TOPIC: CONSUMER COMPLAINT CLASSICATION

Brief Background

There is so much we can learn from our customers' complaints. Understanding your customers is important and all, but it doesn't change the fact that a wall of text from an angry customer is not only strenuous to read through but also demotivating. Furthermore, the complaints can be filed into the wrong sections by the consumers themselves, and it is going to be a huge pain to go through every single complaint and direct them to the relevant departments to be dealt with.

Purpose

The project utilizes artificial intelligence algorithms to process bag of words with classified products, learn with the best methodologies (both Machine Learning or Deep Learning), select the best methodology with the best accuracy and precision to predict future complaint products.

Outline of the Focus of the Analysis

Packages for Analysis

In [1]:
```
import numpy as np
import pandas as pd
seed = 69
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import display
import re
import string
import nltk
from textblob import Word
from gensim.models import Word2Vec, word2vec
import gensim
from sklearn import preprocessing, metrics
from sklearn.metrics import accuracy_score
from sklearn.feature_extraction.text import TfidfVectorizer
import sklearn.feature_extraction.text as text
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.model_selection import cross_val_score, KFold, StratifiedKFold
from sklearn.model_selection import train_test_split
```

In [2]:
```
import warnings
warnings.filterwarnings("ignore")
```

Data Source

In [4]:
```
data_link = 'https://www.consumerfinance.gov/data-research/consumer-complaints/search/api/v1/?company_received_ma
Dataset = pd.read_csv(data_link)
Dataset.head(2)
```

Out[4]:

	Date received	Product	Sub-product	Issue	Sub-issue	Consumer complaint narrative	Company public response	Company	State	ZIP code	Tags	Consumer consent provided?	Submitted via
0	05/30/18	Debt collection	Other debt	Attempts to collect debt not owed	Debt is not yours	I never signed any paperwork with this company...	None	VIRTUOSO SOURCING GROUP, LLC.	FL	320XX	Servicemember	Consent provided	Web
1	09/06/19	Credit reporting, credit repair services, or o...	Credit reporting	Problem with a credit reporting company's inve...	Their investigation did not fix an error on yo...	I submitted a Second written request in writin...	Company has responded to the consumer and the ...	Experian Information Solutions Inc.	TX	774XX	None	Consent provided	Web

Capitalizing Header for Clarification

In [5]:
```
Dataset.rename(columns={'Date received':'DATE_RECEIVED',
                        'Product':'PRODUCT',
                        'Sub-product':'SUB_PRODUCT',
                        'Issue':'ISSUE',
                        'Sub-issue':'SUB_ISSUE',
                        'Consumer complaint narrative':'CONSUMER_COMPLAINT',
                        'Company public response':'COMPANY_PUBLIC_RESPONSE',
                         'Company':'COMPANY',
                         'State':'STATE',
                         'ZIP code':'ZIP_CODE',
                         'Tags':'TAGS',
                         'Consumer consent provided?':'CONSUMER_CONSENT_PROVIDED',
                         'Subbmited via':'SUBMITTED_VIA',
                         'Date sent to company':'DATE_SENT_TO_COMPANY',
                         'Company response to consumer':'COMPANY_RESPONSE_TO_CONSUMER',
                         'Timely response?':'TIMELY_RESPONSE',
                         'Consumer disputed?':'CONSUMER_DISPUTED',
                         'Complaint ID':'COMPLAINT_ID'
                        }, inplace=True)
```

Key Descriptive Statistics with interpretation

In [6]:
```
Dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 683895 entries, 0 to 683894
Data columns (total 18 columns):
 #   Column                         Non-Null Count   Dtype 
---  ------                         --------------   ----- 
 0   DATE_RECEIVED                  683895 non-null  object
 1   PRODUCT                        683895 non-null  object
 2   SUB_PRODUCT                    683895 non-null  object
 3   ISSUE                          683895 non-null  object
 4   SUB_ISSUE                      683895 non-null  object
 5   CONSUMER_COMPLAINT             683895 non-null  object
 6   COMPANY_PUBLIC_RESPONSE        683895 non-null  object
 7   COMPANY                        683895 non-null  object
 8   STATE                          683895 non-null  object
 9   ZIP_CODE                       683895 non-null  object
 10  TAGS                           683895 non-null  object
 11  CONSUMER_CONSENT_PROVIDED      683895 non-null  object
 12  Submitted via                  683895 non-null  object
 13  DATE_SENT_TO_COMPANY           683895 non-null  object
 14  COMPANY_RESPONSE_TO_CONSUMER   683895 non-null  object
 15  TIMELY_RESPONSE                683895 non-null  object
 16  CONSUMER_DISPUTED              47624 non-null   object
 17  COMPLAINT_ID                   683895 non-null  int64 
dtypes: int64(1), object(17)
memory usage: 93.9+ MB
```

In [7]:
```
Dataset.dtypes
```

Out[7]:
```
DATE_RECEIVED                  object
PRODUCT                        object
SUB_PRODUCT                    object
ISSUE                          object
SUB_ISSUE                      object
CONSUMER_COMPLAINT             object
COMPANY_PUBLIC_RESPONSE        object
COMPANY                        object
STATE                          object
ZIP_CODE                       object
TAGS                           object
CONSUMER_CONSENT_PROVIDED      object
Submitted via                  object
DATE_SENT_TO_COMPANY           object
COMPANY_RESPONSE_TO_CONSUMER   object
TIMELY_RESPONSE                object
CONSUMER_DISPUTED              object
COMPLAINT_ID                    int64
dtype: object
```

Checking for missing varaibles

In [8]:
```
Dataset.isnull().sum()
```

Out[8]:
```
DATE_RECEIVED                      0
```

```
PRODUCT                            0
SUB_PRODUCT                        0
ISSUE                              0
SUB_ISSUE                          0
CONSUMER_COMPLAINT                 0
COMPANY_PUBLIC_RESPONSE            0
COMPANY                            0
STATE                              0
ZIP_CODE                           0
TAGS                               0
CONSUMER_CONSENT_PROVIDED          0
Submitted via                      0
DATE_SENT_TO_COMPANY               0
COMPANY_RESPONSE_TO_CONSUMER       0
TIMELY_RESPONSE                    0
CONSUMER_DISPUTED             636271
COMPLAINT_ID                       0
dtype: int64
```

In [9]:
```
Dataset.describe()
```

Out[9]:

	COMPLAINT_ID
count	6.838950e+05
mean	3.407842e+06
std	6.490246e+05
min	2.162970e+06
25%	2.880638e+06
50%	3.388102e+06
75%	3.905162e+06
max	4.754693e+06

Drop insignificant columns without complaints

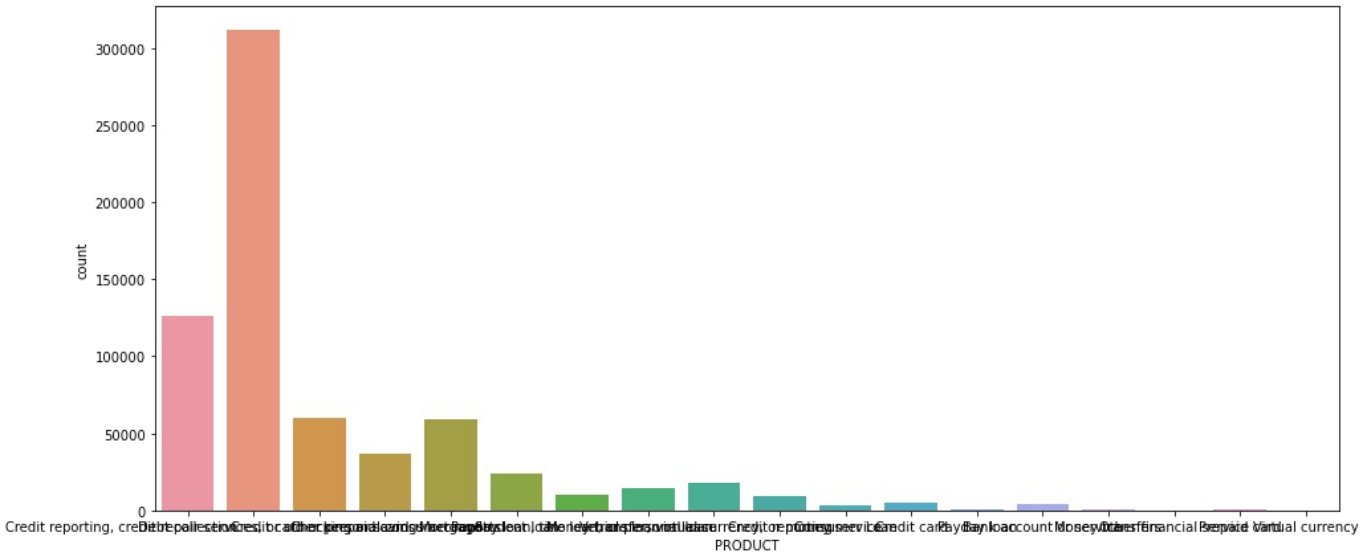
In [10]:
```
Dataset.dropna(axis=0, subset=['CONSUMER_COMPLAINT'],
               inplace=True)
```

Graphical analysis with interpretation

Products graphical analysis

In [11]:
```
fig,ax = plt.subplots(figsize=(16,7))
sns.countplot(x='PRODUCT',data=Dataset)
```

