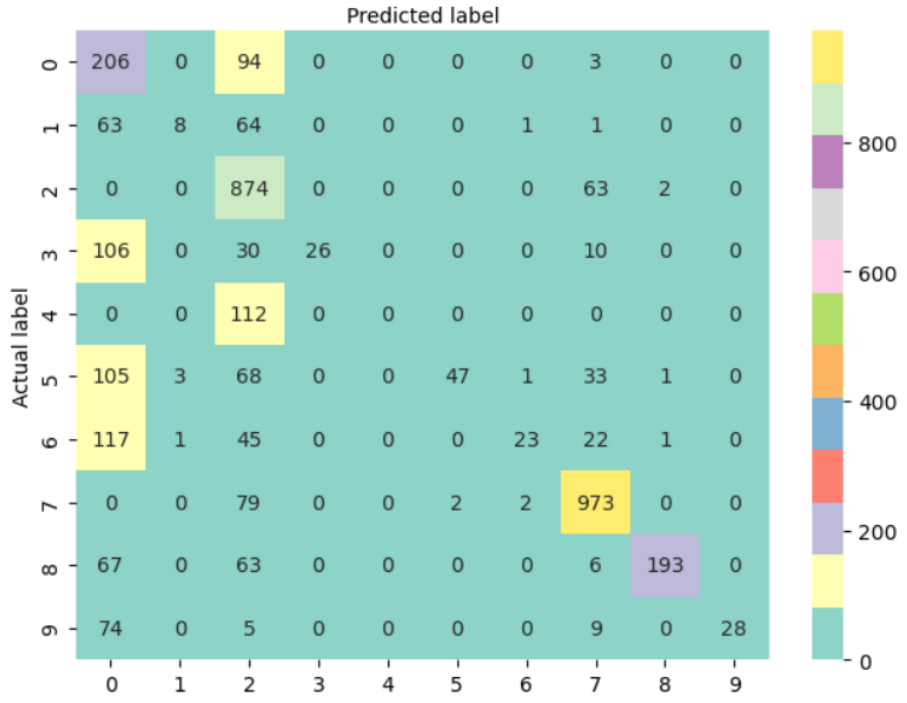
```
print("Random Forest Classifier, WordLevel bigram TF-IDF:", RFCacc5)
plot_confusion_matrix(RFCconfmat5, [0, 1], "Random Forest Classifier")

# Model 6: Random Forest Classifier with ngram Word Level using "TF IDF Vectors"
RFCmod6, RFCacc6, RFCconfmat6 = train_and_evaluate_model(ensemble.RandomForestClassifier
                                                         xtrain_tfidf_freq2, xvalid_tfi
                                                         y_train, y_test, "Random Fores
print("Random Forest Classifier, WordLevel ngram TF-IDF:", RFCacc6)
plot_confusion_matrix(RFCconfmat6, [0, 1], "Random Forest Classifier")

# Model 7: Random Forest Classifier with Character Level using "TF IDF Vectors"
RFCmod7, RFCacc7, RFCconfmat7 = train_and_evaluate_model(ensemble.RandomForestClassifier
                                                         xtrain_tfidf_freq2_str, xvalic
                                                         y_train, y_test, "Random Fores
print("Random Forest Classifier, CharLevel ngram TF-IDF:", RFCacc7)
plot_confusion_matrix(RFCconfmat7, [0, 1], "Random Forest Classifier")
```

```
 ==================================== Random Forest Classifier =========
============================

Random Forest Classifier, Count Vectors:  0.6551914073258056
Random Forest Classifier, unigram Count Vectors: 0.6573946571192509
Random Forest Classifier, bigram Count Vectors: 0.6549160011016248
Random Forest Classifier, ngram Count Vectors: 0.6549160011016248
Random Forest Classifier, WordLevel TF-IDF: 0.6551914073258056
Random Forest Classifier, WordLevel bigram TF-IDF: 0.6576700633434316
Random Forest Classifier, WordLevel ngram TF-IDF: 0.6546405948774442
Random Forest Classifier, CharLevel ngram TF-IDF: 0.6543651886532635
```

