Predykcja ataku **phishingowego** w wiadomości e-mail za pomocą ***nadzorowanego nauczania maszynowegoo***

Dataset:

- ***Phishing Email Curated Datasets***
  - https://zenodo.org/records/8339691

Najważniejsze użyte moduły

- **pandas** - praca z *Data Framami*
- **numpy** - obliczenia
- **matplotlib.pyplot** - wizualizacja
- **sklearn** - wszelakie narzędzia do *Machine Learningu*

In [1]:
```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import KFold, train_test_split, cross_val_score, GridS
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
import sklearn.metrics as skm

import import_ipynb
import re
from unidecode import unidecode
from termcolor import colored

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

Wczytanie przygotowanego data framu

In [2]:
```
learning_set = pd.read_csv('ML_DataFrame.csv', index_col=0)
print(learning_set.head(3))
```

```
   label  suspicious_words_subject  suspicious_words_body  sender_nums_count  \
0    1.0                       0.0                    2.0                0.0
1    1.0                       0.0                    0.0                4.0
2    1.0                       0.0                    4.0                0.0

   sender_domain_num_count  sender_domain_length  urls_count  protocol  \
0                      0.0                   6.0         1.0       0.0
1                      0.0                   6.0         1.0       0.0
2                      0.0                  16.0         3.0       0.0

   contains_ip  url_length  TLD_alpha  subdomain_level  slash_count  \
0          0.0        21.0        1.0              0.0          3.0
1          0.0        25.0        1.0              1.0          2.0
2          0.0        72.0        1.0              1.0          6.0

   dots_count  hyphens_count  has_non_latin
0         1.0            0.0            0.0
1         2.0            0.0            0.0
2         4.0            0.0            0.0
```

Usunięcie wierszy z pustym wartościami

In [3]:
```
print(learning_set.isna().sum())
learning_set.dropna(inplace=True)
```

```
label                        0
suspicious_words_subject     0
suspicious_words_body        0
sender_nums_count            0
sender_domain_num_count      0
sender_domain_length         0
urls_count                   0
protocol                     0
contains_ip                  0
url_length                   0
TLD_alpha                    0
subdomain_level             84
slash_count                  0
dots_count                   0
hyphens_count                0
has_non_latin                0
dtype: int64
```

Wybranie *X* i *y*

X:

- suspicious_words_subject
- suspicious_words_body
- sender_nums_count
- sender_domain_num_count
- sender_domain_length
- urls_count

- protocol
- contains_ip
- url_length
- TLD_alpha
- subdomain_level
- slash_count
- dots_count
- hyphens_count
- has_non_latin

y:

- label

In [4]:
```
X = learning_set.loc[:, 'suspicious_words_subject':'has_non_latin'].values
y = learning_set.loc[: , 'label'].values
print(X.shape, y.shape)
```

```
(41574, 15) (41574,)
```

Rozdzielanie *X* i *y* na treningowe i testowe zestawy

In [5]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

Skalujemy wartości *X*-ów

In [6]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [7]:
```
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(33259, 15) (8315, 15) (33259,) (8315,)
```

Szukanie najlepszych parametrów dla *KNeighborsClassifier*

In [8]:
```
kf = KFold(n_splits=6, shuffle=True, random_state=42)
params = {
    'n_neighbors': np.arange(1, 15, 1),
    'weights': ['uniform', 'distance'],
    'p': [1, 2]
}
knn = KNeighborsClassifier()
knn_cv = GridSearchCV(knn, param_grid=params, cv=kf)
knn_cv.fit(X_train, y_train)

print('Najlepsze parametry dla KNeighborsClassifier:')
for p, val in knn_cv.best_params_.items():
    print('{}: {}'.format(p, val), end='\n')
print('Uzyskana precyzja: ', knn_cv.best_score_)
```

```
best_knn = knn_cv.best_estimator_
test_accuracy = best_knn.score(X_test, y_test)
print("Precyzja zestawu testowego:", test_accuracy)
```