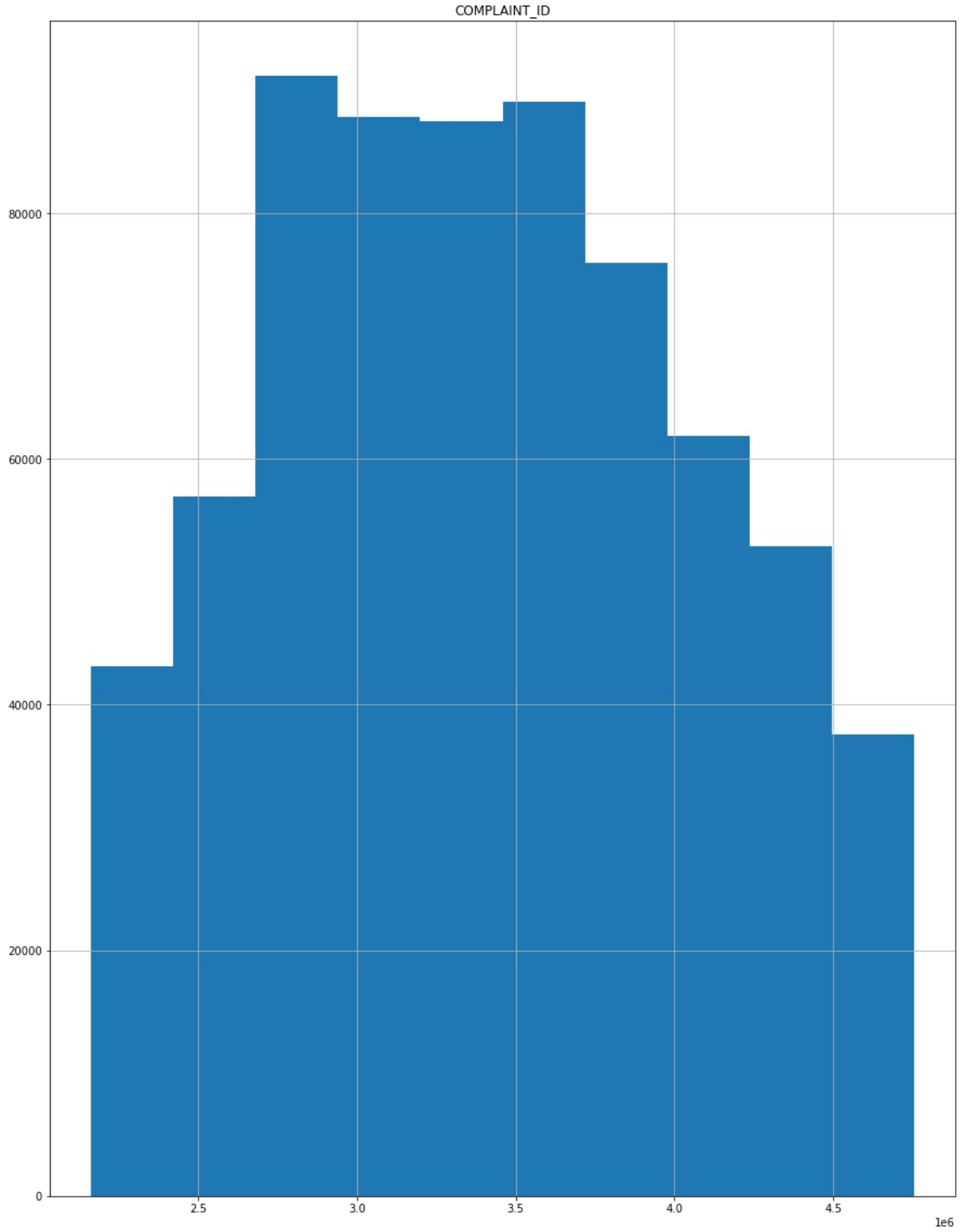
Out[11]:
```
<AxesSubplot:xlabel='PRODUCT', ylabel='count'>
```



Data visualization in histogram

In [12]:
```
Dataset.hist(figsize = (15,20))
```

Out[12]:
```
array([[<AxesSubplot:title={'center':'COMPLAINT_ID'}>]], dtype=object)
```



Word Cloud for Products in the Dataset

In [13]:
```
for product_name in Dataset['PRODUCT'].unique():
    print(product_name)
    all_words = ' '.join([text for text in Dataset.loc[Dataset['PRODUCT'].str.contains(product_name),'CONSUMER_C
```

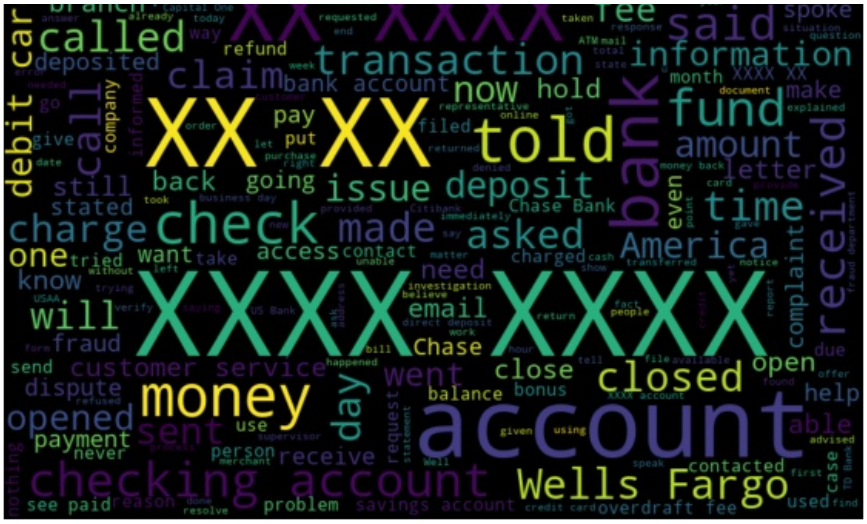
```
    from wordcloud import WordCloud
    wordcloud = WordCloud(width=800, height=500, random_state=21, max_font_size=110).generate(all_words)

    plt.figure(figsize=(10, 7))
    plt.imshow(wordcloud, interpolation="bilinear")
    plt.axis('off')
    plt.show()
```

