Python Programming
for Intermediate

Python Project
Titanic Challenge

2023학년도 2학기

Suk-Hwan Lee

Artificial Intelligence

Creating the Future

Dong-A University

Division of Computer Engineering &
Artificial Intelligence

➢ Titanic - Machine Learning from Disaster

https://www.kaggle.com/c/titanic/overview

[Recommend]

- https://www.kaggle.com/startupsci/titanic-data-science-solutions
- https://www.kaggle.com/pedrodematos/titanic-a-complete-approach-for-data-scientists
- https://kaggle-kr.tistory.com/17

The Challenge

The sinking of the Titanic is one of the most infamous shipwrecks in history.

On April 15, 1912, during her maiden voyage, the widely considered “unsinkable” RMS Titanic sank after colliding with an iceberg. Unfortunately, there weren’t enough lifeboats for everyone onboard, resulting in the death of 1502 out of 2224 passengers and crew.

While there was some element of luck involved in surviving, it seems some groups of people were more likely to survive than others.

In this challenge, we ask you to build a predictive model that answers the question: “what sorts of people were more likely to survive?” using passenger data (ie name, age, gender, socio-economic class, etc).

➢ Titanic - Machine Learning from Disaster

Data Description

Overview

The data has been split into two groups:

- training set (train.csv)
- test set (test.csv)

The training set should be used to build your machine learning models. For the training set, we provide the outcome (also known as the "ground truth") for each passenger. Your model will be based on "features" like passengers' gender and class. You can also use feature engineering to create new features.

The test set should be used to see how well your model performs on unseen data. For the test set, we do not provide the ground truth for each passenger. It is your job to predict these outcomes. For each passenger in the test set, use the model you trained to predict whether or not they survived the sinking of the Titanic.

We also include **gender_submission.csv**, a set of predictions that assume all and only female passengers survive, as an example of what a submission file should look like.

- Data Dictionary

Variable	Definition	Key
survival	Survival	0 = No, 1 = Yes
pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
sex	Sex	
Age	Age in years	
sibsp	# of siblings / spouses aboard the Titanic	
parch	# of parents / children aboard the Titanic	
ticket	Ticket number	
fare	Passenger fare	
cabin	Cabin number	
embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

➢ Titanic - Machine Learning from Disaster

- Data Dictionary

Variable Notes

pclass: A proxy for socio-economic status (SES)
1st = Upper
2nd = Middle
3rd = Lower

age: Age is fractional if less than 1. If the age is estimated, is it in the form of xx.5

sibsp: The dataset defines family relations in this way...
Sibling = brother, sister, stepbrother, stepsister
Spouse = husband, wife (mistresses and fiancés were ignored)

parch: The dataset defines family relations in this way...
Parent = mother, father
Child = daughter, son, stepdaughter, stepson
Some children travelled only with a nanny, therefore parch=0 for them.

Data Explorer

90.9 KB

gender_submission.csv
test.csv
train.csv

gender_submission.cs...

Detail | Compact | Column | 2 of 2 columns

About this file

An example of what a submission file should look like.

These predictions assume only female passengers survive.

PassengerId	Survived
892	0
893	1
894	0

Titanic - Machine Learning from Disaster

➢ Titanic Data Science Solutions

Titanic Data Science Solutions

Python notebook using data from Titanic - Machine Learning from Disaster · 1,184,445 views · 3y ago · feature engineering, model comparison

Titanic Data Science Solutions

This notebook is a companion to the book Data Science Solutions.

The notebook walks us through a typical workflow for solving data science competitions at sites like Kaggle.

There are several excellent notebooks to study data science competition entries. However many will skip some of the explanation on how the solution is developed as these notebooks are developed by experts for experts. The objective of this notebook is to follow a step-by-step workflow, explaining each step and rationale for every decision we take during solution development.

Workflow stages

The competition solution workflow goes through seven stages described in the Data Science Solutions book.

1. Question or problem definition.
2. Acquire training and testing data.
3. Wrangle, prepare, cleanse the data.
4. Analyze, identify patterns, and explore the data.
5. Model, predict and solve the problem.
6. Visualize, report, and present the problem solving steps and final solution.
7. Supply or submit the results.

개념 이해에 좋은 자료

sklearn의 머신러닝 모델 이용

- ✓ Logistic Regression
- ✓ KNN or k-Nearest Neighbors
- ✓ Support Vector Machines
- ✓ Naive Bayes classifier
- ✓ Decision Tree
- ✓ Random Forrest
- ✓ Perceptron
- ✓ RVM or Relevance Vector Machine

	Model	Score
3	Random Forest	86.76
8	Decision Tree	86.76
1	KNN	84.74
0	Support Vector Machines	83.84
2	Logistic Regression	80.36
7	Linear SVC	79.12
6	Stochastic Gradient Decent	78.56
5	Perceptron	78.00
4	Naive Bayes	72.28

Titanic: A complete approach for Data Scientists

➢ Titanic - Machine Learning from Disaster

Titanic: A complete approach for Data Scientists

Python notebook using data from Titanic - Machine Learning from Disaster · 36,111 views · 3mo ago · exploratory data analysis, classification, data cleaning, +2 more

Titanic Dataset: Automatic EDA, different Data Preprocessing & Modeling Techniques compared with Pipelines + RandomSearchCV and much more!

- To try to present a complete approach to modeling problems, that goes from Exploratory Data Analysis (EDA) to applying *Supervised* and *Unsupervised* learning techniques to our data.
- This notebook's content is mainly directed to data scientists, data science students or people interested in how these techniques can be applied into data.

- Thanks to Andreas C. Muller, Sarah Guido & other co-authors, for writing the book "Introduction to Machine Learning with Python". A great source of knowledge for Data Scientists of all levels.
- Notebook written by Pedro de Matos Gonçalves

Section 1 : Data Exploration

- The first step is to import needed libraries.

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load
import string # library used to deal with text data
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns # data visualization library
pd.set_option('display.max_columns', 100) # Setting pandas to display a N number of columns
pd.set_option('display.max_rows', 10) # Setting pandas to display a N number rows
pd.set_option('display.width', 100) # Setting pandas dataframe display width to N

from scipy import stats # statistical library
from statsmodels.stats.weightstats import ztest # statistical library for hypothesis testing
import plotly.graph_objs as go # interactive plotting library
import plotly.express as px # interactive plotting library
from itertools import cycle # used for cycling colors at plotly graphs
import matplotlib.pyplot as plt # plotting library
import pandas_profiling # library for automatic EDA

# installing and importing autoviz, another library for automatic data visualization
from autoviz.AutoViz_Class import AutoViz_Class
from IPython.display import display # display from IPython.display
from itertools import cycle # function used for cycling over values

# installing ppscore, library used to check non-linear relationships between our variables
import ppscore as pps # importing ppscore
```

터미널 상 모듈 설치 필요

- conda install seaborn, statsmodels, plotly,
- pip install pandas_profiling, autoviz, ppscore, xlrd, wordcoud

> conda install seaborn statsmodels plotly
> pip install pandas_profiling autoviz ppscore xlrd wordcloud

plotly : https://plotly.com/python/
autoviz : https://pypi.org/project/autoviz/
ppscore : https://pypi.org/project/ppscore/

Section 1 : Data Exploration

- The first step is to import needed libraries.

```
# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory
import os
print("")
for dirname, _, filenames in os.walk('data/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 5GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

```
Imported AutoViz_Class version: 0.0.84. Call using:
    AV = AutoViz_Class()
    AV.AutoViz(filename, sep=',', depVar='', dfte=None, header=0, verbose=0,
                            lowess=False,chart_format='svg',max_rows_analyzed=150000,max_cols_analyzed=30)
Note: verbose=0 or 1 generates charts and displays them in your local Jupyter notebook.
      verbose=2 does not show plot but creates them and saves them in AutoViz_Plots directory in your local machine.

data/input\gender_submission.csv
data/input\test.csv
data/input\train.csv
```

```
# Check : os.walk
for files in os.walk('data'):
    print(files)
```

```
('data', ['input'], ['report.html'])
('data\\input', [], ['gender_submission.csv', 'test.csv', 'train.csv'])
```

(경로, 경로 내 디렉터리 리스트, 경로 내 파일 리스트)

- os.listdir(path) : 특정 경로 내에 존재하는 폴더(디렉토리)와 파일 리스트를 검색 (1-depth)
- os.walk(path) : 특정 경로 내에 존재하는 폴더(디렉토리)와 파일 리스트 뿐만 아니라, 모든 하위 디렉토리 구조를 다 검색

Section 1 : Data Exploration

Exploratory Data Analysis (EDA) Process

1) 정의

- 수집한 데이터가 들어왔을 때, 이를 다양한 각도에서 관찰하고 이해하는 과정입니다. 한마디로 데이터를 분석하기 전에 그래프나 통계적인 방법으로 자료를 직관적으로 바라보는 과정입니다.

2) 필요한 이유

- 데이터의 분포 및 값을 검토함으로써 데이터가 표현하는 현상을 더 잘 이해하고, 데이터에 대한 잠재적인 문제를 발견할 수 있습니다. 이를 통해, 본격적인 분석에 들어가기에 앞서 데이터의 수집을 결정할 수 있습니다.
- 다양한 각도에서 살펴보는 과정을 통해 문제 정의 단계에서 미쳐 발생하지 못했을 다양한 패턴을 발견하고, 이를 바탕으로 기존의 가설을 수정하거나 새로운 가설을 세울 수 있습니다.

3) 과정

- 기본적인 출발점은 문제 정의 단계에서 세웠던 연구 질문과 가설을 바탕으로 분석 계획을 세우는 것입니다. 분석 계획에는 어떤 속성 및 속성 간의 관계를 집중적으로 관찰해야 할지, 이를 위한 최적의 방법은 무엇인지가 포함되어야 합니다.
- **분석의 목적과 변수가 무엇이 있는지 확인**. 개별 변수의 이름이나 설명을 가지는지 확인
- **데이터를 전체적으로 살펴보기** : 데이터에 문제가 없는지 확인. head나 tail 부분을 확인, 추가적으로 다양한 탐색(이상치, 결측치 등을 확인하는 과정)
- **데이터의 개별 속성값을 관찰** : 각 속성 값이 예측한 범위와 분포를 갖는지 확인. 만약 그렇지 않다면, 이유가 무엇인지를 확인.
- 속성 간의 관계에 초점을 맞추어, **개별 속성 관찰에서 찾아내지 못했던 패턴을 발견** (상관관계, 시각화 등)

참고 예시

- https://eda-ai-lab.tistory.com/13
- https://www.kaggle.com/subinium/kakr-eda

Section 1 : Data Exploration

- To begin our analysis, lets take our first look at the dataset. To save some precious time on our **Exploratory Data Analysis (EDA)** process, we are going to use 2 libraries: "**pandas_profiling**" and "**autoviz**".
- **pandas_profiling** : generates profile reports from a pandas DataFrame.

```
# Importing the data and displaying some rows
df = pd.read_csv("data/input/train.csv")
display(df.head()) # df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

```
# The pandas profiling library is really useful on helping us
# understand the data we're working on.
# It saves us some precious time on the EDA process.
report = pandas_profiling.ProfileReport(df)

# Let's now visualize the report generated by pandas_profiling.
display(report)

# Also, there is an option to generate an .HTML file containing all the information generated by the report.
report.to_file(output_file='data/report.html')
```

```
Summarize dataset: 100% 25/25 [00:03<00:00, 4.91it/s, Completed
Generate report structure: 100% 1/1 [00:03<00:00, 3.10s/it]
Render HTML: 100% 1/1 [00:01<00:00, 1.43s/it]
```

Pandas Profiling Report — Overview | Variables | Interactions | Correlations | Missing values | Sample

Overview

Overview | Warnings 33 | Reproduction

Dataset statistics

Number of variables	12
Number of observations	891
Missing cells	866
Missing cells (%)	8.1%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	83.7 KiB
Average record size in memory	96.1 B

Variable types

Numeric	5
Categorical	7

- report 결과에서 train.csv의 df 데이터 확인 !!!

Section 1 : Data Exploration

- **AutoViz** : great library for automatic EDA

```
# Another great library for automatic EDA is AutoViz.
# With this library, several plots are generated with only 1 line of code.
# When combined with pandas_profiling, we obtain lots of information in a
# matter of seconds, using less than 5 lines of code.
AV = AutoViz_Class()

# Let's now visualize the plots generated by AutoViz.
report_2 = AV.AutoViz(filename="data/input/train.csv", verbose=2, save_plot_dir="data/")
```

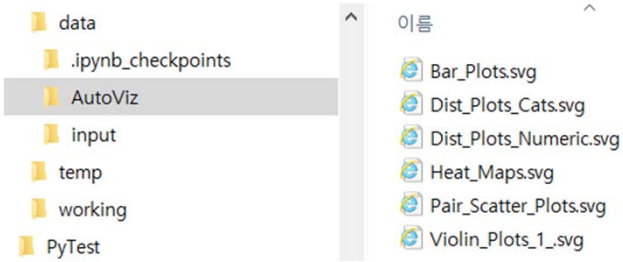
```
Shape of your Data Set loaded: (891, 12)
#######################################################################################
###################### C L A S S I F Y I N G  V A R I A B L E S  ######################
#######################################################################################
Classifying variables in data set...
Data cleaning improvement suggestions. Complete them before proceeding to ML modeling.
```

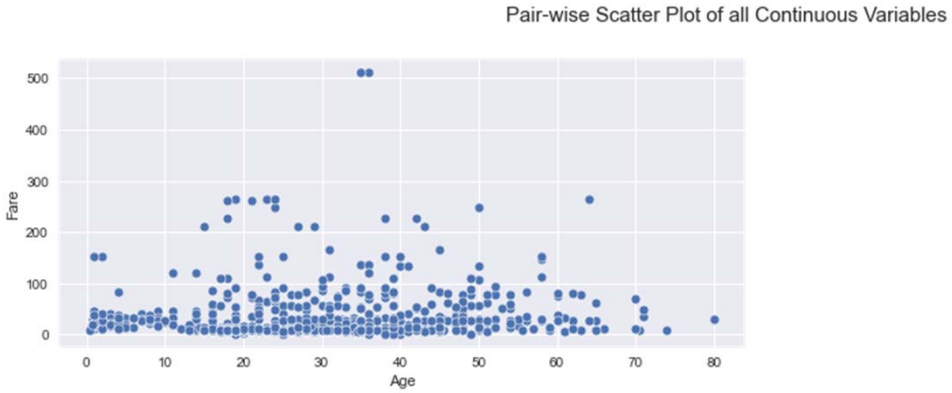
	Nuniques	dtype	Nulls	Nullpercent	NuniquePercent	Value counts Min	Data cleaning improvement suggestions
PassengerId	891	int64	0	0.000000	100.000000	0	possible ID column: drop
Name	891	object	0	0.000000	100.000000	1	combine rare categories, possible ID column: drop
Ticket	681	object	0	0.000000	76.430976	1	combine rare categories
Fare	248	float64	0	0.000000	27.833895	0	skewed: cap or drop outliers
Cabin	147	object	687	77.104377	16.498316	1	combine rare categories, fill missing, fix mixed data types
Age	88	float64	177	19.865320	9.876543	0	fill missing
SibSp	7	int64	0	0.000000	0.785634	0	
Parch	7	int64	0	0.000000	0.785634	0	
Pclass	3	int64	0	0.000000	0.336700	0	
Embarked	3	object	2	0.224467	0.336700	77	fill missing, fix mixed data types
Survived	2	int64	0	0.000000	0.224467	0	
Gender	2	object	0	0.000000	0.224467	314	

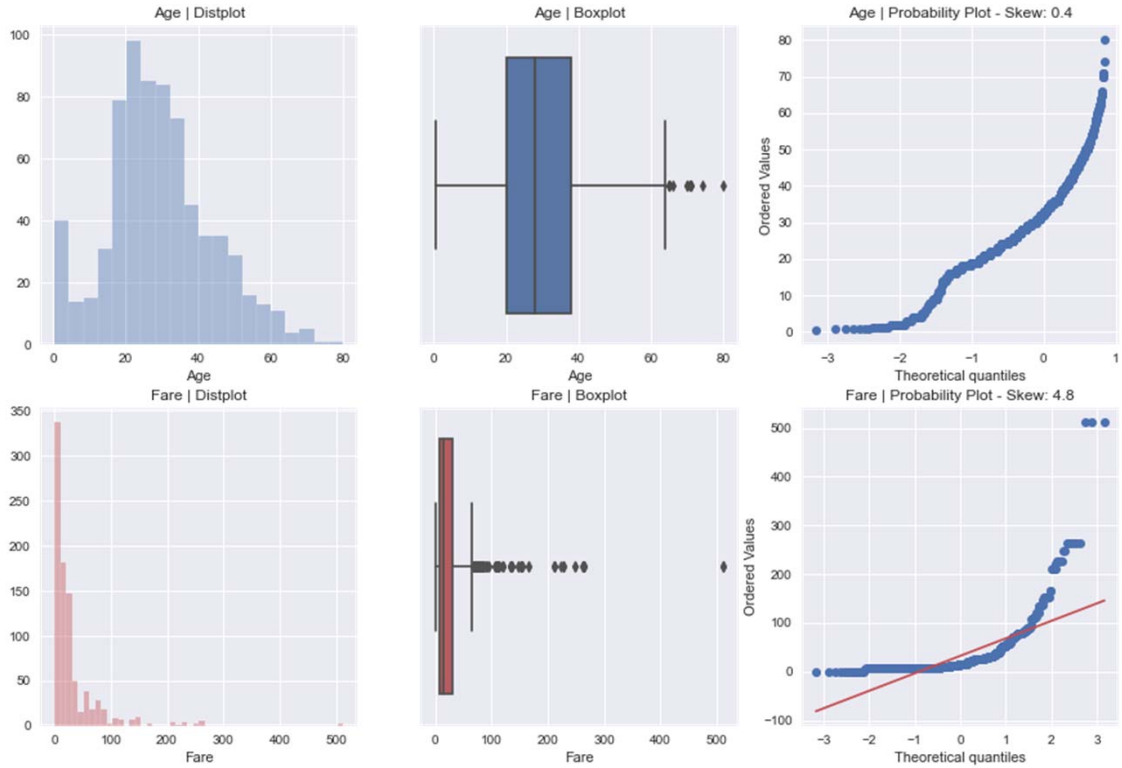
```
Printing upto 30 columns max in each category:
  Numeric Columns : ['Age', 'Fare']
  Integer-Categorical Columns: ['Pclass', 'SibSp', 'Parch']
  String-Categorical Columns: ['Embarked']
  Factor-Categorical Columns: []
  String-Boolean Columns: ['Gender']
  Numeric-Boolean Columns: ['Survived']
  Discrete String Columns: ['Ticket', 'Cabin']
  NLP text Columns: ['Name']
  Date Time Columns: []
  ID Columns: ['PassengerId']
  Columns that will not be considered in modeling: []
  12 Predictors classified...
      1 variables removed since they were ID or low-information variables
      List of variables removed: ['PassengerId']
  Columns to delete:
'   []'
  Boolean variables %s
"   ['Gender', 'Survived']"
  Categorical variables %s
"   ['Embarked', 'Pclass', 'SibSp', 'Parch', 'Gender', 'Survived']"
  Continuous variables %s
"   ['Age', 'Fare']"
  Discrete string variables %s
"   ['Name', 'Ticket', 'Cabin']"
  Date and time variables %s
'   []'
  ID variables %s
"   ['PassengerId']"
  Target variable %s
'   '
Number of All Scatter Plots = 3
```

Section 1 : Data Exploration

- **AutoViz** : great library for automatic EDA



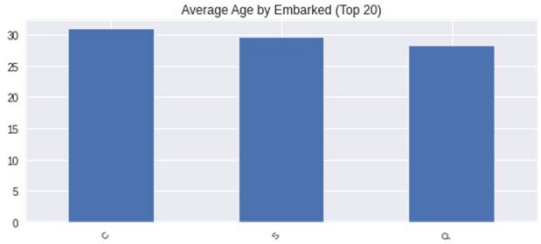


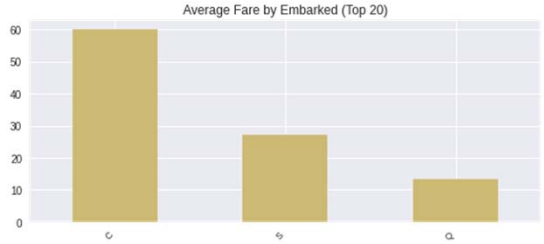


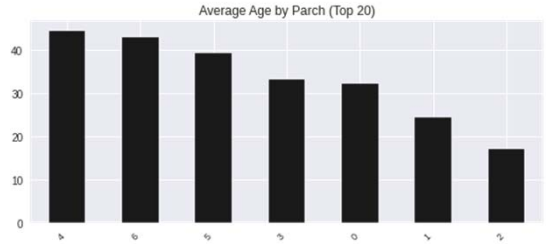
Section 1 : Data Exploration

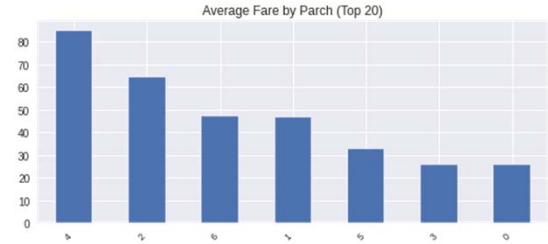
- **AutoViz** : great library for automatic EDA

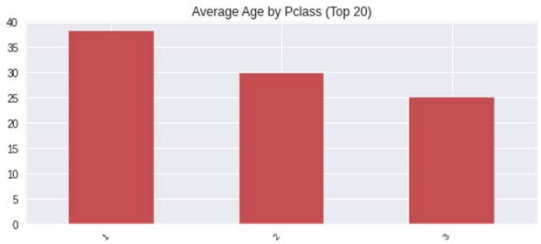
Bar plots for each continuous by each Categorical variable

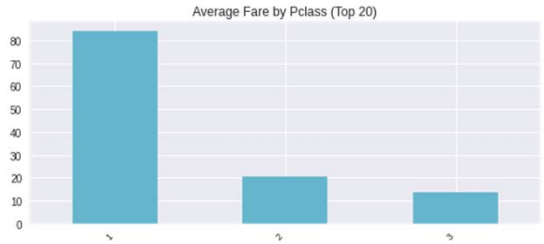


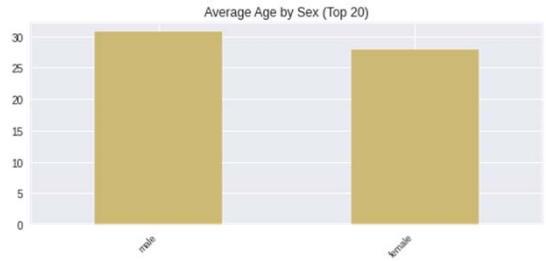


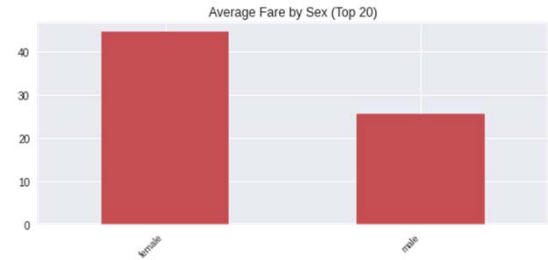


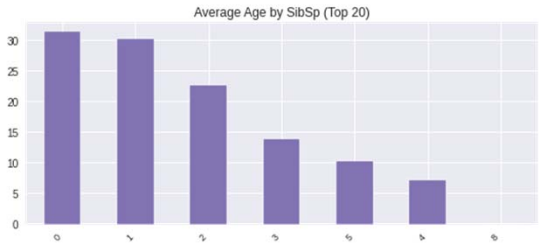


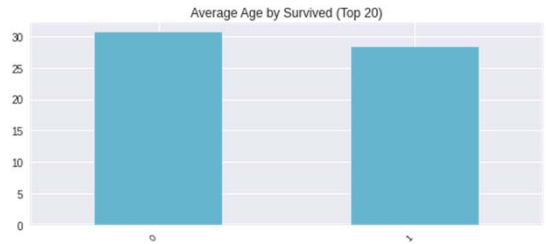


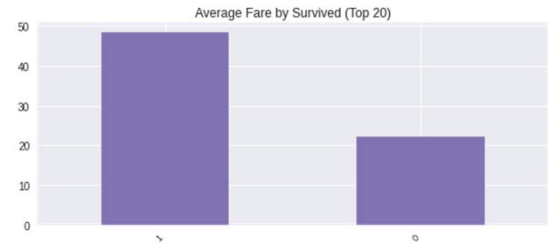












Section 1 : Data Exploration

- Using the power of both automatic EDA libraries listed above, we can observe each variable's behaviour individually, with plots that goes from Histograms to Boxplots, Correlation Matrix and much more. It speeds up time and minimizes the effort spent on the initial process of our work.
- Gather some really useful information from both reports.

✓ [Survive/not] Our classes are not that much disbalanced. We have ~38% of the passengers into class "1" (survived) and ~62% of the passengers into class "0" (didn't survive).

✓ [Pclass] The "Pclass" column, that informs us about the passenger's ticket class, shows us that ~55% of them are on class 3, ~24% of them are on class 2 and ~21% on class 1.

✓ [Gender] Most of the passengers into this dataset are male: ~35% of the passengers are female, and ~65% are male.

✓ [Age] Almost 20% of the values in the "Age" column are missing. We can fill out these nulls with various techniques, such as filling them with the distribution's mean. The ages distribution is a little bit skewed, with it's mean being around 30 years old, and it's standard deviation being close to 15. The oldest passenger we have in this dataset is 80 years old.

✓ [SibSP] According to the "SibSP" column, most of the passengers (~68%) didn't have any spouses or siblings aboard the ship. That is also applied when we check out the "Parch" column.

✓ [Fares] The distribution of Fares is much more skewed. It's mean value is around 32, with it's standard deviation being close to 50. It's minimum value is 0, and it's maximum value is 512.3292. That means that *we're going to have to deal with this column carefully if we plan to use models such as SVMs*.

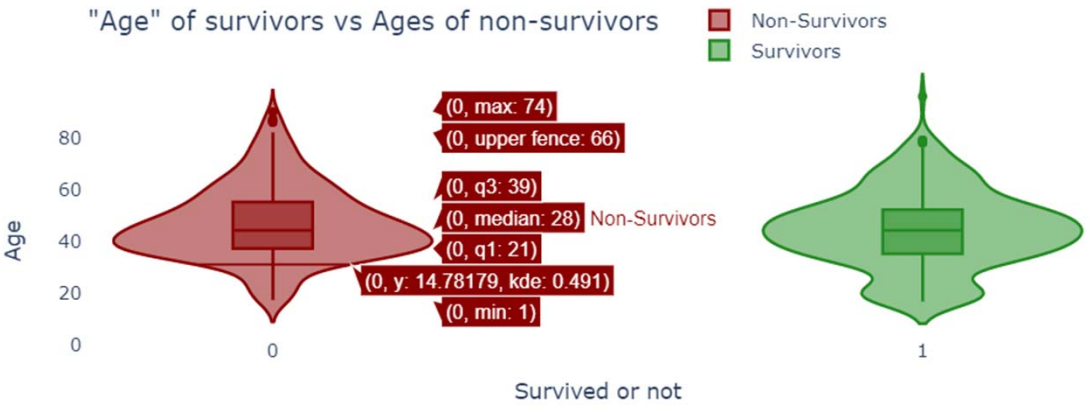
✓ [Embarked] When ckecking the "Embarked" column, it shows us that 72.3% of the passengers embarked at *Southampton* port, 18.9% of the passengers at *Cherbourg* port and 8.6% of the passengers at *Queenstown* port.

✓ "Fare" values are higher for passengers with "Pclass" = 1, lower for passengers with "Pclass" = 2 and even lower for passengers with "Pclass" = 3. Logically, it looks like *the classification of "Pclass" is defined by the value of the passenger's fare*.

Section 1 : Data Exploration

More Exploration

- Before go to the modeling part, let's take a look at a more plots that gives us a different perspective from the ones generated above.
- That may give us further insights and help us understand the differences between the passengers that survived the catastrophe and the people that didn't. For these visualizations, we are going to use **Plotly**, a library that allows us to **interact with** them.
- First, let's take a look at the differences between the ages of both groups, using a **Violin plot**.




```
# Creating different datasets for survivors and non-survivors
df_survivors = df[df['Survived'] == 1]
df_nonsurvivors = df[df['Survived'] == 0]
```

```
# Filling in the data inside the Violin Objects
# (import plotly.graph_objs as go # interactive plotting library)
violin_survivors = go.Violin(
    y=df_survivors['Age'],
    x=df_survivors['Survived'],
    name='Survivors',
    marker_color='forestgreen',
    box_visible=True)

violin_nonsurvivors = go.Violin(
    y=df_nonsurvivors['Age'],
    x=df_nonsurvivors['Survived'],
    name='Non-Survivors',
    marker_color='darkred',
    box_visible=True)
```

import plotly.graph_objs as go

https://plotly.com/python/violin/

```
data = [violin_nonsurvivors, violin_survivors]

# Plot's Layout (background color, title, etc.)
layout = go.Layout(
    paper_bgcolor='rgba(0,0,0,0)',
    plot_bgcolor='rgba(0,0,0,0)',
    title='"Age" of survivors vs Ages of non-survivors',
    xaxis=dict(
        title='Survived or not'
    ),
    yaxis=dict(
        title='Age'
    )
)

fig = go.Figure(data=data, layout=layout)
fig.show()
```

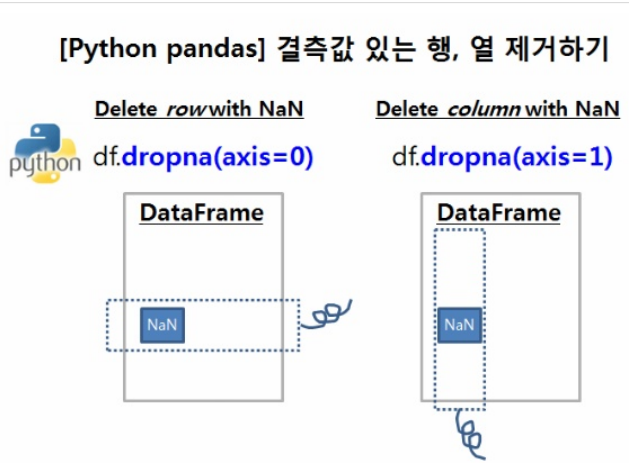
Section 1 : Data Exploration

- z-test / t-test : ages of survivors vs ages of non-survivors

```
# First distribution for the hypothesis test: Ages of survivors
dist_a = df_survivors['Age'].dropna()

# Second distribution for the hypothesis test: Ages of non-survivors
dist_b = df_nonsurvivors['Age'].dropna()
```

```
# Z-test: Checking if the distribution means
# (ages of survivors vs ages of non-survivors) are statistically different
t_stat, p_value = ztest(dist_a, dist_b)
print("----- Z Test Results -----")
print("T stat. = " + str(t_stat))
print("P value = " + str(p_value)) # P-value is less than 0.05

print("")

# T-test: Checking if the distribution means
# (ages of survivors vs ages of non-survivors) are statistically different
t_stat_2, p_value_2 = stats.ttest_ind(dist_a, dist_b)
print("----- T Test Results -----")
print("T stat. = " + str(t_stat_2))
print("P value = " + str(p_value_2)) # P-value is less than 0.05
```

```
----- Z Test Results -----
T stat. = -2.06668694625381
P value = 0.03876366199189209

----- T Test Results -----
T stat. = -2.06668694625381
P value = 0.03912465401348249
```



- **t-test** (또는 Student's t-test) : 일반적으로 test 통계량이 정규 분포를 따르며 분포와 관련된 스케일링 변수값들이 알려진 경우에 사용한다. 이 때 모집단의 분산과 같은 스케일링 항을 알 수 없으나 데이터를 기반으로 한 추정값으로 대체하면 test 통계량은 t-분포를 따른다. 예를 들어 t-테스트를 사용하여 두 데이터 세트(집단)의 평균이 서로 유의하게 다른지 여부를 판별 할 수 있다.
- **z-test** (정규분포 기준) : 분산 σ^2(또는 표준편차)를 이미 알고 있는 모집단 분포의 평균과 샘플(표본)과의 두 평균을 테스트한다. 중심 극한 정리(central limit theorem)로 인해 많은 테스트 통계는 일반적으로 큰 샘플에 대해 대략적으로 정규분포를 따르게 된다. 적어도 표본 크기는 T-테스트와 마찬가지로 일정 샘플수(n <30)를 넘어야한다.

- As we can see both from the plot and hypothesis tests showed above, there is actually a statistically significant difference between the means of both distributions (ages of survivors and non-survivors). Let's do some more exploring to see what further information we can gather from this data.

Section 1 : Data Exploration

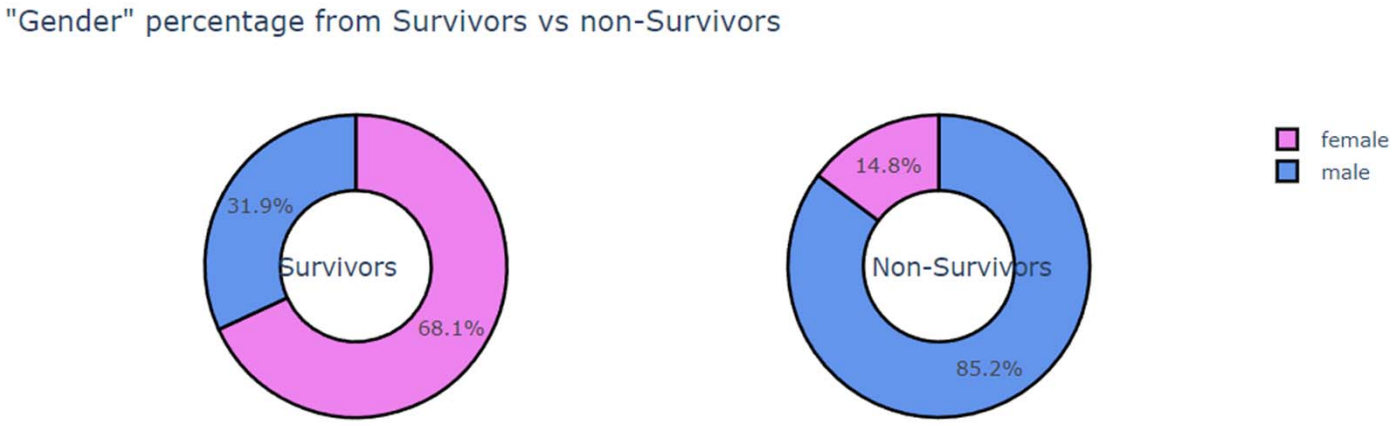
- Male/Female percentage from survivors vs non-survivors

```
# Taking the count of each Gender value inside the Survivors
df_survivors_Gender = df_survivors['Gender'].value_counts()
df_survivors_Gender = pd.DataFrame({'Gender':df_survivors_Gender.index, 'count':df_survivors_Gender.values})

# Taking the count of each Gender value inside the Survivors
df_nonsurvivors_Gender = df_nonsurvivors['Gender'].value_counts()
df_nonsurvivors_Gender = pd.DataFrame({'Gender':df_nonsurvivors_Gender.index, 'count':df_nonsurvivors_Gender.values})

# Creating the plotting objects
pie_survivors_Gender = go.Pie(
   labels = df_survivors_Gender['Gender'],
   values = df_survivors_Gender['count'],
   domain=dict(x=[0, 0.5]),
   name='Survivors',
   hole = 0.5,
   marker = dict(colors=['violet', 'cornflowerblue'], line=dict(color='#000000', width=2))
)

pie_nonsurvivors_Gender = go.Pie(
   labels = df_nonsurvivors_Gender['Gender'],
   values = df_nonsurvivors_Gender['count'],
   domain=dict(x=[0.5, 1.0]),
   name='non-Survivors',
   hole = 0.5,
   marker = dict(colors=['cornflowerblue', 'violet'], line=dict(color='#000000', width=2))
)
```

Section 1 : Data Exploration

- Male/Female percentage from survivors vs non-survivors

```
data = [pie_survivors_Gender, pie_nonsurvivors_Gender]

# Plot's Layout (background color, title, annotations, etc.)
layout = go.Layout(
    paper_bgcolor='rgba(0,0,0,0)',
    plot_bgcolor='rgba(0,0,0,0)',
    title='"Gender" percentage from Survivors vs non-Survivors',
    annotations=[dict(text='Survivors', x=0.18, y=0.5, font_size=15, showarrow=False),
                 dict(text='Non-Survivors', x=0.85, y=0.5, font_size=15, showarrow=False)]
)

fig = go.Figure(data=data, layout=layout)
fig.show()
```

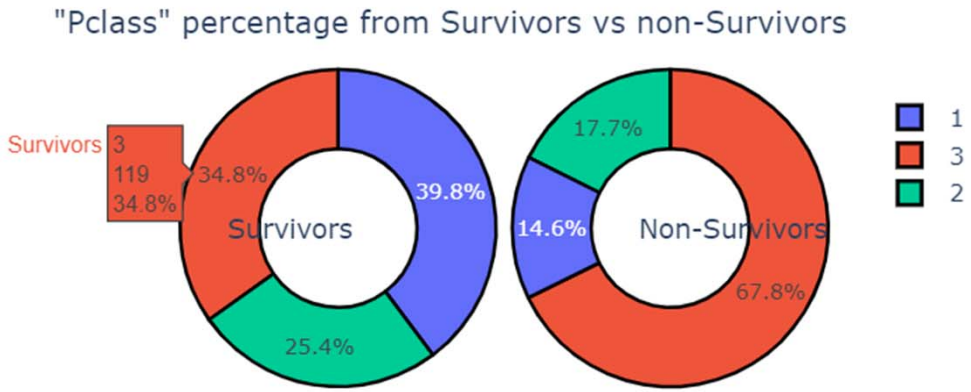


Section 1 : Data Exploration

- "Pclass" percentage from Survivors vs non-Survivors

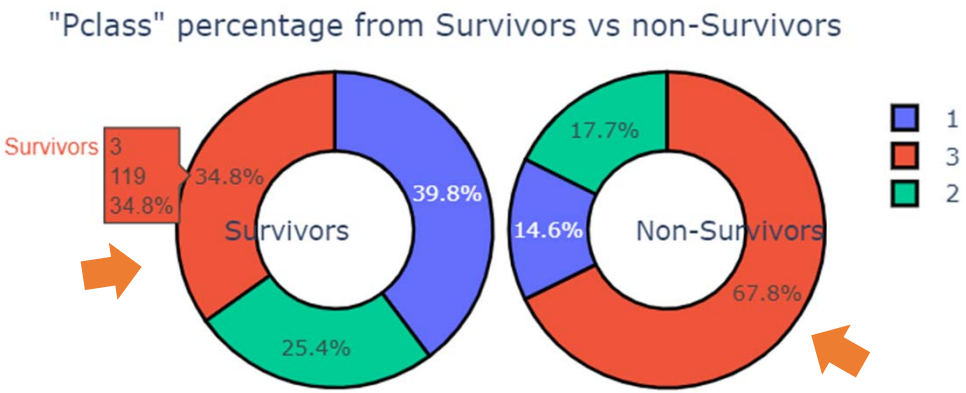
```
# Taking the count of each Pclass value inside the Survivors
df_survivors_pclass = df_survivors['Pclass'].value_counts()
df_survivors_pclass = pd.DataFrame({'Pclass':df_survivors_pclass.index,
                                    'count':df_survivors_pclass.values})

# Taking the count of each Pclass value inside the Survivors
df_nonsurvivors_pclass = df_nonsurvivors['Pclass'].value_counts()
df_nonsurvivors_pclass = pd.DataFrame({'Pclass':df_nonsurvivors_pclass.index,
                                       'count':df_nonsurvivors_pclass.values})

# Creating the plotting objects
pie_survivors_pclass = go.Pie(
    labels = df_survivors_pclass['Pclass'],
    values = df_survivors_pclass['count'],
    domain=dict(x=[0, 0.5]),
    name='Survivors',
    hole = 0.5,
    marker = dict(colors=['#636EFA', '#EF553B', '#00CC96'],
                  line=dict(color='#000000', width=2))
)

pie_nonsurvivors_pclass = go.Pie(
    labels = df_nonsurvivors_pclass['Pclass'],
    values = df_nonsurvivors_pclass['count'],
    domain=dict(x=[0.5, 1.0]),
    name='non-Survivors',
    hole = 0.5,
    marker = dict(colors=['#EF553B', '#00CC96', '#636EFA'],
                  line=dict(color='#000000', width=2))
)
```

```
data = [pie_survivors_pclass, pie_nonsurvivors_pclass]

# Plot's Layout (background color, title, annotations, etc.)
layout = go.Layout(
    paper_bgcolor='rgba(0,0,0,0)',
    plot_bgcolor='rgba(0,0,0,0)',
    title='"Pclass" percentage from Survivors vs non-Survivors',
    annotations=[dict(text='Survivors', x=0.18, y=0.5,
                      font_size=15, showarrow=False),
                 dict(text='Non-Survivors', x=0.85, y=0.5,
                      font_size=15, showarrow=False)]
)

fig = go.Figure(data=data, layout=layout)
fig.show()
```



Section 1 : Data Exploration

- "Pclass" percentage from Survivors vs non-Survivors

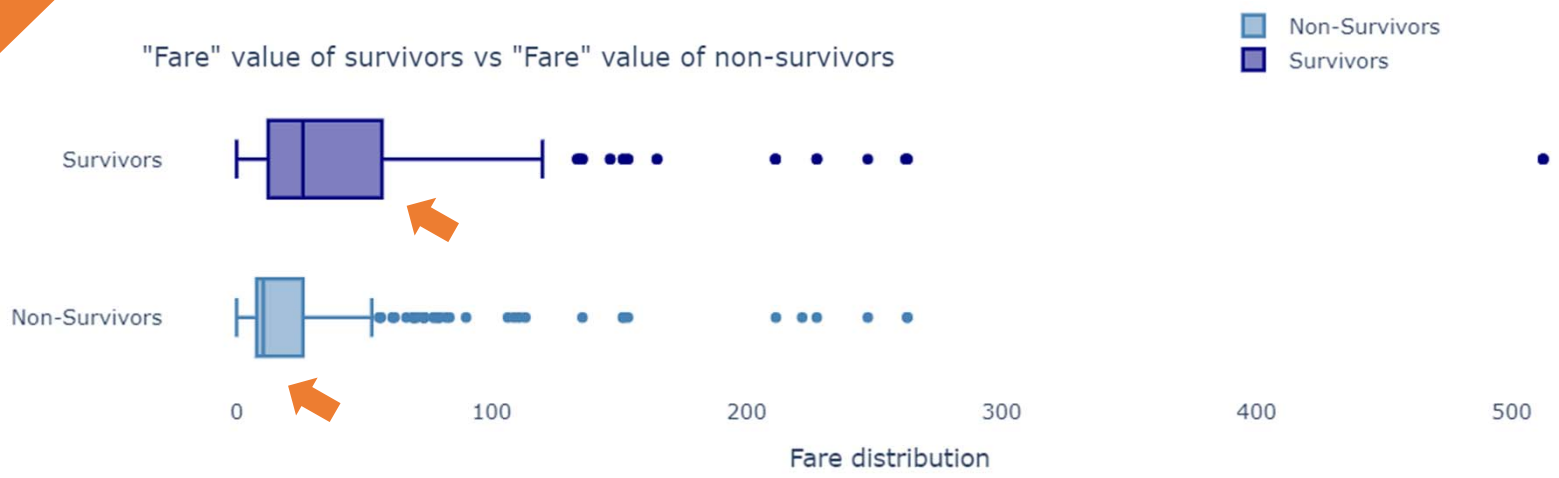


- From the pie chart showed above, we can notice a peculiar behavior: when looking at passengers that didn't survive, ~68% of them were at "Pclass" 3. When looking at passengers that survived, only ~35% of them were at "Pclass" 3.
- At the same point of view, when looking at passengers that survived, ~40% of them were at "Pclass" 1. At the non-survivors, only 14.6% of them were at "Pclass" 1.
- It seems that there is some kind of relation between "P-class" and the fact of a passenger surviving the accident or not.
- Let's get into more detail.

- "Fare" value of survivors vs "Fare" value of non-survivors

```
# Checking out the differences between Fare distribution
# for survivors and non-survivors
fare_survivors_box = go.Box(
    x=df_survivors['Fare'],
    name='Survivors',
    marker=dict(color='navy')
)

fare_nonsurvivors_box = go.Box(
    x=df_nonsurvivors['Fare'],
    name='Non-Survivors',
    marker=dict(color='steelblue')
)

data = [fare_nonsurvivors_box, fare_survivors_box]

# Plot's Layout (background color, title, etc.)
layout = go.Layout(
    paper_bgcolor='rgba(0,0,0,0)',
    plot_bgcolor='rgba(0,0,0,0)',
    title='"Fare" value of survivors vs "Fare" value of non-survivors',
    barmode='stack',
    xaxis=dict(
        title='Fare distribution'
    )
)

fig = go.Figure(data=data, layout=layout)
fig.show()
```

Section 1 : Data Exploration



- Checking out the plots and hypothesis tests over fare distributions, comparing Survivors and non-Survivors, we can again observe that there is a statistically significant difference between the means of both groups.
- When checking out the boxplots, we can see that fare values of survivors are generally higher, when compared to fare values of non-survivors. This information is probably related to the "Pclass" percentages we have seen before on the pie plots.

```
# Third distribution for the hypothesis test - Fares of survivors
dist_c = df_survivors['Fare'].dropna()

# Fourth distribution for the hypothesis test - Fares of non-survivors
dist_d = df_nonsurvivors['Fare'].dropna()

# Z-test: Checking if the distribution means
# (fares of survivors vs fares of non-survivors) are statistically different
t_stat_3, p_value_3 = ztest(dist_c, dist_d)
print("----- Z Test Results -----")
print("T stat. = " + str(t_stat_3))
print("P value = " + str(p_value_3)) # P-value is less than 0.05

print("")
```

```
# T-test: Checking if the distribution means
# (fares of survivors vs fares of non-survivors) are statistically different
t_stat_4, p_value_4 = stats.ttest_ind(dist_c, dist_d)
print("----- T Test Results -----")
print("T stat. = " + str(t_stat_4))
print("P value = " + str(p_value_4)) # P-value is less than 0.05
```

```
----- Z Test Results -----
T stat. = 7.939191660871055
P value = 2.035031103573989e-15

----- T Test Results -----
T stat. = 7.939191660871055
P value = 6.120189341924198e-15
```

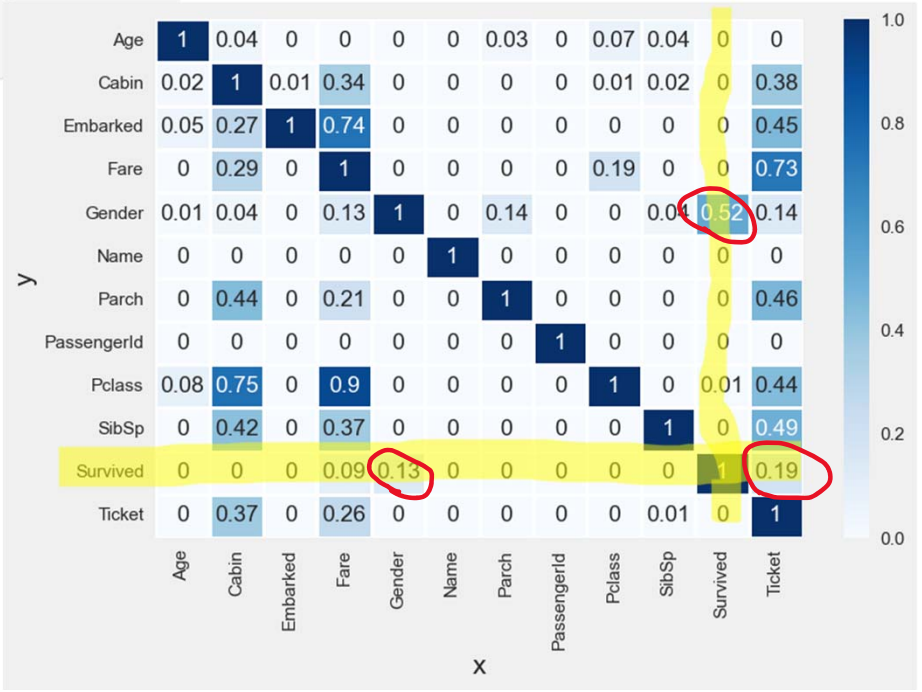
Section 1 : Data Exploration

PPS (Predictive Power Score)

- Basically, ***correlation matrices*** are able to identify linear relationships between variables.
- Because relationships in our data may sometimes be non-linear (most of the times, actually), we can use a **PPS (Predictive Power Score) matrix**, to figure out non-linear relations between columns.
- Introduce PPS : https://towardsdatascience.com/rip-correlation-introducing-the-predictive-power-score-3d90808b9598
- Python PPS implementation : https://github.com/8080labs/ppscore

```
matrix_df = pps.matrix(df)[['x', 'y', 'ppscore']].pivot(columns='x', index='y', values='ppscore')
matrix_df = matrix_df.apply(lambda x: round(x, 2)) # Rounding matrix_df's values to 0,XX

sns.heatmap(matrix_df, vmin=0, vmax=1, cmap="Blues", linewidths=0.75, annot=True)
```

```
<AxesSubplot:xlabel='x', ylabel='y'>
```



- Looking at this PPS matrix, we can see that the best univariate predictor of the Survived variable is the column Ticket, with 0.19 pps, followed by Gender, with 0.13 pps. That makes sense because women were prioritized during the rescue, and ticket is closely related to Pclass. The best univariate predictor of the Parch variable is the column Cabin, with 0.44 pps, and so on.

Section 2 - Supervised Learning: Classification

- Let's dive into the modeling part.
- First of all, we import the libraries we're going to use.

```
import re
import collections
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from category_encoders import TargetEncoder, LeaveOneOutEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import SimpleImputer, IterativeImputer
from sklearn.model_selection import StratifiedShuffleSplit, RandomizedSearchCV, train_test
_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler, PolynomialFeatures
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, log_l
oss, precision_recall_curve, average_precision_score, roc_curve, roc_auc_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from xgboost import XGBClassifier, plot_importance as plot_importance_xgb
from lightgbm import LGBMClassifier, plot_importance as plot_importance_lgbm
```

터미널 상 모듈 설치 필요

- pip install category_encoders lightgbm

Feature Engineering

- Create new features based on the original features of our dataset.

Titanic - Advanced Feature Engineering Tutorial :

https://www.kaggle.com/gunesevitan/titanic-advanced-feature-engineering-tutorial#2.-Feature-Engineering

```
# Creating a categorical variable for Ages
df['AgeCat'] = ''
df['AgeCat'].loc[(df['Age'] < 18)] = 'young'
df['AgeCat'].loc[(df['Age'] >= 18) & (df['Age'] < 56)] = 'mature'
df['AgeCat'].loc[(df['Age'] >= 56)] = 'senior'

# Creating a categorical variable for Family Sizes
df['FamilySize'] = ''
df['FamilySize'].loc[(df['SibSp'] <= 2)] = 'small'
df['FamilySize'].loc[(df['SibSp'] > 2) & (df['SibSp'] <= 5 )] = 'medium'
df['FamilySize'].loc[(df['SibSp'] > 5)] = 'large'

# Creating a categorical variable to tell if the passenger is alone
df['IsAlone'] = ''
df['IsAlone'].loc[((df['SibSp'] + df['Parch']) > 0)] = 'no'
df['IsAlone'].loc[((df['SibSp'] + df['Parch']) == 0)] = 'yes'

# Creating a categorical variable to tell if the passenger is a Young/Mature/Senior male
# or a Young/Mature/Senior female
df['GenderCat'] = ''
df['GenderCat'].loc[(df['Gender'] == 'male') & (df['Age'] <= 21)] = 'youngmale'
df['GenderCat'].loc[(df['Gender'] == 'male') & ((df['Age'] > 21) & (df['Age']) < 50)] = 'maturemale'
df['GenderCat'].loc[(df['Gender'] == 'male') & (df['Age'] > 50)] = 'seniormale'
df['GenderCat'].loc[(df['Gender'] == 'female') & (df['Age'] <= 21)] = 'youngfemale'
df['GenderCat'].loc[(df['Gender'] == 'female') & ((df['Age'] > 21) & (df['Age']) < 50)] = 'maturefemale'
df['GenderCat'].loc[(df['Gender'] == 'female') & (df['Age'] > 50)] = 'seniorfemale'
```

Section 2 - Supervised Learning: Classification

Feature Engineering

```
# Creating a categorical variable for the passenger's title
# Title is created by extracting the prefix before "Name" feature
# This title needs to be a feature because all female titles are grouped with each other
# Also, creating a column to tell if the passenger is married or not
# "Is_Married" is a binary feature based on the Mrs title. Mrs title has the highest survival rate among other female titles
df['Title'] = df['Name'].str.split(', ', expand=True)[1].str.split('.', expand=True)[0]
df['Is_Married'] = 0
df['Is_Married'].loc[df['Title'] == 'Mrs'] = 1
df['Title'] = df['Title'].replace(['Miss', 'Mrs','Ms', 'Mlle', 'Lady', 'Mme', 'the Countess', 'Dona'], 'Miss/Mrs/Ms')
df['Title'] = df['Title'].replace(['Dr', 'Col', 'Major', 'Jonkheer', 'Capt', 'Sir', 'Don', 'Rev'], 'Dr/Military/Noble/Clergy')

# Creating "Ticket Frequency" Feature
# There are too many unique Ticket values to analyze, so grouping them up by their frequencies makes things easier
df['Ticket_Frequency'] = df.groupby('Ticket')['Ticket'].transform('count')
df.head(10)
```

	PassengerId	Survived	Pclass	Name	Gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	AgeCat	FamilySize	IsAlone	GenderCat	Title	Is_Married	Ticket_Frequency
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S	mature	small	no	maturemale	Mr	0	1
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38.0	1	0	PC 17599	71.2833	C85	C	mature	small	no	maturefemale	Miss/Mrs/Ms	1	1
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	mature	small	yes	maturefemale	Miss/Mrs/Ms	0	1
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S	mature	small	no	maturefemale	Miss/Mrs/Ms	1	2
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S	mature	small	yes	maturemale	Mr	0	1

Feature Engineering

- After creating new features, we can drop useless columns that we won't use in the training process.

```
def get_feature_names(df):
    # Splitting the target
    target = df['Survived']

    # Dropping unused columns from the feature set
    df.drop(['PassengerId', 'Survived', 'Ticket', 'Name', 'Cabin'], axis=1, inplace=True)

    # Splitting categorical and numerical column dataframes
    categorical_df = df.select_dtypes(include=['object'])
    numeric_df = df.select_dtypes(exclude=['object'])

    # And then, storing the names of categorical and numerical columns.
    categorical_columns = list(categorical_df.columns)
    numeric_columns = list(numeric_df.columns)

    print("Categorical columns:\n", categorical_columns)
    print("\nNumeric columns:\n", numeric_columns)

    return target, categorical_columns, numeric_columns

target, categorical_columns, numeric_columns = get_feature_names(df)
```

```
Categorical columns:
 ['Sex', 'Embarked', 'AgeCat', 'FamilySize', 'IsAlone', 'SexCat', 'Title']

Numeric columns:
 ['Pclass', 'Age', 'SibSp', 'Parch', 'Fare', 'Is_Married', 'Ticket_Frequency']
```

Section 2 - Supervised Learning: Classification

Model training & Evaluation functions

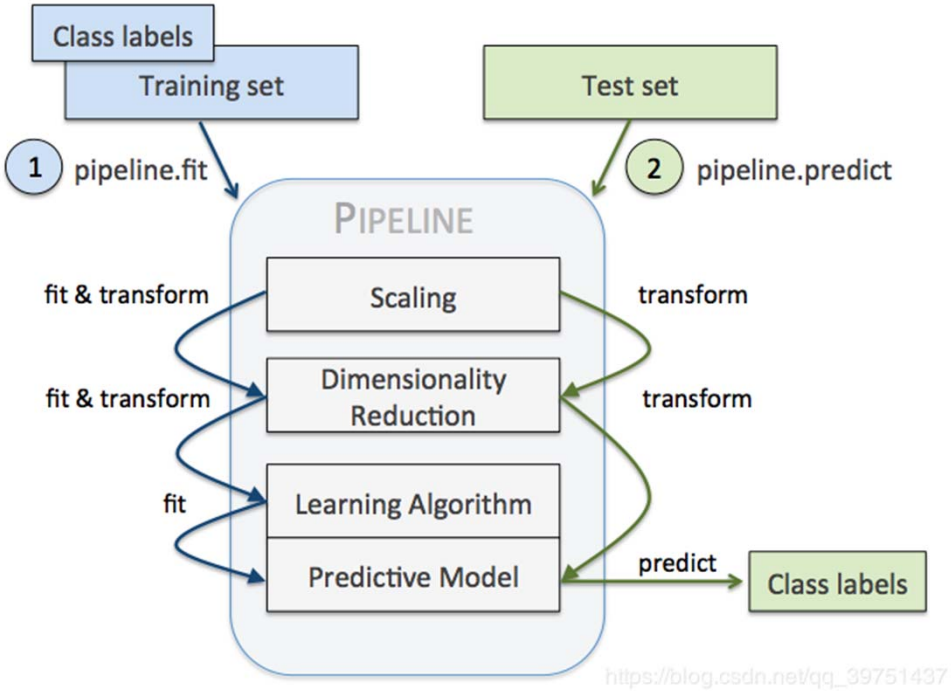
- After all the preprocessing, we are now ready to build and evaluate different Machine Learning models.
- First, let's create a function responsible for evaluating our classifiers on a test set we will create later.

```python
# Function responsible for checking our model's performance on the test data
def testSetResultsClassifier(best_model_pipeline, x_test, y_test):
    results = []

    predictions = best_model_pipeline.best_estimator_.predict(x_test)

    # Metrics applied on Probabilistic models and GLMs (predicted_proba for
    # probabilistic ones, decision_function for GLMs)
    predicted_probas_class1 = best_model_pipeline.best_estimator_.predict_proba(x_test)[:, 1]
    roc_auc = roc_auc_score(y_test, predicted_probas_class1)
    avg_precision = average_precision_score(y_test, predicted_probas_class1)

    # Universal metrics
    accuracy = accuracy_score(y_test, predictions)
    precision = precision_score(y_test, predictions)
    recall = recall_score(y_test, predictions)
    f1 = f1_score(y_test, predictions)

    results.append(accuracy)
    results.append(precision)
    results.append(recall)
    results.append(f1)
    results.append(avg_precision)
    results.append(roc_auc)

    print("\n\n#---------------- Test set results (Best Classifier) ----------------#\n")
    print("Accuracy:", round(results[0], 3))
    print("Precision:", round(results[1], 3))
    print("Recall:", round(results[2], 3))
    print("F1-Score:", round(results[3], 3))
    print("Average Precision (Precision/Recall AUC):", round(results[4], 3))
    print("ROC_AUC:", round(results[5], 3))

    return results
```

sklearn.pipeline.Pipeline

[출처] https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

pipeline



- Pipelines as a sequence of actions applied in data. Just like the image above, you can see that a full pipeline is made of several different small pipes.
- Take this to Data Science: imagine that each small pipe is a step in a modeling process. For example:

Step 1: fill null values from numerical columns.

Step 2: normalize numerical features, so they will be in the same scale.

Step 3: fill null values from categorical features.

Step 4: OneHotEncode categorical features.

Step 5: fit a Machine Learning model and evaluate it.

→ Instead of doing each one of these steps separately, we can create a *Pipeline object* that unites all of them, and then fit this object into our training data.

sklearn.pipeline.Pipeline

[출처] https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

pipeline

- <전처리를 위한 변환기 **transformer**> : 연속된 변환을 순차적으로 처리할 수 있는 기능을 제공하는 유용한 래퍼(Wrapper) 도구
- 여러 변환 단계를 정확한 순서대로 실행할 수 있게 함.
- Pipeline은 연속된 단계를 나타내는 이름/추정기 쌍의 목록을 입력으로 받음.
- 마지막 단계에서는 *transformer*와 *estimator*를 모두 사용할 수 있고, 그 외에는 모두 *transformer*여야 함.
- fit_transform() 메서드를 가지고 있어야 함.
- 파이프라인의 fit() 메서드를 호출하면 모든 변환기의 fit_transform() 메서드를 순서대로 호출하면서 한 단계의 출력이 다음 단계의 입력으로 전달됨.
- 마지막 단계에서는 fit() 메서드만 호출함.
- 파이프라인 객체는 마지막 추정기와 동일한 메서드를 제공합니다. 이 예에서는 마지막 추정기가 변환기 StandardScaler이므로 파이프라인이 데이터에 대해 모든 변환을 순서대로 적용하는 transform() 메서드를 가지고 있습니다 (또한 fit_transform() 메서드도 가지고 있습니다).

class sklearn.pipeline.**Pipeline**(*steps*, *, *memory=None*, *verbose=False*)

steps : *list*

List of (name, transform) tuples (implementing fit/transform) that are chained, in the order in which they are chained, with the last object an estimator.

See also:

`make_pipeline`

Convenience function for simplified pipeline construction.

Pipeline 예제 코드

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import Imputer, StandardScaler
from sklearn.preprocessing import OnehotEncoder, CategoricalEncoder
```

```
# 숫자형 변수를 전처리하는 Pipeline
num_pipeline = Pipeline([
    ('selector', DataFrameSelector(num_attr)),
    ('imputer', Imputer(strategy = 'median')),
    ('std_scaler', StandardScaler())
])
# 범주형 변수를 전처리하는 Pipeline
cat_pipeline = Pipeline([
    ('selector', DataFrameSelector(cat_attr)),
    ('cat_encoder', CategoricalEncoder(encoding = 'onehot-dense'))
])
```

sklearn.pipeline.Pipeline

[출처] https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

pipeline advantages

1) Production code gets much easier to implement

- When deploying a Machine Learning model into production, the main goal is to use it on data it hasn't seen before. To do that, the new data needs to be transformed the same way training data was. Instead of having several different functions for each one of the preprocessing tasks, you can use a single pipeline object to apply all of them sequentially. It means that, in 1 line of code, you can apply all needed transformations.

2) When combined with RandomSearchCV, it is possible to test several different pipeline options

- When training your models: "for this type of data, what works best? Filling missing values with the average or the median of a column? Should I use *MinMaxScaler* or *StandardScaler*? Apply dimensionality reduction? Create more features using, for example, PolynomialFeatures?"
- Using Pipelines and hyperparameter search functions (like RandomSearchCV), you can search through entire sets of data pipelines, models and parameters automatically, saving up effort invested by you in the search for optimal feature engineering methods and models/hyperparameters.
- Suppose we have 4 different pipelines:

- Pipeline 1: fill missing values from numeric features by imputing the mean of each column → apply MinMaxScaler → apply OneHotEncoder to categorical features → fits the data into a KNN Classifier with n_neighbors = 15.
- Pipeline 2: fill missing values from numeric features by imputing the mean of each column → apply StandardScaler → apply OneHotEncoder to categorical features → fits the data into a KNN Classifier with n_neighbors = 30.
- Pipeline 3: fill missing values from numeric features by imputing the median of each column → apply MinMaxScaler → apply OneHotEncoder to categorical features → fits the data into a Random Forest Classifier with n_estimators = 100.
- Pipeline 4: fill missing values from numeric features by imputing the median of each column → apply StandardScaler → apply OneHotEncoder to categorical features → fits the data into a Random Forest Classifier with n_estimators = 150.

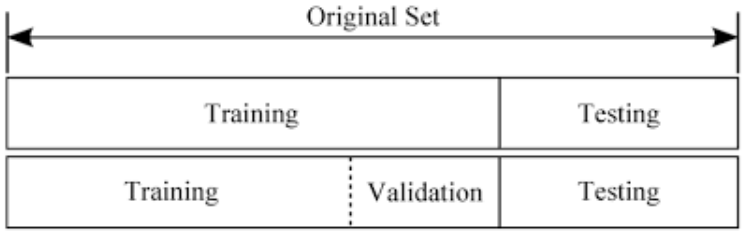
sklearn.pipeline.Pipeline

[출처] https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

pipeline advantages

3) No information leakage when Cross-Validating (교차검증)

- This one is a bit trickier, specially for beginners.
- Basically, when cross-validating, data should be transformed inside each CV step, not before. When doing cross validation after transforming the training set (with a StandardScaler, for example), information from it is leaked to the validation set. This may lead to biased/unoptimal results.
- The right way to do that is to normalize data **inside** cross-validation. That means, for each CV step, a scaler is fitted **only on the training set**. Then, this scaler transforms **the validation set**, and the model is evaluated. This way, no information from the training set is leaked to the validation set. When using pipelines inside RandomSearchCV (or GridSearchCV), this problem is taken care of.



- This is a key concept in Machine Learning, so it's important to understand why.

pipeline construction

- Now, we are going to create our Pipeline, fitting several different data preprocessing and modeling techniques inside a **RandomSearchCV**, to check which group of techniques has better performance.
- Building a Pipeline inside RandomSearchCV, responsible for finding the best model and it's parameters

```
import random

def defineBestModelPipeline(df, target, numeric_columns, categorical_columns):

    # Splitting original data into Train and Test BEFORE applying transformations
    # Later in RandomSearchCV, x_train will be splitted into train/val sets
    # The transformations are going to be fitted specifically on the train set,
    # and then applied to both train/test sets. This way, information leakage is avoided!
    x_train, x_test, y_train, y_test = train_test_split(df, target, test_size=0.10, random_state=42)


    # 1st -> #### ----------- Numeric Transformers ----------- ####

    # Here, we are creating different several different data transformation pipelines
    # to be applied in our numeric features
    numeric_transformer_1 = Pipeline(steps=[('imp', IterativeImputer(max_iter=30, random_state=42)),
                                            ('scaler', MinMaxScaler())])

    numeric_transformer_2 = Pipeline(steps=[('imp', SimpleImputer(strategy='mean')), # or strategy='median'
                                            ('scaler', StandardScaler())])
```

`sklearn.model_selection.train_test_split(*arrays, test_size=None, train_size=None, random_state=None, shuffle=True, stratify=None)` [source]

Split arrays or matrices into random train and test subsets

Quick utility that wraps input validation and `next(ShuffleSplit().split(X, y))` and application to input data into a single call for splitting (and optionally subsampling) data in a oneliner.

random_state : Controls the shuffling applied to the data before applying the split.

Missing Data Imputation

class sklearn.impute.**IterativeImputer**(*estimator=None, *, missing_values=nan, sample_posterior=False, max_iter=10, tol=0.001, n_nearest_features=None, initial_strategy='mean', imputation_order='ascending', skip_complete=False, min_value=- inf, max_value=inf, verbose=0, random_state=None, add_indicator=False*) [source]

Multivariate imputer that estimates each feature from all the others.

A strategy for imputing missing values by modeling each feature with missing values as a function of other features in a round-robin fashion.

class sklearn.impute.**SimpleImputer**(**, missing_values=nan, strategy='mean', fill_value=None, verbose=0, copy=True, add_indicator=False*) [source]

Imputation transformer for completing missing values.

데이터 스케일링 :: 표준화(Standardization)

class sklearn.preprocessing.**MinMaxScaler**(*feature_range=0, 1, *, copy=True, clip=False*) [source]

Transform features by scaling each feature to a given range.

This estimator scales and translates each feature individually such that it is in the given range on the training set, e.g. between zero and one.

The transformation is given by:

```
X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
X_scaled = X_std * (max - min) + min
```

where min, max = feature_range.

This transformation is often used as an alternative to zero mean, unit variance scaling.

class sklearn.preprocessing.**StandardScaler**(**, copy=True, with_mean=True, with_std=True*) [source]

Standardize features by removing the mean and scaling to unit variance

1. StandardScaler

- 평균 = 0 / 표준편차 = 1
- 표준화 Standardization

2. MinMaxScaler

- 최대값 = 1 / 최소값 = 0
- 최소-최대 정규화 Min-Max Normalization
- 이상치에 취약하다.

Section 2 - Supervised Learning: Classification

pipeline construction

```
# 2nd -> #### ---------- Categorical Transformers ---------- ####

# We are going to encode categorical features using LeaveOneOutEncoder and TargetEncoder.
# Note: TargetEncoder uses the mean target values (probabilities for classification and
# continuous values for regression) of each category inside a column to encode them.

# LeaveOneOutEncoder is an alternative to TargetEncoder. It implements an improvement on TargetEncoder's
# "overfitting behaviour", by leaving 1 of the observations out of the mean calculation for a specific category.
# Read about it here: https://towardsdatascience.com/stop-one-hot-encoding-your-categorical-variables-bbb0fba89809
# Documentation: https://contrib.scikit-learn.org/category_encoders/leaveoneout.html

categorical_transformer_1 = Pipeline(steps=[('frequent', SimpleImputer(strategy='most_frequent')),
                                            ('leaveoneout', LeaveOneOutEncoder(sigma=0.1))])

categorical_transformer_2 = Pipeline(steps=[('frequent', SimpleImputer(strategy='most_frequent')),
                                            ('targetencoder', TargetEncoder(min_samples_leaf=3, smoothing=2))])
```

class **category_encoders.target_encoder.TargetEncoder**(*verbose=0, cols=None, drop_invariant=False, return_df=True, handle_missing='value', handle_unknown='value', min_samples_leaf=1, smoothing=1.0)* [source]

Target encoding for categorical features.

Supported targets: binomial and continuous. For polynomial target support, see PolynomialWrapper.

For the case of categorical target: features are replaced with a blend of posterior probability of the target given particular categorical value and the prior probability of the target over all the training data.

For the case of continuous target: features are replaced with a blend of the expected value of the target given particular categorical value and the expected value of the target over all the training data.

class **category_encoders.leave_one_out.LeaveOneOutEncoder**(*verbose=0, cols=None, drop_invariant=False, return_df=True, handle_unknown='value', handle_missing='value', random_state=None, sigma=None)* [source]

Leave one out coding for categorical features.

This is very similar to target encoding but excludes the current row's target when calculating the mean target for a level to reduce the effect of outliers.

Section 2 - Supervised Learning: Classification

pipeline construction

```
# 3rd -> #### ----------- Combining both numerical and categorical data pipelines ----------- ####

# Here, we are creating different ColumnTransformers, each one with a different numerical/categotical transformation
data_transformations_1 = ColumnTransformer(transformers=[('num', numeric_transformer_1, numeric_columns),
                                                         ('cat', categorical_transformer_1, categorical_columns)])

data_transformations_2 = ColumnTransformer(transformers=[('num', numeric_transformer_1, numeric_columns),
                                                         ('cat', categorical_transformer_2, categorical_columns)])

data_transformations_3 = ColumnTransformer(transformers=[('num', numeric_transformer_2, numeric_columns),
                                                         ('cat', categorical_transformer_1, categorical_columns)])

data_transformations_4 = ColumnTransformer(transformers=[('num', numeric_transformer_2, numeric_columns),
                                                         ('cat', categorical_transformer_2, categorical_columns)])
```

```
# 4th -> #### ----------- Testing different data transformation steps and models inside RandomSearchCV ----------- ####

# Finally, we are going to apply these different data transformations to RandomSearchCV,
# trying to find the best imputing strategy, the best feature transformation strategy and the best model with it's respective parameters.
# Below, we just need to initialize a Pipeline object with any transformations we want, on each of the steps.

pipe = Pipeline(steps=[('data_transformations', data_transformations_1), # Initializing data transformation step by choosing any of the above
                       ('clf', SVC())]) # Initializing modeling step with any model object
                       #memory='cache_folder') -> Used to optimize memory when needed
```

pipeline construction

```
# Now, we define the hyperparameter grid that will be used by RandomSearchCV. It will randomly chose options
# for each step inside the dictionaries ('data transformations', 'feature_selection', 'clf' and clf's parameters).
# Then, for each chosen option, it will apply the transformations, train the chosen model and evaluate
# it in the validation fold of the cross validator we define. In the end of it's iterations, RandomSearchCV will return some metrics,
# such as the best pipeline, model results for all iterations and more.

params_grid = [
                {'data_transformations': [data_transformations_1, data_transformations_2,
                                          data_transformations_3, data_transformations_4],
                 'clf': [RandomForestClassifier()],
                 'clf__n_estimators': [int(x) for x in np.linspace(5, 30, num=15)],
                 'clf__max_features': [None, "sqrt", "log2"],
                 'clf__max_depth': [int(x) for x in np.linspace(3, 10, num=5)],
                 'clf__random_state': [int(x) for x in np.linspace(1, 49, num=30)]},

                {'data_transformations': [data_transformations_1, data_transformations_2,
                                          data_transformations_3, data_transformations_4],
                 'clf': [LGBMClassifier()],
                 'clf__n_estimators': [int(x) for x in np.linspace(3, 20, num=10)],
                 'clf__max_depth': [int(x) for x in np.linspace(2, 8, num=6)],
                 'clf__learning_rate': np.linspace(0.1, 0.7)},

                {'data_transformations': [data_transformations_1, data_transformations_2,
                                          data_transformations_3, data_transformations_4],
                 'clf': [XGBClassifier()],
                 'clf__n_estimators': [int(x) for x in np.linspace(3, 15, num=10)],
                 'clf__eta': np.linspace(0.1, 0.9),
                 'clf__max_depth': [int(x) for x in np.linspace(2, 7, num=5)],
                 'clf__gamma': np.linspace(0.1, 1),
                 'clf__lambda': np.linspace(0.1, 1)},
            ]
```

pipeline construction

```
# Now, we fit a RandomSearchCV to search over the grid of parameters defined above
metrics = ['accuracy', 'precision', 'recall', 'f1', 'average_precision', 'roc_auc']

# Creating our cross validator object with StratifiedShuffleSplit (5 folds).
# Stratification assures that we split the data such that the proportions
# between classes are the same in each fold as they are in the whole dataset
cross_validator = StratifiedShuffleSplit(n_splits=5, train_size=0.8, test_size=0.2, random_state=7)

# Creating the randomized search cv object and fitting it
best_model_pipeline = RandomizedSearchCV(estimator=pipe, param_distributions=params_grid,
                                         n_iter=50, scoring=metrics, refit='accuracy',
                                         n_jobs=-1, cv=cross_validator, random_state=21,
                                         error_score='raise', return_train_score=False)

best_model_pipeline.fit(x_train, y_train)

# At last, we check the final results
print("\n\n#--------- Best Data Pipeline found in RandomSearchCV ---------#\n\n", best_model_pipeline.best_estimator_[0])
print("\n\n#--------- Best Classifier found in RandomSearchCV ---------#\n\n", best_model_pipeline.best_estimator_[1])
print("\n\n#--------- Best Estimator's average Accuracy Score on CV (validation set) ---------#\n\n", best_model_pipeline.best_score_)

return x_train, x_test, y_train, y_test, best_model_pipeline
```

testSet results classifier

```
# Calling the function above, returing train/test data and best model's pipeline
x_train, x_test, y_train, y_test, best_model_pipeline = defineBestModelPipeline(df, target, numeric_columns, categorical_columns)

# Checking best model's performance on test data
test_set_results = testSetResultsClassifier(best_model_pipeline, x_test, y_test)
```

```
#------- Best Data Pipeline found in RandomSearchCV -------#

 ColumnTransformer(transformers=[('num',
                                  Pipeline(steps=[('imp', SimpleImputer()),
                                                  ('scaler', StandardScaler())]),
                                  ['Pclass', 'Age', 'SibSp', 'Parch', 'Fare',
                                   'Is_Married', 'Ticket_Frequency']),
                                 ('cat',
                                  Pipeline(steps=[('frequent',
                                                   SimpleImputer(strategy='most_frequent')),
                                                  ('targetencoder',
                                                   TargetEncoder(min_samples_leaf=3,
                                                                 smoothing=2.0))]),
                                  ['Sex', 'Embarked', 'AgeCat', 'FamilySize',
                                   'IsAlone', 'SexCat', 'Title'])])
```

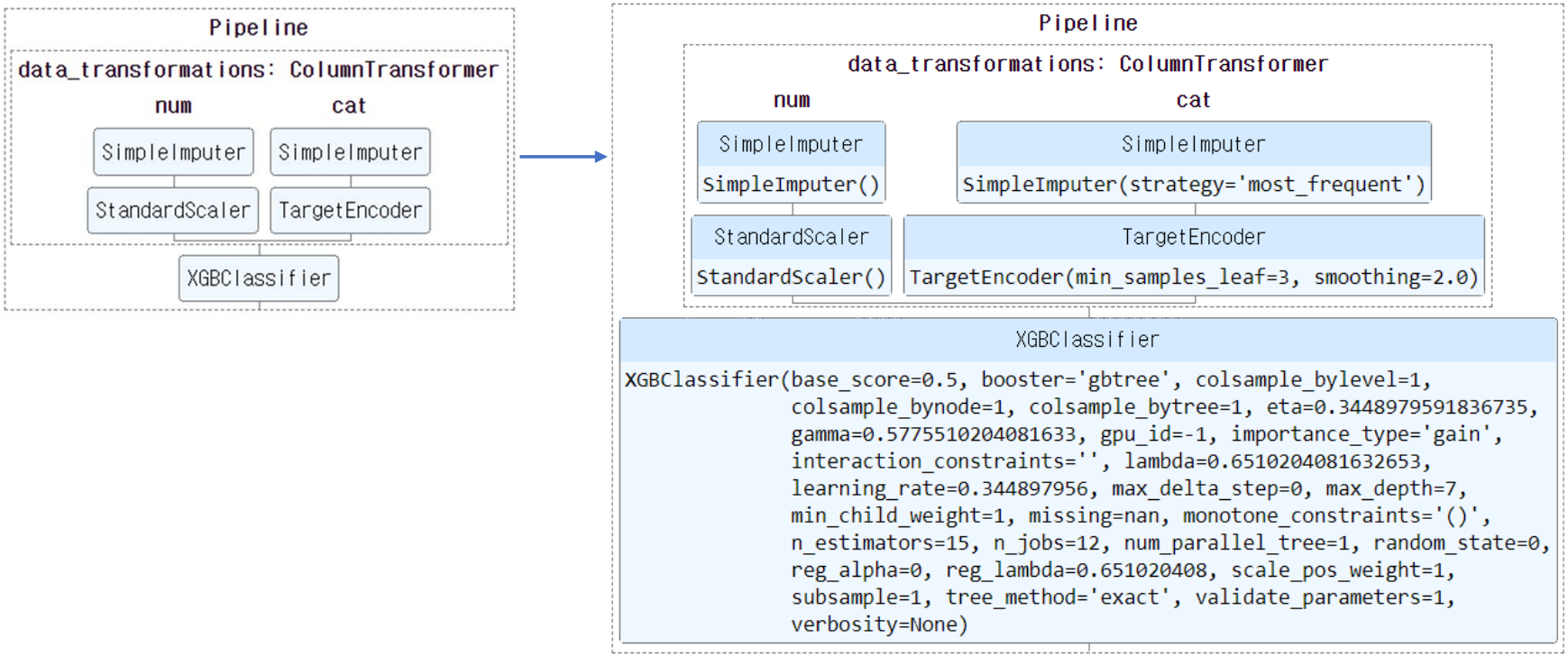
```
#------- Best Estimator's average Accuracy Score on CV (validation set)-------#

 0.8298136645962734
```

```
#---------------- Test set results (Best Classifier) ----------------#

Accuracy: 0.856
Precision: 0.795
Recall: 0.861
F1-Score: 0.827
Average Precision (Precision/Recall AUC): 0.89
ROC_AUC: 0.917
```

```
#------- Best Classifier found in RandomSearchCV -------#

 XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, eta=0.3448979591836735,
              gamma=0.5775510204081633, gpu_id=-1, importance_type='gain',
              interaction_constraints='', lambda=0.6510204081632653,
              learning_rate=0.344897956, max_delta_step=0, max_depth=7,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=15, n_jobs=12, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=0.651020408, scale_pos_weight=1,
              subsample=1, tree_method='exact', validate_parameters=1,
              verbosity=None)
```

- Visual representation of the best pipeline found by RandomSearchCV

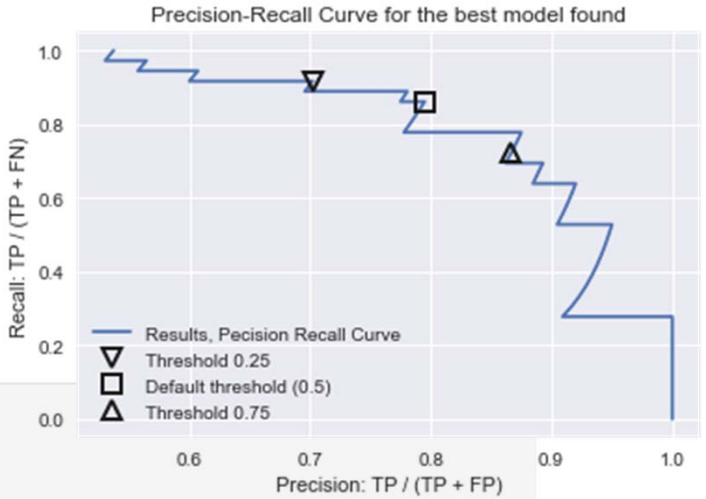
```
from sklearn import set_config
from sklearn.utils import estimator_html_repr

# Set config to 'diagram' so we can visualize pipelines/composite estimators
set_config(display='diagram')

# Visualization of the best estimator found by RandomSearchCV
best_model_pipeline.best_estimator_
```



Section 2 - Supervised Learning: Classification

Precision-Recall and ROC Curves

- Let's take a look at the Precision/Recall and ROC Curves of the best model in our separate test dataset.




```
# Transforming the test data
x_test = best_model_pipeline.best_estimator_[0].transform(x_test)

# Calculating precision/recall threshold values for Probabilistic models
precision, recall, thresholds_prc = precision_recall_curve(y_test, best_model_pipeline.best_estimator_[1].predict_proba(x_test)[:, 1])
closest_to_025_prc = np.argmin(np.abs(thresholds_prc - 0.25))    # Getting information about the points in the graph that
closest_to_default_prc = np.argmin(np.abs(thresholds_prc - 0.5)) # are closer to the default threshold for predict_proba (0.5),
closest_to_075_prc = np.argmin(np.abs(thresholds_prc - 0.75))    # threshold 0.25 and threshold 0.75.

# Plotting the curve
plt.plot(precision, recall, label="Results, Pecision Recall Curve")
plt.plot(precision[closest_to_025_prc], recall[closest_to_025_prc], 'v', c='k', # Plotting the marker for threshold 0.25
         markersize=10, label="Threshold 0.25", fillstyle="none", mew=2)
plt.plot(precision[closest_to_default_prc], recall[closest_to_default_prc], 's', c='k', # Plotting the marker for threshold 0.5 (default)
         markersize=10, label="Default threshold (0.5)", fillstyle="none", mew=2)
plt.plot(precision[closest_to_075_prc], recall[closest_to_075_prc], '^', c='k', # Plotting the marker for threshold 0.75
         markersize=10, label="Threshold 0.75", fillstyle="none", mew=2)
plt.title("Precision-Recall Curve for the best model found")
plt.xlabel("Precision: TP / (TP + FP)")
plt.ylabel("Recall: TP / (TP + FN)")
plt.legend(loc="best")
```

Precision-Recall and ROC Curves

- After going through all steps in RandomSearchCV, we can check the results from it's steps using the "cvresults" atrribute.

```
# Visualizing all results and metrics, from all models, obtained by the RandomSearchCV steps
df_results = pd.DataFrame(best_model_pipeline.cv_results_)

display(df_results)
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_data_transformations	param_clf__n_estimators	param_clf__max_depth	param_clf__lambda	param_clf__gar
0	0.643256	0.154214	0.071609	0.009407	ColumnTransformer(transformers= [('num',\n ...	3	7	0.65102	0.83
1	0.890196	0.216905	0.096143	0.029791	ColumnTransformer(transformers= [('num',\n ...	15	3	0.595918	0.55
2	0.975502	0.305539	0.085970	0.018023	ColumnTransformer(transformers= [('num',\n ...	13	5	0.412245	0.39
3	0.386167	0.110670	0.091356	0.009348	ColumnTransformer(transformers= [('num',\n ...	4	5	0.577551	0.32
4	0.711098	0.047446	0.095545	0.012747	ColumnTransformer(transformers= [('num',\n ...	13	5	0.1	0.98
...	...	...	...	...	...	...	...	...	
45	0.567184	0.164900	0.097539	0.027067	ColumnTransformer(transformers= [('num',\n ...	12	7	0.540816	0.33
46	1.233428	0.062115	0.101529	0.007203	ColumnTransformer(transformers= [('num',\n ...	12	5	0.779592	0.13
47	0.324242	0.077734	0.084379	0.006688	ColumnTransformer(transformers= [('num',\n ...	5	3	0.246939	0.33
48	0.338117	0.056927	0.088763	0.013484	ColumnTransformer(transformers= [('num',\n ...	3	3	0.559184	0.7
49	0.305583	0.017736	0.095943	0.015761	ColumnTransformer(transformers= [('num',\n ...	3	4	0.963265	0.54

50 rows × 60 columns

Section 2 - Supervised Learning: Classification

Precision-Recall and ROC Curves

```
# Visualizing all results and metrics obtained only by the best classifier, considering Accuracy score
display(df_results[df_results['rank_test_accuracy'] == 1])
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_data_transformations	param_clf__n_estimators	param_clf__max_depth	param_clf__lambda
44	0.861866	0.089517	0.082879	0.014037	ColumnTransformer(transformers= [('num',\n ...	15	7	0.65102

```
# Visualizing all results and metrics obtained only by the best classifier, considering ROC_AUC score
display(df_results[df_results['rank_test_roc_auc'] == 1])
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_data_transformations	param_clf__n_estimators	param_clf__max_depth	param_clf__lambda	param_clf__gam
1	0.890196	0.216905	0.096143	0.029791	ColumnTransformer(transformers= [('num',\n ...	15	3	0.595918	0.559

Plotting Feature Importances

- Here, we access the categorical feature names generated by the Categorical Encoder, and then concatenate them with the numerical feature names, in the same order our pipeline is applying data transformations.
- Since we're applying transformations on numeric columns first, and then on categorical ones, our feature names list will be:

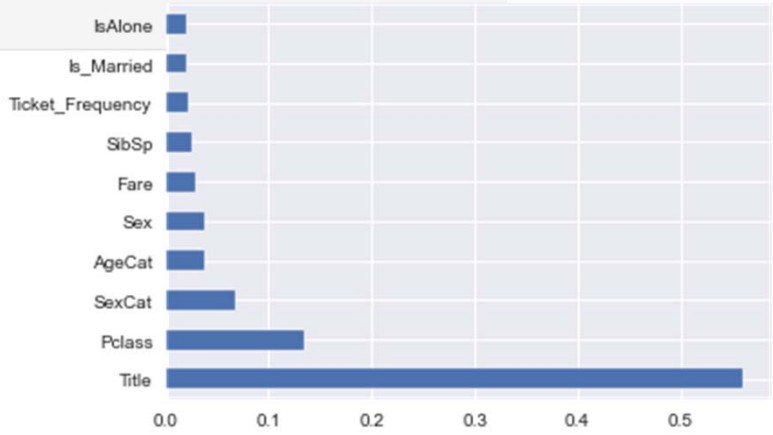
```
feature_names_in_order = numeric_columns + categorical_columns
print(feature_names_in_order)
```

```
['Pclass', 'Age', 'SibSp', 'Parch', 'Fare', 'Is_Married', 'Ticket_Frequency', 'Sex', 'Embarked',
'AgeCat', 'FamilySize', 'IsAlone', 'SexCat', 'Title']
```

- Feature Importances for Tree-Based Models, top 10 values:

```
##### Plotting Feature Importances for Random Forests & XGBoost #####
feat_importances = pd.Series(best_model_pipeline.best_estimator_.named_steps['clf'].feature_importances_,
                             index=feature_names_in_order)
feat_importances.nlargest(10).plot(kind='barh')
```

```
<AxesSubplot:>
```



Predictions

- Now that we have tried different preprocessing and modeling techniques, resulting in a final best pipeline, let's use it to predict the test data provided by kaggle. ***Remember***: All transformations that were done in the training dataset must be done in the test set.

```
# Importing data and displaying some rows
df_test = pd.read_csv("data/input/test.csv")

# Creating a categorical variable for Ages
df_test['AgeCat'] = ''
df_test['AgeCat'].loc[(df_test['Age'] < 18)] = 'young'
df_test['AgeCat'].loc[(df_test['Age'] >= 18) & (df_test['Age'] < 56)] = 'mature'
df_test['AgeCat'].loc[(df_test['Age'] >= 56)] = 'senior'

# Creating a categorical variable for Family Sizes
df_test['FamilySize'] = ''
df_test['FamilySize'].loc[(df_test['SibSp'] <= 2)] = 'small'
df_test['FamilySize'].loc[(df_test['SibSp'] > 2) & (df_test['SibSp'] <= 5 )] = 'medium'
df_test['FamilySize'].loc[(df_test['SibSp'] > 5)] = 'large'

# Creating a categorical variable to tell if the passenger is alone
df_test['IsAlone'] = ''
df_test['IsAlone'].loc[((df_test['SibSp'] + df_test['Parch']) > 0)] = 'no'
df_test['IsAlone'].loc[((df_test['SibSp'] + df_test['Parch']) == 0)] = 'yes'

# Creating a categorical variable to tell if the passenger is a Young/Mature/Senior male or a Young/Mature/Senior female
df_test['GenderCat'] = ''
df_test['GenderCat'].loc[(df_test['Gender'] == 'male') & (df_test['Age'] <= 21)] = 'youngmale'
df_test['GenderCat'].loc[(df_test['Gender'] == 'male') & ((df_test['Age'] > 21) & (df_test['Age']) < 50)] = 'maturemale'
df_test['GenderCat'].loc[(df_test['Gender'] == 'male') & (df_test['Age'] > 50)] = 'seniormale'
df_test['GenderCat'].loc[(df_test['Gender'] == 'female') & (df_test['Age'] <= 21)] = 'youngfemale'
df_test['GenderCat'].loc[(df_test['Gender'] == 'female') & ((df_test['Age'] > 21) & (df_test['Age']) < 50)] = 'maturefemale'
df_test['GenderCat'].loc[(df_test['Gender'] == 'female') & (df_test['Age'] > 50)] = 'seniorfemale'
```

Section 2 - Supervised Learning: Classification

Predictions

```
# Creating a categorical variable for the passenger's title
# Title is created by extracting the prefix before "Name" feature
# This title needs to be a feature because all female titles are grouped with each other
# Also, creating a column to tell if the passenger is married or not
# "Is_Married" is a binary feature based on the Mrs title. Mrs title has the highest survival rate among other female titles
df_test['Title'] = df_test['Name'].str.split(', ', expand=True)[1].str.split('.', expand=True)[0]
df_test['Is_Married'] = 0
df_test['Is_Married'].loc[df['Title'] == 'Mrs'] = 1
df_test['Title'] = df_test['Title'].replace(['Miss', 'Mrs','Ms', 'Mlle', 'Lady', 'Mme', 'the Countess', 'Dona'], 'Miss/Mrs/Ms')
df_test['Title'] = df_test['Title'].replace(['Dr', 'Col', 'Major', 'Jonkheer', 'Capt', 'Sir', 'Don', 'Rev'], 'Dr/Military/Noble/Clergy')

# Creating "Ticket Frequency" Feature
# There are too many unique Ticket values to analyze, so grouping them up by their frequencies makes things easier
df_test['Ticket_Frequency'] = df_test.groupby('Ticket')['Ticket'].transform('count')

# Dropping unnecessary columns
df_test.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1, inplace=True)
```

Predictions

```
# Applying best_model_pipeline
# Step 1 -> Transforming data the same way we did in the training set;
# Step 2 -> making predictions using the best model obtained by RandomSearchCV.
test_predictions = best_model_pipeline.best_estimator_.predict(df_test)

print(test_predictions)
```

```
[0 0 0 0 0 0 0 1 0 0 0 1 0 1 1 0 0 0 0 1 1 1 1 0 1 0 0 0 0 0 1 0 1 0 0
 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 1 1 0 0 0 0 0 1 0 0 0 1 1 1 1 0 0 1 1 0 0 0
 1 0 0 1 0 1 1 0 0 0 0 0 1 0 1 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0
 1 1 1 1 0 0 1 0 1 1 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0
 0 0 1 0 0 0 0 0 1 0 1 0 1 1 1 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 1 1 0 1 1 0 1
 0 1 0 0 0 0 0 1 0 1 0 1 0 0 0 1 1 1 1 0 0 1 0 1 0 0 0 0 1 0 0 1 0 1 0 1 0
 1 0 1 0 0 1 0 0 0 1 0 0 1 0 0 0 1 1 1 1 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 1
 0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 1 0 0 0 0
 1 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 1 0 1 0 0 0 1 0 0
 1 0 0 0 0 0 1 0 0 0 1 1 1 0 1 0 1 1 0 0 0 1 0 1 0 0 0 0 1 1 0 1 0 0 0 1 0
 0 1 0 0 1 1 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 1 1 0 0 1 0 1 0 0 1 0 1 0 0 0 0
 0 1 1 1 1 0 0 1 0 0 1]
```

```
# Generating predictions file that is going to be submitted to the competition
df_submission = pd.read_csv("data/input/test.csv")

# Adding a column with predicted values
df_submission['Survived'] = test_predictions

# Selecting only needed columns
df_submission.drop(df_submission.columns.difference(['PassengerId', 'Survived']), axis=1, inplace=True)

df_submission.head(10)
```

	PassengerId	Survived
0	892	0
1	893	0
2	894	0
3	895	0
4	896	0
5	897	0
6	898	0
7	899	0
8	900	1
9	901	0