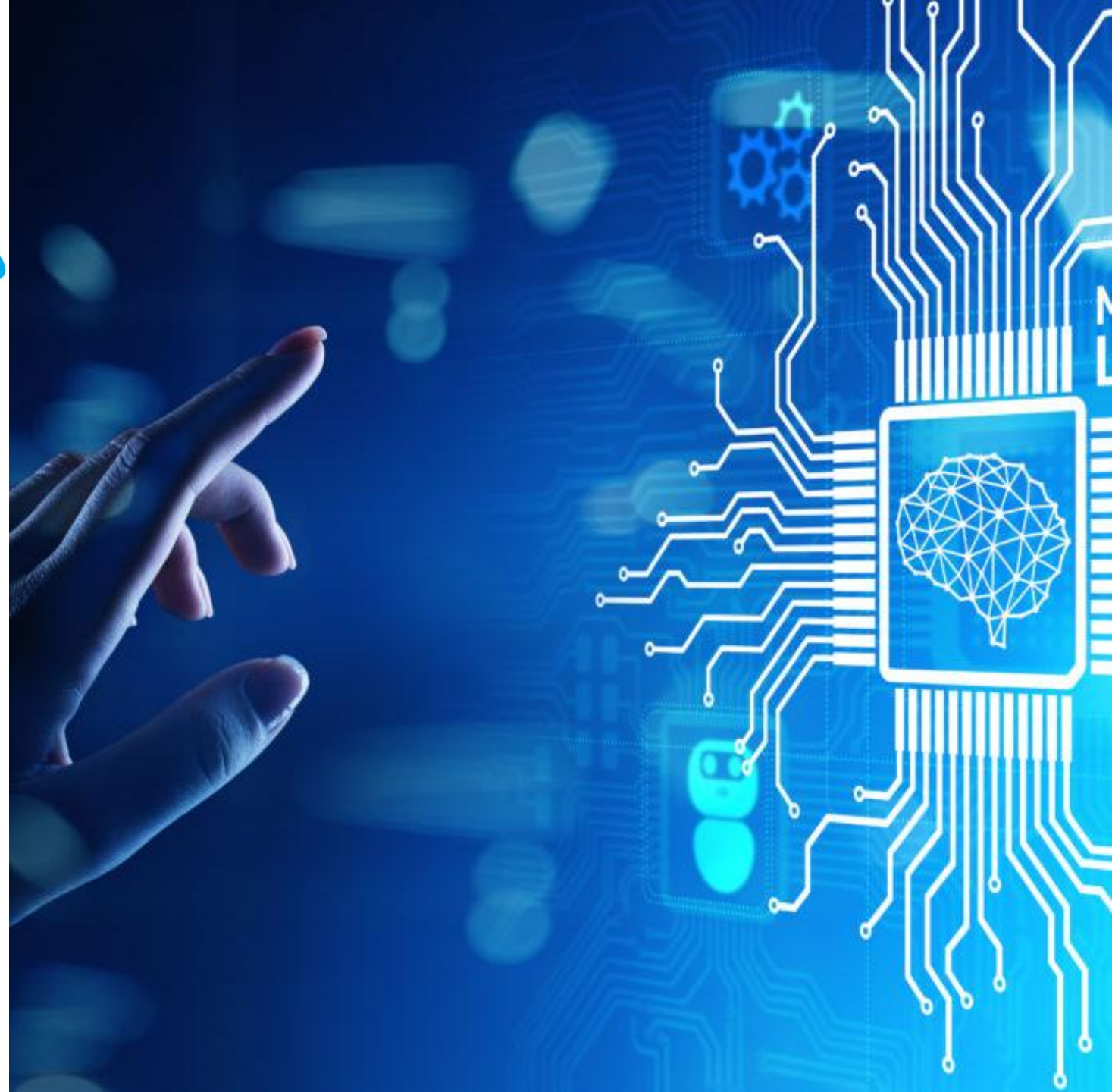
DIGITAL LEARNING HUB

Kozalys
Measure, Decide, Perform

Introduction to Machine Learning

Zeyneb GASMI
Data Science & Business Intelligence Consultant



Agenda: Day 1

9 AM – 12 PM:

1. Introduction to Machine Learning
2. Machine Learning types
3. Machine Learning applications
4. Supervised Machine Learning

1 PM – 4 30 PM:

4. Python for Machine Learning
5. Linear Regression
6. Logistic Regression

Agenda: Day 1

9 AM – 12 PM:

1. Introduction to Machine Learning
2. Machine Learning types
3. Machine Learning applications
4. Supervised Machine Learning

1 PM – 4 30 PM:

1. **Python for Machine Learning**
2. Linear Regression
3. Logistic Regression


Why python?

Python for Machine Learning : Why ?

ML languages Ranking

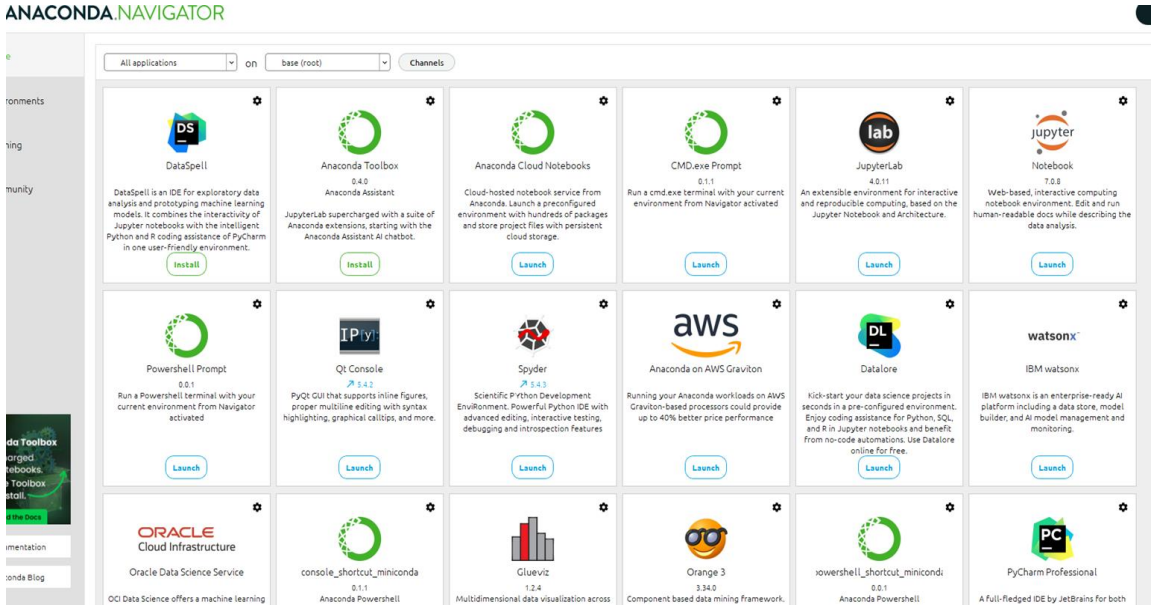
Worldwide, Mar 2023 compared to a year ago:

Rank	Change	Language	Share
1		Python	27.91
2		Java	16.58
3		JavaScript	9.67
4		C/C++	6.93
5		C#	6.88
6		PHP	5.19
7		R	4.23
8	↑	TypeScript	2.81
9	↑	Swift	2.28
10	↓↓	Objective-C	2.26

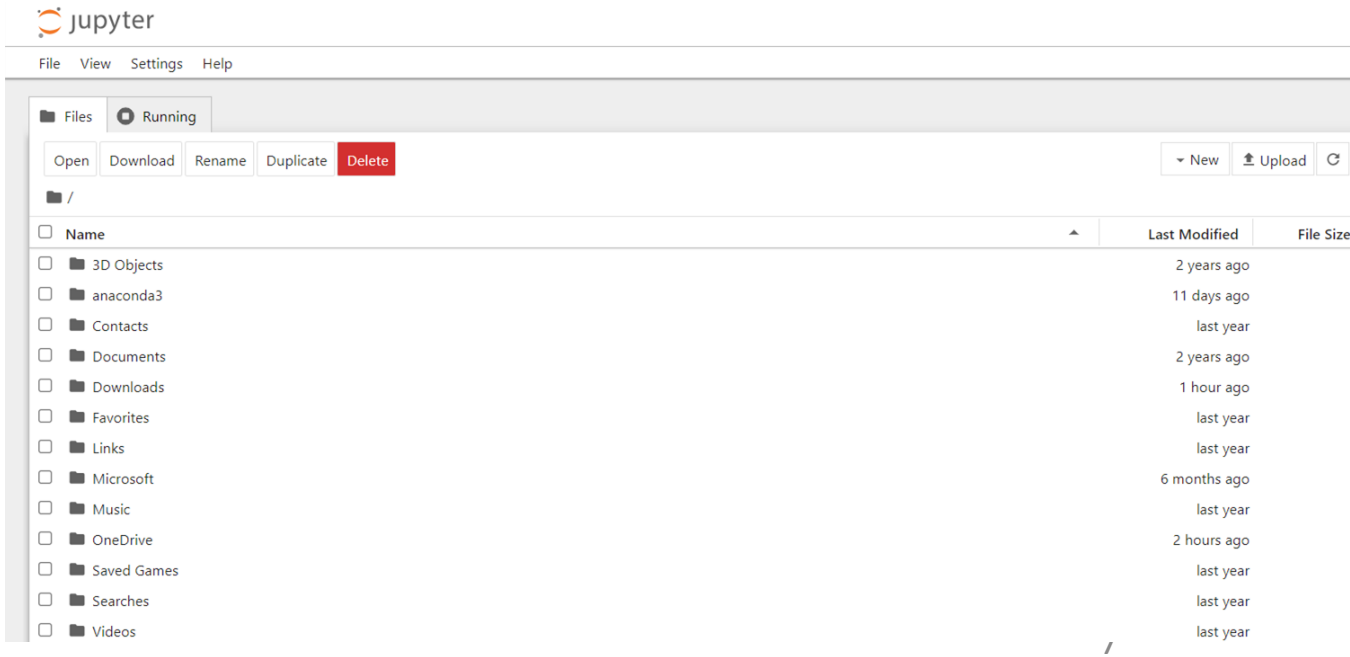
Let’s get started

Python for Machine Learning : Get Started

1. Open ANACONDA Navigator

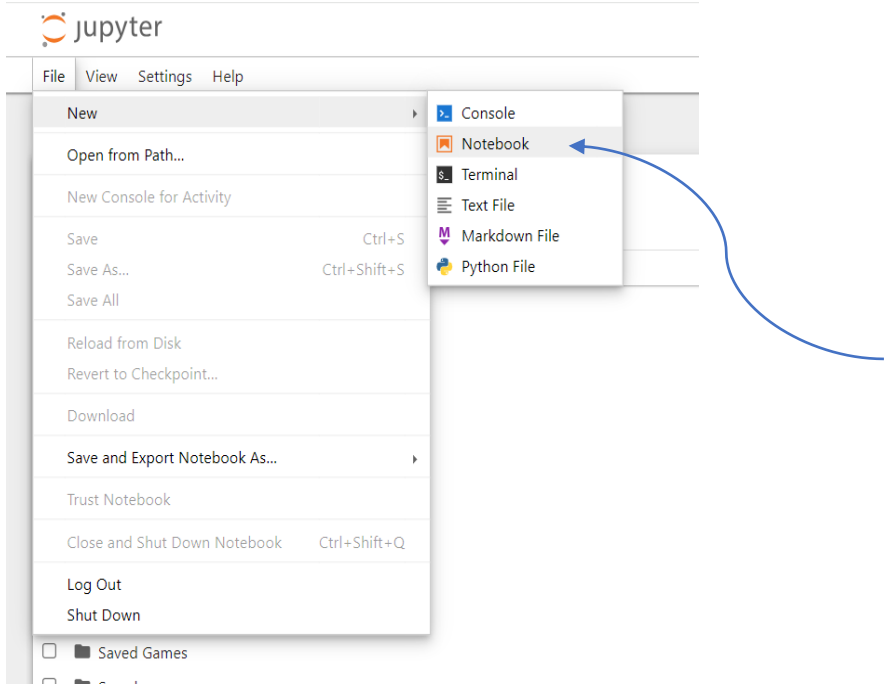


2. Launch Notebook

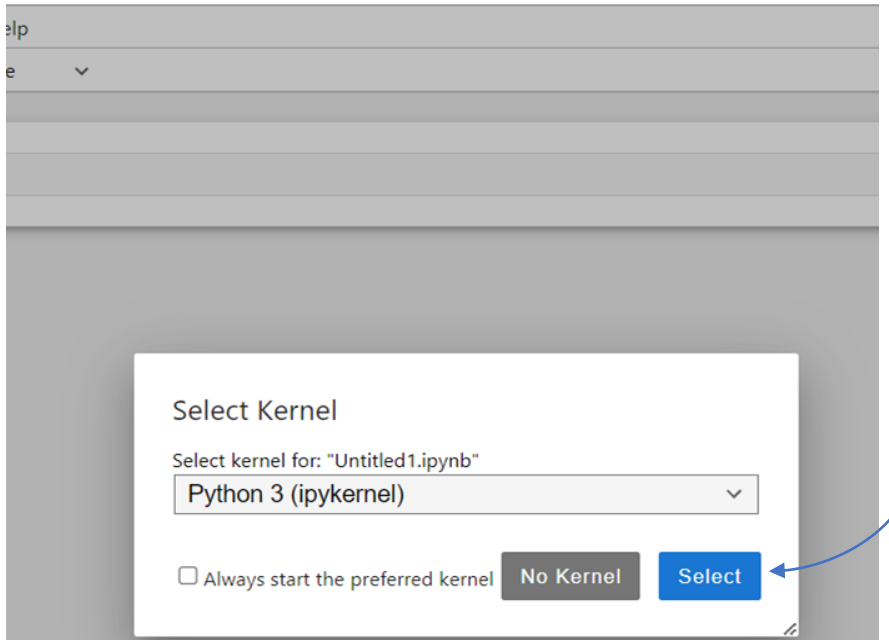


Python for Machine Learning : Get Started

3. Create New File de type ‘Notebook’

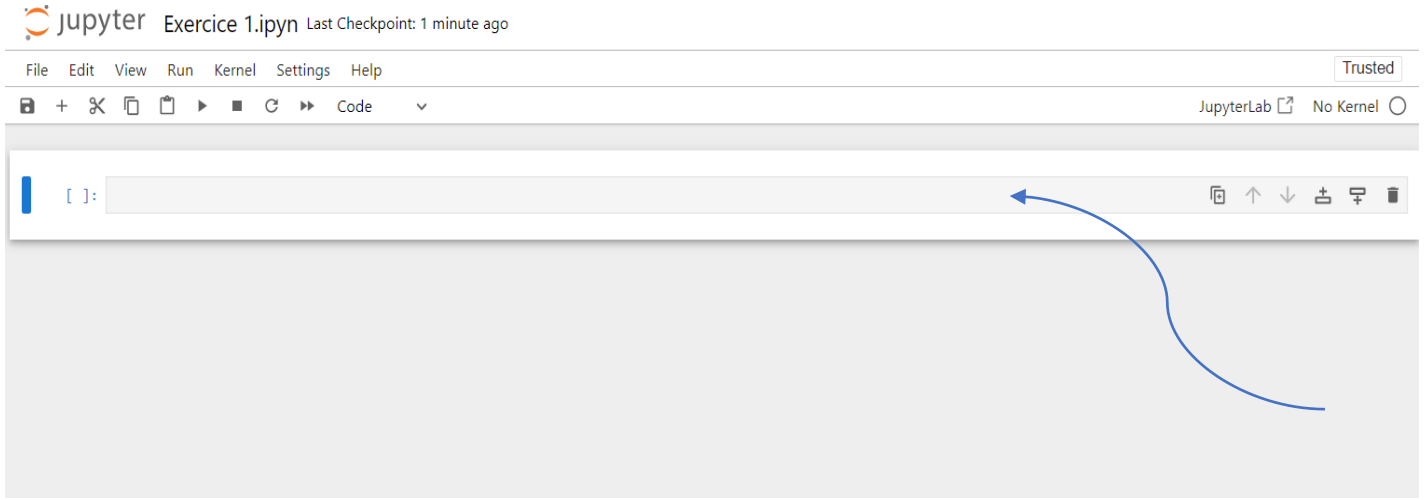


4. Choose ‘Select’



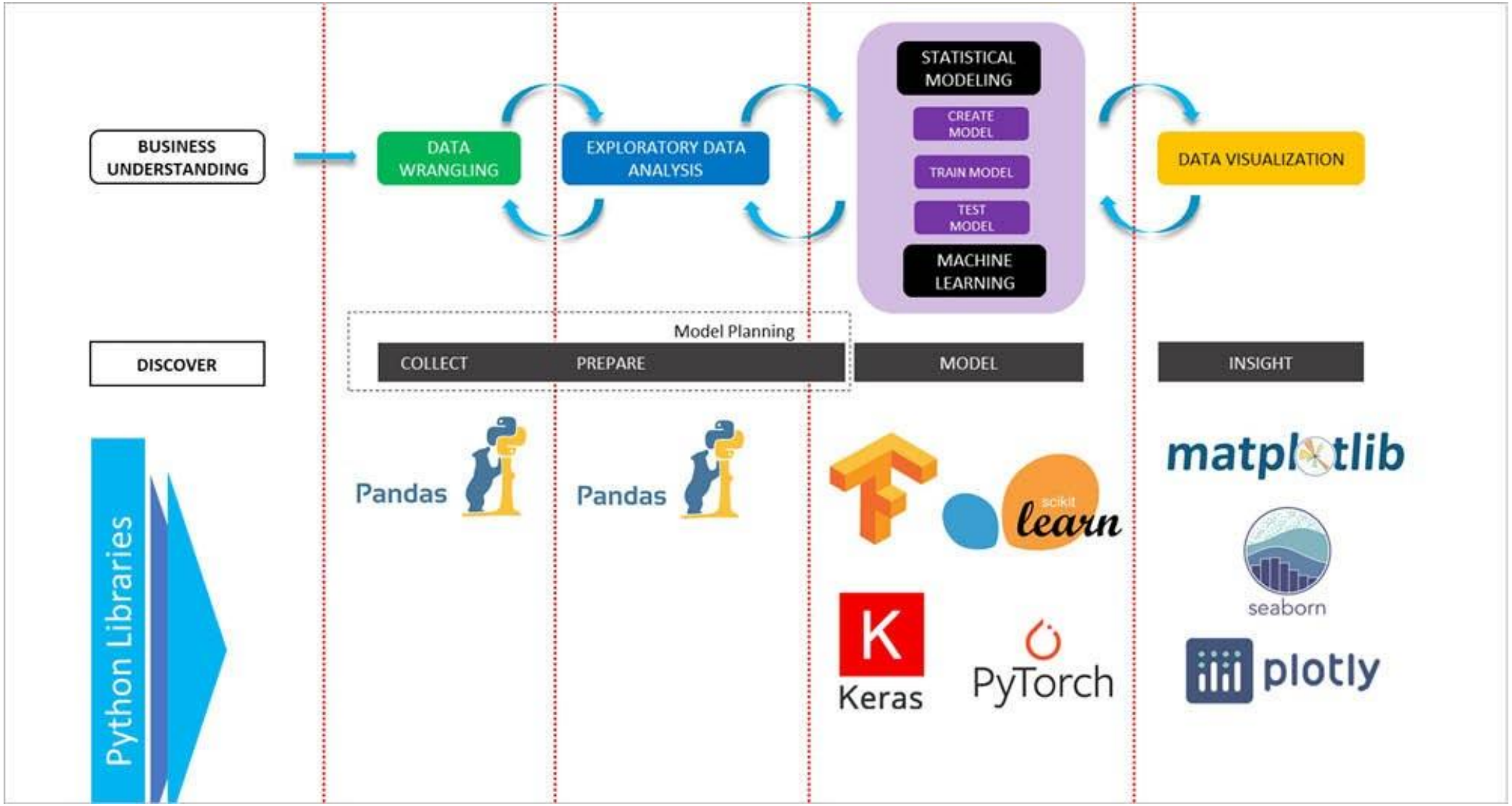
Python for Machine Learning : Jupyter Notebook

