MACHINE LEARNING WITH PYTHON FOR SPACE WEATHER APPLICATIONS

by ITU Upper Atmosphere and Space Weather Laboratory

Lecture 1: Introduction to Python Packages

Instructor: Dogacan Su Ozturk

Contact: dsozturk@alaska.edu

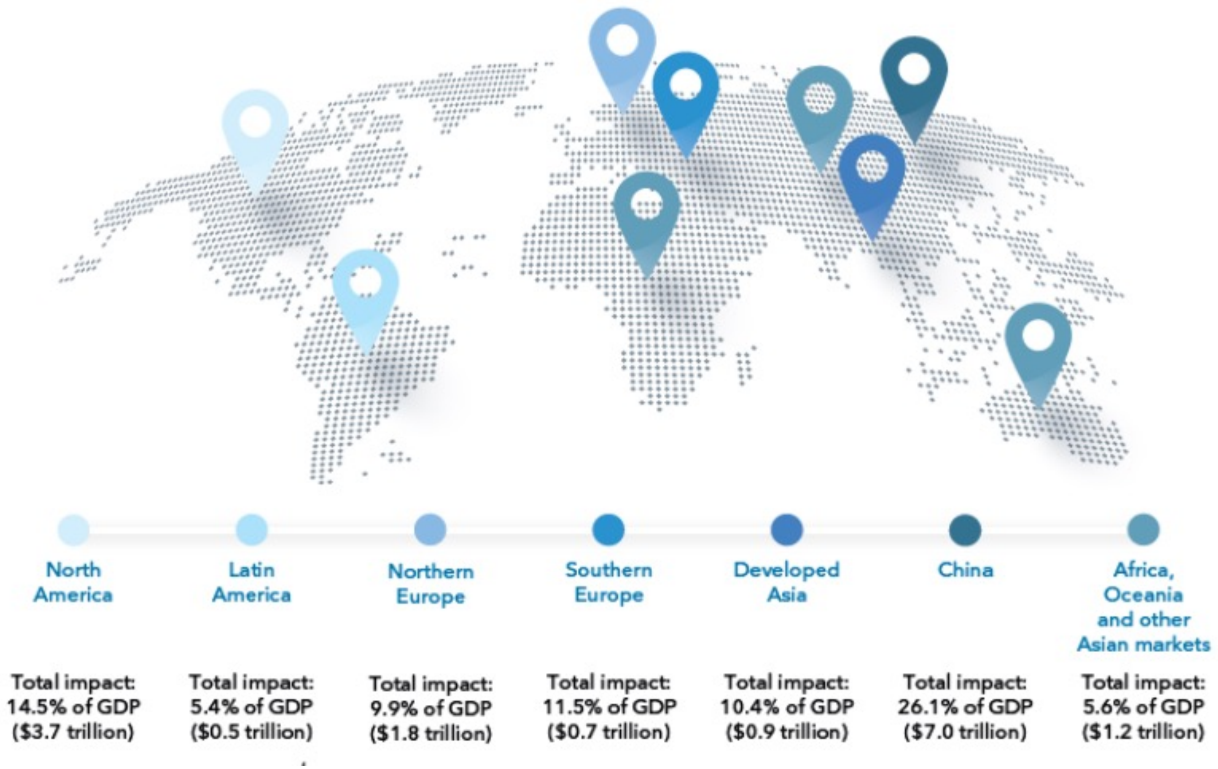
11-12 May 2022 at Upper Atmosphere and Space Weather Laboratory

Introduction: Machine learning

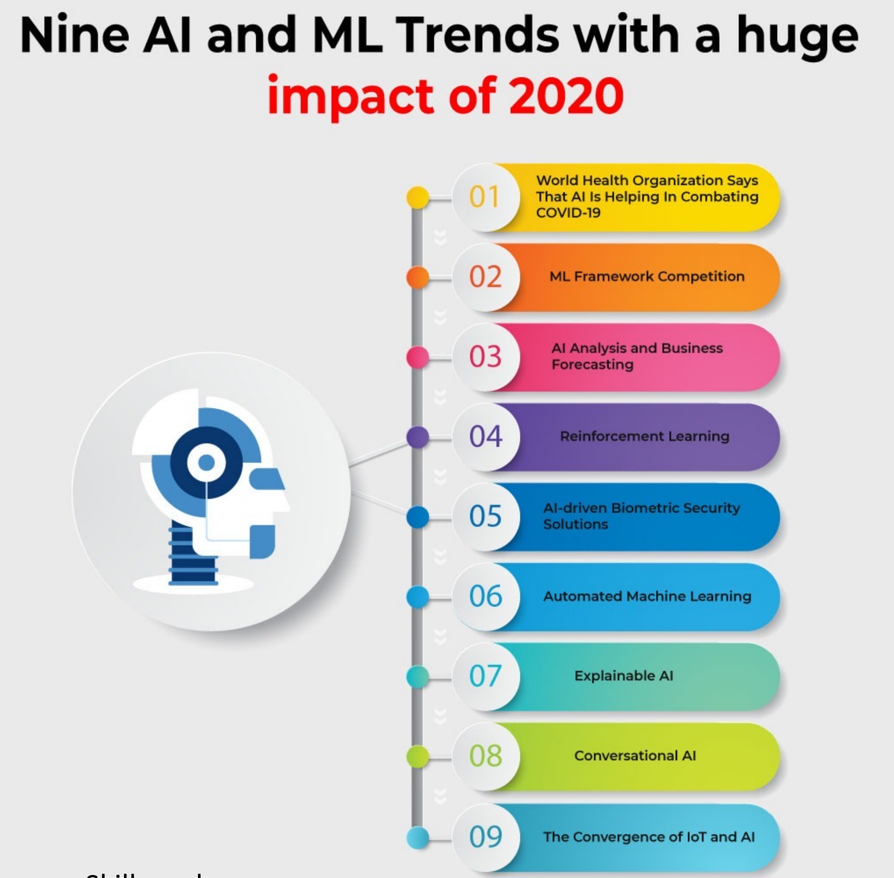
- Machine learning is to use the ability of algorithms to extract information from a database.
  - Predictive analysis
  - Statistical learning

