Przygotowanie danych pod *Machine Learning*

In [1]:
```
import matplotlib.pyplot as plt
from unidecode import unidecode
import pandas as pd
import numpy as np
import spacy
import re

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

Wczytujemy data set ***merged_datasets.csv*** (~100k rekordów)

In [2]:
```
mails = pd.read_csv('merged_datasets.csv')
categories = pd.CategoricalDtype(['safe', 'phishing'], ordered=True)
mails['label'] = mails['label'].astype(categories)
```

Usuwamy **duplikaty** i usuwamy **wiersze**, w których **treść** lub **nadawca** są puste

In [3]:
```
mails.drop_duplicates(subset=['subject', 'sender_mail'], keep='first', inplace=True
mails.dropna(subset=['body', 'sender_mail'], inplace=True)
```

Tworzymy **kolumny wartości**, który pomogą rozpoznawać **phishing**

Wyciągamy informacje o **nadawcy**

In [4]:
```
def get_sender_info(sender_mail: str):
    splitted = sender_mail.split('@')
    if len(splitted) != 2:
        return None, None

    name = splitted[0]
    domain = splitted[1]
    return name, domain


mails[['sender_mail_name', 'sender_mail_domain']] = mails['sender_mail'].apply(lamb
```

Ilość cyfr w **nazwie nadawcy**

```
In [5]: def get_nums_count(name: str):
            name = str(name)
            return sum(1 for char in name if char.isnumeric())


        mails['sender_nums_count'] = mails['sender_mail_name'].apply(get_nums_count)
        print(mails.groupby('label').sender_nums_count.describe())
```

```
            count      mean       std  min  25%  50%  75%   max
label
safe      42678.0  0.136886  1.068021  0.0  0.0  0.0  0.0  37.0
phishing  44955.0  0.707218  1.789063  0.0  0.0  0.0  0.0  75.0
```

Ilość cyfr w domenie; długość domeny

```
In [6]: def get_domain_info(domain: str):
            if domain is None:
                return None

            splitted = domain.split('.')[:-1]
            domain_noTLD = '.'.join(splitted)
            return sum(1 for char in domain_noTLD if char.isnumeric()), len(domain_noTLD)


        mails[['sender_domain_num_count', 'sender_domain_length']] = mails['sender_mail_dom
        print(mails.groupby('label').sender_domain_num_count.describe())
        print(mails.groupby('label').sender_domain_length.describe())
```

```
            count      mean       std  min  25%  50%  75%   max
label
safe      42612.0  0.023913  0.215922  0.0  0.0  0.0  0.0   6.0
phishing  44770.0  0.180902  0.855269  0.0  0.0  0.0  0.0  22.0
            count      mean       std  min  25%  50%   75%   max
label
safe      42612.0  7.017624  4.013944  0.0  5.0  5.0   8.0  36.0
phishing  44770.0  9.184298  4.443968  0.0  6.0  8.0  11.0  53.0
```

Znowu usuwamy wiersze, w których są wartości *None*

```
In [7]: mails.dropna(subset=['sender_domain_num_count', 'sender_domain_length', 'sender_mai
```

Sprawdzamy, czy słowa z tematu lub treści są w naszej liście podejrzanych słów

```
In [8]: nlp = spacy.load('en_core_web_md', disable=["parser", "ner"])
        nlp.max_length = 2500000
        stop_words = spacy.lang.en.STOP_WORDS
```

```
In [9]: with open('suspicious_words.txt', 'r') as file:
            lines = file.readlines()

        suspicious_words = []
```

```
for line in lines:
    word = line.strip()
    suspicious_words.append(word)

suspicious_str = ' '.join(suspicious_words)
doc = nlp(suspicious_str)
suspicious_lemmas = [token.lemma_.lower() for token in doc]

suspicious_set = set(suspicious_lemmas)
```

Podejrzane słowa w temacie

In [10]:
```
def count_suspicious_words(text: str):
    doc = nlp(text)
    tokens = [token.lemma_.lower() for token in doc if token.text.isalnum()]
    unique_words = set(tokens)
    sus_words = unique_words.intersection(suspicious_set)
    return len(sus_words)


mails['suspicious_words_subject'] = mails['subject'].apply(count_suspicious_words)
print(mails.groupby('label').suspicious_words_subject.describe())
```