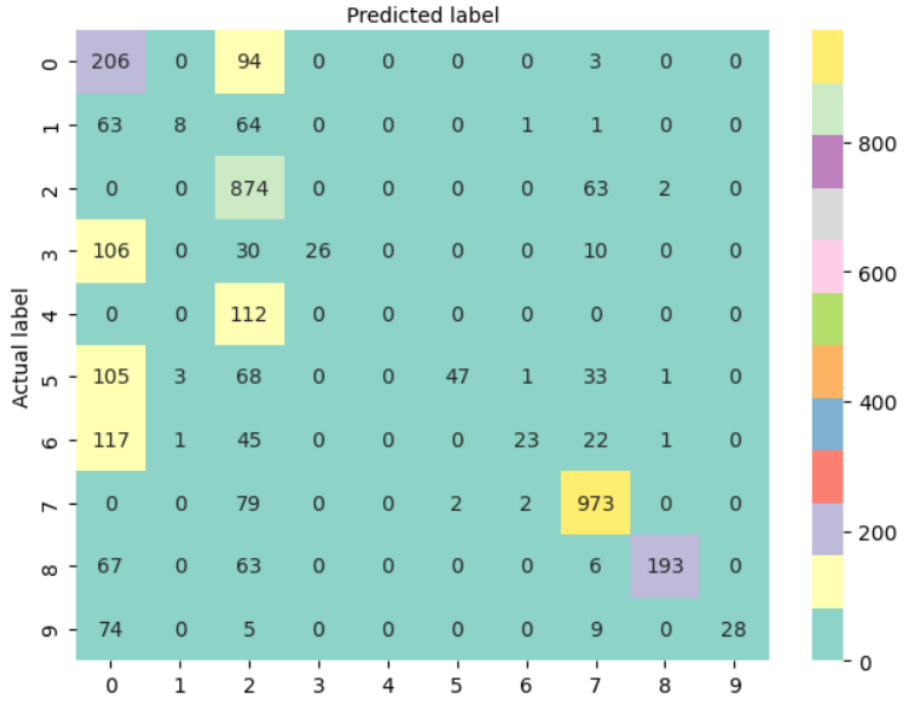
Confusion matrix for the model designed in Random Forest Classifier



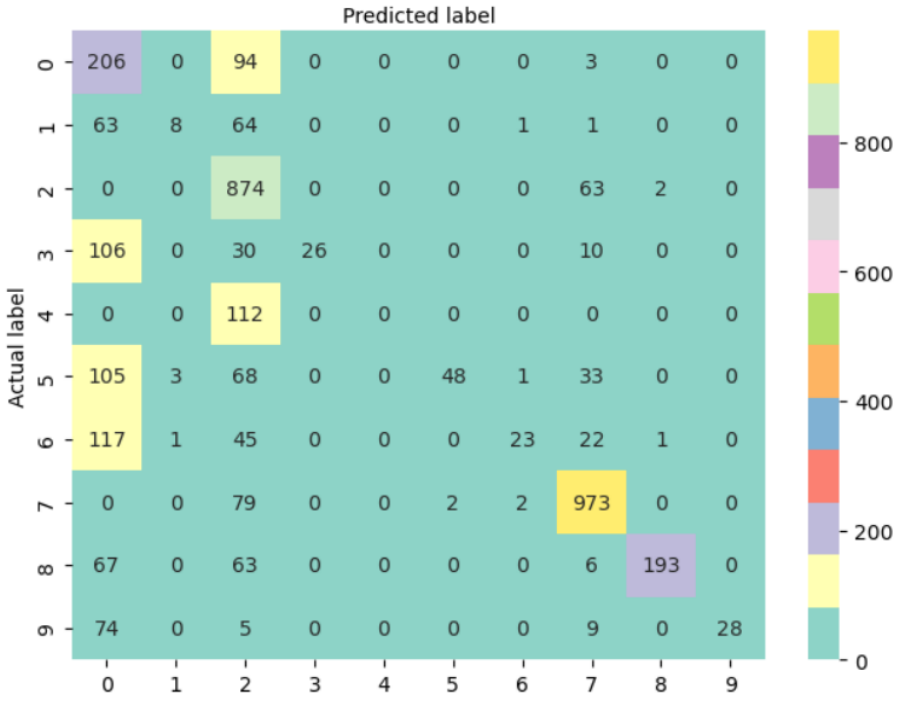
Confusion matrix for the model designed in Random Forest Classifier