Debt collection

Credit reporting, credit repair services, or other personal consumer reports

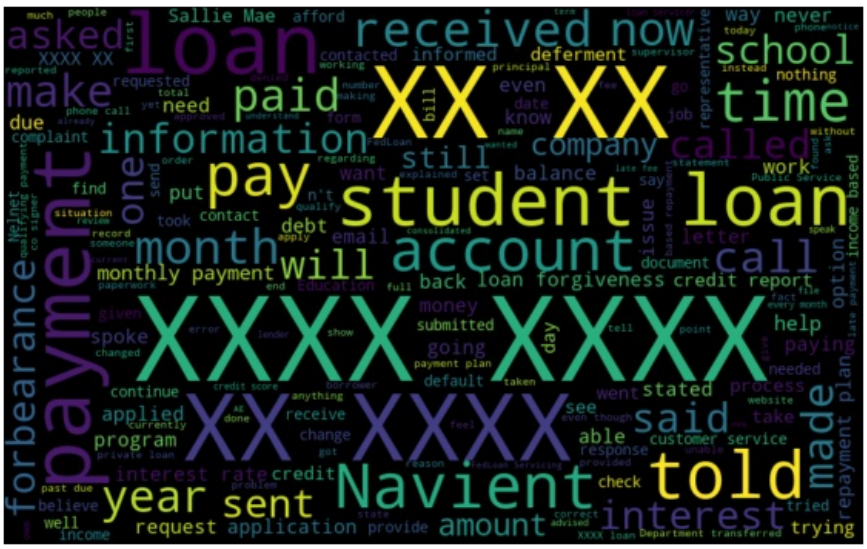
Credit card or prepaid card

Checking or savings account

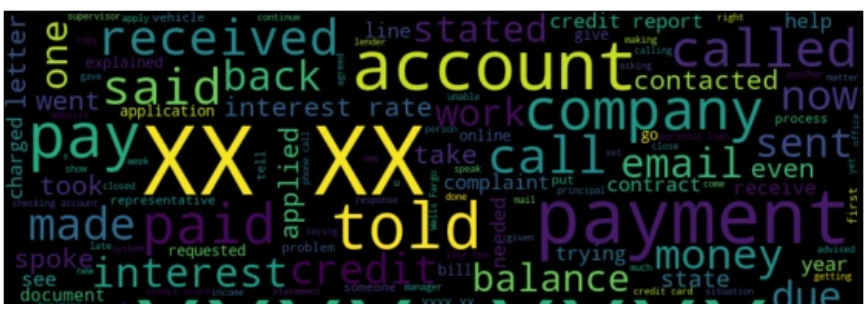


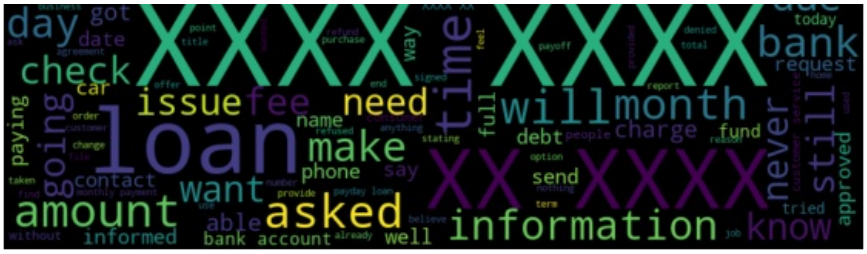
Mortgage

Student loan

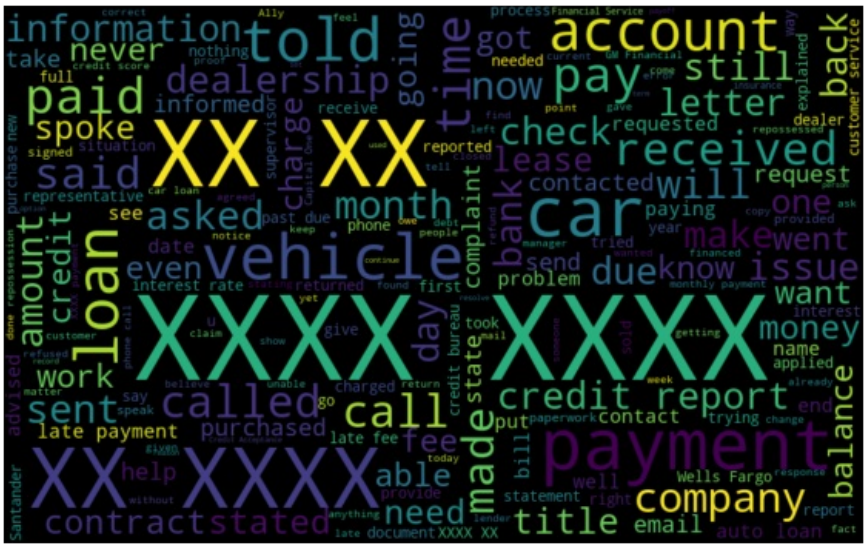


Payday loan, title loan, or personal loan

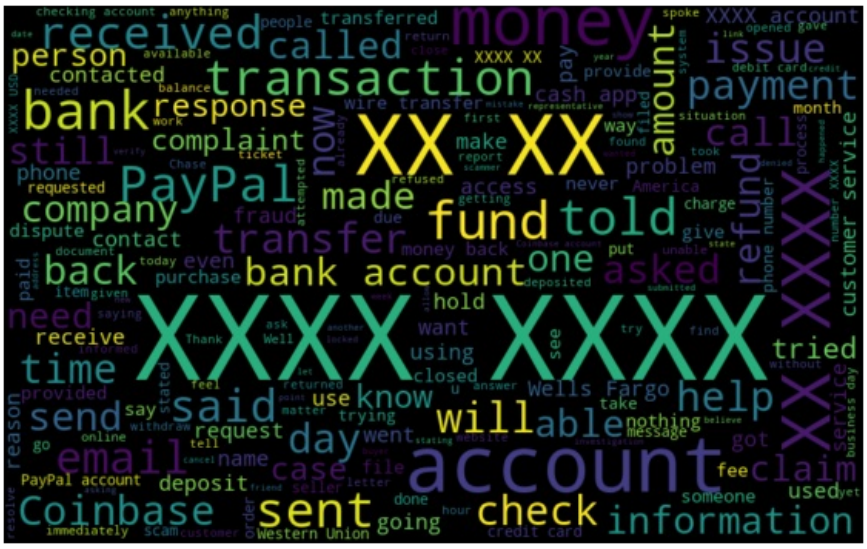




Vehicle loan or lease



Money transfer, virtual currency, or money service



Credit reporting

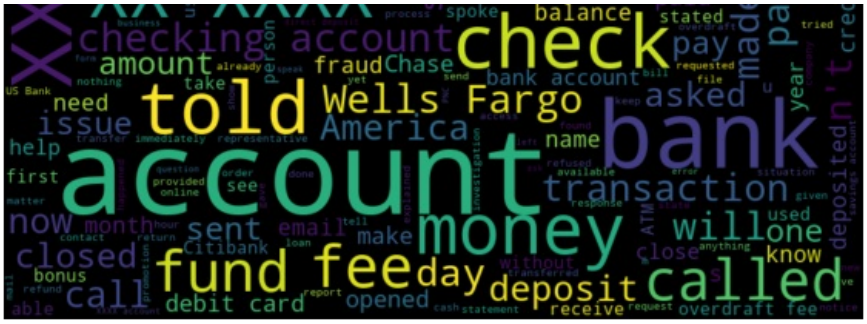
Consumer Loan

Consumer Loan

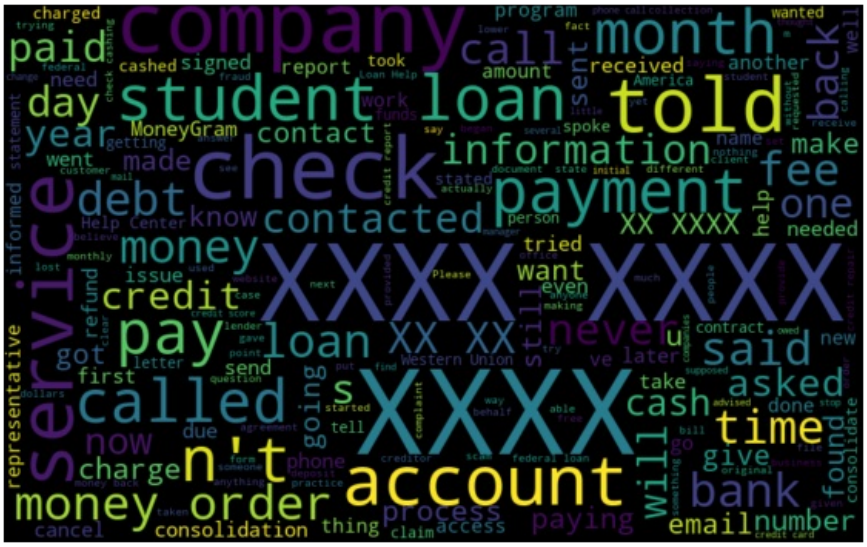
Credit card

Payday loan

Bank account or service


account
bank
told
check
money
fund
fee
called
transaction
Wells Fargo
America
checking
deposit
day
closed
issue
amount
fraud
Chase
bank account
pay
asked
name
transfer
immediately
representative
Citibank
bonus
call
debit card
opened
receive
overdraft fee
statement
balance
stated
now
sent
email
make
close
know
one
will
used
deposited
credit card
first
provided
online
see
order
refund
report
cash
Money transfers


account
transfer
said
made
Western Union
received
will
never
credit card
receive
contacted
told
asked
eBay
money back
seller
XXXX
XXXX
buyer
dispute
one
used
time
went
item
information
call
make
service
customer service
complaint
money transfer
fee
even
Wire
back
bank
payment
paid
refund
company
transferred
online
pay
requested
email
name
fund
provide
wire transfer
claim
send
still
transaction
called
day
fraud
bank account
help
MoneyGram
order
phone
Other financial service


company
month
paid
call
told
student
loan
day
year
MoneyGram
contact
information
check
fee
debt
payment
contacted
XX XXXX
money
tried
credit
pay
loan XX XX
said
called
asked
cash
time
now
n't
account
bank
money order
process
paying
email
number
consolidation
thing
claim
access
representative
service
Prepaid card


said
charge
told
called
card
transaction
fee
customer service
account
received
issue
funds
use
gift card
money
XXXX
XXXX
American Express
back
time
day
dispute
balance
check
purchase
prepaid card
Netspend
made
company
call
now
went
credit
even
complaint
refund
sent
claim
send
take
process
email
information
number
contact
paid
closed
never
tried
need
pay
Bank
still
returned
case
purchased
receipt
nothing

```
Virtual currency
```

Data Processing

Sorting columns relevant for our analysis

The two relevant variables in our dataset are Product and Consumer_Complaint. Hence, we would sort out these columns for our analysis.

In [14]:
```
Dataset_PC = Dataset[['PRODUCT', 'CONSUMER_COMPLAINT']]
```

In [15]:
```
Dataset_PC.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 683895 entries, 0 to 683894
Data columns (total 2 columns):
 #   Column              Non-Null Count   Dtype 
---  ------              --------------   ----- 
 0   PRODUCT             683895 non-null  object
 1   CONSUMER_COMPLAINT  683895 non-null  object
dtypes: object(2)
memory usage: 35.7+ MB
```

In [16]:
```
Dataset_PC.head(4)
```

Out[16]:

	PRODUCT	CONSUMER_COMPLAINT
0	Debt collection	I never signed any paperwork with this company...
1	Credit reporting, credit repair services, or o...	I submitted a Second written request in writin...
2	Credit reporting, credit repair services, or o...	Creditors that should have gone off my report ...
3	Credit reporting, credit repair services, or o...	Two accounts are still on my credit history af...

Converting words to Lowercase

In [17]:
```
Dataset_PC['CONSUMER_COMPLAINT'] =Dataset_PC['CONSUMER_COMPLAINT'].apply(lambda x: ' '.join([i.lower() for i in x
```

Sample Test

In [18]:
```
Dataset_PC['CONSUMER_COMPLAINT'].sample(2)
```

Out[18]:
```
559784    phh in 2013 did not communicate they had not r...
646894    i am no longer associated with these addresses...
Name: CONSUMER_COMPLAINT, dtype: object
```

Removing Text Punctuations

In [19]:
```
Dataset_PC['CONSUMER_COMPLAINT'] =Dataset_PC['CONSUMER_COMPLAINT'].str.replace(r'[^\w\s]',"")
```

Sample Test

In [20]:
```
Dataset_PC['CONSUMER_COMPLAINT'].sample(2)
```

Out[20]:
```
669259    i was approved for a 1100000 personal loan wit...
24191     i obtained netspend prepaid visa card and afte...
Name: CONSUMER_COMPLAINT, dtype: object
```

Stopwords Removal

In [21]:
```
from nltk.corpus import stopwords
stop = stopwords.words('english')
```

In [22]:
```
Dataset_PC['CONSUMER_COMPLAINT'] =Dataset_PC['CONSUMER_COMPLAINT'].apply(lambda x: ' '.join([i for i in x.split()
```

Sample Test

In [23]:
```
Dataset_PC['CONSUMER_COMPLAINT'].head(1)
```

Out[23]:
```
0    never signed paperwork company stating promise...
Name: CONSUMER_COMPLAINT, dtype: object
```

Lemmatizing

In [24]:
```
Dataset_PC['CONSUMER_COMPLAINT'] =Dataset_PC['CONSUMER_COMPLAINT'].apply(lambda x:' '.join([Word(i).lemmatize() f
```

Sample Test

In [25]:
```
Dataset_PC.CONSUMER_COMPLAINT.head(1)
```

Out[25]:
```
0    never signed paperwork company stating promise...
Name: CONSUMER_COMPLAINT, dtype: object
```

Checking list of multiple Products in our data

In [26]:
```
print('--------------')
print('Categories in PRODUCT column:')
print('--------------\n')
print(Dataset_PC['PRODUCT'].unique(), '\n')
print('--------------')
print('# of unique categories: ', Dataset_PC['PRODUCT'].nunique())
print('--------------')
```

```
--------------
Categories in PRODUCT column:
--------------

['Debt collection'
 'Credit reporting, credit repair services, or other personal consumer reports'
 'Credit card or prepaid card' 'Checking or savings account' 'Mortgage'
 'Student loan' 'Payday loan, title loan, or personal loan'
 'Vehicle loan or lease'
 'Money transfer, virtual currency, or money service' 'Credit reporting'
 'Consumer Loan' 'Credit card' 'Payday loan' 'Bank account or service'
 'Money transfers' 'Other financial service' 'Prepaid card'
 'Virtual currency']
```

```
--------------
# of unique categories:  18
--------------
```