```
            count      mean       std  min  25%  50%  75%  max
label
safe      42612.0  0.152797  0.395230  0.0  0.0  0.0  0.0  4.0
phishing  44770.0  0.219455  0.462415  0.0  0.0  0.0  0.0  5.0
```

Podejrzane słowa w treści

In [11]:
```
mails['suspicious_words_body'] = mails['body'].apply(count_suspicious_words)
print(mails.groupby('label').suspicious_words_body.describe())
```

```
            count      mean       std  min  25%  50%  75%   max
label
safe      42612.0  2.668216  2.915032  0.0  1.0  2.0  3.0  51.0
phishing  44770.0  2.208868  2.767256  0.0  0.0  1.0  3.0  39.0
```

Wyciągamy odnośniki z treści wiadomości e-mail i liczymy ich ilość

In [12]:
```
def extract_urls(text: str):
    url_pattern = re.compile(r'https?://\S+|www\.\S+')
    matches = re.findall(url_pattern, text)

    return matches, len(matches)


mails[['extracted_urls', 'urls_count']] = mails['body'].apply(lambda x: pd.Series(e

def map_url_lens(lens: int):
    if lens <= 2:
        return str(lens)
```

```python
    else:
        return '3<='


mails['urls_count'] = mails['urls_count'].apply(map_url_lens)
categories = pd.CategoricalDtype(['0', '1', '2', '3<='], ordered=True)
mails['urls_count'] = mails['urls_count'].astype(categories)
print(mails.groupby('label').urls_count.value_counts(normalize=True))
```

```
label     urls_count
safe      0             0.623486
          1             0.149324
          3<=           0.131817
          2             0.095372
phishing  0             0.427876
          1             0.355193
          3<=           0.113603
          2             0.103328
Name: proportion, dtype: float64
```

Losujemy (dla czystego sumienia) jeden odnośnik z uzyskanej listy (zakładamy, że jeżeli jeden URL w wiadomości jest phishingiem, to inne też)

In [13]:
```python
# Jeżeli jeden url w mailu jest fałszywy, to wychodzimy z założenia, że inne też
def return_random_url(urls: str):
    urls_len = len(urls)
    if urls_len == 0:
        return None

    random_index = np.random.randint(0, urls_len)
    randomized_url = urls[random_index]
    return randomized_url

mails['in_body_url'] = mails['extracted_urls'].apply(return_random_url)
```

Usuwamy wiersze, które nie zawierają żadnych odnośników, są nam zbędne i tylko utrudniają życie

In [14]:
```python
print(mails.shape)
mails.dropna(subset=['in_body_url'], inplace=True)
print(mails.shape)
print(mails.label.value_counts(normalize=True))
```

```
(87382, 15)
(41658, 15)
label
phishing    0.614864
safe        0.385136
Name: proportion, dtype: float64
```

---

87 382 - 41 658 = **45 724** Tyle wierszy poszło z dymem. Z pozostałych wierszy:

- 61.49% stanowi ***phishing***
- 38.51% stanowią ***bezpieczne wiersze***

Wyciągamy z **odnośników** protokoły: ***HTTP*** i ***HTTPS***

In [15]:
```
def is_https(url: str):
    protocol = url[:5].lower()
    return 'https' if protocol == 'https' else 'http'


mails['protocol'] = mails['in_body_url'].apply(is_https)
categories = pd.CategoricalDtype(['http', 'https'], ordered=True)
mails['protocol'] = mails['protocol'].astype(categories)
print(mails.groupby('protocol').label.value_counts(normalize=True))
```

