Fifth Practice ML

In this practice, we will learn about **Confusion Matrix** and **KPI**'s.
We will show how the **Threshold** affects the **F1** score, plot the **ROC** (Receiver Operating Characteristic) curve and calculate the **AUC** (Area Under the Curve) score.

Downloads, Imports, and Definitions ¶

We update packages that their Colab version is too old.

In [ ]:

```
# update plotly and pandas_profiling version
!pip install --upgrade plotly
!pip install --upgrade pandas_profiling==2.9.0
```

```
Requirement already up-to-date: plotly in /usr/local/lib/python3.6/dist-pa
ckages (4.13.0)
Requirement already satisfied, skipping upgrade: six in /usr/local/lib/pyt
hon3.6/dist-packages (from plotly) (1.15.0)
Requirement already satisfied, skipping upgrade: retrying>=1.3.3 in /usr/l
ocal/lib/python3.6/dist-packages (from plotly) (1.3.3)
Requirement already up-to-date: pandas_profiling in /usr/local/lib/python
3.6/dist-packages (2.9.0)
Requirement already satisfied, skipping upgrade: scipy>=1.4.1 in /usr/loca
l/lib/python3.6/dist-packages (from pandas_profiling) (1.4.1)
Requirement already satisfied, skipping upgrade: tqdm>=4.43.0 in /usr/loca
l/lib/python3.6/dist-packages (from pandas_profiling) (4.54.0)
Requirement already satisfied, skipping upgrade: visions[type_image_path]=
=0.5.0 in /usr/local/lib/python3.6/dist-packages (from pandas_profiling)
(0.5.0)
Requirement already satisfied, skipping upgrade: missingno>=0.4.2 in /usr/
local/lib/python3.6/dist-packages (from pandas_profiling) (0.4.2)
Requirement already satisfied, skipping upgrade: tangled-up-in-unicode>=0.
0.6 in /usr/local/lib/python3.6/dist-packages (from pandas_profiling) (0.
0.6)
Requirement already satisfied, skipping upgrade: ipywidgets>=7.5.1 in /us
r/local/lib/python3.6/dist-packages (from pandas_profiling) (7.5.1)
Requirement already satisfied, skipping upgrade: htmlmin>=0.1.12 in /usr/l
ocal/lib/python3.6/dist-packages (from pandas_profiling) (0.1.12)
Requirement already satisfied, skipping upgrade: attrs>=19.3.0 in /usr/loc
al/lib/python3.6/dist-packages (from pandas_profiling) (20.3.0)
Requirement already satisfied, skipping upgrade: seaborn>=0.10.1 in /usr/l
ocal/lib/python3.6/dist-packages (from pandas_profiling) (0.11.0)
Requirement already satisfied, skipping upgrade: jinja2>=2.11.1 in /usr/lo
cal/lib/python3.6/dist-packages (from pandas_profiling) (2.11.2)
Requirement already satisfied, skipping upgrade: joblib in /usr/local/lib/
python3.6/dist-packages (from pandas_profiling) (0.17.0)
Requirement already satisfied, skipping upgrade: phik>=0.9.10 in /usr/loca
l/lib/python3.6/dist-packages (from pandas_profiling) (0.10.0)
Requirement already satisfied, skipping upgrade: numpy>=1.16.0 in /usr/loc
al/lib/python3.6/dist-packages (from pandas_profiling) (1.18.5)
Requirement already satisfied, skipping upgrade: requests>=2.23.0 in /usr/
local/lib/python3.6/dist-packages (from pandas_profiling) (2.23.0)
Requirement already satisfied, skipping upgrade: pandas!=1.0.0,!=1.0.1,!=
1.0.2,!=1.1.0,>=0.25.3 in /usr/local/lib/python3.6/dist-packages (from pan
das_profiling) (1.1.4)
Requirement already satisfied, skipping upgrade: confuse>=1.0.0 in /usr/lo
cal/lib/python3.6/dist-packages (from pandas_profiling) (1.4.0)
Requirement already satisfied, skipping upgrade: matplotlib>=3.2.0 in /us
r/local/lib/python3.6/dist-packages (from pandas_profiling) (3.2.2)
Requirement already satisfied, skipping upgrade: networkx>=2.4 in /usr/loc
al/lib/python3.6/dist-packages (from visions[type_image_path]==0.5.0->pand
as_profiling) (2.5)
Requirement already satisfied, skipping upgrade: imagehash; extra == "type
_image_path" in /usr/local/lib/python3.6/dist-packages (from visions[type_
image_path]==0.5.0->pandas_profiling) (4.2.0)
Requirement already satisfied, skipping upgrade: Pillow; extra == "type_im
age_path" in /usr/local/lib/python3.6/dist-packages (from visions[type_ima
ge_path]==0.5.0->pandas_profiling) (7.0.0)
Requirement already satisfied, skipping upgrade: widgetsnbextension~=3.5.0
in /usr/local/lib/python3.6/dist-packages (from ipywidgets>=7.5.1->pandas_
profiling) (3.5.1)
Requirement already satisfied, skipping upgrade: nbformat>=4.2.0 in /usr/l
ocal/lib/python3.6/dist-packages (from ipywidgets>=7.5.1->pandas_profilin
g) (5.0.8)
Requirement already satisfied, skipping upgrade: traitlets>=4.3.1 in /usr/
```

```
local/lib/python3.6/dist-packages (from ipywidgets>=7.5.1->pandas_profilin
g) (4.3.3)
Requirement already satisfied, skipping upgrade: ipykernel>=4.5.1 in /usr/
local/lib/python3.6/dist-packages (from ipywidgets>=7.5.1->pandas_profilin
g) (4.10.1)
Requirement already satisfied, skipping upgrade: ipython>=4.0.0; python_ve
rsion >= "3.3" in /usr/local/lib/python3.6/dist-packages (from ipywidgets>
=7.5.1->pandas_profiling) (5.5.0)
Requirement already satisfied, skipping upgrade: MarkupSafe>=0.23 in /usr/
local/lib/python3.6/dist-packages (from jinja2>=2.11.1->pandas_profiling)
(1.1.1)
Requirement already satisfied, skipping upgrade: numba>=0.38.1 in /usr/loc
al/lib/python3.6/dist-packages (from phik>=0.9.10->pandas_profiling) (0.4
8.0)
Requirement already satisfied, skipping upgrade: idna<3,>=2.5 in /usr/loca
l/lib/python3.6/dist-packages (from requests>=2.23.0->pandas_profiling)
(2.10)
Requirement already satisfied, skipping upgrade: certifi>=2017.4.17 in /us
r/local/lib/python3.6/dist-packages (from requests>=2.23.0->pandas_profili
ng) (2020.11.8)
Requirement already satisfied, skipping upgrade: chardet<4,>=3.0.2 in /us
r/local/lib/python3.6/dist-packages (from requests>=2.23.0->pandas_profili
ng) (3.0.4)
Requirement already satisfied, skipping upgrade: urllib3!=1.25.0,!=1.25.1,
<1.26,>=1.21.1 in /usr/local/lib/python3.6/dist-packages (from requests>=
2.23.0->pandas_profiling) (1.24.3)
Requirement already satisfied, skipping upgrade: python-dateutil>=2.7.3 in
/usr/local/lib/python3.6/dist-packages (from pandas!=1.0.0,!=1.0.1,!=1.0.
2,!=1.1.0,>=0.25.3->pandas_profiling) (2.8.1)
Requirement already satisfied, skipping upgrade: pytz>=2017.2 in /usr/loca
l/lib/python3.6/dist-packages (from pandas!=1.0.0,!=1.0.1,!=1.0.2,!=1.1.0,
>=0.25.3->pandas_profiling) (2018.9)
Requirement already satisfied, skipping upgrade: pyyaml in /usr/local/lib/
python3.6/dist-packages (from confuse>=1.0.0->pandas_profiling) (3.13)
Requirement already satisfied, skipping upgrade: pyparsing!=2.0.4,!=2.1.
2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotl
ib>=3.2.0->pandas_profiling) (2.4.7)
Requirement already satisfied, skipping upgrade: cycler>=0.10 in /usr/loca
l/lib/python3.6/dist-packages (from matplotlib>=3.2.0->pandas_profiling)
(0.10.0)
Requirement already satisfied, skipping upgrade: kiwisolver>=1.0.1 in /us
r/local/lib/python3.6/dist-packages (from matplotlib>=3.2.0->pandas_profil
ing) (1.3.1)
Requirement already satisfied, skipping upgrade: decorator>=4.3.0 in /usr/
local/lib/python3.6/dist-packages (from networkx>=2.4->visions[type_image_
path]==0.5.0->pandas_profiling) (4.4.2)
Requirement already satisfied, skipping upgrade: PyWavelets in /usr/local/
lib/python3.6/dist-packages (from imagehash; extra == "type_image_path"->v
isions[type_image_path]==0.5.0->pandas_profiling) (1.1.1)
Requirement already satisfied, skipping upgrade: six in /usr/local/lib/pyt
hon3.6/dist-packages (from imagehash; extra == "type_image_path"->visions
[type_image_path]==0.5.0->pandas_profiling) (1.15.0)
Requirement already satisfied, skipping upgrade: notebook>=4.4.1 in /usr/l
ocal/lib/python3.6/dist-packages (from widgetsnbextension~=3.5.0->ipywidge
ts>=7.5.1->pandas_profiling) (5.3.1)
Requirement already satisfied, skipping upgrade: ipython-genutils in /usr/
local/lib/python3.6/dist-packages (from nbformat>=4.2.0->ipywidgets>=7.5.1
->pandas_profiling) (0.2.0)
Requirement already satisfied, skipping upgrade: jsonschema!=2.5.0,>=2.4 i
n /usr/local/lib/python3.6/dist-packages (from nbformat>=4.2.0->ipywidgets
>=7.5.1->pandas_profiling) (2.6.0)
```

```
Requirement already satisfied, skipping upgrade: jupyter-core in /usr/loca
l/lib/python3.6/dist-packages (from nbformat>=4.2.0->ipywidgets>=7.5.1->pa
ndas_profiling) (4.7.0)
Requirement already satisfied, skipping upgrade: tornado>=4.0 in /usr/loca
l/lib/python3.6/dist-packages (from ipykernel>=4.5.1->ipywidgets>=7.5.1->p
andas_profiling) (5.1.1)
Requirement already satisfied, skipping upgrade: jupyter-client in /usr/lo
cal/lib/python3.6/dist-packages (from ipykernel>=4.5.1->ipywidgets>=7.5.1-
>pandas_profiling) (5.3.5)
Requirement already satisfied, skipping upgrade: simplegeneric>0.8 in /us
r/local/lib/python3.6/dist-packages (from ipython>=4.0.0; python_version >
= "3.3"->ipywidgets>=7.5.1->pandas_profiling) (0.8.1)
Requirement already satisfied, skipping upgrade: pygments in /usr/local/li
b/python3.6/dist-packages (from ipython>=4.0.0; python_version >= "3.3"->i
pywidgets>=7.5.1->pandas_profiling) (2.6.1)
Requirement already satisfied, skipping upgrade: setuptools>=18.5 in /usr/
local/lib/python3.6/dist-packages (from ipython>=4.0.0; python_version >=
"3.3"->ipywidgets>=7.5.1->pandas_profiling) (50.3.2)
Requirement already satisfied, skipping upgrade: pickleshare in /usr/loca
l/lib/python3.6/dist-packages (from ipython>=4.0.0; python_version >= "3.
3"->ipywidgets>=7.5.1->pandas_profiling) (0.7.5)
Requirement already satisfied, skipping upgrade: prompt-toolkit<2.0.0,>=1.
0.4 in /usr/local/lib/python3.6/dist-packages (from ipython>=4.0.0; python
_version >= "3.3"->ipywidgets>=7.5.1->pandas_profiling) (1.0.18)
Requirement already satisfied, skipping upgrade: pexpect; sys_platform !=
"win32" in /usr/local/lib/python3.6/dist-packages (from ipython>=4.0.0; py
thon_version >= "3.3"->ipywidgets>=7.5.1->pandas_profiling) (4.8.0)
Requirement already satisfied, skipping upgrade: llvmlite<0.32.0,>=0.31.0d
ev0 in /usr/local/lib/python3.6/dist-packages (from numba>=0.38.1->phik>=
0.9.10->pandas_profiling) (0.31.0)
Requirement already satisfied, skipping upgrade: terminado>=0.8.1 in /usr/
local/lib/python3.6/dist-packages (from notebook>=4.4.1->widgetsnbextensio
n~=3.5.0->ipywidgets>=7.5.1->pandas_profiling) (0.9.1)
Requirement already satisfied, skipping upgrade: Send2Trash in /usr/local/
lib/python3.6/dist-packages (from notebook>=4.4.1->widgetsnbextension~=3.
5.0->ipywidgets>=7.5.1->pandas_profiling) (1.5.0)
Requirement already satisfied, skipping upgrade: nbconvert in /usr/local/l
ib/python3.6/dist-packages (from notebook>=4.4.1->widgetsnbextension~=3.5.
0->ipywidgets>=7.5.1->pandas_profiling) (5.6.1)
Requirement already satisfied, skipping upgrade: pyzmq>=13 in /usr/local/l
ib/python3.6/dist-packages (from jupyter-client->ipykernel>=4.5.1->ipywidg
ets>=7.5.1->pandas_profiling) (20.0.0)
Requirement already satisfied, skipping upgrade: wcwidth in /usr/local/li
b/python3.6/dist-packages (from prompt-toolkit<2.0.0,>=1.0.4->ipython>=4.
0.0; python_version >= "3.3"->ipywidgets>=7.5.1->pandas_profiling) (0.2.5)
Requirement already satisfied, skipping upgrade: ptyprocess>=0.5 in /usr/l
ocal/lib/python3.6/dist-packages (from pexpect; sys_platform != "win32"->i
python>=4.0.0; python_version >= "3.3"->ipywidgets>=7.5.1->pandas_profilin
g) (0.6.0)
Requirement already satisfied, skipping upgrade: entrypoints>=0.2.2 in /us
r/local/lib/python3.6/dist-packages (from nbconvert->notebook>=4.4.1->widg
etsnbextension~=3.5.0->ipywidgets>=7.5.1->pandas_profiling) (0.3)
Requirement already satisfied, skipping upgrade: testpath in /usr/local/li
b/python3.6/dist-packages (from nbconvert->notebook>=4.4.1->widgetsnbexten
sion~=3.5.0->ipywidgets>=7.5.1->pandas_profiling) (0.4.4)
Requirement already satisfied, skipping upgrade: bleach in /usr/local/lib/
python3.6/dist-packages (from nbconvert->notebook>=4.4.1->widgetsnbextensi
on~=3.5.0->ipywidgets>=7.5.1->pandas_profiling) (3.2.1)
Requirement already satisfied, skipping upgrade: pandocfilters>=1.4.1 in /
usr/local/lib/python3.6/dist-packages (from nbconvert->notebook>=4.4.1->wi
dgetsnbextension~=3.5.0->ipywidgets>=7.5.1->pandas_profiling) (1.4.3)
```

```
Requirement already satisfied, skipping upgrade: mistune<2,>=0.8.1 in /us
r/local/lib/python3.6/dist-packages (from nbconvert->notebook>=4.4.1->widg
etsnbextension~=3.5.0->ipywidgets>=7.5.1->pandas_profiling) (0.8.4)
Requirement already satisfied, skipping upgrade: defusedxml in /usr/local/
lib/python3.6/dist-packages (from nbconvert->notebook>=4.4.1->widgetsnbext
ension~=3.5.0->ipywidgets>=7.5.1->pandas_profiling) (0.6.0)
Requirement already satisfied, skipping upgrade: packaging in /usr/local/l
ib/python3.6/dist-packages (from bleach->nbconvert->notebook>=4.4.1->widge
tsnbextension~=3.5.0->ipywidgets>=7.5.1->pandas_profiling) (20.4)
Requirement already satisfied, skipping upgrade: webencodings in /usr/loca
l/lib/python3.6/dist-packages (from bleach->nbconvert->notebook>=4.4.1->wi
dgetsnbextension~=3.5.0->ipywidgets>=7.5.1->pandas_profiling) (0.5.1)
```

When we export Jupiter notebooks to HTML format, Plotly graphs disappear.
To solve it, we need to convert them to images.
If we want to convert Plotly graphs to image formats, we need to install `orca` .
Instruction has been taken from Install orca on Google Colab (https://plotly.com/python/orca-management/#install-orca-on-google-colab).

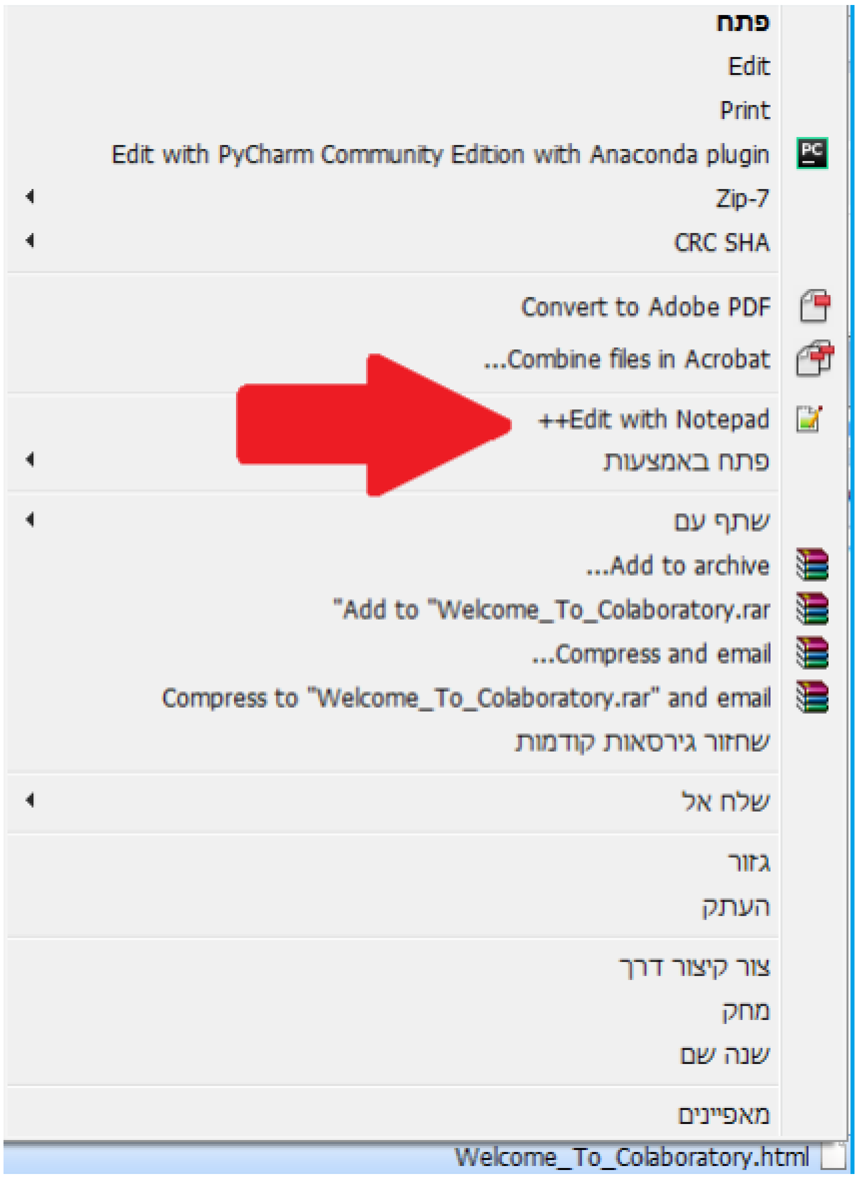
In [ ]:

```
# download and install `orca` and dependencies
!pip install plotly>=4.7.1
!wget https://github.com/plotly/orca/releases/download/v1.2.1/orca-1.2.1-x86_64.AppImag
e -O /usr/local/bin/orca
!chmod +x /usr/local/bin/orca
!apt-get install xvfb libgtk2.0-0 libgconf-2-4
```

```
--2020-11-30 00:39:43--  https://github.com/plotly/orca/releases/download/
v1.2.1/orca-1.2.1-x86_64.AppImage
Resolving github.com (github.com)... 140.82.114.3
Connecting to github.com (github.com)|140.82.114.3|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://github-production-release-asset-2e65be.s3.amazonaws.com/
99037241/9dc3a580-286a-11e9-8a21-4312b7c8a512?X-Amz-Algorithm=AWS4-HMAC-SH
A256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F20201130%2Fus-east-1%2Fs3%2Fa
ws4_request&X-Amz-Date=20201130T003943Z&X-Amz-Expires=300&X-Amz-Signature=
850a04703602e9cfcc60fdc3bbb67a5472b0b7b22fceea7a91e85f5929c09b8c&X-Amz-Sig
nedHeaders=host&actor_id=0&key_id=0&repo_id=99037241&response-content-disp
osition=attachment%3B%20filename%3Dorca-1.2.1-x86_64.AppImage&response-con
tent-type=application%2Foctet-stream [following]
--2020-11-30 00:39:43--  https://github-production-release-asset-2e65be.s
3.amazonaws.com/99037241/9dc3a580-286a-11e9-8a21-4312b7c8a512?X-Amz-Algori
thm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F20201130%2Fus
-east-1%2Fs3%2Faws4_request&X-Amz-Date=20201130T003943Z&X-Amz-Expires=300&
X-Amz-Signature=850a04703602e9cfcc60fdc3bbb67a5472b0b7b22fceea7a91e85f5929
c09b8c&X-Amz-SignedHeaders=host&actor_id=0&key_id=0&repo_id=99037241&respo
nse-content-disposition=attachment%3B%20filename%3Dorca-1.2.1-x86_64.AppIm
age&response-content-type=application%2Foctet-stream
Resolving github-production-release-asset-2e65be.s3.amazonaws.com (github-
production-release-asset-2e65be.s3.amazonaws.com)... 52.217.72.172
Connecting to github-production-release-asset-2e65be.s3.amazonaws.com (git
hub-production-release-asset-2e65be.s3.amazonaws.com)|52.217.72.172|:44
3... connected.
HTTP request sent, awaiting response... 200 OK
Length: 51607939 (49M) [application/octet-stream]
Saving to: ‘/usr/local/bin/orca’

/usr/local/bin/orca 100%[===================>]  49.22M  77.3MB/s    in 0.6
s

2020-11-30 00:39:44 (77.3 MB/s) - ‘/usr/local/bin/orca’ saved [51607939/51
607939]

Reading package lists... Done
Building dependency tree
Reading state information... Done
libgtk2.0-0 is already the newest version (2.24.32-1ubuntu1).
libgconf-2-4 is already the newest version (3.2.6-4ubuntu1).
xvfb is already the newest version (2:1.19.6-1ubuntu4.7).
0 upgraded, 0 newly installed, 0 to remove and 14 not upgraded.
```

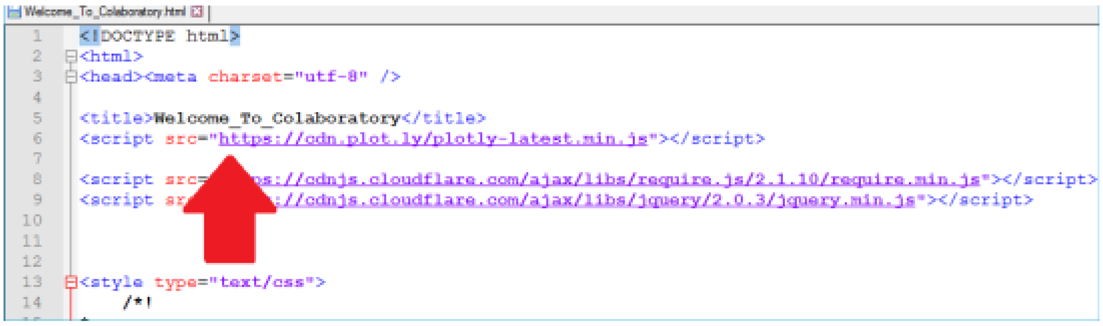
A More Convenient Solution

We can also avoid the disappearance of the plotly graphs by editing the HTML file as follows:


פתח
Edit
Print
Edit with PyCharm Community Edition with Anaconda plugin
Zip-7
CRC SHA
Convert to Adobe PDF
...Combine files in Acrobat
++Edit with Notepad
פתח באמצעות
שתף עם
...Add to archive
"Add to "Welcome_To_Colaboratory.rar
...Compress and email
Compress to "Welcome_To_Colaboratory.rar" and email
שחזור גירסאות קודמות
שלח אל
גזור
העתק
צור קיצור דרך
מחק
שנה שם
מאפיינים
Welcome_To_Colaboratory.html

And adding this line at the top of it:

```
<script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
```



```
Welcome_To_Colaboratory.html
<!DOCTYPE html>
<html>
<head><meta charset="utf-8" />

<title>Welcome_To_Colaboratory</title>
<script src="https://cdn.plot.ly/plotly-latest.min.js"></script>

<script src=   s://cdnjs.cloudflare.com/ajax/libs/require.js/2.1.10/require.min.js"></script>
<script sr     ://cdnjs.cloudflare.com/ajax/libs/jquery/2.0.3/jquery.min.js"></script>



<style type="text/css">
    /*!
```

Yay! Thanks for this awesome solution Dima Tchenak 😀.

Now, we can install our regular packages.

In [ ]:

```
# import numpy, matplotlib, etc.
import numpy as np
import pandas as pd
import seaborn as sns
import plotly.express as px
import matplotlib.pyplot as plt

# sklearn imports
from sklearn import metrics
from sklearn import pipeline
from sklearn import linear_model
from sklearn import preprocessing
from sklearn import neural_network
from sklearn import model_selection
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import SGDClassifier
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

Data Exploration

We use the *Mushroom* dataset (https://archive.ics.uci.edu/ml/datasets/Mushroom) from 1987.

Dataset Information

This data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family.
Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended.
This latter class was combined with the poisonous one.
The Guide clearly states that there is no simple rule for determining the edibility of a mushroom; no rule like ``leaflets three, let it be'' for Poisonous Oak and Ivy.

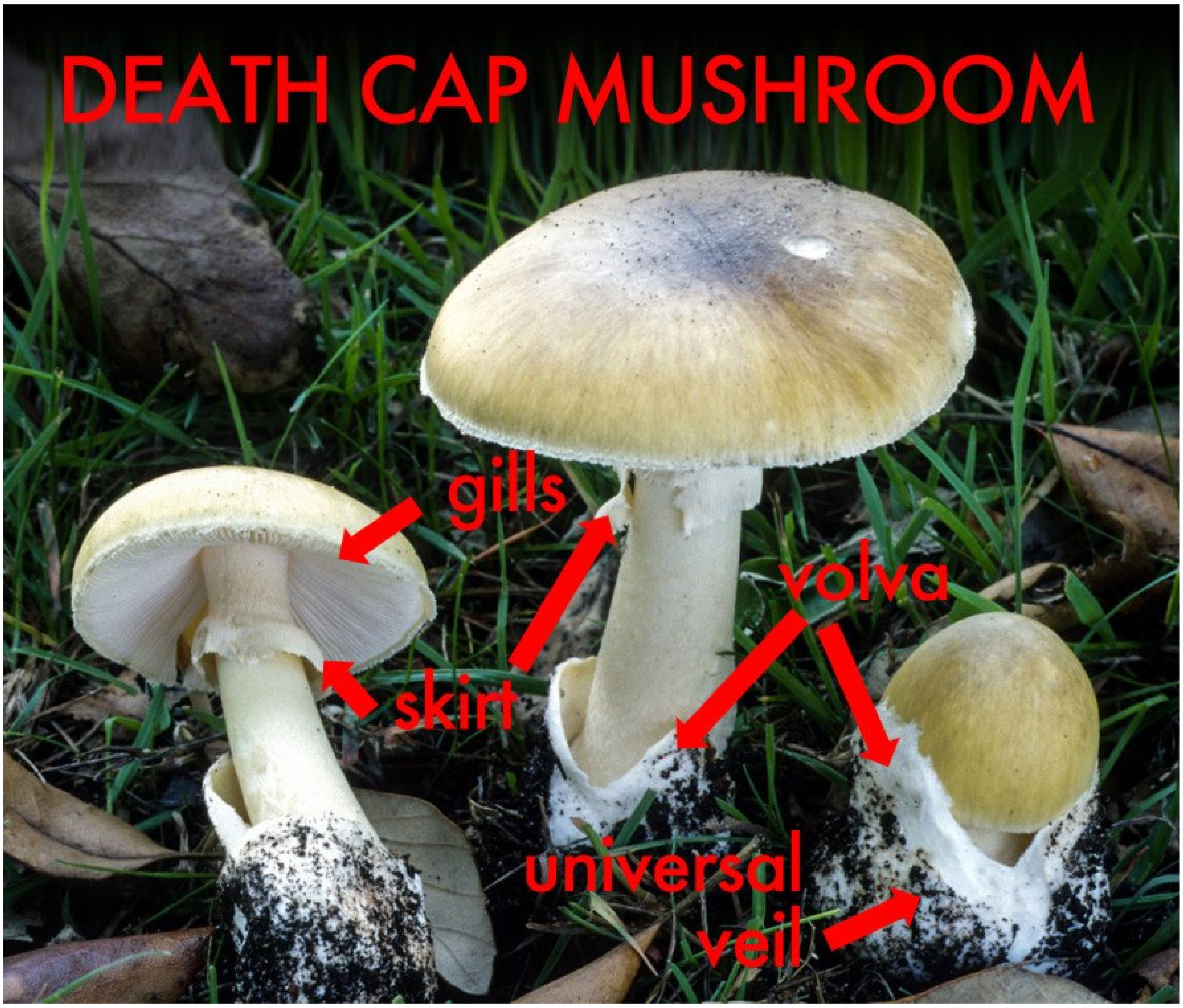
Attribute Information

1. **cap-shape**: bell=b, conical=c, convex=x, flat=f, knobbed=k, sunken=s
2. **cap-surface**: fibrous=f, grooves=g, scaly=y, smooth=s
3. **cap-color**: brown=n, buff=b, cinnamon=c, gray=g, green=r, pink=p, purple=u,red=e,white=w,yellow=y
4. **bruises**?: bruises=t, no=f
5. **odor**: almond=a, anise=l, creosote=c, fishy=y, foul=f, musty=m, none=n, pungent=p, spicy=s
6. **gill-attachment**: attached=a, descending=d, free=f, notched=n
7. **gill-spacing**: close=c, crowded=w, distant=d
8. **gill-size**: broad=b, narrow=n
9. **gill-color**: black=k, brown=n, buff=b, chocolate=h, gray=g, green=r, orange=o, pink=p, purple=u, red=e, white=w, yellow=y
10. **stalk-shape**: enlarging=e, tapering=t
11. **stalk-root**: bulbous=b, club=c, cup=u, equal=e, rhizomorphs=z, rooted=r, missing=?
12. **stalk-surface-above-ring**: fibrous=f, scaly=y, silky=k, smooth=s

13. **stalk-surface-below-ring**: fibrous=f, scaly=y, silky=k, smooth=s
14. **stalk-color-above-ring**: brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y
15. **stalk-color-below-ring**: brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y
16. **veil-type**: partial=p, universal=u
17. **veil-color**: brown=n, orange=o, white=w, yellow=y
18. **ring-number**: none=n, one=o, two=t
19. **ring-type**: cobwebby=c, evanescent=e, flaring=f, large=l, none=n, pendant=p, sheathing=s, zone=z
20. **spore-print-color**: black=k, brown=n, buff=b, chocolate=h, green=r, orange=o, purple=u, white=w, yellow=y
21. **population**: abundant=a, clustered=c, numerous=n, scattered=s, several=v, solitary=y
22. **habitat**: grasses=g, leaves=l, meadows=m, paths=p, urban=u, waste=w, woods=d

Traget Information

- **type**: poisonous=p, edible=e



Let's download the dataset from Github and explore it with Pandas tools.

In [ ]:

```
# download mashrooms.csv file from Github
!wget https://raw.githubusercontent.com/stedy/Machine-Learning-with-R-datasets/master/m
ushrooms.csv
```

```
--2020-11-30 00:39:48--  https://raw.githubusercontent.com/stedy/Machine-L
earning-with-R-datasets/master/mushrooms.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.10
1.0.133, 151.101.64.133, 151.101.128.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.10
1.0.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 374003 (365K) [text/plain]
Saving to: ‘mushrooms.csv.3’

mushrooms.csv.3     100%[===================>] 365.24K  --.-KB/s    in 0.0
4s

2020-11-30 00:39:48 (8.82 MB/s) - ‘mushrooms.csv.3’ saved [374003/374003]
```

In [ ]:

```
# Load the insurance csv file
mushrooms_df = pd.read_csv('mushrooms.csv')
mushrooms_df
```

Out[ ]:

	type	cap_shape	cap_surface	cap_color	bruises	odor	gill_attachment	gill_spacing
0	p	x	s	n	t	p	f	c
1	e	x	s	y	t	a	f	c
2	e	b	s	w	t	l	f	c
3	p	x	y	w	t	p	f	c
4	e	x	s	g	f	n	f	w
...	...	...	...	...	...	...	...	...
8119	e	k	s	n	f	n	a	c
8120	e	x	s	n	f	n	a	c
8121	e	f	s	n	f	n	a	c
8122	p	k	y	n	f	y	f	c
8123	e	x	s	n	f	n	a	c

8124 rows × 23 columns

In [ ]:

```
# show mushrooms_df info
mushrooms_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8124 entries, 0 to 8123
Data columns (total 23 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   type                      8124 non-null   object
 1   cap_shape                 8124 non-null   object
 2   cap_surface               8124 non-null   object
 3   cap_color                 8124 non-null   object
 4   bruises                   8124 non-null   object
 5   odor                      8124 non-null   object
 6   gill_attachment           8124 non-null   object
 7   gill_spacing              8124 non-null   object
 8   gill_size                 8124 non-null   object
 9   gill_color                8124 non-null   object
 10  stalk_shape               8124 non-null   object
 11  stalk_root                8124 non-null   object
 12  stalk_surface_above_ring  8124 non-null   object
 13  stalk_surface_below_ring  8124 non-null   object
 14  stalk_color_above_ring    8124 non-null   object
 15  stalk_color_below_ring    8124 non-null   object
 16  veil_type                 8124 non-null   object
 17  veil_color                8124 non-null   object
 18  ring_number               8124 non-null   object
 19  ring_type                 8124 non-null   object
 20  spore_print_color         8124 non-null   object
 21  population                8124 non-null   object
 22  habitat                   8124 non-null   object
dtypes: object(23)
memory usage: 1.4+ MB
```

Though the dataset shows there are no null values, there are missing values. The missing values are only present within the `stalk_root` column. Let's make pandas recognize those missing values.

In [ ]:

```
mushrooms_df.replace('?', np.NAN, inplace=True)
mushrooms_df
```

In [ ]:

```
mushrooms_df.isna().sum() # Checking how many missing values there are
```

Out[ ]:

```
type                           0
cap_shape                      0
cap_surface                    0
cap_color                      0
bruises                        0
odor                           0
gill_attachment                0
gill_spacing                   0
gill_size                      0
gill_color                     0
stalk_shape                    0
stalk_root                  2480
stalk_surface_above_ring       0
stalk_surface_below_ring       0
stalk_color_above_ring         0
stalk_color_below_ring         0
veil_type                      0
veil_color                     0
ring_number                    0
ring_type                      0
spore_print_color              0
population                     0
habitat                        0
dtype: int64
```

Using the imputer from the previous notebook in order to fill the missing values in our dataset.

In [ ]:

```
def fill_na_random_pick_column_distribution(df, column_name):
    df_not_null = df[~df[column_name].isnull()]
    df_null = df[df[column_name].isnull()]
    options = np.random.choice(df_not_null[column_name])
    df[column_name] = df[column_name].apply(lambda x: np.random.choice(df_not_null[colu
mn_name]) if pd.isnull(x) else x)

fill_na_random_pick_column_distribution(mushrooms_df, 'stalk_root')
mushrooms_df
```

Out[ ]:

	type	cap_shape	cap_surface	cap_color	bruises	odor	gill_attachment	gill_spacing
0	p	x	s	n	t	p	f	c
1	e	x	s	y	t	a	f	c
2	e	b	s	w	t	l	f	c
3	p	x	y	w	t	p	f	c
4	e	x	s	g	f	n	f	w
...	...	...	...	...	...	...	...	...
8119	e	k	s	n	f	n	a	c
8120	e	x	s	n	f	n	a	c
8121	e	f	s	n	f	n	a	c
8122	p	k	y	n	f	y	f	c
8123	e	x	s	n	f	n	a	c

8124 rows × 23 columns

In [ ]:

```
# show mushrooms_df description
mushrooms_df.describe()
```

Out[ ]:

	type	cap_shape	cap_surface	cap_color	bruises	odor	gill_attachment	gill_spacin
count	8124	8124	8124	8124	8124	8124	8124	812
unique	2	6	4	10	2	9	2	
top	e	x	y	n	f	n	f	
freq	4208	3656	3244	2284	4748	3528	7914	681

We can see that `info` and `describe` are helping us understand the dataset.
We can also use pandas_profiling `ProfileReport` .

In [ ]:

```
# import ProfileReport and show report of mushroom_df
from pandas_profiling import ProfileReport

ProfileReport(mushrooms_df, title="Mushroom Profiling Report", explorative=True)
```

Overview

Dataset statistics

Number of variables	23
Number of observations	8124
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	11.7 MiB
Average record size in memory	1.5 KiB

Variable types

CAT	22
BOOL	1

Warnings

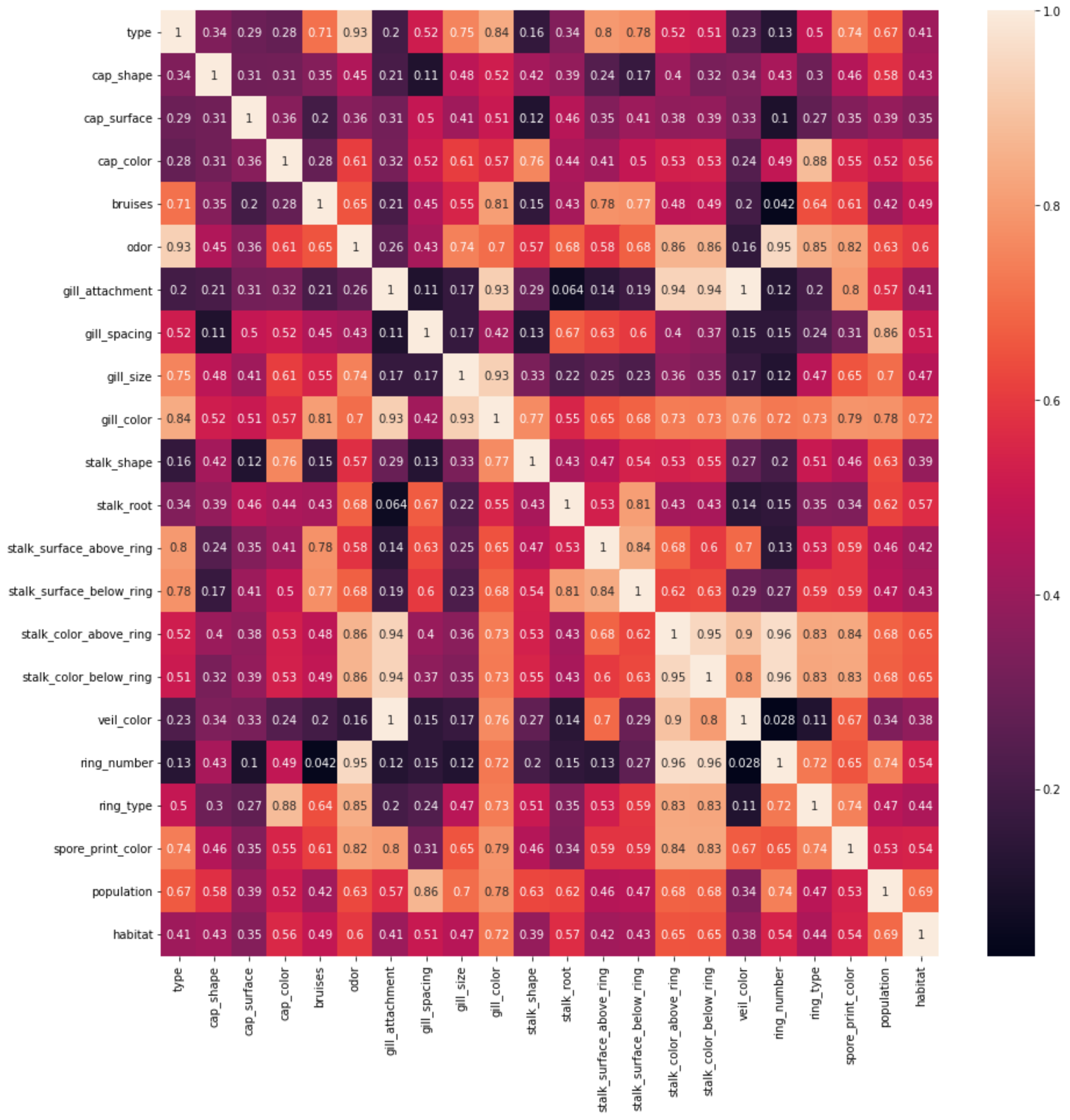
`veil_type` has constant value "8124"	Constant
`odor` is highly correlated with `type`	High correlation
`type` is highly correlated with `odor`	High correlation

```
Out[ ]:
```

A more Zoomed-in Version of The Correlations between Features

In [ ]:

```
mushrooms_df.drop(['veil_type'], axis=1, inplace=True)  # Dropping veil because it does
n't contribute to us
corr = mushrooms_df.phik_matrix()
plt.figure(figsize=(20, 20))
sns.heatmap(corr, annot=True)
plt.show()
```



Data Pre-processing

We see that we have only categorical data, so we need to do feature encoding.
We can prepare the encoding now and use it later.

In [ ]:

```
# prepare OneHotEncoder, DummyEncoder and OrdinalEncoder for latter use
ohe = OneHotEncoder(sparse=False).fit(mushrooms_df)
de = OneHotEncoder(drop="first", sparse=False).fit(mushrooms_df)
oe = OrdinalEncoder().fit(mushrooms_df)

hp_encoders = {'One-Hot Encoder': ohe, 'Dummy Encoder': de, 'Ordinal Encoder': oe}
```

Let's see the categories of the encoding.

In [ ]:

```
# show the categories
ohe.categories_
```

Out[ ]:

```
[array(['e', 'p'], dtype=object),
 array(['b', 'c', 'f', 'k', 's', 'x'], dtype=object),
 array(['f', 'g', 's', 'y'], dtype=object),
 array(['b', 'c', 'e', 'g', 'n', 'p', 'r', 'u', 'w', 'y'], dtype=object),
 array(['f', 't'], dtype=object),
 array(['a', 'c', 'f', 'l', 'm', 'n', 'p', 's', 'y'], dtype=object),
 array(['a', 'f'], dtype=object),
 array(['c', 'w'], dtype=object),
 array(['b', 'n'], dtype=object),
 array(['b', 'e', 'g', 'h', 'k', 'n', 'o', 'p', 'r', 'u', 'w', 'y'],
       dtype=object),
 array(['e', 't'], dtype=object),
 array(['?', 'b', 'c', 'e', 'r'], dtype=object),
 array(['f', 'k', 's', 'y'], dtype=object),
 array(['f', 'k', 's', 'y'], dtype=object),
 array(['b', 'c', 'e', 'g', 'n', 'o', 'p', 'w', 'y'], dtype=object),
 array(['b', 'c', 'e', 'g', 'n', 'o', 'p', 'w', 'y'], dtype=object),
 array(['p'], dtype=object),
 array(['n', 'o', 'w', 'y'], dtype=object),
 array(['n', 'o', 't'], dtype=object),
 array(['e', 'f', 'l', 'n', 'p'], dtype=object),
 array(['b', 'h', 'k', 'n', 'o', 'r', 'u', 'w', 'y'], dtype=object),
 array(['a', 'c', 'n', 's', 'v', 'y'], dtype=object),
 array(['d', 'g', 'l', 'm', 'p', 'u', 'w'], dtype=object)]
```

We can flatten the array and use it as column names for the pre-processed DataFrame.

In [ ]:

```
# flatten nd array to 1d array
def fromNDArrayto1DArray(nd_array):
    if isinstance(nd_array, str):
        return [nd_array]

    oned_array = []
    try:
        for v in nd_array:
            oned_array += fromNDArrayto1DArray(v)
        return oned_array
    except TypeError:
        return [nd_array]

fromNDArrayto1DArray(ohe.categories_)
```

Out[ ]:

```
['e',
 'p',
 'b',
 'c',
 'f',
 'k',
 's',
 'x',
 'f',
 'g',
 's',
 'y',
 'b',
 'c',
 'e',
 'g',
 'n',
 'p',
 'r',
 'u',
 'w',
 'y',
 'f',
 't',
 'a',
 'c',
 'f',
 'l',
 'm',
 'n',
 'p',
 's',
 'y',
 'a',
 'f',
 'c',
 'w',
 'b',
 'n',
 'b',
 'e',
 'g',
 'h',
 'k',
 'n',
 'o',
 'p',
 'r',
 'u',
 'w',
 'y',
 'e',
 't',
 '?',
 'b',
 'c',
 'e',
 'r',
 'f',
 'k',
 's',
 'y',
 'f',
 'k',
 's',
 'y',
 'b',
 'c',
 'e',
 'g',
 'n',
 'o',
 'p',
 'w',
 'y',
 'b',
 'c',
 'e',
 'g',
 'n',
 'o',
 'p',
 'w',
 'y',
 'p',
 'n',
 'o',
 'w',
 'y',
 'n',
 'o',
 't',
 'e',
 'f',
 'l',
 'n',
 'p',
 'b',
 'h',
 'k',
 'n',
 'o',
 'r',
 'u',
 'w',
 'y',
 'a',
 'c',
 'n',
 's',
 'v',
 'y',
 'd',
 'g',
 'l',
 'm',
 'p',
 'u',
 'w']
```

Or we can simply just do that instead:

So let's use it

In [ ]:

```
# show the categories
ohe.get_feature_names_out()
```

Out[ ]:

```
array(['type_e', 'type_p', 'cap_shape_b', 'cap_shape_c', 'cap_shape_f',
       'cap_shape_k', 'cap_shape_s', 'cap_shape_x', 'cap_surface_f',
       'cap_surface_g', 'cap_surface_s', 'cap_surface_y', 'cap_color_b',
       'cap_color_c', 'cap_color_e', 'cap_color_g', 'cap_color_n',
       'cap_color_p', 'cap_color_r', 'cap_color_u', 'cap_color_w',
       'cap_color_y', 'bruises_f', 'bruises_t', 'odor_a', 'odor_c',
       'odor_f', 'odor_l', 'odor_m', 'odor_n', 'odor_p', 'odor_s',
       'odor_y', 'gill_attachment_a', 'gill_attachment_f',
       'gill_spacing_c', 'gill_spacing_w', 'gill_size_b', 'gill_size_n',
       'gill_color_b', 'gill_color_e', 'gill_color_g', 'gill_color_h',
       'gill_color_k', 'gill_color_n', 'gill_color_o', 'gill_color_p',
       'gill_color_r', 'gill_color_u', 'gill_color_w', 'gill_color_y',
       'stalk_shape_e', 'stalk_shape_t', 'stalk_root_b', 'stalk_root_c',
       'stalk_root_e', 'stalk_root_r', 'stalk_surface_above_ring_f',
       'stalk_surface_above_ring_k', 'stalk_surface_above_ring_s',
       'stalk_surface_above_ring_y', 'stalk_surface_below_ring_f',
       'stalk_surface_below_ring_k', 'stalk_surface_below_ring_s',
       'stalk_surface_below_ring_y', 'stalk_color_above_ring_b',
       'stalk_color_above_ring_c', 'stalk_color_above_ring_e',
       'stalk_color_above_ring_g', 'stalk_color_above_ring_n',
       'stalk_color_above_ring_o', 'stalk_color_above_ring_p',
       'stalk_color_above_ring_w', 'stalk_color_above_ring_y',
       'stalk_color_below_ring_b', 'stalk_color_below_ring_c',
       'stalk_color_below_ring_e', 'stalk_color_below_ring_g',
       'stalk_color_below_ring_n', 'stalk_color_below_ring_o',
       'stalk_color_below_ring_p', 'stalk_color_below_ring_w',
       'stalk_color_below_ring_y', 'veil_color_n', 'veil_color_o',
       'veil_color_w', 'veil_color_y', 'ring_number_n', 'ring_number_o',
       'ring_number_t', 'ring_type_e', 'ring_type_f', 'ring_type_l',
       'ring_type_n', 'ring_type_p', 'spore_print_color_b',
       'spore_print_color_h', 'spore_print_color_k',
       'spore_print_color_n', 'spore_print_color_o',
       'spore_print_color_r', 'spore_print_color_u',
       'spore_print_color_w', 'spore_print_color_y', 'population_a',
       'population_c', 'population_n', 'population_s', 'population_v',
       'population_y', 'habitat_d', 'habitat_g', 'habitat_l', 'habitat_m',
       'habitat_p', 'habitat_u', 'habitat_w'], dtype=object)
```

Well it seems far more easy to perform and also the feature names are presented with the values.

In [ ]:

```
# show the dataframe after encoding
pd.DataFrame(ohe.transform(mushrooms_df), columns=fromNDArrayto1DArray(ohe.categories_
))
```

Out[ ]:

	e	p	b	c	f	k	s	x	f	g	s	y	b	c	e	g	n	p	ı
0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.
1	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
2	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
3	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
4	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
8119	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.
8120	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.
8121	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.
8122	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.
8123	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.

8124 rows × 119 columns

Let's take the target out of the original DataFrame.

In [ ]:

```python
# divide the data to features and target
t = mushrooms_df['type'].copy()
X = mushrooms_df.drop(['type'], axis=1)
print('t')
display(t)
print()
print('X')
display(X)
```

```
t

0       p
1       e
2       e
3       p
4       e
       ..
8119    e
8120    e
8121    e
8122    p
8123    e
Name: type, Length: 8124, dtype: object
```

X

	cap_shape	cap_surface	cap_color	bruises	odor	gill_attachment	gill_spacing	gill_si
0	x	s	n	t	p	f	c	
1	x	s	y	t	a	f	c	
2	b	s	w	t	l	f	c	
3	x	y	w	t	p	f	c	
4	x	s	g	f	n	f	w	
...	...	...	...	...	...	...	...	
8119	k	s	n	f	n	a	c	
8120	x	s	n	f	n	a	c	
8121	f	s	n	f	n	a	c	
8122	k	y	n	f	y	f	c	
8123	x	s	n	f	n	a	c	

8124 rows × 22 columns

Let's split the data to train and validation.

In [ ]:

```
# split the data to train and validation
X_train, X_val, t_train, t_val = model_selection.train_test_split(X, t, test_size=0.3,
random_state=1)
print('X_train')
display(X_train)
print()
print('t_train')
display(t_train)
print()
print('X_val')
display(X_val)
print()
print('t_val')
display(t_val)
```

```
X_train
```

	cap_shape	cap_surface	cap_color	bruises	odor	gill_attachment	gill_spacing	gill_si
6848	f	s	e	f	f	f	c	
2246	x	y	g	t	n	f	c	
2095	x	y	g	t	n	f	c	
4535	f	y	y	f	f	f	c	
6865	f	s	n	f	y	f	c	
...	...	...	...	...	...	...	...	
7935	k	y	e	f	y	f	c	
5192	x	y	n	f	s	f	c	
3980	f	y	g	f	f	f	c	
235	x	f	w	t	l	f	w	
5157	x	s	b	t	f	f	c	

5686 rows × 22 columns

```
t_train
```

```
6848    p
2246    e
2095    e
4535    p
6865    p
       ..
7935    p
5192    p
3980    p
235     e
5157    p
Name: type, Length: 5686, dtype: object
```

```
X_val
```

	cap_shape	cap_surface	cap_color	bruises	odor	gill_attachment	gill_spacing	gill_si
1392	f	s	w	f	n	f	w	
4051	x	f	p	f	c	f	w	
3725	x	s	w	f	c	f	w	
7177	k	s	n	f	f	f	c	
103	x	y	y	t	a	f	c	
...	...	...	...	...	...	...	...	
6017	f	y	n	f	f	f	c	
2505	x	y	e	t	n	f	c	
5333	f	s	w	t	f	f	c	
2917	f	f	n	t	n	f	c	
6022	f	y	e	f	s	f	c	

2438 rows × 22 columns

```
t_val
```

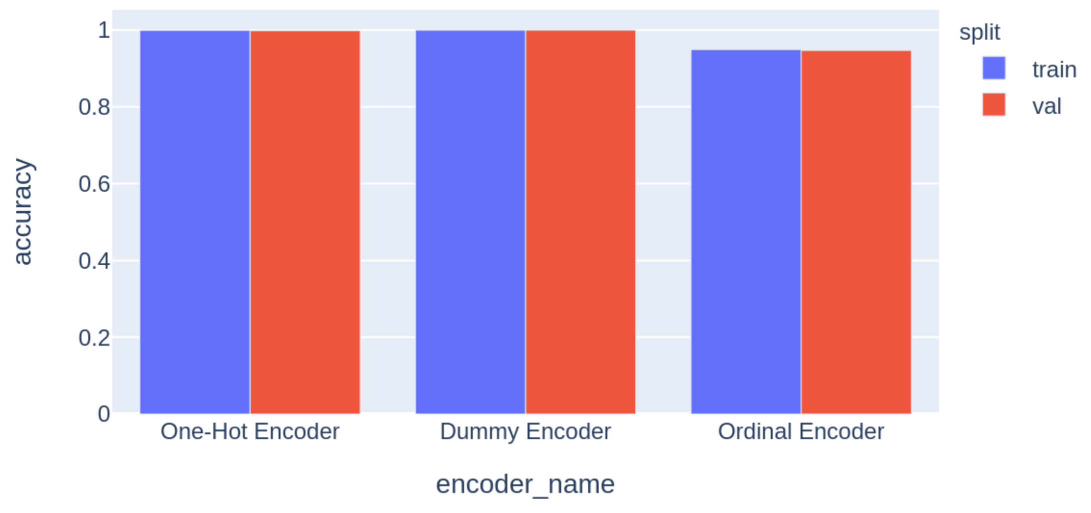
```
1392    e
4051    p
3725    p
7177    p
103     e
       ..
6017    p
2505    e
5333    p
2917    e
6022    p
Name: type, Length: 2438, dtype: object
```

Model Training

Let's train SGDClassifier and see which encoding is better on this dataset.
We will plot bar graphs with Plotly and convert the figures to images using Get Image as Bytes (https://plotly.com/python/static-image-export/#get-image-as-bytes).
Then, we will print the image with IPython.display.Image (https://ipython.readthedocs.io/en/stable/api/generated/IPython.display.html#IPython.display.Image).

In [ ]:

```
# show the accuracy by encoder
from IPython.display import Image

def show_accuracy_by_encoder(X_train, t_train, X_val, t_val, encoders):
    accuracy_df = pd.DataFrame(columns=['encoder_name', 'split', 'accuracy'])

    for encoder_name, encoder in encoders.items():
        model_pipe = make_pipeline(encoder, SGDClassifier(random_state=1))
        model_pipe.fit(X_train, t_train)
        accuracy_df.loc[len(accuracy_df)] = [encoder_name, 'train', model_pipe.score(X_
train, t_train)]
        accuracy_df.loc[len(accuracy_df)] = [encoder_name, 'val', model_pipe.score(X_va
l, t_val)]

    fig = px.bar(accuracy_df, x='encoder_name', y='accuracy', color='split', barmode='g
roup')
    img_bytes = fig.to_image(format="png", width=600, height=350, scale=2)
    display(Image(img_bytes))

show_accuracy_by_encoder(X_train, t_train, X_val, t_val, hp_encoders)
```

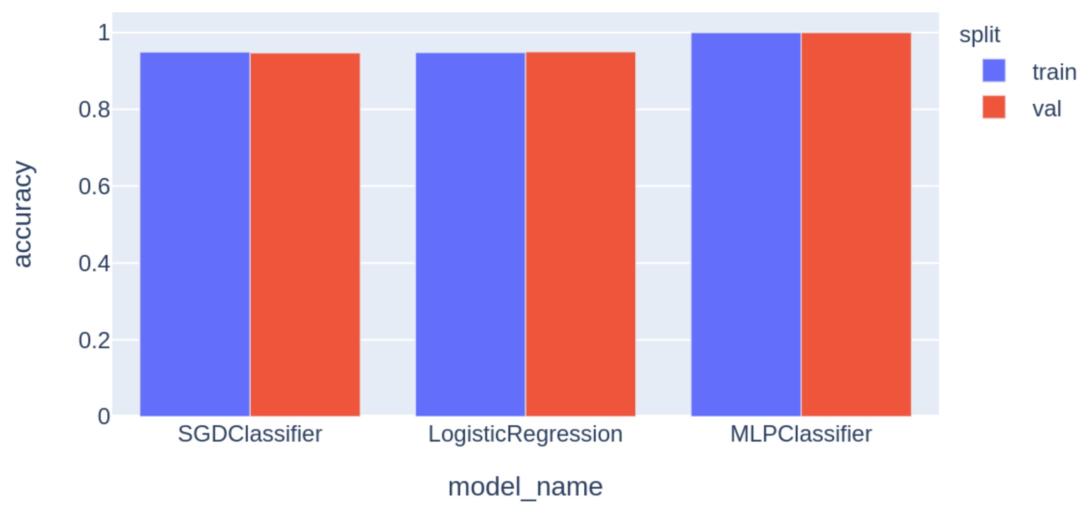


Let's use `OrdinalEncoding` and see which model is better for this dataset.

In [ ]:

```
# show the accuracy by model
hp_models = {'SGDClassifier': SGDClassifier(random_state=1), 'LogisticRegression': Logi
sticRegression(), 'MLPClassifier': MLPClassifier()}

def show_accuracy_by_model(X_train, t_train, X_val, t_val, models):
    accuracy_df = pd.DataFrame(columns=['model_name', 'split', 'accuracy'])

    for model_name, model in models.items():
        model_pipe = make_pipeline(oe, model)
        model_pipe.fit(X_train, t_train)
        accuracy_df.loc[len(accuracy_df)] = [model_name, 'train', model_pipe.score(X_tr
ain, t_train)]
        accuracy_df.loc[len(accuracy_df)] = [model_name, 'val', model_pipe.score(X_val,
t_val)]

    fig = px.bar(accuracy_df, x='model_name', y='accuracy', color='split', barmode='gro
up')
    img_bytes = fig.to_image(format="png", width=600, height=350, scale=2)
    display(Image(img_bytes))

show_accuracy_by_model(X_train, t_train, X_val, t_val, hp_models)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_logistic.py:9
40: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression
```

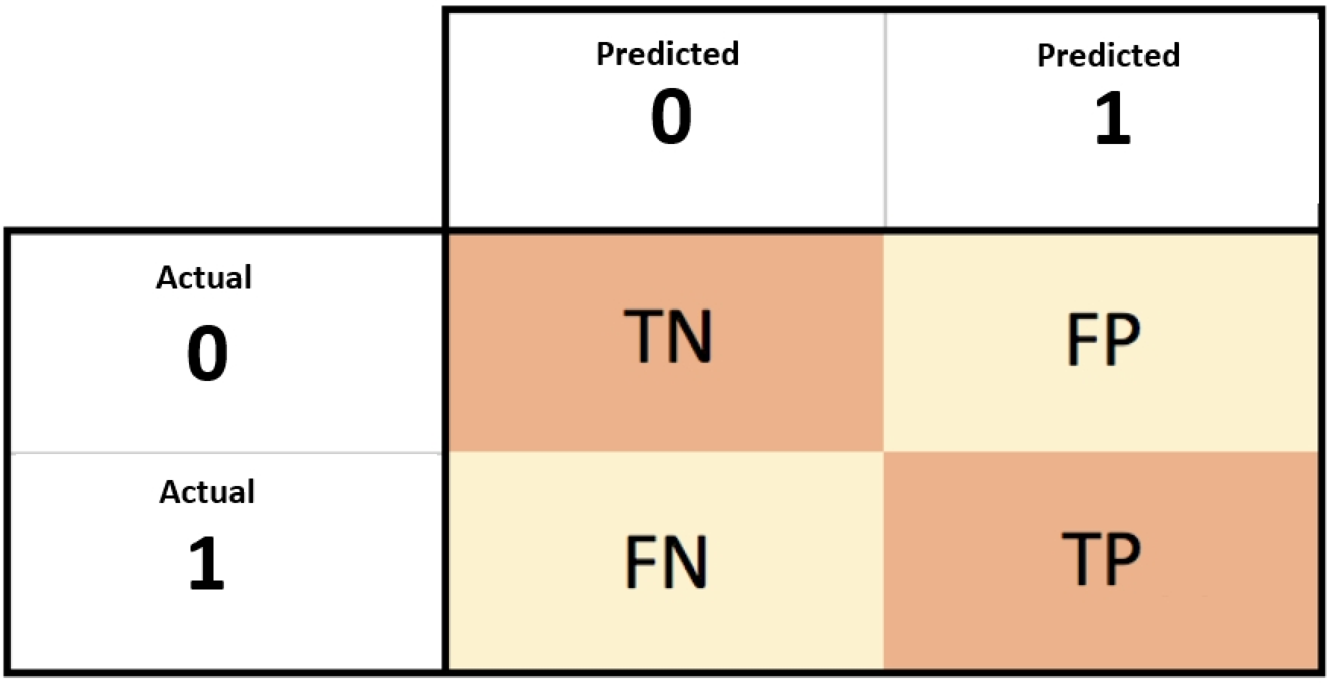


Confusion Matrix

We want to be able to explain the results of a classifier.
When talking about binary classification, the classification type of a sample can be one out of four:

1. **TP (True Positive**) - The model classified correctly that a sample is positive.
2. **TN (True Negative)** - The model classified correctly that a sample is negative.
3. **FP (False Positive)** - The model classified a sample as positive but the sample is actually negative.
4. **FN (False Negative)** - The model classified a sample as negative but the sample is actually positive.

We can show these values in a matrix:



	Predicted 0	Predicted 1
Actual 0	TN	FP
Actual 1	FN	TP

This matrix is called **Confusion Matrix** (https://en.wikipedia.org/wiki/Confusion_matrix).
With these values we can calculate a few **KPI**'s (Key Performance Indicators):

Precision

The percentage of correct positive predictions out of all the positive predictions.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

Recall

The percentage of correct positive predictions out of all the actual positive samples.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}}$$

Specificity

The percentage of correct negative predictions out of all the actual negative samples.

$$\text{Specificity} = \frac{\text{True negative}}{\text{True negative + false positive}}$$

FPR

The percentage of mistaken negative predictions out of all the actual negative samples.

$$\text{FPR} = \frac{\text{False Positive}}{\text{False Positive + True Negative}}$$

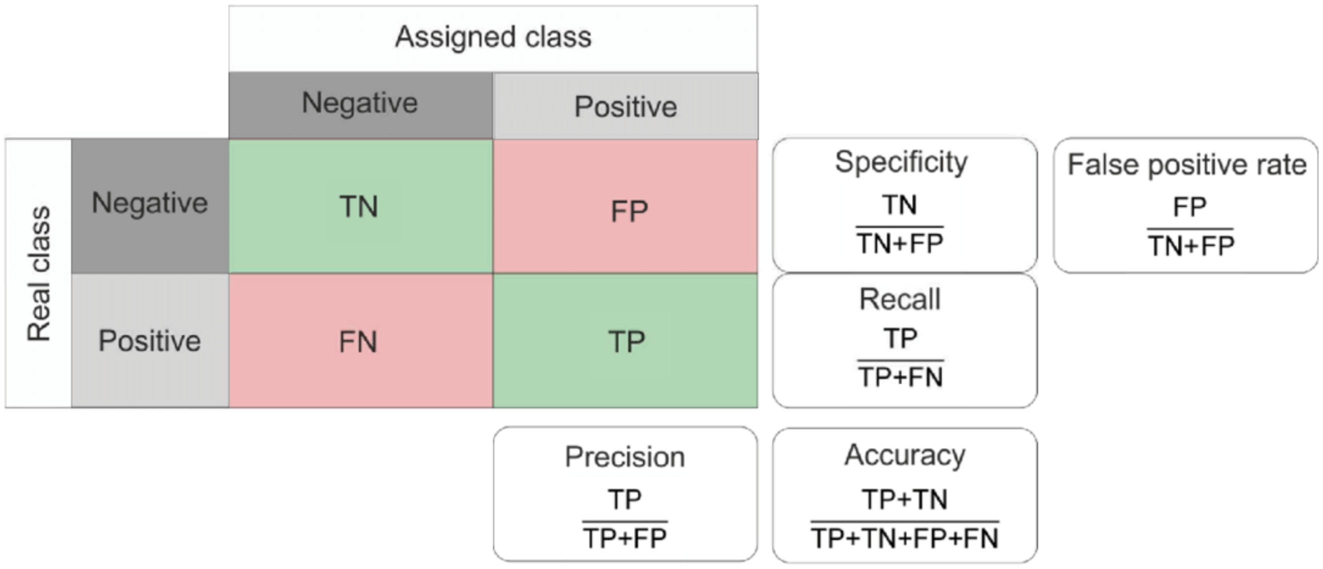
Accuracy

The percentage of correct predictions out of all the data.

$$\text{Accuracy} = \frac{\text{True Positives + True Negatives}}{\text{True Positives + False Positives + False Negatives + True Negatives}}$$

Confusion Matrix with KPI's

Here we can see all the important simple KPI's.
We can see that FPR = 1- Specificity.
There are a lot more KPI's, you can read about them in Wikipedia (https://en.wikipedia.org/wiki/Confusion_matrix).


Assigned class
Negative
Positive
Real class
Negative
Positive
TN
FP
FN
TP
Specificity $\frac{TN}{TN+FP}$
False positive rate $\frac{FP}{TN+FP}$
Recall $\frac{TP}{TP+FN}$
Precision $\frac{TP}{TP+FP}$
Accuracy $\frac{TP+TN}{TP+TN+FP+FN}$

F1 Score

The harmonic mean of precision and recall.

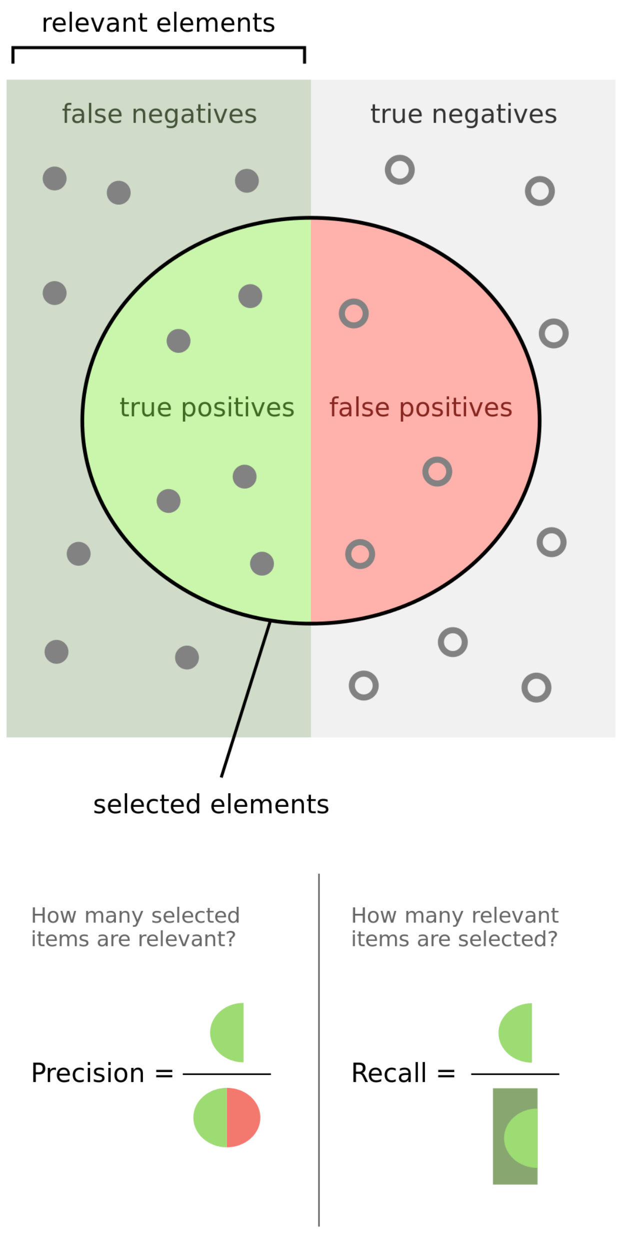
$$\text{F1 Score} = \frac{2 \text{ x } (\text{Precision x Recall})}{\text{Precision} + \text{Recall}}$$

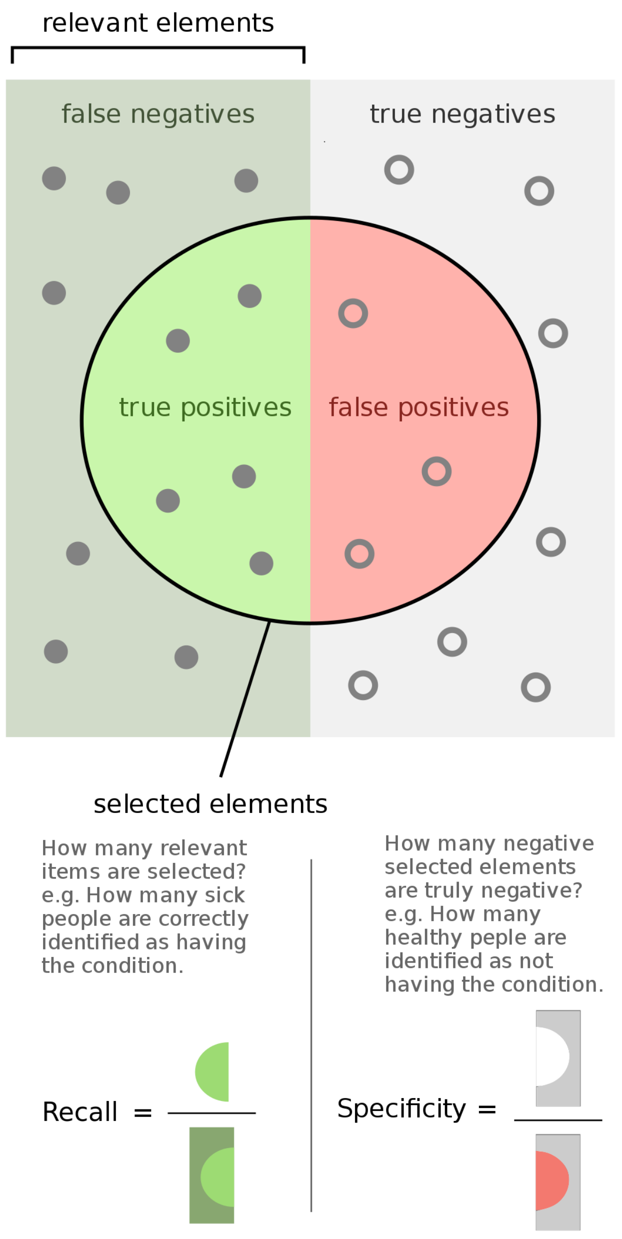
Balanced Accuracy

The arithmetic means of recall and specificity.

$$Balanced\ Accuracy = \frac{Recall + Specificity}{2}$$

Comparisons




relevant elements
false negatives
true negatives
true positives
false positives
selected elements
How many relevant items are selected? e.g. How many sick people are correctly identified as having the condition.
How many negative selected elements are truly negative? e.g. How many healthy peple are identified as not having the condition.
Recall =
Specificity =

When we use it on code, we can use Scikit-learn [confusion_matrix (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html)](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html).
Let's calculate the confusion matrix for SGDClassifier that is trained on the Train Data with Ordinal Encoding.

In [ ]:

```
# calculate cm for train and val
from sklearn.metrics import confusion_matrix

model_pipe = make_pipeline(oe, SGDClassifier(random_state=1)).fit(X_train, t_train)
y_train = model_pipe.predict(X_train)
y_val = model_pipe.predict(X_val)
cm_train = confusion_matrix(t_train, y_train)
cm_val = confusion_matrix(t_val, y_val)

print('cm_train')
print(cm_train)
print()
print('cm_val')
print(cm_val)
```

```
cm_train
[[2794  178]
 [ 109 2605]]

cm_val
[[1164   72]
 [  57 1145]]
```

We can get the `tn` , `fp` , `fn` and `tp` with NumPy ravel (https://numpy.org/doc/stable/reference/generated/numpy.ravel.html) method.

In [ ]:

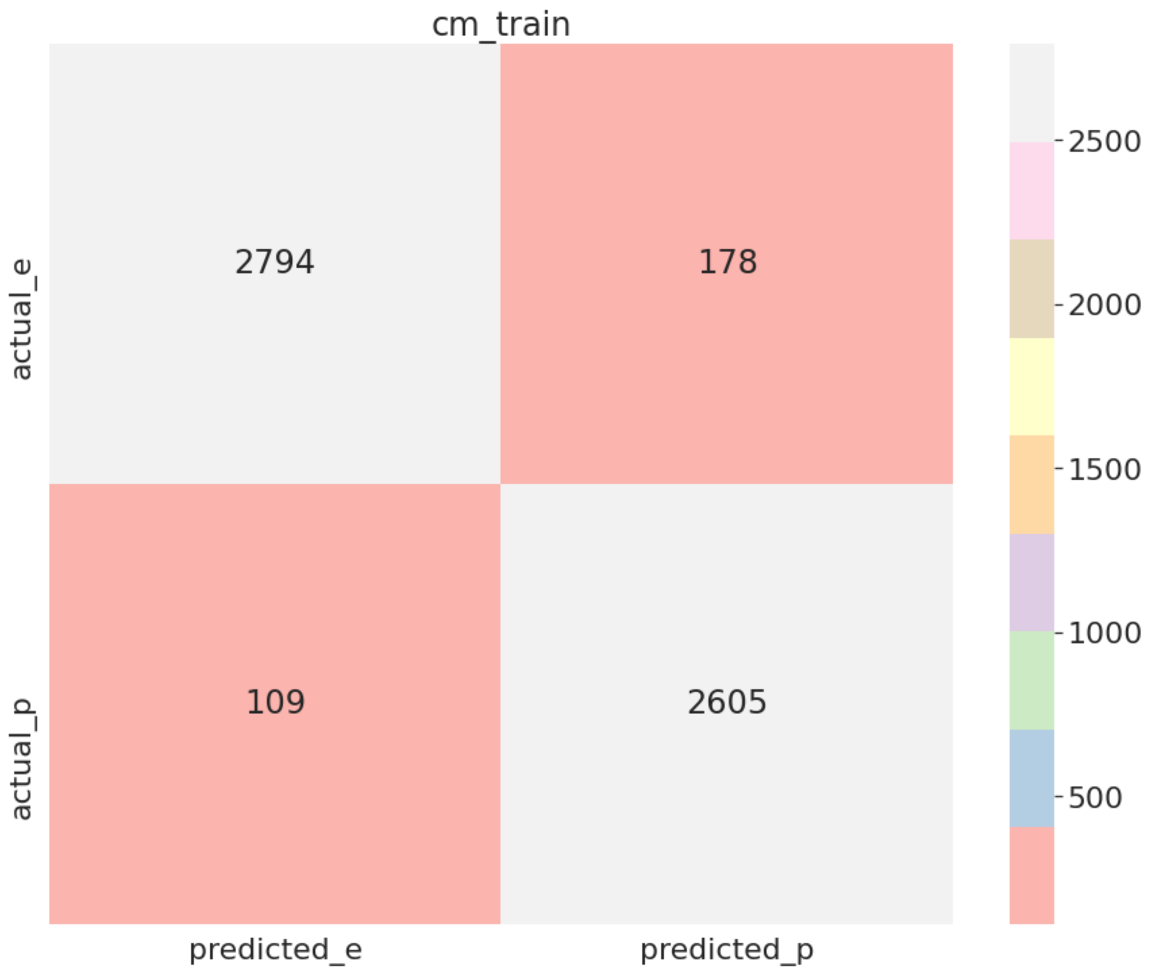
```
# show tn, fp, fn and tp for train and val
tn_train, fp_train, fn_train, tp_train = cm_train.ravel()
tn_val, fp_val, fn_val, tp_val = cm_val.ravel()

print(f'Train: TN {tn_train:4}, FP {fp_train:4}, FN {fn_train:4}, TP {tp_train:4}')
print(f'Val:   TN {tn_val:4}, FP {fp_val:4}, FN {fn_val:4}, TP {tp_val:4}')
```

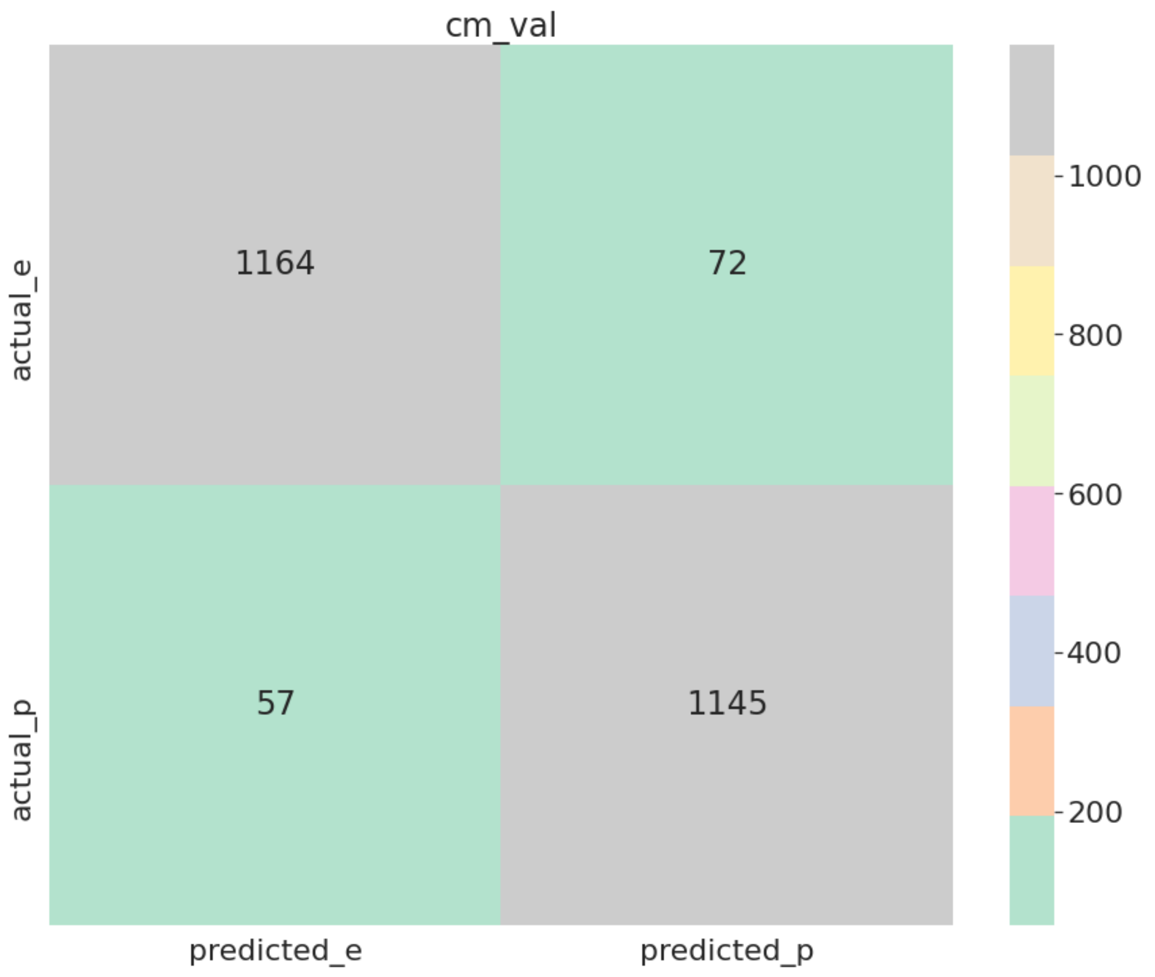
```
Train: TN 2794, FP  178, FN  109, TP 2605
Val:   TN 1164, FP   72, FN   57, TP 1145
```

We can plot the confusion matrix with Seaborn heatmap (https://seaborn.pydata.org/generated/seaborn.heatmap.html).

In [ ]:

```
# lot confusion matrix of train and val as heatmaps in seaborn
cm_train_df = pd.DataFrame(cm_train, index=['actual_e', 'actual_p'], columns=['predicte
d_e', 'predicted_p'])
cm_val_df = pd.DataFrame(cm_val, index=['actual_e', 'actual_p'], columns=['predicted_e'
, 'predicted_p'])

sns.set(font_scale=2)
plt.figure(figsize = (15,12))
fig = sns.heatmap(cm_train_df, annot=True, cmap=plt.cm.Pastel1, fmt='g')
fig.set_title("cm_train")
plt.show()
print()
plt.figure(figsize = (15,12))
fig = sns.heatmap(cm_val_df, annot=True, cmap=plt.cm.Pastel2, fmt='g')
fig.set_title("cm_val")
plt.show()
```


cm_train
actual_e
actual_p
predicted_e
predicted_p
2794
178
109
2605
2500
2000
1500
1000
500


cm_val
actual_e
actual_p
predicted_e
predicted_p
1164
72
57
1145
1000
800
600
400
200

We can calculate the basic KPI's.

In [ ]:

```
# calculate precision, recall, specificity, fpr and accuracy
def precision(cm):
    tn, fp, fn, tp = cm.ravel()
    return tp/(tp+fp)

def recall(cm):
    tn, fp, fn, tp = cm.ravel()
    return tp/(tp+fn)

def specificity(cm):
    tn, fp, fn, tp = cm.ravel()
    return tn/(tn+fp)

def fpr(cm):
    tn, fp, fn, tp = cm.ravel()
    return fp/(fp+tn)

def accuracy(cm):
    tn, fp, fn, tp = cm.ravel()
    return (tp+tn)/(tp+tn+fp+fn)

print(f'Train: precision {precision(cm_train):.2f}, recall {recall(cm_train):.2f}, spec
ificity {specificity(cm_train):.2f}, fpr {fpr(cm_train):.2f}, accuracy {accuracy(cm_tra
in):.2f}')
print(f'Val:   precision {precision(cm_val):.2f}, recall {recall(cm_val):.2f}, specific
ity {specificity(cm_val):.2f}, fpr {fpr(cm_val):.2f}, accuracy {accuracy(cm_val):.2f}')
```

```
Train: precision 0.94, recall 0.96, specificity 0.94, fpr 0.06, accuracy
0.95
Val:   precision 0.94, recall 0.95, specificity 0.94, fpr 0.06, accuracy
0.95
```

We can also use the previous methods to calculate the more complicated KPI's.

In [ ]:

```
# calculate f1 and balanced accuracy
def f1(cm):
    precision_calc = precision(cm)
    recall_calc = recall(cm)
    return (2*precision_calc*recall_calc)/(precision_calc+recall_calc)

def balanced_accuracy(cm):
    recall_calc = recall(cm)
    specificity_calc = specificity(cm)
    return (recall_calc+specificity_calc)/2

print(f'Train: f1 {f1(cm_train):.2f}, balanced_accuracy {balanced_accuracy(cm_train):.2
f}')
print(f'Val:   f1 {f1(cm_val):.2f}, balanced_accuracy {balanced_accuracy(cm_val):.2f}')
```

```
Train: f1 0.95, balanced_accuracy 0.95
Val:   f1 0.95, balanced_accuracy 0.95
```

Let's remember how the classifier makes its decision.
it calculates the confidence score for each of the labels and chooses the label with the highest confidence score.

	negative probability	positive probability	prediction
sample 1	0.1	0.9	positive
sample 2	0.7	0.3	negative
sample 3	0.4	0.6	positive

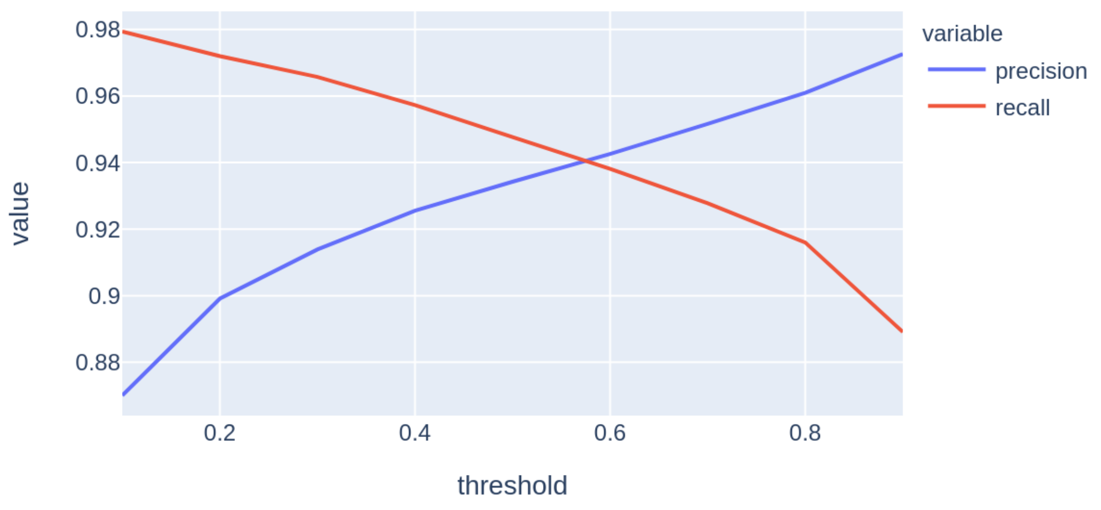
The sum of these scores for each sample is 1 (sum of all the confident scores of all the labels for this specific row).
In binary classification, we can replace these two scores with one confident score, the score of the positive label.

	positive probability	prediction
sample 1	0.9	positive
sample 2	0.3	negative
sample 3	0.6	positive

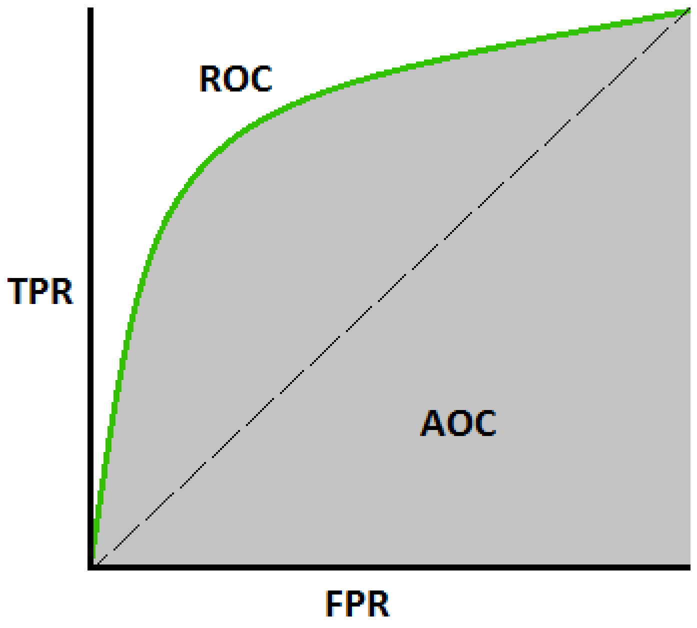
If this score is higher than 0.5, we know that the sample is predicted as positive.
If this score is lower than 0.5, we know that the sample is predicted as negative.
We can say that in this case, the threshold is 0.5.
If we make the threshold bigger, we will say that a sample is positive only if our confident score is high (less fp and more fn - high precision and low recall).
If we make the threshold smaller, we will say that a sample is positive even if our confident score is low (more fp and less fn - high recall and low precision).

In [ ]:

```
# plot graph of precision and recall as function of the threshold
def plot_precision_recall_by_threshold(labels, probabilities):
    precision_recall_df = pd.DataFrame(columns=['threshold', 'precision', 'recall'])
    for threshold in np.linspace(0.1, 1, 9, endpoint=False):
        preds = np.where(probabilities[:,1] > threshold, 'p', 'e')
        cm = confusion_matrix(labels, preds)
        precision_recall_df.loc[len(precision_recall_df)] = [threshold, precision(cm),
recall(cm)]
    fig = px.line(precision_recall_df, x='threshold', y=['precision', 'recall'])
    img_bytes = fig.to_image(format="png", width=600, height=350, scale=2)
    display(Image(img_bytes))

model_pipe_log = make_pipeline(oe, SGDClassifier(loss='log', random_state=1)).fit(X_tra
in, t_train)
y_proba_train = model_pipe_log.predict_proba(X_train)
y_proba_val = model_pipe_log.predict_proba(X_val)
plot_precision_recall_by_threshold(t_train, y_proba_train)
```



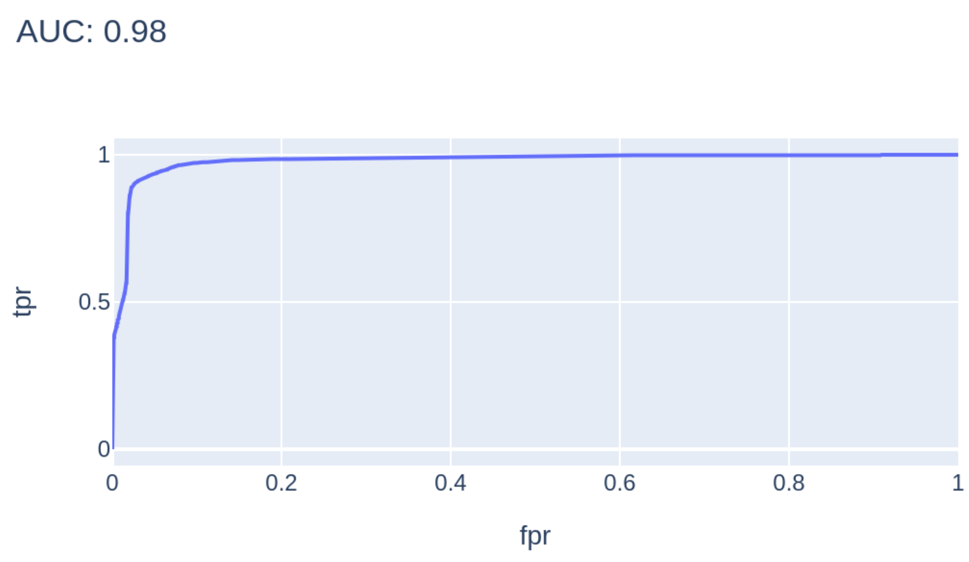
There is another graph that shows our model behavior based on thresholds.
This graph is called ROC (Receiver Operating Characteristic) (https://en.wikipedia.org/wiki/Receiver_operating_characteristic) and it is a graph of recall vs. FPR.



When the threshold is moving from right to left, the FPR is getting bigger (more fp) and the recall is getting bigger too (less fn).
We will say that the model is good if the recall is high even when the FPR is low.
We can make a score out of it and call it AUC (Area Under the Curve).
The bigger the AUC, the better the model performance.
Let's plot the ROC with Sciking-learn roc_curve (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html) and calculate the AUC with Scikit-learn roc_auc_score (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html#sklearn.metrics.roc_auc_score).

In [ ]:

```
# plot ROC curve and calculate AUC score
from sklearn.metrics import roc_curve, roc_auc_score

def plot_roc_curve(labels, probabilities):
    auc_score = roc_auc_score(labels, probabilities[:,1])
    fpr, tpr, thresholds = roc_curve(labels, probabilities[:,1], pos_label='p')
    roc_df = pd.DataFrame({'fpr': fpr, 'tpr': tpr, 'thresholds': thresholds})
    fig = px.line(roc_df, x='fpr', y='tpr', title=f'AUC: {auc_score:.2f}')
    img_bytes = fig.to_image(format="png", width=600, height=350, scale=2)
    display(Image(img_bytes))

model_pipe_log = make_pipeline(oe, SGDClassifier(loss='log', random_state=1)).fit(X_tra
in, t_train)
y_proba_train = model_pipe_log.predict_proba(X_train)
y_proba_val = model_pipe_log.predict_proba(X_val)
plot_roc_curve(t_train, y_proba_train)
```



More Information

Explanation about edible and poisonous mushrooms:
3 edible mushrooms that are easy to find – and how to avoid the poisonous ones (https://inhabitat.com/3-edible-mushrooms-that-are-easy-to-find-and-how-to-avoid-the-poisonous-ones/)

Explanation about mushroom species:
Edible and Poisonous Species of Coastal BC and the Pacific Northwest (https://www.zoology.ubc.ca/~biodiv/mushroom/index.html)

Pandas-Profiling Documentation:
Pandas Profiling Introduction (https://pandas-profiling.github.io/pandas-profiling/docs/master/rtd/pages/introduction.html)

Guide to categorical encoding:
Guide to Encoding Categorical Values in Python (https://pbpython.com/categorical-encoding.html)

Explanation about a few KPI's:
4 things you need to know about AI: accuracy, precision, recall and F1 scores (https://lawtomated.com/accuracy-precision-recall-and-f1-scores-for-lawyers/)

Explanation of a few types of means:
Arithmetic, Geometric, and Harmonic Means for Machine Learning (https://machinelearningmastery.com/arithmetic-geometric-and-harmonic-means-for-machine-learning/)