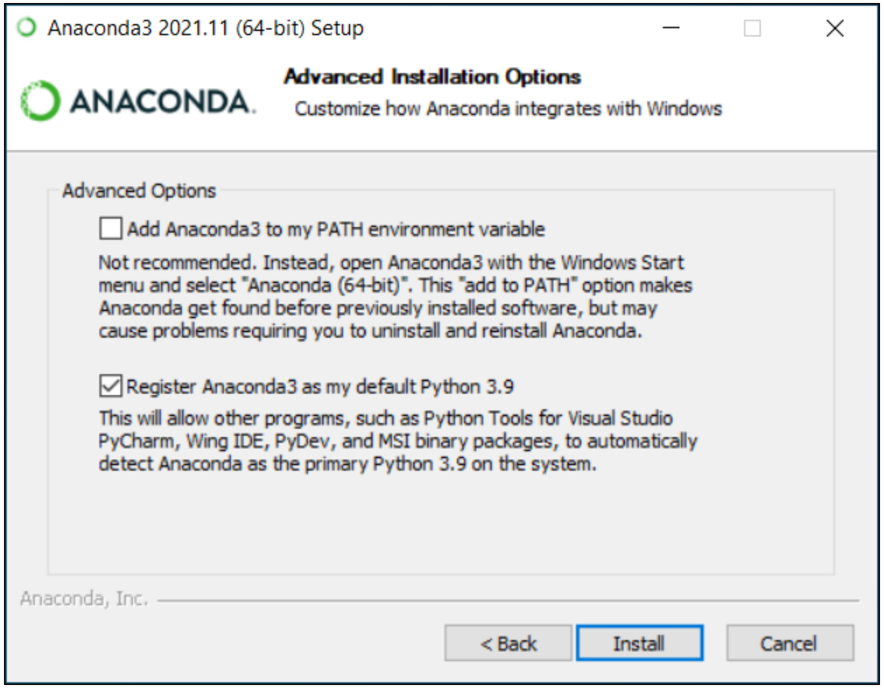
Lab 1: Basic Statistical Analysis with Python

Outline

1. Preparation
    - Anaconda and Jupyter Notebook
    - Python Packages
2. Handling the Data
    - Data Description
    - Load the Data
3. Descriptive Statistics
    - Numerical Measures
    - Tabular and Graphical Displays

Anaconda Installation

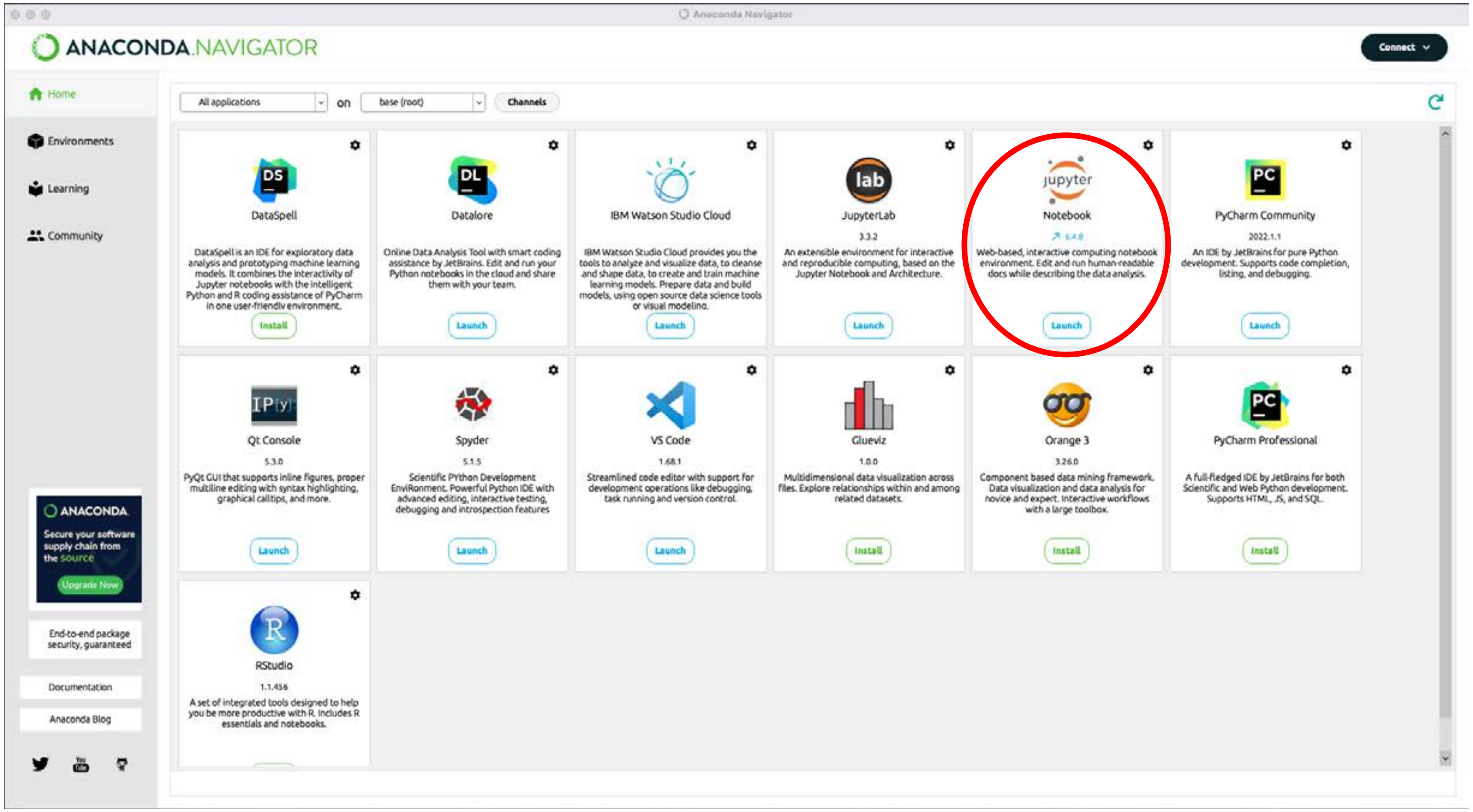
- **Installing Anaconda on *Windows***
  - https://docs.anaconda.com/anaconda/install/windows/
  - https://www.datacamp.com/tutorial/installing-anaconda-windows
- **Installing Anaconda on *MacOS***
  - https://docs.anaconda.com/anaconda/install/mac-os/
  - https://www.datacamp.com/tutorial/installing-anaconda-mac-os-x



- **Note**
  - We don't recommend adding Anaconda to your PATH environment variable, since this can interfere with other software.
  - Unless you plan on installing and running multiple versions of Anaconda or multiple versions of Python, accept the default and leave this box checked. Instead, use Anaconda software by opening Anaconda Navigator or the Anaconda Prompt from the Start Menu.

Jupyter Notebook

- **From the Navigator Home tab, click *Jupyter Notebook***
    - Jupyter Notebook is an increasingly popular system that combine your code, descriptive text, output, images, and interactive interfaces into a single notebook file that is edited, viewed, and used in a web browser.

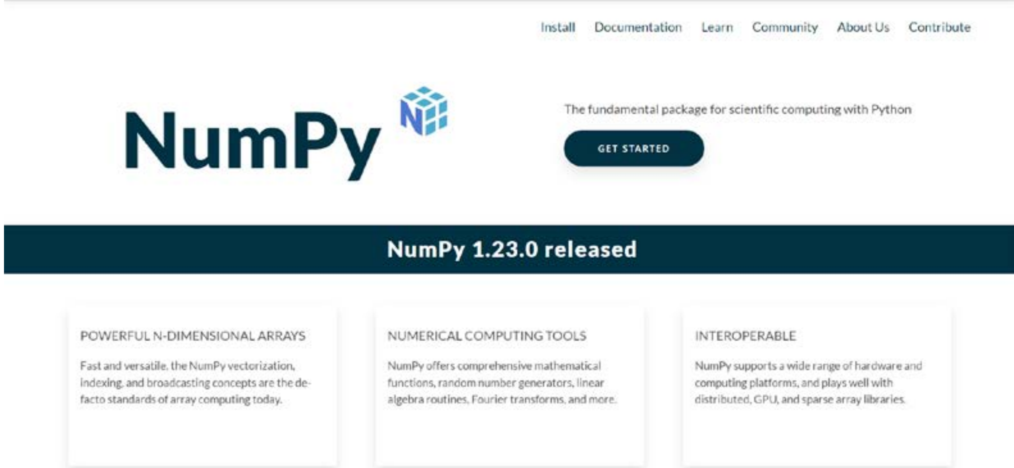


Packages Installation

- **Installing conda packages**
  - https://docs.anaconda.com/anaconda/user-guide/tasks/install-packages/
  - https://datatofish.com/how-to-install-python-package-in-anaconda/
- **Open an *Anaconda Prompt* and enter the command:**
  - *conda* install package_name, e.g. conda install matplotlib
  - *pip* install package_name, e.g. pip install matplotlib
- **conda vs. pip**
  - The essential difference between the two is:
    - conda installs any package in conda environments
    - pip installs python packages in any environment
  - If you installed Python using Anaconda or Miniconda, then use *conda* to install Python packages. If conda tells you the package you want doesn't exist, then use *pip* (or try *conda-forge*, which has more packages available than the default conda channel).
  - If you installed Python any other way (from source, using pyenv, virtualenv, etc.), then use *pip* to install Python packages.
- **Installing packages from a *Jupyter Notebook*:**
  - *! pip* install package_name, e.g. ! pip install matplotlib
    - The '!' tells the notebook to execute the cell as a shell command

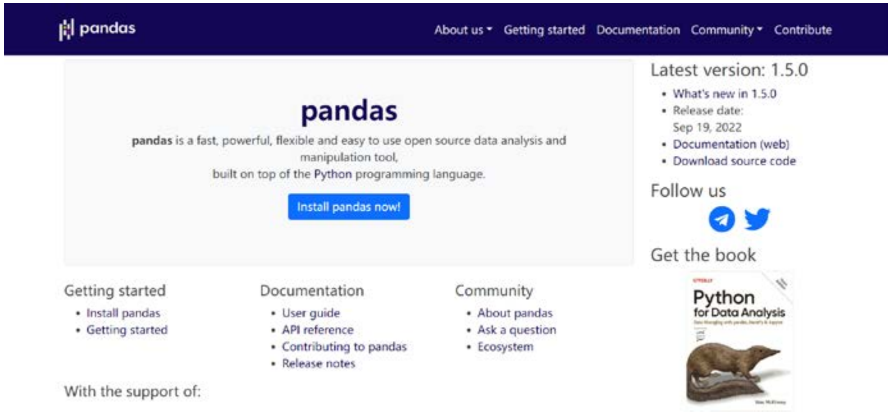
Some Popular Python Packages -- NumPy

- **https://numpy.org/**
- **NumPy is the primary tool for *scientific computing* in Python.**
- **It combines the flexibility and simplicity of Python with the speed of languages like C and Fortran.**
- **NumPy is used for:**
    - Advanced array operations (e.g. add, multiply, slice, reshape, index)
    - Comprehensive mathematical functions
    - Random number generation
    - Linear algebra routines
    - Fourier transforms
    - …
- **import numpy as np**



Some Popular Python Packages -- Pandas

- **https://pandas.pydata.org/**
- **If you work with *tabular*, *time series*, or *matrix* data, pandas is your go-to Python package.**
- **It works with *data frame* objects -- a data frame is a dedicated structure for two-dimensional data; data frames have rows and columns just like database tables or Excel spreadsheets.**
- **Pandas can be used for:**
  - Reading/writing data from/to CSV and Excel files and SQL databases
  - Reshaping and pivoting datasets
  - Slicing, indexing, and subsetting datasets
  - Aggregating and transforming data
  - Merging and joining datasets
  - …
- **import pandas as pd**

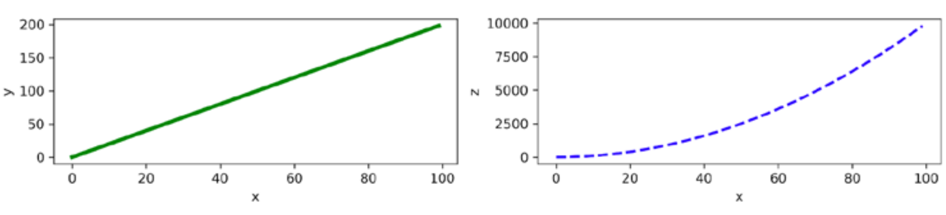


Some Popular Python Packages -- Matplotlib

- https://matplotlib.org/
- **Matplotlib is the most common data exploration and *visualization* library.**
  - You can use it to create *basic* graphs like *line plots*, *histograms*, *scatter plots*, *bar charts*, and *pie charts*.
  - You can also create animated and interactive visualizations with this library.
- **The library offers a great deal of flexibility with regards to formatting and styling plots.**
  - You can freely choose how to display labels, grids, legends, etc.
  - However, to create complex and visually appealing plots, you'll need to write quite a lot of code.

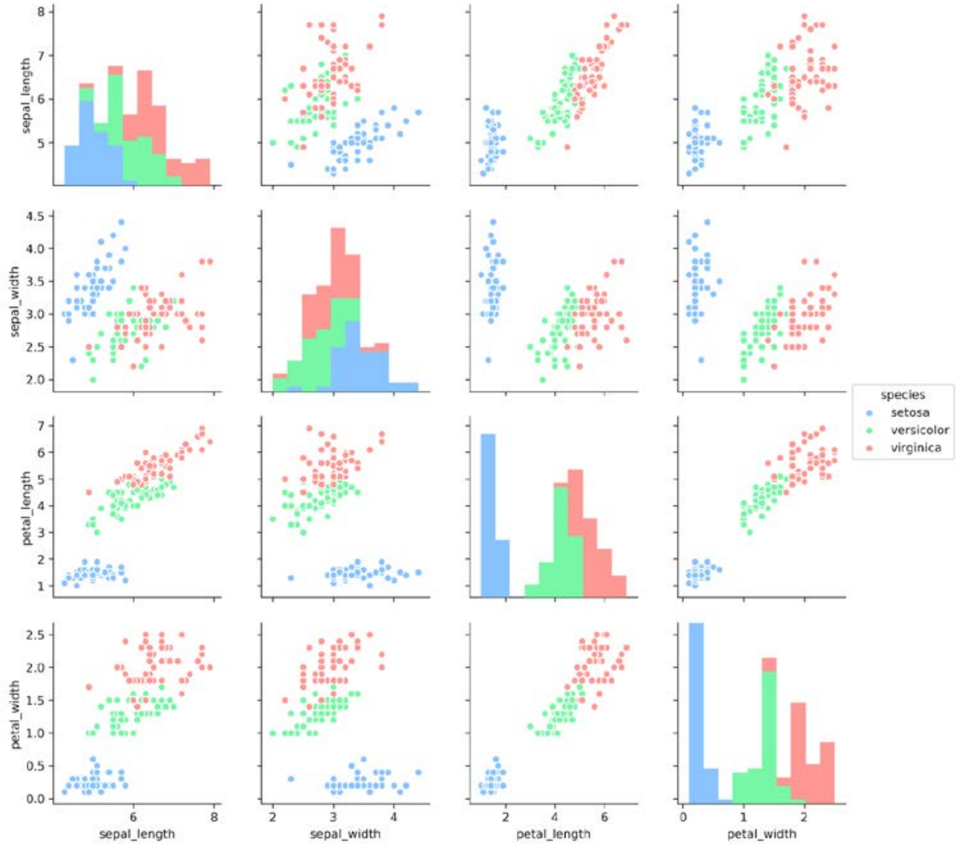
```
import numpy as np
x = np.arange(0,100)
y = x*2
z = x**2
```

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.show()

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12,2))

axes[0].plot(x,y, color="green", lw=3)
axes[0].set_xlabel('x')
axes[0].set_ylabel('y')

axes[1].plot(x,z, color="blue", lw=2, ls='--')
axes[1].set_xlabel('x')
axes[1].set_ylabel('z')
```



Some Popular Python Packages -- Seaborn

- https://seaborn.pydata.org/index.html
- **Seaborn is a high-level interface for drawing attractive statistical graphics with just a few lines of code.**
  - You can create a complex and visually appealing plot with just three lines of code.
    - **import seaborn as sns**
    - iris = sns.load_dataset('iris')
    - sns.pairplot (iris, hue = 'species', palette = 'pastel’)
  - Note how all labels, styles, and a legend have been set automatically.
  - Similarly, you can easily create complex heatmaps, violin plots, joint plots, multi-plot grids, and many other types of plots with this library.



Data Description

- This data set includes customers who have paid off their loans, who have been past due and put into collection without paying back their loan and interests, and who have paid off only after they were put in collection.
- The financial product is a bullet loan that customers should pay off all of their loan debt in just one time by the end of the term, instead of an installment schedule. Of course, they could pay off earlier than their pay schedule

Field	Description
Loan_id	A unique loan number assigned to each applicant
Loan_status	Whether a loan is paid off on in collection
Principal	The basic principal loan amount at origination
Terms	Origination terms which can be weekly (7 days), biweekly, and monthly payoff schedule
Effective_date	When the loan got originated and took effects
Due_date	Since it's one-time payoff schedule, each loan has one single due date
Paid_off_time	The actual time a customer pays off the loan
Past_due_days	How many days a loan has been past due
Age	Age of applicant
Education	Education of applicant
Gender	The gender of applicant

Load the Data

- Pandas DataFrames is an excel like data structure with labeled axes (rows and columns).
- pandas.read_csv() function imports a CSV file to DataFrame format. To load our data, the easiest way is:

```
# read csv file
data = pd.read_csv('Loan payments data.csv')
```

- If your CSV file is not in the current working directory/folder, we need to specify the path of your CSV file:

```
# read csv file
data = pd.read_csv('C:/Users/abc/Desktop/Loan payments data.csv')
```

- For Excel file, we can use pandas.read_excel() function to read an Excel file into a pandas DataFrame.

Overview of the Data

- We can show the first five rows in the dataset like this:

```
# first five rows in dataset
data.head()
```

	Loan_ID	Loan_status	Principal	Terms	Effective_date	Due_date	Paid_off_time	Past_due_days	Age	Education	Gende
0	xqd20166231	PAIDOFF	1000	30	9/8/2016	10/7/2016	9/14/2016 19:31	NaN	45	High School or Below	mal
1	xqd20168902	PAIDOFF	1000	30	9/8/2016	10/7/2016	10/7/2016 9:00	NaN	50	Bechalor	femal
2	xqd20160003	PAIDOFF	1000	30	9/8/2016	10/7/2016	9/25/2016 16:58	NaN	33	Bechalor	femal
3	xqd20160004	PAIDOFF	1000	15	9/8/2016	9/22/2016	9/22/2016 20:00	NaN	27	college	mal
4	xqd20160005	PAIDOFF	1000	30	9/9/2016	10/8/2016	9/23/2016 21:36	NaN	28	college	femal

Overview of the Data

- Pandas DataFrame.describe() is used to view some basic statistical details like percentile, mean, std etc. of a data frame or a series of numeric values.

```
# describe() gives the count, mean, std, min, max, important quantiles of the numeric columns
data.describe()
```

	Principal	Terms	Past_due_days	Age
count	500.000000	500.000000	500.000000	500.000000
mean	943.200000	22.824000	14.404000	31.116000
std	115.240274	8.000064	25.614312	6.084784
min	300.000000	7.000000	0.000000	18.000000
25%	1000.000000	15.000000	0.000000	27.000000
50%	1000.000000	30.000000	0.000000	30.000000
75%	1000.000000	30.000000	12.000000	35.000000
max	1000.000000	30.000000	76.000000	51.000000

Numerical Measures

-

Name	Method/Function
Mean	DataFrame.mean()
Median	DataFrame.median()
Geometric mean	scipy.stats.gmean(x)
Trimmed mean	scipy.stats.trim_mean(x)
Weighted mean	numpy.average(x, weights)
Percentile	DataFrame.quantile(q)
Quartile	DataFrame.quantile(q = 0.25)
Variance	DataFrame.var()
Standard deviation	DataFrame.std()
Coefficient of variation	scipy.stats.variation(x)
z-score	scipy.stats.zscore(x)
skewness	scipy.stats.skew(x)

Measure of Association

- Compute the pairwise covariance and correlation matrix of columns:

```
# covariance and correlation
print('The covariance matrix is: \n')
print(data.iloc[300:, [2, 7, 8]].cov())
print('\nThe correlation matrix is: \n')
print(data.iloc[300:, [2, 7, 8]].corr())
```

```
The covariance matrix is:

                 Principal  Past_due_days        Age
Principal      7708.291457    -240.256281 -92.575377
Past_due_days  -240.256281     863.236080  -9.753518
Age             -92.575377      -9.753518  38.004397

The correlation matrix is:

               Principal  Past_due_days       Age
Principal       1.000000      -0.093139 -0.171041
Past_due_days  -0.093139       1.000000 -0.053849
Age            -0.171041      -0.053849  1.000000
```

- We need to perform the hypothesis testing to test whether the correlation coefficient is significantly different from zero

```
p_value = stats.pearsonr(data.iloc[300:, 2], data.iloc[300:, 8])[1]
print(f'The p-value for the correlation coefficient between the Principal and the Age is: {p_value:.3f}.')
```

```
The p-value for the correlation coefficient between the Principal and the Age is: 0.015.
```

Categorical Data Description

- **Check the categories within the data sets**
  - **df['xxx'].unique()**
    - A function on the entire Dataframe(df) based on the column xxx, which is used to show the categories in categorical columns.
- **Shows the frequency distribution**
  - **df.value_counts()**
    - A function on the entire Dataframe(df) based on the column xxx, which returns the frequency values of categories.
  - **Counter(data['xxx'])**
    - A dictionary to count, the key is item, the value is the number of item occurrences

```
# shows the frequency distribution of 'Pricipal State'
print('Pricipal State\n')
print('Data Principal value counts\n')
print(data.Principal.value_counts())
```

```
Pricipal State

Data Principal value counts

1000    377
800     111
300       6
500       3
900       2
700       1
Name: Principal, dtype: int64
```

```
# check the categories within the data sets
data['Loan_status'].unique()
```

```
array(['PAIDOFF', 'COLLECTION', 'COLLECTION_PAIDOFF'], dtype=object)
```

```
from collections import Counter
Counter(data['Loan_status']) # 300 people have paid off the loan on time while 100 have gone into collection
```

```
Counter({'PAIDOFF': 300, 'COLLECTION': 100, 'COLLECTION_PAIDOFF': 100})
```

Categorical Data Description--Visualization

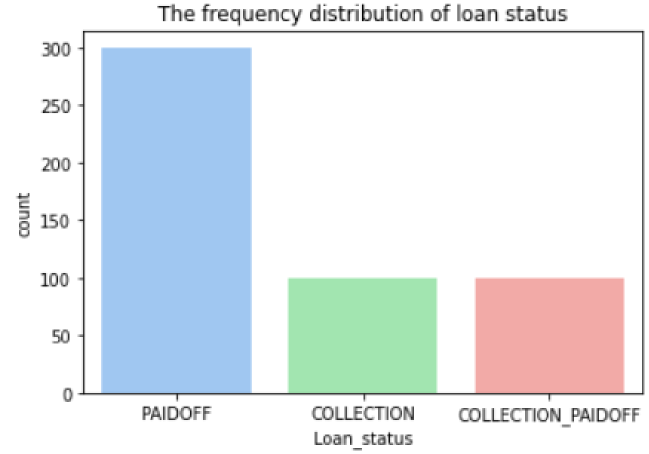
Bar chart

- **plt.bar**(x, y, [width], [color], [edgecolor], [bottom], [linewidth], [align], [tick_label], [align], **kwargs)
- **sns.barplot**(x, y, [data], [hue], [order], [color], [estimator], [ci], **kwargs)
- **sns.countplot**(x, [y], [data], [hue], [order], [color], [orient], **kwargs)

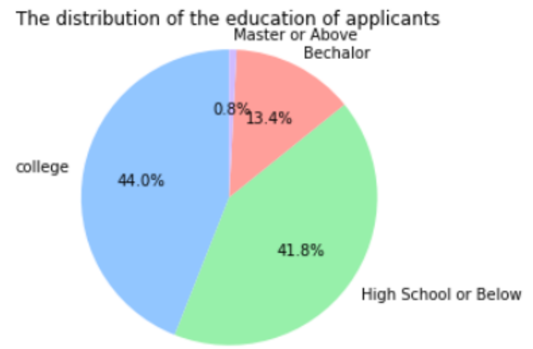
Pie chart

- **plt.pie**(x, [explode], [labels], [colors], [autopct], [pctdistance], [labeldistance], [startangle], [radius], **kwargs)

```
#Visualization process of Loan_Status--bar chart
sns.countplot(data['Loan_status'])
plt.title('The frequency distribution of loan status')
plt.show()
```




```
education_counts = data['Education'].value_counts()
plt.pie(education_counts, labels = education_counts.index,
        startangle = 90, autopct='%1.1f%%',
        counterclock = True);
plt.title('The distribution of the education of applicants')
plt.axis('square')
plt.show()
```



Quantitive Data Description

Shows the frequency distribution

- **pd.cut**(x, bins, [right], [labels], **kwargs)
  - The data values are segmented by themselves and sorted into *bins*, *bins* could be integer, scalar sequence, or interval index
- **df.value_counts**()

```
# shows the frequency distribution of age after grouping
data['Age_range'] = pd.cut(data['Age'], [10, 20, 30, 40, 50, 60], labels=['10s', '20s', '30s', '40s', '50s'])
print('Age Distribution\n')
print('Data Age_range value counts\n')
print(data.Age_range.value_counts())
```

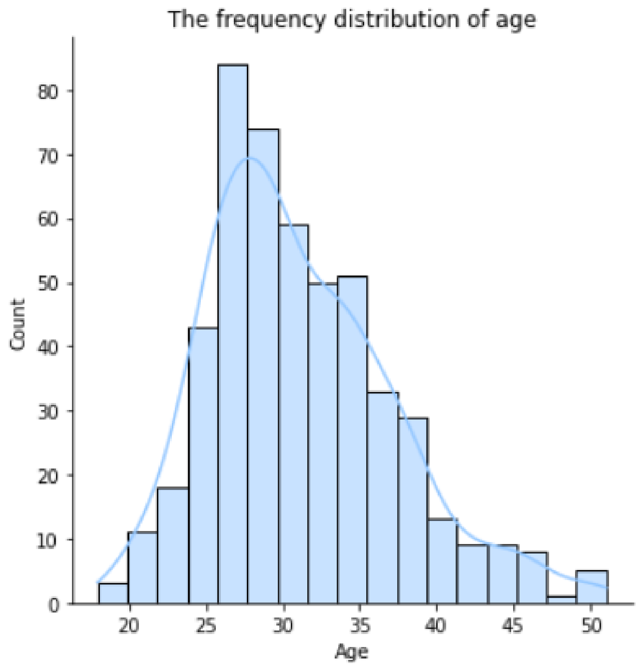
```
Age Distribution

Data Age_range value counts

20s    263
30s    193
40s     36
10s      7
50s      1
Name: Age_range, dtype: int64
```

Quantitive Data Description--Visualization

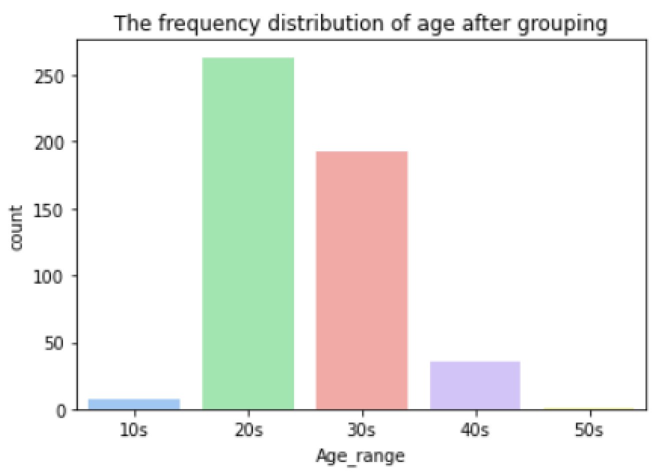
◻ Histogram

- **np.histogram(),** no visualization
- **plt.hist**(x, [bins], [range], [density]. [weights], [cumulative], **kwargs)
- **sns.displot**(x, [bins], [hist], [kde], [rug], [fit], **kwargs)
- **pd.cut()** + **sns.countplot()**

```
sns.displot(data['Age'], kde=True)
plt.title('The frequency distribution of age')
plt.show()
```




```
sns.countplot(data['Age_range'])
plt.title('The frequency distribution of age after grouping')
plt.show()
```



Two Variables Analysis

Shows the crosstabulation

- **df.groupby**(by, [axis], [level], [sort], [group_keys], **kwargs)
  - Group aggregation based on one or more columns of the DataFrame itself
  - The group key could be : Series(columns) , Array, Dict, Function
- **Cooperation function : groupby().xxx()**
  - **.mean()**
  - **.count()**
  - **.agg() :** choose multiple functions
  - **.apply() :** use the function we created
- **pd.crosstab**(index, columns, [values], **kwargs)

```python
# Crosstabulation of gender and loan_status
data[['Loan_status', 'Gender', 'Loan_ID']]
.groupby(['Loan_status', 'Gender']).agg(['count'])
```

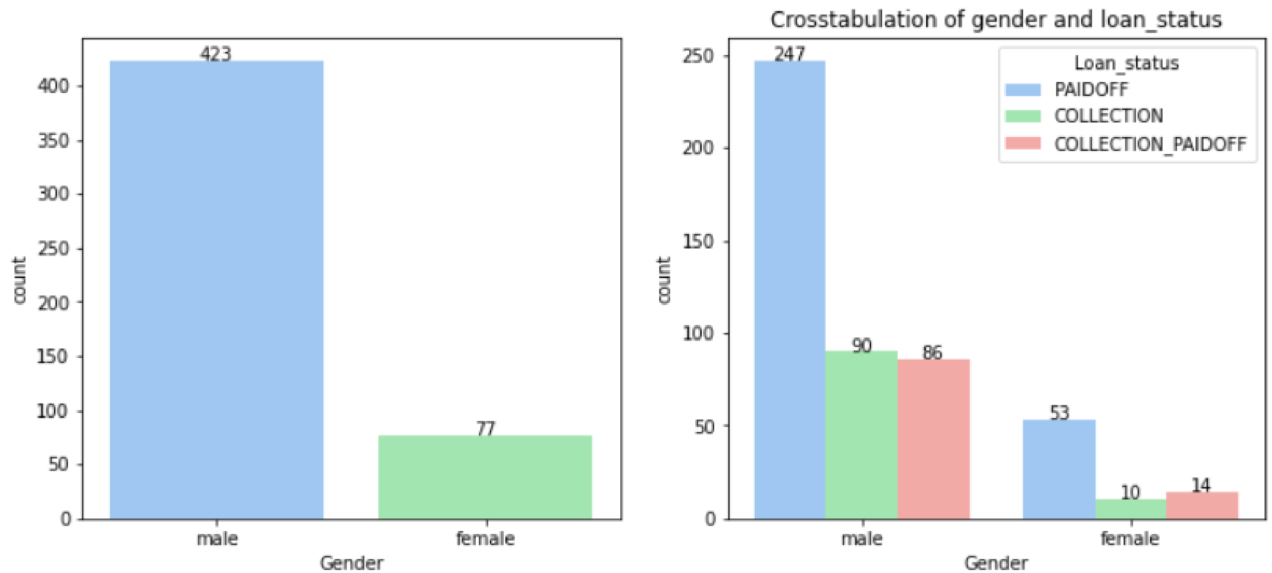
		Loan_ID
		count
Loan_status	Gender	
COLLECTION	female	10
	male	90
COLLECTION_PAIDOFF	female	14
	male	86
PAIDOFF	female	53
	male	247

Two Variables Analysis--Visualization

Side-by-Side Bar Chart

- **sns.countplot**(Series, hue = Series)
- **Create a crosstabulation + pd. plot**(kind='bar', **kwargs)

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))
sns.countplot(data['Gender'], ax=ax1)
sns.countplot(data['Gender'], hue=data['Loan_status'], ax=ax2)
for p in ax1.patches:
    height = p.get_height()
    ax1.text(p.get_x()+p.get_width()/2., height + 0.1, height ,ha="center", fontsize=10)
for p in ax2.patches:
    height = p.get_height()
    ax2.text(p.get_x()+p.get_width()/2., height + 0.1, height ,ha="center", fontsize=10)
plt.title('Crosstabulation of gender and loan_status')
plt.show()
```



Two Variables Analysis--Visualization

Stacked Bar Chart

- Create a crosstabulation + pd. plot(kind='bar', stacked=True, **kwargs)

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))
sns.countplot(data['Education'], ax=ax1)
subdata = pd.crosstab(data.Education, data.Loan_status)
subdata.plot(kind='bar', stacked=True, ax=ax2)
ax1.tick_params(labelrotation=30)
ax2.tick_params(labelrotation=30)
plt.xlabel(u"Education")
plt.ylabel(u"count")
plt.title('Crosstabulation of education and loan_status')
plt.show()
```