```
protocol  label
http      phishing    0.626800
          safe        0.373200
https     safe        0.822785
          phishing    0.177215
Name: proportion, dtype: float64
```

Sprawdzamy, czy **odnośniki** zawierają IP

In [16]:
```
def contains_ip(url: str):
    ip_pattern = re.compile(r'\b(?:\d{1,3}\.){3}\d{1,3}\b|\b(?:[0-9a-fA-F]{1,4}:){7
    ips = ip_pattern.findall(url)

    if ips:
        return True
    else:
        return False


mails['contains_ip'] = mails['in_body_url'].apply(contains_ip)
print(mails.groupby('contains_ip').label.value_counts(normalize=True))
```

```
contains_ip  label
False        phishing    0.613819
             safe        0.386181
True         phishing    0.906040
             safe        0.093960
Name: proportion, dtype: float64
```

Wyodrębniamy długość **odnośników**

In [17]:
```
mails['url_length'] = mails['in_body_url'].apply(len)
```

Wyodrębniamy z **odnośników** domenę

In [18]:
```python
def get_domain(url: str):
    pattern = re.compile(r'https?://([^/?]+)')
    match = pattern.match(url)
    if match:
        domain = match.group(1)
        if '/' in domain:
            return domain.split('/')[0]

        return domain
    else:
        return url


mails['domain'] = mails['in_body_url'].apply(get_domain)
```

Wyciągamy z **domeny TLD** (top-level domain)

- ue.poznan.**pl**
- www.vaticannews.**va**

In [19]:
```python
def get_TLD(domain: str):
    split_domain = domain.split('.')
    n = len(split_domain)
    delimiters = ['/', ':', ')', ']', '%', '_', '=', ',', '>', '"', '#', '!']

    after_dot = split_domain[n-1]
    if len(after_dot) > 2:
        after_dot = after_dot.split('/')[0]

        if len(after_dot) > 2:
            for delimiter in delimiters:
                after_dot = " ".join(after_dot.split(delimiter))

            after_dot = after_dot.split()[0]

    TLD = '.'+after_dot.lower()
    return TLD


mails['TLD'] = mails['domain'].apply(get_TLD)
```

Sprawdzamy, czy **TLD** zawiera tylko litery (a nie na przykład **liczbę**, **adres IP** itd.)

In [20]:
```python
def is_tld_alpha(tld: str):
    return tld[1:].isalpha()


mails['TLD_alpha'] = mails['TLD'].apply(is_tld_alpha)
print(mails.groupby('TLD_alpha').label.value_counts(normalize=True))
```

```
TLD_alpha  label
False      safe        0.624829
           phishing    0.375171
True       phishing    0.619157
           safe        0.380843
Name: proportion, dtype: float64
```

Sprawdzany **poziom subdomeny**

- wikipedia.org = 0
- **en**.wikipedia.org = 1

In [21]:
```
def get_subdomain_level(domain: str):
    return domain.count('.')-1


def map_subdomain_lv(num):
    if num <= 2:
        return str(num)
    else:
        return '3<='


mails['subdomain_level'] = mails['domain'].apply(get_subdomain_level)
mails['subdomain_level'] = mails['subdomain_level'].apply(map_subdomain_lv)
categories = pd.CategoricalDtype(['0', '1', '2', '3<='], ordered=True)
mails['subdomain_level'] = mails['subdomain_level'].astype(categories)
print(mails.groupby('label').subdomain_level.value_counts(normalize=True))
```

```
label     subdomain_level
safe      1                  0.792560
          2                  0.110159
          0                  0.074961
          3<=                0.022319
phishing  1                  0.514563
          0                  0.385316
          2                  0.094726
          3<=                0.005395
Name: proportion, dtype: float64
```

Liczymy wystapienia "/" w **odnośnikach**

In [22]:
```
def count_slashes(url: str):
    return url.count('/')


def map_slashes(num):
    if num <= 5:
        return str(num)
    else:
        return '6<='


mails['slash_count'] = mails['in_body_url'].apply(count_slashes)
```

```
mails['slash_count'] = mails['slash_count'].apply(map_slashes)
categories = pd.CategoricalDtype(['0', '1', '2', '3', '4', '5', '6<='], ordered=Tru
mails['slash_count'] = mails['slash_count'].astype(categories)
print(mails.groupby('label').slash_count.value_counts(normalize=True))
```