Value counts of all products

In [27]:
```
Dataset_PC.PRODUCT.value_counts()
```

Out[27]:
```
Credit reporting, credit repair services, or other personal consumer reports    311468
Debt collection                                                                 126545
Credit card or prepaid card                                                      60020
Mortgage                                                                         59028
Checking or savings account                                                      36842
Student loan                                                                     23786
Money transfer, virtual currency, or money service                               18275
Vehicle loan or lease                                                            14817
Payday loan, title loan, or personal loan                                        10312
Credit reporting                                                                  9188
Credit card                                                                       5232
Bank account or service                                                           4325
Consumer Loan                                                                     2843
Payday loan                                                                        439
Money transfers                                                                    375
Prepaid card                                                                       290
Other financial service                                                            105
Virtual currency                                                                     5
Name: PRODUCT, dtype: int64
```

Removing columns with aggregated values

In [28]:
```
Dataset_PC.drop(
    Dataset_PC[
    Dataset_PC.PRODUCT ==
    'Credit reporting, credit repair services, or other personal consumer reports'].index,
    inplace=True) # credit_aggregated

Dataset_PC.drop(
    Dataset_PC[
    Dataset_PC.PRODUCT ==
    'Credit card or prepaid card'].index,
    inplace=True) # cred_or_prepaid

Dataset_PC.drop(
    Dataset_PC[
    Dataset_PC.PRODUCT ==
    'Money transfer, virtual currency, or money service'].index,
    inplace=True) # money_virtual_service

Dataset_PC.drop(
    Dataset_PC[
    Dataset_PC.PRODUCT ==
    'Payday loan, title loan, or personal loan'].index,
    inplace=True) # payday_title_personal_loan
```

Value counts of the products after dropping some columns

In [29]:
```
Dataset_PC.PRODUCT.value_counts()
```

Out[29]:
```
Debt collection               126545
Mortgage                       59028
Checking or savings account    36842
Student loan                   23786
Vehicle loan or lease          14817
Credit reporting                9188
Credit card                     5232
Bank account or service         4325
Consumer Loan                   2843
Payday loan                      439
Money transfers                  375
Prepaid card                     290
Other financial service          105
Virtual currency                   5
Name: PRODUCT, dtype: int64
```

Combining all loans

In [30]:
```
Dataset_PC.replace('Student loan', 'Loan', inplace=True)
Dataset_PC.replace('Consumer Loan', 'Loan', inplace=True)
Dataset_PC.replace('Payday loan', 'Loan', inplace=True)
Dataset_PC.replace('Vehicle loan or lease', 'Loan', inplace=True)
```

Adding virtual currency to other financial service

In [31]:
```
Dataset_PC.replace('Virtual currency', 'Other financial service', inplace=True)
```

Value counts after combining columns

In [32]:
```
Dataset_PC.PRODUCT.value_counts()
```

Out[32]:
```
Debt collection                126545
Mortgage                        59028
Loan                            41885
Checking or savings account     36842
Credit reporting                 9188
Credit card                      5232
Bank account or service          4325
Money transfers                   375
Prepaid card                      290
Other financial service           110
Name: PRODUCT, dtype: int64
```

Checking update of list of multiple Products in our data

In [33]:
```
print('--------------')
print('Categories in PRODUCT column:')
print('--------------\n')
print(Dataset_PC['PRODUCT'].unique(), '\n')
print('--------------')
print('# of unique categories: ', Dataset_PC['PRODUCT'].nunique())
print('--------------')
```

```
--------------
Categories in PRODUCT column:
--------------

['Debt collection' 'Checking or savings account' 'Mortgage' 'Loan'
 'Credit reporting' 'Credit card' 'Bank account or service'
 'Money transfers' 'Other financial service' 'Prepaid card'] 

--------------
# of unique categories:  10
--------------
```

Encoding the PRODUCT column

In [34]:
```
Dataset_PC['PRODUCT_ID'] = Dataset_PC['PRODUCT'].factorize()[0]
```

Creating dataframe of the PRODUCT to their respective PRODUCT_ID

In [35]:
```
category_id_df = Dataset_PC[['PRODUCT', 'PRODUCT_ID']].drop_duplicates()
```

Creating our cheatsheets for each encoded label

In [36]:
```
category_to_id = dict(category_id_df.values)
id_to_category = dict(category_id_df[['PRODUCT_ID', 'PRODUCT']].values)
```

Checking our Dataframe

In [37]:
```
Dataset_PC.head(10)
```

Out[37]:

	PRODUCT	CONSUMER_COMPLAINT	PRODUCT_ID
0	Debt collection	never signed paperwork company stating promise...	0
4	Debt collection	hii recently discovered bank usaa placed hold ...	0
7	Debt collection	last year xxxxxxxx closed xxxx account returne...	0
11	Checking or savings account	xxxx someone took xxxx dollar acct via debit c...	1
12	Mortgage	xxxx got behind mortgage unforeseen expense un...	2

17	Debt collection	several month writing xxxx exeter account repo...	0
18	Debt collection	victim identity theft debt belong please see i...	0
21	Debt collection	call convergent outsource saying give credit c...	0
22	Debt collection	today received letter dynamic recovery solutio...	0
23	Loan	response letter date xxxxxxxx 1 loan validatio...	3

Loaded Dataframe Review

In [38]:
```
print(Dataset_PC.info())
print('-------------------------------------------------------------------------------------------')
print(Dataset_PC.head().to_string())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 283820 entries, 0 to 683894
Data columns (total 3 columns):
 #   Column              Non-Null Count   Dtype 
---  ------              --------------   ----- 
 0   PRODUCT             283820 non-null  object
 1   CONSUMER_COMPLAINT  283820 non-null  object
 2   PRODUCT_ID          283820 non-null  int64 
dtypes: int64(1), object(2)
memory usage: 8.7+ MB
None
-------------------------------------------------------------------------------------------
                         PRODUCT
CONSUMER_COMPLAINT  PRODUCT_ID
0                Debt collection
never signed paperwork company stating promised anything           0
4                Debt collection
hii recently discovered bank usaa placed hold checking account varying amount money due fact state virginia place
d incorrect fund hold state virginia believed owed back tax 2015 incorrect live virginia 2015 required fax tax do
cument state virginia order removed hold account hold removed discovered bank usaa charged xxxx fee legal service
rendered feel unfair unjust charge owe back tax state virginia therefore owe xxxx disputed charge success stated
policy budge circumstancesthis fee seems like scam bank state state falsely place hold someone account rightfully
disputed release fund bank unfairly profit scam could attacked hundred thousand people creating significant sourc
e income bank bank justify use fee beyond caseis legality legal fee process take get investigated official capaci
ty recourse thisi appreciate help give timethanksxxxx xxxx           0
7                Debt collection
last year xxxxxxxx closed xxxx account returned box wife opened another account take advantage promotion paid las
t bill returning box asked balance pending paid said date payment last week xxxxxxxx received phone call lady cla
iming needed pay 7600 past debt xxxx asked clarification said needed pay right away send collection told paid eve
rything closed account said call xxxx talk hanged called xxxx said date payment asked check previous account didn
t find anything thought call scam due didnt find anything system today xxxxxxxx received letter diversified consu
ltant inc stating collection amount called said credit already affected completely acceptable owe money pay would
like receive clarification due fact customer service agent saying ok xxxx additionally completely unfair damaged
credit score something didnt even know owing owe money pay problem need fix credit record affecting credit histor
y score assure never received anything xxxx regard balance otherwise would called find pay main reason submitting
request remove credit history fix credit score thanks           0
11  Checking or savings account
xxxx someone took xxxx dollar acct via debit card atm bofa xxxx az happened 2 hour prior call bank card pocession
entire time bofa credited account took later stating fraud committed closed claim said asked came conclusion chip
read asked video surveillance told never suggested call police still resolution thievery bofa stole xxxx dollar w
ithout proof fraud resolution filling complaint federal reserve xxxx xxxx xxxx xxxx           1
12                    Mortgage  xxxx got behind mortgage unforeseen expense uncle murdered cover funeral expense
sued foreclosure assistance xxxx xxxx xxxx xxxx xxxx submitted loan modification application counsel foreclosure
plaintiff bayview xxxxxxxx xxxxxxxx employee counsel bayview sent xxxx email confirming receipt application stati
ng eligible modification due file sic 3 prior modification loan day xxxx requested documentation showing 3 prior
modification several day counsel bayview provided documentation five prior modification employee bayviews law fir
m noted loan modified three time additional modification approved investor exception granted investor according l
aw firm employee xxxx xxxx employee provided link online xxxx xxxx xxxx xxxx xxxx xxxx link identify particular p
art user guide relating number modification xxxx xxxx permit nonetheless search user guide reveals xxxx xxxx prov
ides servicers may ask xxxx xxxx permission modify loan modified three time explains mandatory courtmediated sett
lement conference xxxxxxxx xxxx asked bayview request xxxx xxxx exception threemodification cap bayviews document
show done past xxxx xxxx acceding request court referee presiding conference directed bayviews counsel provide do
cumentation subsequent conference request xxxx xxxx xxxx xxxx response despite court clear directive bayviews cou
nsel provide documentation request xxxx xxxx exception xxxx xxxx response letter emailed xxxx xxxxxxxx bayviews c
ounsel stated conclusorily exception request submitted xxxx xxxx xxxxxxxx xxxxxxxx decision investor received adv
ising waiver request denied see appended copy xxxxxxxx letter xxxx responded pointing response satisfy court dire
ctive bayviews counsel reply provide detail however xxxxxxxx surprise relief bayview offered trial modification s
ee appended copy xxxxxxxx trial offer made timely trial payment bayview acceptance offer expressly contemplated t
erm offer elated offered opportunity save home live fianc child first trial payment sent bayview debited bank acc
ount although bayview accepted payment xxxxxxxx bayview sent refund check informed writing trial offer made error
see appended copy xxxxxxxx letter copy refund check time accepted trial modification offerwhich term required mak
e three timely trial payment order successfully complete trialand receiving bayviews purported withdrawal accepte
d offer reasonably relied upon offer spending money repair home received offer accepted term affordable would mad
e repair received offer feared would able keep home light pending foreclosure prior modification denial
2
```

