Exploratory Data Analysis (EDA) with Numpy & Pandas

Hello!

I am Eslam Ahmed

I am a software engineer.

You can find me at jeksogsa@gmail.com

Agenda

- What is Numpy
- Install and use Numpy
- Create Numpy Array
- Built in Methods
- Reshaping
- Indexing
- Selection
- Arithmetic and Logic
- Universal Array Functions
- Combine and split
- What is Pandas
- Series
- Data Frames
- Deal with Missing data
- Grouping data and aggregate functions
- Merging, Joining and Concatenating
- Pivoting
- Useful Methods
- Apply function
- Data Input & Output

Agenda

- What is Numpy
- Install and use Numpy
- Create Numpy Array
- Built in Methods
- Reshaping
- Indexing
- Selection
- Arithmetic and Logic
- Universal Array Functions
- Combine and split
- What is Pandas
- Series
- Data Frames
- Deal with Missing data
- Grouping data and aggregate functions
- Merging, Joining and Concatenating
- Pivoting
- Useful Methods
- Apply function
- Data Input & Output

What is Numpy

NumPy is a Python package which stands for 'Numerical Python'. It is a library consisting of multidimensional array objects and a collection of routines for processing of array.

The most important object defined in NumPy is an N-dimensional array type called **ndarray**

Agenda

- What is Numpy
- **Install and use Numpy**
- Create Numpy Array
- Built in Methods
- Reshaping
- Indexing
- Selection
- Arithmetic and Logic
- Universal Array Functions
- Combine and split
- What is Pandas
- Series
- Data Frames
- Deal with Missing data
- Grouping data and aggregate functions
- Merging, Joining and Concatenating
- Pivoting
- Useful Methods
- Apply function
- Data Input & Output

Install and use Numpy

Install

```
1 >_ conda install numpy
```

Use

```
1 import numpy as np
```

Agenda

- What is Numpy
- Install and use Numpy
- **Create Numpy Array**
- Built in Methods
- Reshaping
- Indexing
- Selection
- Arithmetic and Logic
- Universal Array Functions
- Combine and split
- What is Pandas
- Series
- Data Frames
- Deal with Missing data
- Grouping data and aggregate functions
- Merging, Joining and Concatenating
- Pivoting
- Useful Methods
- Apply function
- Data Input & Output

Create Numpy Array

Numpy arrays essentially come in two Flavors vectors and matrices. Vectors are strictly 1-d arrays and matrices are 2-d, but you should note a matrix can still have only one row or one column.

```
my_list = [1,2,3]
np.array(my_list)

"""
array([1, 2, 3])
"""


my_matrix = [[1,2,3],[4,5,6],[7,8,9]]
np.array(my_matrix)

"""
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
"""
```

Agenda

- What is Numpy
- Install and use Numpy
- Create Numpy Array
- **Built in Methods**
- Reshaping
- Indexing
- Selection
- Arithmetic and Logic
- Universal Array Functions
- Combine and split
- What is Pandas
- Series
- Data Frames
- Deal with Missing data
- Grouping data and aggregate functions
- Merging, Joining and Concatenating
- Pivoting
- Useful Methods
- Apply function
- Data Input & Output

Built in Methods

- arange
- zeros
- ones
- eye
- linspace
- rand
- randn
- randint
- max
- argmax
- min
- argmin

Agenda

- What is Numpy
- Install and use Numpy
- Create Numpy Array
- Built in Methods
- **Reshaping**
- Indexing
- Selection
- Arithmetic and Logic
- Universal Array Functions
- Combine and split
- What is Pandas
- Series
- Data Frames
- Deal with Missing data
- Grouping data and aggregate functions
- Merging, Joining and Concatenating
- Pivoting
- Useful Methods
- Apply function
- Data Input & Output

Reshaping

```python
arr = np.arange(25)

arr.reshape((5, 5))

"""
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
"""
```

Agenda

- What is Numpy
- Install and use Numpy
- Create Numpy Array
- Built in Methods
- Reshaping
- **Indexing**
- Selection
- Arithmetic and Logic
- Universal Array Functions
- Combine and split

- What is Pandas
- Series
- Data Frames
- Deal with Missing data
- Grouping data and aggregate functions
- Merging, Joining and Concatenating
- Pivoting
- Useful Methods
- Apply function
- Data Input & Output

Indexing

- Indexing 1D array
- Indexing 2D array
- Fancy Indexing

```
arr = np.arange(30,40)

"""
array([30, 31, 32, 33, 34, 35, 36, 37, 38, 39])
"""



arr[3:7]

"""
array([33, 34, 35, 36])
"""
```

Indexing

- Indexing 1D array
- **Indexing 2D array**
- Fancy Indexing

```python
arr_2d = np.array(([5,10,15],[20,25,30],[35,40,45]))

"""
array([[ 5, 10, 15],
       [20, 25, 30],
       [35, 40, 45]])
"""



arr_2d[:2,1:]

"""
array([[10, 15],
       [25, 30]])
"""
```

Indexing

- Indexing 1D array
- Indexing 2D array
- Fancy Indexing

```
"""
array([[ 2.,  2.,  2.,  2.,  2.,  2.,  2.,  2.,  2.,  2.],
       [ 3.,  3.,  3.,  3.,  3.,  3.,  3.,  3.,  3.,  3.],
       [ 4.,  4.,  4.,  4.,  4.,  4.,  4.,  4.,  4.,  4.],
       [ 5.,  5.,  5.,  5.,  5.,  5.,  5.,  5.,  5.,  5.],
       [ 6.,  6.,  6.,  6.,  6.,  6.,  6.,  6.,  6.,  6.],
       [ 7.,  7.,  7.,  7.,  7.,  7.,  7.,  7.,  7.,  7.],
       [ 8.,  8.,  8.,  8.,  8.,  8.,  8.,  8.,  8.,  8.],
       [ 9.,  9.,  9.,  9.,  9.,  9.,  9.,  9.,  9.,  9.],
       [10., 10., 10., 10., 10., 10., 10., 10., 10., 10.],
       [11., 11., 11., 11., 11., 11., 11., 11., 11., 11.]])
"""



arr2d[[1,4,6,8]]

"""
array([[ 3.,  3.,  3.,  3.,  3.,  3.,  3.,  3.,  3.,  3.],
       [ 6.,  6.,  6.,  6.,  6.,  6.,  6.,  6.,  6.,  6.],
       [ 8.,  8.,  8.,  8.,  8.,  8.,  8.,  8.,  8.,  8.],
       [10., 10., 10., 10., 10., 10., 10., 10., 10., 10.]])
"""
```

Agenda

- What is Numpy
- Install and use Numpy
- Create Numpy Array
- Built in Methods
- Reshaping
- Indexing
- **Selection**
- Arithmetic and Logic
- Universal Array Functions
- Combine and split
- What is Pandas
- Series
- Data Frames
- Deal with Missing data
- Grouping data and aggregate functions
- Merging, Joining and Concatenating
- Pivoting
- Useful Methods
- Apply function
- Data Input & Output

Selection

```
arr = np.arange(1,10)
"""
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
"""


arr[arr >= 5]
"""
array([5, 6, 7, 8, 9])
"""


arr[arr % 2 == 0]
"""
array([2, 4, 6, 8])
"""
```

Agenda

- What is Numpy
- Install and use Numpy
- Create Numpy Array
- Built in Methods
- Reshaping
- Indexing
- Selection
- **Arithmetic and Logic**
- Universal Array Functions
- Combine and split
- What is Pandas
- Series
- Data Frames
- Deal with Missing data
- Grouping data and aggregate functions
- Merging, Joining and Concatenating
- Pivoting
- Useful Methods
- Apply function
- Data Input & Output

Arithmetic and Logic

```
arr = np.arange(0,10)
"""
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
"""


arr + 3
"""
array([ 3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
"""


arr > 5
"""
array([False,False,False,False,False,False,True,True,True,True])
"""
```

Agenda

- What is Numpy
- Install and use Numpy
- Create Numpy Array
- Built in Methods
- Reshaping
- Indexing
- Selection
- Arithmetic and Logic
- **Universal Array Functions**
- Combine and split
- What is Pandas
- Series
- Data Frames
- Deal with Missing data
- Grouping data and aggregate functions
- Merging, Joining and Concatenating
- Pivoting
- Useful Methods
- Apply function
- Data Input & Output

Universal Array Functions

- max
- min
- average
- sum
- power
- sqrt
- exp
- log
- sin
- cos
- tan
- dot

Agenda

- What is Numpy
- Install and use Numpy
- Create Numpy Array
- Built in Methods
- Reshaping
- Indexing
- Selection
- Arithmetic and Logic
- Universal Array Functions
- **Combine and split**

- What is Pandas
- Series
- Data Frames
- Deal with Missing data
- Grouping data and aggregate functions
- Merging, Joining and Concatenating
- Pivoting
- Useful Methods
- Apply function
- Data Input & Output

Combine and split

- `hstack`
- `vstack`
- `concatenate`
- `hsplit`
- `vsplit`

Agenda

- What is Numpy
- Install and use Numpy
- Create Numpy Array
- Built in Methods
- Reshaping
- Indexing
- Selection
- Arithmetic and Logic
- Universal Array Functions
- Combine and split

- What is Pandas
- Series
- Data Frames
- Deal with Missing data
- Grouping data and aggregate functions
- Merging, Joining and Concatenating
- Pivoting
- Useful Methods
- Apply function
- Data Input & Output

What is Pandas

Pandas is an open-source Python Library used for high-performance data manipulation and data analysis using its powerful data structures.

Install and use Pandas

Install

```
1 >_ conda install pandas
```

Use

```
1 import pandas as pd
```

What is Pandas

A python library used for Data Wrangling

Key Features

- Fast and efficient Data Frame object with customized indexing.
- Tools for loading from different file formats.
- Data alignment and integrated handling of missing data.
- Reshaping and pivoting of date sets.
- Label-based slicing, indexing and subsetting of large data sets.
- Columns from a data structure can be deleted or inserted.
- Group by data for aggregation and transformations.
- High performance merging and joining of data.
- Time Series functionality.

Agenda

- What is Numpy
- Install and use Numpy
- Create Numpy Array
- Built in Methods
- Reshaping
- Indexing
- Selection
- Arithmetic and Logic
- Universal Array Functions
- Combine and split
- What is Pandas
- **Series**
- Data Frames
- Deal with Missing data
- Grouping data and aggregate functions
- Merging, Joining and Concatenating
- Pivoting
- Useful Methods
- Apply function
- Data Input & Output

Series

```python
labels = ['a','b','c']

my_list = [10,20,30]    # or numpy array
arr = np.array([10,20,30])
d = {'a':10,'b':20,'c':30}


pd.Series(data=my_list)
"""
0    10
1    20
2    30
"""


pd.Series(data=my_list, index=labels)
"""
a    10
b    20
c    30
"""


pd.Series(d)
"""
a    10
b    20
c    30
"""
```

Agenda

- What is Numpy
- Install and use Numpy
- Create Numpy Array
- Built in Methods
- Reshaping
- Indexing
- Selection
- Arithmetic and Logic
- Universal Array Functions
- Combine and split

- What is Pandas
- Series
- **Data Frames**
- Deal with Missing data
- Grouping data and aggregate functions
- Merging, Joining and Concatenating
- Pivoting
- Useful Methods
- Apply function
- Data Input & Output

Data Frames

```python
df = pd.DataFrame(np.random.randn(5,4),
        index=['A', 'B', 'C', 'D', 'E'],
        columns=['W', 'X', 'Y', 'Z'])
```

	W	X	Y	Z
A	0.302665	1.693723	-1.706086	-1.159119
B	-0.134841	0.390528	0.166905	0.184502
C	0.807706	0.072960	0.638787	0.329646
D	-0.497104	-0.754070	-0.943406	0.484752
E	-0.116773	1.901755	0.238127	1.996652

Data Frames

- Selecting, Creating and Dropping Columns
- Selecting, Creating and Dropping Rows
- Selecting subset of rows and columns
- Conditional Selection
- Set New Index & Reset Index

```python
# Selecting Columns
df[['Z','W']]


# Create Columns
df['new_col'] = [20, 50, 80, 90, 70] # can be numpy array or series


# Droping Columns
df.drop('new_col', axis=1, inplace=True)
```

Data Frames

- Selecting, Creating and Dropping Columns
- **Selecting, Creating and Dropping Rows**
- Selecting subset of rows and columns
- Conditional Selection
- Set New Index & Reset Index

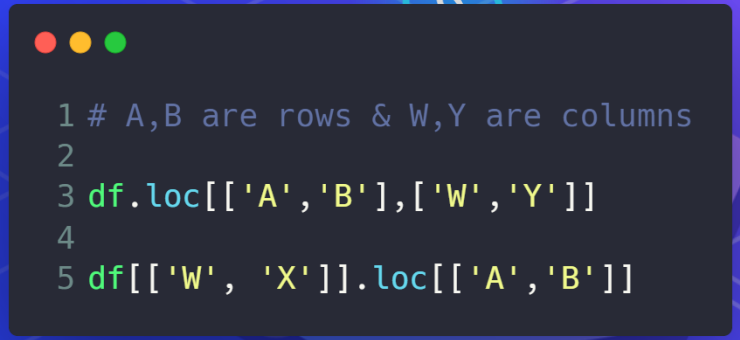
```python
# Selecting Rows
df.loc[['A', 'C']]
df.iloc[[2, 4]]


# Create Rows
df.loc['new_row'] = [10, 20, 50, 80, 90] # also numpy_array or series


# Droping Rows
df.drop('new_col', axis=0, inplace=True)
```

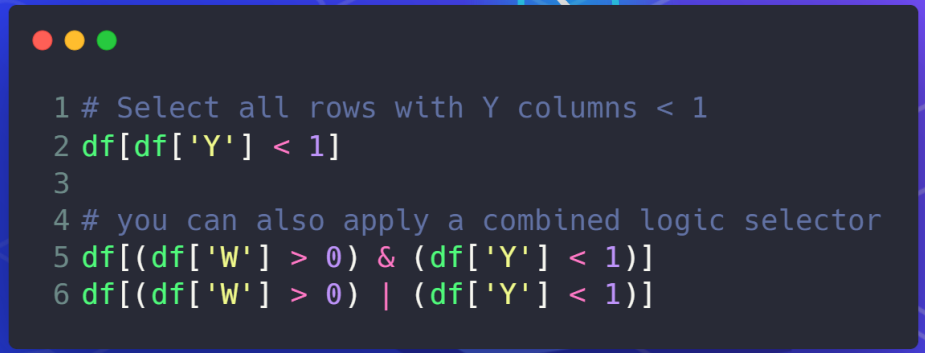
Data Frames

- Selecting, Creating and Dropping Columns
- Selecting, Creating and Dropping Rows
- **Selecting subset of rows and columns**
- Conditional Selection
- Set New Index & Reset Index



Data Frames

- Selecting, Creating and Dropping Columns
- Selecting, Creating and Dropping Rows
- Selecting subset of rows and columns
- **Conditional Selection**
- Set New Index & Reset Index



```
# Select all rows with Y columns < 1
df[df['Y'] < 1]

# you can also apply a combined logic selector
df[(df['W'] > 0) & (df['Y'] < 1)]
df[(df['W'] > 0) | (df['Y'] < 1)]
```

Data Frames

- Selecting, Creating and Dropping Columns
- Selecting, Creating and Dropping Rows
- Selecting subset of rows and columns
- Conditional Selection
- Set New Index & Reset Index

```
df.set_index('States', inplace=True, drop=False)

df.reset_index(inplace=True)
```

Agenda

- What is Numpy
- Install and use Numpy
- Create Numpy Array
- Built in Methods
- Reshaping
- Indexing
- Selection
- Arithmetic and Logic
- Universal Array Functions
- Combine and split
- What is Pandas
- Series
- Data Frames
- **Deal with Missing data**
- Grouping data and aggregate functions
- Merging, Joining and Concatenating
- Pivoting
- Useful Methods
- Apply function
- Data Input & Output

Deal with Missing data

```python
# return dataframe with booleans
df.isna() # also the inverse is df.notna()


# return a series with nan count in each column or row
df.isna().sum(axis=0) # get nan count in each column
df.isna().sum(axis=1) # get nan count in each column


# drop nan values
df.dropna(axis=0, inplace=True) # across rows
df.dropna(axis=1, inplace=True) # across columns


# fill missing data in a columns with the mean value of the column
df['A'].fillna(value=df['A'].mean(), inplace=True)
```

Agenda

- What is Numpy
- Install and use Numpy
- Create Numpy Array
- Built in Methods
- Reshaping
- Indexing
- Selection
- Arithmetic and Logic
- Universal Array Functions
- Combine and split
- What is Pandas
- Series
- Data Frames
- Deal with Missing data
- **Grouping data and aggregate functions**
- Merging, Joining and Concatenating
- Pivoting
- Useful Methods
- Apply function
- Data Input & Output

Grouping data and aggregate functions

```
# 'groupby' used for grouping each value from
# selected column and apply an aggregate function on them
df.groupby('Company').mean() # you can also use min,max,std,count


# show all aggregate functions
df.groupby('Company').describe()
```

Agenda

- What is Numpy
- Install and use Numpy
- Create Numpy Array
- Built in Methods
- Reshaping
- Indexing
- Selection
- Arithmetic and Logic
- Universal Array Functions
- Combine and split
- What is Pandas
- Series
- Data Frames
- Deal with Missing data
- Grouping data and aggregate functions
- **Merging, Joining and Concatenating**
- Pivoting
- Useful Methods
- Apply function
- Data Input & Output

Merging, Joining and Concatenating

Merging

Merging function allows you to merge DataFrames together using a similar logic as merging SQL Tables together

Joining

Joining is a convenient method for combining the columns of two potentially differently-indexed Data Frames into a single result Data Frame

Concatenating

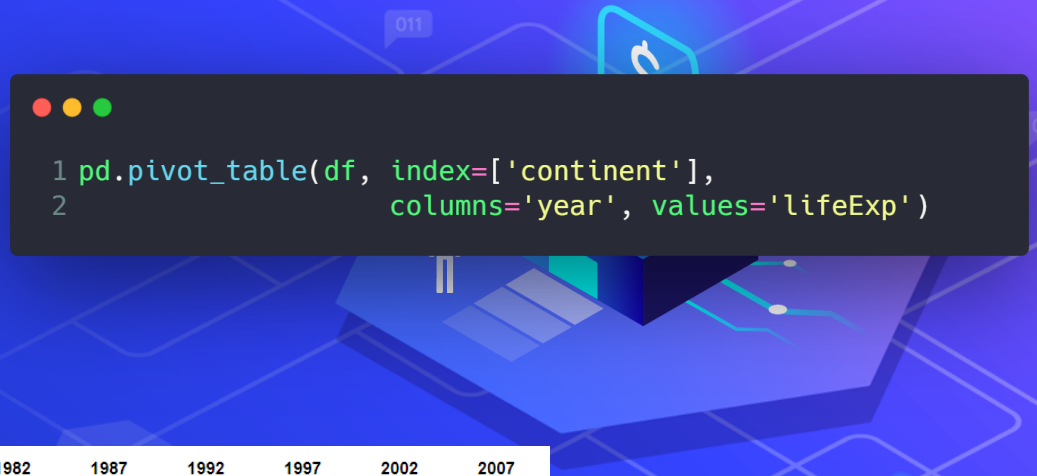
Concatenation basically glues together DataFrames. Keep in mind that dimensions should match along the axis you are concatenating on

Agenda

- What is Numpy
- Install and use Numpy
- Create Numpy Array
- Built in Methods
- Reshaping
- Indexing
- Selection
- Arithmetic and Logic
- Universal Array Functions
- Combine and split
- What is Pandas
- Series
- Data Frames
- Deal with Missing data
- Grouping data and aggregate functions
- Merging, Joining and Concatenating
- **Pivoting**
- Useful Methods
- Apply function
- Data Input & Output

Pivoting

	country	year	pop	continent	lifeExp	gdpPercap
0	Afghanistan	1952	8425333.0	Asia	28.801	779.445314
1	Afghanistan	1957	9240934.0	Asia	30.332	820.853030
2	Afghanistan	1962	10267083.0	Asia	31.997	853.100710
3	Afghanistan	1967	11537966.0	Asia	34.020	836.197138
4	Afghanistan	1972	13079460.0	Asia	36.088	739.981106
...	...	...	...	...	...	...
1699	Zimbabwe	1987	9216418.0	Africa	62.351	706.157306
1700	Zimbabwe	1992	10704340.0	Africa	60.377	693.420786
1701	Zimbabwe	1997	11404948.0	Africa	46.809	792.449960
1702	Zimbabwe	2002	11926563.0	Africa	39.989	672.038623
1703	Zimbabwe	2007	12311143.0	Africa	43.487	469.709298



```
pd.pivot_table(df, index=['continent'],
                   columns='year', values='lifeExp')
```

year	1952	1957	1962	1967	1972	1977	1982	1987	1992	1997	2002	2007
continent												
Africa	39.135500	41.266346	43.319442	45.334538	47.450942	49.580423	51.592865	53.344788	53.629577	53.598269	53.325231	54.806038
Americas	53.279840	55.960280	58.398760	60.410920	62.394920	64.391560	66.228840	68.090720	69.568360	71.150480	72.422040	73.608120
Asia	46.314394	49.318544	51.563223	54.663640	57.319269	59.610556	62.617939	64.851182	66.537212	68.020515	69.233879	70.728485
Europe	64.408500	66.703067	68.539233	69.737600	70.775033	71.937767	72.806400	73.642167	74.440100	75.505167	76.700600	77.648600
Oceania	69.255000	70.295000	71.085000	71.310000	71.910000	72.855000	74.290000	75.320000	76.945000	78.190000	79.740000	80.719500

Agenda

- What is Numpy
- Install and use Numpy
- Create Numpy Array
- Built in Methods
- Reshaping
- Indexing
- Selection
- Arithmetic and Logic
- Universal Array Functions
- Combine and split
- What is Pandas
- Series
- Data Frames
- Deal with Missing data
- Grouping data and aggregate functions
- Merging, Joining and Concatenating
- Pivoting
- **Useful Methods**
- Apply function
- Data Input & Output

Useful Methods

- unique
- value_counts
- sort_values
- mean
- median
- mode
- quantile
- std
- corr
- min
- max
- sum
- count
- head
- tail
- info
- describe
- …

Agenda

- What is Numpy
- Install and use Numpy
- Create Numpy Array
- Built in Methods
- Reshaping
- Indexing
- Selection
- Arithmetic and Logic
- Universal Array Functions
- Combine and split
- What is Pandas
- Series
- Data Frames
- Deal with Missing data
- Grouping data and aggregate functions
- Merging, Joining and Concatenating
- Pivoting
- Useful Methods
- **Apply function**
- Data Input & Output

Apply function

```python
# create circle_area col from applying formula on radius col
df['area'] = df['radius'].apply(lambda r: 3.14*(r**2))

# create Hour col from applying x.hour datetime attribute on Date col
df['Hour'] = df['Date'].apply(lambda x: x.hour)
```

Agenda

- What is Numpy
- Install and use Numpy
- Create Numpy Array
- Built in Methods
- Reshaping
- Indexing
- Selection
- Arithmetic and Logic
- Universal Array Functions
- Combine and split
- What is Pandas
- Series
- Data Frames
- Deal with Missing data
- Grouping data and aggregate functions
- Merging, Joining and Concatenating
- Pivoting
- Useful Methods
- Apply function
- **Data Input & Output**

Data Input & Output

```python
# CSV
df = pd.read_csv('in.csv')
df.to_csv('out.csv')


# Excel
df = pd.read_excel('in.xlsx', sheet_name='Sheet1')
df.to_excel('out.xlsx',sheet_name='Sheet1', index=False)


# Sqlite
conn = sqlite3.connect('test.db')
df = pd.read_sql('SELECT * FROM data', con=conn)
data.to_sql('dummy_table', con=conn)
```

Questions ?!

Thanks!

>_ Live long and prosper