Autor : Adam Smulczyk
email: adam.smulczyk@gmail.com
Profile: https://github.com/AdamSmulczyk
Github
Repository: https://github.com/AdamSmulczyk/ML_Project_001

1 Cel i zakres projektu

1.1 Wstęp

Zatonięcie Titanica to jedna z najbardziej niesławnych katastrof statków w historii. 15 kwietnia 1912 roku, podczas swojego dziewiczego rejsu, powszechnie uważany za „niezatapialny" RMS Titanic zatonął po zderzeniu z górą lodową. Niestety, nie było wystarczającej liczby łodzi ratunkowych dla wszystkich na pokładzie, co spowodowało śmierć 1502 z 2224 pasażerów i członków załogi. Chociaż w przeżyciu był pewien element szczęścia, wydaje się, że niektóre grupy ludzi miały większe szanse na przeżycie niż inne. W tym wyzwaniu prosimy Cię o zbudowanie modelu predykcyjnego, który odpowie na pytanie: „jaki rodzaj ludzi miał większe szanse na przeżycie?" przy użyciu danych pasażerów (tj. imię, wiek, płeć, klasa społeczno-ekonomiczna itp.).

1.2 Cel projektu

Projekt ten będzie realizowany w oparciu o rzeczywisty zbiór danych, dostarczony przez organizatora projektu. Celem jest wykorzystanie tego zbioru danych oraz uczenia maszynowego do stworzenia modelu predykcyjnego szacującego szanse danego pasażera na ocalenie.

1.3 Opis zbioru danych

Dostarczone zostały dwa zestawy danych, zestaw treningowy i testowy.
Zakłada się, że zmienna PassengerID jest losowym, unikalnym identyfikatorem, który nie ma wpływu na zmienną wynikową. Dlatego zostanie wykluczona z analizy. Wszystkie pozostałe zmienne są potencjalnymi predyktorami lub zmiennymi niezależnymi.

Zestaw treningowy składa się zatem z 10 atrybutów predykcyjnych, w których skład wchodzą:

1.Survival: zmienna Survived jest naszym wynikiem lub zmienną zależną. Jest to binarny nominalny typ danych, gdzie 1- przeżył/ocalał i 0 - nie przeżył/nie ocalał

2.Pclass: klasa biletu pasażera. Zmienna Pclass jest porządkowym typem danych dla klasy Ticket, zastępcą statusu ekonomiczno-społeczny (SES), reprezentującym 1 – pierwsza (klasa wyższa), 2 - druga (klasa średnia) i 3 - trzecia (klasa niższa).

3.Name – imię i nazwisko pasażera. Zmienna Name jest nominalnym typem danych. Można jej użyć w inżynierii cech do wyprowadzenia płci z tytułu, wielkości rodziny z nazwiska i SES z tytułów, takich jak doktor lub lord. Ponieważ te zmienne już istnieją, wykorzystamy je, aby sprawdzić, czy tytuł, taki jak doktor, czy lord, robi różnicę

4.Sex – płeć pasażera. Zmienne Sex jest nominalnym/ kategorycznym typem danych. Do celów obliczeniowych modelu zostanie one przekształcona na zmienne numeryczne.

5.Sibsp – reprezentuje liczbę spokrewnionych rodzeństwa/małżonków na pokładzie

6.Parch – reprezentuje liczbę spokrewnionych rodziców/dzieci na pokładzie. Może być ona używana do inżynierii cech w celu utworzenia rozmiaru rodziny.

7.Ticket – numer biletu. Zmienna na pierwszy rzut losowa, unikalny identyfikator, który nie ma wpływu na zmienną wynikową. Zmienna Pclass jest porządkowym typem danych dla tego atrybutu.

8.Fare – opłata za bilet.

9.Cabin – kabina, zmienna Cabin to nominalny typ danych, który może być używany w inżynierii cech do przybliżonej pozycji na statku w momencie wystąpienia incydentu i SES z poziomów pokładu. Jednak ponieważ istnieje wiele wartości zerowych, nie dodaje ona wartości i w związku z tym prawdopodobnie będzie wykluczona z analizy.

10.Embarked – port startowy (C = Cherbourg; Q = Queenstown; S = Southampton).Zmienne Embarked jest nominalnym/ kategorycznym typem danych. Do celów obliczeniowych modelu zostanie one

przekształcona na zmienne numeryczne.

Wybrane cechy pozwolą nam na wyznaczenie wartości zmiennej wynikowej. Wartością, którą będziemy badać/ przewidywać to atrybut „Survival" (0/1).

1.4 Założenia projektu

Cały projekt wykonany został w Python'ie, korzystając z Jupyter'a.

1.4.1 Zakres projektu

Określanie zakresu projektu, jego celu i priorytetów determinuje rodzaj podejścia do niego.
W tym przypadku głównym celem jest osiągnięcie wyniku modelu predykcyjnego, który przekroczy wartość satysfakcjonującą osiągniętą przez organizatora, a mianowicie Accuracy = 0,77990.

1.4.2 Wskaźniki sukcesu

Wskaźnikami sukcesu określonych w wymaganiach są w tym przypadku kolejno: Accuracy, F1 score.
Accuracy- metryka ta mówi nam o stosunku dobrze dokonanych klasyfikacji do ilości wszystkich dokonanych klasyfikacji. Wyraża w procentach i każdy ją od razu zrozumie. Ale ma też wady i jedną z nich jest sytuacja, w której mamy niezrównoważony zbiór. Accuracy może być więc bardzo zwodnicze.
Dlatego, dodatkowym wskaźnikiem sukcesu jest metryka F1, która jest średnią harmoniczną z dwóch poniższych wartości:

F1 is calculated as follows:

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

where:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

Precision - metryka ta mówi nam mniej więcej o tym, w jakim stopniu możemy zaufać naszym pozytywnym predykcjom w danej klasie.
TP - true positive to obserwacja dobrze przypisana do analizowanej klasy
FP - false positive to obserwacja błędnie przypisana.
Recall - koncepcja stojąca za recall jest podobna do precision, dowiadujemy się z niej jednak ile obserwacji zgubiliśmy dla danej klasy. Stosujemy tutaj następujące równanie:

Tak więc możemy mieć sytuację, gdzie accuracy wyszło niby całkiem dobre, ale precision i recall już nie za bardzo. Dlatego przydatne jest również rozpatrzenie metryki F1, za pomocą której uzyskujemy jedną liczbę (między 1 a 0, gdzie 1 jest sytuacją najlepszą), która równomiernie uwzględnia precision i recall.

1.4.3 Studium wykonalności projektu

Musimy dokonać oceny realnej możliwości wykonania planu projektu, aby zdecydować, czy jesteś w stanie z tak małą próbką danych dokonać budowy wiarygodnego modelu predykcyjnego. Ewentualnie czy możemy uzyskać dopływ nowych danych?
W tym celu zostanie zastosowana inżynieria cech (feature engineering), aby otrzymać pełniejszy zestaw atrybutów predykcyjnych. Dodatkowym aspektem, który wpłynie na ocenę wykonalności projektu jest dostarczony zestaw danych testowych, za pomocą którego będziemy weryfikować jakość otrzymanego modelu predykcyjnego.

2 Wczytanie i przygotowanie danych do analizy

2.1Wczytanie danych i niezbędnych bibliotek

Wczytanie niezbędnych bibliotek użytych w projekcie.

Import libraries

```
import pandas as pd
import numpy as np
import scipy.stats as stats
import seaborn as sns
import matplotlib.pyplot as plt
import warnings

from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.gaussian_process.kernels import RBF
from xgboost import XGBRegressor, XGBClassifier

from sklearn.metrics import mean_absolute_error, accuracy_score, f1_score
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV,StratifiedKFold,train_test_split
```

Wczytanie danych.

Import data files

```
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
```

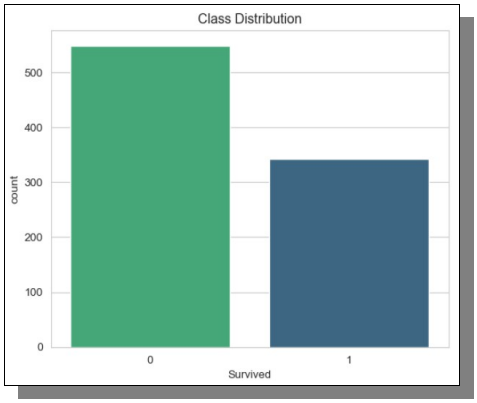
Dostarczone dane są podzielone na dwa zestawy train i test. Zestaw testowy nie zawiera atrybutu „Survived", służy jedynie do przetestowania poprawności zbudowanego modelu na danych train.

2.2 Wstępna eksploracyjna analiza danych (EDA)

2.2.1 Rozkład zmiennej zależnej 'Survival' (ang. Class Distribution)

Oczywiste jest, że niewielu pasażerów przeżyło wypadek. Spośród 891 pasażerów w zestawie szkoleniowym, tylko około 350 przeżyło, tj. Tylko 38,4% całego zestawu szkoleniowego przeżyło katastrofę. Należy dokładniej przeanalizować dane, aby uzyskać lepszy wgląd w nie i zobaczyć, które kategorie pasażerów przeżyły, a które nie. Należy sprawdzić wskaźnik przeżycia, korzystając z różnych cech zestawu danych.

Niezrównoważony zbiór danych to zbiór danych, który ma niezrównoważony rozkład przykładów różnych klas. Próba wytrenowania model klasyfikacji na podstawie takich danych, spowoduje że model będzie stronniczy w kierunku klasy większościowej. Jeżeli taki problem zaistnieje, można go rozwiązać na dwa sposoby:
-działanie na danych – czyli wpływanie w jakiś sposób na dane, które wybieramy do zbioru danych
-działanie na koszcie funkcji – czyli modyfikowanie wag klas jak wpływają na model.



2.2.2 Statystyka zmiennych

Statistical Summary moduł ten został dodany by przyjrzeć się nieco bliżej danym. Pozwala on podejrzeć podstawowe metryki dla każdej kolumny, np. wartości minimalne, wartości maksymalne, średnią, medianę, itd.

Atrybuty numeryczne (numerical): PassengerId, Survived, Pclass, Age, SibSp, Parch, Fare.

Statistical Summary

```
def description_numerical(df: pd.DataFrame):
    df = df.select_dtypes(include='number')
    desc = pd.DataFrame(index=df.columns.to_list())
    desc['type'] = df.dtypes
    desc['count'] = df.count()
    desc['nunique'] = df.nunique()
    desc['%unique'] = desc['nunique'] / len(df) * 100
    desc['null'] = df.isnull().sum()
    desc['%null'] = desc['null'] / len(df) * 100
    desc = pd.concat([desc, df.describe().T.drop('count', axis=1)], axis=1)
    desc["skew"] = df.skew()
    desc["kurtosis"] = df.kurtosis()
    desc = desc.round(2)
    return desc

description_numerical(train)
```

	type	count	nunique	%unique	null	%null	mean	std	min	25%	50%	75%	max	skew	kurtosis
PassengerId	int64	891	891	100.000	0	0.000	446.000	257.350	1.000	223.500	446.000	668.500	891.000	0.000	-1.200
Survived	int64	891	2	0.220	0	0.000	0.380	0.490	0.000	0.000	0.000	1.000	1.000	0.480	-1.780
Pclass	int64	891	3	0.340	0	0.000	2.310	0.840	1.000	2.000	3.000	3.000	3.000	-0.630	-1.280
Age	float64	714	88	9.880	177	19.870	29.700	14.530	0.420	20.120	28.000	38.000	80.000	0.390	0.180
SibSp	int64	891	7	0.790	0	0.000	0.520	1.100	0.000	0.000	0.000	1.000	8.000	3.700	17.880
Parch	int64	891	7	0.790	0	0.000	0.380	0.810	0.000	0.000	0.000	0.000	6.000	2.750	9.780
Fare	float64	891	248	27.830	0	0.000	32.200	49.690	0.000	7.910	14.450	31.000	512.330	4.790	33.400

W przypadku kilku zmiennych jasno widać, że średnia jest znacznie większa lub znacznie mniejsza niż mediana. Jest to objaw skośności rozkładu. Ponieważ niektóre z algorytmów cechują się wrażliwością na dane odstające (np. Regresja Logistyczna). W celu poprawnej predykcji należałoby usunąć wartości odstających obserwacji. Niestety dostarczony zbiór danych jest mały, a taki zabieg dodatkowo uszczupliłoby zbiór. Zatem w celu zachowania danych wartości odstające zostaną zmodyfikowane poprzez funkcje opartą na regule 1.5 wartości rozstępu międzykwartylowego. Dodatkowym krokiem będzie normalizacja danych numerycznych.

Już na tym etapie możemy stwierdzić, że algorytm drzewa decyzyjnego zdobywa pewną przewagę nad swoimi konkurentami. Mianowicie nie jest on wrażliwy na występowanie wartości odstających. W przypadku zmiennych numerycznych podział węzłów w drzewie decyzyjnym jest wykonywany na podstawie proporcji obserwacji w zbiorze, a nie na podstawie zakresu wartości.

Atrybuty kategoryczne (categorical): Sex, Ticket, Cabin, Embarked.

```
def description_categorical(df: pd.DataFrame):
    df = df.select_dtypes(include='object')
    desc = pd.DataFrame(index=df.columns.to_list())
    desc['type'] = df.dtypes
    desc['count'] = df.count()
    desc['nunique'] = df.nunique()
    desc['%unique'] = desc['nunique'] / len(df) * 100
    desc['null'] = df.isnull().sum()
    desc['%null'] = desc['null'] / len(df) * 100
    return desc

description_categorical(train)
```

	type	count	nunique	%unique	null	%null
Name	object	891	891	100.000	0	0.000
Sex	object	891	2	0.224	0	0.000
Ticket	object	891	681	76.431	0	0.000
Cabin	object	204	147	16.498	687	77.104
Embarked	object	889	3	0.337	2	0.224

Wstępne wnioski wyglądają następująco:

-3 kolumny mają brakujące wartości.

-Najwięcej braków zawiera atrybut 'Cabin'- 687/891, co stanowi ok. 77% wszystkich wartości tej kolumny.

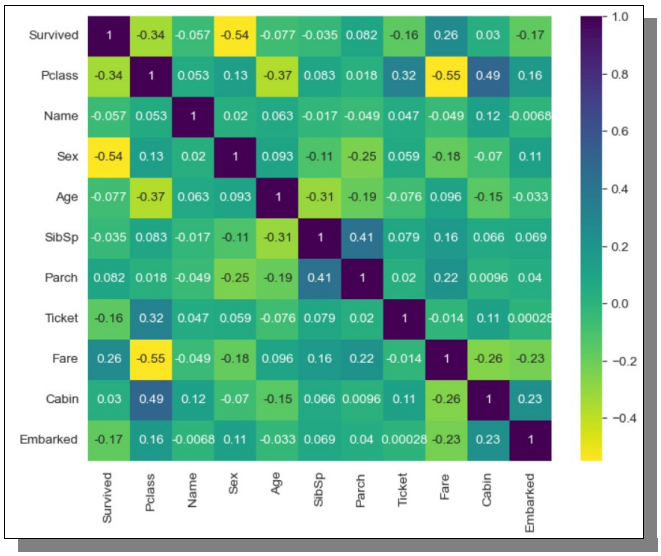
-'Age' - 177 brakujących wartości, co stanowi ok. 20% wszystkich wartości tej kolumny.

Ogólna liczba brakujących wartości występuje tylko w trzech kolumnach. W tym atrybut 'Cabin' ma bardzo duży odsetek wartości brakujących co stanowi o jego małej przydatności. Najlepszym rozwiązaniem byłoby jego pominięcie, aczkolwiek należy to jeszcze rozważyć uwzględniając małą liczbę

atrybutów. Natomiast cecha „Age” zostanie uzupełniona wygenerowanymi danymi poprze algorytm KNN. Należy tutaj zaznaczyć, że algorytmy drzewa decyzyjnego nie są wrażliwe na brakujące wartości.

2.2.3 Weryfikacja statystyk dotyczących połączonego zbioru zmiennych.

Zbadana zostanie tutaj zależność ze zmienną celu poprzez zbudowanie tablicy korelacji (pairplot: badana zmienna vs zmienna celu). Pairplot da ogólne spojrzenie na moc predykcyjną danego atrybutu.



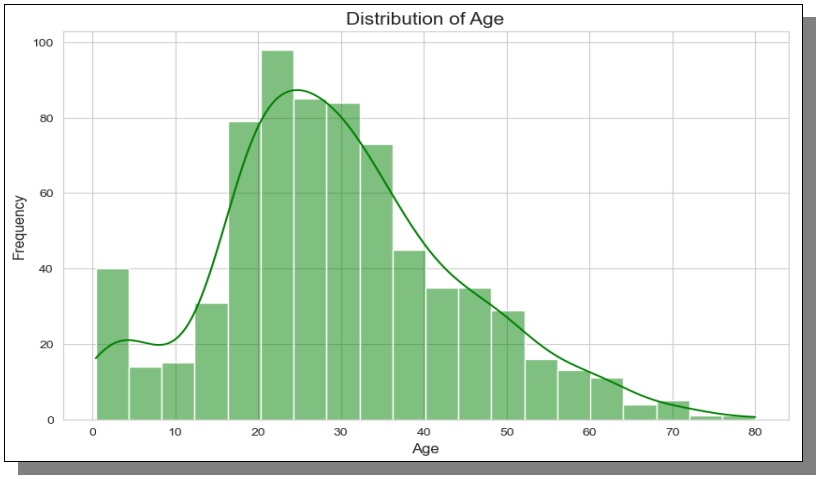
Wstępne wnioski wyglądają następująco:
-'Sex'- posiada zdecydowanie największą korelację.
-'Cabin’ i „Fare” -też wydają się obiecujące.
-'Pclass' -mimo ujemnej korelacji wybija się, ze jest ona zdecydowanie największa.
-'Age’- zaskakująco mała korelacja, możliwe, że zmieni się to gdy podzielimy te dane na szereg podgrup.

2.2.4 Analiza zależności pomiędzy zmiennymi

Następnie zostanie przeprowadzona analiza jednowymiarowa we wszystkich kolumnach, aby zrozumieć sytuację i skupić się na atrybucie 'Survival'.

2.2.4.1 Analiza rozkładu zmiennych liczbowych

Histogram to klasyczne narzędzie wizualizacyjne, które przedstawia rozkład jednej lub większej liczby zmiennych poprzez zliczanie obserwacji mieszczących się w dyskretnych przedziałach. Na poniższym wykresie jest przedstawiony histogram dla zmiennej 'Age'. Reszta jest dostępna w skoroszycie projektu.



Na podstawie zamieszczonych wykresów możemy wywnioskować:
- Większość osób jest w klasie 3, ale liczba osób w klasie 1 i 2 była prawie taka sama .
- Najwięcej osób wsiada na pokład w Southampton w Anglii, a następnie w Cherbourg we Francji

i Queenstown w Irlandii.
- Wszystkie atrybuty numeryczne posiadają wartości odstające.
- 'Age' ma rozkład normalny, natomiast 'Fare' rozkład prawostronnie skośny.

2.2.4.2 Diagram zależności 'Survived' - 'Pclass' (klasa biletu pasażera).

Zmienna 'Pclass' jest tutaj sowjego rodzaju wyznacznikiem statusu ekonomiczno-społeczny (SES), reprezentującym 1 – pierwsza (klasa wyższa), 2 - druga (klasa średnia) i 3 - trzecia (klasa niższa).

Survived	0	1	All
Pclass			
1	80	136	216
2	97	87	184
3	372	119	491
All	549	342	891

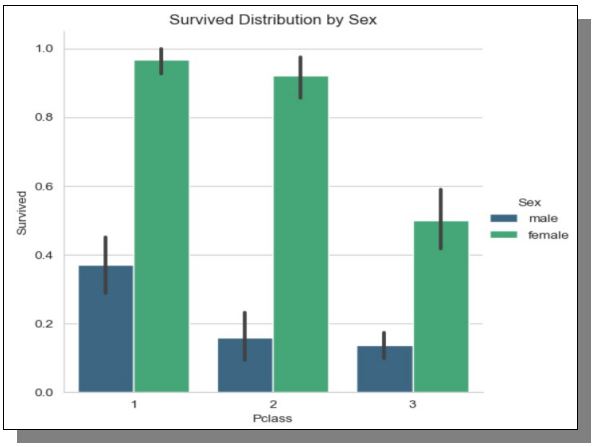
Wnioski:
-Największa liczba pasażerów znajduje się w 3-klasie (~55% wszystkich pasażerów)
-Większość pasażerów 3-klasy nie przeżyła.
-Pasażerowie 1-klasy mieli najwyższy odsetek ocalałych.

2.2.4.3 Diagram zależności 'Survived' – 'Pclass' – 'Sex'.

	Pclass	1	2	3	All
Sex	Survived				
female	0	3	6	72	81
	1	91	70	72	233
male	0	77	91	300	468
	1	45	17	47	109
All		216	184	491	891

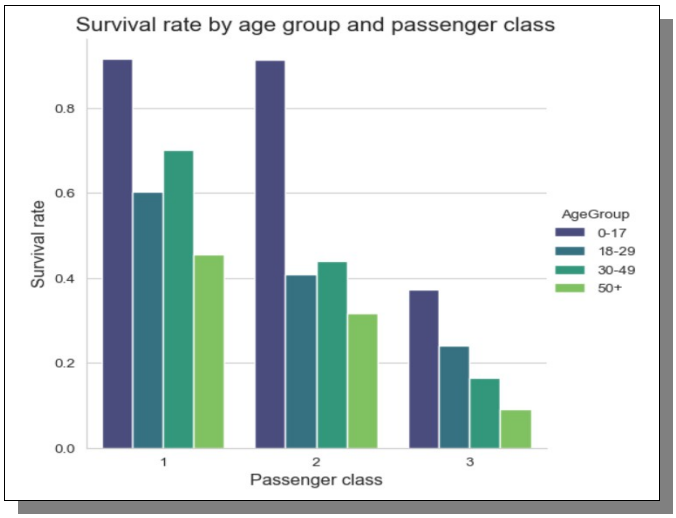
Wnioski:
-Największe szanse na przeżycie miały kobiety w 1-klasie (97%) i 2-klasie (79%)
-Kobiety w 3-klasie miały nawet większe szanse ocalenia niż mężczyźni w 1-klasie (50%)
-Jeżeli chodzi o mężczyzn to największe szanse ocalenia mieli w 1-klasie (38%)
-Płeć jeszcze w większym stopniu niż klasa biletu miała wpływ na przeżycie. Można stwierdzić, że płeć była zdecydowanie determinantą przeżycia.

2.4.4.4 Wykres zależności 'Survived' – 'Sex'.



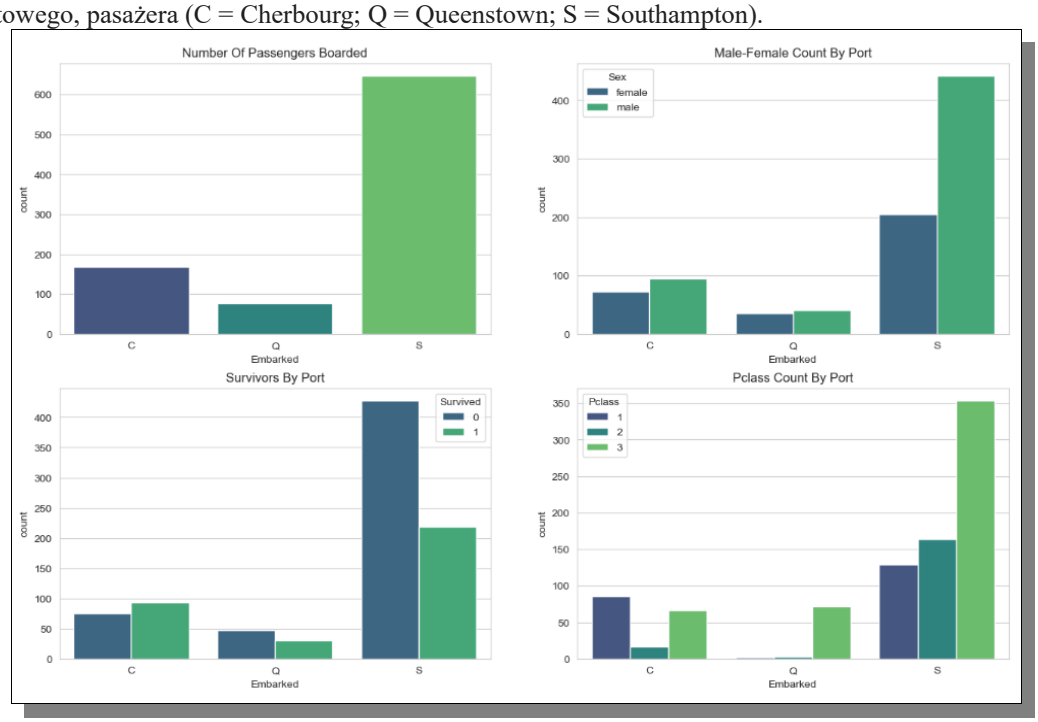
Wykres ten potwierdza tezę zależności 'Survived' – 'Sex' z 2.2.4.3

2.2.4.5 Analiza szans przeżycia w zależności od grupy wiekowej.



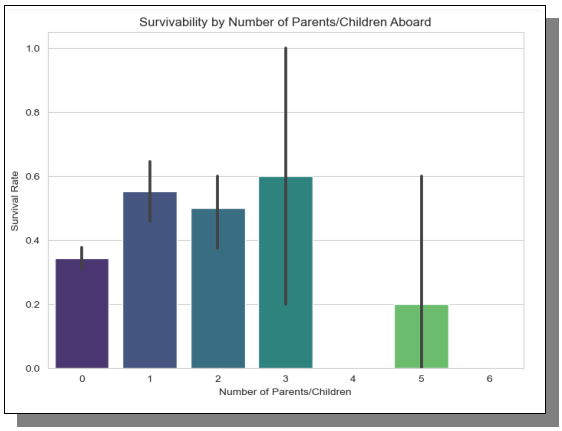
Wnioski:
-Największe szanse na przeżycie miała grupa 1 (0-17) w każdej klasie.
-Najmniejsze szanse na przeżycie miała grupa 4 (50+) w każdej klasie.
-Wiek pasażera był zdecydowanie jednym z najważniejszych czynników wpływających na szanse przeżycia.

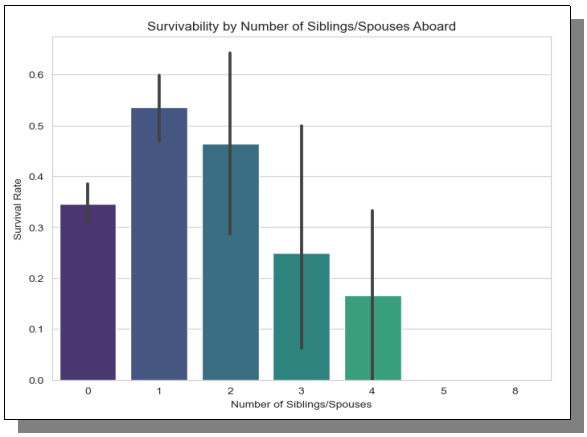
2.2.4.6 Analiza zmiennej 'Embarked'
Zestaw wykresów kolumnowych mających na celu charakterystykę zmiennej 'Embarked', czyli portu startowego, pasażera (C = Cherbourg; Q = Queenstown; S = Southampton).

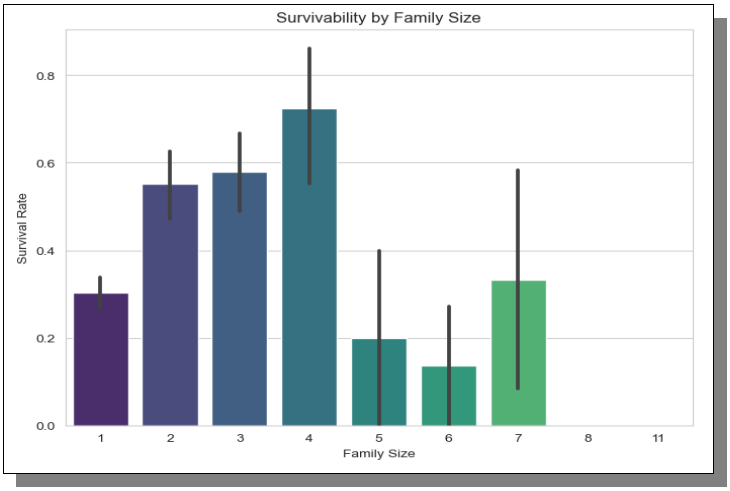


Wnioski:
-Większość pasażerów zaokrętowanych w Cherbourg to pasażerowie 1-szej klasy.
-Większość pasażerów zaokrętowanych w Cherbourg przeżyła
-W Southampton zostało zaokrętowanych zdecydowanie najwięcej mężczyzn.
-Pasażerowie zaokrętowani w Southampton mają najniższy procent ocalałych.
-Queenstown pomimo przytłaczającej liczby pasażerów z 3-klasy ma całkiem wysoki procent ocalałych, co wynika zapewne z faktu dużej liczebności kobiet, a co za tym idzie też dzieci.

2.2.4.7 Analiza szans przeżycia w zależności od wielkości rodziny.

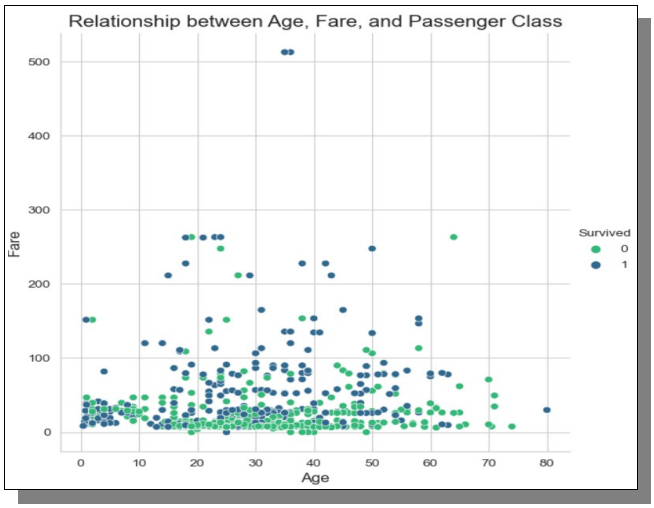






Wnioski:
-Statystycznie największe szanse ocalenia miały osoby z rodzin 2, 3 i nawet czteroosobowych, na co zapewne miał wpływ, że dzieci i kobiety miały pierwszeństwo od szalup ratunkowych,
- Niestety dla rodzin 5-osobowych i większych te szanse diametralnie malały.
-Pasażerowie będący w pojedynkę na rejsie miały statystycznie średnie szanse na przeżycie, wynik zapewne ten obniża duża ilość pojedynczych mężczyzn.

2.2.4.8 Analiza szans przeżycia w zależności 'Age' – 'Fare' (klasa biletu).



Wnioski:
-Pasażerowie z najtańszymi biletami zdecydowanie mieli najmniejsze szanse przeżycia.
-Dla dzieci do lat 8 klasa biletu nie była znacząca, większość przeżyła.

2.2.4.9 Podsumowanie

Na podstawie przeprowadzonej analizy należy stwierdzić, że rozdzielanie miejsc w szalupach ratunkowych odbywało się zgodnie z dogmatem „ kobiety i dzieci na pierwszym miejscu". Koncepcja ta była zwyczajowo uznana w prawie morskim. Drugim czynnikiem decydującym o miejscu w szalupie była klasa zakupionego biletu. Pasażerowie 1-szej klasy choć najmniej licznej byli traktowani wyjątkowo na tle dwóch pozostałych klas. Kobiety z 1-szej klasy zginęły tylko 3, co stanowi o 97% szansie na ocalenie. Jeżeli chodzi o mężczyzn to największe szanse ocalenia też mieli w 1-klasie (38%) . Tak jak to zostało zaznaczone wcześniej dzieci też miały wysoki priorytet. Tutaj należy szczególnie uwzględnić te najmłodsze czyli do 6 roku życia. Jeżeli analizujemy wiek to najmniejsze szanse na przeżycie miała grupa (50+) w każdej klasie. Dodatkowym aspektem jest, że statystycznie największe szanse ocalenia miały osoby złożone z rodzin 2, 3 i nawet czteroosobowych, na co zapewne miał wpływ, że dzieci i kobiety miały pierwszeństwo od szalup ratunkowych. Jeżeli chodzi o miejsce zaokrętowania to większość pasażerów zaokrętowanych w Cherbourg przeżyła. Byli to w większości pasażerowie 1-szej klasy.

Smutną konstatacją jest fakt, że największą liczbę pasażerów stanowiła 3-klasa (~55% wszystkich pasażerów), większość z nich nie przeżyła. W przypadku 1-szej klasy % osób, które przeżyły, wynosi około 63%., natomiast pasażerowie z najtańszymi biletami zdecydowanie mieli nikłe szanse przeżycia.

3 Przygotowanie danych do dalszej analizy (ang. Data cleaning)

3.1 Sprawdzenie brakujących rekordów

Ze względu na ograniczoną liczbę danych do analizy (liczba wierszy to tylko 891) nie jest zalecane usuwanie tych z pustymi jak i odstającymi wartościami. Zamiast tego wykorzystane zostaną odpowiednie algorytmy do ich uzupełnienia lub transformacji w przypadku wartości odstających.
Istnieje kilkaset brakujących wartości dla atrybutów: 'Age', 'Cabin', 'Embarked'.

'Embarked', czyli port, z którego odpływała dana osoba, ma tylko 2 brakujące wartości więc wygenerowane nowe nie będą miały żadnego wpływu na potencjalny błąd modelu.
'Cabin' brakuje tu niemal 77% wartości, więc atrybut ten można z dużą dozą prawdopodobieństwa pominąć ponieważ nie ma znaczącego wpływu na wynik. Jeżeli chodzi o 'Cabin' to pewnego rodzaju substytutem jest tu klasa, w której podróżował pasażer.
'Age' - dla wszystkich pasażerów, z brakiem podanego wieku, dane zostaną uzupełnione o średnią wieku dla danej płci.

3.2 Usuwanie duplikatów i niepotrzebnych kolumn

Korygowanie poprzez usuwanie cech (ang. Correcting), to cel początkowy do wykonania. Usuwając niepotrzebne cechy mamy do czynienia z mniejszą liczbą punktów danych co przyspiesza obliczenia i ułatwia analizę.
Do procesu modelowania nie zostaną rozpatrzone następujące atrybuty:
'PassengerID' - jest losowym, unikalnym identyfikatorem, który nie ma wpływu na zmienną wynikową
'Ticket' – podobnie jak 'PassengerID' unikalnym identyfikatorem, który nie ma wpływu na zmienną wynikową.

3.3 Konwersja typów (ang. Downcasting)

W celu przyspieszenia obliczeń zostanie przeprowadzony downcasting danych, co pozytywnie wpłynie zmniejszenie . Dodatkowo dane zostaną podzielone na kategoryczne(ciągi znaków) i liczbowe.

```
# convert types (downcasting)
def convert_types (df):
    object_to_categorical = df.select_dtypes(include=['object'])
    numerical_int = df.select_dtypes(include=['int64'])
    numerical_float = df.select_dtypes(include=['float64'])

    for i in object_to_categorical:
         df[i] = df[i].astype('category')
    for i in numerical_int:
         df[i] = df[i].astype('int32')
    for i in numerical_float:
         df[i] = df[i].astype('float32')
    return df

train = convert_types(train)
test = convert_types(test)
```

Dodatkowo zostanie przeprowadzona konwersja kolumny 'Sex' zawierająca ciąg znaków na wartości liczbowe gdzie female=1 i male=0. Operacja taka jest wymagana przez większość algorytmów.

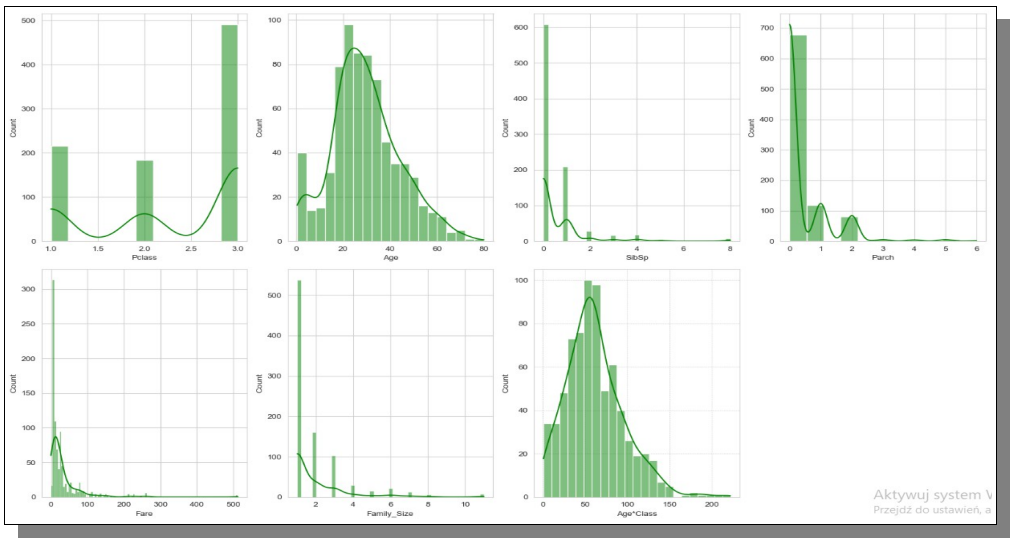
3.4 Inżynieria cech (ang. Feature engineering)

Inżynieria cech jest procesem tworzenia całkowicie nowych cech, lub na podstawie już istniejących, umożliwiających lepsze działanie algorytmów uczenia maszynowego. Poprawne przygotowanie charakterystyk bardzo zwiększa moc predykcyjną algorytmów uczenia maszynowego.
Ponieważ w zadanym projekcie liczba atrybótów jest stosunkowo mała, na podstawie wcześniejszej analizy zostały dodane:
'Title' - wyszczególniony z 'Name'.
'Family_Size' - współczynnik wielkości rodziny = 'Parch' + "SibSp' +1 (czyli pasażer)
'AgeGroup' - współczynnik grupy wiekowej, który został podzielony na 6 grup: '0-7', '8-14', '15-19', '20-29' , '30-49' i '50+'.
'Age*Class' - współczynnik klasa biletu – wiek = 'Age' * 'Pclass'

3.5 Sprawdzenie skośności roskładów (ang. Skewness of distributions)

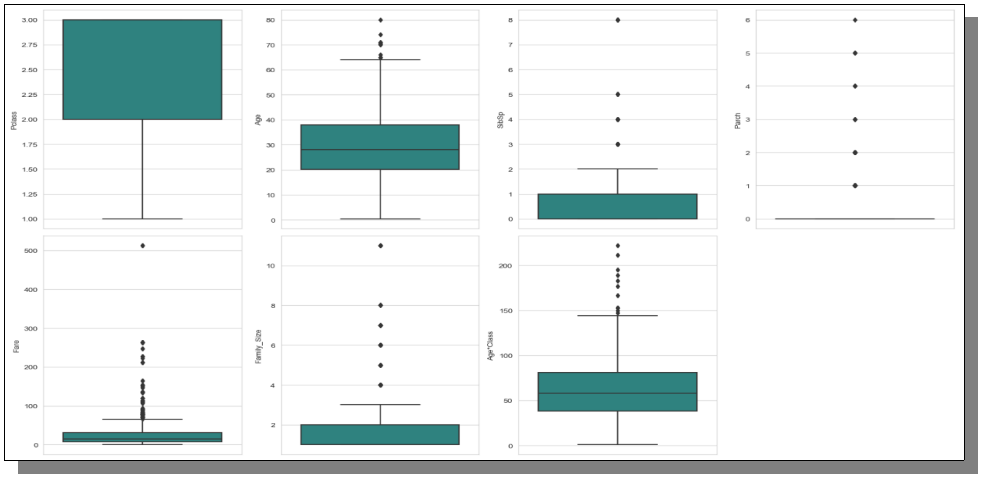
Analiza rozkładów danych numerycznych pod kątem ich skośności. 379 / 5 000 Czasami najlepiej jest całkowicie usunąć te rekordy ze swojego zestawu danych, aby zapobiec zniekształcaniu analizy.
Usuwamy wartości odstające, jeśli jest to spowodowane błędem wprowadzania danych, błędem

przetwarzania danych lub liczba obserwacji odstających jest bardzo mała. Możemy również usunąć jedynie wartości odstające. Jednak w naszym przypadku usuwanie obserwacji nie jest dobrym pomysłem, ze względu na mały zestaw danych.



3.6 Sprawdzenie wartości odstających (ang. Detect outliers)

Wartość odstająca to obserwacja posiadająca nietypową wartość zmiennej. Inaczej mówiąc, jest to punkt danych oddalony od innych obserwacji.



Obserwacje odstające mogą odzwierciedlać rzeczywisty rozkład lub być rezultatem przypadku. Na przykład mogą być błędnym pomiarem, być pomyłką przy wprowadzaniu danych lub wynikać ze zmienności pomiaru.
Należy uważać na wartości odstające, ponieważ jeśli ich nie wykryjemy i nie obsłużymy, to mogą znacząco wpłynąć na przewidywania i dokładność użytego modelu. Przykładem "wrażliwców" mogą być: regresja liniowa i sieci neuronowe.
Do naprawy wartości odstających zostanie zastosowana IQR Method. reguła 1.5 wartości rozstępu międzykwartylowego - metoda analityczna oparta na statystykach. Metoda ta jest używana dla dowolnej zmiennej numerycznej bez względu na rozkład. U jej podstaw leży założenie, że wszystkie "typowe" obserwacje leżą pomiędzy punktami wyznaczonymi przez odległość 1.5 IQR (ang. interquartile range):

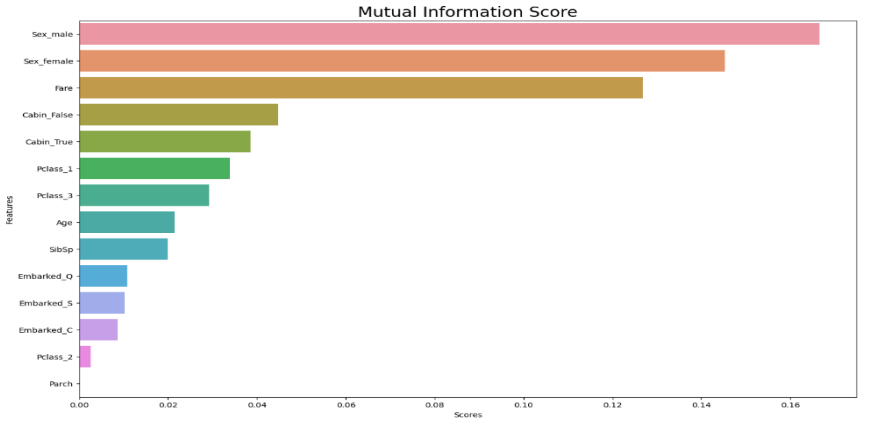
-"na lewo" od granicy pomiędzy pierwszym i drugim kwartylem,
-"na prawo" od granicy pomiędzy trzecim i czwartym kwartylem.

```python
def handle_outliers(df):
    numerical_columns = ['Age', 'SibSp', 'Fare']
    for column in numerical_columns:
        Q1 = df[column].quantile(0.25)
        Q3 = df[column].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        df[column] = df[column].clip(lower=lower_bound, upper=upper_bound)
    return df

X = handle_outliers(X)
test = handle_outliers(test)
```

3.7 Wybór odpowiednich kolumn

Za pomocą funkcji Mutual Information Score zostanie zbadana przydatność zmiennych. Jak widzimy 90% zmiennych jest przydatne, dlatego taki właśnie parametr przy wyborze ilości atrybutów zostanie zadany w docelowym modelu.

```python
plt.figure(figsize=(16, 10))
sns.barplot(x="Scores", y="Features", data=mi_scores)
plt.title("Mutual Information Score", size=20)
plt.show()
```



4 Przygotowanie danych do modelowania

W tej sekcji zostanie przeprowadzonych szerego operacji w bloku 'pipeline' do których zaliczmy:
-Aggregate numerical and categorical - pogrupowaniedanych na kategoryczne i liczbowe.
-Imputing missing values - wstawianie brakujących rekordów dla grupy kategorycznych atrybutów za pomocą funkcji – KNNImputer().
-Wstawianie brakujących rekordów dla grupy liczbowe za pomocą funkcji - SimpleImputer(strategy='median'), metodla mediany, czyli wybór wartości środkowej, wśród danych uporządkowanych od najmniejszej do największej.
-Removing outliers - naprawa wartości odstających za pomocą reguły 1.5xIQR wartości rozstępu międzykwartylowego - klasa Custom_winsorize().
-Normalizing - normalizacja danych – MinMaxScaler().

-Feature selection -wybór najbardziej znaczących atrybutów za pomocą Regresji Lasso(alpha=0.001).

```
# Aggregate numerical and categorical
num_cols = make_column_selector(dtype_include='number')
cat_cols = make_column_selector(dtype_exclude='number')

# Numeric Columns - imputing missing values
imp_median = SimpleImputer(strategy='median', add_indicator=True)
imp_knn = KNNImputer(n_neighbors=5, weights='uniform', metric='nan_euclidean')
minmaxscaler = MinMaxScaler()

# Categorical Columns - imputing missing values
imp_constant = SimpleImputer(strategy='constant', fill_value='missing')
ohe = OneHotEncoder(handle_unknown='ignore')
ori = OrdinalEncoder()

# power = PowerTransformer(method='box-cox', standardize=True) #only works with strictly positive values
power = PowerTransformer(method='yeo-johnson', standardize=True) #default-works with positive and negative values

preprocessor = make_column_transformer(
                                        (make_pipeline(imp_median,customwinsorize, minmaxscaler, power), num_cols),
                                        (make_pipeline(imp_constant, ohe), cat_cols)
                                       )
```

```
# Feature selection
selection = SelectFromModel(Lasso(alpha=0.001))
selection_1 = RFE(DecisionTreeClassifier(random_state=42), n_features_to_select=10)
selection_2 = SelectKBest(f_classif)
```

5.Budowa i ocena modelu

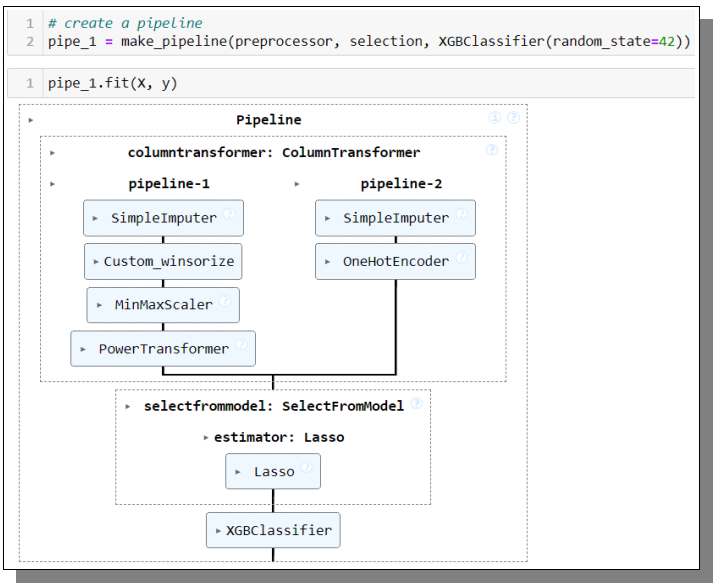
W tym punkcie zostaną przeanalizowane rodzaj problemu i wymagania dotyczące jego rozwiązania, aby zawęzić do kilku wybór najlepszego algorytmu. Usatanowione zadanie jest dwuklasowym problemem klasyfikacyjnym, w którym należy wyznaczyć związek między zmienną zależną 'Survived', a innymi zmiennymi/ cechami ('Sex', 'Age', 'Fare'...). Zadaniem klasyfikatora jest wyznaczenie jednej z wartości 0/1, tak/nie, prawda/fałsz. Rozważany będzie algorytmu należący do grupy algorytmów z dziedziny ucznienia nadzorowanego (ang. supervised learning), ze względu na fakt, że uczenie maszynowe odbywać sie będzie przy użyciu określonego zestawu danych oznaczonyh. Po analizie kryteriów do budowy modelu zostały przewidziane trzy algorytmy. Dwa oparte na wzmocnieniu gradientowym (ang. Gradient Boosting) XGBClassifier i LGBMClassifier oraz jeden oparty o algorytm drzewa decyzyjnego, czyli las losowy - RandomForestClassifier

Plan modelowania:
a).Budowa „czystego" model, który będzie benchmarkiem. Zawierać on będzie wszystkie zmienne i domyślne parametry algorytmu.
b).Optymalizacja parametrów modelu dla wszystkich zmiennych z użyciem RandomSearchCV/ Optuna.
c).Wybór najlepszego zestawu zmiennych opisujących zmienną celu. Proces ten zostanie przeprowadzony przy pomocy jednej z metod (ang Feature selection) RFE/ Lasso/ Mutual Information Score.
d).Ponowna optymalizacja parametrów modelu, tym razem dla podzbioru zmiennych.
e).Walidacja otrzymanego wyniku.

5.1 Model 1 (XGBClassifier)

```
# create a pipeline
pipe_1 = make_pipeline(preprocessor, selection, XGBClassifier(random_state=42))
```

```
pipe_1.fit(X, y)
```



5.1.2 Model 1 (XGBClassifier) - walidacja modelu

V.1.1 XGBClassifier - Evaluation

```
kf = KFold(n_splits=5, shuffle=True, random_state=42)
# kf = RepeatedKFold(n_splits=10, n_repeats=10, random_state=42)
# cross_val_score(pipe, X, y, cv=kf, scoring='r2').mean()

n_scores = cross_val_score(pipe_1, X, y, cv=kf, scoring='accuracy')
print('Accuracy: %.2f (%.2f)'% (np.mean(n_scores), np.std(n_scores)))
```

```
Accuracy: 0.82 (0.02)
```

```
cv = RepeatedKFold(n_splits=5, n_repeats=3, random_state=42)
n_scores = cross_val_score(pipe_1, X, y, cv=cv, scoring='f1', n_jobs=1)
print('F1 score: %.2f (%.2f)'% (np.mean(n_scores), np.std(n_scores)))
```

```
F1 score: 0.77 (0.03)
```

```
kf = KFold(n_splits=5, shuffle=True, random_state=42)
n_scores = cross_val_score(pipe_1, X, y, cv=kf, scoring='f1')
print('F1 score: %.2f (%.2f)'% (np.mean(n_scores), np.std(n_scores)))
```

```
F1 score: 0.76 (0.04)
```

```
kf = KFold(n_splits=5, shuffle=True, random_state=42)
n_scores = cross_val_score(pipe_1, X, y, cv=kf, scoring='roc_auc_ovo')
print('AUC: %.2f (%.2f)'% (np.mean(n_scores), np.std(n_scores)))
```

```
AUC: 0.87 (0.02)
```

5.1.3 Model 1 (XGBClassifier) - tuning modelu – RandomizedSearchCV()

```
params_13 = {}
# params['columntransformer__countvectorizer__min_df'] = [1, 2]
params_13['xgbclassifier__C'] = [0.1, 1, 10]
params_13['xgbclassifier__penalty'] = ['l1', 'l2']
params_13['xgbclassifier__max_depth'] = range(2, 9)
params_13['xgbclassifier__learning_rate'] = np.logspace(np.log10(0.005), np.log10(0.5), base = 10, num = 1000)
params_13['xgbclassifier__n_estimators'] = range(100, 500, 50)
params_13['xgbclassifier__gamma'] = np.linspace(0, 0.4, 6)
params_13['xgbclassifier__min_child_weight'] = range(1,20, 5)
params_13['xgbclassifier__subsample'] = np.linspace(0.5, 1, 11)
# params['xgbclassifier__colsample_bytree'] = np.linspace(0.5, 1)
# params['xgbclassifier__colsample_bylevel'] = np.linspace(0.6, 1)
```

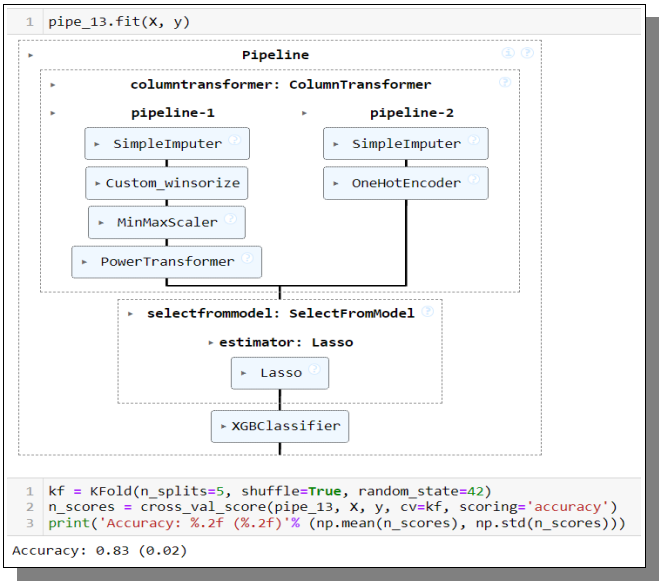
```
grid_13 = RandomizedSearchCV(pipe_1,
                    params_13,
                    cv=5,
                    scoring='accuracy')
```

```
grid_13.fit(X, y);
```

```
grid_13.best_params_
```

```
{'xgbclassifier__subsample': 0.9,
 'xgbclassifier__penalty': 'l1',
 'xgbclassifier__n_estimators': 200,
 'xgbclassifier__min_child_weight': 11,
 'xgbclassifier__max_depth': 8,
 'xgbclassifier__learning_rate': 0.4082083802460733,
 'xgbclassifier__gamma': 0.0,
 'xgbclassifier__C': 0.1}
```

```
pipe_13 = make_pipeline(preprocessor, selection, XGBClassifier(**grid_13.best_params_, random_state=42))
```



5.2 Model 2 (LGBMClassifier) - Tuning modelu – Optuna

```
def objective(trial):

    lgbm_param = {
        "verbosity": -1,
        "boosting_type": "gbdt",
        "random_state": 42,
        #"num_class": 7,
        "learning_rate": trial.suggest_float("learning_rate", 0.01, 0.03),
        "n_estimators": trial.suggest_int("n_estimators", 400, 600),
        "lambda_l1": trial.suggest_float("lambda_l1", 0.005, 0.025),
        "lambda_l2": trial.suggest_float("lambda_l2", 0.02, 0.06),
        "max_depth": trial.suggest_int("max_depth", 6, 14),
        "colsample_bytree": trial.suggest_float("colsample_bytree", 0.3, 0.9),
        "subsample": trial.suggest_float("subsample", 0.5, 1.0),
        "min_child_samples": trial.suggest_int("min_child_samples", 10, 50),
    }

    model_2 = LGBMClassifier( **lgbm_param)

    # aggregate numerical and categorical
    num_cols = make_column_selector(dtype_include='number')
    cat_cols = make_column_selector(dtype_exclude='number')

    # Numeric Columns
    imp_median = SimpleImputer(strategy='median', add_indicator=True)
    imp_knn = KNNImputer(n_neighbors=5, weights='uniform', metric='nan_euclidean')
    minmaxscaler = MinMaxScaler()
    power =  PowerTransformer()

    # Categorical Columns
    imp_constant = SimpleImputer(strategy='constant', fill_value='missing')
    ohe = OneHotEncoder(handle_unknown='ignore')
    ori = OrdinalEncoder()

    preprocessor = make_column_transformer(
                                        (make_pipeline(imp_median,customwinsorize, minmaxscaler, power), num_cols),
                                        (make_pipeline(imp_constant, ohe), cat_cols)
                                        )

    pipe_2 = make_pipeline(preprocessor, selection, model_2)
    skf = KFold(n_splits=5)
    cv_score = cross_val_score(pipe_2, X, y,  cv=skf, scoring='accuracy',verbose=1)
    return cv_score.mean()


study_2 = optuna.create_study(direction="maximize")
study_2.optimize(objective, n_trials=100)

print(study_2.best_trial)
```

5.2.1 Model 2 (LGBMClassifier) - Walidacja modelu

```
study_2.best_params
```

```
{'learning_rate': 0.015087750481586282,
 'n_estimators': 440,
 'lambda_l1': 0.01742988075002506,
 'lambda_l2': 0.050831431915479855,
 'max_depth': 11,
 'colsample_bytree': 0.8572089953298442,
 'subsample': 0.6314962155385591,
 'min_child_samples': 31}
```

```
pipe_2 = make_pipeline(preprocessor, selection, LGBMClassifier(**study_2.best_params, random_state=42))
```

```
pipe_2.fit(X,y)
```

...

```
kf = KFold(n_splits=5, shuffle=True, random_state=42)
n_scores = cross_val_score(pipe_2, X, y, cv=kf, scoring='accuracy')
print('Accuracy: %.2f (%.2f)'% (np.mean(n_scores), np.std(n_scores)))
```

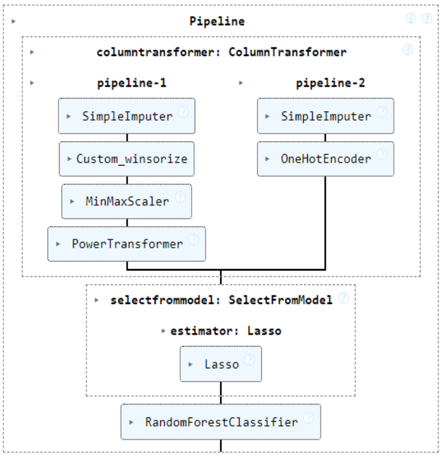
```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] lambda_l1 is set=0.01742988075002506, reg_alpha=0.0 will be ignored. Current value: lambda_l1=0.01742988
075002506
[LightGBM] [Warning] lambda_l2 is set=0.050831431915479855, reg_lambda=0.0 will be ignored. Current value: lambda_l2=0.050831
431915479855
Accuracy: 0.83 (0.01)
```

5.3 Model 3 (RandomForestClassifier) – budowa i walidacja modelu

V.3 RandomForestClassifier

```
pipe_3 = make_pipeline(preprocessor, selection, RandomForestClassifier(random_state=42))
```

```
pipe_3.fit(X, y)
```




```
kf = KFold(n_splits=5, shuffle=True, random_state=42)
n_scores = cross_val_score(pipe_3, X, y, cv=kf, scoring='accuracy')
print('Accuracy: %.2f (%.2f)'% (np.mean(n_scores), np.std(n_scores)))
```

```
Accuracy: 0.82 (0.02)
```

5.3.1 Model 3 (RandomForestClassifier) -Tuning modelu

V.3.1 RandomForestClassifier Tuning - RandomizedSearchCV

```
params_31 = {}
params_31['randomforestclassifier__n_estimators'] = range(100, 500)
params_31['randomforestclassifier__min_samples_split'] = range(2,9)
params_31['randomforestclassifier__min_samples_leaf'] = range(1,3)
params_31['randomforestclassifier__max_features'] = ['auto', 'sqrt', 'log2']
params_31['randomforestclassifier__max_depth'] = range(2, 9)
params_31['randomforestclassifier__criterion'] = ["gini", "entropy", "log_loss"]
params_31['randomforestclassifier__bootstrap'] = [True, False]
```

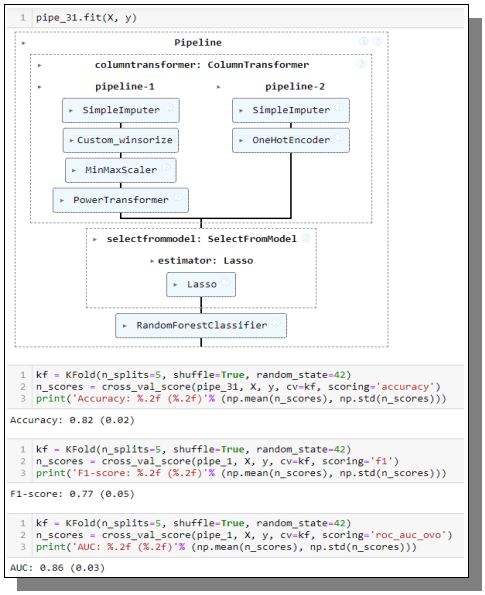
```
grid_31 = RandomizedSearchCV(pipe_3,
                    params_31,
#                             n_iter=4*4*4*2,
                    cv=5,
                    scoring='accuracy',
                    random_state=42
                    )
```

```
grid_31.fit(X, y);
```

```
grid_31.best_params_
```

```
{'randomforestclassifier__n_estimators': 368,
 'randomforestclassifier__min_samples_split': 4,
 'randomforestclassifier__min_samples_leaf': 2,
 'randomforestclassifier__max_features': 'sqrt',
 'randomforestclassifier__max_depth': 8,
 'randomforestclassifier__criterion': 'gini',
 'randomforestclassifier__bootstrap': True}
```

```
pipe_31 = make_pipeline(preprocessor, selection, RandomForestClassifier(n_estimators= 368,
 min_samples_split= 4,
 min_samples_leaf= 2,
 max_features= 'sqrt',
 max_depth= 8,
 criterion= 'gini',
 bootstrap= True, random_state=42))
```




```
# Function to take the true and predicted values and provide scoring
def evaluate_preds(true,preds):
    """
    Performs evalution comparisons on y_true values vs. y_pred values on a classification.
    """

    # create a dictionary of scoring values
    metric_dict = {
        'accuracy':round(accuracy_score(true,preds),2),
        'precision':round(precision_score(true,preds),2),
        'recall':round(recall_score(true,preds),2),
        'f1':round(f1_score(true,preds),2)
    }

    # print the calculated metrics
    for item in metric_dict:
        if item == 'accuracy':
            print(f'{item}: {metric_dict[item]*100:.2f}%')
        else:
            print(f'{item}: {metric_dict[item]:.2f}')

    return metric_dict
```

6 Opis uzyskanych wyników /Finalna weryfikacja jakości modelu

Model 1 XGBClassifier
Opis modelu:
brak parametrów
random state = 42
KFold(n_splits=5, shuffle=True, random_state=42)

Model ten będzie wyznacznikiem dla kolejnych iteracji.
Wynik będzie każdorazowo badany na podstawie średniej z 5 zbudowanych modeli w walidacji krzyżowej, która została zastosowana w celu eliminacji tendencji do zbytniego dopasowywania się do zbioru.
Przy klasycznym podziale na zbiór uczący i walidacyjny, przy optymalizacji parametrów modelu można osiągnąć błędny wynik będący niemiarodajnym odzwierciedleniem rozkładu danych.
Wykonując 5-krotny podział na zbiór uczący i walidacyjny, wynik zostanie uśredniony i jest on bardziej miarodajny. Walidacja krzyżowa jest sposobem, który pozwoli uniknąć przeuczenia modelu.

Drugim sposób na uniknięcie zbytniego dopasowania modelu to dobór parametrów modelu. W przypadku lasu losowego będzie to przycięcie drzewa i sprawienie by nie było ono „zbyt pewne" podejmowanych decyzji.

Wyniki modelu:
Accuracy: 0.82%
F1-score: 77%
AUC: 87%

Model 1 XGBClassifier -Tuning modelu metodą RandomizedSearchCV.
Opis modelu:
Tuning parameters:

```
params_13['xgbclassifier__C'] = [0.1, 1, 10]
params_13['xgbclassifier__penalty'] = ['l1', 'l2']
params_13['xgbclassifier__max_depth'] = range(2, 9)
params_13['xgbclassifier__learning_rate'] = np.logspace(np.log10(0.005), np.log10(0.5), base = 10,
num = 1000)
params_13['xgbclassifier__n_estimators'] = range(100, 500, 50)
params_13['xgbclassifier__gamma'] = np.linspace(0, 0.4, 6)
params_13['xgbclassifier__min_child_weight'] = range(1,20, 5)
params_13['xgbclassifier__subsample'] = np.linspace(0.5, 1, 11)
```

random state = 42
KFold(n_splits=5, shuffle=True, random_state=42)

Wyniki modelu po tunningu metodą RandomizedSearchCV:
Accuracy: 0.83%
F1-score: %
AUC: %

Model 1 XGBClassifier -Tuning modelu metodą Optuna
Opis modelu:
Optuna: sampler=optuna.samplers.TPESampler() - algorytm optymalizacji Bayesowskiej
random state = 42
KFold(n_splits=5, shuffle=True, random_state=42)
Parametry, które będą tuningowane:

```
    "objective": "reg:squarederror",
    "n_estimators": trial.suggest_int("n_estimators", 50, 10000),
    "verbosity": 0,
    "learning_rate": trial.suggest_float("learning_rate", 1e-3, 0.1, log=True),
    "max_depth": trial.suggest_int("max_depth", 2, 15),
    "subsample": trial.suggest_discrete_uniform("subsample", 0.6, 1.0, 0.05),
    "colsample_bytree": trial.suggest_float("colsample_bytree", 0.05, 1.0),
    "min_child_weight": trial.suggest_int("min_child_weight", 1, 20),
    "reg_alpha": trial.suggest_int('reg_alpha', 1, 50),
```

```
    "reg_lambda": trial.suggest_int('reg_lambda', 5, 100),
```

Wyniki modelu po tunningu metodą Optuna:
Accuracy: 0.83%
F1-score: 77%
AUC: 86%

Model 2 LGBMClassifier -Tuning modelu metodą Optuna
Opis modelu:
Optuna: sampler=optuna.samplers.TPESampler() - algorytm optymalizacji Bayesowskiej
random state = 42
KFold(n_splits=5, shuffle=True, random_state=42)
Parametry, które będą tuningowane:

```
    "verbosity": -1,
    "boosting_type": "gbdt",
    "random_state": 42,
    #"num_class": 7,
    "learning_rate": trial.suggest_float("learning_rate", 0.01, 0.03),
    "n_estimators": trial.suggest_int("n_estimators", 400, 600),
    "lambda_l1": trial.suggest_float("lambda_l1", 0.005, 0.025),
    "lambda_l2": trial.suggest_float("lambda_l2", 0.02, 0.06),
    "max_depth": trial.suggest_int("max_depth", 6, 14),
    "colsample_bytree": trial.suggest_float("colsample_bytree", 0.3, 0.9),
    "subsample": trial.suggest_float("subsample", 0.5, 1.0),
    "min_child_samples": trial.suggest_int("min_child_samples", 10, 50),
```

Wyniki modelu po tunningu metodą Optuna:
Accuracy: 0.83%
F1-score: 77%
AUC: 86%

Model 3 RandomForestClassifier -Tuning modelu metodą RandomizedSearchCV.
Opis modelu:
-random state = 42
-KFold(n_splits=5, shuffle=True, random_state=42)
-Lasso(0.01)
-Parametry, które będą tuningowane:

- n_estimators (liczba drzew decyzyjnych w lesie)
- min_samples_split (kontroluje minimalną liczbę próbek wymaganą do podziału węzła wewnętrznego, jeśli liczba dostępnych próbek w węźle jest mniejsza niż ustalona wartość min_samples_split, to drzewo nie będzie już dzielić się na mniejsze gałęzie w tym miejscu. To pomaga kontrolować, jak daleko drzewo idzie w dopasowywaniu się do szczegółów danych treningowych, co może pomóc w uniknięciu zbytniego dopasowani)
- min_samples_leaf (minimalna liczba obserwacji do zbudowania liścia w drzzewie decyzyjnym, jeśli liczba próbek w danym węźle spadnie poniżej wartości min_samples_leaf, to nie będziemy już kontynuować dzielenia drzewa w tym kierunku, a ten węzeł będzie traktowany jako liść)
- max_features (maksymalna liczba cech, jaką losowy las może wykorzystać w danym drzewie)
- max_depth (maksymalna głębokość każdego drzewa decyzyjnego w lesie)
- criterion (kryterium podziału drzewa)
- bootstrap (procedura zmniejszenia wariancji)

```
params_31['randomforestclassifier__n_estimators'] = range(100, 500)
params_31['randomforestclassifier__min_samples_split'] = range(2,9)
params_31['randomforestclassifier__min_samples_leaf'] = range(1,3)
params_31['randomforestclassifier__max_features'] = ['auto', 'sqrt', 'log2']
params_31['randomforestclassifier__max_depth'] = range(2, 9)
params_31['randomforestclassifier__criterion'] = ["gini", "entropy", "log_loss"]
params_31['randomforestclassifier__bootstrap'] = [True, False]
```

-random state = 42

-KFold(n_splits=5, shuffle=True, random_state=42)
-Lasso(0.001)

Wyniki modelu:
Accuracy: 0.82%
F1-score: 77%
AUC: 86%

7 Podsumowanie

W powyższym „projekcie” zaprezentowano jedynie trzy iteracje, które okazały się zdecydowanie najlepsze. Warto zaznaczyć, że iteracji prowadzących do końcowego wyniku było zdecydowanie więcej. Użycie tego samego zbioru danych i innej implementacji drzewa decyzyjnego (losowy las, xgboost itp.) z tuningiem nie przekracza dokładności przesyłania 0,77990. Co ciekawe w przypadku tego zbioru danych, prosty algorytm drzewa decyzyjnego miał najlepszy domyślny wynik przesłania, a po dostrojeniu osiągnął ten sam najlepszy wynik dokładności. Pomimo dostrojenia żaden algorytm uczenia maszynowego nie przewyższył podstawowego algorytmu. W przypadku małych zbiorów danych barierą do pokonania jest nie tyle algorytm co przystosowanie do niego danych. Mając to na uwadze, w drugiej iteracji należało poświęcić więcej czasu na przetwarzanie wstępne i inżynierię funkcji. Aby lepiej wyrównać wynik i poprawić ogólną dokładność. Podsumujmy, dzięki podstawowym algorytmom czyszczenia, analizy i uczenia maszynowego (EDA) danych jesteśmy w stanie przewidzieć przeżycie pasażerów z dokładnością ~81%. Pytanie, które zawsze zadajemy, brzmi: czy możemy działać lepiej i, co ważniejsze, uzyskać zwrot z inwestycji (ROI) za zainwestowany czas? Na przykład, jeśli mamy zamiar zwiększyć naszą dokładność tylko o 1/10 procent, czy naprawdę warto przeprowadzać kosztowne 3-miesięczne prace badawcze. Jeśli projekt jest szczególnie uwarunkowany na jak najwyższą dokładność np. badania naukowe, być może odpowiedź brzmi „tak”, ale jeśli projekt odnosi się do biznesu, odpowiedź brzmi „nie”.