Examining a sample of text using TfidfVectorizer

In [39]:
```
sample_complaint = list(Dataset_PC.CONSUMER_COMPLAINT[:5])[4]

list_sample_complaint = []
list_sample_complaint.append(sample_complaint)
list_sample_complaint
```

Out[39]:
```
['xxxx got behind mortgage unforeseen expense uncle murdered cover funeral expense sued foreclosure assistance xx
xx xxxx xxxx xxxx xxxx submitted loan modification application counsel foreclosure plaintiff bayview xxxxxxxx xxx
xxxxx employee counsel bayview sent xxxx email confirming receipt application stating eligible modification due f
ile sic 3 prior modification loan day xxxx requested documentation showing 3 prior modification several day couns
el bayview provided documentation five prior modification employee bayviews law firm noted loan modified three ti
me additional modification approved investor exception granted investor according law firm employee xxxx xxxx emp
loyee provided link online xxxx xxxx xxxx xxxx xxxx xxxx link identify particular part user guide relating number
modification xxxx xxxx permit nonetheless search user guide reveals xxxx xxxx provides servicers may ask xxxx xxx
x permission modify loan modified three time explains mandatory courtmediated settlement conference xxxxxxxx xxxx
asked bayview request xxxx xxxx exception threemodification cap bayviews document show done past xxxx xxxx accedi
ng request court referee presiding conference directed bayviews counsel provide documentation subsequent conferen
ce request xxxx xxxx xxxx xxxx response despite court clear directive bayviews counsel provide documentation requ
est xxxx xxxx exception xxxx xxxx response letter emailed xxxx xxxxxxxx bayviews counsel stated conclusorily exce
ption request submitted xxxx xxxx xxxxxxxx xxxxxxxx decision investor received advising waiver request denied see
appended copy xxxxxxxx letter xxxx responded pointing response satisfy court directive bayviews counsel reply pro
vide detail however xxxxxxxx surprise relief bayview offered trial modification see appended copy xxxxxxxx trial
offer made timely trial payment bayview acceptance offer expressly contemplated term offer elated offered opportu
nity save home live fianc child first trial payment sent bayview debited bank account although bayview accepted p
ayment xxxxxxxx bayview sent refund check informed writing trial offer made error see appended copy xxxxxxxx lett
er copy refund check time accepted trial modification offerwhich term required make three timely trial payment or
der successfully complete trialand receiving bayviews purported withdrawal accepted offer reasonably relied upon
offer spending money repair home received offer accepted term affordable would made repair received offer feared
would able keep home light pending foreclosure prior modification denial']
```

Examining the words extracted through the TfidfVectorizer

In [40]:
```
tf_idf3 = TfidfVectorizer(stop_words='english')
check3 = tf_idf3.fit_transform(list_sample_complaint)

print(tf_idf3.get_feature_names())
```

```
['able', 'acceding', 'acceptance', 'accepted', 'according', 'account', 'additional', 'advising', 'affordable', 'a
ppended', 'application', 'approved', 'ask', 'asked', 'assistance', 'bank', 'bayview', 'bayviews', 'cap', 'check',
'child', 'clear', 'complete', 'conclusorily', 'conference', 'confirming', 'contemplated', 'copy', 'counsel', 'cou
rt', 'courtmediated', 'cover', 'day', 'debited', 'decision', 'denial', 'denied', 'despite', 'directed', 'directiv
e', 'document', 'documentation', 'elated', 'eligible', 'email', 'emailed', 'employee', 'error', 'exception', 'exp
ense', 'explains', 'expressly', 'feared', 'fianc', 'file', 'firm', 'foreclosure', 'funeral', 'got', 'granted', 'g
uide', 'home', 'identify', 'informed', 'investor', 'law', 'letter', 'light', 'link', 'live', 'loan', 'make', 'man
datory', 'modification', 'modified', 'modify', 'money', 'mortgage', 'murdered', 'nonetheless', 'noted', 'number',
'offer', 'offered', 'offerwhich', 'online', 'opportunity', 'order', 'particular', 'past', 'payment', 'pending', '
permission', 'permit', 'plaintiff', 'pointing', 'presiding', 'prior', 'provide', 'provided', 'provides', 'purport
ed', 'reasonably', 'receipt', 'received', 'receiving', 'referee', 'refund', 'relating', 'relied', 'relief', 'repa
ir', 'reply', 'request', 'requested', 'required', 'responded', 'response', 'reveals', 'satisfy', 'save', 'search'
, 'sent', 'servicers', 'settlement', 'showing', 'sic', 'spending', 'stated', 'stating', 'submitted', 'subsequent'
, 'successfully', 'sued', 'surprise', 'term', 'threemodification', 'time', 'timely', 'trial', 'trialand', 'uncle'
, 'unforeseen', 'user', 'waiver', 'withdrawal', 'writing', 'xxxx', 'xxxxxxxx']
```

Statistical Analysis

Divide your dataset into Train (80 Percent) and Test (20 percent)

In [41]:
```
X, y = Dataset_PC.CONSUMER_COMPLAINT, Dataset_PC.PRODUCT_ID
print('X shape:', X.shape, 'y shape:', y.shape)
```

```
X shape: (283820,) y shape: (283820,)
```

In [42]:
```
from sklearn.model_selection import train_test_split
X_valid, X_test, y_valid, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=seed)
print('X_valid', X_valid.shape)
print('y_valid', y_valid.shape)
print('X_test', X_test.shape)
print('y_test', y_test.shape)
```

```
X_valid (227056,)
y_valid (227056,)
```

```
X_test (56764,)
y_test (56764,)
```

Model Selection

Machine Learning

TF-IDF

In [43]:
```
enc = preprocessing.LabelEncoder()
y_test = enc.fit_transform(y_test)
y_valid = enc.fit_transform(y_valid)
```

tf-idf representation

In [44]:
```
tfidf_vect = TfidfVectorizer(analyzer='word', token_pattern=r'\w{1,}', max_features=5000)
tfidf_vect.fit(Dataset_PC['CONSUMER_COMPLAINT'])
xtest_tfidf =  tfidf_vect.transform(X_test)
xvalid_tfidf =  tfidf_vect.transform(X_valid)
```

LogisticRegression

In [45]:
```
from sklearn.model_selection import GridSearchCV
lr = LogisticRegression()
lr_params = {'C':[int(x) for x in np.linspace(1,10,10)]}
grid_lr = GridSearchCV(estimator=lr,param_grid=lr_params,cv=5,n_jobs=-1)
grid_lr.fit(xvalid_tfidf,y_valid)
```

Out[45]:
```
GridSearchCV(cv=5, estimator=LogisticRegression(), n_jobs=-1,
             param_grid={'C': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]})
```

In [46]:
```
print(grid_lr.best_params_)
print(grid_lr.best_score_)
```

```
{'C': 3}
0.8990689507483781
```

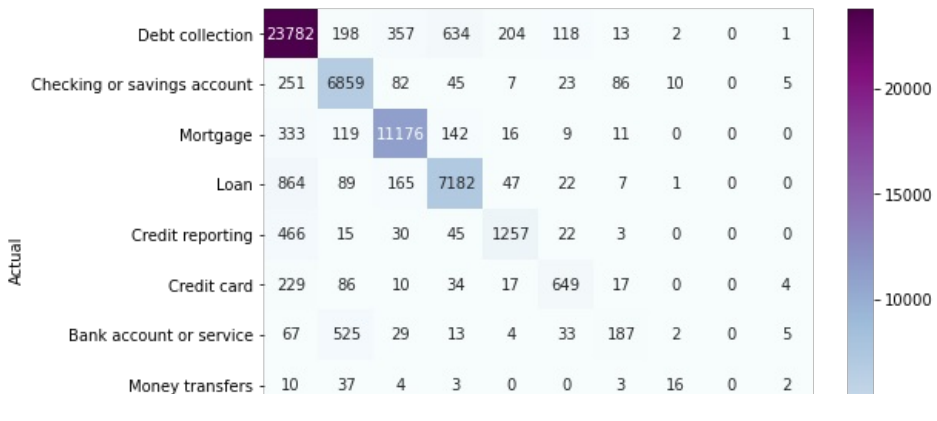
In [47]:
```
final_lr = LogisticRegression(C=5)
final_lr.fit(xvalid_tfidf,y_valid)
```

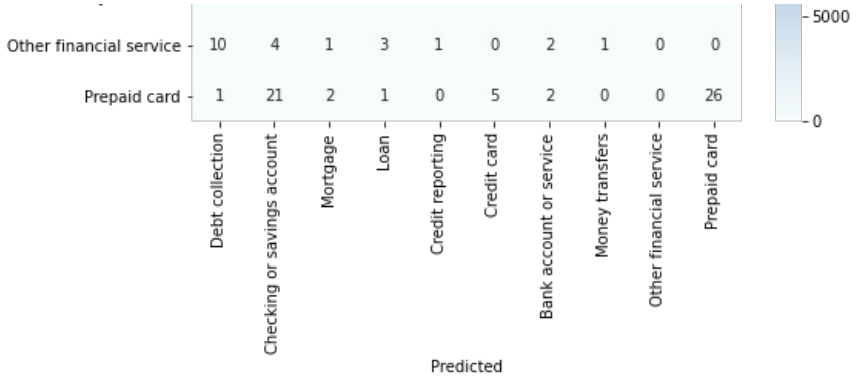
Out[47]:
```
LogisticRegression(C=5)
```