```
label     slash_count
safe      5              0.285527
          3              0.209424
          4              0.161057
          6<=            0.142545
          2              0.118113
          0              0.072675
          1              0.010658
phishing  3              0.456235
          4              0.193956
          2              0.177403
          6<=            0.109120
          5              0.035215
          0              0.025103
          1              0.002967
Name: proportion, dtype: float64
```

Liczymy wystapienia "." w **odnośnikach**

In [23]:
```
def count_dots(url: str):
    return url.count('.')


def map_dots(num):
    if num <= 4:
        return str(num)
    else:
        return '5<='


mails['dots_count'] = mails['in_body_url'].apply(count_dots)

mails['dots_count'] = mails['dots_count'].apply(map_dots)
categories = pd.CategoricalDtype(['0', '1', '2', '3', '4', '5<='], ordered=True)
mails['dots_count'] = mails['dots_count'].astype(categories)
print(mails.groupby('label').dots_count.value_counts(normalize=True))
```

```
label     dots_count
safe      2             0.528546
          3             0.319808
          4             0.067627
          1             0.055161
          5<=           0.027362
          0             0.001496
phishing  1             0.344616
          2             0.338955
          3             0.129734
          4             0.097095
          5<=           0.088662
          0             0.000937
Name: proportion, dtype: float64
```

Liczymy wystapienia "-" w **odnośnikach**

In [24]:
```
def count_hyphens(url: str):
    return url.count('-')


def map_hyphens(num):
    if num <= 1:
        return str(num)
    else:
        return '2<='


mails['hyphens_count'] = mails['in_body_url'].apply(count_hyphens)

mails['hyphens_count'] = mails['hyphens_count'].apply(map_hyphens)
categories = pd.CategoricalDtype(['0', '1', '2<='], ordered=True)
mails['hyphens_count'] = mails['hyphens_count'].astype(categories)
print(mails.groupby('label').hyphens_count.value_counts(normalize=True))
```

```
label     hyphens_count
safe      0                0.706245
          1                0.203815
          2<=              0.089940
phishing  0                0.898376
          1                0.066214
          2<=              0.035410
Name: proportion, dtype: float64
```

Sprawdzamy, czy w **odnośnikach** znajdują się litery z alfabetu innego niż ***łaciński*** (例 itd.)

In [25]:
```
def has_non_latin_chars(url: str):
    ascii = unidecode(url)
    return url != ascii


mails['has_non_latin'] = mails['in_body_url'].apply(has_non_latin_chars)
print(mails.groupby('has_non_latin').label.value_counts(normalize=True))
```

```
has_non_latin  label
False          phishing    0.614415
               safe        0.385585
True           phishing    0.796117
               safe        0.203883
Name: proportion, dtype: float64
```

Przygotowujemy zebrane dane pod *Machine Learning*

Dane muszą być w postaci liczbowej.

In [26]:
```
print(mails.head(1))
```

```
   Unnamed: 0      sender_mail                   subject  \
0           0  Young@iworld.de  Never agree to be a loser

                                                body     label  \
0  Buck up, your troubles caused by small dimensi...  phishing

  sender_mail_name sender_mail_domain  sender_nums_count  \
0            Young          iworld.de                  0

   sender_domain_num_count  sender_domain_length  ...  contains_ip  \
0                      0.0                   6.0  ...        False

   url_length          domain   TLD TLD_alpha subdomain_level  slash_count  \
0          21  whitedone.com  .com      True               0            3

   dots_count hyphens_count has_non_latin
0           1             0         False

[1 rows x 26 columns]
```

Tworzymy kolumny is_phishing oraz is_safe na bazie kolumny label

(one-hot encode)

In [ ]:
```
one_hot_encoded = pd.get_dummies(mails['label'], prefix='is')
mails = pd.concat([mails, one_hot_encoded], axis=1)
mails.drop('label', axis=1, inplace=True)
```

Zamiana kolejności [y | x_i] (dla wygody)

In [28]:
```
mails_ML = mails[
    ['is_phishing', 'is_safe', 'suspicious_words_subject', 'suspicious_words_body',
     'urls_count', 'protocol', 'contains_ip', 'url_length', 'TLD_alpha', 'subdomain
]
print(mails_ML.head(3))
```

