Double-click (or enter) to edit

Short HW1 - Preparing for the course

Useful python libraries, Probability, and Linear algebera

Instructions

General

- **First, don't panic!**
  - This assignment seems longer than it actually is.
  - In the first part, you are mostly required to run *existing* code and complete short python commands here and there.
  - In the two other parts you need to answer overall 4 analytic questions.
  - **Note**: The other 3 *short* assignments will be shorter and will *not* require programming.
- **Individually or in pairs?** Individually only.
- **Where to ask?** In the Piazza forum.
- **How to submit?** In the webcourse.
- **What to submit?** A pdf file with the completed jupyter notebook (including the code, plots and other outputs) and the answers to the probability/algebra questions (Hebrew or English are both fine).
  Or two separate pdf files in a zip file. All submitted files should contain **your ID number** in their names.
- **When to submit?** Sunday 28.01.2024 at 23:59.

Specific

- First part: get familiar with popular python libraries useful for machine learning and data science. We will use these libraries heavily throughout the major programming assignments.
  - You should read the instructions and run the code blocks sequentially.
    In 10 places you are reqired to complete missing python commands or answer short questions (look for the TODO comments, or notations like **(T3)** etc.). Try to understand the flow of this document and the code you run.
  - Start by loading the provided jupyter notebook file (*Short_HW1.ipynb*) to Google Colab, which is a very convenient online tool for running python scripts combined with text, visual plots, and more.
  - Alternatively, you can install jupyter locally on your computer and run the provided notebook there.
- Second and third parts: questions on probability and linear algebra to refresh your memory and prepare for the rest of this course.
  The questions are mostly analytic but also require completing and running simple code blocks in the jupyter notebook.
  - Forgot your linear algebra? Try watching Essence of LA or reading The Matrix Cookbook.
  - Forgot your probability? Try reading Probability Theory Review for Machine Learning.
    - Correction: In 3.2 it says that $X \perp Y \implies \mathrm{Var}(X+Y) = \mathrm{Var}(X)\mathrm{Var}(Y)$ but it should say $X \perp Y \implies \mathrm{Var}(X+Y) = \mathrm{Var}(X) + \mathrm{Var}(Y)$.

Important: How to submit the notebook's output?

You should only submit PDF file(s). In the print dialog of your browser, you can choose to `Save as PDF`. However, notice that some of the outputs may be cropped (become invisible), which can harm your grade.

To prevent this from happening, tune the "scale" of the printed file, to fit in the *entire* output. For instance, in Chrome you should lower the value in `More settings->Scale->Custom` to contain the entire output (50%~ often work well).

Good luck!

What is pandas?

Python library for Data manipulation and Analysis

- Provide expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive.
- Aims to be the fundamental high-level building block for doing practical, real world data analysis in Python.
- Built on top of NumPy and is intended to integrate well within a scientific computing.
- Inspired by R and Excel.

Pandas is well suited for many different kinds of data:

- **Tabular data** with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet
- Ordered and unordered (not necessarily fixed-frequency) **time series data**.
- **Arbitrary matrix data** (homogeneously typed or heterogeneous) with row and column labels
- Any other form of observational / statistical data sets (can be unlabeled)

Two primary data structures

- **Series** (1-dimensional) – Similar to a column in Excel’s spreadsheet
- **Data Frame** (2-dimensional) – Similar to R’s data frame

A few of the things that Pandas does well

- Easy handling of **missing data** (represented as NaN)
- Automatic and explicit **data alignment**
- Read and Analyze **CSV** , Excel Sheets Easily
- Operations
- Filtering, Group By, Merging, Slicing and Dicing, Pivoting and Reshaping
- Plotting graphs

Pandas is very useful for interactive data exploration at the data preparation stage of a project

The offical guide to Pandas can be found here

Pandas Objects

```
import pandas as pd
import numpy as np
```

Series is like a column in a spreadsheet.

```
s = pd.Series([1,3.2,np.nan,'string'])
s
```

```
0         1
1       3.2
2       NaN
3    string
dtype: object
```

DataFrame is like a spreadsheet – a dictionary of Series objects

```
data = [['ABC', -3.5, 0.01], ['ABC', -2.3, 0.12], ['DEF', 1.8, 0.03],
['DEF', 3.7, 0.01], ['GHI', 0.04, 0.43], ['GHI', -0.1, 0.67]]

df = pd.DataFrame(data, columns=['gene', 'log2FC', 'pval'])

df
```

	gene	log2FC	pval
0	ABC	-3.50	0.01
1	ABC	-2.30	0.12
2	DEF	1.80	0.03
3	DEF	3.70	0.01
4	GHI	0.04	0.43
5	GHI	-0.10	0.67

Input and Output

How do you get data into and out of Pandas as spreadsheets?

- Pandas can work with XLS or XLSX files.
- Can also work with CSV (comma separated values) file
- CSV stores plain text in a tabular form
- CSV files may have a header
- You can use a variety of different field delimiters (rather than a 'comma'). Check which delimiter your file is using before import!

Import to Pandas

```
df = pd.read_csv('data.csv', sep='\t', header=0)
```

For Excel files, it's the same thing but with read_excel

Export to text file

```
df.to_csv('data.csv', sep='\t', header=True, index=False)
```

The values of header and index depend on if you want to print the column and/or row names

Case Study – Analyzing Titanic Passengers Data

```
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import pandas as pd
import os
```

```
#set your working_dir
working_dir = os.path.join(os.getcwd(), 'titanic')

url_base = 'https://github.com/Currie32/Titanic-Kaggle-Competition/raw/master/{}.csv'
train_url = url_base.format('train')
test_url = url_base.format('test')

# For .read_csv, always use header=0 when you know row 0 is the header row
train = pd.read_csv(train_url, header=0)
test = pd.read_csv(test_url, header=0)
# You can also load a csv file from a local file rather than a URL
```

(T1) Use `pandas.DataFrame.head` to display the top 6 rows of the `train` table

```
# TODO: print the top 6 rows of the table
df_head = pd.DataFrame(train).head(6)

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
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q

VARIABLE DESCRIPTIONS:

Survived - 0 = No; 1 = Yes
Age - Passenger's age
Pclass - Passenger Class (1 = 1st; 2 = 2nd; 3 = 3rd)
SibSp - Number of Siblings/Spouses Aboard
Parch - Number of Parents/Children Aboard
Ticket - Ticket Number
Fare - Passenger Fare
Cabin - Cabin ID
Embarked - Port of Embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)

```
train.columns
```

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

Understanding the data (Summarizations)

```
train.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
```

```
train.shape
```

```
(891, 12)
```

```
# Count values of 'Survived'
train.Survived.value_counts()
```

```
0    549
1    342
Name: Survived, dtype: int64
```

```
# Calculate the mean fare price
train.Fare.mean()
```

```
32.204207968574636
```

```
# General statistics of the dataframe
train.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Selection examples

Selecting columns

```
# Selection is very similar to standard Python selection
df1 = train[["Name", "Sex", "Age", "Survived"]]
df1.head() # default is 5 rows
```

	Name	Sex	Age	Survived
0	Braund, Mr. Owen Harris	male	22.0	0
1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1
2	Heikkinen, Miss. Laina	female	26.0	1
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1
4	Allen, Mr. William Henry	male	35.0	0

Selecting rows

```
df1[10:15]
```

	Name	Sex	Age	Survived
10	Sandstrom, Miss. Marguerite Rut	female	4.0	1
11	Bonnell, Miss. Elizabeth	female	58.0	1
12	Saundercock, Mr. William Henry	male	20.0	0
13	Andersson, Mr. Anders Johan	male	39.0	0
14	Vestrom, Miss. Hulda Amanda Adolfina	female	14.0	0

Filtering Examples

Filtering with one condition

```
# Filtering allows you to create masks given some conditions
df1.Sex == 'female'
```

```
0      False
1       True
2       True
3       True
4      False
       ...  
886    False
887     True
888     True
889    False
890    False
Name: Sex, Length: 891, dtype: bool
```

```
onlyFemale = df1[df1.Sex == 'female']
onlyFemale.head()
```

	Name	Sex	Age	Survived
1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1
2	Heikkinen, Miss. Laina	female	26.0	1
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1
8	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	1
9	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1

Filtering with multiple conditions

(T2) Alter the following command so `adultFemales` will contain only females whose age is 18 and above.
You need to filter using a **single** mask with multiple conditions (google it!), i.e., without creating any temporary dataframes.

Additionally, update the `survivalRate` variable to show the correct rate.

```
# TODO: update the mask
adultFemales = df1[(df1.Sex == 'female') & (df1.Age >= 18)]

# TODO: Update the survival rate
survivalRate = adultFemales.Survived.mean()
print("The survival rate of adult females was: {:.2f}%".format(survivalRate * 100))
```

```
The survival rate of adult females was: 77.18%
```

Aggregating

Pandas allows you to aggregate and display different views of your data.

```
# Calculate the mean fare grouped by Pclass and Sex
df2 = train.groupby(['Pclass', 'Sex']).Fare.agg(np.mean)
df2
```

```
Pclass  Sex
1       female    106.125798
        male       67.226127
2       female     21.970121
        male       19.741782
3       female     16.118810
        male       12.661633
Name: Fare, dtype: float64
```

```
pd.pivot_table(train, index=['Pclass'], values=['Survived'], aggfunc='count')
```

Pclass	Survived
1	216
2	184
3	491

The following table shows the survival rates for each combination of passenger class and sex.
(T3) Add a column showing the mean **age** for such a combination.

```
# TODO: Also show the mean age per group
pd.pivot_table(train, index=['Pclass', 'Sex'], values=['Survived', 'Age'], aggfunc='mean')
```

Pclass	Sex	Age	Survived
1	female	34.611765	0.968085
	male	41.281386	0.368852
2	female	28.722973	0.921053
	male	30.740707	0.157407
3	female	21.750000	0.500000
	male	26.507589	0.135447

(T4) Use this question on stackoverflow, to find the mean survival rate for ages 0-10, 10-20, etc.).

Hint: the first row should roughly look like this:

```
             Age     Survived
Age
(0, 10]    4.268281    0.593750
```

```
# TODO: find the mean survival rate per age group
# Extracting age groups in intervals of 10 based on the observed age range
# in the result of train.describe()
# train.Age.min() = 0 , train.Age.max() = 80
ageGroups = np.arange(0, 81, 10)

survivalPerAgeGroup = train.groupby(pd.cut(train["Age"],
                            ageGroups))[['Age', 'Survived']].agg(np.mean)

survivalPerAgeGroup
```

Age	Age	Survived
(0, 10]	4.268281	0.593750
(10, 20]	17.317391	0.382609
(20, 30]	25.423913	0.365217
(30, 40]	35.051613	0.445161
(40, 50]	45.372093	0.383721
(50, 60]	54.892857	0.404762
(60, 70]	63.882353	0.235294
(70, 80]	73.300000	0.200000

```
type(train.groupby(pd.cut(train.Age, ageGroups)).Survived.mean())
```

```
pandas.core.series.Series
```

Filling missing data (data imputation)

Note that some passenger do not have age data.

```
# The first .shape[0] is used to get the number of rows in the resulting DataFrame,
# which corresponds to the count of passengers with missing age values.
print("{} out of {} passengers do not have a recorded age".format(df1[df1.Age.isna()].shape[0], df1.shape[0]))
```

```
177 out of 891 passengers do not have a recorded age
```

```
df1[df1.Age.isna()].head()
```

	Name	Sex	Age	Survived
5	Moran, Mr. James	male	NaN	0
17	Williams, Mr. Charles Eugene	male	NaN	1
19	Masselmani, Mrs. Fatima	female	NaN	1
26	Emir, Mr. Farred Chehab	male	NaN	0
28	O'Dwyer, Miss. Ellen "Nellie"	female	NaN	1

Let's see the statistics of the column **before** the imputation.

```
df1.Age.describe()
```

```
count    714.000000
mean      29.699118
std       14.526497
min        0.420000
25%       20.125000
50%       28.000000
75%       38.000000
max       80.000000
Name: Age, dtype: float64
```

Read about `pandas.Series.fillna`.

(T5) Replace the missing ages `df1` with the general age *median*, and insert the result into variable `filledDf` (the original `df1` should be left unchanged).

```
# TODO : Fill the missing values
age_median = df1['Age'].median()
values = {"Age": age_median}
filledDf = df1
filledDf = filledDf.fillna(values)

# check
# print("{} out of {} passengers do not have a recorded age".format(df1[df1.Age.isna()].shape[0], df1.shape[0]))
```

```
print("{} out of {} passengers do not have a recorded age".format(filledDf[filledDf.Age.isna()].shape[0], filledDf.shape[0]))
```

```
0 out of 891 passengers do not have a recorded age
```

Let's see the statistics of the column **after** the imputation.

```
filledDf.Age.describe()
```

```
count    891.000000
mean      29.361582
std       13.019697
min        0.420000
25%       22.000000
50%       28.000000
75%       35.000000
max       80.000000
Name: Age, dtype: float64
```

(T6) Answer below: which statistics changed, and which did not? Why? (explain briefly, no need to be very formal.)

Answer: I examined each row :

The 'count' in the 'Age' represents the number of non-missing (non-NaN) values. Upon reviewing the print result, it is evident that 177 out of 891 passengers in df1 lack a recorded age. Subsequently, in filledDf, we replaced all NaN values in the 'Age' column with the median. Consequently, the entire column is now filled, comprising 891 non-NaN values, corresponding to the total number of rows.

The 'mean' value has been affected by the presence of missing values. Initially, in df1, the mean is calculated without considering these missing values. Subsequently, in filledDf, where missing values are filled, the mean() function considers all values, leading to a different average. This change is observable in the results, as the mean is now lower due to the inclusion of values equal to 28. The original mean in df1 was 29.699118

The standard deviation ('std') functions is a measure of the expected variation of a random variable around its mean. A low standard deviation suggests that values are typically close to the mean. In the context of filling missing values, when we introduce additional data, such as setting missing values to 28 (close to the mean), the overall distribution becomes more concentrated around the mean. Consequently, the standard deviation in filledDf is lower compared to its value before filling missing values.

The 'Min' and 'Max' values remain unchanged filledDf. This is expected since we added values within the existing range, the 'median' value are larger than the minimum and smaller than the maximum originally present in the 'Age' column

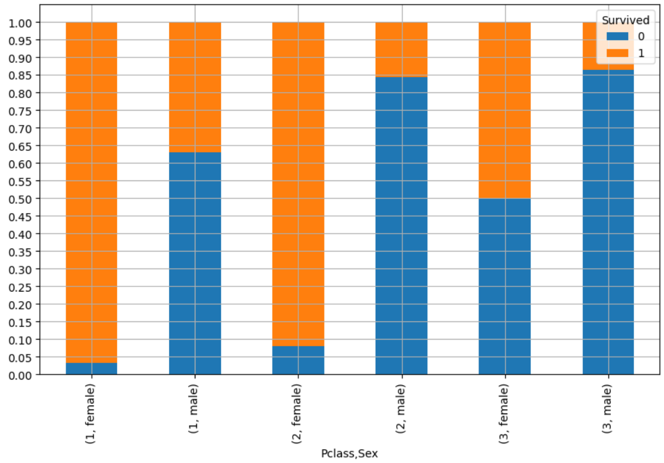
As observed in the 'count' value, missing values were not considered in the count in '25%', '50%', '75%' values, In filledDf now incorporates more values, especially around the median (50% percentile). As a result, the '25%' value, representing the age below which 25% of the values fall, has increased. Conversely, the '75%' value, representing the age below which 75% of the values fall, has decreased. The '50%' value, which corresponds to the median, remains unchanged since the median itself has not changed, given that we added values equal to the median.

In summary: the 'count,' 'mean,' 'std,' '25%,' and '50%' values have changed , while 'min,' 'max,' and '50%' values remain unchanged.

Plotting

Basic plotting in pandas is pretty straightforward

```
new_plot = pd.crosstab([train.Pclass, train.Sex], train.Survived, normalize="index")
new_plot.plot(kind='bar', stacked=True, grid=False, figsize=(10,6))
plt.yticks(np.linspace(0,1,21))
plt.grid()
```



(T7) Answer below: which group (class × sex) had the best survival rate? Which had the worst?

Answer:

The plot suggests a higher survival rate for Group 1 (1, female), as evident from the increased prominence of the orange color in that column.
This is attributed to the fact that the orange color signifies a '1' in the 'Survived' category

for the same reason the worst survival rate is (3,male).

What is Matplotlib

A 2D plotting library which produces publication quality figures.

- Can be used in python scripts, the python and IPython shell, web application servers, and more …
- Can be used to generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc.
- For simple plotting, pyplot provides a MATLAB-like interface
- For power users, a full control via OO interface or via a set of functions

There are several Matplotlib add-on toolkits

- Projection and mapping toolkits basemap and cartopy.
- Interactive plots in web browsers using Bokeh.
- Higher level interface with updated visualizations Seaborn.

Matplotlib is available at www.matplotlib.org

```
import matplotlib.pyplot as plt
import numpy as np
```

Line Plots

The following code plots the survival rate per age group (computed above, before the imputation).

(T8) Use the matplotlib documentation to add a grid and suitable axis labels to the following plot.

```
plt.plot(survivalPerAgeGroup.Age, survivalPerAgeGroup.Survived)
_ = plt.title("Survival per age group")
# TODO : Update the plot as required.
plt.grid(True)
plt.xlabel('Age Groups')
plt.ylabel('Survival Rate')
```

```
Text(0, 0.5, 'Survival Rate')
```


```
survivalPerAgeGroup
```

Age	Age	Survived
(0, 10]	4.268281	0.593750
(10, 20]	17.317391	0.382609
(20, 30]	25.423913	0.365217
(30, 40]	35.051613	0.445161
(40, 50]	45.372093	0.383721
(50, 60]	54.892857	0.404762
(60, 70]	63.882353	0.235294
(70, 80]	73.300000	0.200000

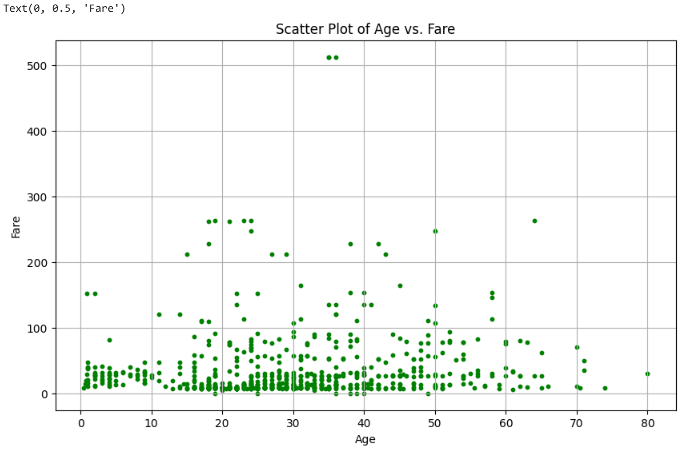
Scatter plots

(T9) Alter the matplotlib.pyplot.scatter command, so that the scattered dots will be `green`, and their size will be `10`.

Also, add a grid and suitable axis labels.

```
# TODO : Update the plot as required.
plt.figure(figsize=(10,6))
plt.scatter(train.Age, train.Fare, color='green', s=10)
_ = plt.title("Scatter Plot of Age vs. Fare")
plt.grid(True)
plt.xlabel('Age')
plt.ylabel('Fare')
```

```
Text(0, 0.5, 'Fare')
```



(T10) Answer below: approximately how old are the two highest paying passengers?

Answer:

The two passengers who made the highest payments stand out as the prominent green dots at the top of the scatter plot. these individuals fall within the age range of 30 to 40, approximately around 35 years old.

Probability refresher

Q1 - Variance of empirical mean

Let $X_1, \ldots, X_m$ be i.i.d random variables with mean $\mathbb{E}[X_i] = \mu$ and variance $\mathrm{Var}(X_i) = \sigma^2$.
We would like to "guess", or more formally, estimate (לשערך), the mean μ from the observations $x_1, \ldots, x_m$.
We use the empirical mean $\overline{X} = \frac{1}{m}\sum_i X_i$ as an estimator for the unknown mean μ. Notice that $\overline{X}$ is itself a random variable.
Note: The instantiation of $\overline{X}$ is usually denoted by $\hat{\mu} = \frac{1}{m}\sum_i x_i$, but this is currently out of scope.

1. Express analytically the expectation of $\overline{X}$.
   since **Answer**: $\mathbb{E}\left[\overline{X}\right] = \mathbb{E}\left[\frac{1}{m}\sum_i X_i\right] = \mathrm{TODO}.$

   X_i Using in the linearity of expectation:

   $$E[\bar{X}] = \frac{1}{m}\sum_i E[X_i] = \frac{1}{m}\cdot m\cdot\mu = \mu$$

   Therefore, $E[\bar{X}] = \mu$.

2. Express analytically the variance of $\overline{X}$.
   Answer: $\mathrm{Var}\left[\overline{X}\right] = \mathrm{TODO}.$

   X_i are independent random variavles(i.i.d) then the variance of their sum is equal to the sum of their variaces, so if the variance equal to σ^2:

   $$Var[\bar{X}] = \frac{1}{m^2}\sum_i Var[X_i] = \frac{1}{m^2}\cdot m\cdot\sigma^2 = \frac{\sigma^2}{m}$$

You will now verify the expression you wrote for the variance.
We assume $\forall i : X_i \sim \mathcal{N}(0,1)$.
We compute the empirical mean's variances for sample sizes $m = 1, \ldots, 30$.
For each sample size m, we sample m normal variables and compute their empirical mean. We repeat this step 50 times, and compute the variance of the empirical means (for each m).

3 . Complete the code blocks below according to the instructions and verify that your analytic function of the empirical mean's variance against as a function of m suits the empirical findings.

```
all_sample_sizes = range(1, 31)
repeats_per_size = 50

allVariances = []

for m in all_sample_sizes:
    empiricalMeans = []

    for _ in range(repeats_per_size):
        # Random m examples and compute their empirical mean
        X = np.random.randn(m)
        empiricalMeans.append(np.mean(X))

    # TODO: Using numpy, compute the variance of the empirical means that are in
    # the `empiricalMeans` list (you can google the numpy function for variance)
    variance = np.var(empiricalMeans)

    allVariances.append(variance)
```

Complete the following computation of the anayltic variance (according to the your answers above). You can try to use simple arithmetic operations between an `np.array` and a scalar, and see what happens! (for instance, `2 * np.array(all_sample_sizes)`.)

```
# TODO: compute the analytic variance
# (the current command wrongfully sets the variance of an empirical mean
```

```
#  of a sample with m variables simply as 2*m)
# X_i is a normally distributed variable so sigma^2 = 1
# analyticVariance =sigma^2 / all_sample_sizes
analyticVariance = 1 / np.array(all_sample_sizes).astype(float)
```

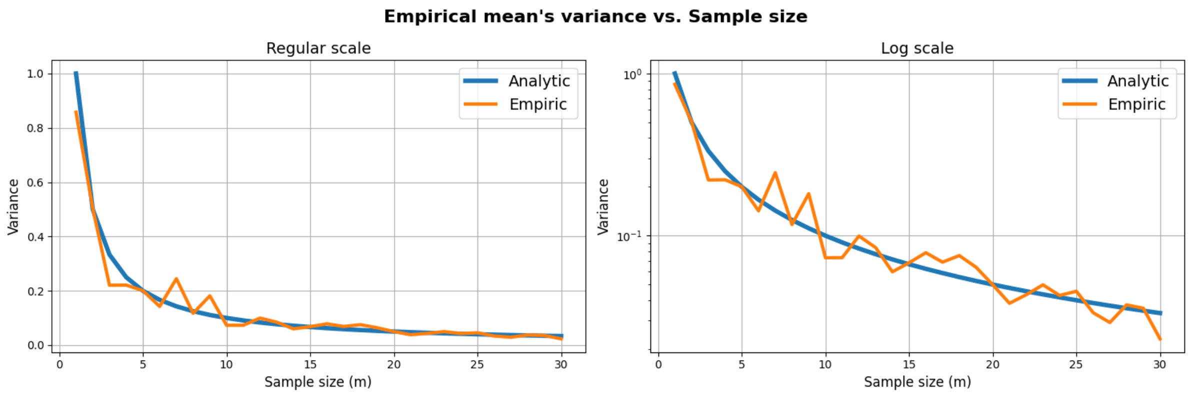
The following code plots the results from the above code. **Do not** edit it, only run it and make sure that the figures make sense.

```
fig, axes = plt.subplots(1,2, figsize=(15,5))
axes[0].plot(all_sample_sizes, analyticVariance, label="Analytic", linewidth=4)
axes[0].plot(all_sample_sizes, allVariances, label="Empiric", linewidth=3)
axes[0].grid()
axes[0].legend(fontsize=14)
axes[0].set_title("Regular scale", fontsize=14)
axes[0].set_xlabel("Sample size (m)", fontsize=12)
axes[0].set_ylabel("Variance", fontsize=12)

axes[1].semilogy(all_sample_sizes, analyticVariance, label="Analytic", linewidth=4)
axes[1].semilogy(all_sample_sizes, allVariances, label="Empiric", linewidth=3)
axes[1].grid()
axes[1].legend(fontsize=14)
axes[1].set_title("Log scale", fontsize=14)
axes[1].set_xlabel("Sample size (m)", fontsize=12)
axes[1].set_ylabel("Variance", fontsize=12)

_ = plt.suptitle("Empirical mean's variance vs. Sample size",
            fontsize=16, fontweight="bold")

plt.tight_layout()
```



Reminder - Hoeffding's Inequality

Let $\theta_1, \ldots, \theta_m$ be i.i.d random variables with mean $\mathbb{E}[\theta_i] = \mu$.

Additionally, assume all variables are bound in $[a, b]$ such that $\Pr[a \le \theta_i \le b] = 1$.

Then, for any $\epsilon > 0$, the empirical mean $\bar{\theta}(m) = \frac{1}{m}\sum_i \theta_i$ holds:

$$\Pr\left[\left|\bar{\theta}(m) - \mu\right| > \epsilon\right] \le 2\exp\left\{-\frac{2m\epsilon^2}{(b-a)^2}\right\}.$$

Q2 - Identical coins and the Hoeffding bound

We toss $m \in \mathbb{N}$ identical coins, each coin 40 times.
All coins have the same *unknown* probability of showing "heads", denoted by $p \in (0, 1)$.
Let θ_i be the (observed) number of times the i-th coin showed "heads".

1. What is the distribution of each θ_i?
   Answer: $\theta_i \sim \text{TODO}$.

   The variable θ_i represents the number of times a specific coin shows "heads" in a series of 40 tosses. the distribution of θ_i suitable to a binomial distribution because it describes the number of successes (in this case, heads with p to success) in a fixed number of independent and identical trials (in this case 40 tosses)

   $\theta_i \sim Bin(40, p)$
2. What is the mean $\mu = \mathbb{E}[\theta_i]$?
   Answer: $\mathbb{E}[\theta_i] = \text{TODO}$.

   The mean of a binomial distribution is n · p, therefore

   $\mathbb{E}[\theta_i] = 40 \cdot p$

3. We would like to use the empirical mean defined above as an estimator $\bar{\theta}(m)$ for μ.
   Use Hoeffding's inequality to compute the *smallest* error ϵ that can guaranteed given a sample size $m = 20$ with confidence 0.95 (notice that we wish to estimate μ, not p).
   That is, find the smallest ϵ that holds $\Pr\left[\left|\bar{\theta}(20) - \mu\right| > \epsilon\right] \le 0.05$.

   Answer:
   TODO

   for $\epsilon > 0$, the empirical mean $\bar{\theta}(20) = \frac{1}{20}\sum_i \theta_i$ with θ_i bound in [a,b] = [0,40] because $\Pr[0 \le \theta_i \le 40] = 1$. holds:

$$\Pr\left[\left|\bar{\theta}(20) - \mu\right| > \epsilon\right] \le 2\exp\left\{-\frac{2 \cdot 20\epsilon^2}{40^2}\right\}$$
$$2\exp\left\{-\frac{1}{40}\epsilon^2\right\} \le 0.05$$
$$-0.025\epsilon^2 \le \ln\left\{\frac{0.05}{2}\right\}$$
$$\epsilon^2 \ge \frac{\ln\left\{\frac{0.05}{2}\right\}}{-0.025}$$
$$\epsilon \ge \sqrt{\frac{\ln\left\{\frac{0.05}{2}\right\}}{-0.025}}$$

   so the smallest error ϵ equal to $\sqrt{\frac{\ln\left\{\frac{0.05}{2}\right\}}{-0.025}}$ = 12.14722924

4 . The following code simulates tossing $m = 10^4$ coins, each 50 times. For each coin, we use the empirical mean as the estimator and save it in the `all_estimators` array. The (unknown) probability of each coin is 0.75.
Complete the missing part so that for each coin, an array of 50 binary `observations` will be randomized according to the probability p.

```
m = 10**4
tosses = 50
p = 0.75
all_estimators = []

# Repeat for n coins
for coin in range(m):
    # TODO: Use Google to find a suitable numpy.random function that creates
    # a binary array of size (tosses,), where each element is 1
    # with probability p, and 0 with probability (1-p).
    observations = np.random.binomial(1, p, tosses)
    # result of flipping a coin 50(tosses) times.

    # Compute and save the empirical mean
    estimator = np.mean(observations)
    all_estimators.append(estimator)
```

5 . The following code plots the histogram of the estimators (empirical means). Run it. What type of distribution is obtained (no need to specify the exact paramters of the distribution)? Explain **briefly** what theorem from probability explains this behavior (and why).

Answer: TODO

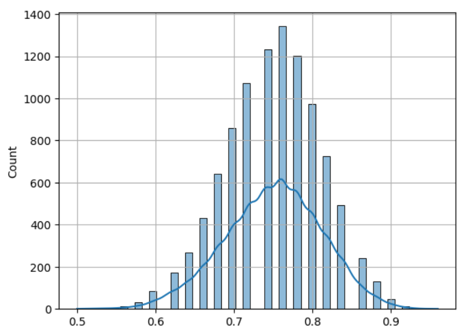
The normal distribution

The Central Limit Theorem states that distibution of the avarages of sum of a large number of i.i.d random variables approaches a normal distribution, regardless of the original distribution of variables(bernoulli variables).

Explaining this behavior it is the Central Limit Thorem. the array of "all_estimators" is a set of means, each mean is an avarage of the sum Barnoulli random varibles with parameter P, representing coin flips 50 times with probability of 0.75 to get 'heads'(50 is a large number of trials in this case).

Therefore, according to the Central Limit Behavior of this variable (the avargae of the Barnouli variables) approaches that of a normal distribution with mean p.

```
import seaborn as sns
sns.histplot(all_estimators, bins=tosses, kde=True)
plt.grid()
```



Numerical linear algebera refresher

Reminder - Positive semi-definite matrices

A symmetric real matrix $A \in \mathbb{R}^{n\times n}$ is called positive semi-definite (PSD) iff:

$$\forall x \in \mathbb{R}^n \setminus \{0_n\} : x^\top Ax \geq 0.$$

If the matrix holds the above inequality *strictly*, the matrix is called positive definite (PD).

Q3 - PSD matrices

1. Let $A \succeq \mathbf{0}_{n\times n}$ be a symmetric PSD matrix in $\mathbb{R}^{n\times n}$.
   Recall that all eigenvalues of real symmetric matrices are real.
   Prove that all the eigenvalues of A are non-negative.
   Answer:
   TODO

   Every symmetric matrix has a spectral decomposition

   $$\forall x \neq 0, x^\top Ax = x^\top QDQ^\top x = (Q^\top x)^\top D(Q^\top x) \geq 0$$

   define

   $$z = Q^\top x$$

   (we can define it because Q is an orthogonal matrix, so matrix Q is invertible, and the vector x is non-zero, therefore, their product is non-zero.)

   $$\forall z \neq 0, x^\top Kx = z^\top Dz = \sum_{k=1}^{n} \lambda_k z_k^2 \geq 0$$

   therefore all the eigenvalues of A are non-negative.

   Another option to prove it is the reason of a symatric matrix is diagonalizable matrix, therefore the algebraic multiplicity of each eigenvalue is equal to its geometric multiplicity.

   let $\lambda \in \mathbb{R}$ some eigenvalue of A, existing $0 \neq x_\lambda \in \mathbb{R}$, givan the fact of A is PSD exist:

   $$x_\lambda^\top Ax_\lambda \geq 0$$

   holds for any vactor and in particular for x_λ.

   let $x = (a_1, a_2, \ldots\ldots a_n)$ so:

   $$0 \leq \lambda \cdot \sum_{i=1}^{n} a_i^2$$

   from the above equality, we infer that 0≤λ. (because $\sum_{i=1}^{n} a_i^2 \geq 0$ and not all a_i are equal to 0, so 0 $\leq$ λ).

2. Let $A \in \mathbb{R}^{n\times n}$ be a symmetric PSD matrix and $B \in \mathbb{R}^{n\times n}$ a square matrix.
   What can be said about the symmetric matrix $(B^\top AB)$? Specifically, is it necessarily PSD? is it necessarily PD? Explain.
   Answer:
   TODO

   $$\forall x \neq 0, x^\top B^\top ABx = (Bx)^\top A(Bx)$$

   because A PSD and $Bx \neq 0$ so:

   $$(Bx)^\top A(Bx) \geq 0$$

   therefore $B^\top AB$ is symmetric and necessarily PSD, but not necessarily PD. for example if B is invertible, exist vector $x \neq 0$ so Bx=0, therefore $x^\top B^\top ABx = 0$, then $B^\top AB$ is not PD.

Q4 - Gradients

Define $f : \mathbb{R}^d \to \mathbb{R}$, where $f(w) = w^\top x + b$, for some given vector $x \in \mathbb{R}^d$ and a scalar $b \in \mathbb{R}$.

Recall: the gradient vector is defined as $\nabla_w f = \left[\frac{\partial f}{\partial w_1}, \ldots, \frac{\partial f}{\partial w_d}\right]^\top \in \mathbb{R}^d$.

1. Prove that $\nabla_w f = x$.

$$f(\mathbf{w}) = \begin{bmatrix} w_1 & w_2 & \cdots & w_d \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} + b$$
$$= w_1 x_1 + w_2 x_2 + \ldots + w_d x_d + b$$
$$\frac{\partial f}{\partial w_i} = x_i$$

so $\nabla_w f = \left[\frac{\partial f}{\partial w_1}, \ldots, \frac{\partial f}{\partial w_d}\right]^\top = \begin{bmatrix} x_1 & x_2 & \cdots & x_d \end{bmatrix} = x$

Recall/read the definition of the Hessian matrix $\nabla_w^2 f \in \mathbb{R}^{d \times d}$.

2 . Find the Hessian matrix $\nabla_w^2 f$ of the function f defined in this question.

3 . Is the matrix you found positive semi-definite? Explain.

Now, define $g : \mathbb{R}^d \to \mathbb{R}$, where $\lambda > 0$ and $g(w) = \frac{1}{2}\lambda\|w\|^2$.

4 . Find the gradient vector $\nabla_w g$.

5 . Find the Hessian matrix $\nabla_w^2 g$.

6 . Is the matrix you found positive semi-definite? is it positive definite? Explain.

Answers:

TODO

2.

The Hessian matrix of $f(w) = w^\top x + b$ is the second-order partial derivatives of f since f is linear, all the second-order partial derivatives are zero. therefore, $\nabla_w^2 f = 0_{d_x d}$

3.

yes. matrix $A \in \mathbb{R}^{n \times n}$ is called positive semi-definite (PSD) iff:

$$\forall x \in \mathbb{R}^n \setminus \{0_n\} : x^\top A x \geq 0.$$

because $\nabla_w^2 f = 0_{d_x d}$

$$\forall x \in \mathbb{R}^d \setminus \{0_d\} : x^\top 0_{d_x d} x = 0.$$

Therefore, $\nabla_w^2 f$is PSD.

4.

for

$$g(w) = \frac{1}{2}\lambda\|w\|^2$$

$g(w) = 0.5\lambda w^\top w = 0.5 \cdot \lambda(w_1^2 + w_2^2 + \ldots + w_d^2)$.

$$\frac{\partial g}{\partial w_i} = \lambda w_i$$
$$\nabla_w g = \lambda w$$

5.

The Hessian matrix of $g(w) = \frac{1}{2}\lambda\|w\|^2$ is the second-order partial derivatives of g.

$$\frac{\partial^2 g}{\partial w_i^2} = \lambda$$
$$\forall i, j, i \neq j : \frac{\partial^2 g}{\partial w_j \partial w_i} = 0$$

Therefore, $\nabla_w^2 g = \lambda I$

6.

for λ > 0 The matrix λI is PD. This is because, for any non-zero vector x, exist $x^T \lambda I x = \lambda\|x\|^2 > 0$. Therefore, the Hessian matrix $\nabla_w^2 g$ is PD.