Wydział Matematyki i Nauk Informacyjnych
Politechnika Warszawska

Wstęp do uczenia maszynowego (projekt grupowy)

Sprawozdanie z projektu

Autorzy:
Mikołaj Jakubowski, Marceli Korbin, Mariusz Słapek

Prowadzący:
inż. Paulina Tomaszewska

Warszawa, 2020

Spis treści

I. Analiza i obróbka danych

II. Modele

Część I

Analiza i obróbka danych

1. Analiza danych

1.1. Zbiory danych

W tym projekcie analizujemy zbiór danych *german_credit_data* informujący o kredytach branych w Niemczech w roku 1994 i ich kredytobiorcach. Dane pobieramy online w postaci dwóch plików CSV, dostępnych pod adresami: `www.mldata.io/download-csv-weka/german_credit_data` i `www.mldata.io/download-attributes/german_credit_data`. Pierwszy z plików jest naszym zbiorem danych, a drugi zestawem atrybutów opisujących zmienne.

1.2. Wstępna analiza danych

Zbiór danych posiada 21 kolumn i 1000 wierszy. Kolumny i ich typy:

```
| checking_account_status - object
| duration - int
| credit_history - object
| purpose - object
| credit_amount - float
| savings - object
| present_employment - object
| installment_rate - float
| personal - object
| other_debtors - object
| present_residence - float
| property - object
| age - float
| other_installment_plans - object
| housing - object
| existing_credits - float
| job - object
| dependents - int
| telephone - int*
| foreign_worker - int*
| customer_type - int
```

Kolumny oznaczone gwiazdką oryginalnie były typu `object`, ale zawierają tylko dwie unikalne wartości, zatem zmieniamy im typ na `int`. Nigdzie w zbiorze **nie stwierdziliśmy braków danych**.

1.2.1. Analiza danych numerycznych

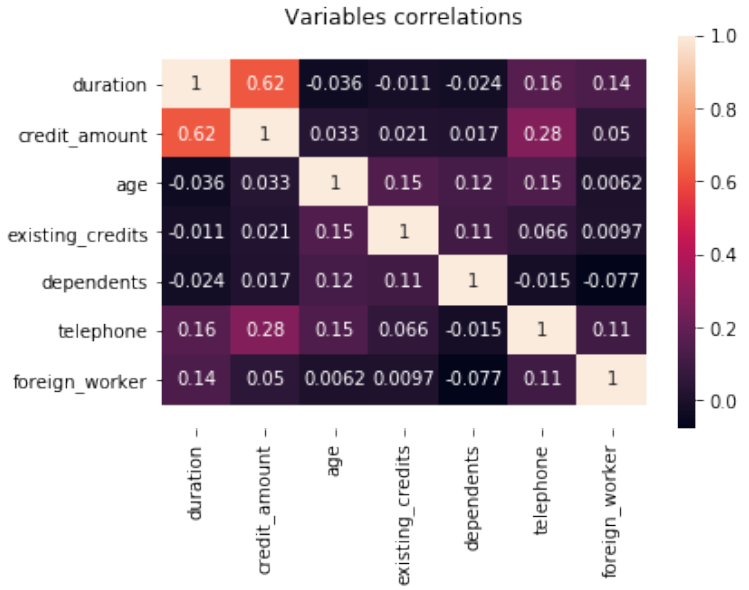
	duration	credit_amount	age	existing_credits	dependents	telephone	foreign_worker
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	20.903000	3271.258000	35.546000	1.407000	1.155000	0.404000	0.963000
std	12.058814	2822.736876	11.375469	0.577654	0.362086	0.490943	0.188856
min	4.000000	250.000000	19.000000	1.000000	1.000000	0.000000	0.000000
25%	12.000000	1365.500000	27.000000	1.000000	1.000000	0.000000	1.000000
50%	18.000000	2319.500000	33.000000	1.000000	1.000000	0.000000	1.000000
75%	24.000000	3972.250000	42.000000	2.000000	1.000000	1.000000	1.000000
max	72.000000	18424.000000	75.000000	4.000000	2.000000	1.000000	1.000000

Wnioski:

— Tylko 3,7% kredytobiorców pracuje za granicą (kolumna foreign_worker).
— W kolumnie credit_amount są znaczące outliery.

Sprawdziliśmy ponadto, że zbiór danych nie jest zbalansowany. W zmiennej odpowiedzi - customer_type - 70% rekordów ma wartość opisaną jako pozytywna.

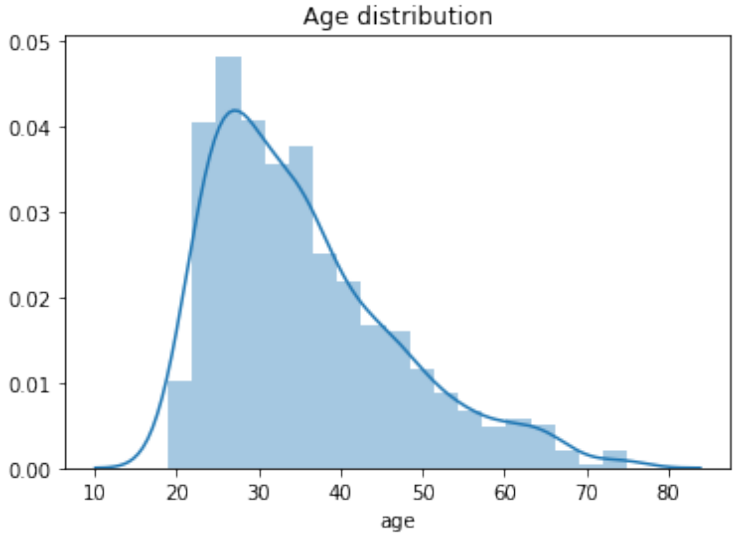
1.3. Macierz korelacji zmiennych numerycznych

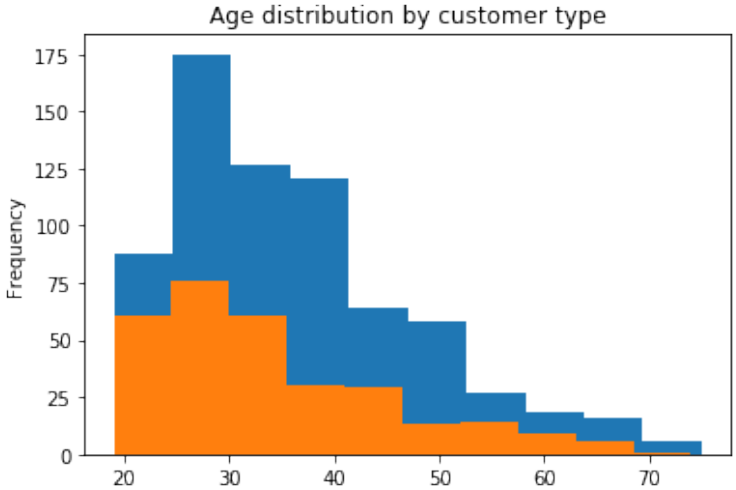


Zgodnie z naszymi przypuszczeniami, największą korelację można odnotować między zmiennymi credit_amount i duration.

1.4. Zaawansowana analiza - dane numeryczne

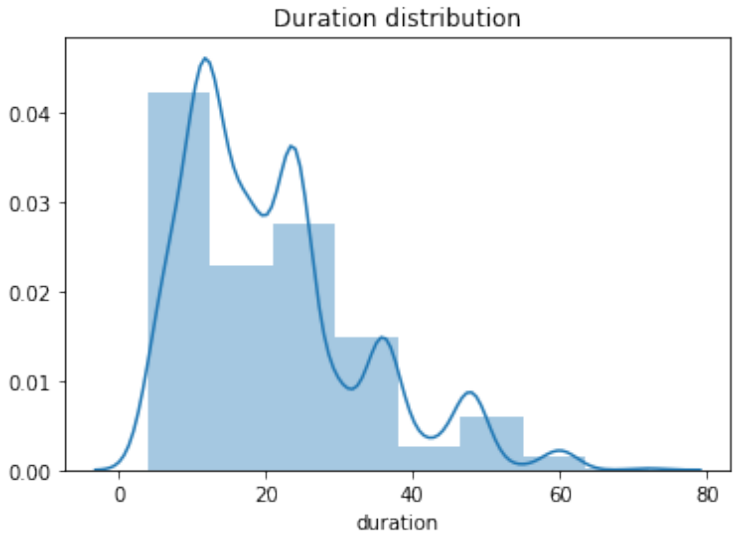
1.4.1. Rozkład wieku klientów - age

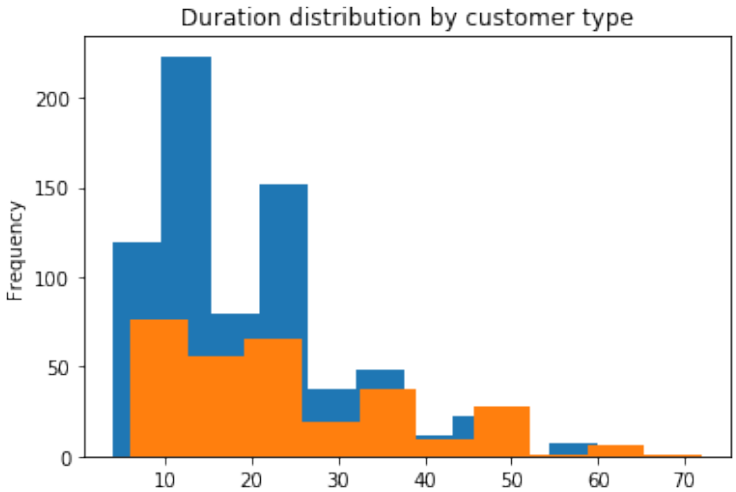




Kredyt jest brany głównie przez 30-latków; od wieku kredytobiorcy nie zależy jego wiarygodność.

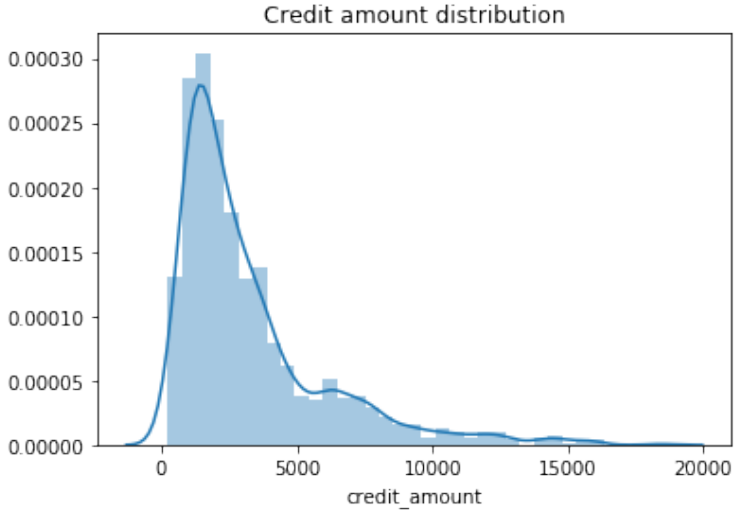
1.4.2. Rozkład okresu kredytu - `duration`

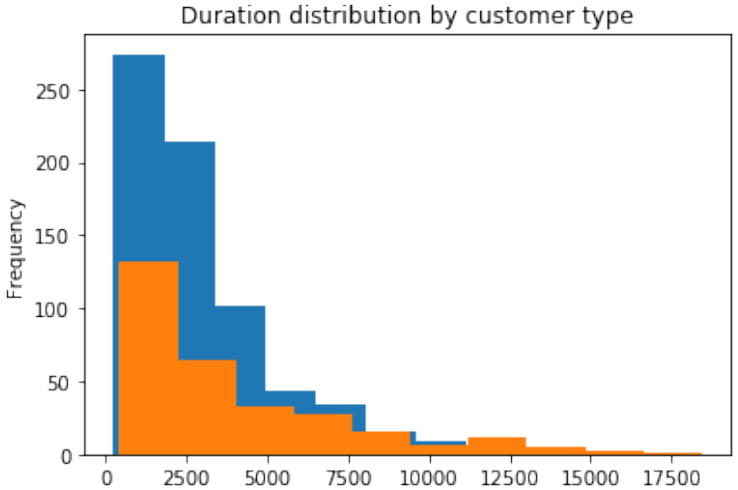




Rozkład ten jest nieregularny. Najczęściej kredyty bierze się na okres 12 miesięcy, a te o krótszym okresie są wcześniej spłacane.

1.4.3. Wysokości kredytu - credit_amount





Pierwszy wykres bliżej przedstawia mocną korelację między wysokością kredytu a okresem jego wzięcia. Niższe kredyty częściej się spłaca z powrotem.

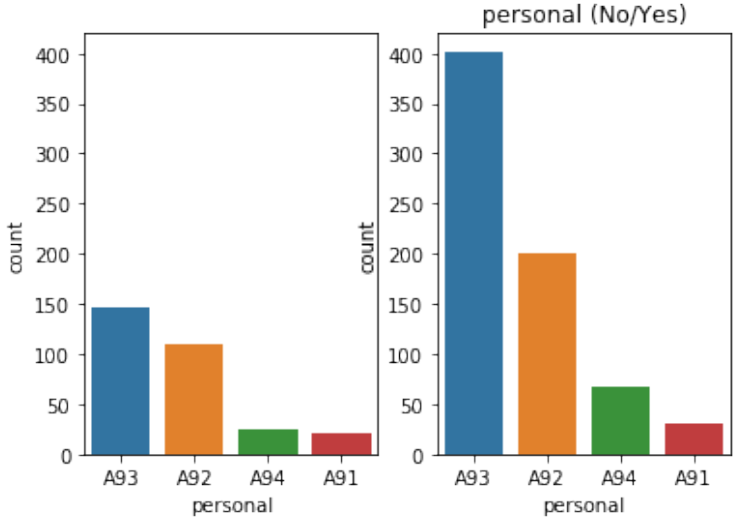
1.5. Zaawansowana analiza - dane kategoryczne

Dane kategoryczne analizujemy pod kątem zmiennej odpowiedzi.

1.5.1. Dane osobiste - `personal`

Zmienna opisuje płeć i stan małżeński. Wartości:

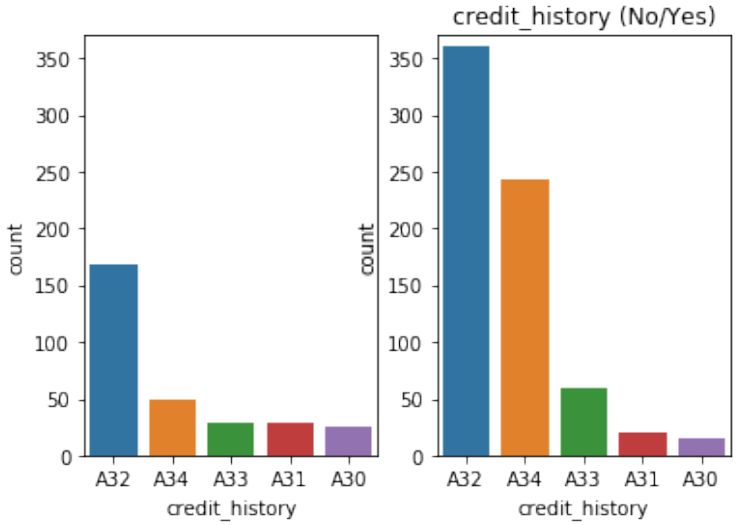
A91 Mężczyzna po rozwodzie lub w separacji
A92 Kobieta w małżeństwie, po rozwodzie lub w separacji
A93 Kawaler
A94 Mężczyzna w małżeństwie lub wdowiec
A95 Panna



Nie ma klientek płci żeńskiej, które nigdy nie miały partnera. Ponadto zmienna A91 nie podąża za trendem 70/30.

1.5.2. Historia pobierania kredytów - `credit_history`

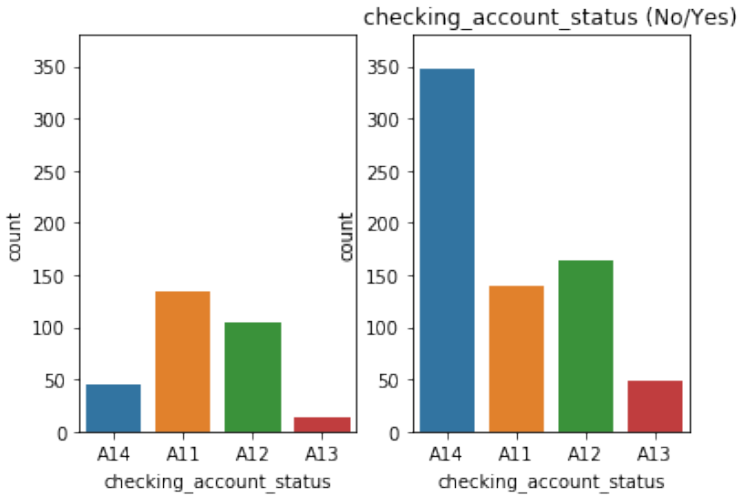
A30 Nie brano kredytów lub wszystkie spłacono na czas
A31 Wszystkie kredyty w tym banku spłacono na czas
A32 Istniejące kredyty jak dotąd spłacono na czas
A33 Opóźnienie w spłacaniu w przeszłości
A34 Krytyczna wartość lub kredyty istniejące w innych bankach



W przeciwieństwie do naszych oczekiwań, klienci, którzy dotychczas na czas spłacali kredyty, teraz tak niekoniecznie robią - i na odwrót.

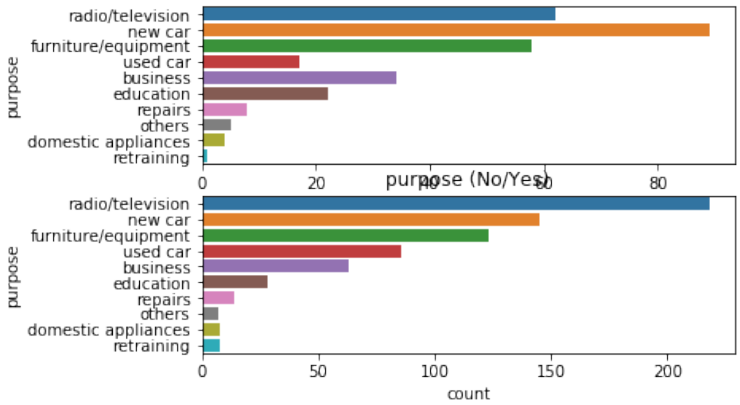
1.5.3. Rachunek bieżący - `checking_account`

A11 ujemny
A12 od 0 do 200 DM
A13 ponad 200 DM
A14 Brak rachunku bieżącego



Klienci bez rachunku bieżącego lub z rachunkiem o wysokiej wartości (ponad 200 DM) częściej spłacają kredyty. Również dla klientów o ujemnym rachunku bieżącym spłacenie kredytu jest bardziej prawdopodobne.

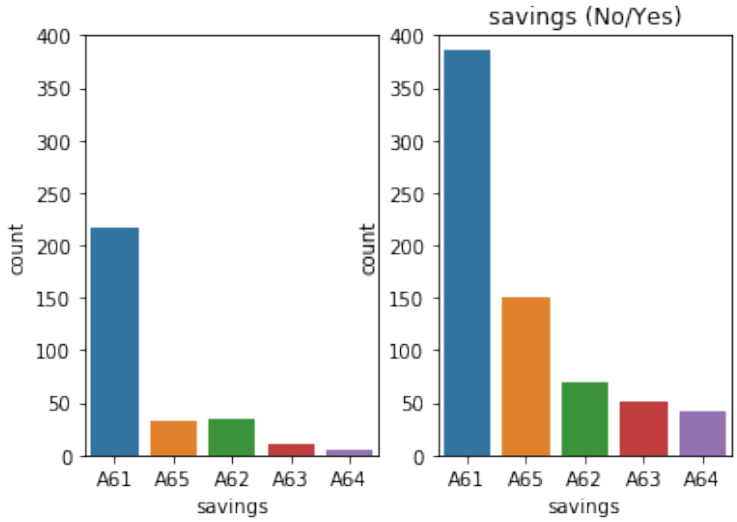
1.5.4. Cel skorzystania z kredytu - purpose



Co ciekawe, częściej się spłaca kredyt przeznaczany na używane auto, niż na nowe. Najniższą wiarygodność mają klienci o niestandardowym celu użycia kredytu (kategoria "others"), co trudno nam wytłumaczyć. Rzadziej spłaca się również kredyty przeznaczane na biznes lub edukację.

1.5.5. Oszczędności - savings

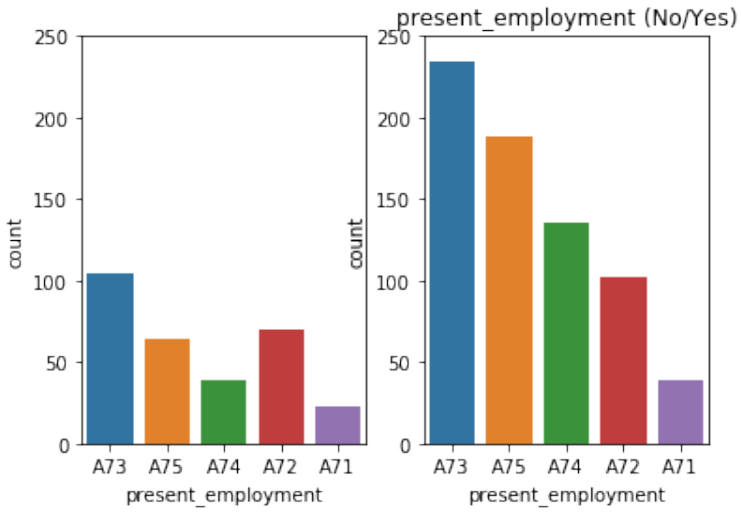
A61 do 100 DM
A62 od 100 do 500 DM
A63 od 500 do 1000 DM
A64 od 1000 DM
A65 brak lub nieznana ilość



Klienci częściej spłacają kredyt, jeśli mają większe oszczędności. Ci o nieznanych oszczędnościach również są bardziej wiarygodni.

1.5.6. Czas obecnego zatrudnienia - present_employment

A71 bezrobotni
A72 krócej niż rok
A73 od roku do 4 lat
A74 od 4 do 7 lat
A75 ponad 7 lat



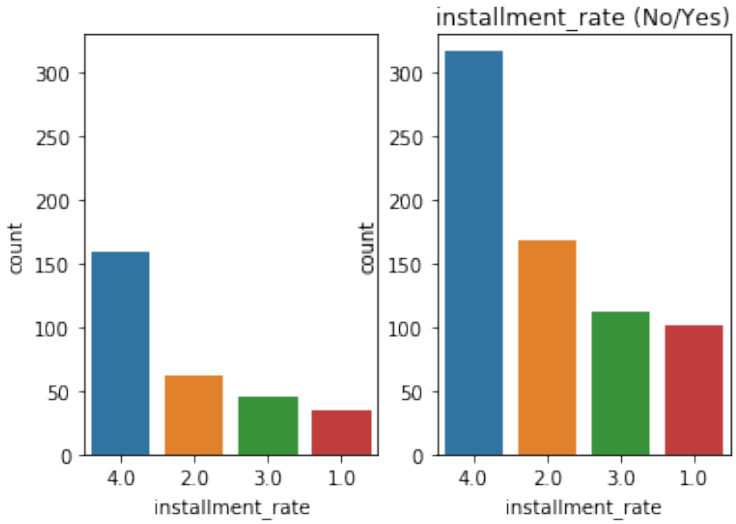
Kredytobiorcy bezrobotni lub pracujący krócej niż rok trochę rzadziej spłacają kredyt.

1.5.7. Rozkłady pozostałych zmiennych

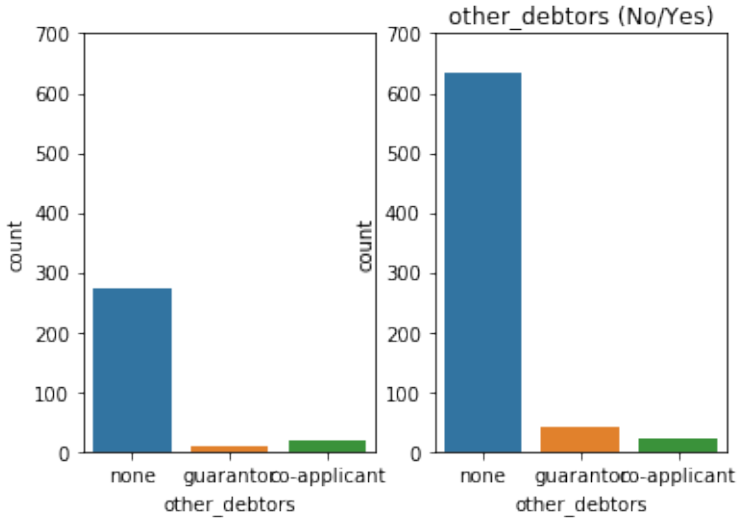
Pozostałe zmienne nie mają szczególnego wpływu na wyniki. Z ich analizy jesteśmy skłonni stwierdzić jeszcze następujące rzeczy:

— wśród klientów o innych dłużnikach, lepsi mają poręczyciela, a gorsi współdłużnika;
— kredytobiorcy o pewnym mieniu lub własnym mieszkaniu częściej spłacają kredyt.

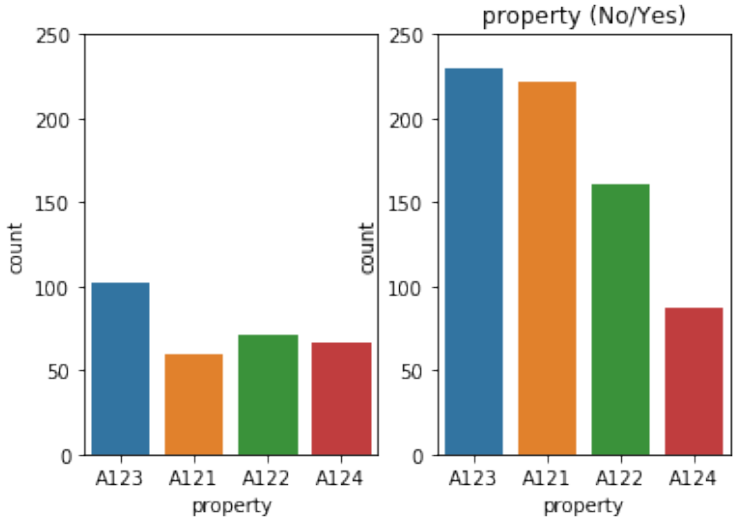
`installment_rate` - spłacalność



`other_debtors` - **inni dłużnicy**

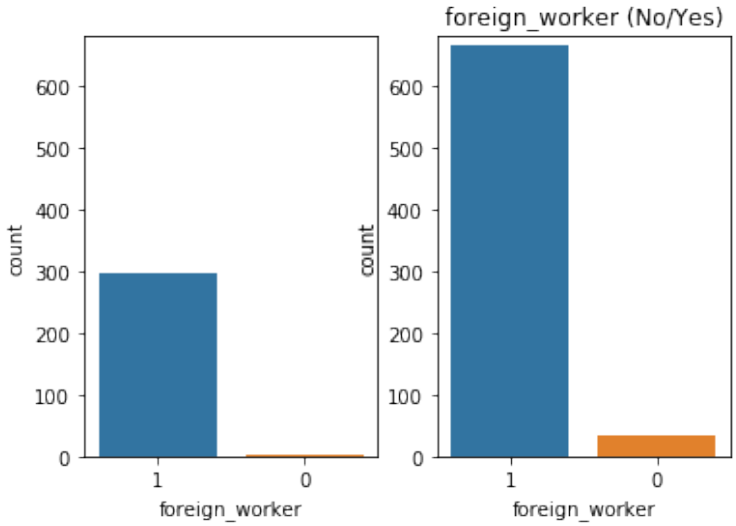


`property` - **mienie**



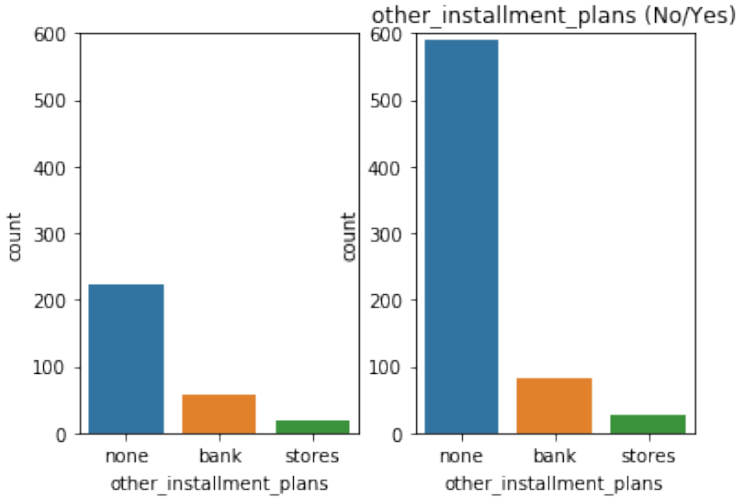
A121: nieruchomość, A122: umowa budowlano-oszczędnościowa lub ubezpieczenie na życie, A123: samochód, A124: brak mienia

`foreign_worker` - **czy klient pracuje za granicą**

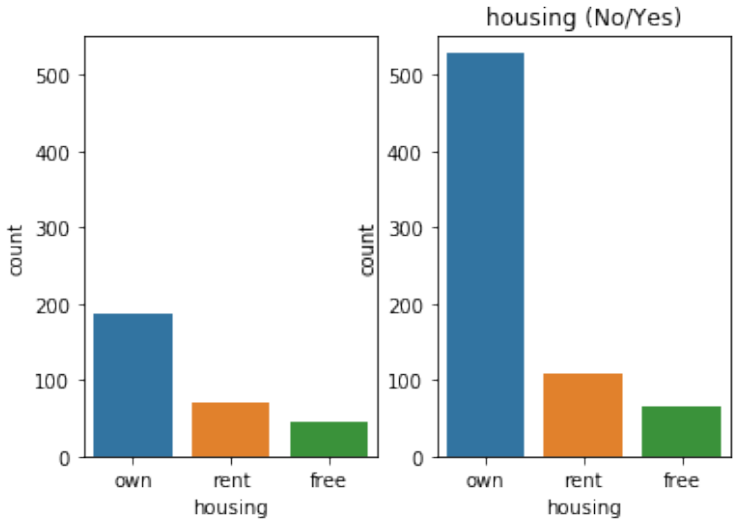


0: tak, 1: nie

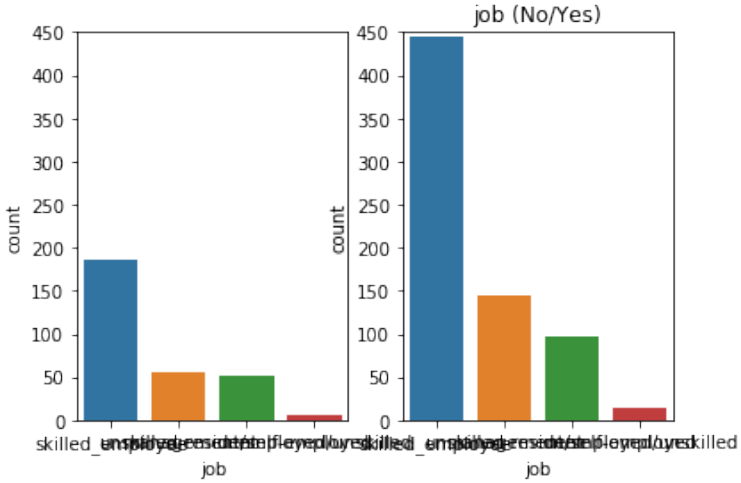
`other_installment_plans` - **inne plany spłacenia**



`housing` - **rodzaj zamieszkania**

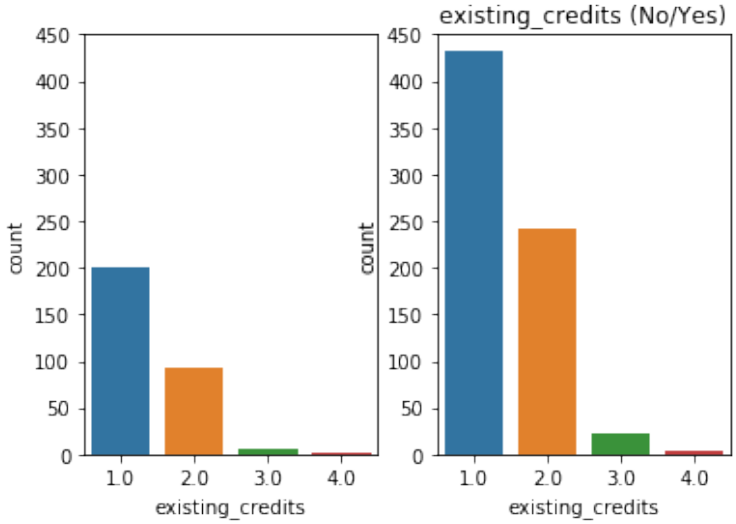


`job` - rodzaj zatrudnienia

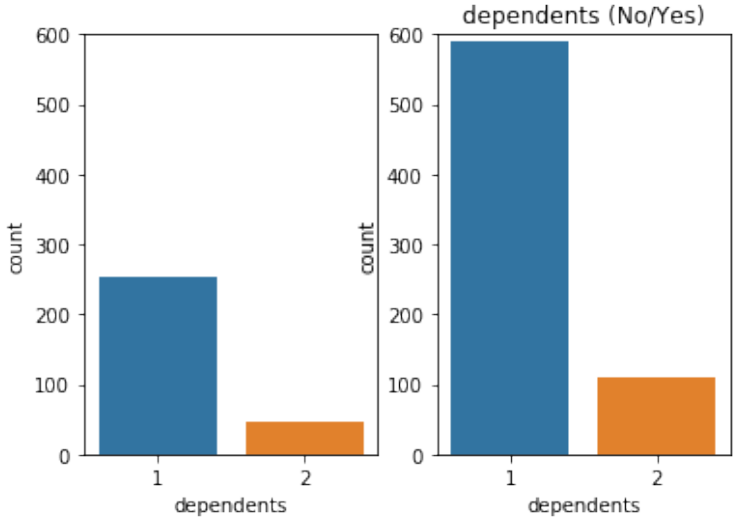


Kategorie od lewej: *skilled_employee*, *unskilled-resident*, *management/self-employed*, *unemployed/unskilled*.

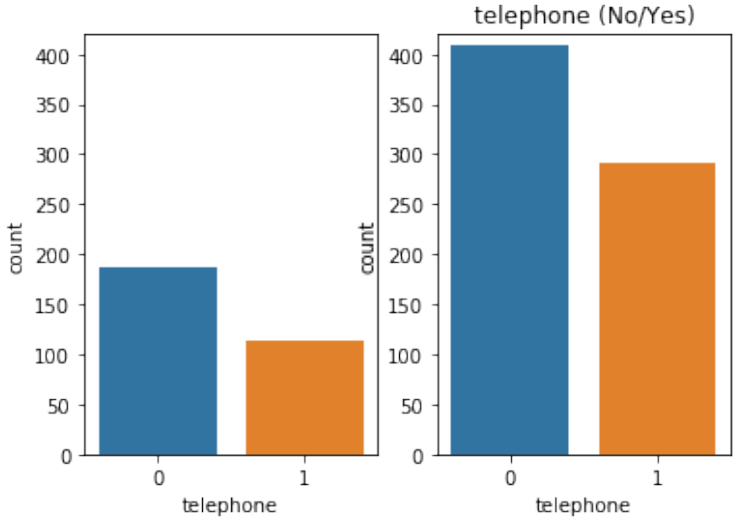
`existing_credits` - liczba kredytów wziętych w tym banku



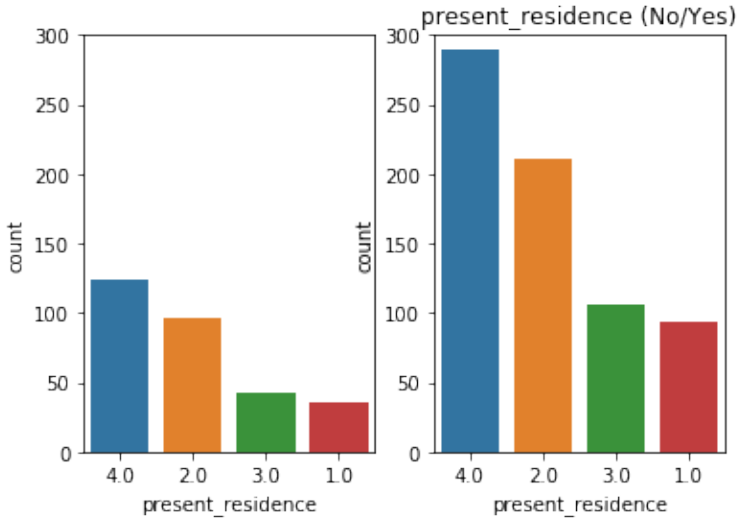
`dependents` - liczba utrzymywanych osób



`telephone` - obecność telefonu



`present_residence` - czas aktualnego zamieszkania w latach



2. Obróbka danych

Przed tworzeniem modelu potrzebujemy oczyścić niektóre dane. Na wykresach w poprzednim rozdziale już było widać niektóre zmiany, a tutaj wszystkie opiszemy.

2.1. Zmienna odpowiedzi

Wartości pozytywne i negatywne były zakodowane odpowiednio jako 1 i 2; dwójki zmieniamy na zera, dzięki czemu średnia z tej zmiennej staje się bezpośrednio miarą prawdopodobieństwa wiarygodności klienta.

```
data.customer_type.replace([1,2], [1,0], inplace=True)
```

2.2. Zmiana nazw wartości

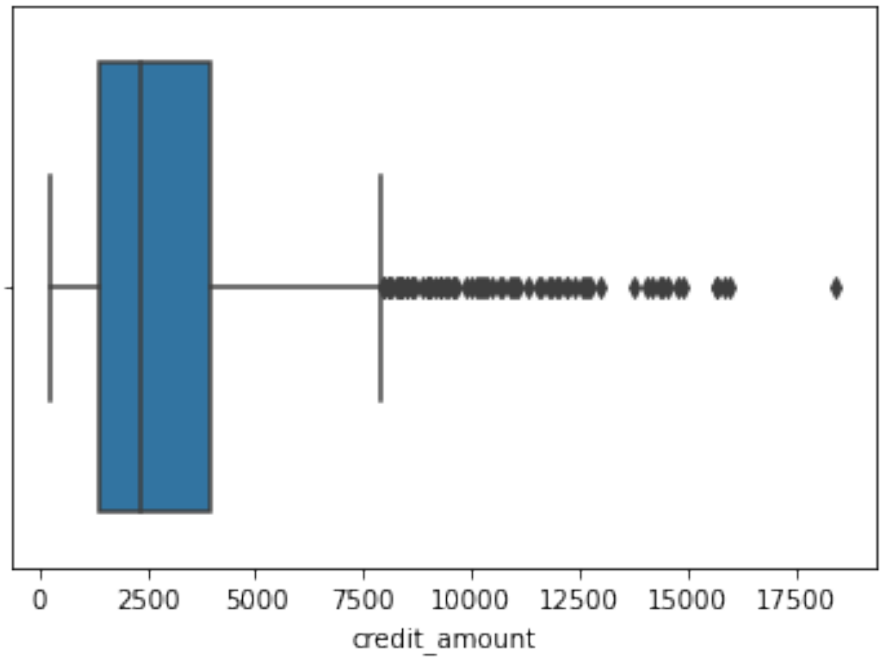
Siedmiu zmiennym kategorycznym zmieniamy nazwy wartości na bardziej opisowe. Zmiany dotyczą `telephone`, `foreign_worker`, `job`, `housing`, `purpose`, `other_debtors` i `other_installment_plans`.

```
data.telephone = data.telephone.map({"A191": 0, "A192": 1})
data.foreign_worker = data.foreign_worker.map({"A201": 1, "A202": 0})
data.job = data.job.map({
    "A171" : "unemployed/unskilled",
    "A172": "unskilled-resident",
    "A173": "skilled_employee",
    "A174": "management/self-employed"
})
data.housing = data.housing.map({
    'A151': 'rent',
    'A152': 'own',
    'A153': 'free'
})
data.purpose = data.purpose.map({
    'A40': 'new car',
    'A41': 'used car',
    'A42': 'furniture/equipment',
    'A43': 'radio/television',
    'A44': 'domestic appliances',
    'A45': 'repairs',
    'A46': 'education',
    'A47': 'vacation',
    'A48': 'retraining',
    'A49': 'business',
    'A410':'others'
})
data.other_debtors = data.other_debtors.map({
```

```
    'A101': 'none',
    'A102': 'co-applicant',
    'A103': 'guarantor'
})
data.other_installment_plans = data.other_installment_plans.map({
    'A141': 'bank',
    'A142': 'stores',
    'A143': 'none'
})
```

2.3. Outliery w kolumnie `credit_amount`

Tylko obserwowalne outliery mogą znaleźć się w kolumnie `credit_amount`. Przyjrzyjmy się im bliżej.



Średnią ze zmiennej odpowiedzi `customer_type` sprawdzamy prawdopodobieństwo spłacenia kredytu w zależności od zakresu wysokości kredytu. Wnioski:

— dla wysokości kredytu ponad 13000 DM: 30.77%
— ponad 10000 DM: 40%
— ponad 8000 DM: 45.71%
— ogólne prawdopodobieństwo: 70%
— mniej niż 10000 DM: 71.25%

Kredyty o wysokości większej niż 10000 DM pojawiają się jednak tylko 40 razy na 1000. Nie zmieniają zbytnio rozkładu prawdopodobieństwa wiarygodności klienta, a ich obecność może dostarczyć kilka przydatnych informacji i nie powinna wpłynąć negatywnie na skuteczność modeli, zatem te rekordy zostawiamy.

2.4. Nowe zmienne

Dodajemy do zbioru nowe zmienne na podstawie już istniejących cech: płeć, obecność rachunku bieżącego i obecność oszczędności.

```
data['sex'] = data.personal.apply(
    lambda x: 1 if x in ['A91', 'A93', 'A94'] else 0
    )
# male = 1, female = 0

data['checking_account_exists'] = np.where(
    data['checking_account_status']=='A14',
    0, 1)
data['savings_account_exists'] = np.where(
    data['savings']=='A65',
    0, 1)
```

2.5. Normalizacja zmiennych ciągłych

```
from sklearn import preprocessing
def normalize(df, columns):
    for column in columns:
        x = df[[column]].values.astype(float)
        min_max_scaler = preprocessing.MinMaxScaler()
        x_scaled = min_max_scaler.fit_transform(x)
        df[[column]] = x_scaled
    return df

data = normalize(data,
    ['duration', 'credit_amount', 'age',
    'installment_rate', 'present_residence',
    'dependents', 'existing_credits']
)

data.to_csv('../processed_data/out.csv', index=False)
```

Gotowe dane zapisujemy w nowym pliku. W dalszej części będziemy dodatkowo przekształcać zmienne w ramach kodowań.

Część II

Modele

3. Przygotowanie do modelowania

3.1. Podział danych

Żeby podzielić zbiór tak, że w obu częściach mamy te same ilości dużych i małych kredytów, musimy rozdzielić je na grupy. W tradycyjnym podziale randomizacja powoduje, że wyniki wyglądają na mniej zrównoważone.

```
from sklearn import preprocessing

x = data[['credit_amount']].values.astype(float)
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
data['amount_groups'] = x_scaled
bins = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8]
data['amount_groups'] = np.digitize(data['amount_groups'], bins)
unique, counts = np.unique(data['amount_groups'], return_counts=True)
dict(zip(unique, counts))
> {1: 445, 2: 293, 3: 97, 4: 80, 5: 38, 6: 19, 7: 14, 8: 8, 9: 6}

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    data.drop(['customer_type', 'amount_groups'], axis=1),
    data.customer_type, test_size=0.20,
    stratify = data[['amount_groups', 'customer_type']])
```

3.2. Encoding

Tworzymy funkcję `multiEnc`, która pozwala zautomatyzować kodowanie zmiennych w zbiorze danych według przydzielonego im rodzaju kodowania, reprezentowanego przez pojedynczy znak dla kondensacji kodu. Funkcja obsługuje wszystkie 15 kodowań dostępnych w pakiecie `category_encoders`.

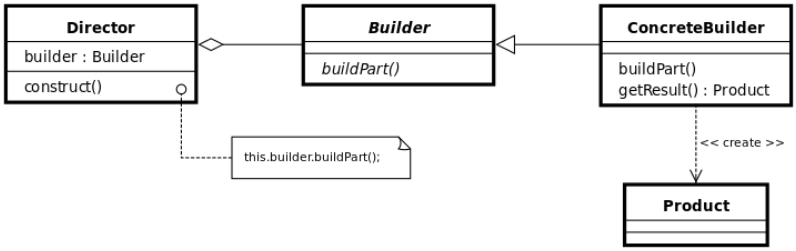
```
import category_encoders as ce
class Error(Exception):
    pass
class NonMatchingLengthsError(Error):
    pass
def multiEnc(X_train, X_test, target_train, cols, encodings):
    """
    Lista znak w do "encodings":
        d - backward difference
        n - base N
        b - binary
```

```
        c - cat boost
        # - hashing
        h - helmert
        j - James-Stein
        l - leave one out
        m - m-estimate
        1 - one-hot
        o - ordinal
        p - polynomial
        s - sum coding
        t - target encoding
        w - weight of evidence
    """
    ce_map = {"d": ce.backward_difference.BackwardDifferenceEncoder,
              "n": ce.basen.BaseNEncoder,
              "b": ce.binary.BinaryEncoder,
              "c": ce.cat_boost.CatBoostEncoder,
              "#": ce.hashing.HashingEncoder,
              "h": ce.helmert.HelmertEncoder,
              "j": ce.james_stein.JamesSteinEncoder,
              "l": ce.leave_one_out.LeaveOneOutEncoder,
              "m": ce.m_estimate.MEstimateEncoder,
              "1": ce.one_hot.OneHotEncoder,
              "o": ce.ordinal.OrdinalEncoder,
              "p": ce.polynomial.PolynomialEncoder,
              "s": ce.sum_coding.SumEncoder,
              "t": ce.target_encoder.TargetEncoder,
              "w": ce.woe.WOEEncoder}
    try:
        if len(cols)!=len(encodings):
            raise(NonMatchingLengthsError)
    except NonMatchingLengthsError:
        print("Lengths do not match")
        return None
    e=0
    for c in cols:
        if X_train[c].dtypes=='object':
            enc=ce_map[encodings[e]](cols=c)
            enc=enc.fit(X_train, target_train)
            X_train=enc.transform(X_train)
            X_test=enc.transform(X_test)
        e=e+1
    return (X_train, X_test)
```

3.3. Budowanie modeli

Teraz tworzymy klasy do generycznego budowania list modeli klasyfikacyjnych. W celu łatwej ewaluacji modeli skorzystaliśmy z Buildera (diagram klas niżej) - wzorca projektowego, który pozwala w łatwy sposób dodawać wiele modeli, a następnie ewaluować je na konkretnych zbio-

rach danych. Wykorzystujemy osiem rodzajów modeli: drzewo decyzyjne, regresję logistyczną, SVC, naiwny klasyfikator Bayesowski, lasy losowe, Ada Boost, Gradient Boosting i XGBoost.




```
from abc import (ABC,
                 abstractmethod,
                 abstractproperty)
from typing import Any
from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
import xgboost as xgb

class Builder(ABC):

    @abstractproperty
    def product(self) -> None:
        pass

class Product():

    def __init__(self) -> None:
        self.parts = []

    def add(self, part: Any) -> None:
        self.parts.append(part)

    def list_parts(self):
        return self.parts

class ConcreteBuilder(Builder):

    def __init__(self) -> None:
        self.reset()

    def reset(self) -> None:
        self._product = Product()
```

```
    @property
    def product(self) -> Product:
        product = self._product
        self.reset()
        return product

    def _add_model(self, model_type: str, params: dict):
        if model_type == 'logistic_regression':
            self._logistic_regression(params)
        if model_type == 'decision_tree':
            self._decision_tree(params)
        if model_type == 'svm':
            self._svm(params)
        if model_type == 'naive_bayes':
            self._naive_bayes(params)
        if model_type == 'random_forest':
            self._random_forest(params)
        if model_type == 'ada_boost':
            self._ada_boost(params)
        if model_type == 'gradient_boost':
            self._gradient_boost(params)
        if model_type == 'xgboost':
            self._xgboost(params)

    def _logistic_regression(self, params: dict):
        lr = LogisticRegression(**params)
        return self._product.add(lr)

    def _decision_tree(self, params: dict):
        dt = DecisionTreeClassifier(**params)
        dt_params = {'criterion': dt.criterion}
        return self._product.add(dt)

    def _svm(self, params: dict):
        svm = SVC(**params)
        return self._product.add(svm)

    def _naive_bayes(self, params: dict):
        nb = GaussianNB(**params)
        return self._product.add(nb)

    def _random_forest(self, params: dict):

        rf = RandomForestClassifier(**params)
        return self._product.add(rf)

    def _ada_boost(self, params: dict):
        ada = AdaBoostClassifier(**params)
        return self._product.add(ada)

    def _gradient_boost(self, params: dict):
```

```
        gb= GradientBoostingClassifier(**params)
        return self._product.add(gb)

    def _xgboost(self, params: dict):
        xg=XGBClassifier(**params)
        return self._product.add(xg)

class Director:

    def __init__(self) -> None:
        self._builder = None

    @property
    def builder(self) -> Builder:
        return self._builder

    @builder.setter
    def builder(self, builder: Builder) -> None:

        self._builder = builder


    def add_model(self, model_type, params):
        return self.builder._add_model(model_type, params)

    def add_all_models(self):
        self.add_model('logistic_regression', {})
        self.add_model('decision_tree', {})
        self.add_model('svm', {})
        self.add_model('naive_bayes', {})
        self.add_model('random_forest', {})
        self.add_model('ada_boost', {})
        self.add_model('gradient_boost', {})
        self.add_model('xgboost', {})
        return builder.product.list_parts()

    def get_all_models(self, metric_name: str = ''):
        parts =  builder.product.list_parts()
        return parts

director = Director()
builder = ConcreteBuilder()
director.builder = builder
```

3.4. Miary oceny modeli

3.4.1. Metryka biznesowa

Nasz zbiór danych jest natury biznesowej, w związku z czym zdecydowaliśmy się zaimplementować specjalną miarę. Jest to metryka stworzona na praktyczne potrzeby projektu. Zamiast

oceniać trafność algorytmów, ocenia ona, jak duże zyski dany algorytm jest w stanie przynieść bankowi. Wielkości kredytów, w zależności od tego, czy są spłacane, czy nie, przeliczane są przez odpowiednie współczynniki. Następnie obliczane jest, jak duży procent maksymalnego możliwego zysku wypracował algorytm.

Wykorzystujemy w tej metryce następujące parametry:

stopa procentowa = 13% Według 'Overseas Business Reports', U.S. Department of Commerce, Bureau of International Commerce, 1991 (rozdział: 'banking and credit') stopa procentowa dla hipoteki wynosiła około 10%, a dla innych pożyczek około 13%.

LGD = 38% 'Banking Systems Simulation: Theory, Practice, and Application of Modeling Shocks, Losses, and Contagion', Stefano Zedda (rozdział 2.10.1) Miara LGD (loss given default - strata z tytułu niewykonania zobowiązania) pomiędzy 1990 a 2008 wynosiła średnio 38%.

```
ir_loan = 0.13
lgd = 0.38
def calculateEarningsLosses(X_test, y_pred, y_test):
    '''
    As declared, takes test data and predicted classes
    and calculates:
    - earnings made by following prediction
    - losses made by following prediction
    - earnings omited by following prediction
    - losses omited by following prediction
    '''
    amounts = X_test['credit_amount']
    balance_all = y_test.apply(lambda x:
        ir_loan if x==1 else -lgd) * amounts
    earnings_made = balance_all.iloc[np.logical_and(y_test==1,
    y_pred==1).array].sum()
    earnings_omitted = balance_all.iloc[np.logical_and(y_test==1,
    y_pred==0).array].sum()
    losses_made = balance_all.iloc[np.logical_and(y_test==0,
    y_pred==1).array].sum()
    losses_omitted = balance_all.iloc[np.logical_and(y_test==0,
    y_pred==0).array].sum()

    results = pd.DataFrame(columns=['Earnings made',
    'Earnings omitted', 'Losses made', 'Losses omitted'])
    results.loc[0] = [earnings_made, earnings_omitted,
    losses_made, losses_omitted]

    final_balance = earnings_made + losses_made
    max_income = balance_all.iloc[(y_test==1).array].sum()
    perc_of_max_income = final_balance/max_income
    return (results, final_balance, perc_of_max_income)
```

3.4.2. Inne metryki

Oprócz `calculateEarningsLosses` wykorzystujemy miary `accuracy` i `F1`.

3.5. Funkcje do porównania modeli i sposobów kodowania

```
from typing import List
from sklearn.metrics import f1_score

def compare_models(models_list: List, X_train, y_train, X_test,
y_test, categorical_variables, encoding_list):
    results = dict()

    df_train, df_test = multiEnc(X_train, X_test, y_train,
    categorical_variables, encoding_list)

    for model in models_list:
        training = model.fit(df_train, y_train)
        score = training.score(df_test, y_test)
        f1 = f1_score(y_test, model.predict(df_test))
        _, _, business = calculateEarningsLosses(X_test,
        model.predict(df_test), y_test)
        results[model] = (['score:', score], ['f1:', f1],
        ['business:', business])

    return results

def compare_encoders(model, X_train, y_train, X_test, y_test,
columns_enc, enc_nominal, enc_ordinal):
    results = pd.DataFrame(columns = enc_nominal,
    index = enc_ordinal)
    results_f1 = pd.DataFrame(columns = enc_nominal,
    index = enc_ordinal)
    results_bus = pd.DataFrame(columns = enc_nominal,
    index = enc_ordinal)

    for nom in enc_nominal:
        for ordi in enc_ordinal:
            encoding_list = encoding_list_gen(nom, ordi)
            df_train, df_test = multiEnc(X_train, X_test, y_train,
            columns_enc, encoding_list)
            training = model.fit(df_train, y_train)
            score = training.score(df_test, y_test)
            f1 = f1_score(y_test, model.predict(df_test))
            _, _, business = calculateEarningsLosses(X_test,
            model.predict(df_test), y_test)

            results.loc[ordi, nom] = score
            results_f1.loc[ordi, nom] = f1
            results_bus.loc[ordi, nom] = business

    return (results, results_f1, results_bus)
```

4. Modelowanie właściwe

Nasze poprzednie eksperymenty wskazały, że najbardziej obiecującymi klasyfikatorami są XGBoost, Gradient Boosting, lasy losowe i SVC. Te cztery algorytmy testujemy teraz z różnymi rodzajami kodowania. Większość z nich jest oparta o drzewa klasyfikacyjne, więc np. ordinal encodingu używamy jedynie dla SVC.

4.1. Ostatnia funkcja pomocnicza

```
columns_enc = ['checking_account_status', 'credit_history',
    'purpose', 'savings', 'present_employment',
    'personal', 'other_debtors', 'property',
    'other_installment_plans', 'housing', 'job', 'telephone']
def encoding_list_gen(nominal, ordinal):
    enc = [ordinal, nominal, nominal, ordinal,
    nominal, nominal, nominal, nominal,
    nominal, nominal, nominal, nominal]
    return enc
#enc = ['j', '1', '1', 'j', '1', '1', '1', '1', '1', '1', '1', '1']
```

Grupujemy kolumny w numeryczne i kategoryczne, by wewnątrz tych grup kodować je tym samym typem.

4.2. Przygotowanie

```
director.add_model('gradient_boost', {})
director.add_model('xgboost', {})
director.add_model('random_forest', {})
director.add_model('svm', {})
models = director.get_all_models()
enc_nominal = ['l', 'j', 'm'] # zmienne kategoryczne
enc_ordinal = ['p', 'h', 'd', 'l', 'j'] # zmienne numeryczne
```

Odrzucamy kodowania, które prowadzą do utraty danych.

4.3. Ewaluacja

4.3.1. Miary oceny

Tabele reprezentują kolejno miarę accuracy, wynik F1 i naszą metrykę biznesową dla danych połączeń encodingów.

Gradient Boosting

Kodowania:

L leave one out
J James-Stein
M m-estimate
P polynomial
H Helmert
D backward difference

```
(score, f1, buss) = compare_encoders(models[0], X_train, y_train,
X_test, y_test, columns_enc, enc_nominal, enc_ordinal)
```

Do dalszych tabel wykorzystujemy analogiczne polecenie jak wyżej, zmieniając tylko model z listy i - w dwóch przypadkach - spis kodowań (ostatnie dwa parametry).

Każda kolumna w tabelach reprezentuje kodowanie zmiennych kategorycznych, a każdy wiersz - kodowanie zmiennych numerycznych.

accuracy	L	J	M
P	0.76	0.76	0.76
H	0.745	0.745	0.745
D	0.77	0.77	0.775
L	0.75	0.75	0.75
J	0.75	0.75	0.75

F1	L	J	M
P	0.833333	0.833333	0.833333
H	0.8223	0.8223	0.8223
D	0.840278	0.840278	0.843206
L	0.827586	0.827586	0.827586
J	0.827586	0.827586	0.827586

biznesowa	L	J	M
P	0.205109	0.205109	0.205109
H	0.12244	0.12244	0.12244
D	0.109999	0.109999	0.188288
L	0.0454018	0.0454018	0.0454018
J	0.0454018	0.0454018	0.0454018

XGBoost

Kodowania:

L leave one out
J James-Stein
M m-estimate

XGBoost nie działa najlepiej z kodowaniem zmiennych numerycznych, stąd mniejszy wybór encodingów.

accuracy	L	J	M
L	0.71	0.71	0.71
J	0.71	0.71	0.71

F1	L	J	M
L	0.8	0.8	0.8
J	0.8	0.8	0.8

biznesowa	L	J	M
L	-0.30158	-0.30158	-0.30158
J	-0.30158	-0.30158	-0.30158

Lasy losowe (Random Forest)

Kodowania takie same jak dla Gradient Boostingu:

L leave one out
J James-Stein
M m-estimate
P polynomial
H Helmert
D backward difference

accuracy	L	J	M
P	0.74	0.745	0.72
H	0.755	0.75	0.75
D	0.765	0.75	0.77
L	0.75	0.715	0.725
J	0.745	0.74	0.74

F1	L	J	M
P	0.821918	0.825939	0.812081
H	0.832765	0.829932	0.832215
D	0.840678	0.828767	0.842466
L	0.829932	0.804124	0.813559
J	0.824742	0.82069	0.824324

biznesowa	L	J	M
P	0.026208	0.0106196	-0.158493
H	0.129899	-0.0433056	-0.0763733
D	0.0774394	0.106196	0.185287
L	0.072924	-0.0781775	-0.169518
J	0.0105949	0.0400962	-0.105141

SVC

Zgodnie z tym, co napisaliśmy na początku, dodajemy jeszcze jedno kodowanie.

L leave one out
J James-Stein
M m-estimate
O ordinal
P polynomial
H Helmert
D backward difference

accuracy	L	J	M	O
P	0.71	0.71	0.71	0.71
H	0.71	0.71	0.71	0.71
D	0.71	0.71	0.71	0.71
L	0.71	0.71	0.71	0.71
J	0.71	0.71	0.71	0.71

F1	L	J	M	O
P	0.826347	0.826347	0.826347	0.826347
H	0.826347	0.826347	0.826347	0.826347
D	0.826347	0.826347	0.826347	0.826347
L	0.826347	0.826347	0.826347	0.826347
J	0.826347	0.826347	0.826347	0.826347

biznesowa	L	J	M	O
P	-0.400685	-0.400685	-0.400685	-0.400685
H	-0.400685	-0.400685	-0.400685	-0.400685
D	-0.400685	-0.400685	-0.400685	-0.400685
L	-0.400685	-0.400685	-0.400685	-0.400685
J	-0.400685	-0.400685	-0.400685	-0.400685

4.3.2. Wnioski

Według wyników accuracy i F1, wszystkie algorytmy wypadają dobrze. Jednak najważniejszym dla potencjalnego kredytobiorcy aspektem jest liczba pieniędzy, którą należałoby mu wydać. W ten sposób zawężamy nasz wybór do dwóch modeli, jako jedynych możliwych do rozwinięcia:

— Gradient Boosting z kodowaniem **ordinal** dla zmiennych numerycznych i **James-Stein** dla kategorycznych. (Wszystkie kodowania z tej drugiej kategorii wypadły tak naprawdę równie dobrze, wobec czego nasz wybór jest tu arbitralny.)
— Lasy losowe (Random Forest) z kodowaniem **backward difference** dla z. numerycznych i **m-estimate** dla kategorycznych.

Pozostałe modele wykazują bardzo niski przychód, albo i nawet straty.

4.3.3. Strojenie parametrów

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer
# wynik metryki biznesowej
def bussScore(y, y_pred, X):
    amounts = X['credit_amount']
    balance_all = y.apply(lambda x:
        ir_loan if x==1 else -lgd) * amounts
    earnings_made = balance_all
        .iloc[np.logical_and(y==1, y_pred==1).array].sum()
    losses_made = balance_all
        .iloc[np.logical_and(y==0, y_pred==1).array].sum()

    final_balance = earnings_made + losses_made
    max_income = balance_all.iloc[(y==1).array].sum()
    perc_of_max_income = final_balance/max_income
    return perc_of_max_income
# nasz wlasny pomiar dla strojenia parametrow
b_scorer = make_scorer(bussScore, greater_is_better=True)
# ustawiamy kodowania
gboost_enc = encoding_list_gen('j', 'p')
rforest_enc = encoding_list_gen('m', 'b')
```

Gradient boosting

```
parameters = {
    "loss":["deviance"],
    "learning_rate": [0.1, 0.15, 0.2],
    "min_samples_split": [0.01, 0.03, 0.05],
    "min_samples_leaf": [0.01, 0.02, 0.03],
    "max_depth":[3,5,8],
    "max_features":["log2","sqrt"],
    "criterion": ["friedman_mse", "mae"],
    "subsample":[0.8, 0.85, 0.9, 1.0],
    "n_estimators":[10, 100]
    }
df_train, df_test = multiEnc(X_train, X_test,
    y_train, columns_enc, gboost_enc)
grid = GridSearchCV(estimator=models[0],
    param_grid = parameters, scoring = 'f1', cv=4, n_jobs=-1)
# kroswalidacja
grid_result = grid.fit(df_train, y_train)
print("Best: %f using %s" % (grid_result.best_score_,
    grid_result.best_params_))
> Best: 0.868532 using {'criterion': 'friedman_mse',
> 'learning_rate': 0.15, 'loss': 'deviance',
> 'max_depth': 8, 'max_features': 'sqrt',
> 'min_samples_leaf': 0.02, 'min_samples_split': 0.05,
> 'n_estimators': 10, 'subsample': 0.9}
```

Wypisaliśmy najlepsze parametry dla metody Grid Search.

```
best_model_gboost = grid_result.best_estimator_
best_model_gboost.score(df_test, y_test)
bussScore(y_test, best_model_gboost.predict(df_test), df_test)
```

Wynik najlepszego estymatora wyniósł 0.755, a funkcja bussScore dała wartość około -0.17847.

Gradient boosting

```
parameters = {
    "min_samples_split": [0.01, 0.03, 0.05],
    "min_samples_leaf": [0.01, 0.02, 0.03],
    "max_depth":[3,5,8],
    "max_features":["log2","sqrt"],
    "criterion": ["gini", "mae"],
    "n_estimators":[10, 100, 150],
    "ccp_alpha":[0.0, 0.01, 0.1]
    }
df_train, df_test = multiEnc(X_train, X_test, y_train,
    columns_enc, rforest_enc)
grid = GridSearchCV(estimator=models[2], param_grid = parameters,
    scoring = 'f1', cv=4, n_jobs=-1)
grid_result = grid.fit(df_train, y_train)
print("Best: %f using %s" % (grid_result.best_score_,
    grid_result.best_params_))
```

```
> Best: 0.854381 using {'ccp_alpha': 0.0,
> 'criterion': 'gini', 'max_depth': 8,
> 'max_features': 'sqrt', 'min_samples_leaf': 0.01,
> 'min_samples_split': 0.01, 'n_estimators': 150}

best_model_gboost = grid_result.best_estimator_
best_model_gboost.score(df_test, y_test)
bussScore(y_test, best_model_gboost.predict(df_test), df_test)
```

Wynik najlepszego estymatora wyniósł 0.75, a funkcja bussScore dała wartość około -0.18053.

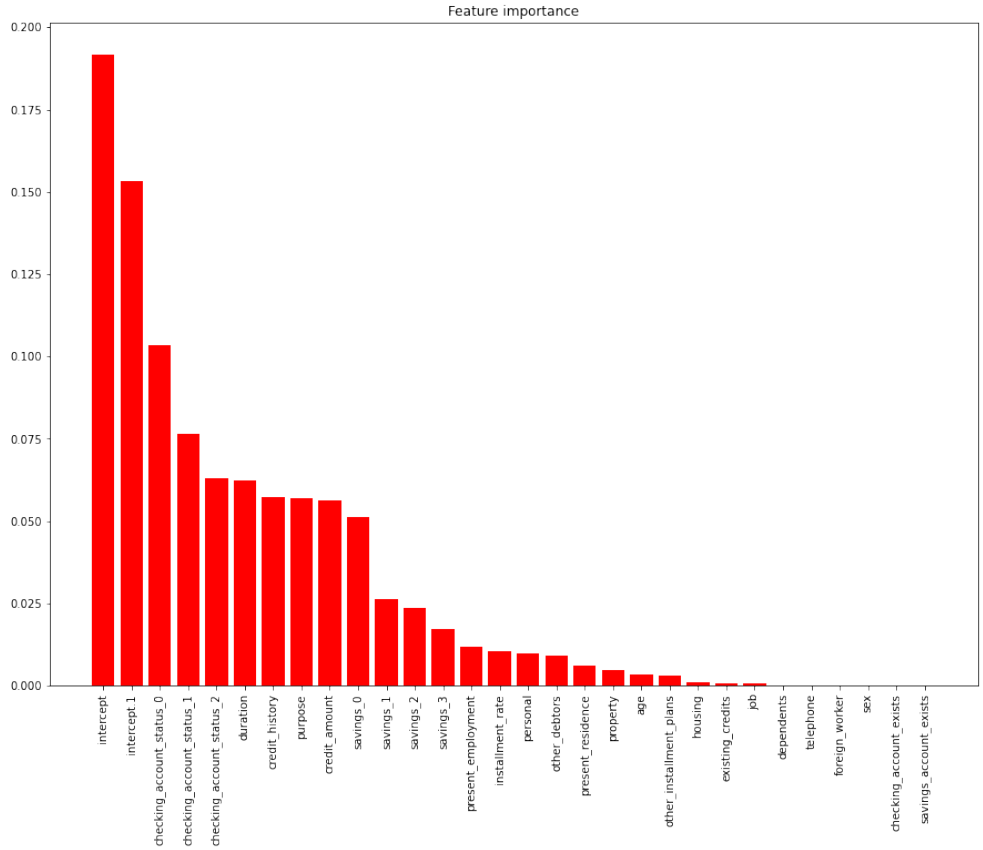
4.3.4. Feature importance

Na koniec sprawdzamy, które zmienne miały największy wpływ na wydajność naszych dwóch modeli. Ponownie tworzymy generyczną funkcję do nakreślenia wykresów:

```
def feature_imp(mod, df, y):
    importances = mod.feature_importances_
    indices = np.argsort(importances)[::-1]
    plt.figure(figsize=(15, 11))
    plt.title("Feature importance")
    plt.bar(df.columns, importances[indices], color="r",
    align="center")
    plt.xticks(rotation=90)
    plt.show()
```

(Oryginalna zawartość funkcji zakładała obsługę większej ilości modeli, gdyż dla różnych modeli wydobycie informacji przebiega trochę inaczej. Praktyczne działanie dla naszych dwóch klasyfikatorów ogranicza się jednak do takiej samej postaci wyżej.)

Gradient Boosting



Lasy losowe