In [48]:
```
final_lr_predict = final_lr.predict(xtest_tfidf)
lr_accuracy = metrics.accuracy_score(final_lr_predict, y_test)
print ("Logistic Regression > Accuracy: ", lr_accuracy)
```

```
Logistic Regression > Accuracy:  0.9008174194912268
```

In [49]:
```
from sklearn.metrics import confusion_matrix
conf_mat = confusion_matrix(y_test, final_lr_predict)
fig, ax = plt.subplots(figsize=(8,6))
sns.heatmap(conf_mat, annot=True, fmt="d", cmap="BuPu",xticklabels=Dataset_PC['PRODUCT'].unique(),yticklabels=Dat
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```





SVC

In [50]:
```
from sklearn.svm import LinearSVC
svc = LinearSVC()
svc_params = {'C':[0.01,0.1, 1, 10, 100, 1000]}
grid_svc = GridSearchCV(estimator=svc,param_grid=svc_params,cv=5,n_jobs=-1)
grid_svc.fit(xvalid_tfidf,y_valid)
```

Out[50]:
```
GridSearchCV(cv=5, estimator=LinearSVC(), n_jobs=-1,
             param_grid={'C': [0.01, 0.1, 1, 10, 100, 1000]})
```

In [51]:
```
print(grid_svc.best_params_)
print(grid_svc.best_score_)
```

```
{'C': 1}
0.8981308582567753
```

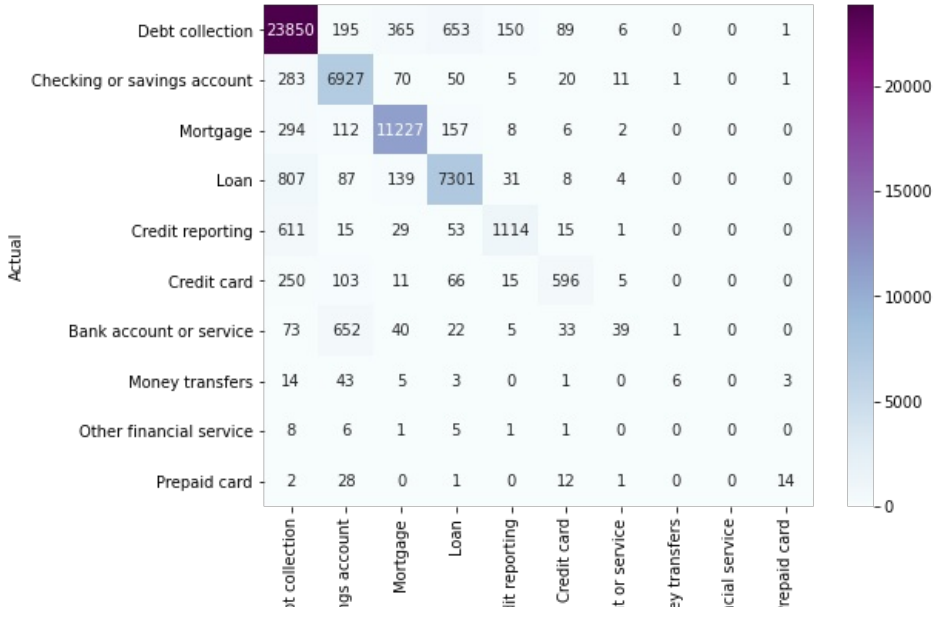
In [52]:
```
final_svc = LinearSVC(C=0.1)
final_svc.fit(xvalid_tfidf,y_valid)
```

Out[52]:
```
LinearSVC(C=0.1)
```

In [53]:
```
final_svc_predict = final_svc.predict(xtest_tfidf)
svc_accuracy = metrics.accuracy_score(final_svc_predict, y_test)
print ("SVC > Accuracy: ", svc_accuracy)
```

```
SVC > Accuracy:  0.8997604115284336
```

In [54]:
```
conf_mat = confusion_matrix(y_test, final_svc_predict)
fig, ax = plt.subplots(figsize=(8,6))
sns.heatmap(conf_mat, annot=True, fmt="d", cmap="BuPu",xticklabels=Dataset_PC['PRODUCT'].unique(),yticklabels=Dat
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



MultinomialNB

In [55]:
```
from sklearn.naive_bayes import MultinomialNB
NB = MultinomialNB()
NB.fit(xvalid_tfidf,y_valid)
```

Out[55]:
```
MultinomialNB()
```

In [56]:
```
NB_predict = NB.predict(xtest_tfidf)
NB_accuracy = metrics.accuracy_score(NB_predict, y_test)
print ("MultinomialNB > Accuracy: ", NB_accuracy)
```

```
MultinomialNB > Accuracy:  0.8669403142837009
```

In [57]:
```
from sklearn.metrics import confusion_matrix
conf_mat = confusion_matrix(y_test, NB_predict)
fig, ax = plt.subplots(figsize=(8,6))
sns.heatmap(conf_mat, annot=True, fmt="d", cmap="BuPu",xticklabels=Dataset_PC['PRODUCT'].unique(),yticklabels=Dat
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

RandomForestClassifier

In [58]:
```
randomforest = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=0)
randomforest.fit(xvalid_tfidf,y_valid)
```

Out[58]:
```
RandomForestClassifier(max_depth=5, random_state=0)
```

In [59]:
```
randomforest_predict = randomforest.predict(xtest_tfidf)
randomforest_accuracy = metrics.accuracy_score(randomforest_predict, y_test)
print ("RandomForestClassifier > Accuracy: ", randomforest_accuracy)
```

```
RandomForestClassifier > Accuracy:  0.5637551969558171
```

In [60]:
```
from sklearn.metrics import confusion_matrix
conf_mat = confusion_matrix(y_test, randomforest_predict)
fig, ax = plt.subplots(figsize=(8,6))
sns.heatmap(conf_mat, annot=True, fmt="d", cmap="BuPu",xticklabels=Dataset_PC['PRODUCT'].unique(),yticklabels=Dat
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

KNeighborsClassifier

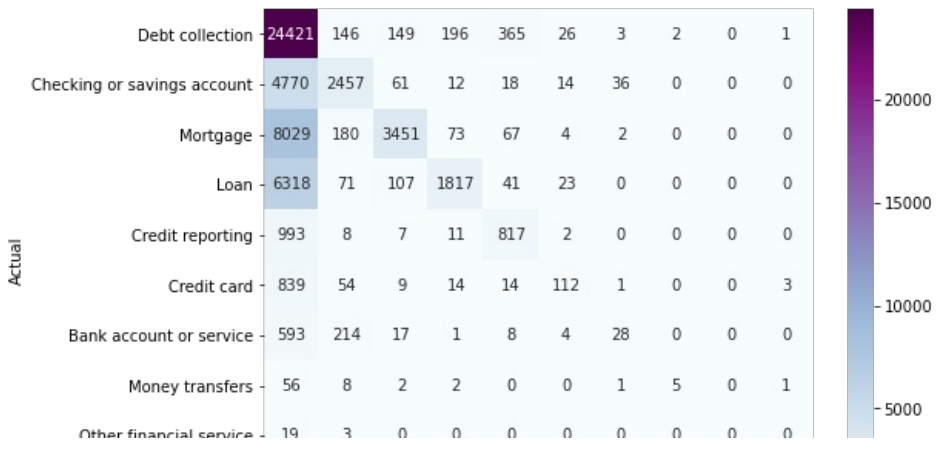
In [61]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(xvalid_tfidf,y_valid)
```

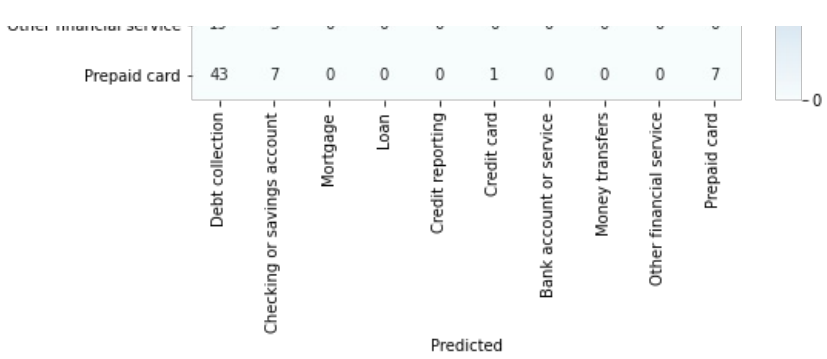
Out[61]:
```
KNeighborsClassifier(n_neighbors=3)
```

In [62]:
```
knn_predict = knn.predict(xtest_tfidf)
knn_accuracy = metrics.accuracy_score(knn_predict, y_test)
print ("KNeighborsClassifier > Accuracy: ", knn_accuracy)
```

```
KNeighborsClassifier > Accuracy:  0.583380311465013
```

In [63]:
```
from sklearn.metrics import confusion_matrix
conf_mat = confusion_matrix(y_test, knn_predict)
fig, ax = plt.subplots(figsize=(8,6))
sns.heatmap(conf_mat, annot=True, fmt="d", cmap="BuPu",xticklabels=Dataset_PC['PRODUCT'].unique(),yticklabels=Dat
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```





Deep Learning

Neural Network

In [64]:
```
from keras.preprocessing.text import Tokenizer
from keras.layers import Dense, Input, LSTM, Embedding, Dropout, Activation
from keras.layers import Bidirectional, GlobalMaxPool1D, Conv1D, SimpleRNN
from keras.models import Model
from keras import initializers, regularizers, constraints, optimizers, layers
from keras.layers import Dense, Input, Flatten, Dropout, BatchNormalization
from keras.layers import Conv1D, MaxPooling1D, Embedding
from keras.preprocessing.sequence import pad_sequences
from keras.callbacks import ModelCheckpoint
from keras.layers import LSTM,Dropout
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from keras.layers import LSTM, Conv1D, MaxPooling1D, Dropout
from keras.callbacks import EarlyStopping
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.wrappers.scikit_learn import KerasClassifier
from keras.preprocessing import text, sequence
from keras import layers, models, optimizers
```