```
   is_phishing  is_safe  suspicious_words_subject  suspicious_words_body  \
0         True    False                         0                      2   
1         True    False                         0                      0   
2         True    False                         0                      4   

   sender_nums_count  sender_domain_num_count  sender_domain_length  \
0                  0                      0.0                   6.0   
1                  4                      0.0                   6.0   
2                  0                      0.0                  16.0   

  urls_count protocol  contains_ip  url_length  TLD_alpha subdomain_level  \
0          1     http        False          21       True               0   
1          1     http        False          25       True               1   
2        3<=     http        False         107       True               1   

  slash_count dots_count hyphens_count  has_non_latin  
0           3          1             0          False  
1           2          2             0          False  
2         6<=        5<=             0          False  
```

Mapowanie wartości (z ***categories/object*** (str) na ***int***)

In [30]:

```python
urls_count_map = {
    '1': 1,
    '2': 2,
    '3<=': 3,
}

protocol_map = {
    'https': 1,
    'http': 0
}

contains_ip_map = {
    True: 1,
    False: 0
}

TLD_alpha_map = {
    True: 1,
    False: 0
}

subdomain_level = {
    '0': 0,
    '1': 1,
    '2': 2,
    '3<=': 3
}

slash_count_map = {
    '0': 0,
    '1': 1,
    '2': 2,
    '3': 3,
```

```python
    '4': 4,
    '5': 5,
    '6<=': 6,
}

dots_count_map = {
    '0': 0,
    '1': 1,
    '2': 2,
    '3': 3,
    '4': 4,
    '5<=': 5,
}

hyphens_count_map = {
    '0': 0,
    '1': 1,
    '2<=': 2,
}

has_non_latin_map = {
    True: 1,
    False: 0
}

mails_ML.loc[:, 'urls_count'] = mails_ML['urls_count'].map(urls_count_map)
mails_ML.loc[:, 'protocol'] = mails_ML['protocol'].map(protocol_map)
mails_ML.loc[:, 'contains_ip'] = mails_ML['contains_ip'].map(contains_ip_map)
mails_ML.loc[:, 'TLD_alpha'] = mails_ML['TLD_alpha'].map(TLD_alpha_map)
mails_ML.loc[:, 'subdomain_level'] = mails_ML['subdomain_level'].map(subdomain_leve
mails_ML.loc[:, 'slash_count'] = mails_ML['slash_count'].map(slash_count_map)
mails_ML.loc[:, 'dots_count'] = mails_ML['dots_count'].map(dots_count_map)
mails_ML.loc[:, 'hyphens_count'] = mails_ML['hyphens_count'].map(hyphens_count_map)
mails_ML.loc[:, 'has_non_latin'] = mails_ML['has_non_latin'].map(has_non_latin_map)
mails_ML = mails_ML.astype(float)
print(mails_ML.head(3))
```

```
   is_phishing  is_safe  suspicious_words_subject  suspicious_words_body  \
0          1.0      0.0                       0.0                    2.0   
1          1.0      0.0                       0.0                    0.0   
2          1.0      0.0                       0.0                    4.0   

   sender_nums_count  sender_domain_num_count  sender_domain_length  \
0                0.0                      0.0                   6.0   
1                4.0                      0.0                   6.0   
2                0.0                      0.0                  16.0   

   urls_count  protocol  contains_ip  url_length  TLD_alpha  subdomain_level  \
0         1.0       0.0          0.0        21.0        1.0              0.0   
1         1.0       0.0          0.0        25.0        1.0              1.0   
2         3.0       0.0          0.0       107.0        1.0              1.0   

   slash_count  dots_count  hyphens_count  has_non_latin  
0          3.0         1.0            0.0            0.0  
1          2.0         2.0            0.0            0.0  
2          6.0         5.0            0.0            0.0  
```

Zapisujemy **rezultat** do pliku .csv

In [31]:
```
mails_ML.to_csv('ML_DataFrame.csv')
```