In [1]:

```
%%html
<style>
    .blue {
        background-color: #0074D9;
    }
    .green {
        background-color: #2ECC40;
    }

        .purple {
        background-color: #CC99FF;
    }
</style>
```

This Project is to create a model which determines the password strength

In [2]:

```
import os
import numpy as np
import pandas as pd
import sqlite3 as sq
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_selection import mutual_info_regression
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score, confusion
```

In [3]:

```
current_dir=os.getcwd()
db_file_name="password_data.sqlite"
```

In [4]:

```
file_path=os.path.join(current_dir,db_file_name)
```

In [5]:

```
file_path
```

Out[5]:

```
'/Users/amitnayan/Documents/Nayan/Learning/Password_Strength_Check_MLProject
/password_data.sqlite'
```

In [6]:

```
#sql_connection_object=sq.connect(r"C:\Users\nayanam\AppData\Roaming\Python\
sql_connection_object=sq.connect(file_path)
```

In [7]:

```
sql_connection_object.execute("select * from Users")
```

Out[7]:

```
<sqlite3.Cursor at 0x1418ae140>
```

In [8]:

```
data=pd.read_sql_query("select * from Users",sql_connection_object)
```

In [9]:

```
data_copy=data.copy()
```

In [10]:

```
data_copy.head(10)
```

Out[10]:

	index	password	strength
0	0	zxe870819	1
1	1	xw46454nr23l	1
2	2	soporte13	1
3	3	accounts6000webhost.com	2
4	4	c443balg	1
5	5	16623670p	1
6	6	yj9q3f8p	1
7	7	180ZIRUVIcuFERy	2
8	8	djredd09	1
9	9	yin172015	1

In [11]:

```
data_copy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 3 columns):
 #   Column    Non-Null Count   Dtype 
---  ------    --------------   ----- 
 0   index     100000 non-null  int64 
 1   password  100000 non-null  object
 2   strength  100000 non-null  int64 
dtypes: int64(2), object(1)
memory usage: 2.3+ MB
```

In [12]:

```
data_copy.shape
```

Out[12]:

```
(100000, 3)
```

Data cleaning steps
1. check for duplicates
2. check for missing values
3. check for irrelevant rows
4. check for irrelevant features
5. check if the data type of a feature is correct

1. checking for duplicates

In [13]:
```
data_copy.duplicated().sum()
```

Out[13]:
```
0
```

2. checking for missing values

In [14]:
```
data_copy.isna().sum()
```

Out[14]:
```
index       0
password    0
strength    0
dtype: int64
```

In [15]:
```
data_copy.isnull().any()
```

Out[15]:
```
index       False
password    False
strength    False
dtype: bool
```

In [16]:
```
data_copy.isnull().any().sum()
```

Out[16]:
```
0
```

3. checking for irrelevant feature

In [17]:
```
data_copy.columns
```

Out[17]:
```
Index(['index', 'password', 'strength'], dtype='object')
```

In [18]:
```
data_copy.head(5)
```

Out[18]:

	index	password	strength
0	0	zxe870819	1
1	1	xw46454nr23l	1
2	2	soporte13	1
3	3	accounts6000webhost.com	2
4	4	c443balg	1

In [19]:
```
"""index feature is not relavent so drop it."""
data_copy.drop(columns=['index'],axis=1,inplace=True)
```

In [20]:
```
data_copy.head(5)
```

Out[20]:

	password	strength
0	zxe870819	1
1	xw46454nr23l	1
2	soporte13	1
3	accounts6000webhost.com	2
4	c443balg	1

4. check if the data type of a feature is correct

In [21]:

```
data_copy.dtypes
```

Out[21]:

```
password    object
strength     int64
dtype: object
```

3. check for irrelevant rows

In [22]:

```
""" Check if any value in feature strength is negative. If its negative then

data_copy['strength'].unique()
```

Out[22]:

```
array([1, 2, 0])
```

Data Analysis

1. check how many passwords are only numeric

2. check how many passwords have only upper case characters

3. check how many passwords are alphanumeric

4. check how many passwords have title case characters

5. check how many passwords have some special charecters

1. check how many passwords are only numeric

In [23]:

```
data_copy[data_copy["password"].str.isnumeric()]
```

Out[23]:

	password	strength
12280	943801	0
14992	12345	0
20958	147856	0
21671	140290	0
23269	123987	0
28569	1233214	0
31329	0159456	0
32574	363761	0
37855	4524344	0
43648	5521597	0
45271	626262	0
52266	156651	0
58717	369	0
59619	151106	0
67723	1234	0
68106	1995151	0
68592	112233	0
69255	9562489	0
74938	12	0
77298	18731	0
86406	1050	0
86608	158491	0
94908	060415	0
96459	1	0
98122	6975818	0
98248	454545	0

In [24]:

```
data_copy[data_copy["password"].str.isnumeric()].shape
```

Out[24]:

```
(26, 2)
```

2. check how many passwords have only upper case characters

In [25]:

```
data_copy[data_copy["password"].str.isupper()]
```

Out[25]:

	password	strength
115	EYT63119	1
273	INSPIRON6	1
338	1A2S3D4F	1
367	13269123A	1
373	YAMAZAKI82	1
...	...	...
99590	V13000993J	1
99692	65925013ABC	1
99784	01EDD055	1
99893	1UPONYOU	1
99910	UNION1	0

1506 rows × 2 columns

In [26]:

```
data_copy[data_copy["password"].str.isupper()].shape
```

Out[26]:

```
(1506, 2)
```

3. check how many passwords are alphanumeric

In [27]:

```
data_copy[data_copy["password"].str.isalnum()]
```

Out[27]:

	password	strength
0	zxe870819	1
1	xw46454nr23l	1
2	soporte13	1
4	c443balg	1
5	16623670p	1
...	...	...
99995	obejofi215	1
99996	fmiopvxb64	1
99997	czvrbun38	1
99998	mymyxe430	1
99999	glqjhkxb467	1

97203 rows × 2 columns

In [28]:

```
data_copy[data_copy["password"].str.isalnum()].shape
```

Out[28]:

```
(97203, 2)
```

4. check how many passwords have title case characters

In [29]:

```
data_copy[data_copy["password"].str.istitle()]
```

Out[29]:

	password	strength
64	Hisanthoshjasika0	2
242	Therockrockbottom72	2
338	1A2S3D4F	1
367	13269123A	1
526	Csicskarozsika1	2
...	...	...
99168	1053815198M	1
99192	Alfranx05122023	2
99375	Kensington1956	2
99590	V13000993J	1
99654	94010Centuripe	2

932 rows × 2 columns

In [30]:

```
data_copy[data_copy["password"].str.istitle()].shape
```

Out[30]:

```
(932, 2)
```

5. check how many passwords have some special charecters
Created two functions to check if a string has special character. Can use anyone of those.

In [31]:

```
""" Function 1 """
def check_special_char_function1(df_row):
    for char in df_row:
        if char.isalpha() or char.isdigit():
            pass
        else:
            return True
    return False
```

In [32]:

```
import string
```

In [33]:

```
string.punctuation
```

Out[33]:

```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

In [34]:

```
"""Function 2 """
def check_special_char_function2(df_row):
    for char in df_row:
        if char in string.punctuation:
            return True
        else:
            pass
    return False
```

In [35]:

```
data_copy["password"].apply(check_special_char_function1)
```

Out[35]:

```
0        False
1        False
2        False
3         True
4        False
         ...
99995    False
99996    False
99997    False
99998    False
99999    False
Name: password, Length: 100000, dtype: bool
```

In [36]:

```
data_copy["password"].apply(check_special_char_function2)
```

Out[36]:

```
0        False
1        False
2        False
3         True
4        False
         ...
99995    False
99996    False
99997    False
99998    False
99999    False
Name: password, Length: 100000, dtype: bool
```

In [37]:

```
""" retrive rows where password has special character"""

data_copy[data_copy["password"].apply(check_special_char_function2)==True]
```

Out[37]:

	password	strength
3	accounts6000webhost.com	2
68	12463773800+	1
98	p.r.c.d.g.	1
145	cita-cita	1
180	karolina.susnina0U	2
...	...	...
99748	maiselis.com	1
99845	hosting4meze!@#	2
99954	semista_bakung15	2
99980	halflife2010!LEB	2
99988	lbhtrnjh@	1

2663 rows × 2 columns

In [38]:

```
data_copy[data_copy["password"].apply(check_special_char_function2)==True].s
```

Out[38]:

```
(2663, 2)
```

Feature Engineering

Create below features that will help to determine the strength of password

1. Length of password

2. Lower Case letter frequency

3. Upper Case letter frequency

4. Digit frequency

5. Special Character frequency

NB - Divide frequency by length of password to normalize data(ie 0<=frequency<=1) and avoid outlier.

1. Creating function to determine password length and add a feature "length"

In [39]:

```
def password_length(password):
    return len(password)
```

In [40]:

```
data_copy["length"]=data_copy["password"].apply(password_length)
```

In [41]:

```
data_copy.head(5)
```

Out[41]:

	password	strength	length
0	zxe870819	1	9
1	xw46454nr23l	1	12
2	soporte13	1	9
3	accounts6000webhost.com	2	23
4	c443balg	1	8

2. Creating function to determine Lower Case letter frequency and add a feaure called "lower_freq"

In [42]:

```
def password_lower_freq(password):
  return len([char for char in password if char.islower()])/len(password)
```

In [43]:

```
np.round(data_copy["password"].apply(password_lower_freq),3)
```

Out[43]:

```
0        0.333
1        0.417
2        0.778
3        0.783
4        0.625
         ...  
99995    0.700
99996    0.800
99997    0.778
99998    0.667
99999    0.727
Name: password, Length: 100000, dtype: float64
```

In [44]:

```
data_copy["lower_freq"]=np.round(data_copy["password"].apply(password_lower_
```

In [45]:

```
data_copy.head(5)
```

Out[45]:

	password	strength	length	lower_freq
0	zxe870819	1	9	0.333
1	xw46454nr23l	1	12	0.417
2	soporte13	1	9	0.778
3	accounts6000webhost.com	2	23	0.783
4	c443balg	1	8	0.625

2. Creating function to determine Upper Case letter frequency and add a feaure called "upper_freq"

In [46]:
```
def password_upper_freq(password):
   return len([char for char in password if char.isupper()])/len(password)
```

In [47]:
```
np.round(data_copy["password"].apply(password_upper_freq),3)
```

Out[47]:
```
0        0.0
1        0.0
2        0.0
3        0.0
4        0.0
        ...
99995    0.0
99996    0.0
99997    0.0
99998    0.0
99999    0.0
Name: password, Length: 100000, dtype: float64
```

In [48]:
```
data_copy["upper_freq"]=np.round(data_copy["password"].apply(password_upper_
```

In [49]:
```
data_copy.head(10)
```

Out[49]:

	password	strength	length	lower_freq	upper_freq
0	zxe870819	1	9	0.333	0.0
1	xw46454nr23l	1	12	0.417	0.0
2	soporte13	1	9	0.778	0.0
3	accounts6000webhost.com	2	23	0.783	0.0
4	c443balg	1	8	0.625	0.0
5	16623670p	1	9	0.111	0.0
6	yj9q3f8p	1	8	0.625	0.0
7	180ZIRUVIcuFERy	2	15	0.200	0.6
8	djredd09	1	8	0.750	0.0
9	yin172015	1	9	0.333	0.0

In [50]:
```
data_copy[data_copy["upper_freq"]!=0]
```

Out[50]:

	password	strength	length	lower_freq	upper_freq
7	180ZIRUVIcuFERy	2	15	0.200	0.600
14	crnogorac381PG	2	14	0.643	0.143
26	0Y1QKoDUzOAb83Zs	2	16	0.250	0.500
29	greatPERSON123	2	14	0.357	0.429
30	354OfaWaPemymIr	2	15	0.533	0.267
...	...	...	...	...	...
99950	KDys96jkyNQ46Dvh	2	16	0.438	0.312
99963	FvGoE3H3Xg3M4DOouE9k	2	20	0.300	0.450
99980	halflife2010!LEB	2	16	0.500	0.188
99985	IYdKYnTM2NwvRUhA	2	16	0.312	0.625
99990	hxymUnjM1NghqCOE	2	16	0.562	0.375

13012 rows × 5 columns

4. Creating function to determine numeric letter frequency and add a feaure called "digit_freq"

In [51]:

```
def password_digit_freq(password):
  return len([char for char in password if char.isdigit()])/len(password)
```

In [52]:

```
np.round(data_copy["password"].apply(password_digit_freq),3)
```

Out[52]:

```
0        0.667
1        0.583
2        0.222
3        0.174
4        0.375
         ...
99995    0.300
99996    0.200
99997    0.222
99998    0.333
99999    0.273
Name: password, Length: 100000, dtype: float64
```

In [53]:

```
data_copy["digit_freq"]=np.round(data_copy["password"].apply(password_digit_
```

In [54]:

```
data_copy.head(5)
```

Out[54]:

	password	strength	length	lower_freq	upper_freq	digit_freq
0	zxe870819	1	9	0.333	0.0	0.667
1	xw46454nr23l	1	12	0.417	0.0	0.583
2	soporte13	1	9	0.778	0.0	0.222
3	accounts6000webhost.com	2	23	0.783	0.0	0.174
4	c443balg	1	8	0.625	0.0	0.375

5. Creating function to determine special character frequency and add a feaure called "special_char_freq"

In [55]:

```
import string
def password_special_char_freq(password):
    return len([char for char in password if char in string.punctuation])/le
```

In [56]:

```
np.round(data_copy["password"].apply(password_special_char_freq),3)
```

Out[56]:

```
0        0.000
1        0.000
2        0.000
3        0.043
4        0.000
         ...
99995    0.000
99996    0.000
99997    0.000
99998    0.000
99999    0.000
Name: password, Length: 100000, dtype: float64
```

In [57]:

```
data_copy["special_char_freq"]=np.round(data_copy["password"].apply(password
```

In [58]:

```
data_copy.head(5)
```

Out[58]:

	password	strength	length	lower_freq	upper_freq	digit_freq	special_ch
0	zxe870819	1	9	0.333	0.0	0.667	
1	xw46454nr23l	1	12	0.417	0.0	0.583	
2	soporte13	1	9	0.778	0.0	0.222	
3	accounts6000webhost.com	2	23	0.783	0.0	0.174	
4	c443balg	1	8	0.625	0.0	0.375	

In [59]:

```
data_copy[data_copy["special_char_freq"]!=0]
```

Out[59]:

	password	strength	length	lower_freq	upper_freq	digit_freq	spec
3	accounts6000webhost.com	2	23	0.783	0.000	0.174	
68	12463773800+	1	12	0.000	0.000	0.917	
98	p.r.c.d.g.	1	10	0.500	0.000	0.000	
145	cita-cita	1	9	0.889	0.000	0.000	
180	karolina.susnina0U	2	18	0.833	0.056	0.056	
...	...	...	...	...	...	...	
99748	maiselis.com	1	12	0.917	0.000	0.000	
99845	hosting4meze!@#	2	15	0.733	0.000	0.067	
99954	semista_bakung15	2	16	0.812	0.000	0.125	
99980	halflife2010!LEB	2	16	0.500	0.188	0.250	
99988	lbhtrnjh@	1	9	0.889	0.000	0.000	

2663 rows × 7 columns

In [60]:

```
data_copy[(data_copy["special_char_freq"]!=0) & (data_copy["digit_freq"]==0.
```

Out[60]:

	password	strength	length	lower_freq	upper_freq	digit_freq	spe
3	accounts6000webhost.com	2	23	0.783	0.000	0.174	
23244	verifyacc2013@gmail.com	2	23	0.739	0.000	0.174	
30020	Dabgdanrizky571n6>cinta	2	23	0.739	0.043	0.174	
38000	qwerty4000webhost%ASDF$	2	23	0.565	0.174	0.174	
45390	galcivar8294@utm.edu.ec	2	23	0.696	0.000	0.174	
52284	1963savitamore@ramtirth	2	23	0.783	0.000	0.174	
96521	mifamiliaeslamejor-1984	2	23	0.783	0.000	0.174	

Descristive Statistics

check min, max, mean, median values group by strength, lower_freq, upper_freq, digit_freq and special_char_freq

In [61]:

```
data_copy[['length','strength']].groupby(['strength']).agg(['min','max','mea
```

Out[61]:

	length			
	min	max	mean	median
strength				
0	1	7	6.550947	7.0
1	8	13	9.611074	9.0
2	14	220	15.953421	16.0

In [62]:

```
data_copy[['strength','lower_freq']].groupby(['strength']).agg(['min','max',
```

Out[62]:

	lower_freq			
	min	max	mean	median
strength				
0	0.0	1.000	0.708050	0.714
1	0.0	0.923	0.630067	0.667
2	0.0	0.917	0.424679	0.400

In [63]:

```
data_copy[['strength','upper_freq']].groupby(['strength']).agg(['min','max',
```

Out[63]:

	upper_freq			
	min	max	mean	median
strength				
0	0.0	1.000	0.012872	0.000
1	0.0	0.923	0.007915	0.000
2	0.0	0.889	0.367633	0.429

In [64]:

```
data_copy[['strength','digit_freq']].groupby(['strength']).agg(['min','max',
```

Out[64]:

	digit_freq			
	min	max	mean	median
strength				
0	0.0	1.000	0.275383	0.286
1	0.0	0.923	0.360123	0.333
2	0.0	0.895	0.193796	0.188

In [65]:

```python
data_copy[['strength','special_char_freq']].groupby(['strength']).agg(['min'
```

Out[65]:

	special_char_freq			
	min	**max**	**mean**	**median**
strength				
0	0.0	1.000	0.003195	0.0
1	0.0	0.818	0.001729	0.0
2	0.0	0.741	0.013602	0.0

In [66]:

```python
data_copy[['length','strength','lower_freq','upper_freq','digit_freq','speci
```

Out[66]:

	length				lower_freq				u		
	min	**max**	**mean**	**median**	**min**	**max**	**mean**	**median**	**min**	**max**	**mea**
strength											
0	1	7	6.550947	7.0	0.0	1.000	0.708050	0.714	0.0	1.000	0.01287
1	8	13	9.611074	9.0	0.0	0.923	0.630067	0.667	0.0	0.923	0.00791
2	14	220	15.953421	16.0	0.0	0.917	0.424679	0.400	0.0	0.889	0.36763

In [67]:

```python
data_copy.columns
```

Out[67]:

```
Index(['password', 'strength', 'length', 'lower_freq', 'upper_freq',
       'digit_freq', 'special_char_freq'],
      dtype='object')
```

In [68]:

```python
features=['length','lower_freq', 'upper_freq',
        'digit_freq', 'special_char_freq']
```

In [69]:

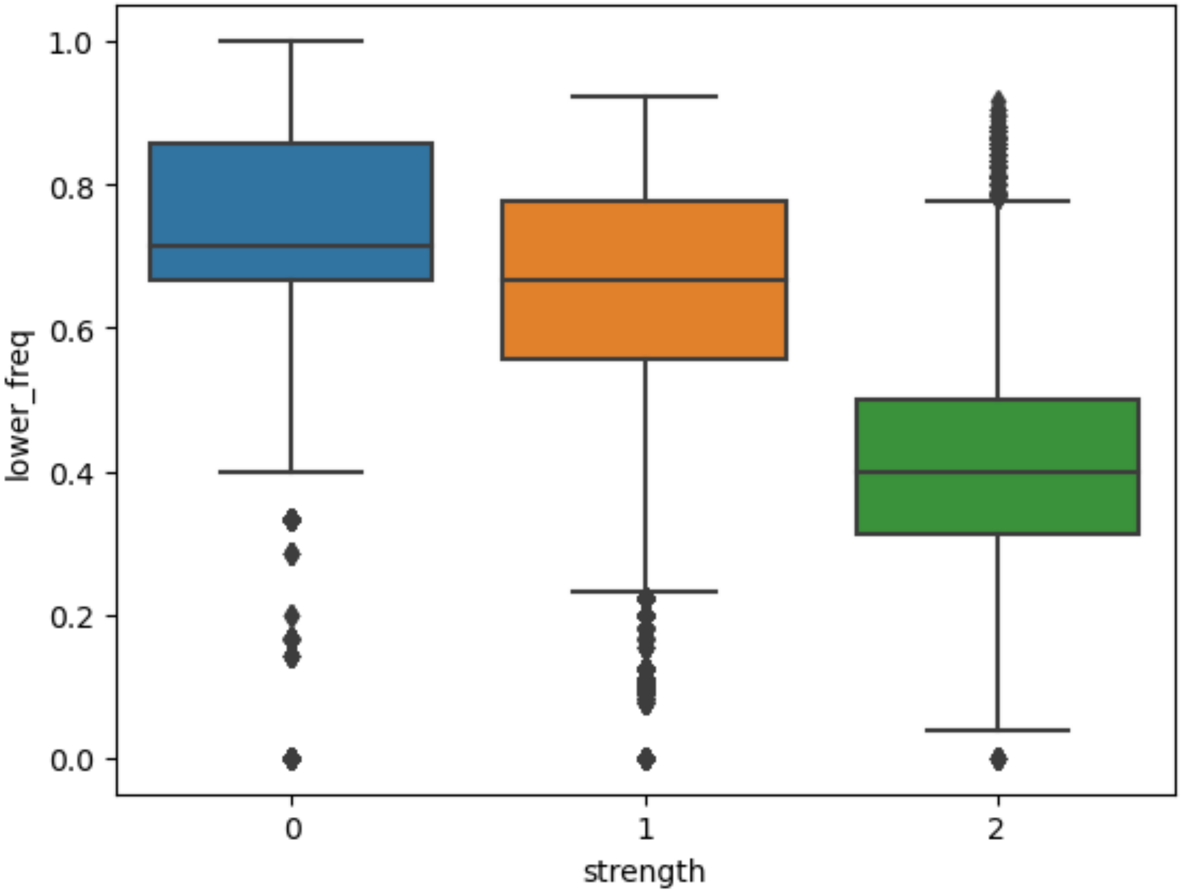
```python
for feature in features:
    print( data_copy[['strength',feature]].groupby('strength').agg(['min','ma
         length
            min  max       mean median
strength
0             1    7   6.550947    7.0
1             8   13   9.611074    9.0
2            14  220  15.953421   16.0
         lower_freq
                min    max      mean median
strength
0               0.0  1.000  0.708050  0.714
1               0.0  0.923  0.630067  0.667
2               0.0  0.917  0.424679  0.400
         upper_freq
                min    max      mean median
strength
0               0.0  1.000  0.012872  0.000
1               0.0  0.923  0.007915  0.000
2               0.0  0.889  0.367633  0.429
         digit_freq
                min    max      mean median
strength
0               0.0  1.000  0.275383  0.286
1               0.0  0.923  0.360123  0.333
2               0.0  0.895  0.193796  0.188
         special_char_freq
                       min    max      mean median
strength
0                      0.0  1.000  0.003195    0.0
1                      0.0  0.818  0.001729    0.0
2                      0.0  0.741  0.013602    0.0
```

In [70]:

```
sns.boxplot(x=data_copy['strength'],y=data_copy['lower_freq'])
```

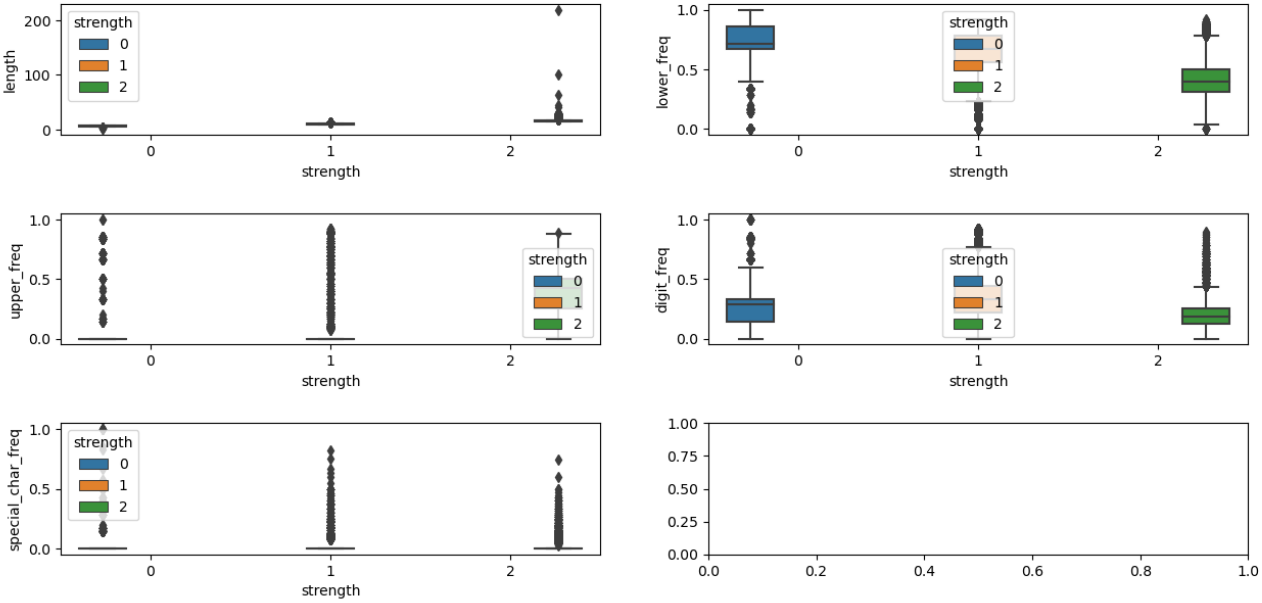
Out[70]:

```
<Axes: xlabel='strength', ylabel='lower_freq'>
```



In [71]:

```
fig,((ax11,ax12),(ax21,ax22),(ax31,ax32))=plt.subplots(3,2,figsize=(15,7))
sns.boxplot(x='strength',y='length', hue='strength',ax=ax11, data=data_copy)
sns.boxplot(x='strength',y='lower_freq',hue='strength',ax=ax12, data=data_cc
sns.boxplot(x='strength',y='upper_freq',hue='strength',ax=ax21, data=data_cc
sns.boxplot(x='strength',y='digit_freq',hue='strength',ax=ax22, data=data_cc
sns.boxplot(x='strength',y='special_char_freq',hue='strength',ax=ax31, data=
plt.subplots_adjust(hspace=0.6)
```



Calculate Feature's Mutual Information (MI) score to determine Which features are important

In [72]:
```
y=data_copy['strength']
X=data_copy.drop(columns=['strength','password'],axis=1)
```

In [73]:
```
mi_score=mutual_info_regression(X,y)
```

In [74]:
```
mi_score
```

Out[74]:
```
array([0.75460901, 0.5006334 , 0.31014861, 0.49814378, 0.03158723])
```

In [75]:
```
mi_score_df=pd.DataFrame(mi_score, index=X.columns,columns=['Feature_Importa
```

In [76]:
```
mi_score_df.sort_values(by= 'Feature_Importance_Score/MI Score', ascending=F
```

Out[76]:

	Feature_Importance_Score/MI Score
length	0.754609
lower_freq	0.500633
digit_freq	0.498144
upper_freq	0.310149
special_char_freq	0.031587

Univariate Analysis to determine Which features are important

In [77]:

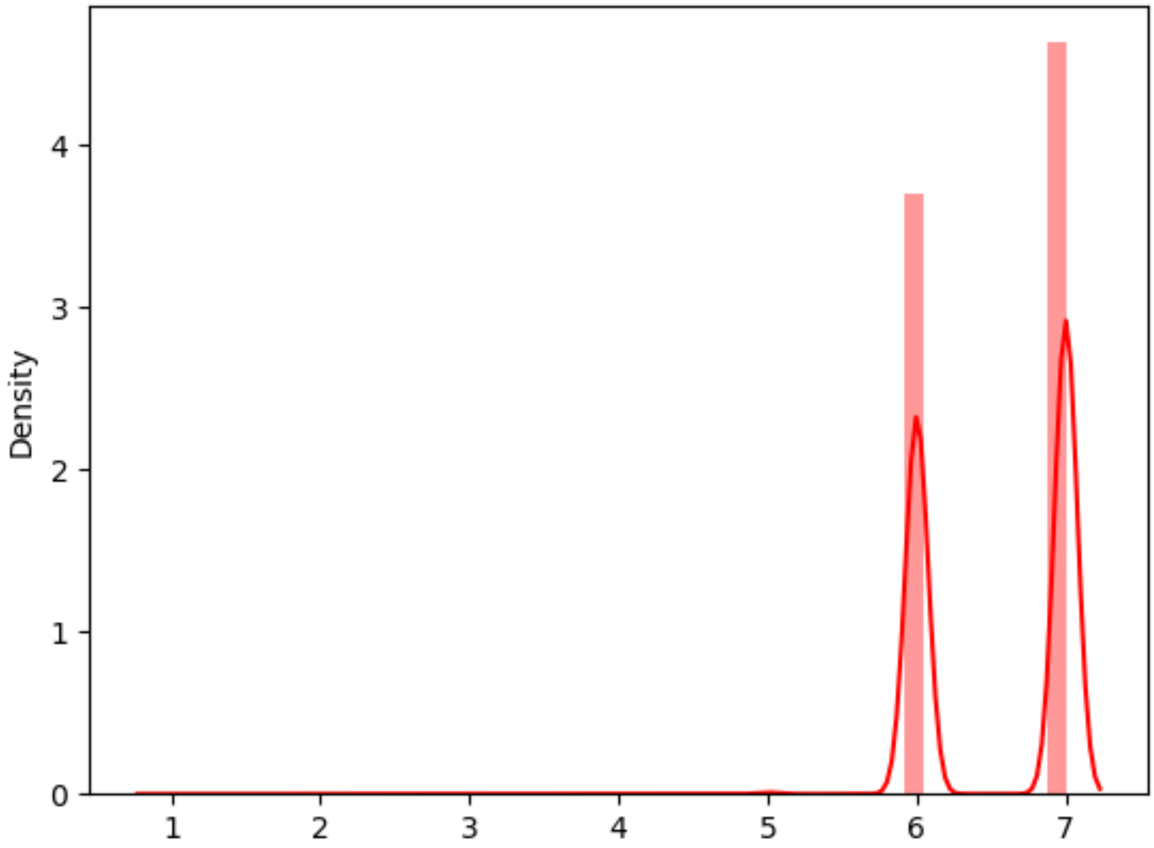
```
from warnings import filterwarnings
filterwarnings("ignore")
```

In [78]:

```
sns.distplot(x=data_copy[data_copy['strength']==0]['length'],color='red')
```

Out[78]:

```
<Axes: ylabel='Density'>
```

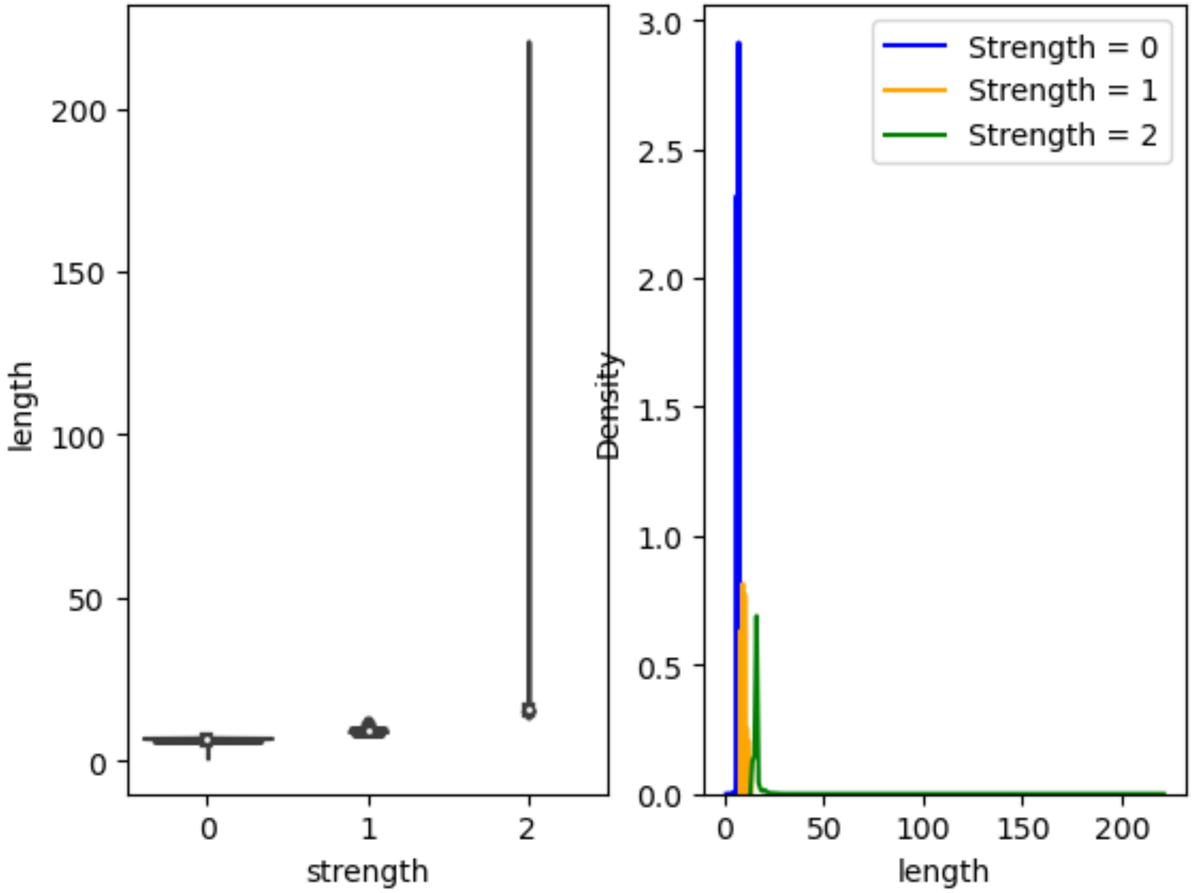


In [79]:

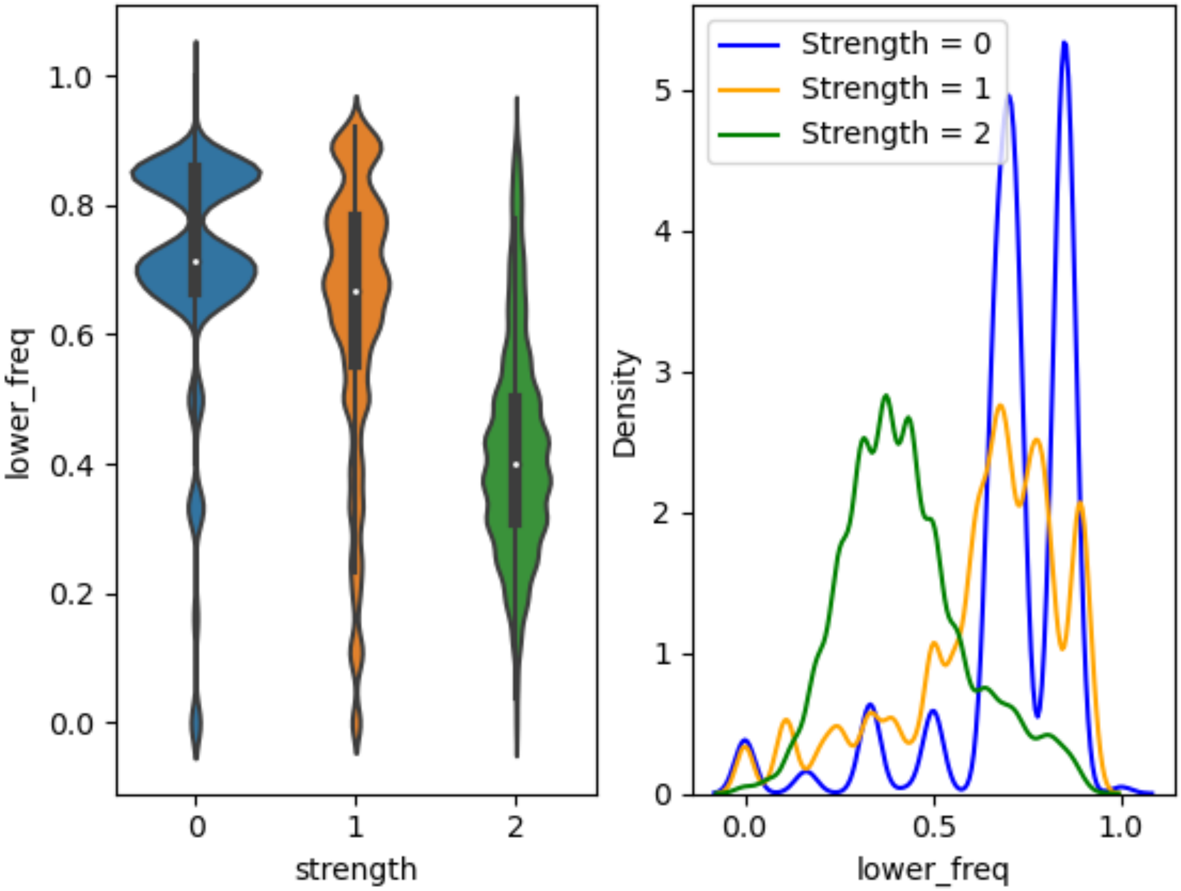
```
def get_plot(df,feature):
    #plt.figure(figsize=(10,10))
    fig, (ax11,ax12)=plt.subplots(1,2)
    sns.violinplot(x='strength',y=feature,data=df,ax=ax11)
    sns.distplot(x=df[df['strength']==0][feature],ax=ax12,color='blue', labe
    sns.distplot(x=df[df['strength']==1][feature],ax=ax12,color='orange', la
    sns.distplot(x=df[df['strength']==2][feature],ax=ax12,color='green', lab
    plt.subplots_adjust(hspace=0.6)
    plt.legend()
    plt.show()
```

Looking at below graphs for different features, we concluded we have less overlapping in case of length feature and lower_freq feature.
Also MI score for features length and lower_freq are greater than 50% and hence these two features are important to consider.

```
In [80]: get_plot(data_copy,'length')
```

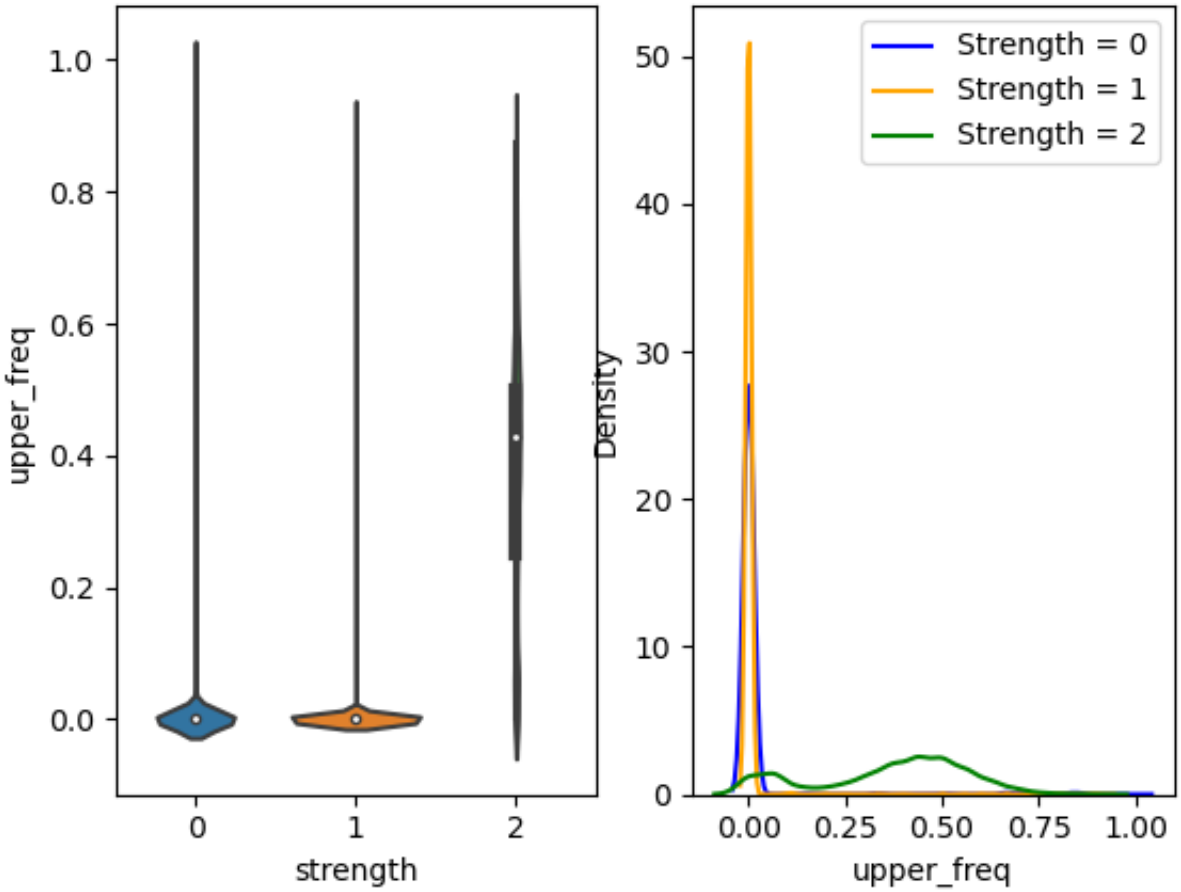


```
In [81]: get_plot(data_copy,'lower_freq')
```



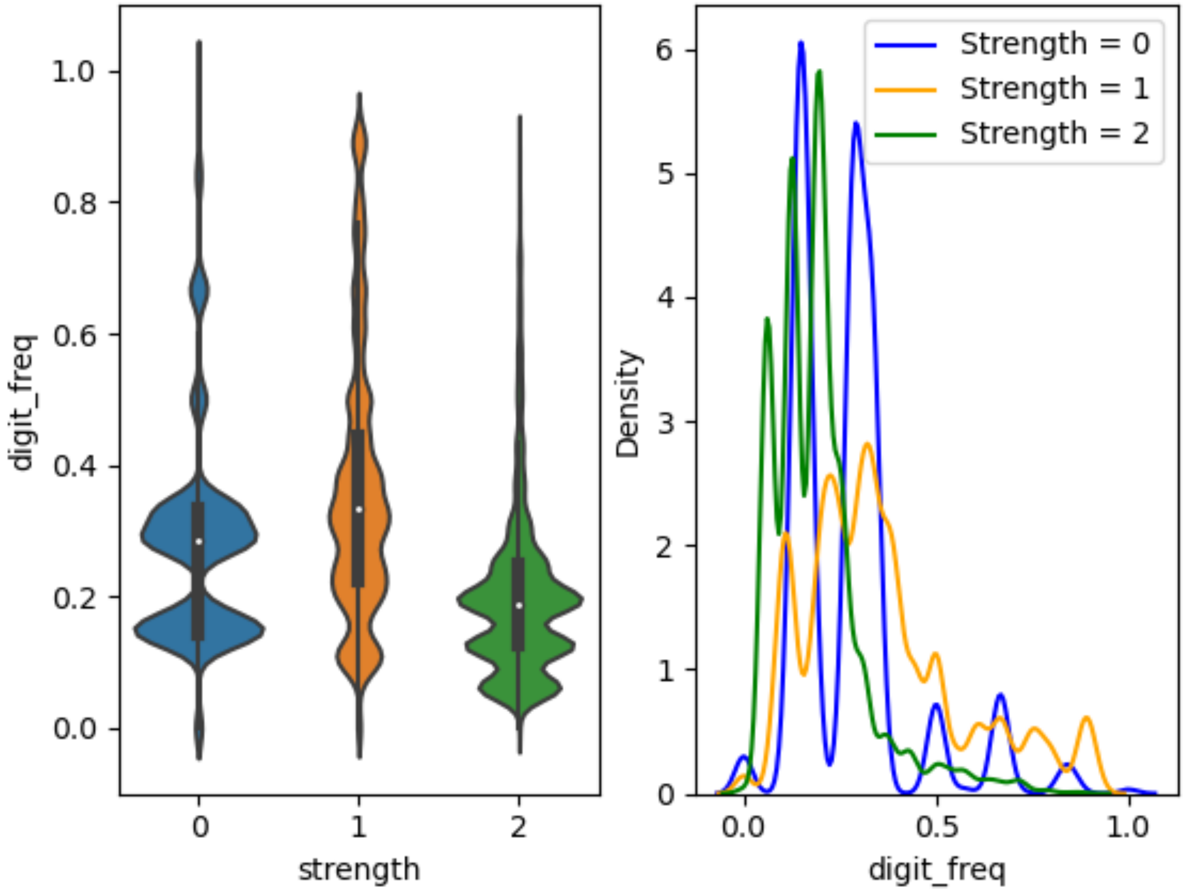
In [82]:

```
get_plot(data_copy,'upper_freq')
```



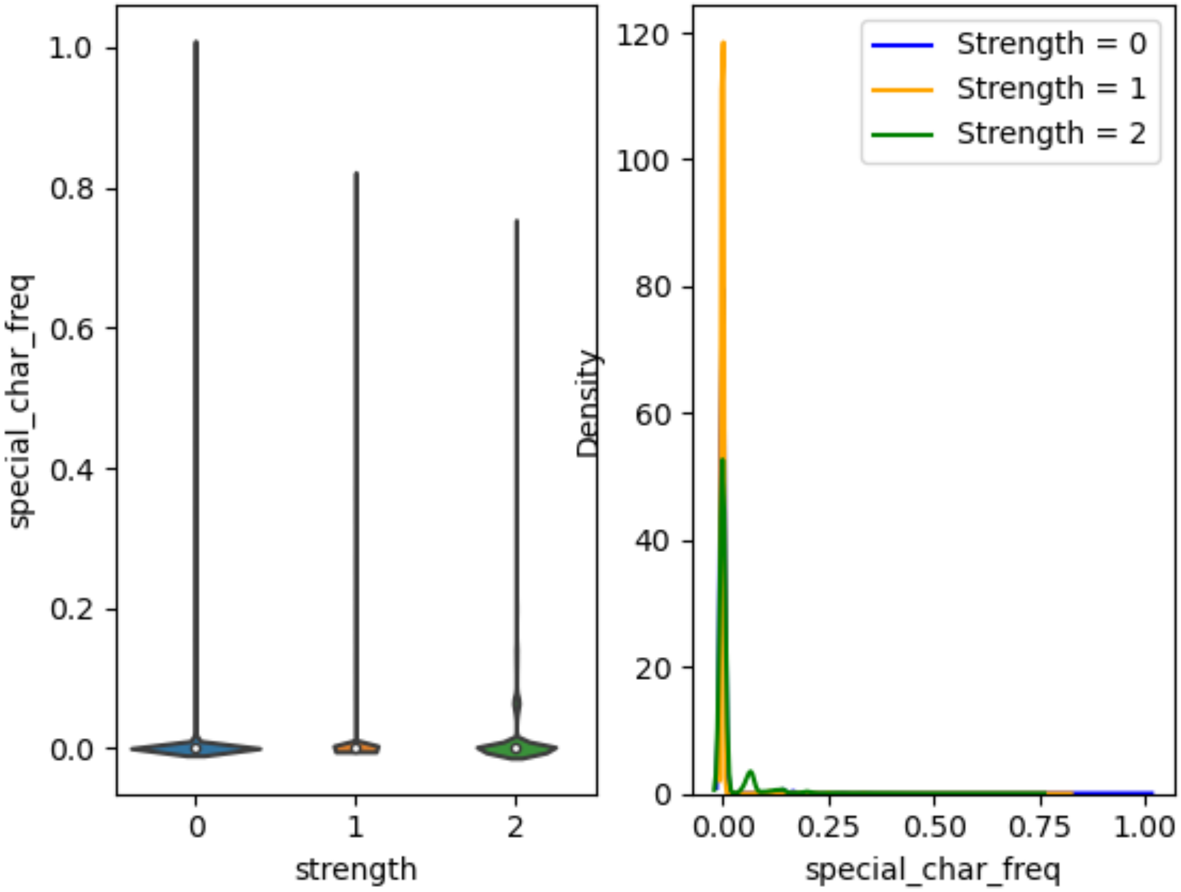
In [83]:

```
get_plot(data_copy,'digit_freq')
```



In [84]:

```
get_plot(data_copy,'special_char_freq')
```



Feature Engineering. Check if any feature need to be converted into numerical. Here password feature is categorical and needs conversion/encoding before passing to model. We will convert password feature into TF-IDF matrix feature using TfidfVectorizer

In [85]:

```
data_copy.head()
```

Out[85]:

	password	strength	length	lower_freq	upper_freq	digit_freq	special_
0	zxe870819	1	9	0.333	0.0	0.667	
1	xw46454nr23l	1	12	0.417	0.0	0.583	
2	soporte13	1	9	0.778	0.0	0.222	
3	accounts6000webhost.com	2	23	0.783	0.0	0.174	
4	c443balg	1	8	0.625	0.0	0.375	

Shuffle the data using sample() from padas dataframe

In [86]:

```
# Shuffle the data using pandas df sample() function
data_copy_frame=data_copy.sample(frac=1)
```

In [87]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [88]:
```
vectorizer=TfidfVectorizer(analyzer='char')
```

In [89]:
```
x=list(data_copy_frame['password'])
```

In [90]:
```
X=vectorizer.fit_transform(raw_documents=x)
```

fit_transform converted password feature to a sparse matrix using 99 dimensions and the no of rows are same ie 100000. See below the comparison

In [91]:
```
X
```

Out[91]:
```
<100000x99 sparse matrix of type '<class 'numpy.float64'>'
	with 842571 stored elements in Compressed Sparse Row format>
```

In [92]:
```
# dimension is 99 ie every password is represented using 99 dimensions
X.shape
```

Out[92]:
```
(100000, 99)
```

In [93]:
```
#dimension is 1 ie every password is represented using 1 dimension
data_copy['password'].shape
```

Out[93]:
```
(100000,)
```

In [94]:
```
X.toarray()
```

Out[94]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

In [95]:
```
# accessing sparse matrix(capital X) value at zeroth index which corresponds
X.toarray()[0]
```

Out[95]:

```
array([0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.56366446, 0.        ,
       0.21003369, 0.        , 0.        , 0.21226811, 0.20272964,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.13963909, 0.        , 0.43743822,
       0.21032528, 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.17158389, 0.23054295, 0.26449466, 0.        ,
       0.        , 0.19801902, 0.        , 0.        , 0.24208309,
       0.        , 0.21577512, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        ])
```

In [96]:

```
data_copy_frame['password']
```

Out[96]:

```
69949          d3CPCWTY3OA3Q589
34122              m1i2a3m4o5r6i
29744                     hrs1da
24228              pem5571bdzpem
55518      juanki968@hotmail.com
                   ...
35329          AH3p09DE5MgZreQK
80078                   mf123456
30597          4RrWfYzg5MwNk4zQ
57265               mahekricha98
77753                    ninja12
Name: password, Length: 100000, dtype: object
```

There are 99 char/dimension that is sparse matrix's feature and is used to encode/convert each character of passoword.

In [97]:

```
vectorizer.get_feature_names_out()
```

Out[97]:

```
array(['\x04', '\x06', '\x08', '\x0e', '\x10', '\x11', '\x17', ' ', '!',
       '#', '$', '%', '&', '(', ')', '*', '+', '-', '.', '/', '0', '1',
       '2', '3', '4', '5', '6', '7', '8', '9', ';', '<', '=', '>', '?',
       '@', '[', '\\', ']', '^', '_', '`', 'a', 'b', 'c', 'd', 'e', 'f',
       'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's',
       't', 'u', 'v', 'w', 'x', 'y', 'z', '{', '|', '}', '~', '¡', '¨',
       '°', '±', '³', '´', 'µ', '·', 'ß', 'à', 'á', 'ä', 'æ', 'ç', 'é',
       'ê', 'í', 'ñ', 'ó', 'õ', 'ö', '÷', 'ú', 'ü', 'ý', 'þ', '›'],
      dtype=object)
```

In [98]:

```
df2=pd.DataFrame(X.toarray(),columns=vectorizer.get_feature_names_out())
```

In [99]:

```
df2
```

Out[99]:

	☒	☒	▯	☒	☒	☒	☒		!	#	...	ñ	ó	õ	ö	÷	ú	
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
99995	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.
99996	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.
99997	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.
99998	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.
99999	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.

100000 rows × 99 columns

**during our analysis above to determine which feature is important, we colcluded feature 'length' and 'lower_freq' are important.
So include both the features in dataframe df2 as well.**

In [100…

```
df2['length']=data_copy_frame['length']
df2['lower_freq']=data_copy_frame['lower_freq']
```

In [101…

```
df2
```

Out[101]:

	☒	☒	▯	☒	☒	☒	☒		!	#	...	õ	ö	÷	ú	ü	ý
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
99995	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
99996	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
99997	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
99998	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
99999	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0

100000 rows × 101 columns

Model Building
We have below features as independent variables in data frame df2

1. password - represented as different characters/features of sparse matrix
2. length
3. lower_freq

We need to predict password's strength so feature strength is dependent variable
Since we have discreate dependent variable, its a classification problem and we will use classification ML model ie LogisticRegression

Split train and test data

In [102…
```
from sklearn.model_selection import train_test_split
```

In [103…
```
y=data_copy_frame['strength']
X=df2
```

In [104…
```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.20)
```

In [105…
```
X_train.shape
```

Out[105]:
```
(80000, 101)
```

In [106…

```
y_train.shape
```

Out[106]:

```
(80000,)
```

train model and predict password strength on test data

In [107…

```
from sklearn.linear_model import LogisticRegression
```

In [108…

```
# parameter multi_class is set to "multinomial" because our dependent varial
ml_classification_model=LogisticRegression(multi_class="multinomial")
```

In [109…

```
ml_classification_model.fit(X_train,y_train)
```

Out[109]:

```
▾ LogisticRegression
LogisticRegression(multi_class='multinomial')
```

In [110…

```
y_predict=ml_classification_model.predict(X_test)
```

In [111…

```
y_predict
```

Out[111]:

```
array([2, 1, 1, ..., 0, 1, 1])
```

In [112…

```
pd.DataFrame(y_predict,columns=['Strength'])
```

Out[112]:

	Strength
0	2
1	1
2	1
3	1
4	1
...	...
19995	1
19996	2
19997	0
19998	1
19999	1

20000 rows × 1 columns

Check how many paswords from testing data sets are of different strengths

```
In [113…  from collections import Counter
```

```
In [114…  Counter(y_predict)
```

```
Out[114]:  Counter({2: 1644, 1: 17189, 0: 1167})
```

calculate Accuracy score for predicted password strength based on test data

```
In [115…  from sklearn.metrics import classification_report, accuracy_score, confusio
```

```
In [116…  accuracy_score(y_test, y_predict)
```

```
Out[116]:  0.79435
```

```
In [117…  confusion_matrix(y_test, y_predict)
```

```
Out[117]:  array([[  602,  2134,     4],
                  [  487, 13955,   310],
                  [   78,  1100,  1330]])
```

```
In [118…  print(classification_report(y_test, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.52      0.22      0.31      2740
           1       0.81      0.95      0.87     14752
           2       0.81      0.53      0.64      2508

    accuracy                           0.79     20000
   macro avg       0.71      0.57      0.61     20000
weighted avg       0.77      0.79      0.77     20000
```

Define a function to input password from user and call our ML model to check if it is week or strong

In [119…

```
def check_passoword_strength():
    password=input("Enter a password :")
    sample_array=np.array([password])
    sample_matrix=vectorizer.transform(sample_array)
    target_matrix=np.append(sample_matrix.toarray(),(password_length(passwo
    strength=ml_classification_model.predict(target_matrix)
    if strength==0:
        print("Your Password Strength :{}\n Password is week.".format(stren
    elif strength==1:
        print("Your Password Strength :{}\n Password is Normal.".format(str
    else:
        print("Your Password Strength :{}\n Password is Strong.".format(str
```

In [120…

```
check_passoword_strength()
```

```
Enter a password :234hTbYkP@3
Your Password Strength :2
 Password is Strong.
```

In [ ]: