Short_HW1

April 2, 2023

ID - 213871668

1 Short HW1 - Preparing for the course

1.0.1 Useful python libraries, Probability, and Linear algebera

1.1 Instructions

1.1.1 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 4 *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?** Tuesday 04.04.2023 at 23:59.

1.1.2 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 $X \perp Y \implies \text{Var}(X+Y) = \text{Var}(X)\text{Var}(Y)$ but it should say $X \perp Y \implies \text{Var}(X+Y) = \text{Var}(X) + \text{Var}(Y)$.

1.1.3 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).

2 Good luck!

---

3 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

3.1 Pandas Objects

[5]:
```
import pandas as pd
import numpy as np
```

Series is like a column in a spreadsheet.

[6]:
```
s = pd.Series([1,3.2,np.nan,'string'])
s
```

[6]:
```
0          1
1        3.2
2        NaN
3     string
dtype: object
```

DataFrame is like a spreadsheet – a dictionary of Series objects

[7]:
```
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
```

[7]:
```
  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
```

3.2 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

4 Case Study – Analyzing Titanic Passengers Data

[8]:
```
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

[9]:
```
# TODO: print the top 6 rows of the table
pd.DataFrame.head(train,6)
```

[9]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3
5            6         0       3

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

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
```

```
3       0             113803  53.1000  C123        S
4       0             373450   8.0500   NaN        S
5       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)

[10]:
```
train.columns
```

[10]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

4.1 Understanding the data (Summarizations)

[11]:
```
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
```

[12]:
```
train.shape
```

[12]:
```
(891, 12)
```

[13]:
```
# Count values of 'Survived'
train.Survived.value_counts()
```

[13]:
```
0    549
1    342
Name: Survived, dtype: int64
```

[14]:
```
# Calculate the mean fare price
train.Fare.mean()
```

[14]:
```
32.2042079685746
```

[15]:
```
# General statistics of the dataframe
train.describe()
```

[15]:
```
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
std     257.353842    0.486592    0.836071   14.526497    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   20.125000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.500000    1.000000    3.000000   38.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000

            Parch        Fare
count  891.000000  891.000000
mean     0.381594   32.204208
std      0.806057   49.693429
min      0.000000    0.000000
25%      0.000000    7.910400
50%      0.000000   14.454200
75%      0.000000   31.000000
max      6.000000  512.329200
```

4.1.1 Selection examples

Selecting columns

[16]:
```
# Selection is very similar to standard Python selection
df1 = train[["Name", "Sex", "Age", "Survived"]]
df1.head()
```

[16]:
```
                                                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

[17]:
```
df1[10:15]
```

[17]:
```
                               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
```

4.1.2 Filtering Examples

Filtering with one condition

[18]:
```
# Filtering allows you to create masks given some conditions
df1.Sex == 'female'
```

[18]:
```
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
```

[19]:
```
onlyFemale = df1[df1.Sex == 'female']
onlyFemale.head()
```

[19]:
```
                                                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.

[20]:
```
# 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: 78.24%
```

4.2 Aggregating

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

[21]:
```
df2 = train.groupby(['Pclass', 'Sex']).Fare.agg(np.mean)
df2
```

[21]:
```
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
```

[22]:
```
pd.pivot_table(train, index=['Pclass'], values=['Survived'], aggfunc='count')
```

[22]:
```
        Survived
Pclass
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.

[23]:
```
# TODO: Also show the mean age per group
pd.pivot_table(train, index=['Pclass', 'Sex'], values=['Age', 'Survived'],␣
    ↪aggfunc='mean')
```

[23]:
```
                     Age  Survived
Pclass Sex
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

```
[24]: # TODO: find the mean survival rate per age group
ageGroups = np.arange(0, 81, 10)
survivalPerAgeGroup = pd.pivot_table(train, index=pd.cut(train.Age, ageGroups),␣
 ↪values=['Age', 'Survived'], aggfunc='mean')
survivalPerAgeGroup
```

```
[24]:                 Age  Survived
Age
(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
```

```
[25]: type(train.groupby(pd.cut(train.Age, ageGroups)).Survived.mean())
```

```
[25]: pandas.core.series.Series
```

4.3 Filling missing data (data imputation)

Note that some passenger do not have age data.

```
[26]: 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
```

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

```
[27]:                           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.

```
[28]: df1.Age.describe()
```

```
[28]: 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).

```
[29]: # TODO : Fill the missing values
      filledDf = df1.copy()
      filledDf.Age = df1.Age.fillna(df1.Age.mean())
```

```
[30]: 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.

```
[31]: filledDf.Age.describe()
```

```
[31]: count    891.000000
      mean      29.699118
      std       13.002015
      min        0.420000
      25%       22.000000
      50%       29.699118
      75%       35.000000
      max       80.000000
      Name: Age, dtype: float64
```

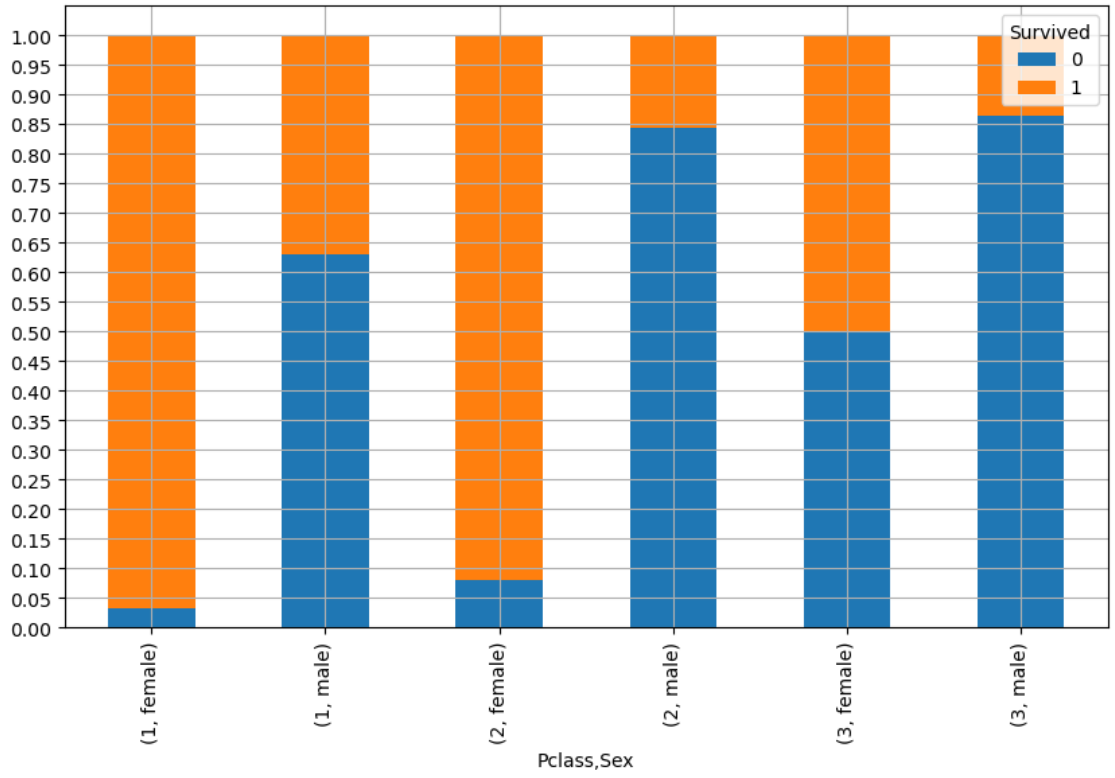
(T6) Answer below: which statistics changed, and which did not? Why? (explain briefly, no need to be very formal.)

Answer: TODO 1. The "count" statistic changed since we filled the missing values with real values. 2. The "mean" statistic is unchanced because the values we added were exactly the "previous" mean, so the we didn't change the overall mean. 3. The standard deviation decreased since we added more "mass" around (spot on) the mean, and as we recall the STD is a measure of deviations from the mean. 4. "min", "max" didn't change since the mean (the only values that we changed) is somewhere inside [min, max]. 5. "Q1", "Q2", "Q3" changed in the following way: "Q1" increased - meaning that the mean is valued at above Q1. "Q2" increased - same thing. "Q3" decreased - meaning that the mean is somewhere in between Q2 and Q3.

4.4 Plotting

Basic plotting in pandas is pretty straightforward

[32]:
```
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: (1, female) had the best survival rate. (3, male) had the worst.

5 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`

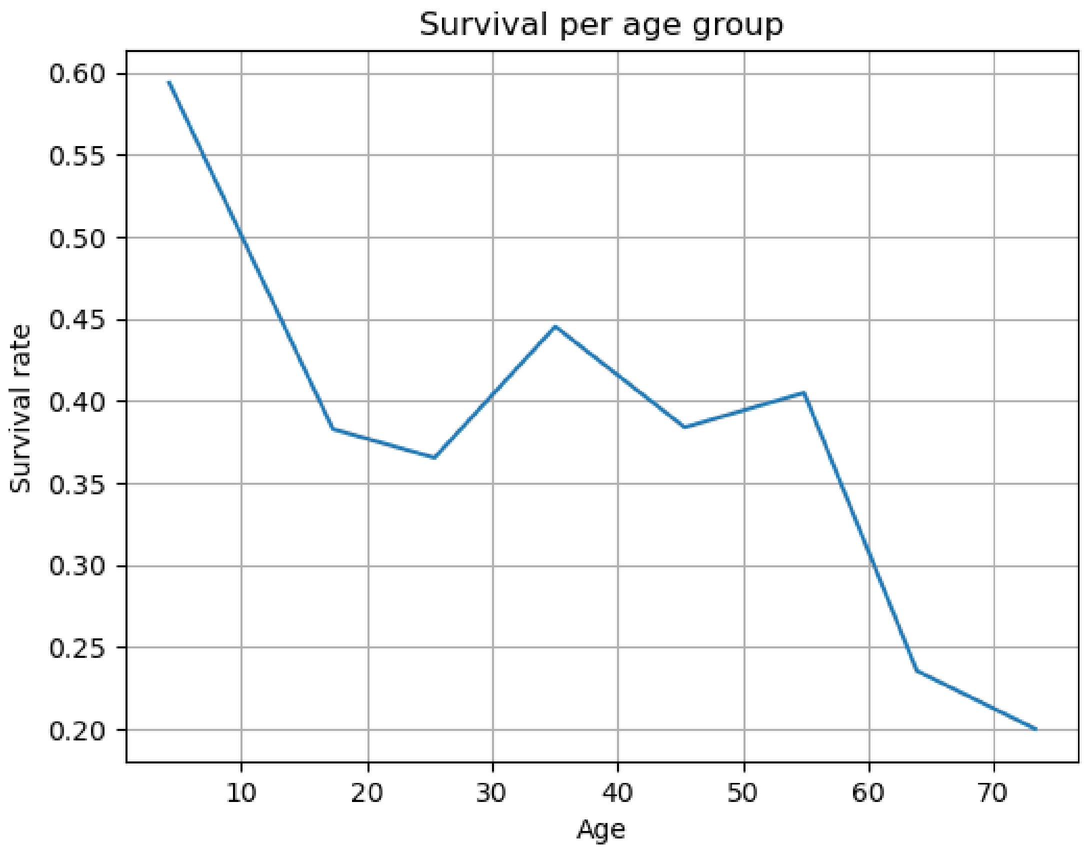
[33]:
```
import matplotlib.pyplot as plt
import numpy as np
```

5.1 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.

[34]:
```
plt.plot(survivalPerAgeGroup.Age, survivalPerAgeGroup.Survived)
_ = plt.title("Survival per age group")
# TODO : Update the plot as required.
plt.xlabel("Age")
plt.ylabel("Survival rate")
plt.grid()
plt.show()
```



[35]:
```
survivalPerAgeGroup
```

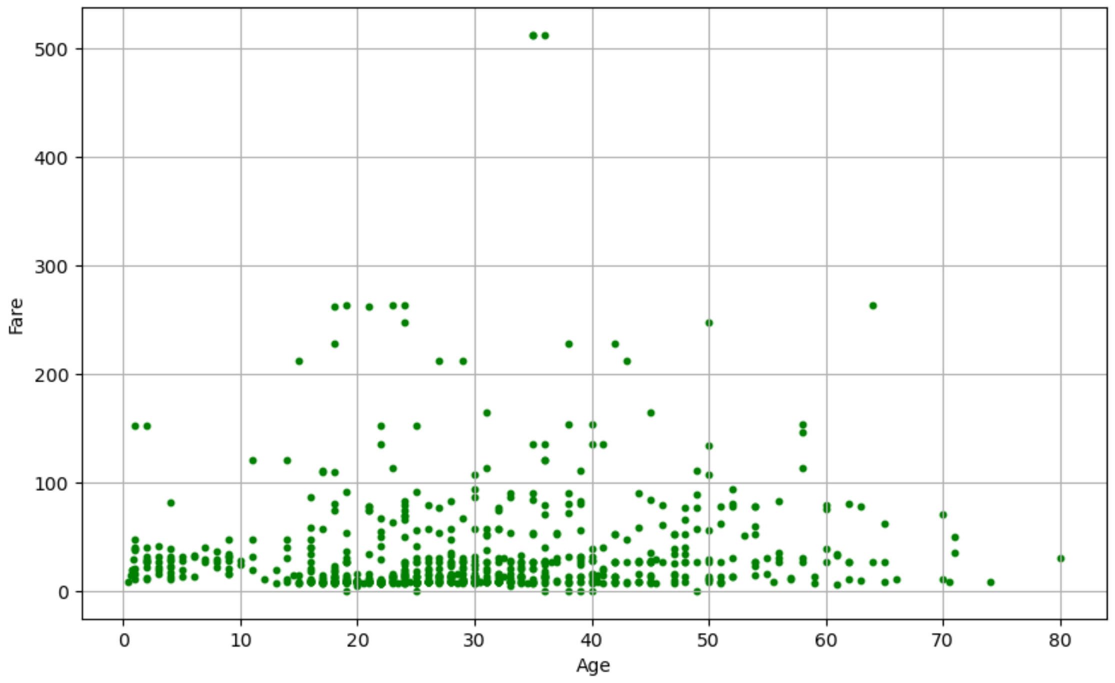
[35]:
```
                 Age  Survived
Age
(0, 10]     4.268281  0.593750
(10, 20]   17.317391  0.382609
(20, 30]   25.423913  0.365217
(30, 40]   35.051613  0.445161
(40, 50]   45.372093  0.383721
(50, 60]   54.892857  0.404762
(60, 70]   63.882353  0.235294
(70, 80]   73.300000  0.200000
```

5.2 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.

[36]:
```python
# TODO : Update the plot as required.
plt.figure(figsize=(10,6))
plt.scatter(train.Age, train.Fare, c='g', s=10)
plt.xlabel("Age")
plt.ylabel("Fare")
plt.grid()
plt.show()
```



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

Answer: around 34-36

---

6 Probability refresher

6.1 Q1 - Variance of empirical mean

Let $X_1, \dots, X_m$ 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 ,( ) the mean μ from the observations $x_1, \dots, x_m$.
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 $\overline{X}$.

Answer: $\mathbb{E}\left[\overline{X}\right] = \mathbb{E}\left[\frac{1}{m}\sum_i X_i\right] = \mathbb{E}\left[\overline{X}\right] = \mathbb{E}\left[\frac{1}{m}\sum_i X_i\right] \underset{linearity\,of\,expectation}{=} \frac{1}{m}\sum_i \mathbb{E}\left[X_i\right] \underset{i.i.d}{=} \frac{1}{m}m\mu = \mu$

2. Express analytically the variance of $\overline{X}$.
   Answer: $\mathrm{Var}\left[\overline{X}\right] = \mathrm{Var}\left[\frac{1}{m}\sum_i X_i\right] \underset{variance\,rules}{=} \frac{1}{m^2}\mathrm{Var}\left[\sum_i X_i\right] \underset{independence}{=} \frac{1}{m^2}\sum_i \mathrm{Var}\left[X_i\right] \underset{identically\,distributed}{=} \frac{1}{m^2}\sum_i \sigma^2 = \frac{\sigma^2}{m}$

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 $m = 1, \ldots, 35$.
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.

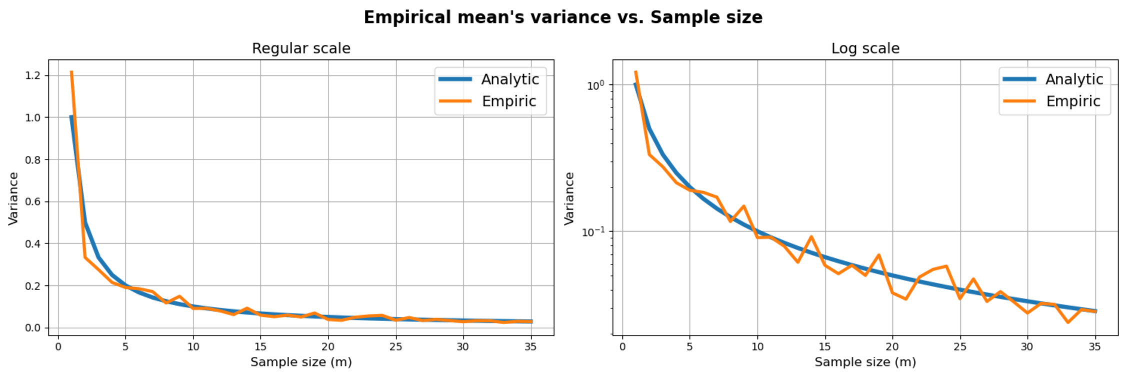
[37]:
```
all_sample_sizes = range(1, 36)
repeats_per_size = 50

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)`.)

[38]:
```
# 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)

# Normal distrubution with mean 0 and variance 1, so the variance of the␣
 ↪empirical mean is 1/m for each m
analyticVariance = 1/np.array(all_sample_sizes)
```

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.

[39]:
```
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()
```



6.2 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 $\overline{\theta}(m) = \frac{1}{m}\sum_i \theta_i$ holds:

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

6.3 Q2 - Identical coins and the Hoeffding bound

We toss $m \in \mathbb{N}$ identical coins, each coin 50 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: θ_i is the number of successes of bernoulli random variable with parameter p, out of 50 tries. The
   $Bin\left(50, p\right)$

2. What is the mean $\mu = E\left[\theta_i\right]$?
   Answer: $E[\theta_i] = 50p$.

3. We would like to use the empirical mean defined above as an estimator $\overline{\theta}(m)$ for μ.
   Use Hoeffding's inequality to compute the *smallest* sample size $m \in \mathbb{N}$ that can guarantee an error of $\epsilon = 1$ with confidence 0.99 (notice that we wish to estimate μ, not p).
   That is, find the smallest m that holds $\Pr\left[\left|\overline{\theta}(m) - \mu\right| > 1\right] \le 0.01$.
   Answer:
   using Hoeffding's inequality we get :
   $\Pr\left[\left|\overline{\theta}(m) - \mu\right| > 1\right] \le 2exp\left\{-\frac{2m\cdot 1^2}{\left(50-0\right)^2}\right\} \underset{set}{\le} 0.01$
   $-\frac{2m}{2500} \le \ln 0.005$
   $m \ge -\frac{2500\cdot\ln 0.005}{2} = 6622.9$
   Therefore, the minimal m is 6623

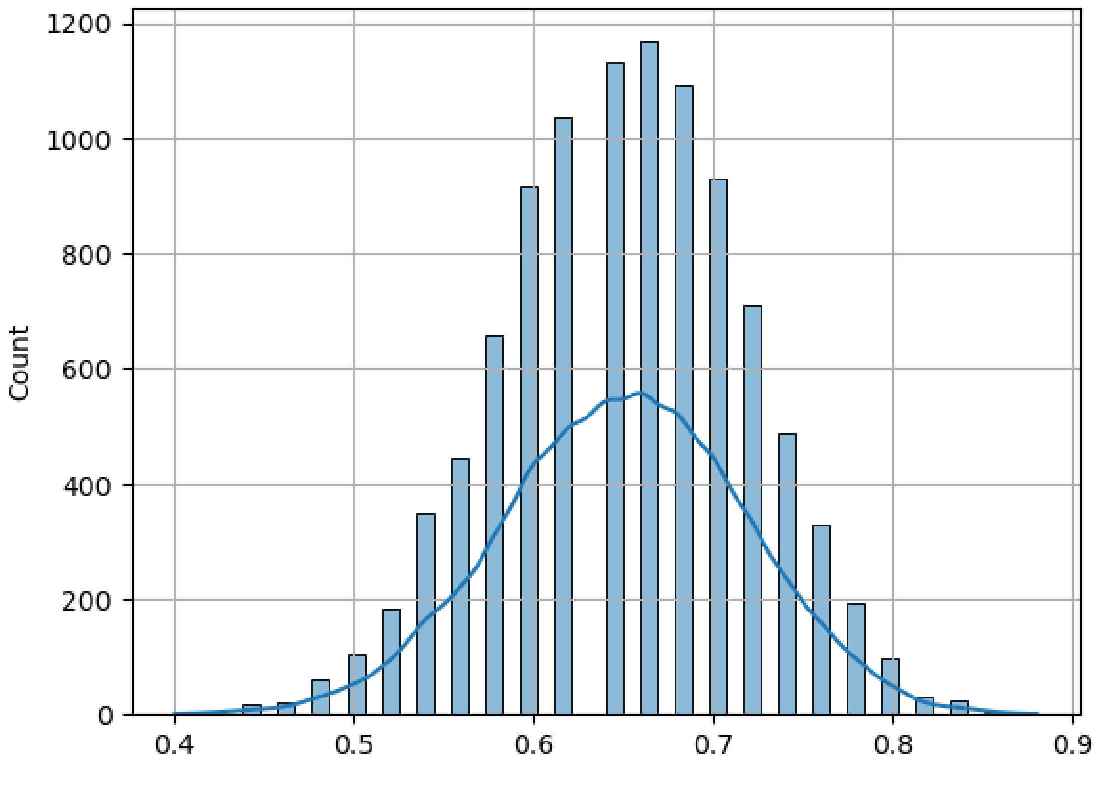
4. The following code simulates tossing $m = 10^4$ coins, each 50 times. For each coin, we use the empirical mean as the estimator and save it in the `all_estimators` array. The (unknown) probability of each coin is 0.65.
   Complete the missing part so that for each coin, an array of 50 binary `observations` will be randomized according to the probability p.

[40]:
```
m = 10**4
tosses = 50
p = 0.65
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).

    # a bernoulli distribution is a binomial distribution with n=1. therefore we␣
    ↪can use random.binomial
    # to generate a bernoulli distribution for each index in the array
    observations = np.random.binomial(1, p, tosses)

    # 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: The distribution that is obtained is a Gaussian. The CLT (central limit theorem) explains this behaivor. It does so because practically what we're doing is looking at a large sample size of i.i.d bernoulli distributed samples with parameter p, and averaging over the results. We know from CLT that as the number of samples approaches infinity, the estimator approaches a Gaussian with mean p (0.65), and that's what we see in the histogram.

[41]:
```
import seaborn as sns
sns.histplot(all_estimators, bins=tosses, kde=True)
plt.grid()
```



6.4 Numerical linear algebera refresher

6.5 Reminder - Positive semi-definite matrices

A symmetric real matrix $A \in \mathbb{R}^{n\times n}$ is called positive semi-definite (PSD) iff: $\forall x \in \mathbb{R}^n \quad \{0_n\}$: $x^\top Ax \geq 0$. If the matrix holds the above inequality *strictly*, the matrix is called positive definite (PD).

6.6 Q3 - PSD matrices

1. Let $A \succ \mathbf{0}_{n\times n}$ be a symmetric PD 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 strictly positive.
   Answer:
   We know from the spectral decomposition that any symmetric real values matrix is orthogonally diagonalizable.
   Suppose a spectral decomposition of A, with $\{e_i, \lambda_i\}_{i=1}^n$ being the eigenvectors and eigenvalues, respectively
   $\forall i \in [n] : e_i^T Ae_i = \lambda_i e_i^T e_i = \lambda_i \left|\left|e_i\right|\right|_2^2$. From positivity of A we conclude that $\lambda_i > 0$

2. Let $A, B \in \mathbb{R}^{n\times n}$ be two symmetric PSD matrices.
   Prove or refute: the matrix $(2A - B)$ is also PSD.
   Answer:
   $Wrong!$
   $Let\, A = I_n,\, B = 5 \cdot I_n.$
   $Show\, that\, A, B > 0 :$
   $\forall x \in R^n \backslash \{0\}, x^T Ax = x^T x = \left|\left|x\right|\right|_2^2 > 0$
   $\forall x \in R^n \backslash \{0\},\, x^T Bx = x^T 5x = 5 \left|\left|x\right|\right|_x^2 > 0$
   $But\; 2A - B = -3I_n,\; and\; (e_1^n)^T (-3I_n)\, e_1^n = -3 < 0 \Rightarrow 2A - B\, isn't\; PSD$

6.7 Q4 - Gradients

Define $f : \mathbb{R}^d \to \mathbb{R}$, where $f(w) = w^\top x + b$, for some 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 $\nabla_w f = x$.
$w = \begin{pmatrix} w_1 & w_2 & ,\ldots, & w_d \end{pmatrix}^T, x = \begin{pmatrix} x_1 & x_2 & ,\ldots, & x_d \end{pmatrix}^T$
$\Rightarrow f(w) = w^T x + b = w_1x_1 + w_2x_2 + .. + w_dx_d + b$
$\Rightarrow \frac{\partial f}{\partial w_i} = x_i$
$\Rightarrow \nabla_w f = \begin{bmatrix} \frac{\partial f}{\partial w_1} & ,\ldots, & \frac{\partial f}{\partial w_d} \end{bmatrix}^T = \begin{bmatrix} x_1 & ,\ldots, & x_d \end{bmatrix}^T = x$

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.

To calculate the Hessian matrix, we will calculate each index independently:
$(\nabla_w^2 f)_{i,j} = \frac{\partial f}{\partial w_i \partial w_j} = \frac{\partial}{\partial w_i}\left(\frac{\partial f}{\partial w_j}\right) = \frac{\partial}{\partial w_i} x_j = 0$
From that we conclude that the hessian matrix is 0. it is PSD, since for every vector $x \in R$, $x^T \cdot 0 \cdot x = 0 \geq 0$

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

4. Find the gradient vector $\nabla_w g$.
5. Find the Hessian matrix $\nabla^2_w g$.
6. Is the matrix you found positive semi-definite? is it positive definite? Explain.

Answers:

Write $w = (w_1, ..., w_d)^T$. so we get $g(w) = \lambda w^T w = \lambda(w_1, ..., w_d)(w_1, ..., w_d)^T = \lambda(w_1^2 + ... + w_d^2)$
$\Rightarrow \nabla_w g = [\frac{\partial g}{\partial w_1}, ..., \frac{\partial g}{\partial w_d}] = [2\lambda w_1, ..., 2\lambda w_d]$. We can easily see that for the second derivatives, the diagonal of the hessian is 2 and the rest are zeros. That's because every element not in the diagonal calculates the derivative of a "constant" (in respect to the respective variable).
We conclude that $\nabla^2_w g = 2I_d$
It is PD because for every non-zero vector $x \in R^d \, we \, get \; x^T \nabla^2_w g x = 2x^T x = 2\,||x||_2^2 > 0$
We know that every PD matrix is PSD so it is also PSD.