In [65]:
```
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(Dataset_PC.CONSUMER_COMPLAINT)

xvalid_tfidf = tokenizer.texts_to_sequences(X_valid)
xtest_tfidf = tokenizer.texts_to_sequences(X_test)
vocab_size = len(tokenizer.word_index) + 1
```

In [66]:
```
seq_lens = [len(s) for s in xvalid_tfidf]
print("average length: %0.1f" % np.mean(seq_lens))
print("max length: %d" % max(seq_lens))
```

```
average length: 106.5
max length: 3087
```

In [67]:
```
maxlen = 150
xvalid_tfidf = pad_sequences(xvalid_tfidf, padding='post', maxlen=maxlen)
xtest_tfidf = pad_sequences(xtest_tfidf, padding='post', maxlen=maxlen)
```

In [68]:
```
embedding_dim = 50
callback = EarlyStopping(monitor='val_loss', patience=2)

model = Sequential()
model.add(layers.Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=maxlen))
model.add(layers.Flatten())
model.add(layers.Dense(10, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 150, 50)           7097150
_________________________________________________________________
flatten (Flatten)            (None, 7500)              0
_________________________________________________________________
dense (Dense)                (None, 10)                75010
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 11
```

```
=================================================================
Total params: 7,172,171
Trainable params: 7,172,171
Non-trainable params: 0
_________________________________________________________________
```

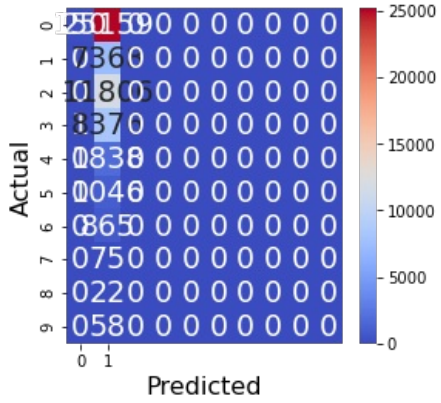
In [69]:
```
history = model.fit(xvalid_tfidf, y_valid,epochs=10,verbose=True,validation_data=(xtest_tfidf, y_test),batch_size
```

```
Epoch 1/10
228/228 [==============================] - 15s 62ms/step - loss: -296.8082 - accuracy: 0.1308 - val_loss: -1191.3
368 - val_accuracy: 0.1306
Epoch 2/10
228/228 [==============================] - 14s 62ms/step - loss: -4369.2715 - accuracy: 0.1312 - val_loss: -9374.
6152 - val_accuracy: 0.1322
Epoch 3/10
228/228 [==============================] - 14s 62ms/step - loss: -18796.2090 - accuracy: 0.1318 - val_loss: -3107
8.8320 - val_accuracy: 0.1323
Epoch 4/10
228/228 [==============================] - 15s 64ms/step - loss: -49473.1055 - accuracy: 0.1319 - val_loss: -7137
1.5625 - val_accuracy: 0.1313
Epoch 5/10
228/228 [==============================] - 15s 64ms/step - loss: -100604.2188 - accuracy: 0.1321 - val_loss: -133
966.7812 - val_accuracy: 0.1324
Epoch 6/10
228/228 [==============================] - 15s 65ms/step - loss: -175980.5156 - accuracy: 0.1320 - val_loss: -222
002.4688 - val_accuracy: 0.1324
Epoch 7/10
228/228 [==============================] - 15s 65ms/step - loss: -277507.0312 - accuracy: 0.1321 - val_loss: -337
497.7188 - val_accuracy: 0.1324
Epoch 8/10
228/228 [==============================] - 17s 74ms/step - loss: -408029.0000 - accuracy: 0.1323 - val_loss: -482
879.8125 - val_accuracy: 0.1324
Epoch 9/10
228/228 [==============================] - 15s 64ms/step - loss: -569783.0000 - accuracy: 0.1322 - val_loss: -660
161.5000 - val_accuracy: 0.1324
Epoch 10/10
228/228 [==============================] - 15s 64ms/step - loss: -763887.0000 - accuracy: 0.1322 - val_loss: -869
760.4375 - val_accuracy: 0.1324
```

In [70]:
```
NN_score=accuracy_score(y_test, (model.predict(xtest_tfidf) > 0.5).astype("int32"))
print(NN_score)
```

```
0.13244309773800295
```

In [71]:
```
plt.figure(figsize=(4,4))
sns.heatmap(confusion_matrix(y_test, (model.predict(xtest_tfidf) > 0.5).astype("int32")),annot=True,cmap='coolwar
plt.xlabel("Predicted",fontsize=16)
plt.ylabel("Actual",fontsize=16)
plt.show()
```



CNN

In [72]:
```
embedding_vecor_length = 32
callback = EarlyStopping(monitor='val_loss', patience=2)

model = Sequential()
model.add(Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=maxlen))
model.add(Conv1D(filters=32, kernel_size=3, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=2))
```

```
model.add(LSTM(100))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

print(model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 150, 50)           7097150
_________________________________________________________________
conv1d (Conv1D)              (None, 150, 32)           4832
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 75, 32)            0
_________________________________________________________________
lstm (LSTM)                  (None, 100)               53200
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 101
=================================================================
Total params: 7,155,283
Trainable params: 7,155,283
Non-trainable params: 0
_________________________________________________________________
None
```

In [73]:
```
model.fit(xvalid_tfidf, y_valid, epochs=10, batch_size=256,verbose = 1,validation_data=(xtest_tfidf,y_test),callb
```

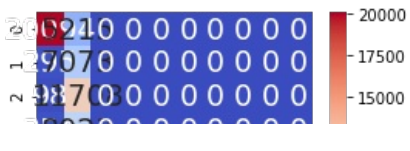
```
Epoch 1/10
887/887 [==============================] - 214s 239ms/step - loss: -16.6276 - accuracy: 0.1302 - val_loss: -30.32
16 - val_accuracy: 0.1298
Epoch 2/10
887/887 [==============================] - 206s 232ms/step - loss: -72.1808 - accuracy: 0.3296 - val_loss: -113.4
179 - val_accuracy: 0.1319
Epoch 3/10
887/887 [==============================] - 204s 230ms/step - loss: -137.0159 - accuracy: 0.2402 - val_loss: -187.
6218 - val_accuracy: 0.4310
Epoch 4/10
887/887 [==============================] - 204s 229ms/step - loss: -216.5824 - accuracy: 0.4460 - val_loss: -251.
7042 - val_accuracy: 0.4486
Epoch 5/10
887/887 [==============================] - 206s 232ms/step - loss: -279.2773 - accuracy: 0.4480 - val_loss: -297.
3667 - val_accuracy: 0.4697
Epoch 6/10
887/887 [==============================] - 204s 230ms/step - loss: -330.4332 - accuracy: 0.4282 - val_loss: -368.
0677 - val_accuracy: 0.4562
Epoch 7/10
887/887 [==============================] - 203s 229ms/step - loss: -397.7878 - accuracy: 0.4523 - val_loss: -428.
9879 - val_accuracy: 0.4661
Epoch 8/10
887/887 [==============================] - 204s 230ms/step - loss: -408.5009 - accuracy: 0.3463 - val_loss: -477.
0526 - val_accuracy: 0.4423
Epoch 9/10
887/887 [==============================] - 204s 230ms/step - loss: -510.8165 - accuracy: 0.4429 - val_loss: -540.
0013 - val_accuracy: 0.4849
Epoch 10/10
887/887 [==============================] - 204s 230ms/step - loss: -580.9500 - accuracy: 0.4806 - val_loss: -604.
0995 - val_accuracy: 0.4786
```

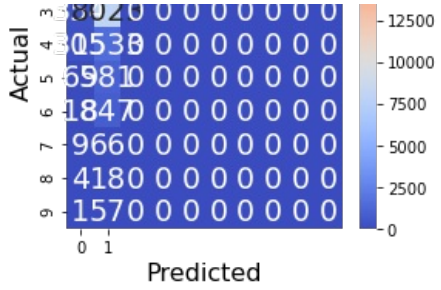
Out[73]:
```
<keras.callbacks.History at 0x7fd6f06d6490>
```

In [74]:
```
CNN_score=accuracy_score(y_test, (model.predict(xtest_tfidf) > 0.5).astype("int32"))
print(CNN_score)
```

```
0.4785955887534353
```

In [75]:
```
plt.figure(figsize=(4,4))
sns.heatmap(confusion_matrix(y_test, (model.predict(xtest_tfidf) > 0.5).astype("int32")),annot=True,cmap='coolwar
plt.xlabel("Predicted",fontsize=16)
plt.ylabel("Actual",fontsize=16)
plt.show()
```





LSTM

In [76]:
```
embedding_vecor_length = 32
callback = EarlyStopping(monitor='val_loss', patience=2)

model = Sequential()
model.add(Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=maxlen))
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

print(model.summary())
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 150, 50)           7097150
_________________________________________________________________
lstm_1 (LSTM)                (None, 100)               60400
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 101
=================================================================
Total params: 7,157,651
Trainable params: 7,157,651
Non-trainable params: 0
_________________________________________________________________
None
```

In [77]:
```
model.fit(xvalid_tfidf, y_valid, epochs=10, batch_size=256,verbose = 1,validation_data=(xtest_tfidf,y_test),callb
```