```
Najlepsze parametry dla KNeighborsClassifier:
n_neighbors: 8
p: 1
weights: distance
Uzyskana precyzja:  0.8931714625269559
Precyzja zestawu testowego: 0.8939266386049308
```

Szukanie najlepszych parametrów dla *LogisticRegression*

In [ ]:
```
from sklearn.linear_model import LogisticRegression
kf = KFold(n_splits=6, shuffle=True, random_state=42)
params = {
    'penalty': [None, 'l2'],
    'C': [0.001, 0.01, 0.1, 1, 10]
}
logreg_forest = LogisticRegression()
logreg_cv = GridSearchCV(logreg_forest, param_grid=params, cv=kf)
logreg_cv.fit(X_train, y_train)

print('Najlepsze parametry dla LogisticRegression:')
for p, val in logreg_cv.best_params_.items():
    print('{}: {}'.format(p, val), end='\n')
print('Uzyskana precyzja: ', logreg_cv.best_score_)

best_logreg = logreg_cv.best_estimator_
test_accuracy = best_logreg.score(X_test, y_test)
print("Precyzja zestawu testowego:", test_accuracy)
```

Szukanie najlepszych parametrów dla *DecisionTreeClassifier*

In [10]:
```
kf = KFold(n_splits=6, shuffle=True, random_state=42)
params = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
tree = DecisionTreeClassifier()
tree_cv = GridSearchCV(tree, param_grid=params, cv=kf)
tree_cv.fit(X_train, y_train)

print('Najlepsze parametry dla DecisionTreeClassifier:')
for p, val in tree_cv.best_params_.items():
    print('{}: {}'.format(p, val), end='\n')
print('Uzyskana precyzja: ', tree_cv.best_score_)

best_tree = tree_cv.best_estimator_
test_accuracy = best_tree.score(X_test, y_test)
print("Precyzja zestawu testowego:", test_accuracy)
```

```
Najlepsze parametry dla DecisionTreeClassifier:
criterion: gini
max_depth: None
min_samples_leaf: 1
min_samples_split: 10
Uzyskana precyzja:  0.8818364460390439
Precyzja zestawu testowego: 0.8817799158147925
```

Szukanie najlepszych parametrów dla *RandomForestClassifier*

In [11]:
```
from sklearn.ensemble import RandomForestClassifier

kf = KFold(n_splits=6, shuffle=True, random_state=42)
params = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
rand_forest = RandomForestClassifier()
rand_forest_cv = GridSearchCV(rand_forest, param_grid=params, cv=kf)
rand_forest_cv.fit(X_train, y_train)

print('Najlepsze parametry dla RandomForestClassifier:')
for p, val in rand_forest_cv.best_params_.items():
    print('{}: {}'.format(p, val), end='\n')
print('Uzyskana precyzja: ', rand_forest_cv.best_score_)

best_rand_forest = rand_forest_cv.best_estimator_
test_accuracy = best_rand_forest.score(X_test, y_test)
print("Precyzja zestawu testowego:", test_accuracy)
```

```
Najlepsze parametry dla RandomForestClassifier:
max_depth: None
min_samples_leaf: 1
min_samples_split: 5
n_estimators: 100
Uzyskana precyzja:  0.9137373190683672
Precyzja zestawu testowego: 0.9128081779915814
```

Szukanie najlepszych parametrów dla *SVM Classifier*

In [12]:
```
from sklearn.svm import SVC

kf = KFold(n_splits=6, shuffle=True, random_state=42)
params = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto', 0.1, 1]
}
svm = SVC()
svm_cv = GridSearchCV(svm, param_grid=params, cv=kf)
svm_cv.fit(X_train, y_train)

print('Najlepsze parametry dla SVM Classifier:')
```

```
for p, val in svm_cv.best_params_.items():
    print('{}: {}'.format(p, val), end='\n')
print('Uzyskana precyzja: ', svm_cv.best_score_)

best_svm = svm_cv.best_estimator_
test_accuracy = best_svm.score(X_test, y_test)
print("Precyzja zestawu testowego:", test_accuracy)
```

```
Najlepsze parametry dla SVM Classifier:
C: 1
gamma: 1
kernel: rbf
Uzyskana precyzja:  0.8840010675208222
Precyzja zestawu testowego: 0.8849067949488876
```

Ewaluacja modeli

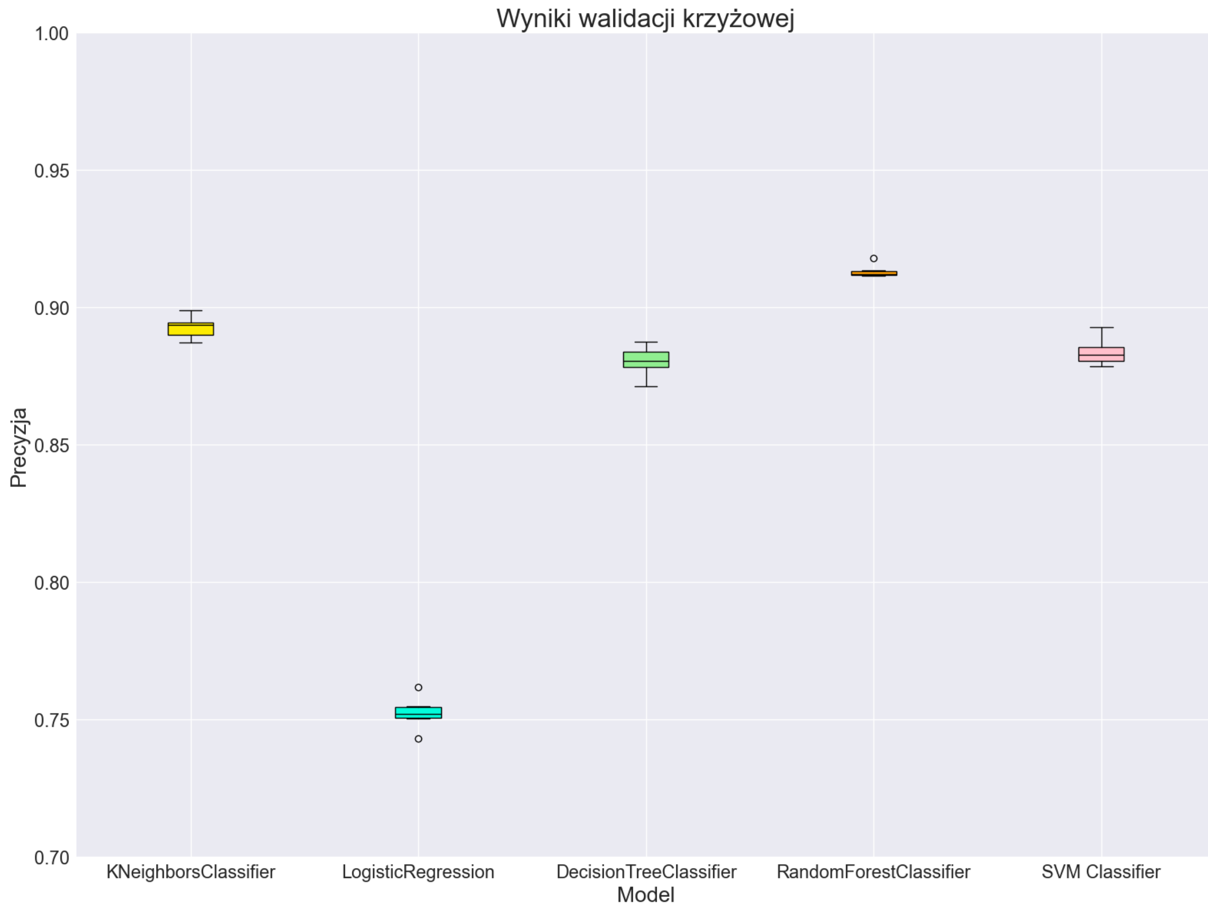
In [13]:

```
models = {
    'KNeighborsClassifier': KNeighborsClassifier(n_neighbors=8, p=1, weights='dista
    'LogisticRegression': LogisticRegression(C=1, penalty='l2'),
    'DecisionTreeClassifier': DecisionTreeClassifier(criterion='gini', max_depth=No
    'RandomForestClassifier': RandomForestClassifier(max_depth=None, min_samples_le
    'SVM Classifier': SVC(C=10, gamma=1, kernel='rbf')
}

results = []
for model in models.values():
    kf = KFold(n_splits=6, shuffle=True, random_state=10)
    cv_results = cross_val_score(model, X_train, y_train, cv=kf)
    results.append(cv_results)
```

In [14]:

```
plt.style.use('seaborn-v0_8')
fig, ax = plt.subplots(figsize=(16, 12))

boxplot = ax.boxplot(
    results,
    labels=models.keys(),
    patch_artist=True,
    boxprops=dict(edgecolor='black'),
    whiskerprops=dict(linewidth=1),
    capprops=dict(linewidth=1),
    medianprops=dict(color='black', linewidth=1),
    widths=0.2,
)

box_colors = ['#fceb03', '#03fcdb', '#90EE90', '#fc9d03', '#FFC0CB']
for box, color in zip(boxplot['boxes'], box_colors):
    box.set(facecolor=color)

ax.set_title('Wyniki walidacji krzyżowej', fontsize=24)
ax.tick_params(axis='both', labelsize=17)
ax.set_xlabel('Model', fontsize=20)
ax.set_ylabel('Precyzja', fontsize=20)
ax.set_ylim(0.70, 1)
ax.yaxis.grid(True)
```

```
plt.tight_layout()
plt.show()
```



Nauczenie najskuteczniejszego modelu

- Najlepszy okazał się ***RandomForestClassifier***

In [15]:
```
import sklearn.metrics as skm
random_forest = RandomForestClassifier(max_depth=None,
                                       min_samples_leaf=1,
                                       min_samples_split=10,
                                       n_estimators=50)
rand_forest.fit(X_train, y_train)
y_pred = rand_forest.predict(X_test)
```

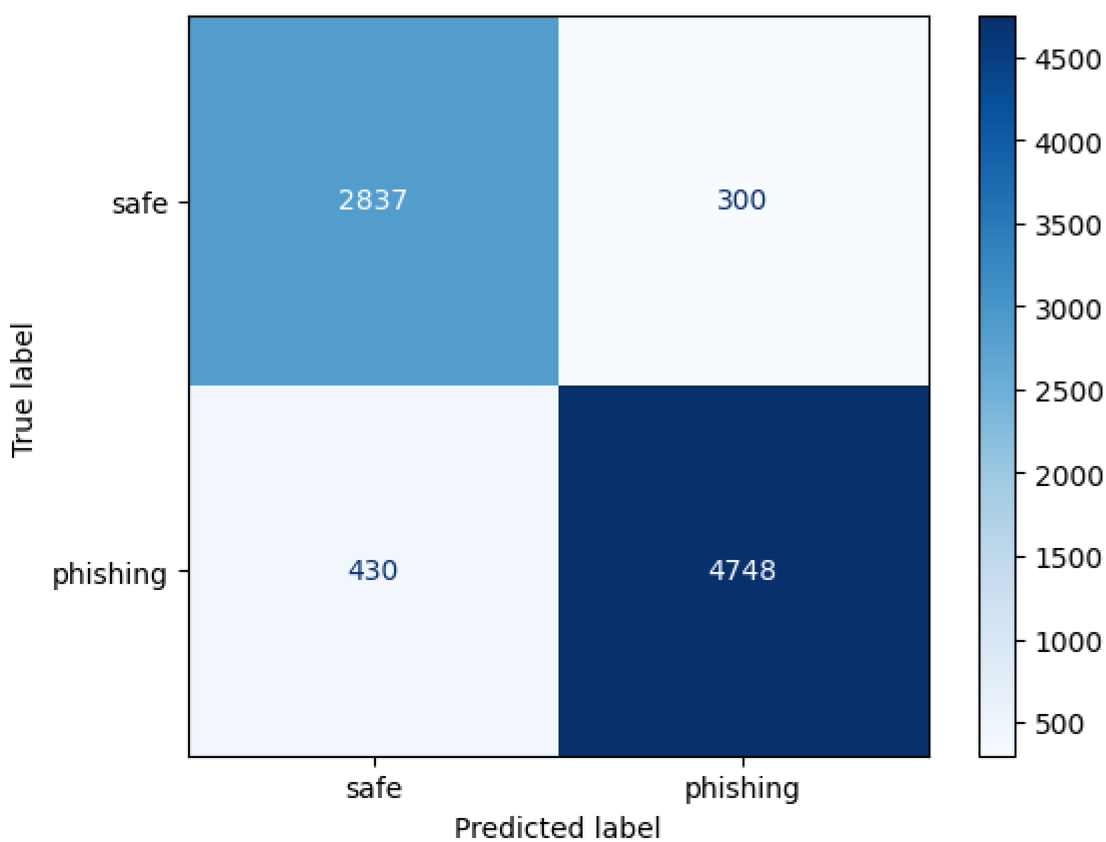
Ponowne sprawdzenie skuteczności modelu

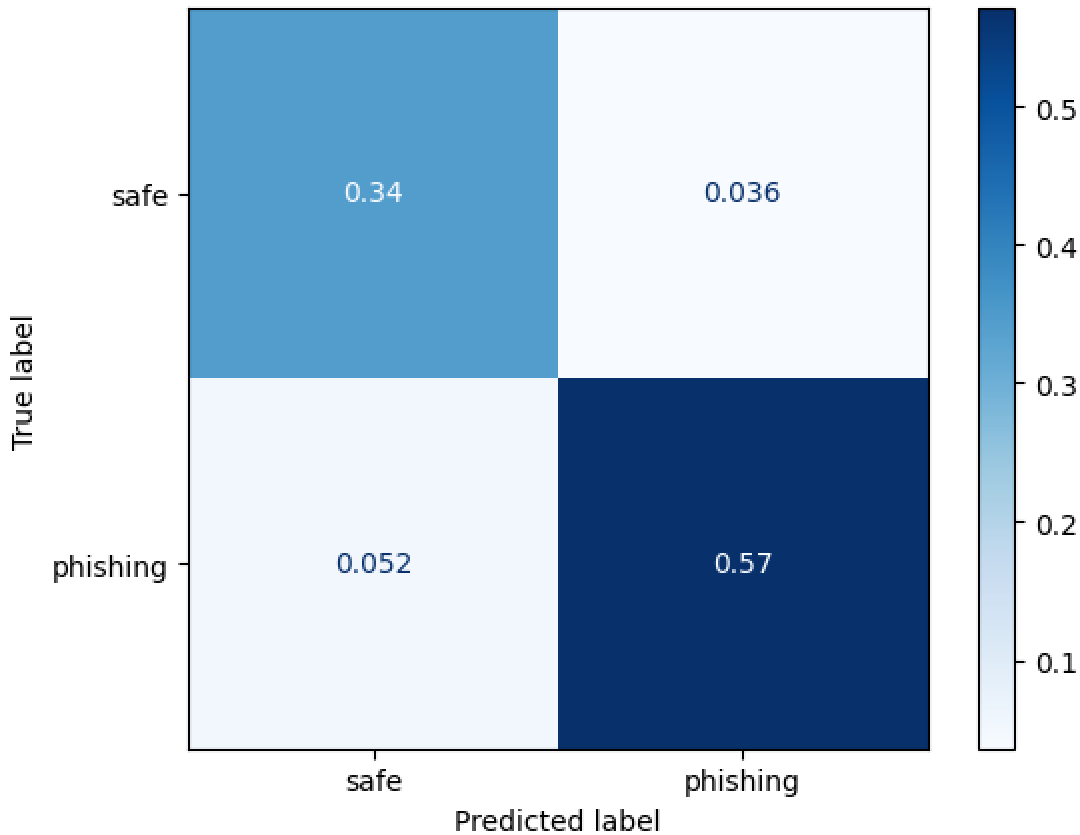
Tym razem dokładniej, przy pomocy:

- *macierzy pomyłek* (confusion matrix)
- *classification report*

In [16]:
```
# Confusion Matrix
confusion_matrix = skm.confusion_matrix(y_test, y_pred)
```

```
plt.style.use('default')
cm_display = skm.ConfusionMatrixDisplay(
    confusion_matrix=confusion_matrix,
    display_labels=['safe', 'phishing'])

cm_display.plot(cmap='Blues')
plt.show()

# Wartości %
cm_display = skm.ConfusionMatrixDisplay(
    confusion_matrix=confusion_matrix/np.sum(confusion_matrix),
    display_labels=['safe', 'phishing'])

cm_display.plot(cmap='Blues')
plt.show()

# Classification report
target_names = ['safe', 'phishing']
class_report = skm.classification_report(y_test, y_pred, target_names=target_names)
print(class_report)
```






```
              precision    recall  f1-score   support

        safe       0.87      0.90      0.89      3137
    phishing       0.94      0.92      0.93      5178

    accuracy                           0.91      8315
   macro avg       0.90      0.91      0.91      8315
weighted avg       0.91      0.91      0.91      8315
```

In [ ]:

```
# Confusion Matrix
confusion_matrix = skm.confusion_matrix(y_test, y_pred)

plt.style.use('default')
fig, ax = plt.subplots(ncols=2, nrows=1)
ax[0] = skm.ConfusionMatrixDisplay(
    confusion_matrix=confusion_matrix,
    display_labels=['safe', 'phishing'])

ax[0].plot(cmap='Blues')


# Wartości %
ax[1] = skm.ConfusionMatrixDisplay(
    confusion_matrix=confusion_matrix/np.sum(confusion_matrix),
    display_labels=['safe', 'phishing'])

ax[1].plot(cmap='Blues')
plt.show()
```

```
# Classification report
target_names = ['safe', 'phishing']
class_report = skm.classification_report(y_test, y_pred, target_names=target_names)
print(class_report)
```

Tworzymy metodę do konwertowania maila na wartości liczbowe na wzór tych użytych przy nauczaniu modelu

In [23]:
```
from unidecode import unidecode
import spacy
import re

nlp = spacy.load('en_core_web_md', disable=["parser", "ner"])
def import_suspicious_words():
    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)
    return suspicious_set
def parse_mail_to_nums(mail: dict):

    sender = mail['sender']
    splitted = sender.split('@')

    sender_name = splitted[0]
    sender_domain = splitted[1]

    sender_num_count = sum(1 for char in sender_name if char.isnumeric())

    splitted_dom = sender_domain.split('.')[:-1]
    domain_noTLD = '.'.join(splitted_dom)
    print(domain_noTLD)
    sender_domain_num_count = sum(1 for char in domain_noTLD if char.isnumeric())
    print('sender_domain_num_count ', sender_domain_num_count)
    sender_domain_length = len(domain_noTLD)

    # subject and body text
    suspicious_set = import_suspicious_words()

    doc = nlp(mail['subject'])
    tokens = [token.lemma_.lower() for token in doc if token.text.isalnum()]
    unique_words = set(tokens)
    sus_words = unique_words.intersection(suspicious_set)
    suspicious_words_subject = len(sus_words)

    doc = nlp(mail['body'])
```

```
    tokens = [token.lemma_.lower() for token in doc if token.text.isalnum()]
    unique_words = set(tokens)
    sus_words = unique_words.intersection(suspicious_set)
    suspicious_words_body = len(sus_words)


    # body urls
    body = mail['body']
    url_pattern = re.compile(r'https?://\S+|www\.\S+')
    urls = re.findall(url_pattern, body)
    # urls_count
    urls_count = len(urls)
    url = urls[np.random.randint(0, urls_count)]
    # protocol
    protocol = url[:5].lower()
    protocol = 'https' if protocol == 'https' else 'http'
    # contains_ip
    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)
    contains_ip = 1 if IPs else 0
    # url_length
    url_length = len(url)
    # TLD_alpha
    pattern = re.compile(r'https?://([^/?]+)')
    match = pattern.match(url)
    if match:
        domain = match.group(1)
        if '/' in domain:
            domain = domain.split('/')[0]
    else:
        domain = url

    split_domain = domain.split('.')
    n = len(split_domain)
    delimiters = ['/', ':', ')', ']', '%', '_', '=', ',', '>', '"', '#', '!']
    # Check if not weird ending
    after_dot = split_domain[n-1]
    if len(after_dot) > 2 and not after_dot.isalpha():
        after_dot = after_dot.split('/')[0]

        if len(after_dot) > 2 and not after_dot.isalpha():
            print(after_dot, 'INSIDE')
            for delimiter in delimiters:
                after_dot = " ".join(after_dot.split(delimiter))

            after_dot = after_dot.split()[0]

    TLD = '.'+after_dot.lower()
    TLD_alpha = TLD[1:].isalpha()
    # subdomain_level
    subdomain_level = domain.count('.')-1
    # slash_count
    slash_count = url.count('/')
    # dots_count
    dots_count = url.count('.')
    # hyphens_count
```

```python
    hyphens_count = url.count('-')
    # has_non_latin
    ascii = unidecode(url)
    has_non_latin = url != ascii

    data = {
        'sender_num_count': sender_num_count,
        'sender_domain_num_count': sender_domain_num_count,
        'sender_domain_length': sender_domain_length,
        'suspicious_words_subject': suspicious_words_subject,
        'suspicious_words_body': suspicious_words_body,
        'urls_count': urls_count,
        'protocol': protocol,
        'contains_ip': contains_ip,
        'url_length': url_length,
        'TLD_alpha': TLD_alpha,
        'subdomain_level': subdomain_level,
        'slash_count': slash_count,
        'dots_count': dots_count,
        'hyphens_count': hyphens_count,
        'has_non_latin': has_non_latin,
    }

    for k, v in data.items():
        print('{}: {}'.format(k, v), end='\n')

    # numeric values
    urls_count_out = urls_count if urls_count <= 2 else 3
    protocol_out = 1 if protocol=='https' else 0
    contains_ip_out = contains_ip
    url_length_out = url_length
    TLD_alpha_out = 1 if TLD_alpha is True else 0
    subdomain_level_out = subdomain_level if subdomain_level <= 2 else 3
    slash_count_out = slash_count if slash_count <= 5 else 6
    dots_count_out = dots_count if dots_count <= 4 else 5
    hyphens_count_out = hyphens_count if hyphens_count <= 1 else 2
    has_non_latin_out = has_non_latin

    X_output = np.array(list([
        [sender_num_count],
        [sender_domain_num_count],
        [sender_domain_length],
        [suspicious_words_subject],
        [suspicious_words_body],
        [urls_count_out],
        [protocol_out],
        [contains_ip_out],
        [url_length_out],
        [TLD_alpha_out],
        [subdomain_level_out],
        [slash_count_out],
        [dots_count_out],
        [hyphens_count_out],
        [has_non_latin_out]
    ]))
    return X_output.reshape(1, -1)
```

Testy na życiowych przykładach

In [50]:
```
mail_example = {
'sender': 'Hanna.Wdowicka@ue.poznan.pl',
'subject': 'Reaktywacja SKN Estymator',
'body': '''
        Szanowni Państwo,

        Poniżej przekazuję wiadomość od dr Macieja Beęsewicza, prof. UEP. Osoby zai
        Pozdrawiam,
        Hanna Wdowicka

        Temat: Reaktywacja SKN Estymator

        Treść:

        Szanowni Państwo,

        dr Maciej Beręsewicz z Katedry Statystyki poszukuje studentów/tek zainteres

        Poniżej przedstawiono przykłady prac, które będą realizowane w ramach pracy

        1. Estimating the number of entities with vacancies using administrative an
        2. Inferring job vacancies from online job advertisements -- https://ec.eur
        3. Enhancing the Demand for Labour survey by including skills from online j
        4. Maximum entropy classification for record linkage – https://htmlpreview.
        5. Developing methods for determining the number of unauthorized foreigners

