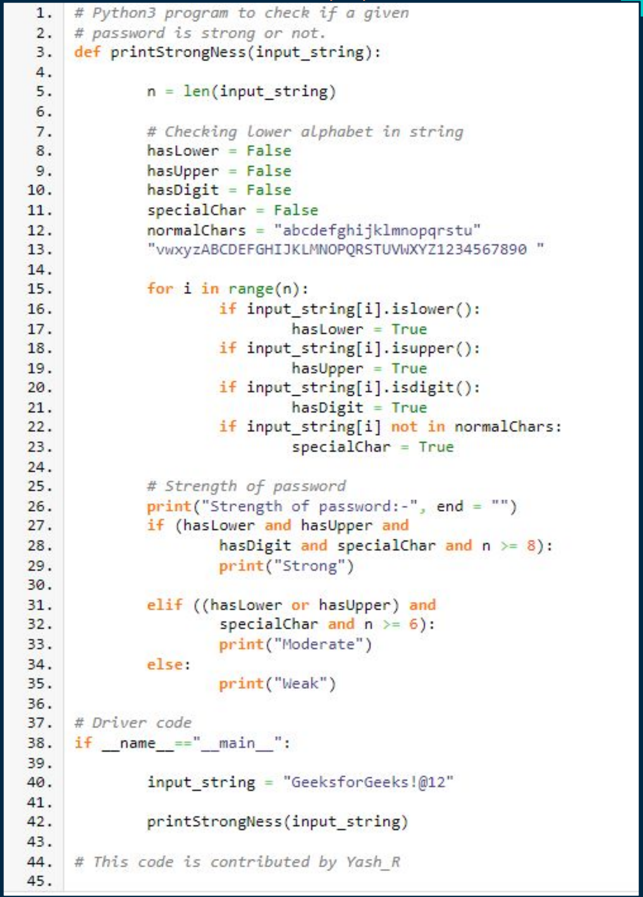
PASSWORD STRENGTH CLASSIFICATION

BY: RAY THIBODEAUX
& ABIOLA ADEGBOYEGA

The Problem

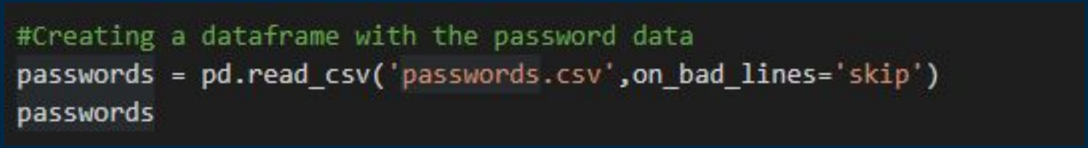
Password Strength Classification

- Improve password strength classification
- Normal password meters - conditional
- Machine learning strategy - mathematical model

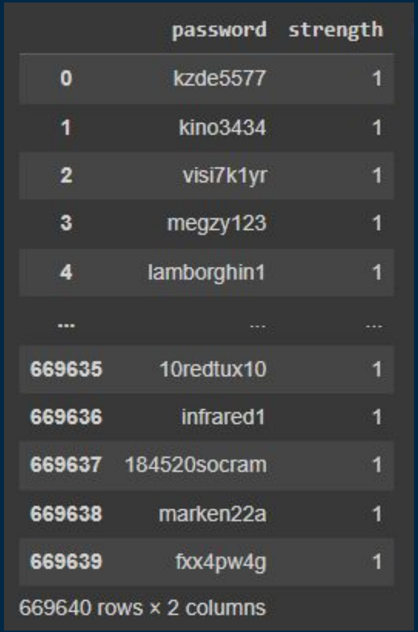


```python
# Python3 program to check if a given
# password is strong or not.
def printStrongNess(input_string):

        n = len(input_string)

        # Checking lower alphabet in string
        hasLower = False
        hasUpper = False
        hasDigit = False
        specialChar = False
        normalChars = "abcdefghijklmnopqrstu"
        "vwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890 "

        for i in range(n):
                if input_string[i].islower():
                        hasLower = True
                if input_string[i].isupper():
                        hasUpper = True
                if input_string[i].isdigit():
                        hasDigit = True
                if input_string[i] not in normalChars:
                        specialChar = True

        # Strength of password
        print("Strength of password:-", end = "")
        if (hasLower and hasUpper and
                hasDigit and specialChar and n >= 8):
                print("Strong")

        elif ((hasLower or hasUpper) and
                specialChar and n >= 6):
                print("Moderate")
        else:
                print("Weak")

# Driver code
if __name__=="__main__":

        input_string = "GeeksforGeeks!@12"

        printStrongNess(input_string)

# This code is contributed by Yash_R

```

The Data



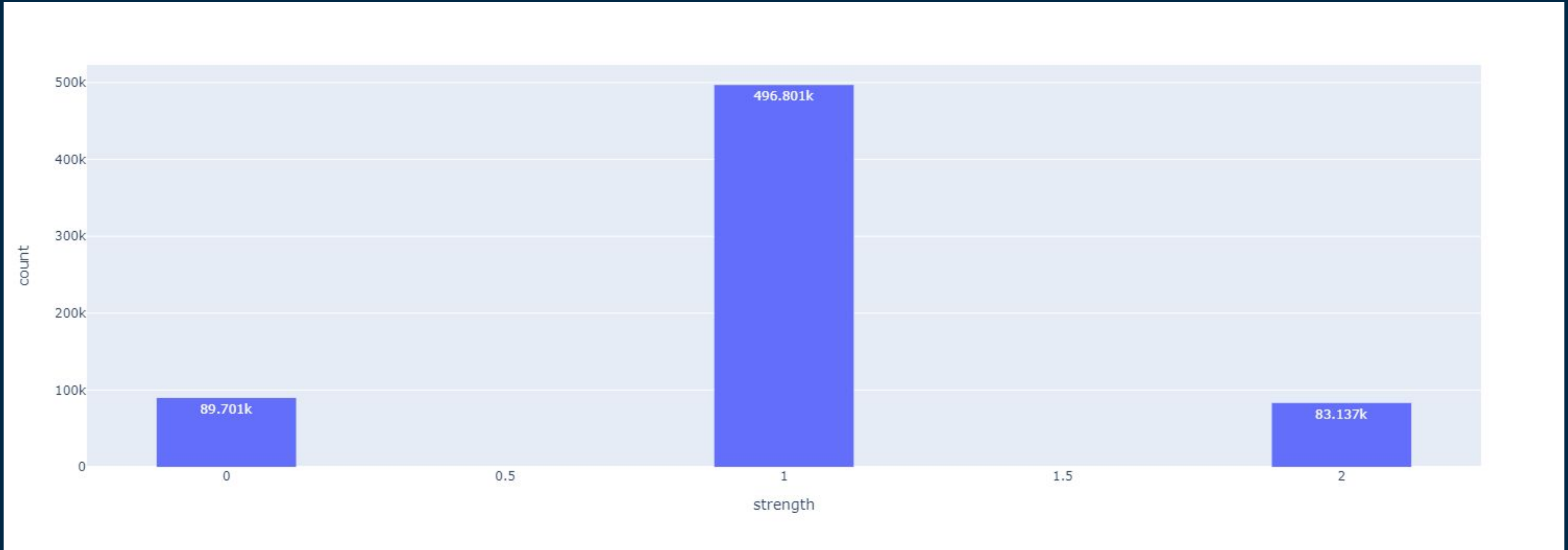
```python
#Creating a dataframe with the password data
passwords = pd.read_csv('passwords.csv',on_bad_lines='skip')
passwords
```



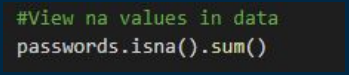
	password	strength
0	kzde5577	1
1	kino3434	1
2	visi7k1yr	1
3	megzy123	1
4	lamborghin1	1
...	...	...
669635	10redtux10	1
669636	infrared1	1
669637	184520socram	1
669638	marken22a	1
669639	fxx4pw4g	1

669640 rows × 2 columns

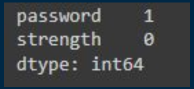
The Data


500k
400k
300k
200k
100k
0
count
496.801k
89.701k
83.137k
0
0.5
1
1.5
2
strength

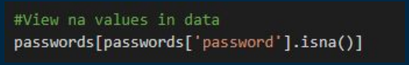
Data Cleaning



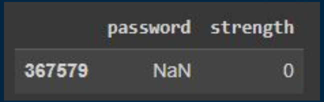
```
#View na values in data
passwords.isna().sum()
```



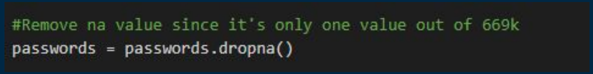
```
password    1
strength    0
dtype: int64
```



```
#View na values in data
passwords[passwords['password'].isna()]
```



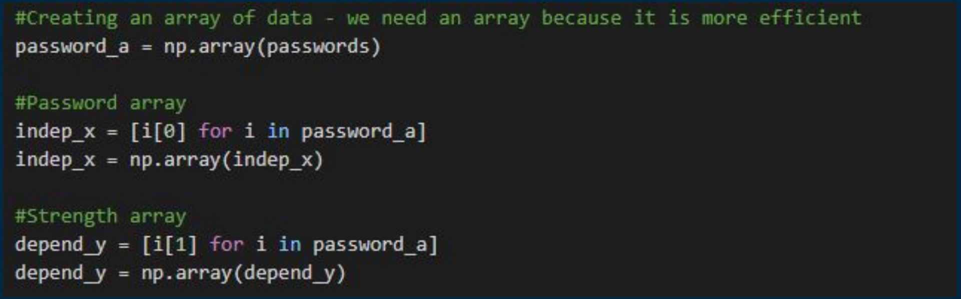
```
        password  strength
367579       NaN         0
```



```
#Remove na value since it's only one value out of 669k
passwords = passwords.dropna()
```

Data Pre-Processing - Tokenization

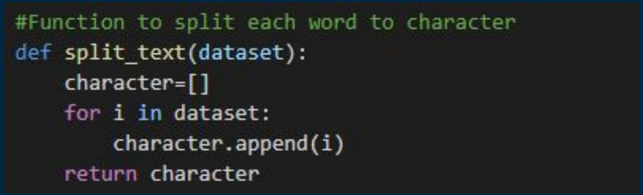
- Tokenization - splitting each word into character so the computer can vectorize the data.
    - Create an array of data (passwords and strengths)



```
#Creating an array of data - we need an array because it is more efficient
password_a = np.array(passwords)

#Password array
indep_x = [i[0] for i in password_a]
indep_x = np.array(indep_x)

#Strength array
depend_y = [i[1] for i in password_a]
depend_y = np.array(depend_y)
```

Data Pre-Processing - Tokenization

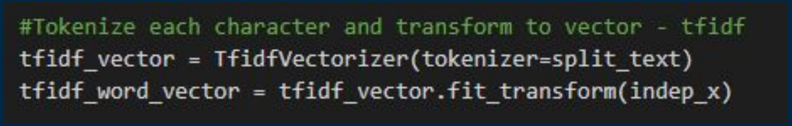
- Tokenization - splitting each word into character so the computer can vectorize the data.
    - Create a function to split each password into characters



```
#Function to split each word to character
def split_text(dataset):
    character=[]
    for i in dataset:
        character.append(i)
    return character
```

Data Pre-Processing - Tokenization

- Tokenization - splitting each word into character so the computer can vectorize the data.
    - Pass the function into the tokenizer so the method vectorizes characters and not the full password



```
#Tokenize each character and transform to vector - tfidf
tfidf_vector = TfidfVectorizer(tokenizer=split_text)
tfidf_word_vector = tfidf_vector.fit_transform(indep_x)
```

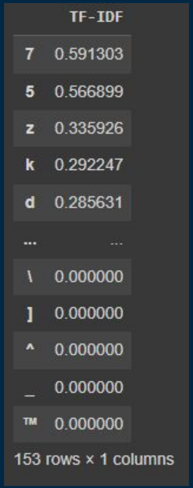
Data Pre-Processing - Side Note

- Other methods of natural language processing aren’t used because we want the purest form of the passwords.
  - Removing special characters
  - Removing numbers
  - All lowercase text
  - Lemmatization - relating words to their root format. (ex. kept to keep)

https://towardsdatascience.com/nlp-text-preprocessing-a-practical-guide-and-template-d80874676e79

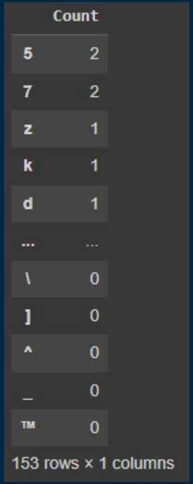
Password Vectorization

- TF-IDF (Term Frequency Inverse Document Frequency)
  - Overall document weight
- Count
  - Frequency compared to the index



	TF-IDF
7	0.591303
5	0.566899
z	0.335926
k	0.292247
d	0.285631
...	...
\	0.000000
]	0.000000
^	0.000000
_	0.000000
™	0.000000

153 rows × 1 columns



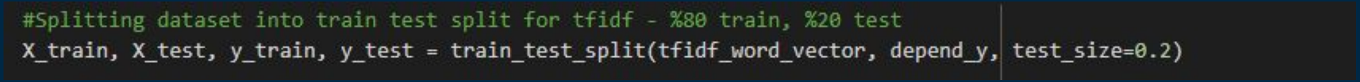
	Count
5	2
7	2
z	1
k	1
d	1
...	...
\	0
]	0
^	0
_	0
™	0

153 rows × 1 columns

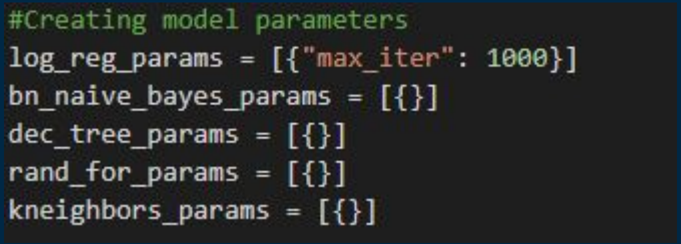
Password Vectorization - Problems

- Some vectorization models take advantage of the word meaning.
  - GloVe
  - FastText
  - Word2Vec

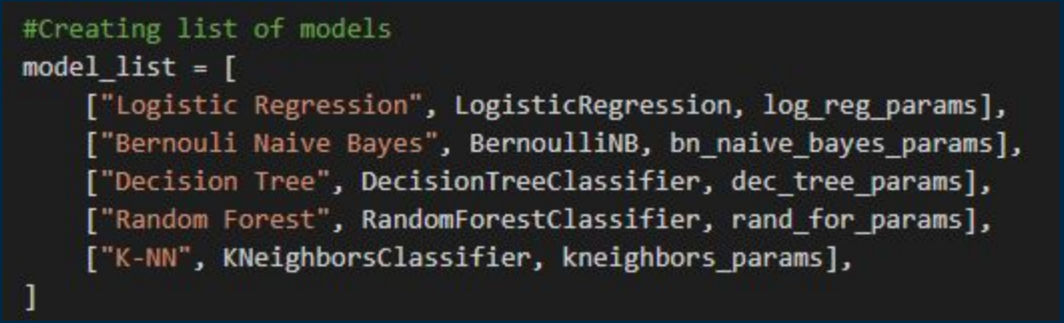
Split Data and Model Selection



```
#Splitting dataset into train test split for tfidf - %80 train, %20 test
X_train, X_test, y_train, y_test = train_test_split(tfidf_word_vector, depend_y, test_size=0.2)
```



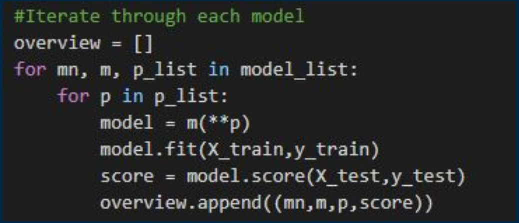
```
#Creating model parameters
log_reg_params = [{"max_iter": 1000}]
bn_naive_bayes_params = [{}]
dec_tree_params = [{}]
rand_for_params = [{}]
kneighbors_params = [{}]
```



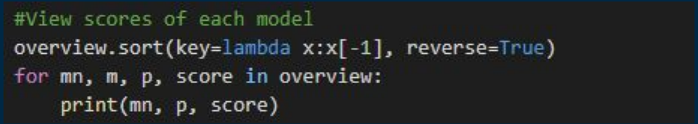
```
#Creating list of models
model_list = [
    ["Logistic Regression", LogisticRegression, log_reg_params],
    ["Bernouli Naive Bayes", BernoulliNB, bn_naive_bayes_params],
    ["Decision Tree", DecisionTreeClassifier, dec_tree_params],
    ["Random Forest", RandomForestClassifier, rand_for_params],
    ["K-NN", KNeighborsClassifier, kneighbors_params],
]
```

Fit, Train, and Test Models

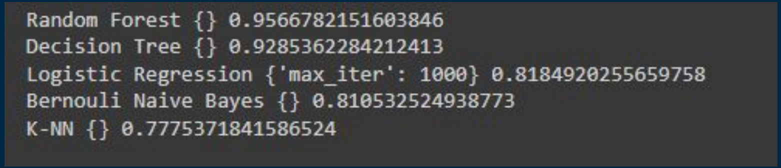
- Loop through each model in the model list to fit the data and test the data
- Store values in a list “overview”
- Sort the results
- Output results by accuracy



```
#Iterate through each model
overview = []
for mn, m, p_list in model_list:
    for p in p_list:
        model = m(**p)
        model.fit(X_train,y_train)
        score = model.score(X_test,y_test)
        overview.append((mn,m,p,score))
```



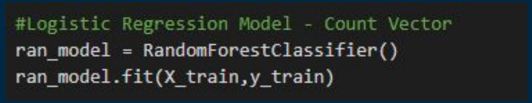
```
#View scores of each model
overview.sort(key=lambda x:x[-1], reverse=True)
for mn, m, p, score in overview:
    print(mn, p, score)
```



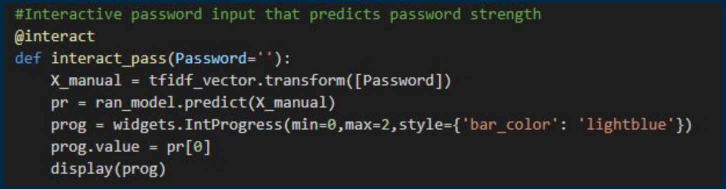
```
Random Forest {} 0.9566782151603846
Decision Tree {} 0.9285362284212413
Logistic Regression {'max_iter': 1000} 0.8184920255659758
Bernouli Naive Bayes {} 0.810532524938773
K-NN {} 0.7775371841586524
```

Interactive Password Classification

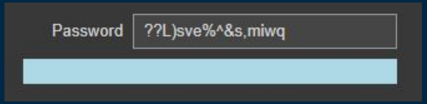
- Picking a model for real time classification
  - Random forest had the highest percentage for the TF-IDF vector
- Defined a function to interact with passwords
  - 0-2 classification



```
#Logistic Regression Model - Count Vector
ran_model = RandomForestClassifier()
ran_model.fit(X_train,y_train)
```



```
#Interactive password input that predicts password strength
@interact
def interact_pass(Password=''):
    X_manual = tfidf_vector.transform([Password])
    pr = ran_model.predict(X_manual)
    prog = widgets.IntProgress(min=0,max=2,style={'bar_color': 'lightblue'})
    prog.value = pr[0]
    display(prog)
```



Password ??L)sve%^&s,miwq

Sources:

- https://medium.com/analytics-vidhya/data-science-for-cybersecurity-password-strength-meter-b933b96bff32
- https://towardsdatascience.com/nlp-text-preprocessing-a-practical-guide-and-template-d80874676e79
- https://www.section.io/engineering-education/building-a-password-strength-classifier-model-using-machine-learning/
- https://thecleverprogrammer.com/2022/08/22/password-strength-checker-with-machine-learning/
- https://scikit-learn.org/stable/index.html
- https://ipywidgets.readthedocs.io/en/stable/index.html
- https://www.geeksforgeeks.org/program-check-strength-password/