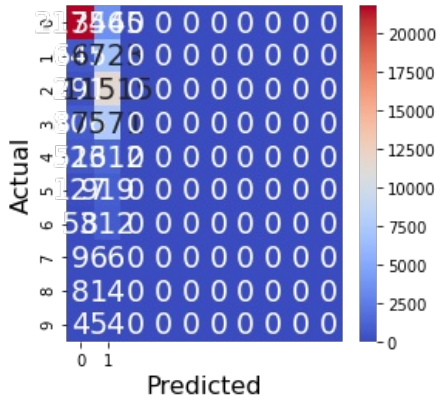
```
Epoch 1/10
887/887 [==============================] - 428s 479ms/step - loss: -18.5178 - accuracy: 0.1299 - val_loss: -33.16
23 - val_accuracy: 0.1298
Epoch 2/10
887/887 [==============================] - 425s 479ms/step - loss: -47.1748 - accuracy: 0.1298 - val_loss: -61.21
73 - val_accuracy: 0.1298
Epoch 3/10
887/887 [==============================] - 425s 479ms/step - loss: -75.0793 - accuracy: 0.1298 - val_loss: -89.00
56 - val_accuracy: 0.1298
Epoch 4/10
887/887 [==============================] - 494s 557ms/step - loss: -102.9152 - accuracy: 0.1298 - val_loss: -116.
9911 - val_accuracy: 0.1298
Epoch 5/10
887/887 [==============================] - 430s 485ms/step - loss: -133.4171 - accuracy: 0.2671 - val_loss: -190.
9111 - val_accuracy: 0.4323
Epoch 6/10
887/887 [==============================] - 426s 480ms/step - loss: -197.6985 - accuracy: 0.3435 - val_loss: -226.
9104 - val_accuracy: 0.3611
Epoch 7/10
887/887 [==============================] - 430s 485ms/step - loss: -240.9272 - accuracy: 0.3375 - val_loss: -250.
2864 - val_accuracy: 0.3296
Epoch 8/10
887/887 [==============================] - 425s 479ms/step - loss: -294.7857 - accuracy: 0.3741 - val_loss: -344.
6214 - val_accuracy: 0.4225
Epoch 9/10
887/887 [==============================] - 425s 479ms/step - loss: -377.3726 - accuracy: 0.4250 - val_loss: -407.
2430 - val_accuracy: 0.4481
Epoch 10/10
887/887 [==============================] - 425s 479ms/step - loss: -436.8148 - accuracy: 0.4314 - val_loss: -459.
5068 - val_accuracy: 0.5015
```

Out[77]:
```
<keras.callbacks.History at 0x7fd6f0fdaa00>
```

In [78]:
```
LSTM_score=accuracy_score(y_test, (model.predict(xtest_tfidf) > 0.5).astype("int32"))
print(LSTM_score)
```

```
0.5014974279472906
```

In [79]:
```
plt.figure(figsize=(4,4))
sns.heatmap(confusion_matrix(y_test, (model.predict(xtest_tfidf) > 0.5).astype("int32")),annot=True,cmap='coolwar
plt.xlabel("Predicted",fontsize=16)
plt.ylabel("Actual",fontsize=16)
plt.show()
```

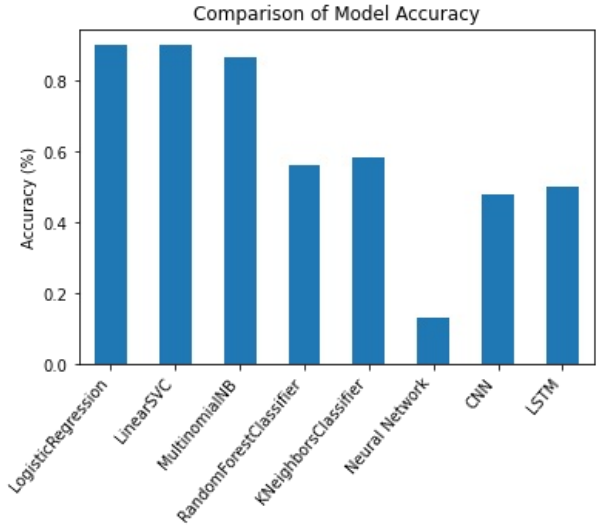


Comparing Performance

Machine Learning & Deep Learning

In [80]:
```
plotdata = pd.DataFrame( {"scores": [lr_accuracy,svc_accuracy,NB_accuracy,randomforest_accuracy,knn_accuracy,NN_s
    index=['LogisticRegression','LinearSVC','MultinomialNB','RandomForestClassifier','KNeighborsClassifier','Neur
plotdata['scores'].plot(kind='bar')
plt.xticks(rotation=50, horizontalalignment='right')
plt.title("Comparison of Model Accuracy")
plt.ylabel("Accuracy (%)")
```

Out[80]:
```
Text(0, 0.5, 'Accuracy (%)')
```



TESTING BEST MODEL FOR PREDICTION

In [81]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
X_valid, X_test, y_valid, y_test = train_test_split(Dataset_PC['CONSUMER_COMPLAINT'], Dataset_PC['PRODUCT'], rand
count_vect = CountVectorizer()
X_valid_counts = count_vect.fit_transform(X_valid)
tfidf_transformer = TfidfTransformer()
X_valid_tfidf = tfidf_transformer.fit_transform(X_valid_counts)
```

In [82]:
```
X_test_counts = count_vect.transform(X_test)
```

```
X_test_tfidf = tfidf_transformer.transform(X_test_counts)
```

In [83]:
```
Cs = [0.001, 0.01, 0.1, 1, 100, 1000 ]
for c in Cs:
    print(c)
    LR = LogisticRegression(C = c,random_state=0)
    LR.fit(X_valid_tfidf, y_valid)
    print(LR.score(X_test_tfidf, y_test))
```

```
0.001
0.6657740821647523
0.01
0.8462405750123317
0.1
0.888295398492002
1
0.9009090268480022
100
0.8994010288210837
1000
0.89841448805581
```

In [84]:
```
Cs = [0.001, 0.01, 0.1, 1, 100, 1000 ]
for c in Cs:
    print(c)
    SVC = LinearSVC(C = c)
    SVC.fit(X_valid_tfidf, y_valid)
    print(SVC.score(X_test_tfidf, y_test))
```

```
0.001
0.8427454020153619
0.01
0.8866182791910365
0.1
0.9010499612430414
1
0.9027129871045029
100
0.867239799873159
1000
0.8503558593474737
```

In [85]:
```
Cs = [0.001, 0.01, 0.1, 1, 100, 1000 ]
for c in Cs:
    print(c)
    NB = MultinomialNB()
    NB.fit(X_valid_tfidf, y_valid)
    print(NB.score(X_test_tfidf, y_test))
```

```
0.001
0.820562328236206
0.01
0.820562328236206
0.1
0.820562328236206
1
0.820562328236206
100
0.820562328236206
1000
0.820562328236206
```

In [86]:
```
Cs = [0.001, 0.01, 0.1, 1, 100, 1000 ]
for c in Cs:
    print(c)
    randomforest = RandomForestClassifier()
    randomforest.fit(X_valid_tfidf, y_valid)
    print(randomforest.score(X_test_tfidf, y_test))
```

```
0.001
0.8669579310830808
0.01
0.8652526249031076
0.1
0.8640405891057713
1
0.8648016348389824
100
0.8642097103798182
```

```
1000
0.8653512789796349
```

In [87]:
```
Cs = [0.001, 0.01, 0.1, 1, 100, 1000 ]
for c in Cs:
    print(c)
    knn = KNeighborsClassifier()
    knn.fit(X_valid_tfidf, y_valid)
    print(knn.score(X_test_tfidf, y_test))
```

```
0.001
0.8039884433796068
0.01
0.8039884433796068
0.1
0.8039884433796068
1
0.8039884433796068
100
0.8039884433796068
1000
0.8039884433796068
```

In [88]:
```
plotdata = pd.DataFrame({"Model_Name":['LR','SVC','NB','randomforest','knn','Neural Network','CNN','LSTM'],
                         "scores": [lr_accuracy,svc_accuracy,NB_accuracy,randomforest_accuracy,knn_accuracy,NN_sc
plotdata1 = plotdata.sort_values(by="scores", ascending=False)
globals()['best_accuracy']  = plotdata1.Model_Name.to_numpy()[0]
best_accuracy
```

Out[88]: 'LR'

TESTING PYTHON APP

GMAIL APP

In [89]:
```
import os.path, pytz, json, selenium, requests, datetime, time, re, base64
import pandas as pd
import numpy as np
from statistics import mode
```

In [90]:
```
tz =pytz.timezone('europe/london')
```

In [91]:
```
from __future__ import print_function
import os.path
from googleapiclient.discovery import build
from google_auth_oauthlib.flow import InstalledAppFlow
from google.auth.transport.requests import Request
from google.oauth2.credentials import Credentials
```

In [92]:
```
import matplotlib as plt
import json
%matplotlib inline
```

In [93]:
```
SCOPES = ['https://www.googleapis.com/auth/gmail.readonly']

def main():
    """Shows basic usage of the Gmail API.
    Lists the user's Gmail labels.
    """
    creds = None
    if os.path.exists('token.json'):
        creds = Credentials.from_authorized_user_file('token.json', SCOPES)
    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            creds.refresh(Request())
        else:
            flow = InstalledAppFlow.from_client_secrets_file(
                'credentials.json', SCOPES)
            creds = flow.run_local_server(port=0)
        with open('token.json', 'w') as token:
            token.write(creds.to_json())

    service = build('gmail', 'v1', credentials=creds)

    results = service.users().messages().list(userId='me').execute()
```

```
    messages =results.get('messages')

    for message in messages:
        full_message = service.users().messages().get(userId='me',
                                                      id=message['id'],
                                                     format ='full').execute()
        for header in full_message['payload']['headers']:
            if header['name'] == 'Subject':
                content  = full_message['payload']['parts'][0]['body']['data']
                msg_body = base64.urlsafe_b64decode(content).decode('utf-8')
                quote_msg_body = (f'"{str(msg_body)}"')
                message_timestamp = int(full_message['internalDate'])/1000
                dt = datetime.datetime.fromtimestamp(message_timestamp, tz).isoformat()

                if time.time() - message_timestamp <1500000:
                    msg_headers = full_message['payload']['headers']
                    for k in range(len(msg_headers)):
                        if msg_headers[k]['name'] == 'From':
                            print(msg_headers[k]['value'])
                            print("\n")

                    print(header['value'])
                    print(msg_body)
                    print(globals()[best_accuracy].predict(count_vect.transform([quote_msg_body])))

main()
```

```
Leo Awe <leo.awe3@gmail.com>


Complaint Test 6
I cannot access my credit card information online.

Thanks,

Complaint Test 6

['Credit card']
```

In [ ]: