Statystyczne reguły decyzyjne - projekt zaliczeniowy

Szkoła Główna Handlowa w Warszawie

Alan Kashkash
Kamil Książek
Magdalena Kollar

Wprowadzenie

Celem niniejszego projektu była maksymalizacja zysku ze sprzedaży internetowej pewnej sieci księgarni przy nadchodzącej kampanii marketingowej na podstawie danych źródłowych w postaci bazy danych liczącej około 10 tysięcy klientów.
Projekt został zrealizowany z użyciem języka programowania Python. Na początku na podstawie rodzaju posiadanych informacji określono postać funkcji celu, następnie zwizualizowano dane z bazy oraz przeprowadzono modelowanie, końcowo uzyskując wartość zysku według zbudowanego modelu.

Analiza danych

Na początek wczytano dane źródłowe z bazy *klienci.txt.*

Wykres 1. Podgląd pierwszych dziesięciu rekordów z bazy *klienci txt*.

numer	wiek	plec	wyksztalcenie	miejsce	kwota	wojewodztwo	wysylka_oferty	zakup
1	32	1	srednie	wies	110.810558	dolsl	0	0
2	26	1	wyzsze	m100_500	100.857513	zachpom	0	0
3	20	0	srednie	m100_500	66.199017	zachpom	0	0
4	18	1	policealne	m100_500	77.017056	sl's	1	0
5	26	1	srednie	m100_500	137.221167	zachpom	0	0
6	33	0	wyzsze	m0_10	46.609796	dolsl	0	0
7	18	1	srednie	m_500	51.622458	dolsl	0	0
8	20	0	podstawowe	m0_10	28.060977	dolsl	1	1
9	22	0	policealne	m10_100	102.594906	dolsl	0	0
10	26	1	srednie	wies	74.678722	dolsl	0	0

Każdy z klientów ma przypisany wiek, płeć, miejsce zamieszkania, kwotę, na którą dokonał zakupu w okresie 1 (czyli okresie tworzenia bazy), województwo oraz dwie zmienne o charakterze binarnym: *wysylka_oferty*, która przyjmuje wartość 0, jeśli do klienta nie wysłana została oferta lub 1, jeśli taką ofertę otrzymał, a także *zakup*, którego wartość równa 0 oznacza finalny brak zakupu oraz 1 przy finalnym zakupie. Zakup jest jednocześnie naszą zmienną celową.

Dzięki tym założeniom wywnioskowano następujące założenia, będące podwalinami do późniejszego modelu:

```
# 1. if wysylka = 1 & zakup = 1 TAK!
# 2. if wysylka = 1 & zakup 0 NIE WYSYLAMY!
```

```
# 3. if wysylka = 0 & zakup = 1 TAK!
# # 4. if wysylka = 0 & zakup = 0 NIE WIEMY
# 4a if wysylka = 1 & zakup = 1 TAK!
# 4b if wysylka = 1 & zakup = 0 NIE WYSYLAMY!
```

Jak widać powyżej, możliwe konfiguracje wartości, które mogą przybrać razem zmienna *wysylka_oferty* oraz *zakup* podzielono na cztery możliwości. Na podstawie opcji nr 1, 2 i 3 (rekordy, które do nich łącznie przynależą liczą łącznie 2,5 tysiąca) sporządzono jeden model; rekordy przynależące do opcji 4 będą wykorzystane później.

Jeśli klient dostał od nas ofertę, a następnie dokonał zakupu, to według modelu takiemu klientowi będziemy wysyłać ofertę. Jeśli pomimo otrzymania oferty klient nie dokonał zakupu, to takiemu klientowi według modelu nie będziemy chcieli wysyłać reklamy. Jeśli pomimo tego, że nie wysłaliśmy oferty klient i tak dokonał zakupu, to model ma go wskazywać jako odbiorcę naszej reklamy (kierowaliśmy się tu założeniem, że klient jest dobrze rokujący i warto w niego inwestować).
Końcowo, w przypadku gdy klient ani nie dostał oferty, ani nie dokonał zakupu, kwalifikuje się on do wspomnianej opcji 4., stanowiącej inny podzbiór, liczący 7,5 tysiąca rekordów.

Przystąpiono do analizy podzbioru składającego się z rekordów zakwalifikowanych do opcji 1, 2 oraz 3.

ze wszystkich osób którym wysłaliśmy oferty 36% kupiło.
breakdown całego 123 pod wzgledem czy wysłaliśmy czy nie.

```
data.loc[(data.wysylka_oferty == 0)].zakup.value_counts(normalize=True)
```

czy wszyscy ci ktorym nie wyslalismy w 12 kupili? tak. upewnial sie.

```
data2 = data.loc[(data.wysylka_oferty==0)]
data2.zakup.value_counts(normalize=False)
```

```
data3 = data.loc[(data.wysylka_oferty==1)]
data3.zakup.value_counts(normalize=False)
```

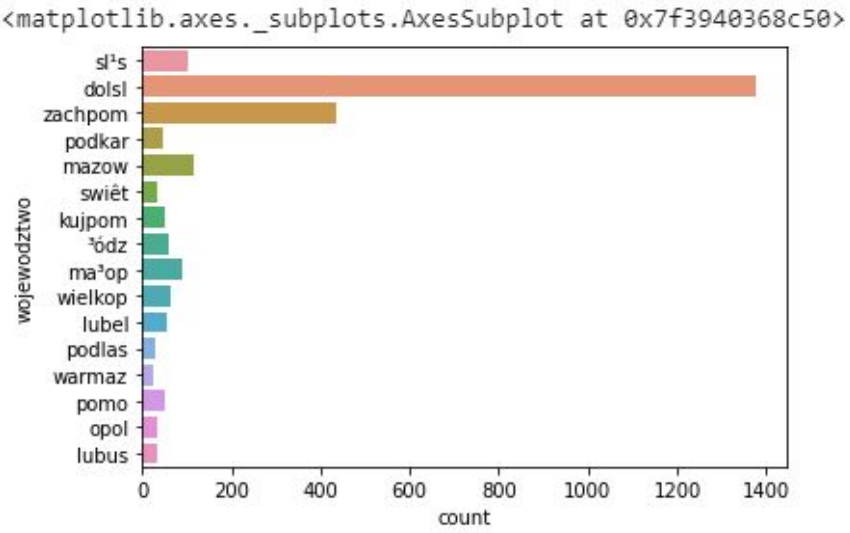
breakdown wszystkich co im wysłaliśmy czy kupili czy nie

Tabela 2. Podstawowe charakterystyki podzbioru.

	wiek	plec	kwota	wysylka_oferty	zakup
count	2597.000000	2597.000000	2597.000000	2597.000000	2597.000000
mean	42.251444	0.370812	83.784958	0.772430	0.507124
std	15.127282	0.483115	62.371061	0.419344	0.500046
min	18.000000	0.000000	0.000000	0.000000	0.000000
25%	29.000000	0.000000	41.963055	1.000000	0.000000
50%	42.000000	0.000000	73.450104	1.000000	1.000000
75%	53.000000	1.000000	115.098220	1.000000	1.000000
max	82.000000	1.000000	420.190823	1.000000	1.000000

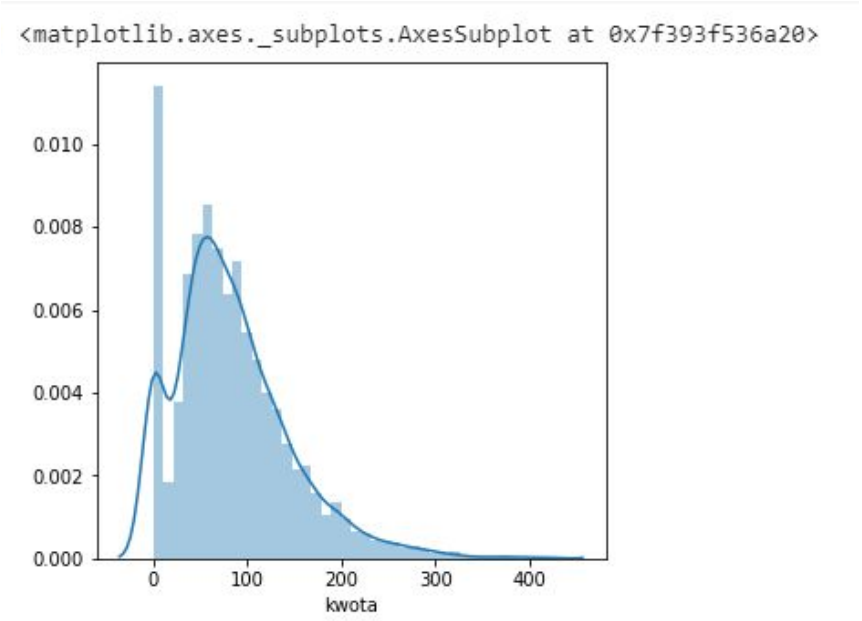
Następnie funkcją
`data.isna().sum()`
sprawdzono, czy w podzbiorze występują jakiekolwiek braki danych. Jako, że nie ma braków danych, przystąpiono do następnego kroku.

Wykres 1. Rozkład udziału poszczególnych województw w podzbiorze danych.



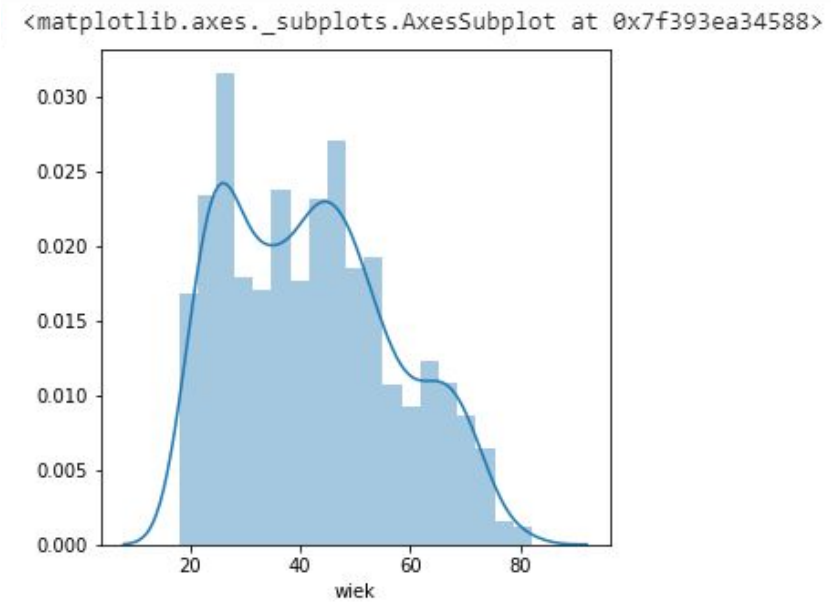
Na podstawie Wykresu 1. można wnioskować, że siedziba sklepu prawdopodobnie znajduje się w województwie dolnośląskim.

Wykres 2. Rozkład zmiennej ciągłej *kwota*.



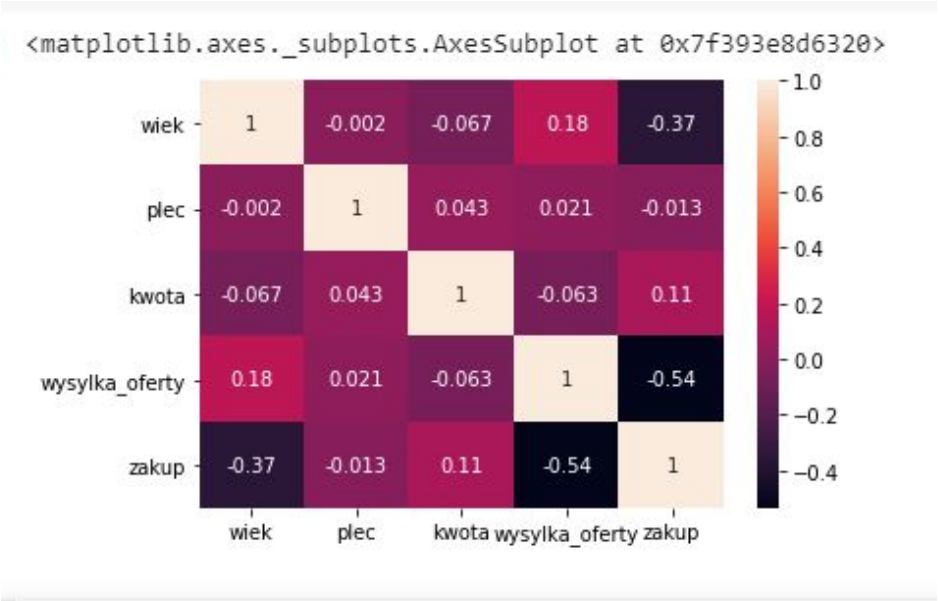
Kształt rozkładu zmiennej *kwota* przypomina rozkład normalny prawostronnie asymetryczny - za wyjątkiem wartości 0, w której zawierają się wszyscy ci, którzy założyli konto w sklepie internetowym księgarni, jednak nie dokonali żadnego zakupu w okresie 1 (ale została im wysłana oferta). Było to 296 osób, czyli 11.4% wszystkich osób z analizowanego podzbioru danych.

Wykres 3. Rozkład zmiennej ciągłej *wiek*.



Z kolei rozkład zmiennej *wiek* przybiera niesprecyzowany kształt rozkładu wielomodalnego, z ekstremami lokalnymi funkcji gęstości rozkładu w okolicach dwudziestu kilku, czterdziestu kilku oraz sześćdziesięciu kilku lat użytkowników sklepu.

Wykres 4. Macierz korelacji zmiennych przedstawiona na heatmapie



Można zaobserwować, że na granicy istotnej statystycznie korelacji balansuje jedynie ta między zmienną binarną przenoszącą informację o wysyłce oferty a zakupem.

Po dokonaniu powyższych obserwacji stosowano OneHotEncoder w celu zamiany zmiennych kategorycznych (*zakup, województwo, wysylka_oferty*) na zmienne numeryczne kolejno w podzbiorze dla opcji 1, 2 i 3 i podzbiorze dla opcji 4, zakładając, że pozwoli to uzyskać precyzyjniejsze wyniki. Natomiast gdyby zestaw danych był obszerniejszy, to nie zdecydowalibyśmy się na użycie takiego narzędzia - na tej wielkości zbiorach proces przeprowadzony został relatywnie szybko (ok. kilkunastu sekund).

Model

Podzielono zbiór 2,6 tysiąca klientów na podzbiór treningowy i testowy:

```
test = pd.read_csv("/content/klienci_final.txt", header=0, encoding='cp1252')
test = test.set_index("numer")
```

Podgląd pierwszych pięciu rekordów zbioru testowego (przed zamianą zmiennych kategorycznych na numeryczne) przedstawiono w Tabeli 3.

Tabela 3. Fragment zbioru testowego.

numer	wiek	plec	wyksztalcenie	miejsce	kwota	wojewodztwo
1	35	0	podstawowe	wies	97.965209	zachpom
2	26	1	podstawowe	wies	38.377882	dolsl
3	31	0	srednie	m10_100	136.401297	dolsl
4	25	0	policealne	m100_500	118.728067	dolsl
5	18	1	srednie	m_500	79.311481	dolsl

Zmienną dotyczącą wysyłki oferty usunięto, żeby uniknąć przeuczenia modelu (jednocześnie eliminując zagadnienie statystycznie istotnej korelacji pomiędzy tą zmienną a zmienną celową). Zmienna ta była również analogicznie usuwana w podzbiorach w następnych krokach.

W celu wybrania metody, która da jak najlepszą dokładność wypróbowano random forest, potem decision tree (które dało gorszy wynik), gradient boosting classifier, kończąc na ADA, która na tle pozostałych wypadła najlepiej (Tabela 4.).

Tabela 4. Porównanie uzyskanych wartości dokładności dla poszczególnych metod

Metoda	Random forest	Decision tree	Gradient boosting	ADA
Dokładność	72,9%	67,8%	76,2%	77,2%

Następnie wytrenowano więc analogicznie model na podzbiorze zawierającym rekordy dla opcji 4 metodą ADA, jako że miała ona największą dokładność przy poprzednim podzbiorze. Tak więc w linii poniżej robimy predykcję dla 7,5 tysiąca użytkowników sklepu internetowego, którzy nie dostali oferty ani nie dokonali zakupu.

```
data2_y = ada_clf.predict(data2_enc)
```

Następnie oba podzbiory połączono z powrotem w zbiór o nazwie data_final, zawierający 10 tysięcy rekordów i wytrenowano na nim model w celu zwiększenia ilości danych do trenowania.

```
X = data_final.drop(columns=["zakup"])
Y = data_final['zakup']

model2 = AdaBoostClassifier(n_estimators=800, base_estimator=logistic,
learning_rate=0.18, random_state=0)
model2.fit(X,Y)
```

Wtedy też można było ustalić predykcję dla zbioru liczącego 100 tysięcy rekordów.

```
predictions2 = model2.predict(test_enc)
```

Końcowo wyniki wyeksportowano do wektora w pliku w formacie .csv.

```
output = pd.DataFrame({"x": test_enc.index, "wysylka_oferty": predictions2})
output.to_csv("wysylka_tuned.csv", index=False)
```

Podsumowanie

Do przeprowadzenia predykcji do kogo należy wysłać ofertę marketingową wykorzystano zbiór danych, który następnie podzielono na 2 grupy. Grupę 2,6 tys. klientów, który składał się z klientów, którzy:

1. otrzymali wcześniej ofertę i dokonali zakupu bądź nie dokonali
2. mimo, że nie otrzymali oferty marketingowej zdecydowali się na dokonanie zakupu

Jest to segment klientów, przy którym mamy pewność co do wyników.

Oraz grupę 7.5 tys. klientów, którzy nie otrzymali wcześniej oferty zakupowej, ani nie zdecydowali się na dokonanie zakupu.

W powyższych zbiorach nie znaleziono żadnych brakujących danych. Po przeprowadzeniu EDA (Exploratory Data Analysis) i zbadaniu rozkładów zmiennych. Z uwagi na fakt, że zbiór zawierał tylko 3 zmienne kategoryczne, w których nie były wiele unikatowych wartości, zdecydowano o wykonaniu transformacji zmiennych kategorycznych z użyciem One Hot Encodera. Enkoder ten zapewnia większą dokładnośc modelu, natomiast w przypadku dużej ilości zmiennych kategorycznych i liczby unikatowych wartości w nich zawartych prowadzi do znacznego zwiększenia liczebności modelu, tym samym powodując wydłużenie czasu potrzebnego na wytrenowanie modelu. Dzieje się tak, ponieważ OH Encoder tworzy oddzielną kolumnę dla każdej unikatowej wartości każdej zmiennej kategorycznej, którą przekształcamy. Po transformacji zbioru “klienci.txt” powstało 25 nowych kolumn z wartościami binarnymi.

Następnie celem wyboru nalepszego modelu do zrobienia predykcji wyników, wykorzystano kilka algorytmów klasyfikujących, poczynając od najprostszego drzewa decyzyjnego, po coraz to bardziej zaawansowane metody z obszaru Ensemble Learning, celem uzyskania wyniku z najmniejszym błędem. Do porównania modeli użyta została metryka “Accuracy Score”, która jest używana do mierzenia wydajności algorytmów klasyfikujących.

Wykorzystane algorytmy oraz ich wyniki zostały opisane powyżej w cz. raportu opisującej “Model”. Modelem z najwyższym wynikiem w postaci 77,2% był algorytm `AdaBoostClassifier`. Poza standardowym użyciem modeli postanowiono także

poświęcić dużo uwagi optymalizacji ich hiperparametrów. Po wiele próbach ustalono, że najlepszy wynik jest otrzymywany przy użyciu parametrów, takich jak:

- liczba estymatorów w postaci 800
- Regresja Logistyczna jako podstawowy estymator
- learning rate na poziomie 0.18

Próbowano także wykorzystać inne modele jako bazowy estymator, jak np. Klasyfikator Naive Bayes, natomiast nie przyniosły one lepszych wyników niż regresja logistyczna.

Model wytrenowano na zbiorze 2,6 tys. obserwacji, który opisano powyżej, a następnie użyto go do zrobienia predykcji na zbiorze 7.5 tys. klientów, którzy nie otrzymali wcześniej oferty, celem zwiększenia liczebności zbioru treningowego do 10 tys. obserwacji.

Finalnie połączono obydwa zbiory, opisane powyżej (2,6tys. oraz 7.5 tys) a połączonego zbioru użyto do określenia, którym klientom należy wysłać ofertę robiąc predykcję na zbiorze testowym “klienci_final.txt”.

Niestety, finalnie mimo usilnych prób model wytrenowany na zbiorze 2.6 tys. klientów wykazał lepszy wynik w postaci 2,424,530 zł, aniżeli model wytrenowany na całych 10 tys. obserwacji, który wyniósł 2,396,510 zł.

Kod użyty do wykonania analizy

```
# -*- coding: utf-8 -*-
"""SRD projekt

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/19A0wz3xvezpUxPqfb95XkU9kZ5-Yu5IA
"""

# Commented out IPython magic to ensure Python compatibility.
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
# %matplotlib inline

# wczytanie danych
data_start = pd.read_csv("/content/klienci.txt", header=0, encoding='cp1252', index_col="numer")
data_start.head(10)
```

```
# 1. if wysylka = 1 & zakup = 1 TAK!
# 2. if wysylka = 1 & zakup 0 NIE WYSYLAMY!
# 3. if wysylka = 0 & zakup = 1 TAK!
# # 4. if wysylka = 0 & zakup = 0 NIE WIEMY
# 4a if wysylka = 1 & zakup = 1 TAK!
# 4b if wysylka = 1 & zakup = 0 NIE WYSYLAMY!

# 1&3&4a = 1
# 2&4b = 0

#1 dataset ze zmienna wysylka_oferty
#2 dataset ze zmienna

data1a = data_start.loc[(data_start.wysylka_oferty == 1)]
data1b = data_start.loc[(data_start.wysylka_oferty == 0) & (data_start.zakup == 1)]
data = pd.concat([data1a, data1b])

data.loc[(data.wysylka_oferty == 1)].zakup.sum() / data.wysylka_oferty.sum()

data.wysylka_oferty.value_counts()

data.loc[(data.wysylka_oferty == 0)].zakup.value_counts(normalize=True)
# data.zakup.value_counts()

data2 = data.loc[(data.wysylka_oferty==0)]
data2.zakup.value_counts(normalize=False)

data3 = data.loc[(data.wysylka_oferty==1)]
data3.zakup.value_counts(normalize=False)

# podstawowe statystyki
data.describe()

# sprawdzenie czy są missingi
data.isna().sum()

# jakie mamy rodzaje zmiennych
data.dtypes

# ile mamy różnych wartości w danych dyskretnych
discrete_data = data.drop(columns=['wiek','kwota'])
for c in discrete_data.columns:
    print("---- %s ---" % c)
    print(data[c].value_counts(normalize=True))

# ile mamy poszczegolnych wartosci w zmiennej wojewodztwo
```

```
sns.countplot(y=data.wojewodztwo)

# średnio wydana kwota przez osoby, ktore otrzymaly oferte
got_offer = data.loc[(data.wysylka_oferty == 1)]
got_offer.kwota.mean()

# średnio wydana kwota przez osoby, ktore nie otrzymaly oferty
didnt_get_offer = data.loc[(data.wysylka_oferty == 0)]
didnt_get_offer.kwota.mean()

# Rozkład zmiennej ciągłej - kwota
plt.figure(figsize=(5,5))
sns.distplot(data.kwota)

# ile osób nic nie kupiło
nothing = sum(data.kwota==0)
procentowo = round(nothing /data.shape[0] *100,2)
print(f"Nie kupiło nic {nothing} osób \n")
print(f"Czyli {procentowo}% wszystkich osób")

# Rozkład zmiennej ciągłej - Wiek
plt.figure(figsize=(5,5))
sns.distplot(data.wiek)

# Macierz korelacji danych przedstawiona na heatmapie
corrMatrix = data.corr()
sns.heatmap(corrMatrix, annot=True)

# wgrywamy dataset testowy
test = pd.read_csv("/content/klienci_final.txt", header=0, encoding='cp1252')
test = test.set_index("numer")
test.head()

# zamiana zmiennych kategorycznych na liczbowe z użyciem OH Encodera
from sklearn.preprocessing import OneHotEncoder
from sklearn import preprocessing
categorical_cols = ['wyksztalcenie', 'miejsce', 'wojewodztwo']
OH_encoder = OneHotEncoder(handle_unknown='ignore', sparse=False)
OH_cols_train = pd.DataFrame(OH_encoder.fit_transform(data[categorical_cols]))
OH_cols_valid = pd.DataFrame(OH_encoder.transform(test[categorical_cols]))

OH_cols_train.index = data.index
OH_cols_valid.index = test.index

# Remove categorical columns (will replace with one-hot encoding)
num_X_train = data.drop(categorical_cols, axis=1)
```

```
num_X_valid = test.drop(categorical_cols, axis=1)

# Add one-hot encoded columns to numerical features
data_enc = pd.concat([num_X_train, OH_cols_train], axis=1)
test_enc = pd.concat([num_X_valid, OH_cols_valid], axis=1)

data_enc.head()

x = data_enc.drop(columns=["wysylka_oferty","zakup"])
y = data_enc['zakup']

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split

# training the model only on training data split
x_train, x_test, y_train, y_test = train_test_split(x,y, random_state=0)

from sklearn import metrics
from sklearn.metrics import r2_score
model = RandomForestClassifier(n_estimators=80, random_state=0)
model.fit(x_train,y_train)
predictions = model.predict(x_test)
metrics.accuracy_score(y_test, predictions)

from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(random_state=0)
clf.fit(x_train,y_train)
preds = clf.predict(x_test)
metrics.accuracy_score(y_test, preds)

clf = GradientBoostingClassifier(n_estimators=88,learning_rate=0.18, random_state=0)
clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)
metrics.accuracy_score(y_test, y_pred)

import xgboost as xgb

gbm = xgb.XGBClassifier(
    learning_rate = 0.1,
 n_estimators= 4000,
 max_depth= 6,
 min_child_weight= 3,
 #gamma=1,
 gamma=0.5,
 subsample=0.8,
```

```
 colsample_bytree=0.8,
 objective= 'binary:logistic',
 nthread= -1,
 scale_pos_weight=1).fit(x_train, y_train)
predictions = gbm.predict(x_test)
print(metrics.accuracy_score(y_test, predictions))

from sklearn.ensemble import AdaBoostClassifier,GradientBoostingClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from time import time
# Import Support Vector Classifier
# from sklearn.svm import SVC
# svc=SVC(probability=True, kernel='linear')
# gnb = GaussianNB()
logistic = LogisticRegression()
start = time()
ada_clf = AdaBoostClassifier(n_estimators=800, base_estimator=logistic,
learning_rate=0.18, random_state=0)
ada_clf.fit(x_train, y_train)
end=time()

ada_y_pred = ada_clf.predict(x_test)
# print(ada_clf.get_params())
print(metrics.accuracy_score(y_test, ada_y_pred))
print(f'training time: {end-start}')

data2 = data_start.loc[(data_start.wysylka_oferty == 0) & (data_start.zakup == 0)]
data2_pre = data2.drop(columns=['wysylka_oferty','zakup'])
data2_pre.shape

# data2 encoding
categorical_cols2 = ['wyksztalcenie', 'miejsce', 'wojewodztwo']
OH_cols_train2 = pd.DataFrame(OH_encoder.fit_transform(data2_pre[categorical_cols2]))

OH_cols_train2.index = data2_pre.index

# Remove categorical columns (will replace with one-hot encoding)
num_X_train2 = data2_pre.drop(categorical_cols, axis=1)

# Add one-hot encoded columns to numerical features
data2_enc = pd.concat([num_X_train2, OH_cols_train2], axis=1)

data2_y = ada_clf.predict(data2_enc)
data2_enc['zakup'] = data2_y
```

```
data_enc = data_enc.drop(columns=["wysylka_oferty"])

data2_enc.zakup.sum()

data_final = pd.concat([data_enc, data2_enc])

data_final.zakup.sum()

X = data_final.drop(columns=["zakup"])
Y = data_final['zakup']

# training on full data (2 csv files)

model2 = AdaBoostClassifier(n_estimators=800, base_estimator=logistic,
learning_rate=0.18, random_state=0)
model2.fit(X,Y)

# finally we can predict our results
predictions2 = model2.predict(test_enc)

output = pd.DataFrame({"x": test_enc.index, "wysylka_oferty": predictions2})
output.to_csv("wysylka_tuned.csv", index=False)
```