DATA SCIENCE 102: NUMPY & PANDAS

AGENDA

- Exploratory Data Analysis
- Why Pandas
- Introduction to Pandas
- Basic Statistics

DATA SCIENCE PROJECT STAGES

Problem Specification	Data Gathering & Preprocessing	Descriptive Analytics	Machine Learning	Deployment
• Understand business scenario • Define the project problem and scope • Define limitations of the project	• Develop a system to gather data • Clean and prepare raw data for processing • Usually the most time consuming stage in a data science project	• Exploratory Data Analysis • Basic understanding of the dataset • Answer the initial assumptions you may have of the data	• Depending on the scope/nature of your project, apply necessary machine learning models to tackle the problem • Train the machine learning model and assess its' performance	• Consult with project stakeholders on suitability of model • Deploy model for live usage

EXPLORATORY DATA ANALYSIS

- Main reasons for Exploratory Data Analysis
  - Detection of mistakes
  - Checking of assumptions
  - Preliminary selection of appropriate models
  - Identifying relationships among variables

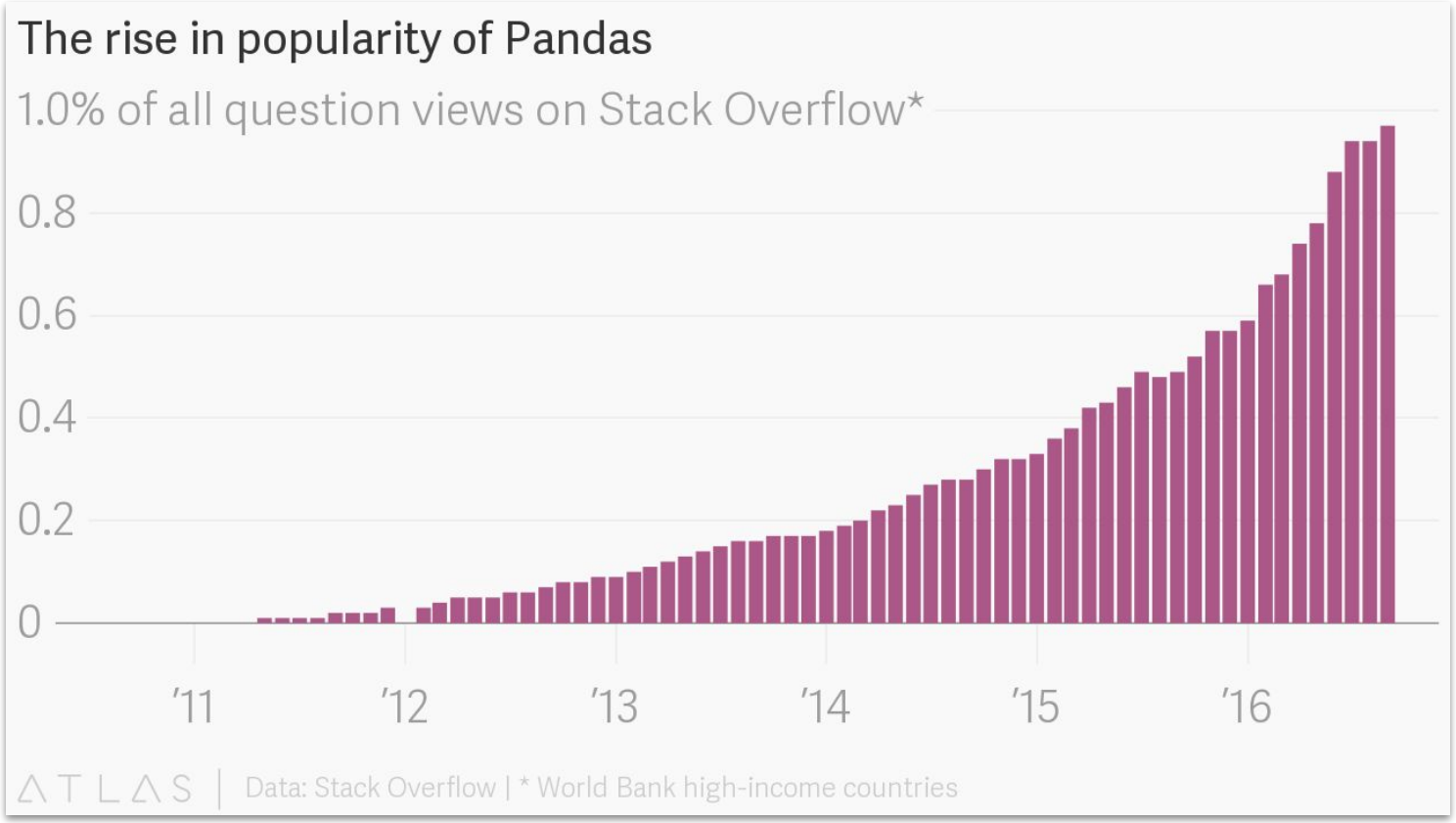
EXPLORATORY DATA ANALYSIS - METHODS OF ANALYSIS

- Graphical
  - Summarize data in diagrammatic or pictorial ways
- Non-graphical
  - Generally involve calculation of summary statistics

WHY PANDAS

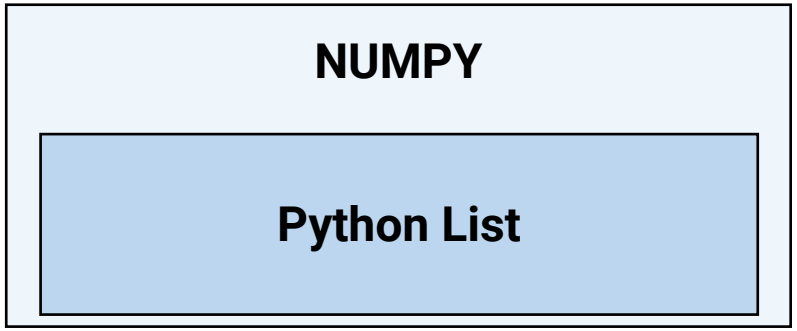
WHY PANDAS?

1. Extremely popular, in-demand package for data science and analytics
2. High performance, even higher convenience
3. Open source - absolutely free

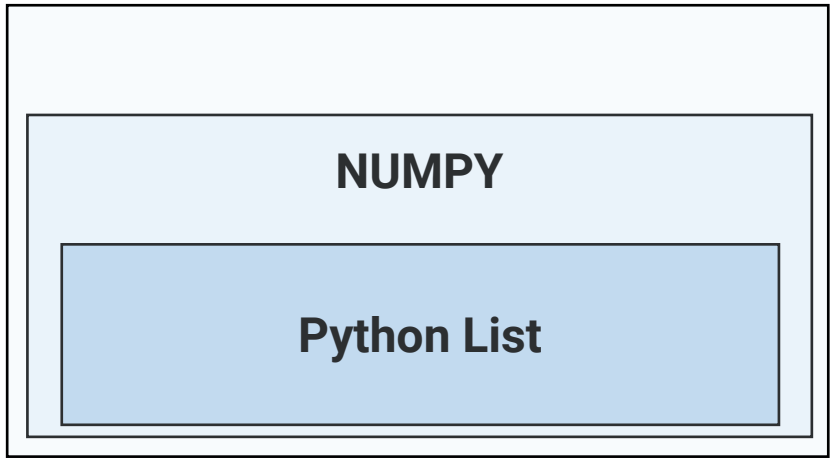


PANDAS & OTHER STRUCTURES

Python List: As we learnt in DS101, one of the main reference variables commonly used in python is Python list. While it is easy to use, it is not very efficient and suitable for numeric computing since as linear algebra



Numpy: Numpy is a python library for that makes working with 2D arrays a lot simpler and faster. The way you use a numpy object is similar to that of a python list. As such when you use Numpy array you are essentially using a python list with added functionality to make your life easier.



Pandas: Pandas is a python library for data analysis. Data storage in Pandas is implemented using Numpy's array. As such when you use Pandas data structures you are actually using a Numpy array that has even greater functionality to make your life even easier.

INTRODUCTION TO PANDAS

- Series
- DataFrame
  - Attributes
  - Methods
- Filtering
- Appending Data

SERIES

`Series` is a one-dimensional array with homogeneous data. All the elements of series should be of **same data type**.

Key Features of a Series:

- Homogeneous data
- Size Immutable – size cannot be changed (Means you can't add new data)
- Values of Data Mutable (Means you can change the values of existing data)

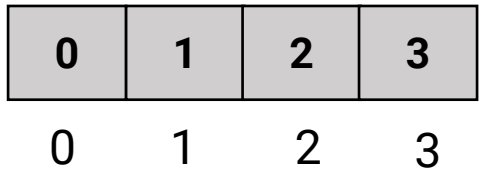
SERIES

Pandas Series

A Series is a one-dimensional object similar to an array, list, or column in a table. It will assign a labelled index to each item in the Series. By default, each item will receive an index label from 0 to N, where N is the length of the Series minus one

Python List

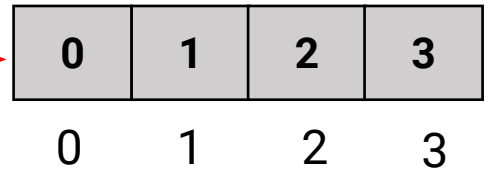
```
X = [0, 1, 2, 3]
```



While the structure looks alike, Numpy Array are stored more efficiently than list

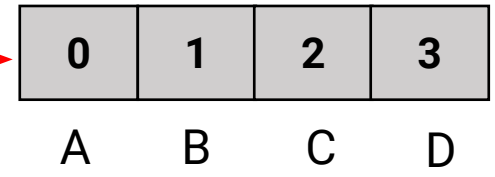
Numpy Array (Aka List)

```
import numpy as np
ax = np.array(x, int)
```



Pandas Series (Aka List)

```
import pandas as pd
ax = pd.Series(ax)
```



Numpy and pandas series are almost similar, except that in pandas you can label the index in a non-integer format like a python dictionary

SERIES

If you stack multiple Series objects together, side-by-side, or like a stack of CDs, they'd resemble the shape of a table. And that's precisely how you should visualise DataFrames

DataFrames are simply stacks of Series objects put together

Currency	Value	Change	Net Change	Time (EDT)	2 Day
EUR-USD	1.1412	-0.0011	-0.10%	6:51 AM	NaN
USD-JPY	113.6700	0.4500	+0.40%	6:51 AM	NaN
GBP-USD	1.2895	-0.0076	-0.59%	6:51 AM	NaN
AUD-USD	0.7595	0.0009	+0.12%	6:51 AM	NaN
USD-CAD	1.2974	-0.0004	-0.03%	6:50 AM	NaN
USD-CHF	0.9626	0.0022	+0.23%	6:50 AM	NaN
EUR-JPY	129.7300	0.4000	+0.31%	6:50 AM	NaN
EUR-GBP	0.8850	0.0044	+0.50%	6:50 AM	NaN
USD-HKD	7.8113	0.0008	+0.01%	6:50 AM	NaN
EUR-CHF	1.0985	0.0016	+0.15%	6:51 AM	NaN
USD-KRW	1154.3200	-3.0600	-0.26%	2:29 AM	NaN

DATAFRAME

Dataframe

A DataFrame is a tabular data structure comprised of rows and columns, akin to a spreadsheet, or database table. You can also think of a DataFrame as a group of Series objects that share an index

Currency	Value	Change	Net Change	Time (EDT)	2 Day
EUR-USD	1.1412	-0.0011	-0.10%	6:51 AM	NaN
USD-JPY	113.6700	0.4500	+0.40%	6:51 AM	NaN
GBP-USD	1.2895	-0.0076	-0.59%	6:51 AM	NaN
AUD-USD	0.7595	0.0009	+0.12%	6:51 AM	NaN
USD-CAD	1.2974	-0.0004	-0.03%	6:50 AM	NaN
USD-CHF	0.9626	0.0022	+0.23%	6:50 AM	NaN
EUR-JPY	129.7300	0.4000	+0.31%	6:50 AM	NaN
EUR-GBP	0.8850	0.0044	+0.50%	6:50 AM	NaN
USD-HKD	7.8113	0.0008	+0.01%	6:50 AM	NaN
EUR-CHF	1.0985	0.0016	+0.15%	6:51 AM	NaN
USD-KRW	1154.3200	-3.0600	-0.26%	2:29 AM	NaN

The rows and columns of dataframe are actually made up of Pandas series

DATAFRAME - CREATION

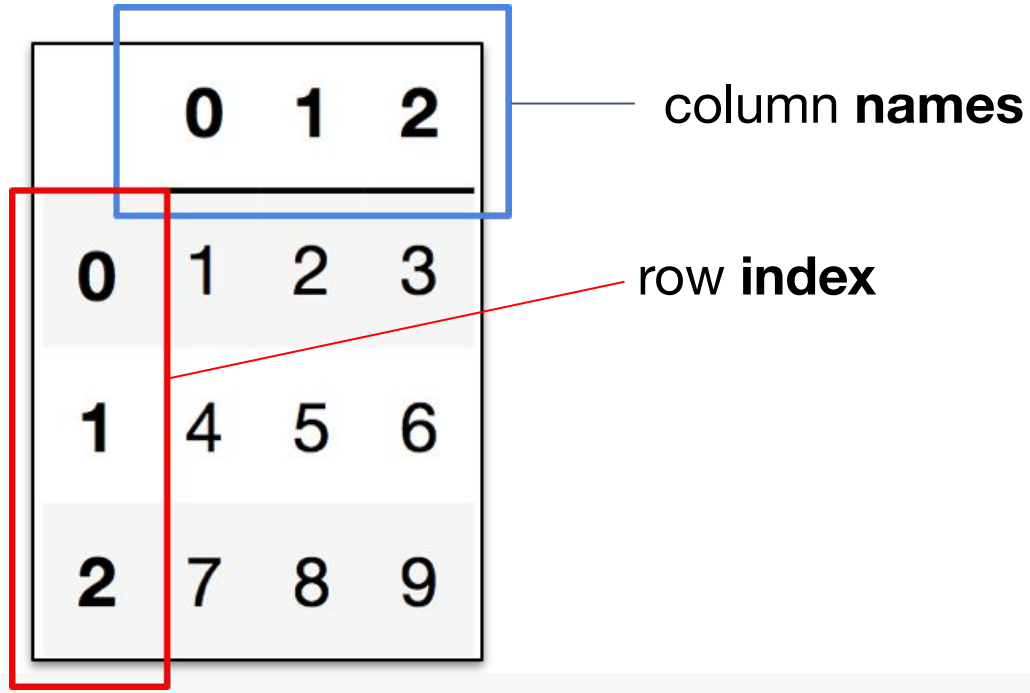
- There are numerous ways to create a pandas dataframe; one way is with a nested list

- The row indexes and column names serve the same purpose as the rows and columns in Excel, where you reference a cell with a (row, column) *coordinate*

In [3]:

```
1  nested_list= [[1, 2, 3],
2                [4, 5, 6],
3                [7, 8, 9]]
4
5  pd.DataFrame(nested_list)
```

Out[3]:



DATAFRAME - CREATION

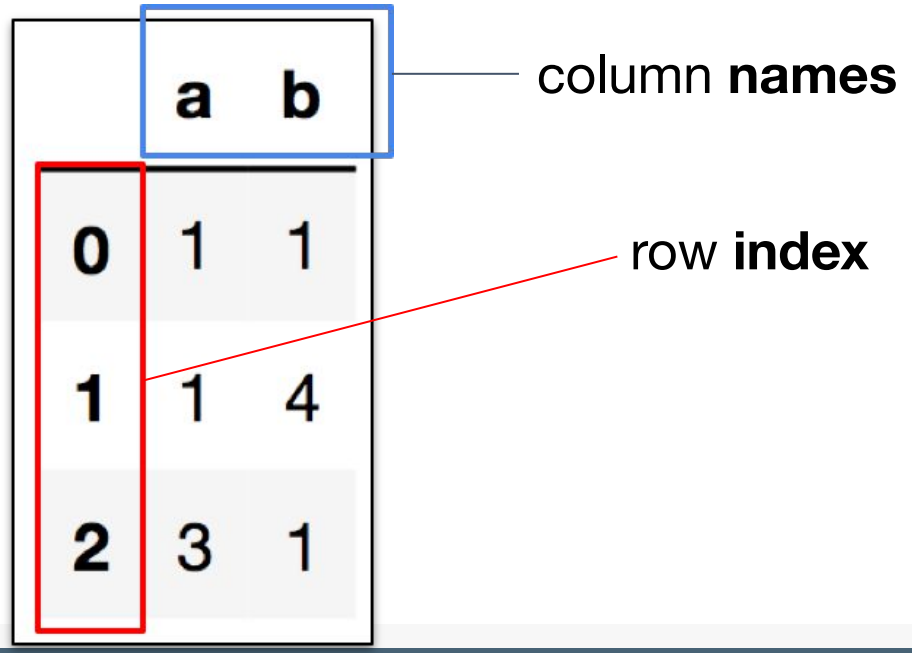
- Another way to create a dataframe is with a dictionary, where the key-value pairs are

  `{col_name: [values]}`

- You can construct a dataframe with numerous data structures. However, you rarely need to build and populate a dataframe with values (like a list in python), as the next slide will show

In [5]:

```
1  dictionary = {
2      'a': [1, 2, 3],
3      'b': [4, 5, 6]
4  }
5
6  pd.DataFrame(dictionary)
```



DATAFRAME - CREATION

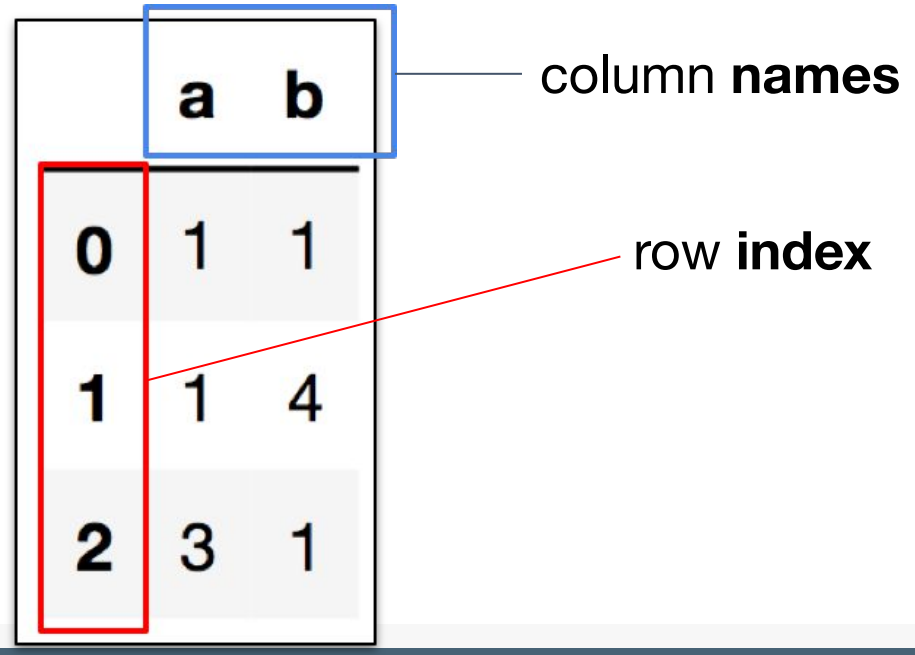
- Another way to create a dataframe is with a dictionary, where the key-value pairs are

  `{col_name: [values]}`

- You can construct a dataframe with numerous data structures. However, you rarely need to build and populate a dataframe with values (like a list in python), as the next slide will show

In [5]:

```
1  dictionary = {
2      'a': [1, 2, 3],
3      'b': [4, 5, 6]
4  }
5
6  pd.DataFrame(dictionary)
```



DATAFRAME - CREATION

We typically “read” data files into Python and stick them directly into dataframes using this lovely function.

```
In [2]:  1  df = pd.read_csv('employees-1k.csv')
```

pd.DataFrame object
2-dimensional collection that looks just like a table / spreadsheet. It has many useful *attributes* & *methods* for us to work with the data.

DATAFRAME - ATTRIBUTES

```
In [3]:  1  df.shape

Out[3]: (1000, 4)
```

```
In [4]:  1  print(df.dtypes)

employee_id          int64
annual_inc         float64
employee_title      object
home_ownership      object
dtype: object
```

```
In [5]:  1  print(df.columns)

Index(['employee_id', 'annual_inc', 'emp
```

Attributes are variables that are “attached” to an object and are accessible with the . (dot) operator. They provide us with useful information.

For example, we can obtain the shape (dimensions), data types and columns of the current df with these 3 lines of code. Try it out!

DATAFRAME - METHODS

In [6]:
```
# Find the first few records with .head()
# Add 10 as parameter in head() to show first 10 records

df.head()
```

Out[6]:

	employee_id	annual_inc	employee_title	home_ownership
0	44037036	55000.0	Account Specialist	OWN
1	41036228	184000.0	IT Architect	MORTGAGE
2	67601397	30000.0	night auditor	RENT
3	64972576	72000.0	Field Service Representative	MORTGAGE
4	59922079	84000.0	Senior Pastor	MORTGAGE

In [7]:
```
# Randomly sample (without replacement) with .sample()
# Add 8 as parameter to sample 8 records e.g. df.sample(8)
df.sample()
```

Out[7]:

	employee_id	annual_inc	employee_title	home_ownership
323	1739637	43000.0	JH Cohn LLP	MORTGAGE

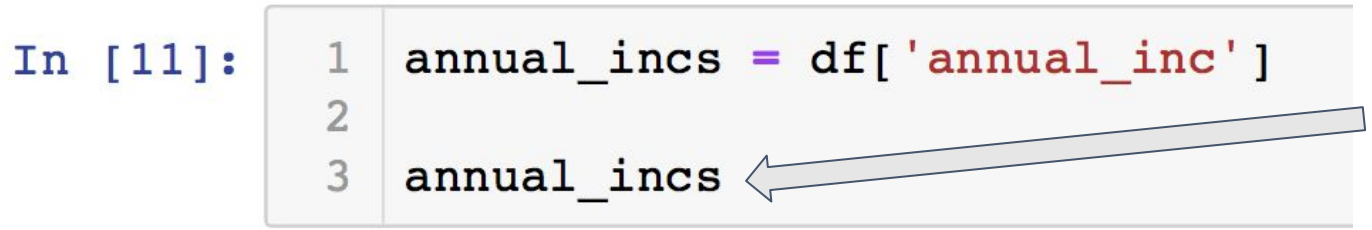
Methods are functions that are “attached” to an object. They provide us with useful functionality

For example, we can call the .head() method here to output the first 5 rows of our dataframe

Similarly, call the .sample() method to get a random row

Dataframes have many more methods for our convenience!

DATAFRAME - ACCESS BY COLUMN



```
In [11]:
1  annual_incs = df['annual_inc']
2
3  annual_incs
```

```
Out[11]:
0      55000.00
1     184000.00
2      30000.00
3      72000.00
4      84000.00
5      65000.00
6     130000.00
7     108000.00
8     150000.00
9     190000.00
10     52000.00
```

pd.Series object

Series objects are one-dimensional, unlike DataFrames. You can think of DataFrames as a bunch of *Series* objects stacked side-by-side

Like a DataFrame, it has many convenient methods with useful functionality

DATAFRAME - ACCESS BY ROW

In [13]:
```
1  # Retrieve one record from the df using .iloc[3]
2  row_3 = df.iloc[3]
3
4  row_3
```

Out[13]:
```
employee_id                                64972576
annual_inc                                    72000
employee_title     Field Service Representative
home_ownership                             MORTGAGE
Name: 3, dtype: object
```

In [14]:
```
1  # Given a record, specify the column name to get the value
2  row_3['employee_id']
```

Out[14]:
```
64972576
```

You can also retrieve a particular row from a dataframe with the `.iloc` -- integer locate -- property.

A *Series* is also returned in this case, with its indexes corresponding to the DataFrame columns.

FILTERING - BOOLEAN ARRAY

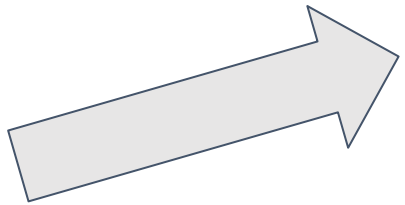
This is an example dataframe with all of its rows. Say we want the records of all employees annual income **>= 50000**, what do we do?
We can make use of a **boolean array** to filter the rows for us.

	employee_id	annual_inc	employee_title
0	44037036	55000.00	Account Specialist
1	41036228	184000.00	IT Architect
2	67601397	30000.00	night auditor
3	64972576	72000.00	Field Service Representative
4	59922079	84000.00	Senior Pastor
	...		
995	59649746	62000.0	instrument fitter
996	40523409	32000.0	Laborer
997	26637691	32000.0	Waitress
998	44806874	70000.0	Case Manager
999	64019889	58000.0	Machinist

FILTERING - BOOLEAN ARRAY

```
In [6]:  1  df['annual_inc'] >= 50000
```

```
Out[6]: 0      True
        1      True
        2     False
        3      True
        4      True
        5      True
        6      True
        7      True
        8      True
        9      True
        10     True
        11    False
        12     True
        13     True
        14     True
        15    False
        16    False
        17    False
```



A boolean array is returned from this comparison statement.

We can use this boolean array to filter our dataframe!

FILTERING - BOOLEAN ARRAY

```
df[df['annual_inc'] >= 50000]
```

df

```
0       True
1       True
2      False
3       True
4       True
    ...
995     True
996    False
997    False
998     True
999     True
```

All rows with `False` will be excluded from the dataframe.
Thus, this line will return a dataframe of rows where `'annual_inc' >= 50000`

FILTERING - BOOLEAN ARRAY

```
df[df['annual_inc'] >= 50000]
```

	employee_id	annual_inc	employee_title
0	44037036	55000.00	Account Specialist
1	41036228	184000.00	IT Architect
2	67601397	30000.00	night auditor
3	64972576	72000.00	Field Service Representative
4	59922079	84000.00	Senior Pastor
...			
995	59649746	62000.0	instrument fitter
996	40523409	32000.0	Laborer
997	26637691	32000.0	Waitress
998	44806874	70000.0	Case Manager
999	64019889	58000.0	Machinist

0	True
1	True
2	False
3	True
4	True
...	
995	True
996	False
997	False
998	True
999	True

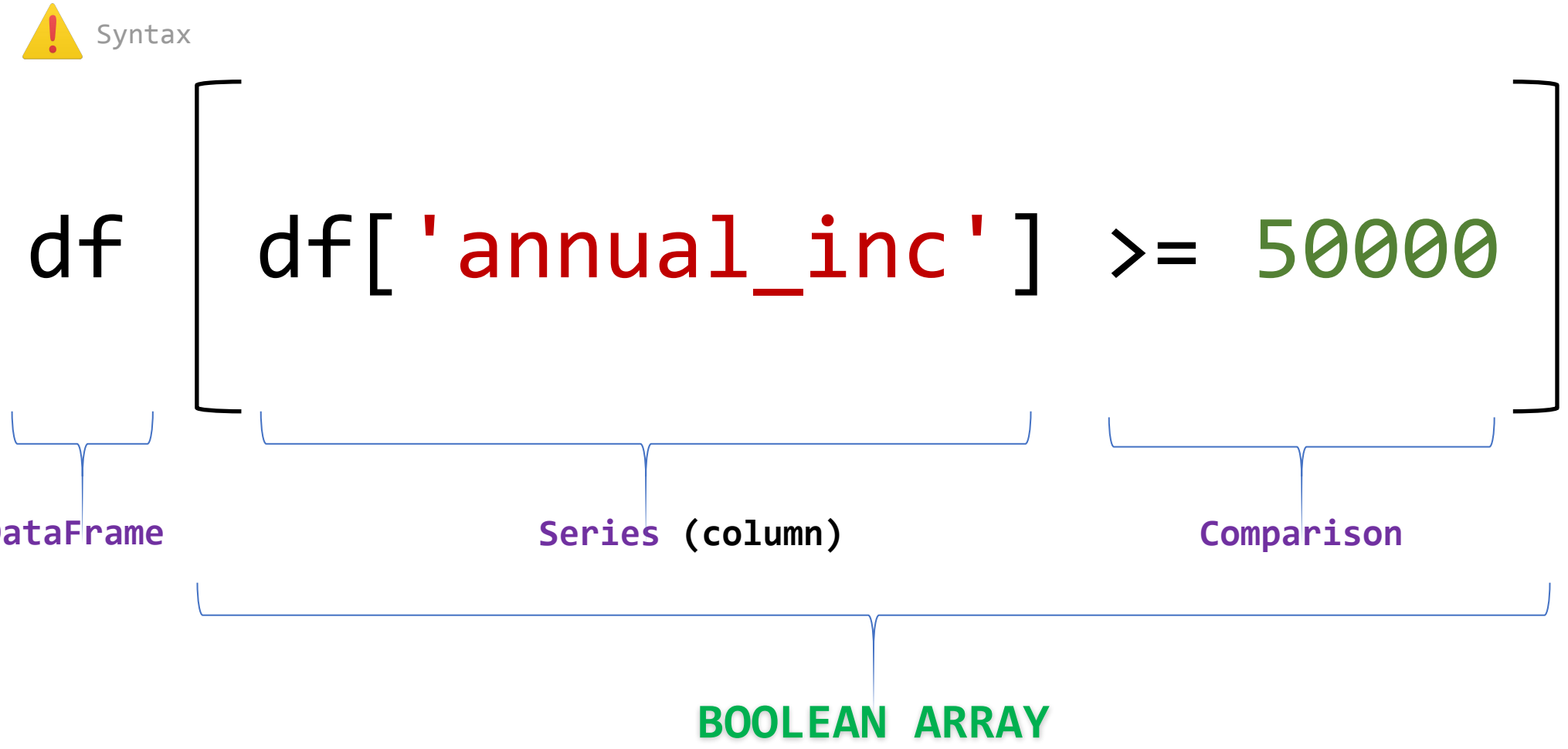
FILTERING - BOOLEAN ARRAY

```
df[df['annual_inc'] >= 50000]
```

	employee_id	annual_inc	employee_title
0	44037036	55000.00	Account Specialist
1	41036228	184000.00	IT Architect
3	64972576	72000.00	Field Service Representative
4	59922079	84000.00	Senior Pastor
...			
995	59649746	62000.0	instrument fitter
998	44806874	70000.0	Case Manager
999	64019889	58000.0	Machinist

0	True
1	True
3	True
4	True
...	
995	True
998	True
999	True

FILTERING - BOOLEAN ARRAY



FILTERING - BOOLEAN ARRAY

AND

```
df[(_____) & (_____)]
```

OR

```
df[(_____) | (_____)]
```

Previously we filtered a dataframe with only one condition. What if there are two?

All you have to do is follow this syntax.

Each condition rests in a bracket and is separated by the & or | operator.

FILTERING - BOOLEAN ARRAY

AND

```
df[(df['annual_inc'] >= 50000) & (df['home_ownership'] == 'OWN')]
```

Out[8]:

	employee_id	annual_inc	employee_title	home_ownership
0	44037036	55000.0	Account Specialist	OWN
25	34691613	50000.0	General mgr	OWN
32	36665106	73000.0	VP Operations	OWN
43	10018508	75000.0	Police Officer	OWN
66	899241	153000.0	united states air force	OWN
82	42134060	55000.0	Career and Tech Ed Specialist	OWN
97	21047758	98000.0	Assistant Vice President	OWN

FILTERING - BOOLEAN ARRAY

OR

```
df[(df['annual_inc'] >= 100000) | (df['home_ownership'] == 'OWN')]
```

Out[9]:

	employee_id	annual_inc	employee_title	home_ownership
0	44037036	55000.0	Account Specialist	OWN
1	41036228	184000.0	IT Architect	MORTGAGE
6	64816480	130000.0	Sr. System Engineer	RENT
7	37075903	108000.0	President	RENT
8	26377546	150000.0	Owner	MORTGAGE
9	22974788	190000.0	Vice President, SOuthwest Division	MORTGAGE
12	63058354	135000.0	Managing Director	MORTGAGE
19	35845083	45000.0	claims	OWN

BASIC STATISTICS

- Categorical vs. Quantitative Data
- Summary Statistics
- Descriptive Statistics with Pandas

CATEGORICAL VS. QUANTITATIVE

Categorical data

We care about:

The range of values and (relative) frequency of occurrence for each value.

	employee_id	annual_inc	employee_title	home_ownership
0	44037036	55000.00	Account Specialist	OWN
1	41036228	184000.00	IT Architect	MORTGAGE
2	67601397	30000.00	night auditor	RENT
3	64972576	72000.00	Field Service Representative	MORTGAGE
4	59922079	84000.00	Senior Pastor	MORTGAGE
5	4417511	65000.00	Market Resource Partners	RENT
6	64816480	130000.00	Sr. System Engineer	RENT
7	37075903	108000.00	President	RENT
8	26377546	150000.00	Owner	MORTGAGE
9	22974788	190000.00	Vice President, SOuthwest Division	MORTGAGE
10	20967626	52000.00	senior associate rep	RENT
11	34050773	26000.00	EHS	MORTGAGE
12	63058354	135000.00	Managing Director	MORTGAGE
13	72648159	84840.00	Carpenter	MORTGAGE
14	14978604	60000.00	Principal	MORTGAGE
15	47033713	48000.00	Owner	RENT
16	58339021	44459.22	Planner Buyer	RENT
17	42363432	49000.00	recieving lead	MORTGAGE
18	14678996	59300.00	District Treasurer	MORTGAGE
19	35845083	45000.00	claims	OWN

CATEGORICAL VS. QUANTITATIVE

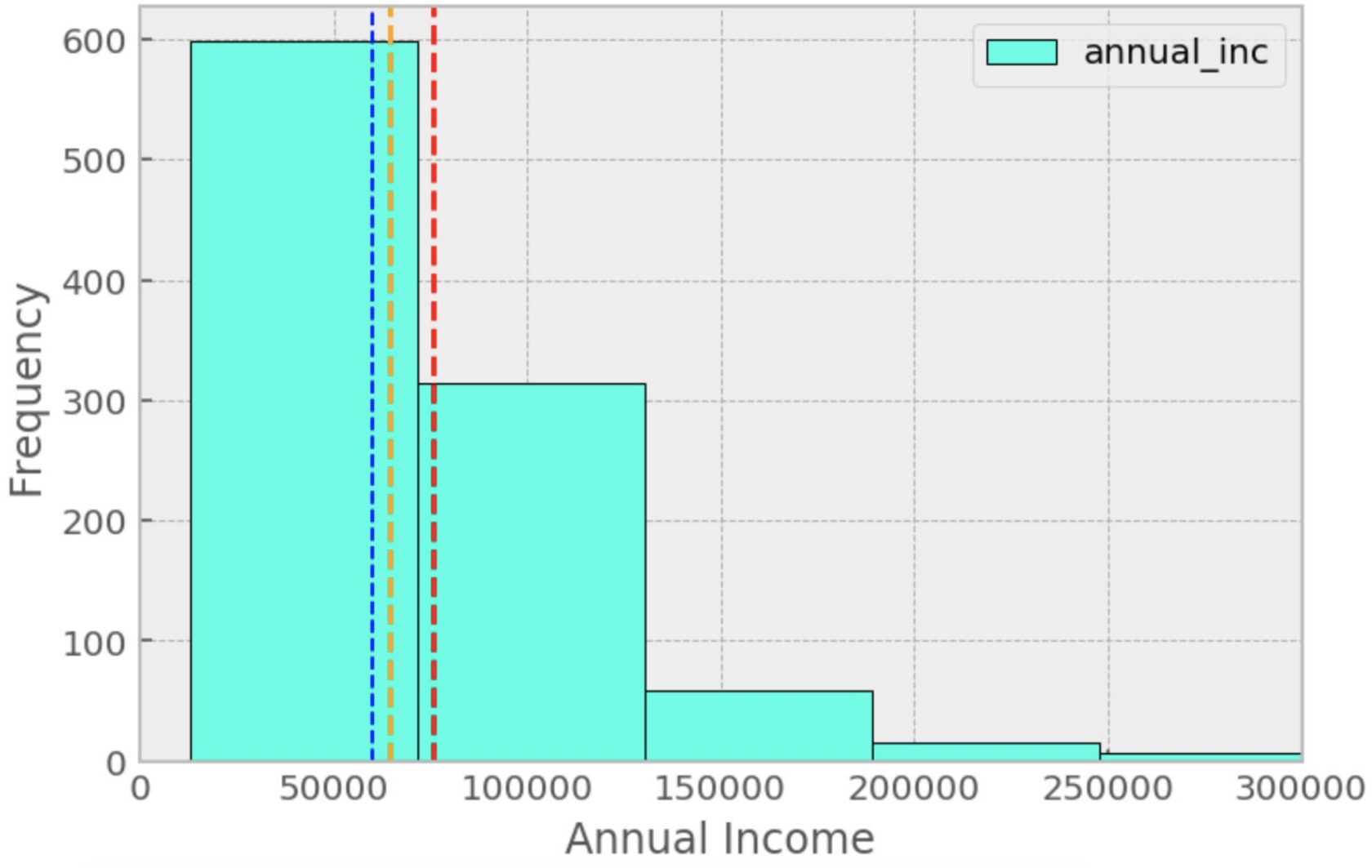
Quantitative Data

Make preliminary assessments about the population using the data of a chosen sample.

Based on the characteristics of a few, we can make predictions about the many

	employee_id	annual_inc	employee_title	home_ownership
0	44037036	55000.00	Account Specialist	OWN
1	41036228	184000.00	IT Architect	MORTGAGE
2	67601397	30000.00	night auditor	RENT
3	64972576	72000.00	Field Service Representative	MORTGAGE
4	59922079	84000.00	Senior Pastor	MORTGAGE
5	4417511	65000.00	Market Resource Partners	RENT
6	64816480	130000.00	Sr. System Engineer	RENT
7	37075903	108000.00	President	RENT
8	26377546	150000.00	Owner	MORTGAGE
9	22974788	190000.00	Vice President, SOuthwest Division	MORTGAGE
10	20967626	52000.00	senior associate rep	RENT
11	34050773	26000.00	EHS	MORTGAGE
12	63058354	135000.00	Managing Director	MORTGAGE
13	72648159	84840.00	Carpenter	MORTGAGE
14	14978604	60000.00	Principal	MORTGAGE
15	47033713	48000.00	Owner	RENT
16	58339021	44459.22	Planner Buyer	RENT
17	42363432	49000.00	recieving lead	MORTGAGE
18	14678996	59300.00	District Treasurer	MORTGAGE
19	35845083	45000.00	claims	OWN

STATISTICS - CENTRAL TENDENCY



Mean

- Average. Also known as the expected value of a set of data values.

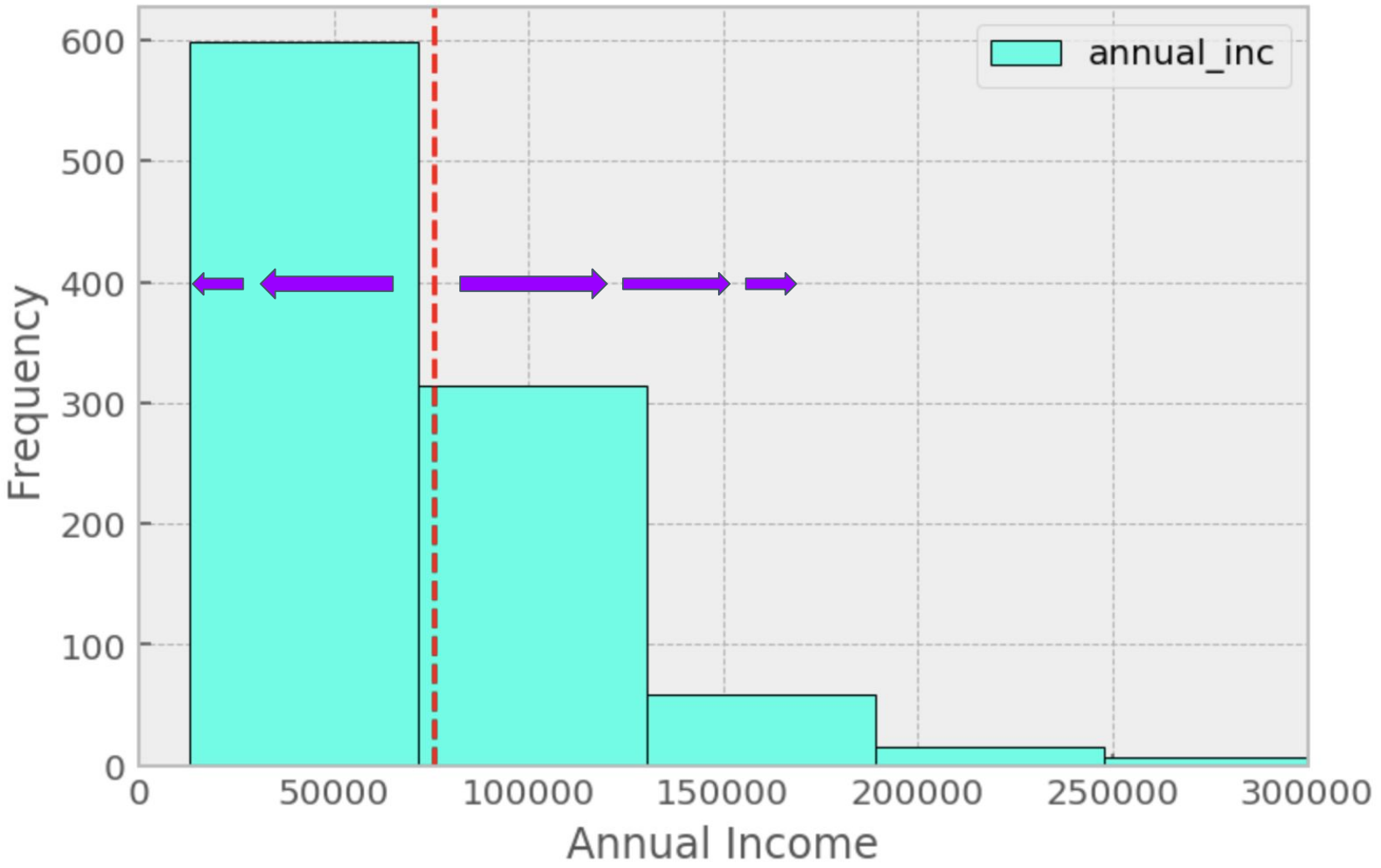
Median

- The midpoint of the sorted set of data values.

Mode

- The value / datapoint that appears most often.

STATISTICS - SPREAD



Variance

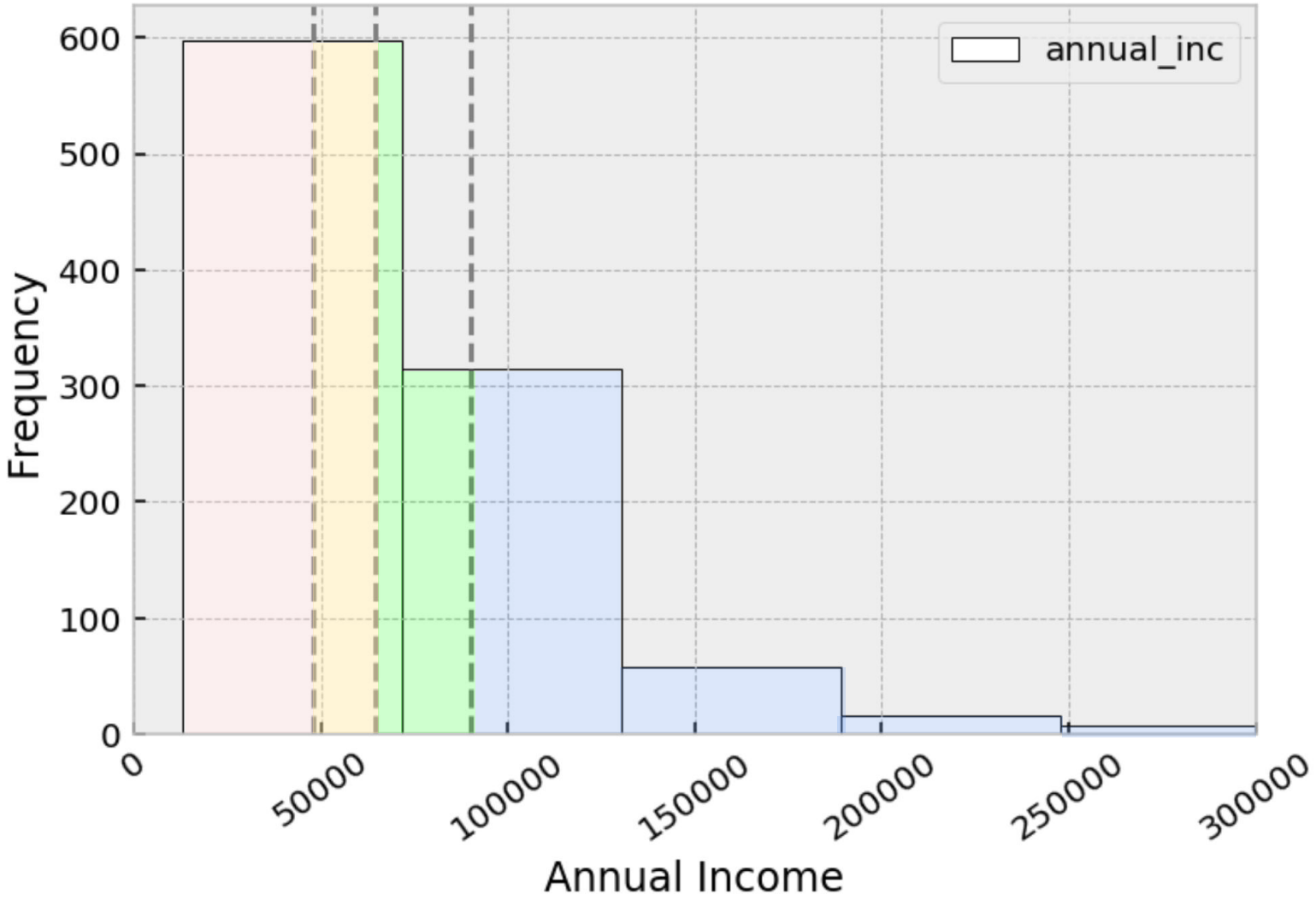
- Mean squared distance of any value from the mean of the distribution

Standard Deviation

- Square root of variance.

The higher the values of each, the greater the spread / variation of the data.

STATISTICS - SPREAD

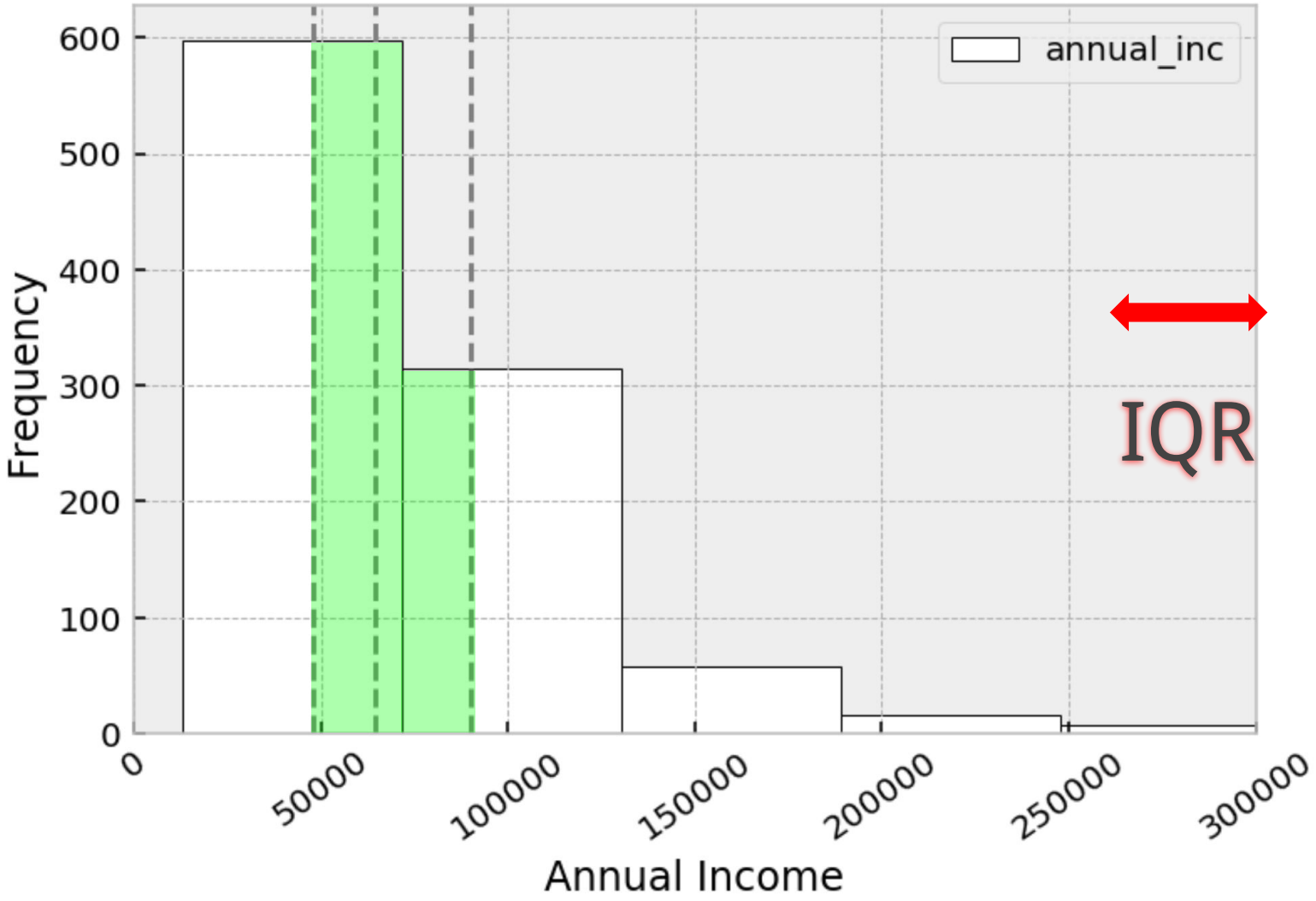


Quartiles

Divides the data into 4 parts, each separated from another by points known as quartiles;

25%, 50%, 75%

STATISTICS - SPREAD



Interquartile Range (IQR)

Half of the values (specifically the middle half) fall within the IQR.

```
IQR = Q3 - Q1
```

STATISTICS - SPREAD

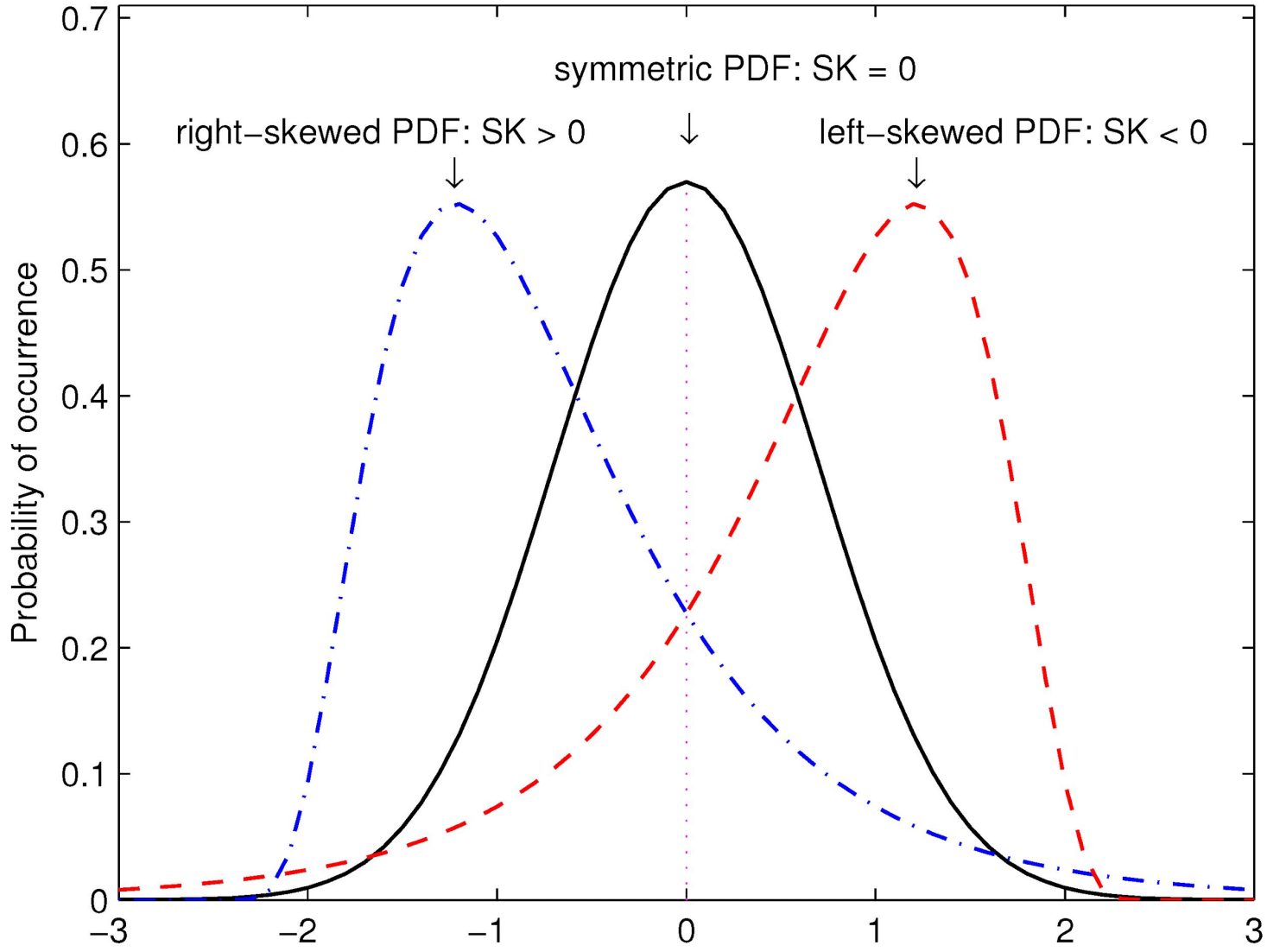
Five number summary

The five number summary provides a intuitive snapshot of the distribution of the data.
It shows the range of values the data takes.

```
[10, 17, 22, 30, 41, 56]
```

min Q1 Q2 Q3 max

STATISTICS - SKEWNESS

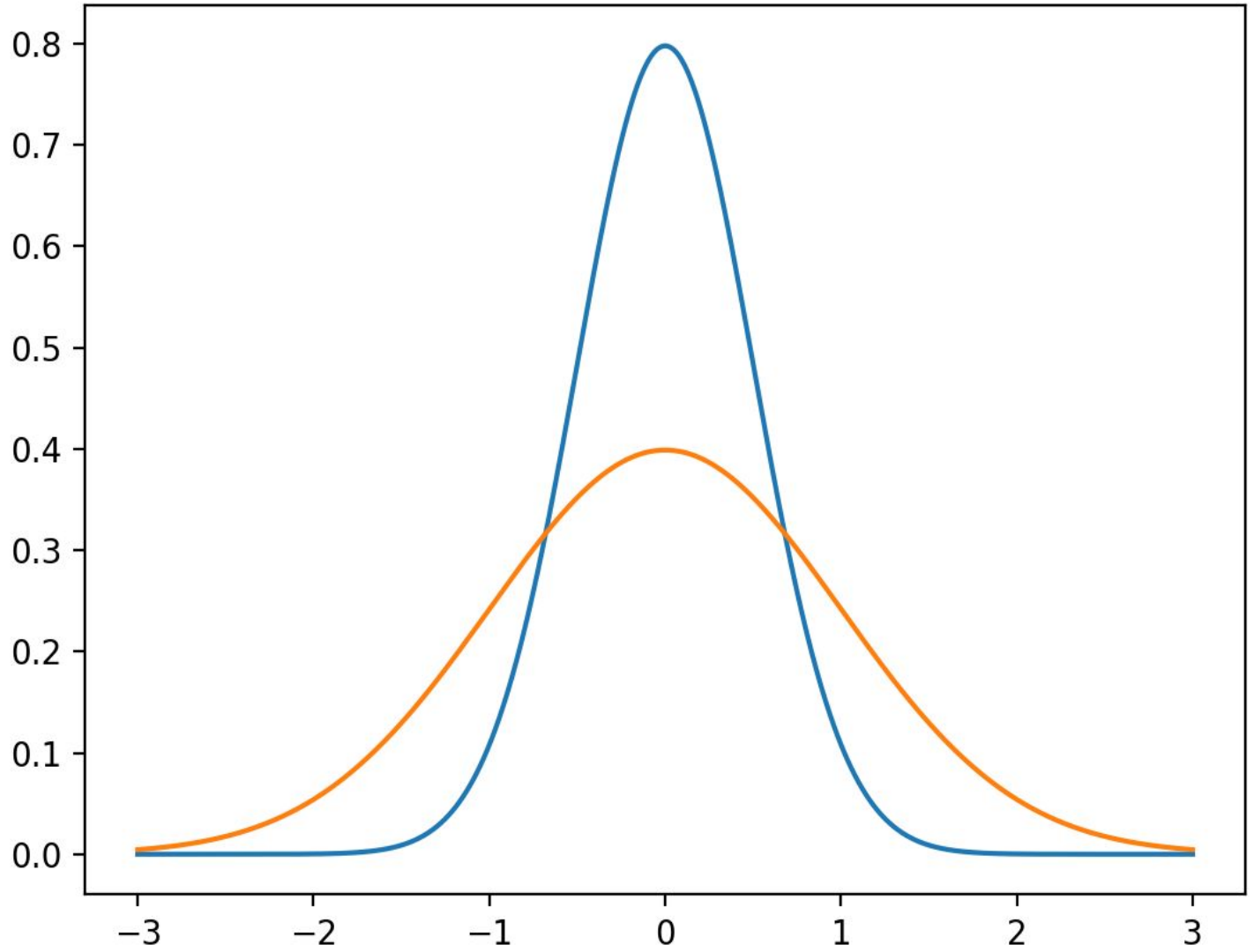


Measure of (a)symmetry of a distribution.

As a general rule of thumb:

Value	Skewness
-1 < SK > 1	High
-1 < SK < -0.5	Moderate
-0.5 < SK < 0.5	Low

STATISTICS - KURTOSIS



Measure of the *peakedness* or *flatness* of a distribution

As a general rule of thumb:

Value	Peakedness
> 3	High
0	Moderate
< -3	Low

DESCRIPTIVE STATISTICS WITH PANDAS

Mean of each column

```
df.mean()
```

Mean of the column

```
df['col'].mean()
```

Median of each column

```
df.median()
```

Median of the column

```
df['col'].median()
```

Summary statistics of each column

```
df.describe()
```

Summary statistics of the column

```
df['col'].describe()
```

DESCRIPTIVE STATISTICS WITH PANDAS

```
df.describe()
```

	points	price
count	50000.000000	50000.000000
mean	87.781120	33.041220
std	3.230892	34.710394
min	80.000000	4.000000
25%	86.000000	16.000000
50%	88.000000	24.000000
75%	90.000000	40.000000
max	100.000000	1200.000000

The **five-number summary** is a set of descriptive statistics that provide some useful information about a dataset. It is made up of the five most important sample percentiles:

1. the sample minimum (smallest observation)
2. the lower quartile (25th percentile)
3. the median (50th percentile)
4. the upper quartile (75th percentile)
5. the sample maximum (largest observation)

DESCRIPTIVE STATISTICS WITH PANDAS

*Skewness of **each** column*

```
df.skew()
```

Skewness of a single column

```
df['col'].skew()
```

*Kurtosis of **each** column*

```
df.kurtosis()
```

Kurtosis of a single column

```
df['col'].kurtosis()
```

Skewness (e) or kurtosis (u)	Conclusion
$-2\mathrm{SE}(\mathrm{e}) < e < 2\mathrm{SE}(\mathrm{e})$	not skewed
$e \leq -2\mathrm{SE}(\mathrm{e})$	negative skew
$e \geq 2\mathrm{SE}(\mathrm{e})$	positive skew
$-2\mathrm{SE}(\mathrm{u}) < u < 2\mathrm{SE}(\mathrm{u})$	not kurtotic
$u \leq -2\mathrm{SE}(\mathrm{u})$	negative kurtosis
$u \geq 2\mathrm{SE}(\mathrm{u})$	positive kurtosis

Source: http://www.stat.cmu.edu/~hseltman/309/Book/Book.pdf