Python/Juypter Notebook Example

Kaggle Competition | Titanic Machine Learning from Disaster

This is a Colab Python IDE based jupyter notebook (previously called as ipython notebook).
You can see this as a lecture handout, where you can execute the code (cells) and see the output for yourselves.

Everything with a google account and inside your browser, no local installation.

Source:

- Courtesy by *Amrith Krishna* at CNeRG, CSE, IIT Kharagpur
- The Kaggle competition *Titanic Machine Learning from Disaster*
- The Competition Homepage.

Data:

- TITANIC_train.csv from the Kaggle Competition Homepage

Note: You must upload this dataset into your colab workspace before starting the program execution!

Content:

Author:

- dr.daniel benninger

History:

- v1, Febr 2023, dbe --- initial version for BINA FS23
-

1.The Problem

> The sinking of the RMS Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg, killing 1502 out of 2224 passengers and crew. This sensational tragedy shocked the international community and led to better safety regulations for ships.

> One of the reasons that the shipwreck led to such loss of life was that there were **not enough lifeboats** for the passengers and crew. Although there was some element of **luck** involved in surviving the sinking, **some groups of people were more likely to survive than others, such as women, children, and the upper-class**.

> In this contest, we ask you to complete the analysis of what sorts of people were likely to survive. In particular, we ask you to apply the tools of machine learning to predict which passengers survived the tragedy.

> This Kaggle Getting Started Competition provides an ideal starting place for people who may not have a lot of experience in data science and machine learning."

Setup/Check environment and import necessary libraries:

```
# [Optional] Mount GDrive and access files at your Google Drive directory
# from google.colab import drive
# drive.mount("/content/gdrive")
# %cd /gdrive

# [Standard] Check Colab Workspace directory
%cd sample_data/
%ls
```

```
[Errno 2] No such file or directory: 'sample_data/'
/content/sample_data
anscombe.json*              mnist_test.csv         TITANIC_test.csv
```

```
     california_housing_test.csv   mnist_train_small.csv
     california_housing_train.csv  README.md*
```

```
# import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

2. Data Handling

Let's read our data in using the famous pandas library

Note: Datafile need to be uploaded initially into the Colab file system/directory!

```
df = pd.read_csv("TITANIC_train.csv")
```

Show an overview of our data (loaded into a dataframe variable *df*):

```
df
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0
				Cumings, Mrs. John				

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

```
df.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	P
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.00

Let's take a look:

Above is a summary of our data contained in a `Pandas DataFrame`. Think of a `DataFrame` as a Python's super charged version of the workflow in an Excel table. As you can see the summary holds quite a bit of information. First, it lets us know we have 891 observations, or passengers, to analyze here:

```
Int64Index: 891 entries, 0 to 890
```

Next it shows us all of the columns in `DataFrame`. Each column tells us something about each of our observations, like their `name`, `sex` or `age`. These colunms are called a features of our dataset. You can think of the meaning of the words column and feature as interchangeable for this notebook.

After each feature it lets us know how many values it contains. While most of our features have complete data on every observation, like the `survived` feature here:

```
survived    891  non-null values
```

some are missing information, like the `age` feature:

```
age         714  non-null values
```

These missing values are represented as `NaN`s.

Take care of missing values:

The features `ticket` and `cabin` have many missing values and so can’t add much value to our analysis. To handle this we will drop them from the dataframe to preserve the integrity of our dataset.

To do that we'll use this line of code to drop the features entirely:

```
df = df.drop(['ticket','cabin'], axis=1)
```

While this line of code removes the `NaN` values from every remaining column / feature:

```
df = df.dropna()
```

Now we have a clean and tidy dataset that is ready for analysis. Because `.dropna()` removes an observation from our data even if it only has 1 `NaN` in one of the features, it would have removed most of our dataset if we had not dropped the `ticket` and `cabin` features first.

```
# Drop features with missing values
df = df.drop(['Ticket','Cabin'], axis=1)

# Remove NaN values
df = df.dropna()
```

For a detailed look at how to use pandas for data analysis, the best resource is Wes Mckinney's book. Additional interactive tutorials that cover all of the basics can be found here (they're free). If you still need to be convinced about the power of pandas check out this wirlwhind look at all that pandas can do.

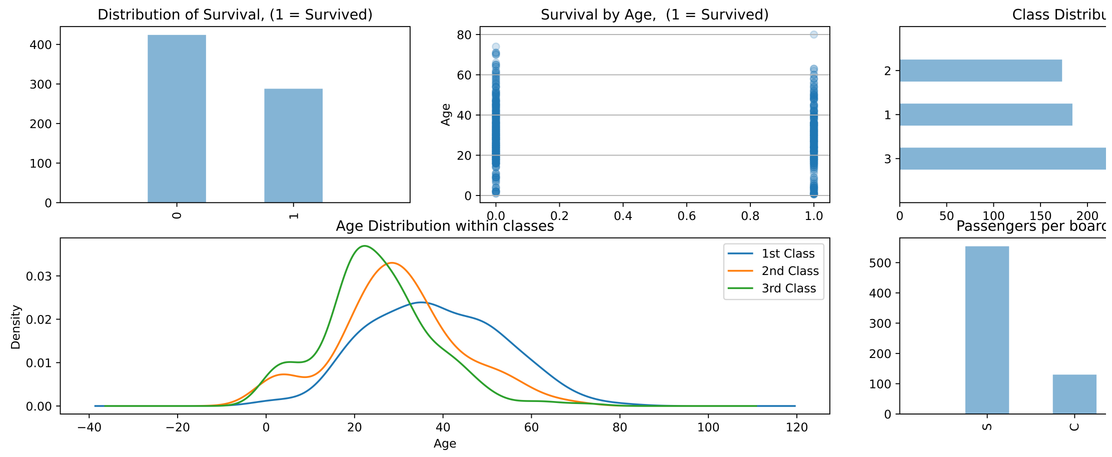
3. Data Visualization

Let's take a Look at our data graphically using matplotlib library:

```
# Specifies the (general) parameters of our graphs
fig = plt.figure(figsize=(18,6), dpi=1600)
alpha=alpha_scatterplot = 0.2
alpha_bar_chart = 0.55

# lets us plot many different shaped graphs together in a 2x3 grid
ax1 = plt.subplot2grid((2,3),(0,0))

# 1) plots a bar graph of those Passengers who surived vs those who did not.
```

```
df.Survived.value_counts().plot(kind='bar', alpha=alpha_bar_chart)
# this nicely sets the margins in matplotlib to deal with a recent bug 1.3.1
ax1.set_xlim(-1, 2)
# puts a title on our graph
plt.title("Distribution of Survival, (1 = Survived)")

# 2) plots a scatter of Passenger age versus who survival or not.
plt.subplot2grid((2,3),(0,1))
plt.scatter(df.Survived, df.Age, alpha=alpha_scatterplot)
# sets the y axis lable
plt.ylabel("Age")
# formats the grid line style of our graphs
plt.grid(b=True, which='major', axis='y')
plt.title("Survival by Age,  (1 = Survived)")

# 3) Passengers distribution across classes
ax3 = plt.subplot2grid((2,3),(0,2))
df.Pclass.value_counts().plot(kind="barh", alpha=alpha_bar_chart)
ax3.set_ylim(-1, len(df.Pclass.value_counts()))
plt.title("Class Distribution")

# 4) plots a kernel density estimate of the subset of the 1st class Passengers
plt.subplot2grid((2,3),(1,0), colspan=2)
df.Age[df.Pclass == 1].plot(kind='kde')
df.Age[df.Pclass == 2].plot(kind='kde')
df.Age[df.Pclass == 3].plot(kind='kde')
 # plots an axis lable
plt.xlabel("Age")
plt.title("Age Distribution within classes")
# sets our legend for our graph.
plt.legend(('1st Class', '2nd Class','3rd Class'),loc='best')

# 5) Passengers per boarding location
ax5 = plt.subplot2grid((2,3),(1,2))
df.Embarked.value_counts().plot(kind='bar', alpha=alpha_bar_chart)
ax5.set_xlim(-1, len(df.Embarked.value_counts()))
# specifies the parameters of our graphs

# Overall graph title
plt.title("Passengers per boarding location")
```

```
Text(0.5, 1.0, 'Passengers per boarding location')
```



Exploratory Visualization:

The point of this competition is to predict if an individual will survive based on the features in the data like:

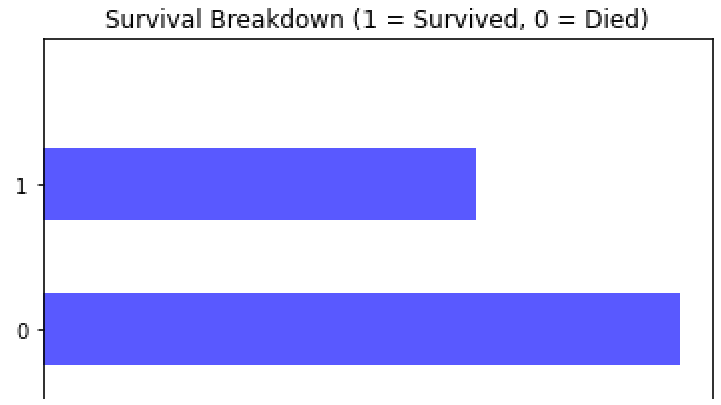
- Traveling Class (called pclass in the data)
- Sex
- Age
- Fare Price

Let’s see if we can gain a better understanding of who survived and died.

First let’s plot a bar graph of those who Survived Vs. Those who did not.

```
plt.figure(figsize=(6,4))
fig, ax = plt.subplots()
df.Survived.value_counts().plot(kind='barh', color="blue", alpha=.65)
ax.set_ylim(-1, len(df.Survived.value_counts()))

# General title
plt.title("Survival Breakdown (1 = Survived, 0 = Died)")
```

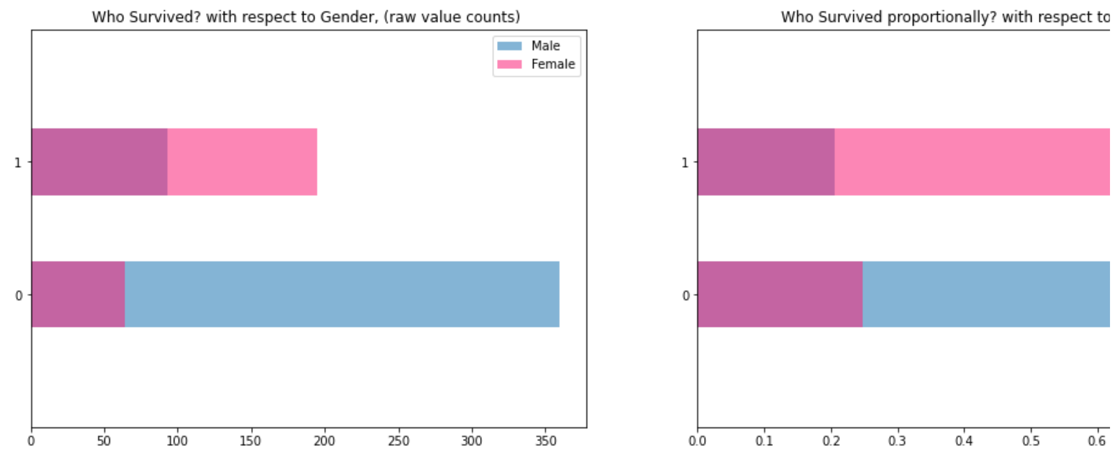
```
Text(0.5, 1.0, 'Survival Breakdown (1 = Survived, 0 = Died)')
<Figure size 432x288 with 0 Axes>
```



Now let’s tease more structure out of the data. Let’s break the previous graph down by gender

```
fig = plt.figure(figsize=(18,6))

#create a plot of two subsets, male and female, of the survived variable.
#After we do that we call value_counts() so it can be easily plotted as a bar
#'barh' is just a horizontal bar graph
df_male = df.Survived[df.Sex == 'male'].value_counts().sort_index()
df_female = df.Survived[df.Sex == 'female'].value_counts().sort_index()

# Subplot 1
ax1 = fig.add_subplot(121)
df_male.plot(kind='barh',label='Male', alpha=0.55)
df_female.plot(kind='barh', color='#FA2379',label='Female', alpha=0.55)
plt.title("Who Survived? with respect to Gender, (raw value counts) "); plt.le
ax1.set_ylim(-1, 2)

# Subplot 2
# adjust graph to display the proportions of survival by gender
ax2 = fig.add_subplot(122)
(df_male/float(df_male.sum())).plot(kind='barh',label='Male', alpha=0.55)
(df_female/float(df_female.sum())).plot(kind='barh', color='#FA2379',label='Fe
plt.title("Who Survived proportionally? with respect to Gender"); plt.legend(l

ax2.set_ylim(-1, 2)
```

```
(-1.0, 2.0)
```



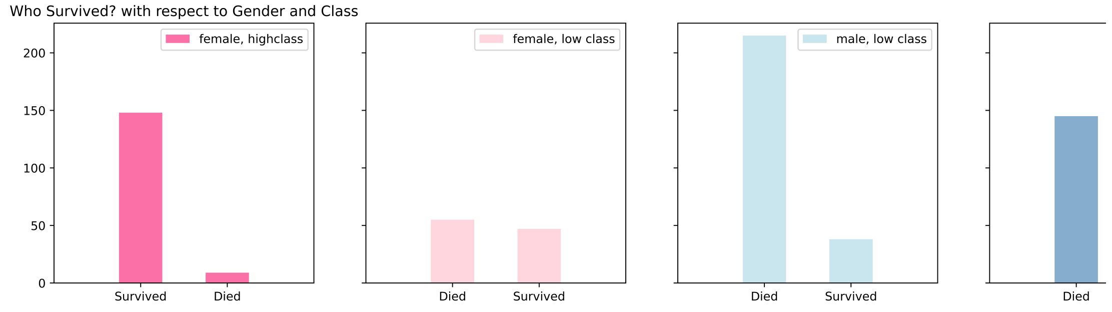
Here it’s clear that although more men died and survived in raw value counts, females had a greater survival rate proportionally (approx. 25%), than men (approx. 20%)

Great! But let’s go down even further:

Can we capture more of the structure by using Pclass? Here we will bucket classes as lowest class or any of the high classes (classes 1 - 2). 3 is lowest class. Let’s break it down by Gender and what Class they were traveling in.

```
fig = plt.figure(figsize=(18,4), dpi=1600)
alpha_level = 0.65

# building on the previous code, here we create an additional subset with in t
# we created for the survived variable. I know, thats a lot of subsets. After
# value_counts() so it it can be easily plotted as a bar graph. this is repeat
# class pair.

# Subplot 1
ax1=fig.add_subplot(141)
female_highclass = df.Survived[df.Sex == 'female'][df.Pclass != 3].value_count
female_highclass.plot(kind='bar', label='female, highclass', color='#FA2479',
ax1.set_xticklabels(["Survived", "Died"], rotation=0)
ax1.set_xlim(-1, len(female_highclass))
plt.title("Who Survived? with respect to Gender and Class"); plt.legend(loc='b

# Subplot 2
ax2=fig.add_subplot(142, sharey=ax1)
female_lowclass = df.Survived[df.Sex == 'female'][df.Pclass == 3].value_counts
female_lowclass.plot(kind='bar', label='female, low class', color='pink', alph
ax2.set_xticklabels(["Died","Survived"], rotation=0)
ax2.set_xlim(-1, len(female_lowclass))
plt.legend(loc='best')

# Subplot 3
ax3=fig.add_subplot(143, sharey=ax1)
male_lowclass = df.Survived[df.Sex == 'male'][df.Pclass == 3].value_counts()
male_lowclass.plot(kind='bar', label='male, low class',color='lightblue', alph
ax3.set_xticklabels(["Died","Survived"], rotation=0)
ax3.set_xlim(-1, len(male_lowclass))
plt.legend(loc='best')

# Subplot 4
ax4=fig.add_subplot(144, sharey=ax1)
male_highclass = df.Survived[df.Sex == 'male'][df.Pclass != 3].value_counts()
male_highclass.plot(kind='bar', label='male, highclass', alpha=alpha_level, co
ax4.set_xticklabels(["Died","Survived"], rotation=0)
ax4.set_xlim(-1, len(male_highclass))
plt.legend(loc='best')
```

```
<matplotlib.legend.Legend at 0x7f69fe084280>
```



Awesome! Now we have a lot more information on who survived and died in the tragedy. With this deeper understanding, we are better equipped to create better more insightful models. This is a typical process in interactive data analysis. First you start small and understand the most basic relationships and slowly increment the complexity of your analysis as you discover more and more about the data you’re working with.

(Disclaimer) I've done my best to make the plotting code readable and intuitive, but if you’re looking for a more detailed look on how to start plotting in matplotlib, check out this beautiful notebook here.