Spam Email Detection(Project)

numpy : good for numpy arrays

pandas: good for data-frames

re : regular expression for texts

Stopwords : words that aren't too valueable ('a', 'the')

PorterStemmer: Stem our words. gives root word, removes prefix and suffix

TfIdVectorizer: Converts text to vectors

logisticRegression model : Binary Classification

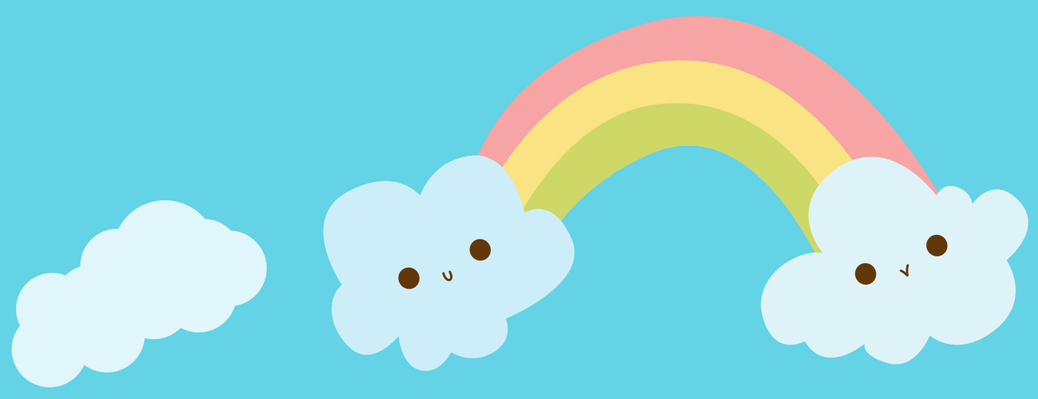
Data Cleansing:

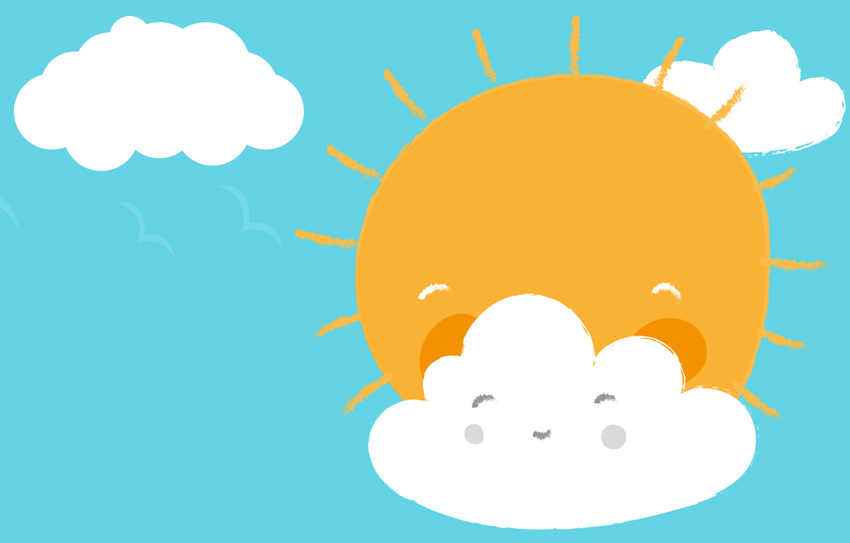
Before cleaning, features like punctuation of capitalization that might be helpful are retained.

```python
SPECIALSYMBOLS =set("@$%&*#^~+=")
def styling(content):
    #for letters only
    letters = [c for c in content if c.isalpha()]
    upper_ratio = sum(1 for c in letters if c.isupper())/len(letters) if letters else 0
    #counts :
    exclamation_count = content.count("!")
    question_count = content.count("?")
    #special signs
    special_sign = sum(c in SPECIALSYMBOLS for c in content)
    #URL count
    url_count = len(re.findall(r"http://", content))

    return pd.Series({
        "upper ratio": upper_ratio,
        "exclamation count ": exclamation_count ,
        "question count": question_count,
        "special count": special_sign,
        "url count": url_count
    })
```

```python
pstem = PorterStemmer()
def cleanse(content):
    new_content = re.sub(r"[^a-z]",'',content)   # → only keeps letters
    new_content = new_content.tolower()
    new_content = new_content.split()   → splits to linked list
    new_content=[]
    new_content = new_content.split   → only keeps root word
    new_content =[pstem.stem(word) for word in new_content if not word in stopwords.words('english')]   → does not take insignificant words
    new_content= ' '.join(new_content)
    return new_content
```





TF ID Vectorizer: Term frequency - Inverse Document frequency

↳ scores words based on how important they are

$$TF = \frac{\text{no. of occurence of t}}{\text{no. of terms in doc}}$$

$$IDF = \log\left(\frac{\text{Total no. of docs}}{\text{no. of docs with t}}\right)$$

$$TFID(t) = TF(t) \times IDF(t)$$

↳ more if occurs more in one doc

↳ less if found in more docs

→ max range keeps vocab to a fixed set to dull noise

→ ngram_range determines how many "bumbs" to conside

```
# print(email_data.columns)
vectorizer = TfidfVectorizer(max_features =3500, ngram_range=(1,2))
X_vec = vectorizer.fit_transform(email_data['cleaned']) #a sparse matrix
```

✓ 0.2s Python

top 3500 words

accepts "totally free")

get the numeric data :

```
x_num = email_data[["upper ratio", "exclamation count", "question count", "special c
x_all = sp.hstack([X_vec,x_num])
```

Python

↳ scipy.sparse — Stacks horizontally first we have the feature matrix and then 5 numeric cols

we had 5 numeric columns, and suppose we get a vectorized matrix (no. of rows, 3500) and x_num(no. of rows, 5)

So the numbers get stacked to form one frame.

Splittting the Dataset to training and testing models

```
X_train, X_test, Y_train, Y_test = train_test_split(x_all,y,test_size=0.1, stratify=y, random_state=2)
```

[ ]

Python

10 % from testing

properly distributed with regards to y