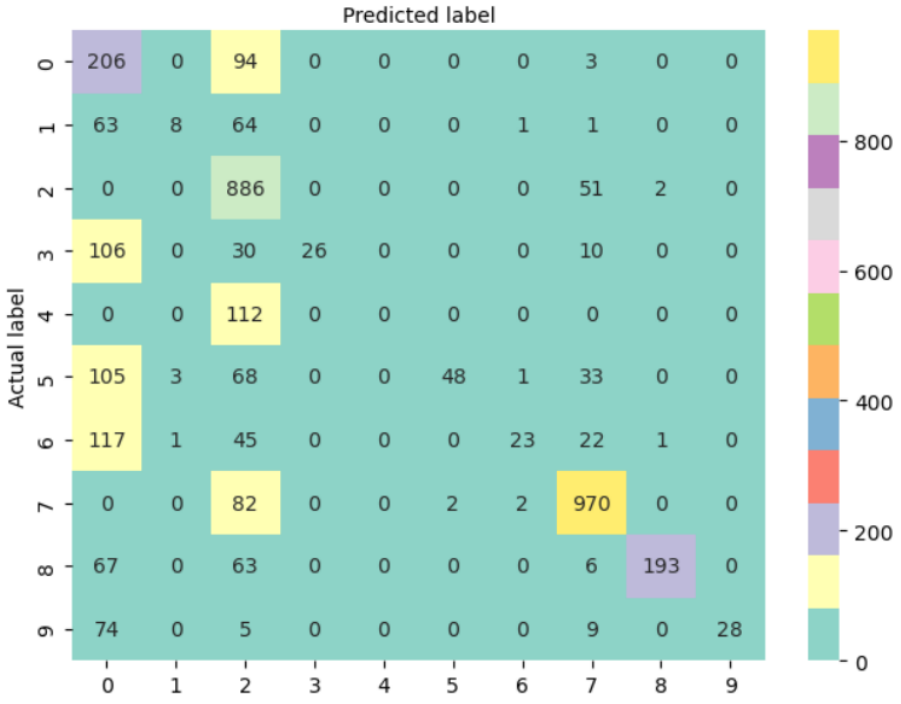
Confusion matrix for the model designed in Random Forest Classifier



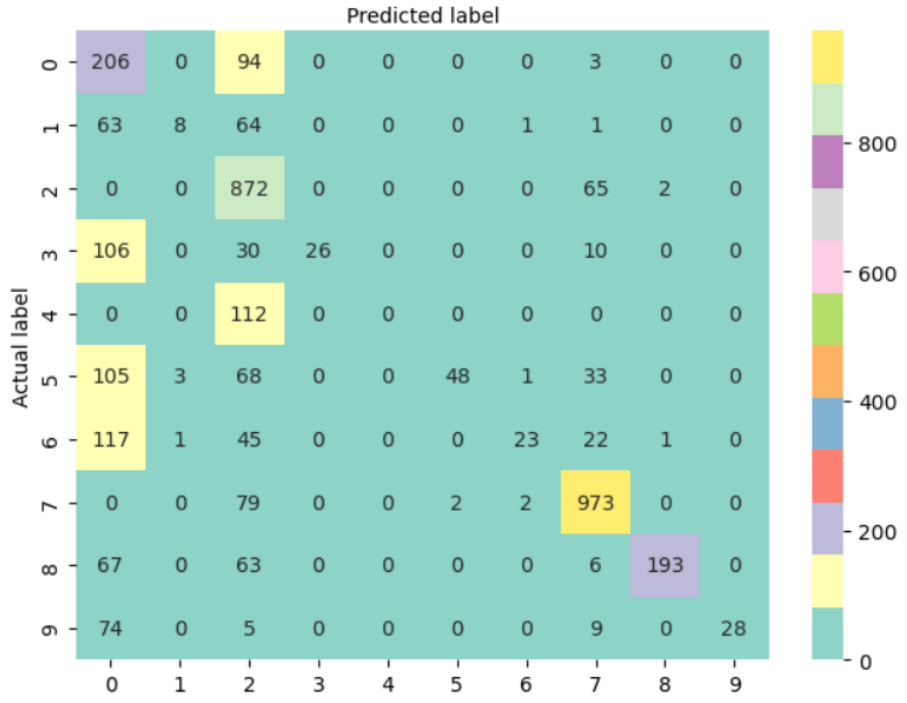
Confusion matrix for the model designed in Random Forest Classifier



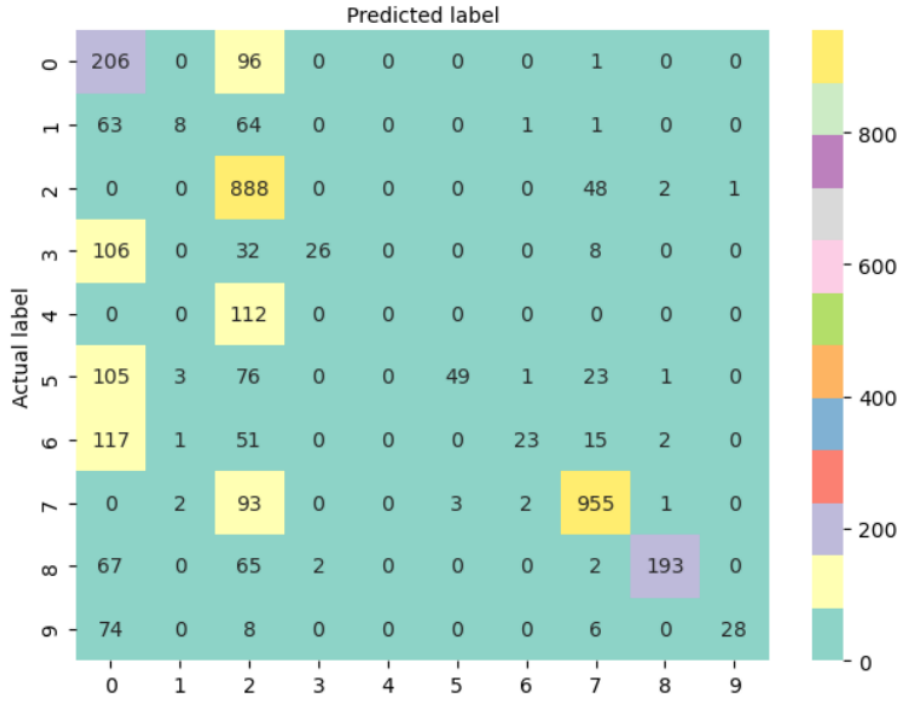
Confusion matrix for the model designed in Random Forest Classifier



Confusion matrix for the model designed in Random Forest Classifier



Confusion matrix for the model designed in Random Forest Classifier



In [30]:

```
dat = pd.read_csv(r'flipkart_com-ecommerce_sample.csv')

sentence = dat.loc[1]['description']

for sent in nltk.sent_tokenize(sentence):
  for chunk in nltk.ne_chunk(nltk.pos_tag(nltk.word_tokenize(sent))):
     if hasattr(chunk, 'label'):
        print(chunk.label(), ' '.join(c[0] for c in chunk))
```

```
ORGANIZATION FabHomeDecor
PERSON Fabric Double Sofa Bed
ORGANIZATION Finish Color
PERSON Leatherette Black Mechanism Type
PERSON Pull Out
PERSON Sofa Bed
PERSON Double
PERSON Sofa Bed
ORGANIZATION FabHomeDecor Fabric Double Sofa Bed
ORGANIZATION Finish Color
PERSON Leatherette Black Mechanism Type
PERSON Pull Out
ORGANIZATION Demo Installation
ORGANIZATION Demo Details Installation
PERSON Brand FabHomeDecor Mattress
PERSON Modern Filling Material
PERSON Shape Square Suitable
PERSON Living Room Model
GPE Avoid
PERSON Keep
PERSON Vacuum
PERSON Avoid
GPE Avoid
PERSON Keep
PERSON Vacuum
PERSON Try
PERSON Type Matte Important Note
PERSON Warranty Covered
GPE Warranty
GPE Service
PERSON Type Manufacturer Warranty Warranty
PERSON Months Domestic Warranty
GPE Warranty
PERSON Improper Handling Dimensions
ORGANIZATION Disclaimer
GPE Please
GPE Please
GPE Please
GPE Flipkart
ORGANIZATION Seller
PERSON Color Upholstery Color Leatherette Black
PERSON Fabric Secondary Material Subtype Mango Wood
```

9. Model Comparison

In [31]:

```
NB = {
    'Model Name': ['Model trained using the Count Vectors', 'Model trained on the bigram
                   'Model trained on the ngram using the Count Vectors', 'Model trained
                   'Model trained on the Word Level using the bigram TF IDF Vectors', 'M
        'Accuracy Scores': [NBacc1, NBacc2, NBacc3,
                            NBacc4, NBacc5, NBacc6, NBacc7]
}
```

In [32]:

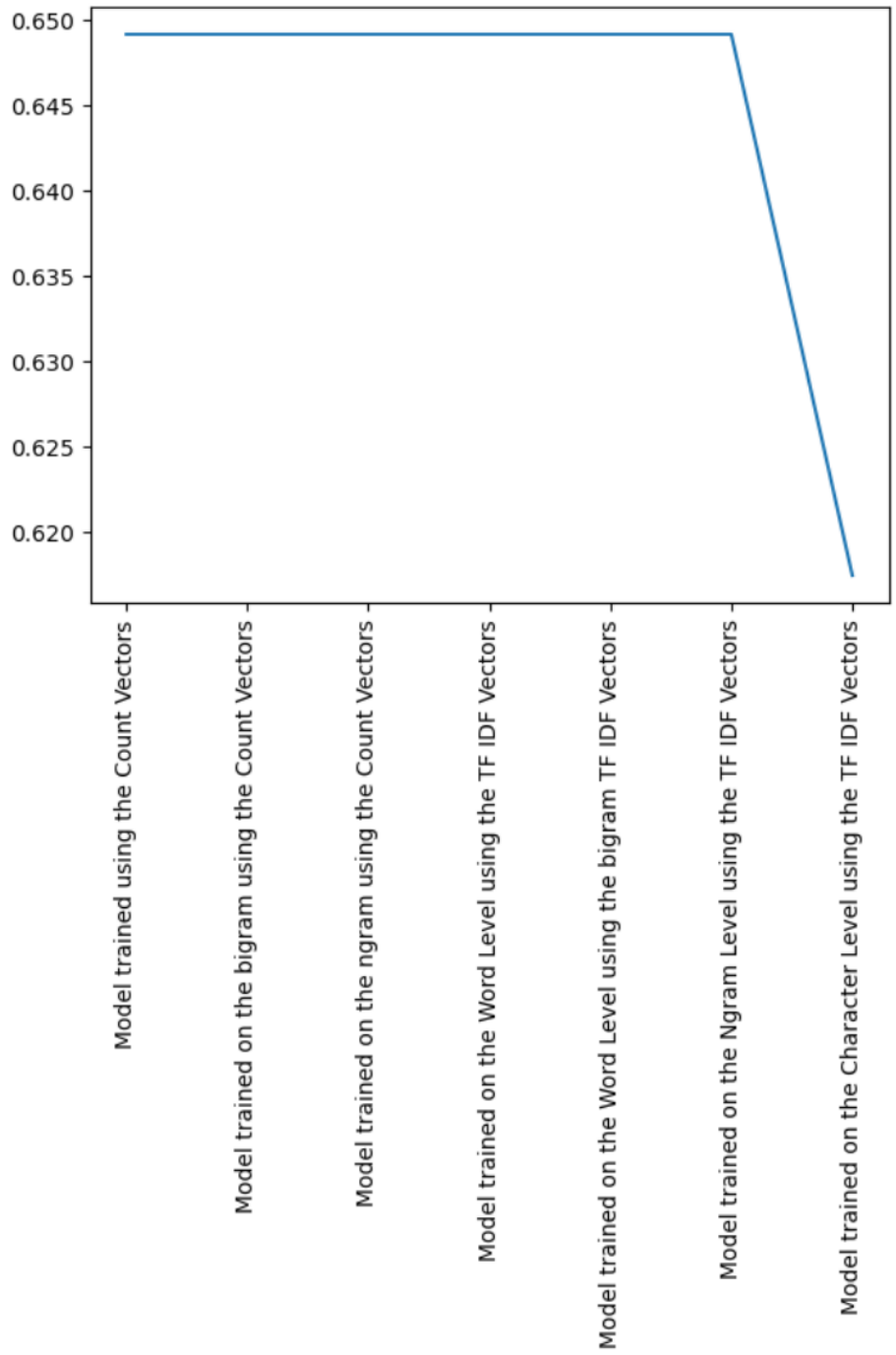
```
Model_In_NB = pd.DataFrame(NB)
Model_In_NB
```

Out[32]:

	Model Name	Accuracy Scores
0	Model trained using the Count Vectors	0.649132
1	Model trained on the bigram using the Count Ve...	0.649132
2	Model trained on the ngram using the Count Vec...	0.649132
3	Model trained on the Word Level using the TF I...	0.649132
4	Model trained on the Word Level using the bigr...	0.649132
5	Model trained on the Ngram Level using the TF ...	0.649132
6	Model trained on the Character Level using the...	0.617461

In [33]:

```
plt.plot(Model_In_NB['Model Name'], Model_In_NB['Accuracy Scores'])
plt.xticks(rotation='vertical')
plt.show()
```



In [34]:

```
RFC = {
    'Model Name': ['Model trained using the Count Vectors', 'Model trained on the bigram
                   'Model trained on the ngram using the Count Vectors', 'Model trained
                   'Model trained on the Word Level using the bigram TF IDF Vectors', 'M
        'Accuracy Scores': [RFCacc1, RFCacc2, RFCacc3,
                            RFCacc4, RFCacc5, RFCacc6, RFCacc7]
}
```

In [35]:

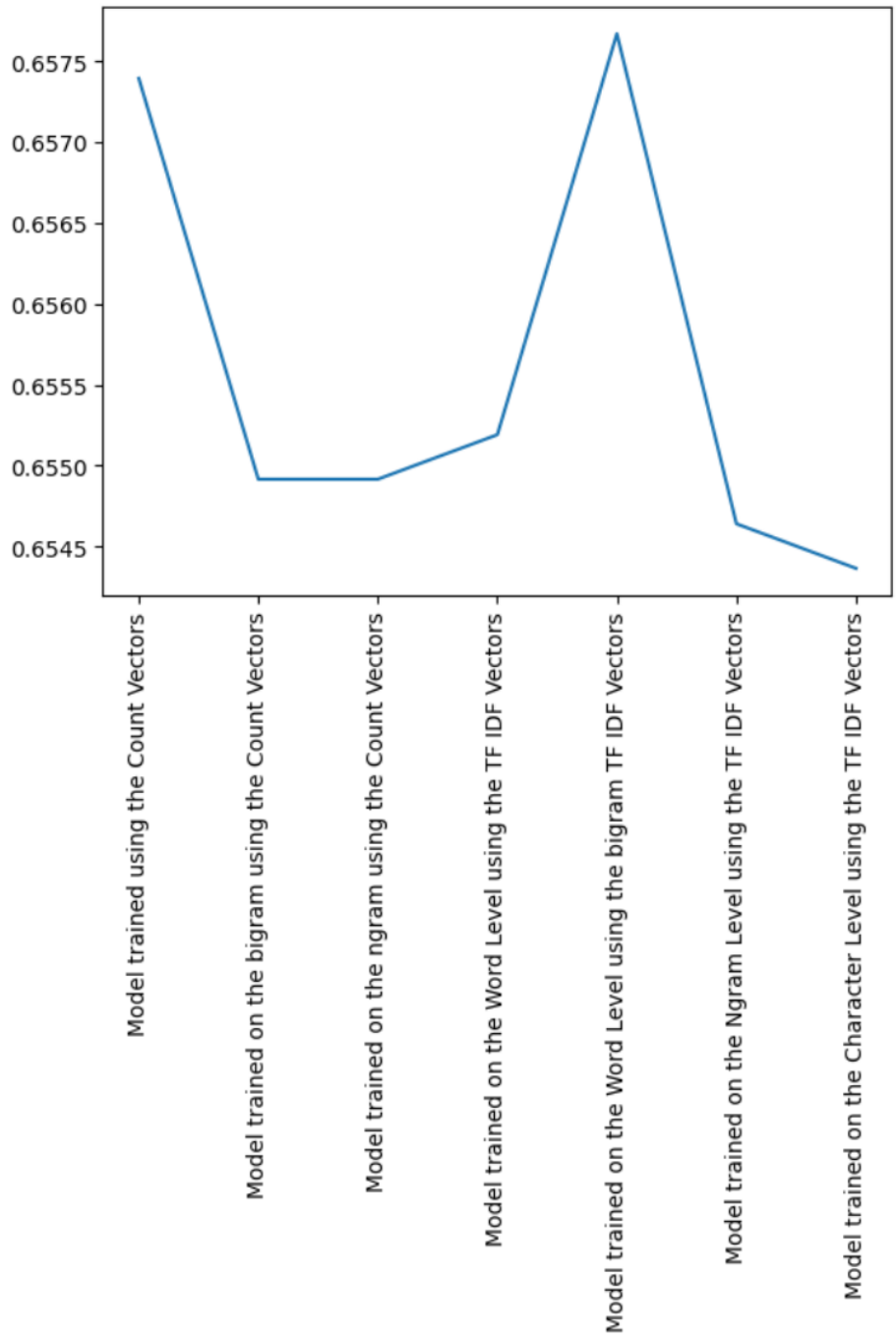
```
Model_In_RFC = pd.DataFrame(RFC)
Model_In_RFC
```

Out[35]:

	Model Name	Accuracy Scores
0	Model trained using the Count Vectors	0.657395
1	Model trained on the bigram using the Count Ve...	0.654916
2	Model trained on the ngram using the Count Vec...	0.654916
3	Model trained on the Word Level using the TF I...	0.655191
4	Model trained on the Word Level using the bigr...	0.657670
5	Model trained on the Ngram Level using the TF ...	0.654641
6	Model trained on the Character Level using the...	0.654365

In [36]:

```
plt.plot(Model_In_RFC['Model Name'], Model_In_RFC['Accuracy Scores'])
plt.xticks(rotation='vertical')
plt.show()
```



In [ ]:

Mahesh Code

ORIGINALITY REPORT

23%	19%	15%	17%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	rstudio-pubs-static.s3.amazonaws.com Internet Source	3%
2	Akshay Kulkarni, Adarsha Shivananda, Anoosh Kulkarni. "Natural Language Processing Projects", Springer Science and Business Media LLC, 2022 Publication	2%
3	Submitted to BPP College of Professional Studies Limited Student Paper	2%
4	github.com Internet Source	2%
5	www.analyticsvidhya.com Internet Source	2%
6	Submitted to University of Sunderland Student Paper	1%
7	Submitted to Rutgers University, New Brunswick Student Paper	1%
	gitlab.sliit.lk	

8	Internet Source	1 %
9	Submitted to University of Surrey Student Paper	1 %
10	Submitted to University of Lancaster Student Paper	1 %
11	ebin.pub Internet Source	1 %
12	Submitted to University of Liverpool Student Paper	1 %
13	dev.to Internet Source	1 %
14	Submitted to AUT University Student Paper	1 %
15	Submitted to Institute of Technology Carlow Student Paper	1 %
16	Submitted to UC, Irvine Student Paper	<1 %
17	Submitted to University of the Pacific Student Paper	<1 %
18	Submitted to University of Wales Institute, Cardiff Student Paper	<1 %
19	keras.io Internet Source	

		<1%
20	Submitted to Nanyang Polytechnic Student Paper	<1%
21	Submitted to University of Western Ontario Student Paper	<1%
22	towardsdatascience.com Internet Source	<1%
23	blog.csdn.net Internet Source	<1%
24	www.coursehero.com Internet Source	<1%
25	smcse.city.ac.uk Internet Source	<1%
26	www.fatalerrors.org Internet Source	<1%
27	www.igi-global.com Internet Source	<1%
28	Submitted to Cornell University Student Paper	<1%

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		