        Proszę się kontaktować przez mail maciej.beresewicz@ue.poznan.pl lub przez

        Z poważaniem

        dr Maciej Beręsewicz, prof. UEP

        Katedra Statystyki
        Uniwersytet Ekonomiczny w Poznaniu
        Department of Statistics
        Poznań University of Economics and Business
        al. Niepodległości 10 | 61-875 Poznań
        tel. + 48 61-854-36-80 | maciej.beresewicz@ue.poznan.pl
        www.ue.poznan.pl
'''
}
prediction = rand_forest.predict(
    scaler.transform(parse_mail_to_nums(mail_example))
)

print('-=-=-=-=-\nPredykcja:\n-=-=-=-=-')
print(colored('Phishing! >:(', 'red')) if prediction == 1 else print(colored('E-mai
```

```
ue.poznan
sender_domain_num_count  0
sender_num_count: 0
sender_domain_num_count: 0
sender_domain_length: 9
suspicious_words_subject: 0
suspicious_words_body: 2
urls_count: 6
protocol: https
contains_ip: 0
url_length: 52
TLD_alpha: True
subdomain_level: 2
slash_count: 6
dots_count: 3
hyphens_count: 1
has_non_latin: False
-=-=-=-=-
Predykcja:
-=-=-=-=-
E-mail bezpieczny 8)
```

In [32]:
```
mail_example = {
'sender': 'x3r0@ysadna321.com',
'subject': 'Your xero invoice available now.',
'body': '''
        Hi,
        Thanks for working with us. Your bill for $373.75 was due on 28 Aug 2016.
        If you've already paid it, please ignore this email and sorry for bothering
        To view your bill visit http://in.x312412.qwe12/5LQDhRwfvoQfeDtLDMqkk1JWSqC
        If you've got any questions, or want to arrange alternative payment don't h
        Thanks
        NJW Limited
        Download PDF
'''
}
prediction = rand_forest.predict(
    scaler.transform(parse_mail_to_nums(mail_example))
)

print('-=-=-=-=-\Predykcja:\n-=-=-=-=-')
print(colored('Phishing! >:(', 'red')) if prediction == 1 else print(colored('E-mai
```

```
ysadna321
sender_domain_num_count  3
qwe12 INSIDE
sender_num_count: 2
sender_domain_num_count: 3
sender_domain_length: 9
suspicious_words_subject: 1
suspicious_words_body: 2
urls_count: 1
protocol: http
contains_ip: 0
url_length: 65
TLD_alpha: False
subdomain_level: 1
slash_count: 3
dots_count: 3
hyphens_count: 0
has_non_latin: False
-=-=-=-=-\Predykcja:
-=-=-=-
Phishing! >:(
```

✨ *Rezultat projektu* ✨

Przy pomocy ***RandomForestClassifier*** udało się wytrenować model z wynikami:

- **accuracy**: 0.91
- **precision**:
  - safe: 0.87
  - phishing: 0.94
- **recall**:
  - safe: 0.90
  - phishing: 0.92
- **f1**:
  - safe: 0.89
  - phishing: 0.93

Problemy projektu

- mało **uniwersalny**
- **model** za bardzo opiera się na odnośnikach znajdujących się w e-mailu
  - bardzo dużo maili **phishingowych**zawiera buttony, nie klasyczne odnośniki
- dostęp do jakiegokolwiek sensownego API jest płatny
  - brak sprawdzania domen w blacklistach
  - brak sprawdzania adresów e-mail w blacklistach
  - słabe/brak informacji o szyfrowaniu SSL
  - brak sprawdzania wieku domeny

- Dane zbierane w latach **1998**-**2022**
  - prawdopodobnie wiele maili pochodzi z wczesnych lat 2000
    - (bardzo dużo http, nawet wśród bezpiecznych domen)
- Człowiek jakkolwiek obeznany w internecie bez problemu poradziłby sobie z klasyfikowaniem ataków **phishingowych** z datasetu.