5. We Can Start Coding Now in the Notebook



6. Try your First Code

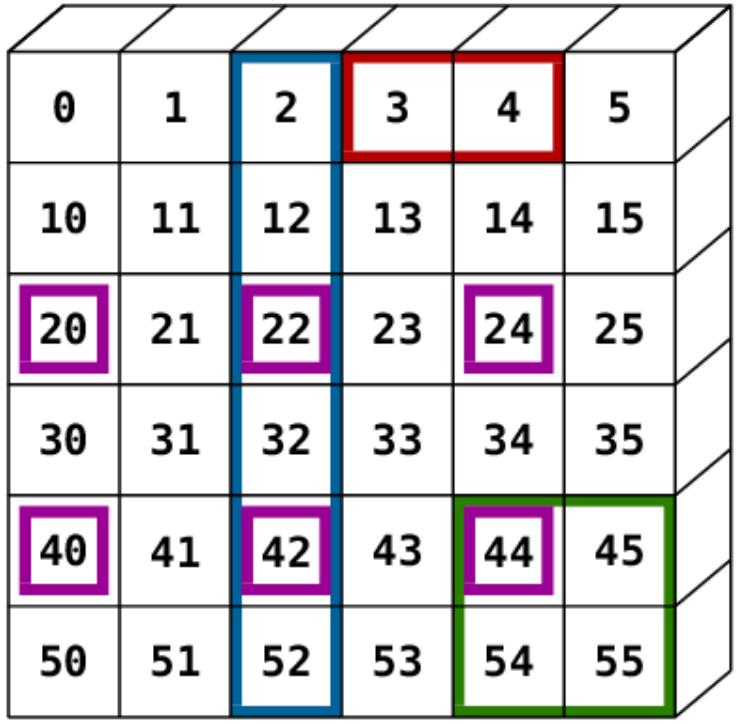
Python for Machine Learning : Packages



Python for Machine Learning : NumPy

- **Numpy** is the fundamental package for scientific computing in python.

```
>>> a[0, 3:5]
array([3, 4])

>>> a[4:, 4:]
array([[44, 55],
       [54, 55]])

>>> a[:, 2]
a([2, 12, 22, 32, 42, 52])

>>> a[2::2, ::2]
array([[20, 22, 24],
       [40, 42, 44]])
```



Python for Machine Learning : NumPy - Examples

```
import numpy as np
```

```
A=np.array([[1,2,3],[4,5,6]])
```

```
A.shape
```

```
(2, 3)
```

```
A.ndim
```

```
2
```

```
A.size
```

```
6
```

```
B=np.array([
[
[12,11,10],
[9,8,7],
],
[
[6,5,4],
[3,2,1]
    ]
])
```

```
B
```

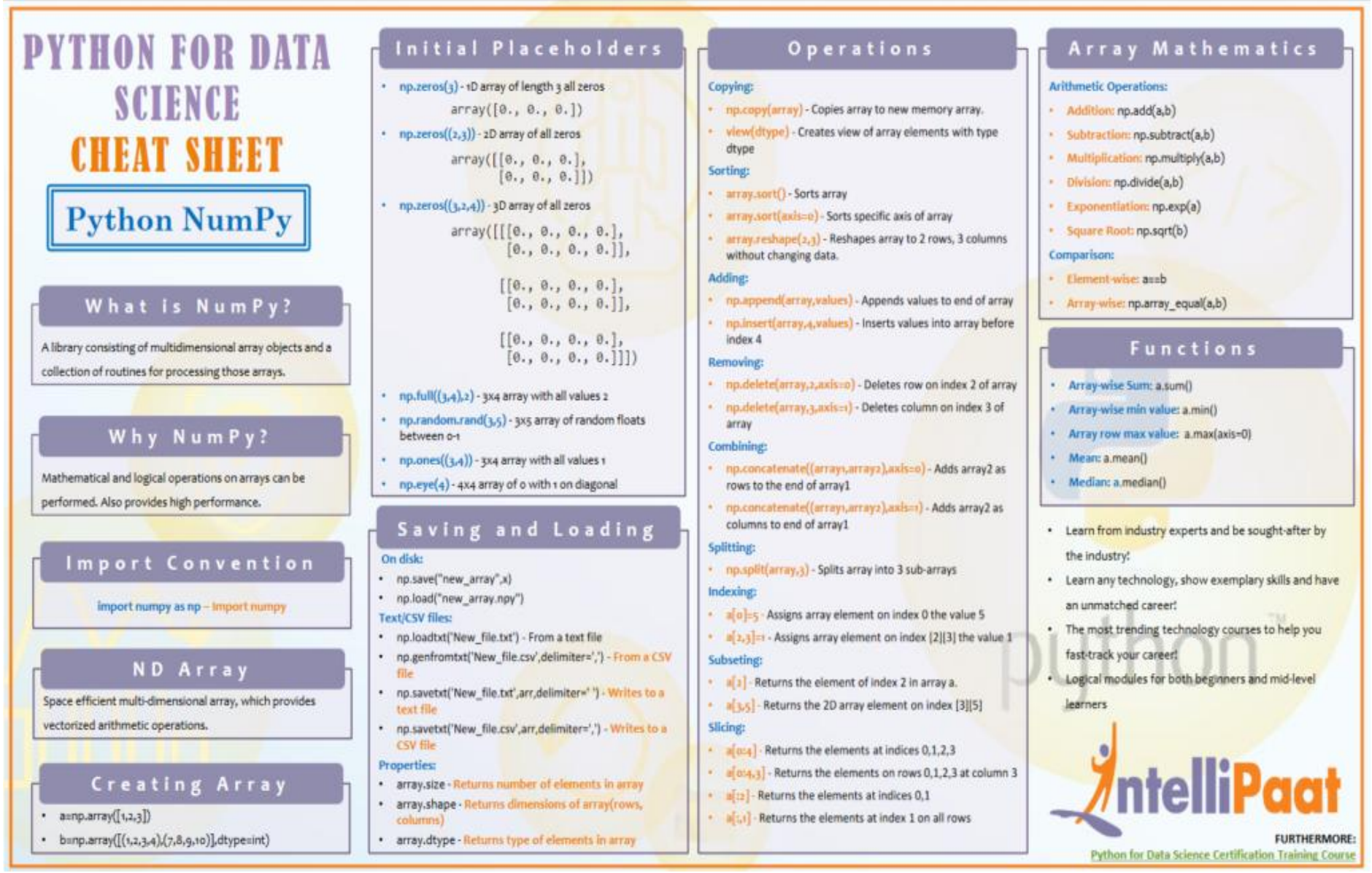
Python for Machine Learning : NumPy - Examples

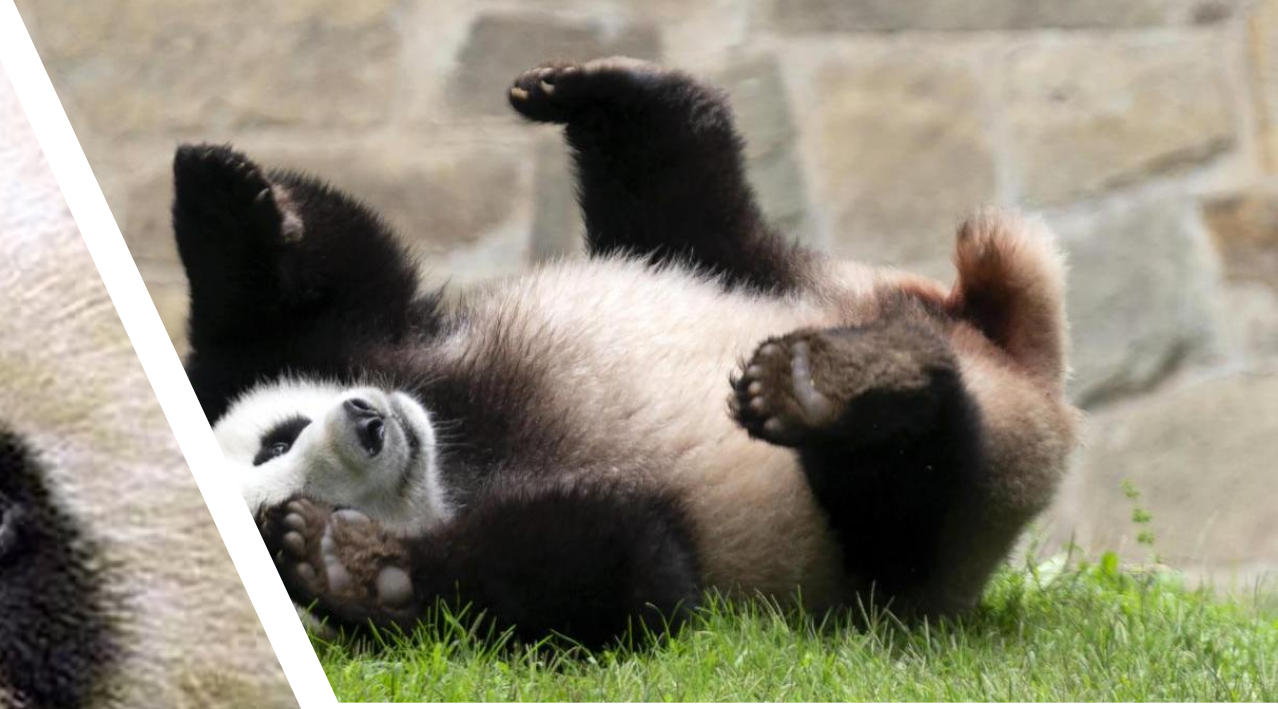
```
[7]: B

[7]: array([[[12, 11, 10],
             [ 9,  8,  7]],

            [[ 6,  5,  4],
             [ 3,  2,  1]]])

[0]: B.shape

[0]: (2, 2, 3)

[1]: A[0]

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

[2]: A[0,1]

[2]: 2

[4]: A[0],A[1]

[4]: (array([1, 2, 3]), array([4, 5, 6]))

[5]: A[0:]

[5]: array([[1, 2, 3],
            [4, 5, 6]])
```

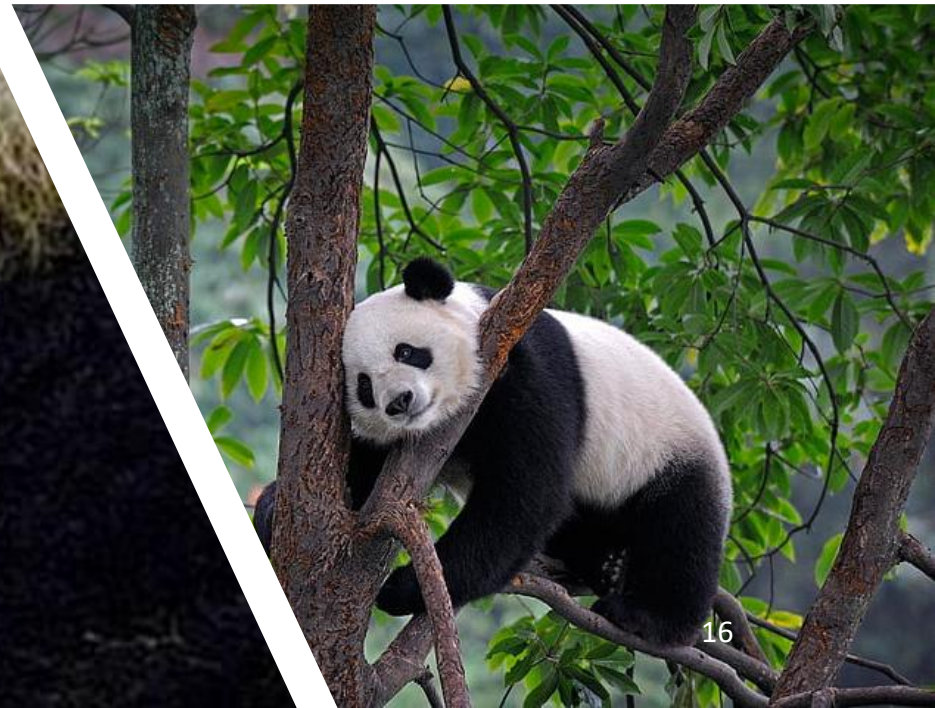
Python for Machine Learning : NumPy - Examples

```
[21]: A[1:-1]

[21]: array([], shape=(0, 3), dtype=int32)

[22]: A[::1]

[22]: array([[1, 2, 3],
             [4, 5, 6]])

[23]: A[::2]

[23]: array([[1, 2, 3]])
```

Python for Machine Learning : NumPy - Functions







Python for Machine Learning : Pandas

- **Pandas** is a fast, powerful, flexible and easy to use open source **Data Analysis and Data Manipulation tool,** built on top of the Python programming language**.**
- **Data Frame :** A data structure constructed with rows and columns, similar to a database or **Excel spreadsheet.**

Pandas

Python for Machine Learning : Pandas - Functions

Python For Data Science *Cheat Sheet*

Pandas Basics

Learn Python for Data Science Interactively at www.DataCamp.com

Pandas

The Pandas library is built on NumPy and provides easy-to-use data structures and data analysis tools for the Python programming language.

Use the following import convention:

```
>>> import pandas as pd
```

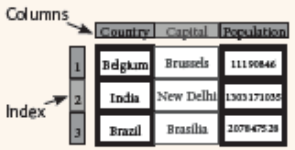
Pandas Data Structures

Series

A one-dimensional labeled array capable of holding any data type


```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

DataFrame



A two-dimensional labeled data structure with columns of potentially different types

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
            'Capital': ['Brussels', 'New Delhi', 'Brasilia'],
            'Population': [11190846, 1303171035, 207847528]}

>>> df = pd.DataFrame(data,
                      columns=['Country', 'Capital', 'Population'])
```

Asking For Help

```
>>> help(pd.Series.loc)
```

Selection

Also see NumPy Arrays

Getting

```
>>> s['b']
 7

>>> df[1:]
   Country     Capital  Population
1   India   New Delhi  1303171035
2  Brazil    Brasilia   207847528
```

Get one element

Get subset of a DataFrame

Selecting, Boolean Indexing & Setting

By Position

```
>>> df.iloc[[0],[0]]
'Belgium'
>>> df.iat[[0],[0]]
'Belgium'
```

Select single value by row & column

By Label

```
>>> df.loc[[0], ['Country']]
'Belgium'
>>> df.at[[0], ['Country']]
'Belgium'
```

Select single value by row & column labels

By Label/Position

```
>>> df.ix[2]
Country      Brazil
Capital    Brasilia
Population  207847528
```

Select single row of subset of rows

```
>>> df.ix[:,'Capital']
0     Brussels
1    New Delhi
2     Brasilia
```

Select a single column of subset of columns

```
>>> df.ix[1,'Capital']
'New Delhi'
```

Select rows and columns

Boolean Indexing

```
>>> s[~(s > 1)]
>>> s[(s < -1) | (s > 2)]
>>> df[df['Population']>1200000000]
```

Series s where value is not >1
s where value is <-1 or >2
Use filter to adjust DataFrame

Setting

```
>>> s['a'] = 6
```

Set index a of Series s to 6

Dropping

```
>>> s.drop(['a', 'c'])
>>> df.drop('Country', axis=1)
```

Drop values from rows (axis=0)
Drop values from columns(axis=1)

Sort & Rank

```
>>> df.sort_index(by='Country')
>>> s.order()
>>> df.rank()
```

Sort by row or column index
Sort a series by its values
Assign ranks to entries

Retrieving Series/DataFrame Information

Basic Information

```
>>> df.shape
>>> df.index
>>> df.columns
>>> df.info()
>>> df.count()
```

(rows,columns)
Describe index
Describe DataFrame columns
Info on DataFrame
Number of non-NA values

Summary

```
>>> df.sum()
>>> df.cumsum()
>>> df.min()/df.max()
>>> df.idmin()/df.idmax()
>>> df.describe()
>>> df.mean()
>>> df.median()
```

Sum of values
Cummulative sum of values
Minimum/maximum values
Minimum/Maximum index value
Summary statistics
Mean of values
Median of values

Applying Functions

```
>>> f = lambda x: x*2
>>> df.apply(f)
>>> df.applymap(f)
```

Apply function
Apply function element-wise

Data Alignment

Internal Data Alignment

NA values are introduced in the indices that don't overlap:

```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
  a    10.0
  b     NaN
  c     5.0
  d     7.0
```

Arithmetic Operations with Fill Methods

You can also do the internal data alignment yourself with the help of the fill methods:

```
>>> s.add(s3, fill_value=0)
  a    10.0
  b    -5.0
  c     5.0
  d     7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
>>> s.mul(s3, fill_value=3)
```

I/O

Read and Write to CSV

```
>>> pd.read_csv('file.csv', header=None, nrows=5)
>>> pd.to_csv('myDataFrame.csv')
```

Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> pd.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')
```

Read multiple sheets from the same file

```
>>> xlsx = pd.ExcelFile('file.xls')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

Read and Write to SQL Query or Database Table

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///:memory:')
>>> pd.read_sql("SELECT * FROM my_table;", engine)
>>> pd.read_sql_table('my_table', engine)
>>> pd.read_sql_query("SELECT * FROM my_table;", engine)
```

read_sql() is a convenience wrapper around read_sql_table() and read_sql_query()

```
>>> pd.to_sql('myDf', engine)
```

DataCamp
Learn Python for Data Science Interactively

Python for Machine Learning : Scikit learn

- ✓ **Scikit-learn**, also known as sklearn, is an open-source, machine learning and data modeling **library for Python**.
- ✓ It features various classification, regression and clustering algorithms, and is designed to interoperate with Python libraries, NumPy and SciPy.
- ✓ Scikit-learn was first released in 2010, and it has since gained a prominent place in the Python machine learning ecosystem.

Python for Machine Learning : Scikit learn

- Integrates well with many other Python libraries, such as matplotlib and plotly for plotting, NumPy for array vectorization, Pandas dataframes, SciPy, and many more.
- We can pass NumPy arrays and Pandas dataframes directly to Scikit-learn’s algorithms.
- It is largely written in Python, and uses NumPy extensively for high-performance linear algebra and array operations.

Python for Machine Learning : Scikit learn

Scikit learn provides a comprehensive set of supervised and unsupervised learning algorithms, covering areas such as:

- Classification
- Regression
- Clustering
- Dimensionality Reduction
- Model selection
- Pre-processing

Lab 0 : Introduction to Python

1. Create a One column Series

```
]: s = pd.Series([1, 3, 5, np.nan, 6, 8])
   s
```

```
]: 0    1.0
   1    3.0
   2    5.0
   3    NaN
   4    6.0
   5    8.0
   dtype: float64
```

Lab 0 : Introduction to python

2. Create an Index Column with Random Dates
3. Create a Dataframe with Random values

```
dates = pd.date_range("20130101", periods=6)
dates
```

```
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
               '2013-01-05', '2013-01-06'],
              dtype='datetime64[ns]', freq='D')
```

```
df = pd.DataFrame(np.random.randn(6, 5), index=dates, columns=list("ABCDE"))
df
```

	A	B	C	D	E
2013-01-01	-1.472056	1.824725	0.134344	0.816483	-0.153126
2013-01-02	-0.848848	-0.167449	-0.536584	-0.928158	0.242486
2013-01-03	0.414928	-1.242451	-0.163722	-0.271138	-0.439420
2013-01-04	-1.084634	-0.083594	-0.316206	0.424999	-1.466551
2013-01-05	-1.385467	-1.393815	0.321079	0.617400	-0.286634
2013-01-06	-0.732275	1.162741	-1.811739	0.002125	-0.741887

Lab 0 : Introduction to python

4. Print DataType of each column in the Dataframe df
5. Create a Dataframe with different DataTypes

[27]:
```
df.dtypes
```

[27]:
```
A    float64
B    float64
C    float64
D    float64
E    float64
dtype: object
```

[28]:
```
df2 = pd.DataFrame(
    {
        "A": 1.0,
        "B": pd.Timestamp("20130102"),
        "C": pd.Series(1, index=list(range(4)), dtype="float32"),
        "D": np.array([3] * 4, dtype="int32"),
        "E": pd.Categorical(["test", "train", "test", "train"]),
        "F": "foo",
    }
)
```

[29]:
```
df2.dtypes
```

[29]:
```
A          float64
B    datetime64[s]
C          float32
D            int32
E         category
F           object
dtype: object
```

Lab 0 : Introduction to python

6. Use ‘head’ and ‘tail’ functions to see the top or the bottum of the dataframe
7. Use ‘index’ to get the index of the dataframe
8. Get Columns Name with ‘columns’ function

```
df.head()
```

	A	B	C	D	E
2013-01-01	-1.472056	1.824725	0.134344	0.816483	-0.153126
2013-01-02	-0.848848	-0.167449	-0.536584	-0.928158	0.242486
2013-01-03	0.414928	-1.242451	-0.163722	-0.271138	-0.439420
2013-01-04	-1.084634	-0.083594	-0.316206	0.424999	-1.466551
2013-01-05	-1.385467	-1.393815	0.321079	0.617400	-0.286634

```
df.tail(3)
```

	A	B	C	D	E
2013-01-04	-1.084634	-0.083594	-0.316206	0.424999	-1.466551
2013-01-05	-1.385467	-1.393815	0.321079	0.617400	-0.286634
2013-01-06	-0.732275	1.162741	-1.811739	0.002125	-0.741887

```
[32]: df.index
```

```
[32]: DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
                     '2013-01-05', '2013-01-06'],
                    dtype='datetime64[ns]', freq='D')
```

```
[34]: df.columns
```

```
[34]: Index(['A', 'B', 'C', 'D', 'E'], dtype='object')
```

Lab 0 : Introduction to python

9. Get a basic Analysis view of Numerical Columns with ‘describe’
10. Perform the Transpose of a Dataframe with ‘T’ function

[35]:
```
df.describe()
```

[35]:

	A	B	C	D	E
count	6.000000	6.000000	6.000000	6.000000	6.000000
mean	-0.851392	0.016693	-0.395471	0.110285	-0.474189
std	0.684489	1.280730	0.758929	0.646602	0.584728
min	-1.472056	-1.393815	-1.811739	-0.928158	-1.466551
25%	-1.310259	-0.973701	-0.481489	-0.202823	-0.666270
50%	-0.966741	-0.125522	-0.239964	0.213562	-0.363027
75%	-0.761418	0.851157	0.059827	0.569299	-0.186503
max	0.414928	1.824725	0.321079	0.816483	0.242486

[36]:
```
df.T
```

[36]:

	2013-01-01	2013-01-02	2013-01-03	2013-01-04	2013-01-05	2013-01-06
A	-1.472056	-0.848848	0.414928	-1.084634	-1.385467	-0.732275
B	1.824725	-0.167449	-1.242451	-0.083594	-1.393815	1.162741
C	0.134344	-0.536584	-0.163722	-0.316206	0.321079	-1.811739
D	0.816483	-0.928158	-0.271138	0.424999	0.617400	0.002125
E	-0.153126	0.242486	-0.439420	-1.466551	-0.286634	-0.741887

Lab 0 : Introduction to python

11. Sort Dataframe by index
12. Sort Dataframe by columns

[37]:
```
df.sort_index(axis=1, ascending=False)
```

[37]:

	E	D	C	B	A
2013-01-01	-0.153126	0.816483	0.134344	1.824725	-1.472056
2013-01-02	0.242486	-0.928158	-0.536584	-0.167449	-0.848848
2013-01-03	-0.439420	-0.271138	-0.163722	-1.242451	0.414928
2013-01-04	-1.466551	0.424999	-0.316206	-0.083594	-1.084634
2013-01-05	-0.286634	0.617400	0.321079	-1.393815	-1.385467
2013-01-06	-0.741887	0.002125	-1.811739	1.162741	-0.732275

[38]:
```
df.sort_values(by="B")
```

[38]:

	A	B	C	D	E
2013-01-05	-1.385467	-1.393815	0.321079	0.617400	-0.286634
2013-01-03	0.414928	-1.242451	-0.163722	-0.271138	-0.439420
2013-01-02	-0.848848	-0.167449	-0.536584	-0.928158	0.242486
2013-01-04	-1.084634	-0.083594	-0.316206	0.424999	-1.466551
2013-01-06	-0.732275	1.162741	-1.811739	0.002125	-0.741887
2013-01-01	-1.472056	1.824725	0.134344	0.816483	-0.153126

Agenda: Day 1

9 AM – 12 PM:

1. Introduction to Machine Learning
2. Machine Learning types
3. Machine Learning applications
4. Supervised Machine Learning

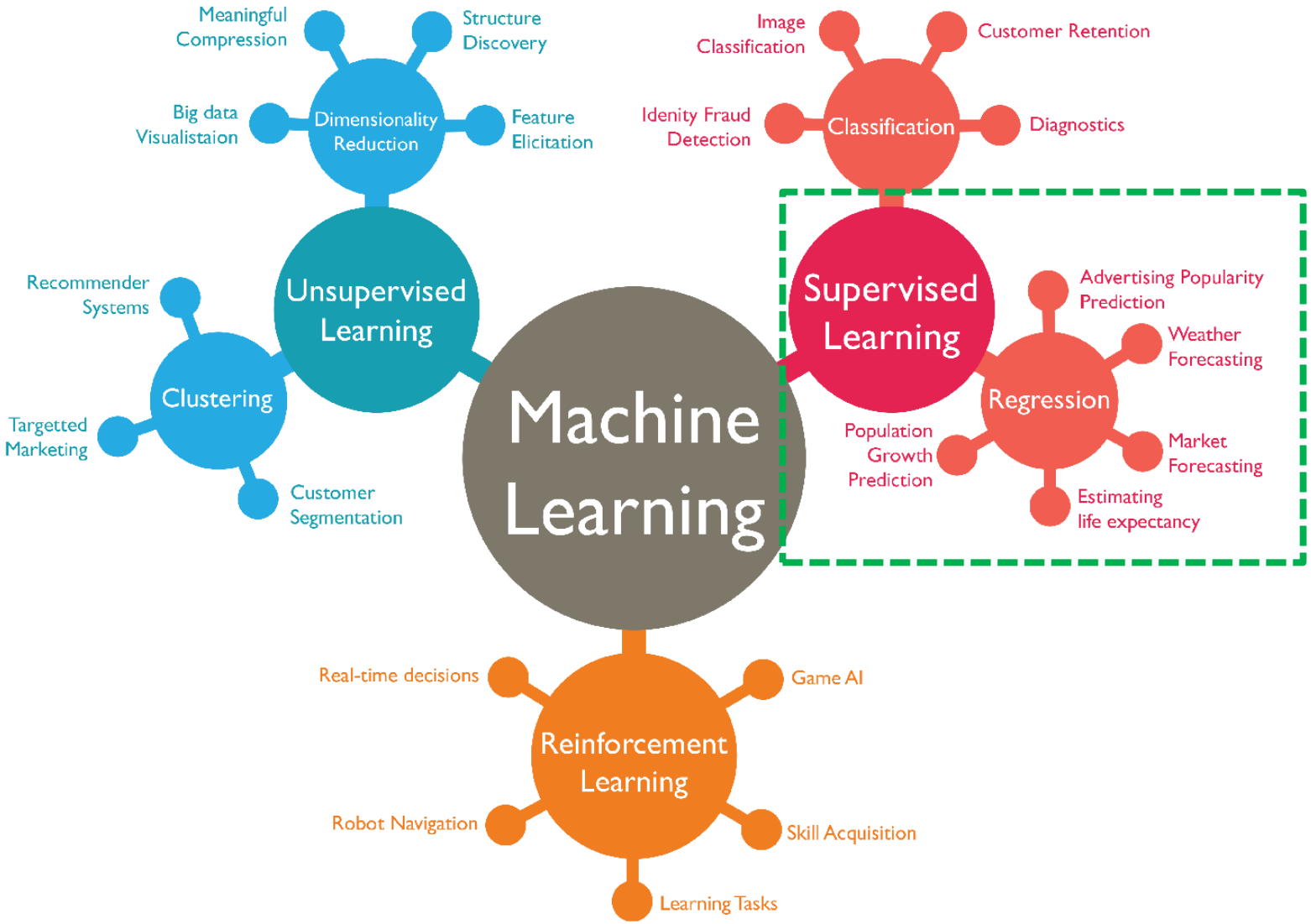
1 PM – 4 30 PM:

4. Python for Machine Learning
5. **Linear Regression**
6. **Logistic Regression**

Reminder - What's an algorithm ?

An algorithm is the description of a series of steps allowing a result to be obtained from elements provided as input. For example, a cooking recipe is an algorithm for obtaining a dish from its ingredients!

Linear regression : A Regression Model



Linear Regression : Dependant and independent variable

Features X

Independant Variables

Target Y

Dependant variable

Features					Label
Position	Experience	Skill	Country	City	Salary ($)
Developer	0	1	USA	New York	103100
Developer	1	1	USA	New York	104900
Developer	2	1	USA	New York	106800
Developer	3	1	USA	New York	108700
Developer	4	1	USA	New York	110400
Developer	5	1	USA	New York	112300
Developer	6	1	USA	New York	114200
Developer	7	1	USA	New York	116100
Developer	8	1	USA	New York	117800
Developer	9	1	USA	New York	119700
Developer	10	1	USA	New York	121600

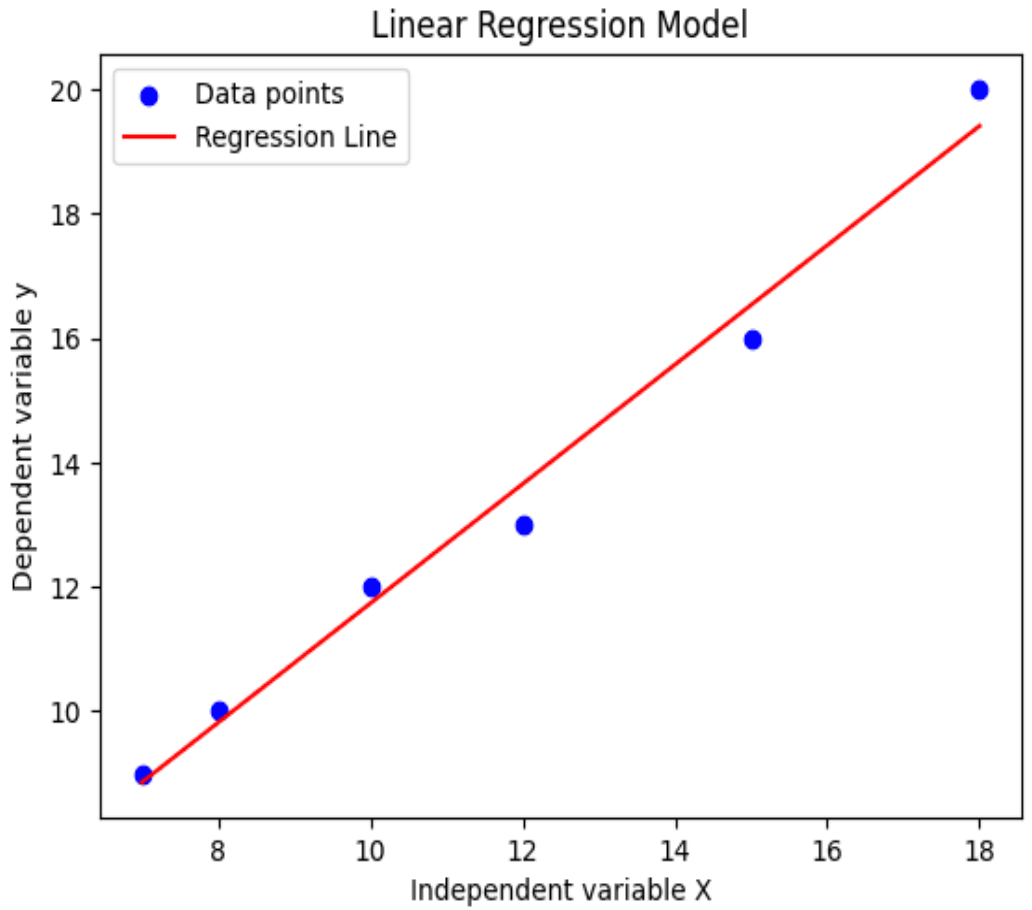
Simple linear regression

In a simple regression problem (a single x and a single y), the form of the model would be :

$$y = B0 + B1*x$$

Where:

- y is the dependent variable (target)
- x is the independent variable (feature)
- B1 is the slope of the line (also called the weight or coefficient)
- B0 is the y-intercept.



Multiple linear regression

In a multiple regression problem (x vector and y vector), the form of the model becomes :

$$\hat{Y} = \theta_0 * 1 + \theta_1 X_1 + \cdots + \theta_p X_p$$

Matrix representation

$$\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}, \begin{bmatrix} 1 & a_{1,1} & \cdots & a_{1,p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & a_{n,1} & \cdots & a_{n,p} \end{bmatrix}, \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_p \end{bmatrix}$$

O Vector

Matrix X

Agenda: Day 1

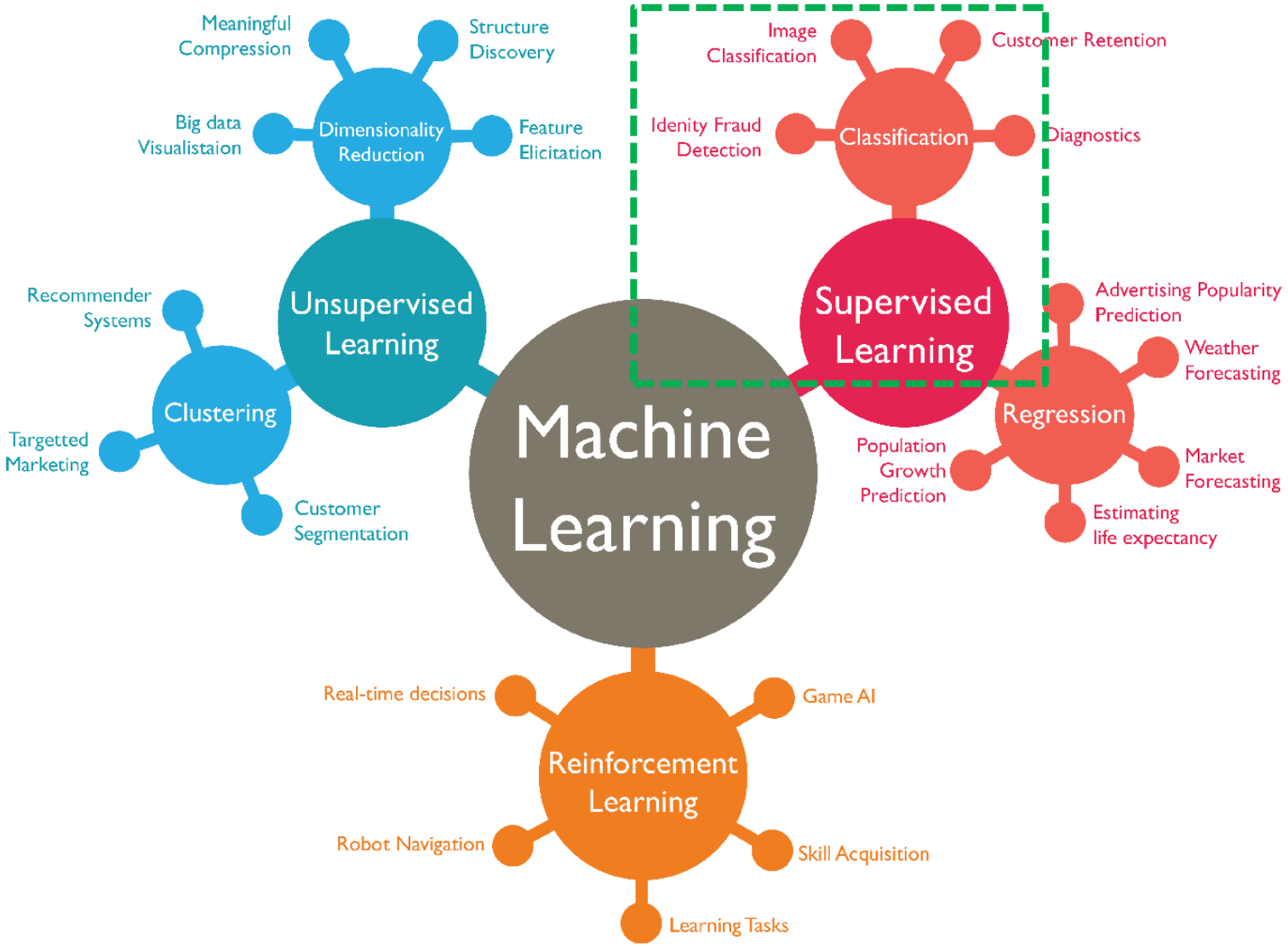
9 AM – 12 PM:

1. Introduction to Machine Learning
2. Machine Learning types
3. Machine Learning applications
4. Supervised & Unsupervised Learning

1 PM – 4 30 PM:

4. Python for Machine Learning – Lab 0
5. **Linear Regression**
6. **Logistic Regression**

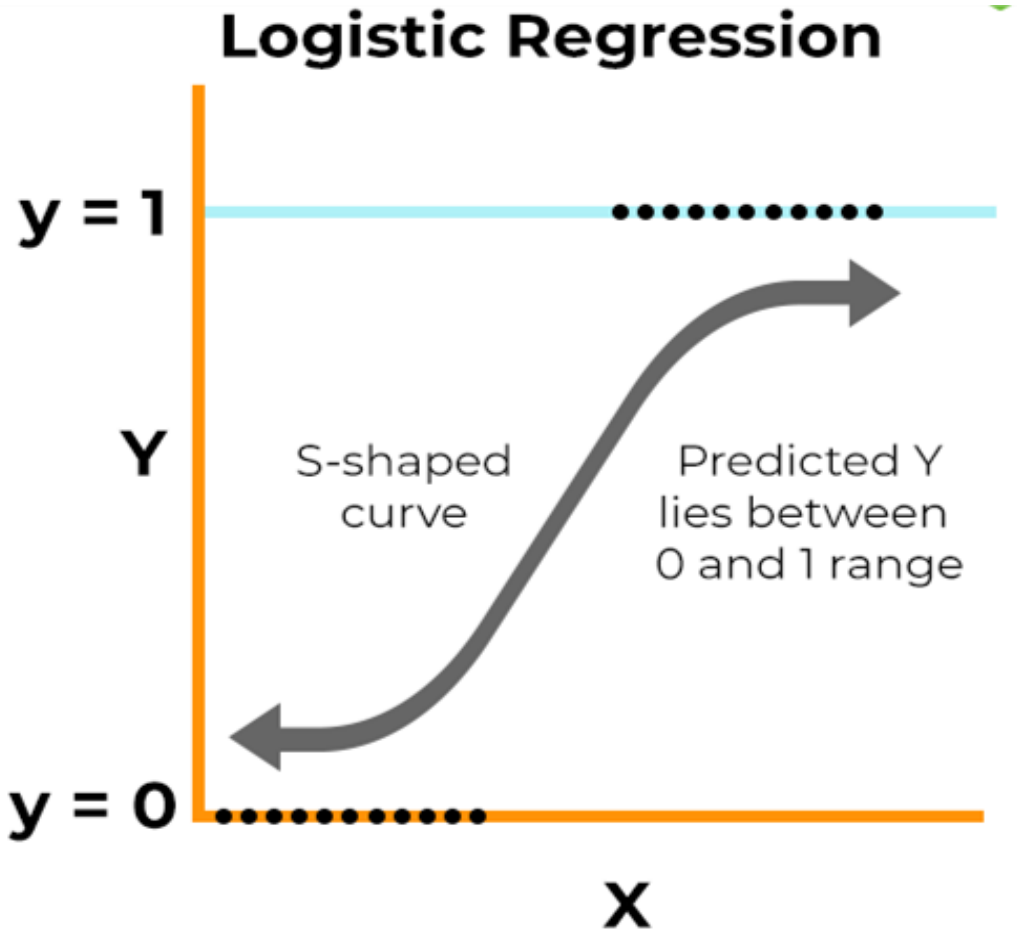
Logistic regression : A Classification Model



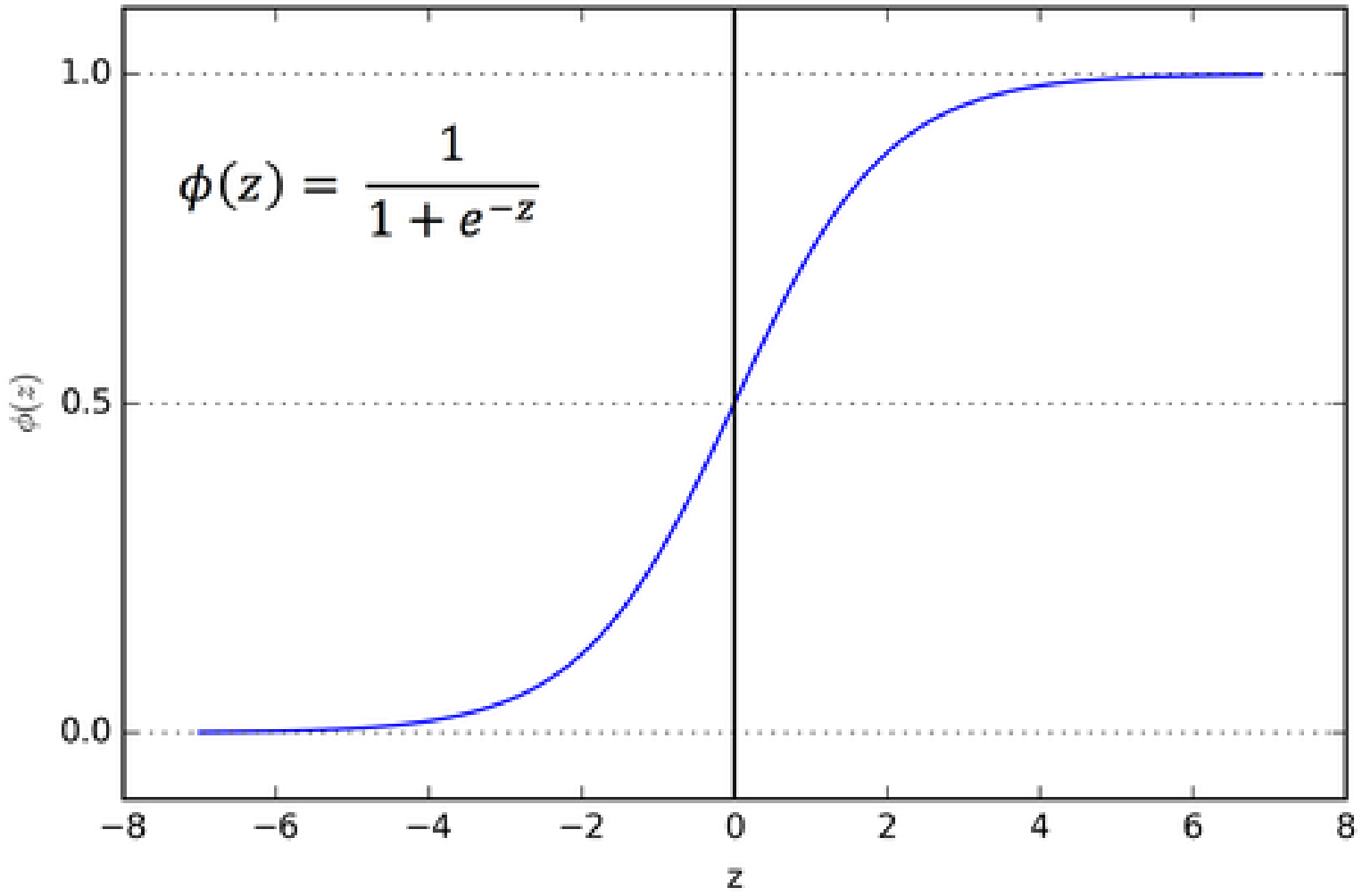
Logistic regression : Definition

Logistic regression is a powerful supervised ML algorithm used for binary classification problems (when target is categorical)

$$Y = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}}$$

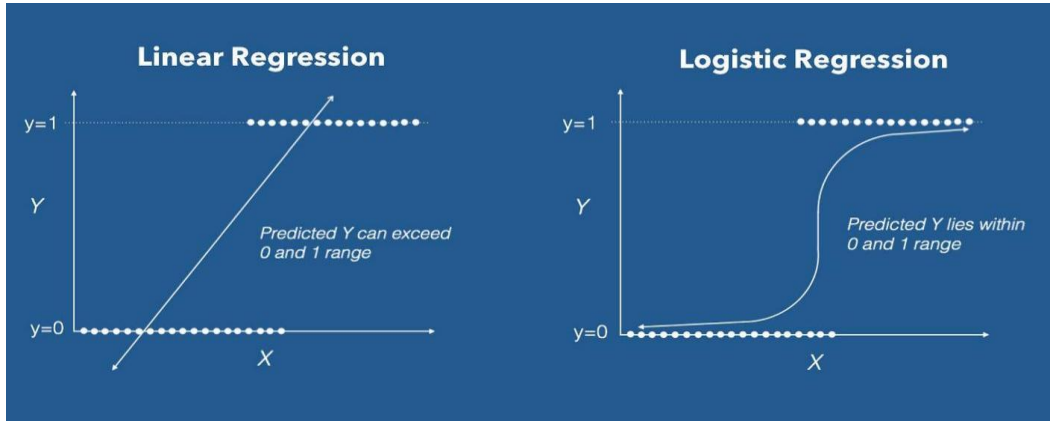


Logistic regression : Sigmoid Function



Linear regression vs Logistic regression

Linear Regression	Logistic Regression
Used to predict a dependent output variable based on independent input variable	Used to classify a dependent output variable based on independent input variable
Accuracy is measured using Least squares estimation	Accuracy is measured using Maximum Likelihood estimation
The best fit line is a straight line	The best fit is given by a curve
The output is a predicted integer value	The output is a binary value between 0 and 1 value
Used in business domain, forecasting stocks	Used for classification, image processing



Lab 1 : Logistic regression for Cancer Detection

- ✓ In this lab, we will be using the “breast cancer dataset”. it contains medical records of breast tumor characteristics, with features like radius, texture, and concavity. Each record is labeled as benign or malignant

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	target
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	0
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	0
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	1
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	0
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	1

5 rows × 31 columns

Lab 1 : Logistic regression for Cancer Detection

Import necessary libraries

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

Step 1: Load the dataset

```
data = load_breast_cancer()
X = data.data
y = data.target
```

Step 2: Split the dataset into training and testing sets

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Lab 1 : Logistic regression for Cancer Detection

Step 3: Train Logistic Regression Model

```
model = LogisticRegression()
model.fit(X_train, y_train)
```

Step 4: Predict on testing set

```
y_pred = model.predict(X_test)
```

Step 5: Calculate the number and pourcentage of correct and incorrect predictions

	Predicitions	Count	Percentage
0	Correct Predictions	110	96.491228
1	Incorrect Predictions	4	3.508772

Lab 2 : House Pricing with Regression Models

House pricing dataset :

Every row in this dataset represents a house, including the price, area, bedrooms, bathrooms, stories, main roads, guest room, basement, hot water heating, air conditioning, parking, pref area and furnishing status

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	yes	furnished
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	no	furnished
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	yes	semi-furnished
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes	furnished
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	no	furnished

- ✓ In this lab, we will use the column “price” as a target to train a model that will be able to predect the price of a new house

Lab 2 : House Pricing with Regression Models

1. Import the necessary libraries and load the dataset.

Importing dataset and libraries

```
# Importing necessary libraries
import numpy as np # import de numpy
import pandas as pd # import de pandas
from sklearn.model_selection import train_test_split # import de train test split to split the dataset into train and test set
from sklearn.linear_model import LinearRegression # import linear regression model
from sklearn.preprocessing import PolynomialFeatures #import polynomial features
import matplotlib.pyplot as plt

# Load the dataset

house_data = pd.read_csv('./Documents/Housing.csv')
# Display the first few rows of the dataset
print("Dataset preview:")
house_data.head()
```

Lab 2 : House Pricing with Regression Models

The goal of this lab is to predict the price of the new house based on its characteristics , for this purpose we will use Regression models to learn the relationship between the target ‘Price’ and the features of the dataset

2. Keep only numerical Features by removing the other columns :

```
# Selecting only numerical features for simplicity
numerical_features = house_data.select_dtypes(include=[np.number])
print ("here are the features we kept, only numerical features")
print(numerical_features.head())
```

```
here are the features we kept, only numerical features
```

	price	area	bedrooms	bathrooms	stories	parking
0	13300000	7420	4	2	3	2
1	12250000	8960	4	4	4	3
2	12250000	9960	3	2	2	2
3	12215000	7500	4	2	2	3
4	11410000	7420	4	1	2	2

Lab 2 : House Pricing with Regression Models

3. Splitting Dataset :
   - Since we are in supervised learning, split the dataset into Features and target (price)
   - Split both target and features into training and testings sets
   - Keep 20% of the data set for the test

```
# Splitting the dataset into features (X) and target variable (y)
X = numerical_features.drop(columns=['price'])  # here we drop the price because it is the target variable
y = numerical_features['price']  # Target variable "price"

# Splitting the dataset into train and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Lab 2 : House Pricing with Regression Models

4. Fit the data and get the predictions using 2 models of regression : Linear and Polynomial

Model 1: Linear regression

```
# Linear Regression
linear_reg = LinearRegression()
linear_reg.fit(X_train, y_train)
linear_pred = linear_reg.predict(X_test)
```

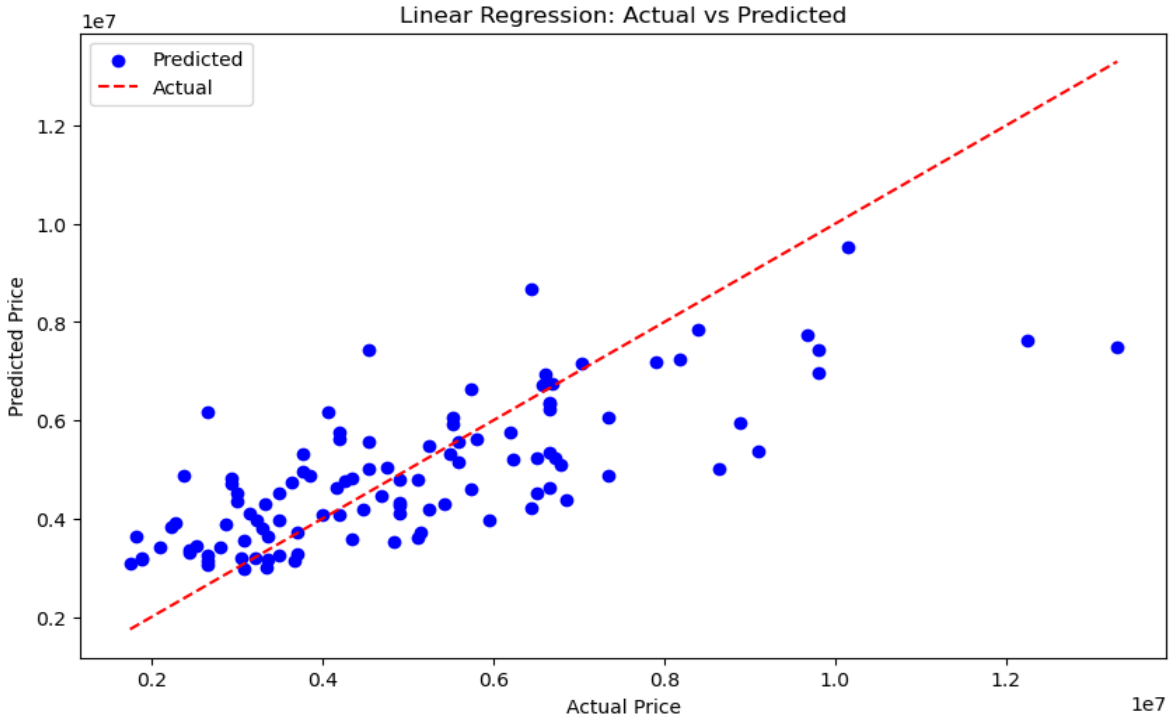
Model 2: Polynomial regression

```
# Polynomial Regression (degree=2)
poly_features = PolynomialFeatures(degree=2)
X_train_poly = poly_features.fit_transform(X_train)
X_test_poly = poly_features.transform(X_test)

poly_reg = LinearRegression()
poly_reg.fit(X_train_poly, y_train)
poly_pred = poly_reg.predict(X_test_poly)
```

Lab 2 : House Pricing with Regression Models

5. Plot the actual prices in X axis vs the predicted prices with linear regression model in Y axis

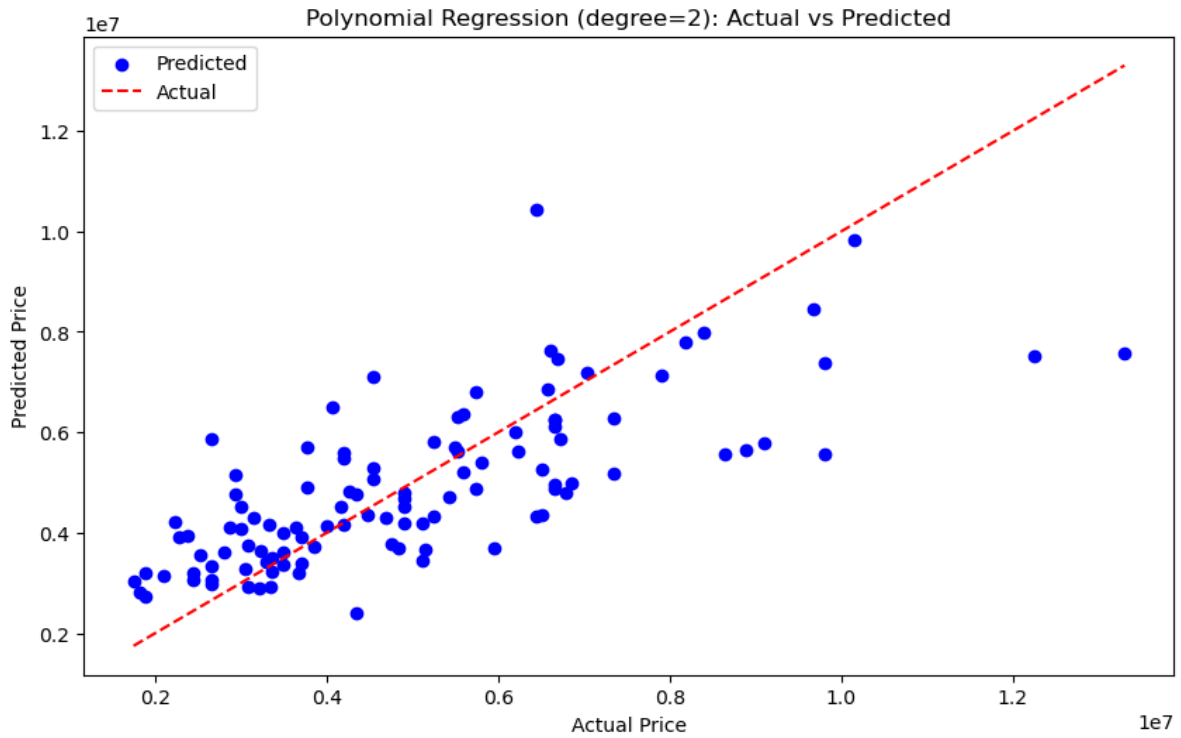
```
# Plotting real vs predicted values for Linear Regression
plt.figure(figsize=(10, 6))
plt.scatter(y_test, linear_pred, color='blue', label='Predicted')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--', label='Actual')
plt.title('Linear Regression: Actual vs Predicted')
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.legend()
plt.show()
```



Lab 2 : House Pricing with Regression Models

6. Plot the actual prices in X axis vs the predicted prices with Polynomial regression model in Y axis

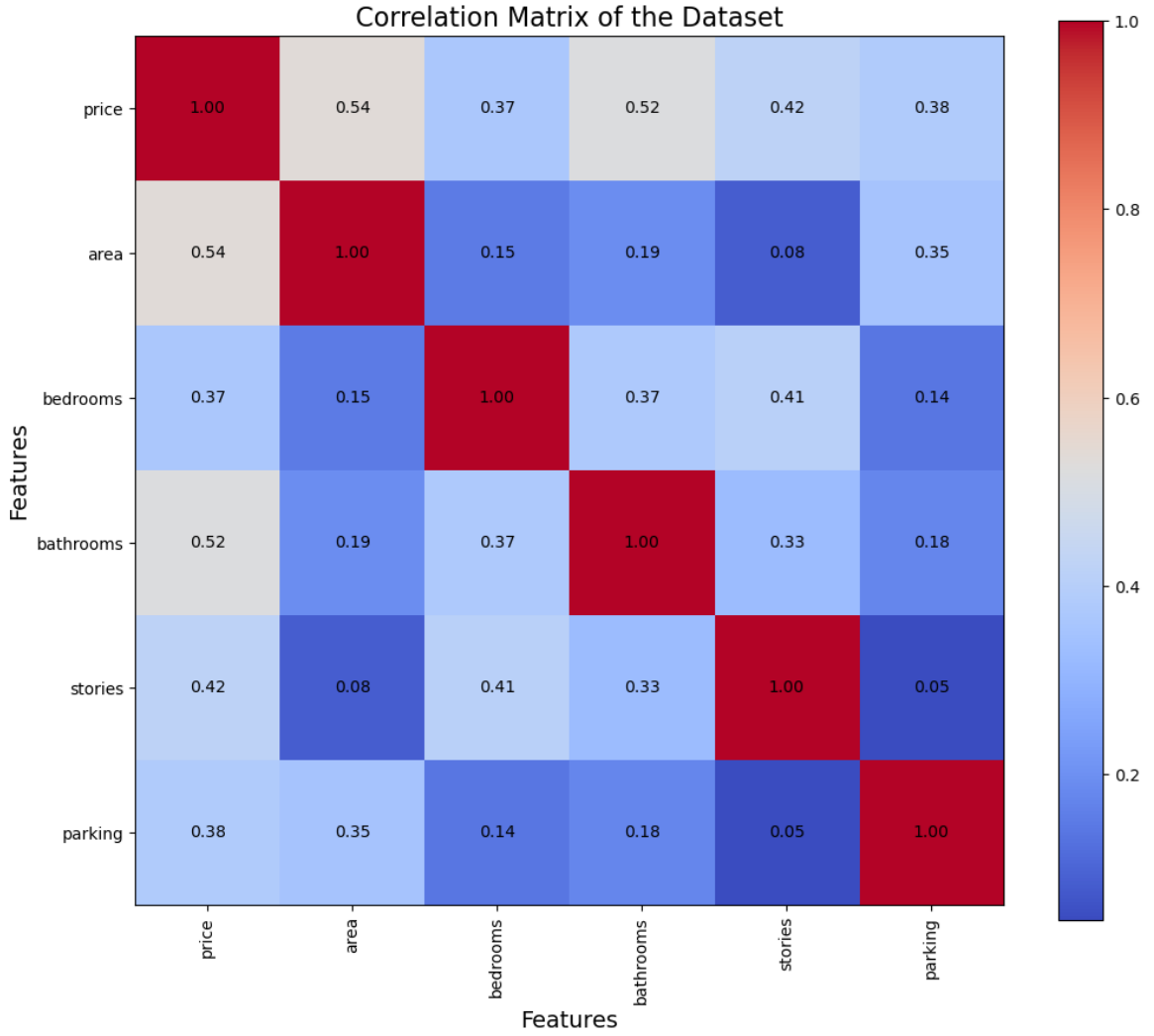
```
# Plotting real vs predicted values for Polynomial Regression
plt.figure(figsize=(10, 6))
plt.scatter(y_test, poly_pred, color='blue', label='Predicted')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--', label='Actual')
plt.title('Polynomial Regression (degree=2): Actual vs Predicted')
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.legend()
plt.show()
```



Lab 2 : House Pricing with Regression Models

Correlation matrix : it represents the correlation between each feature and the other features but also the correlation with the target in a dataset

7. Generate the correlation matrix of house pricing dataset, and analyse the correlation



Lab 2 : House Pricing with Regression Models

8. Train the linear Regression model with only ‘Area’ as Feature and predict the price of the house
9. Plot the real Price and the predicted Price using the linearity model

```python
# Prepare features and target variable
X = house_data[['area']]
y = house_data['price']

model = LinearRegression()
model.fit(X, y)

# Predict house prices using the trained model
y_pred = model.predict(X)

# Plot the scatter plot of 'Price' vs. 'Area'
plt.scatter(X, y, color="blue", label="Data")

# Plot the regression line
plt.plot(X, y_pred, color="red", linewidth=3, label="Regression Line")

plt.xlabel('Area')
plt.ylabel('Price')
plt.title("Price vs. Area ")
plt.legend()
plt.show()
```

Thank you!

www.easi.net

z.gasmi@easi.net