- Machine learning applications are widely used in our every day life.
  - Can you think of any commercial use?
  - Can you think of any research applications?

Introduction: Machine learning



Source: Digital Regulation Platform

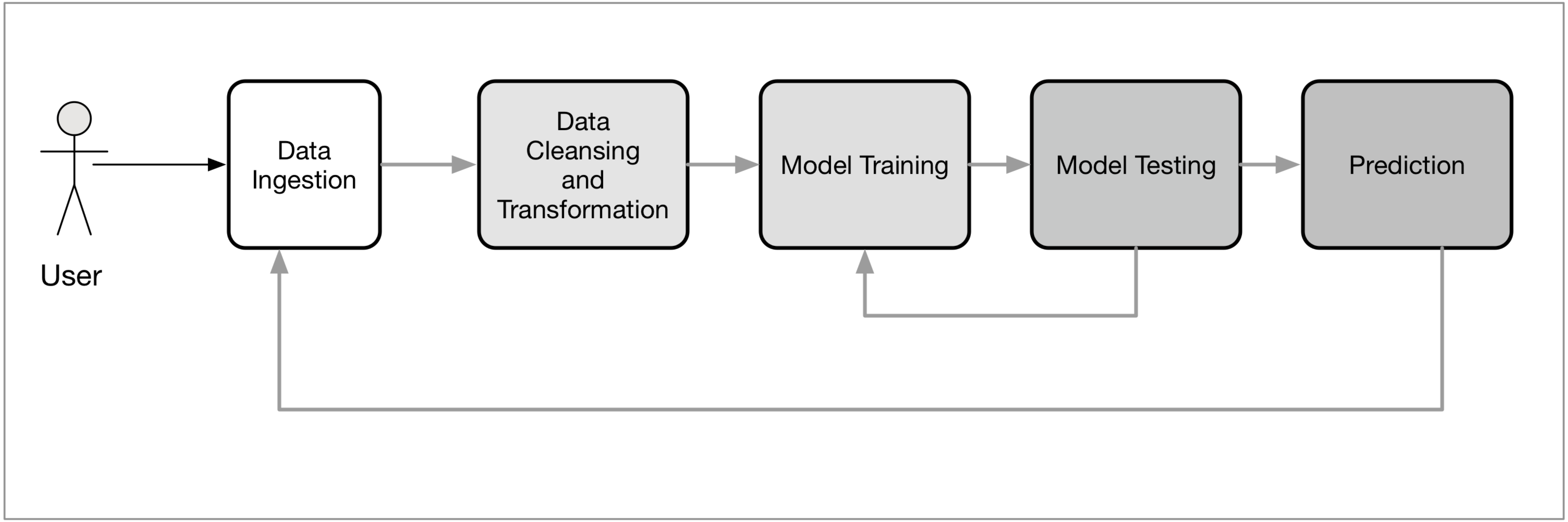


Source: Skillmonks

Machine learning skills will continue to be on high demand in the foreseeable future.

Introduction: Machine learning

A machine learning application consists of the following steps:



1. Data ingestion: Frame the problem

1. Define the objective in business/research terms.
2. Determine how your solutions will be used.
3. Identify the current solutions/workarounds (if any).
4. Determine how to frame the problem (supervised/unsupervised?).
5. Determine performance metrics.
6. Check if the performance metric is aligned with the business/research objectives.
7. Determine the minimum performance necessary to reach the objectives.
8. Determine if the experience and tools are transferrable.
9. Check if human expertise is available.
10. Identify how the solution would look like manually.
11. List all assumptions.
12. Verify the assumptions.

Image Credit: Double Take by Susan Hood and Jay Fleck

1. Data ingestion: Data acquisition

1. List the data you need and how much of it you need.
2. Find and document where you can get the data.
3. Check how much space the data will take.
4. Check legal obligations (and ethical!). Acquire authorization if necessary.
5. Create a workspace.
6. Get the data.
7. Convert the data to a format you can easily manipulate.
8. Ensure sensitive information is protected.
9. Check the size of the data.
10. Sample a test set, put it aside, and never look at it until its time.

Illustration by Andrew Krasovitckii

1. Data Ingestion: Exploratory Analysis

1. Create a copy of data for exploration.
2. Create a Jupyter notebook to keep a record of your exploration.
3. Study each attribute and characteristics:
   a. Name
   b. Type
   c. % of missing values
   d. Noisiness
   e. Usefulness
   f. Distribution
4. Identify feature and target for supervised tasks.
5. Visualize the data.
6. Study the correlations between attributes.
7. Study how you would solve the problem manually.
8. Identify necessary transformations.
9. Document what you have learnt.

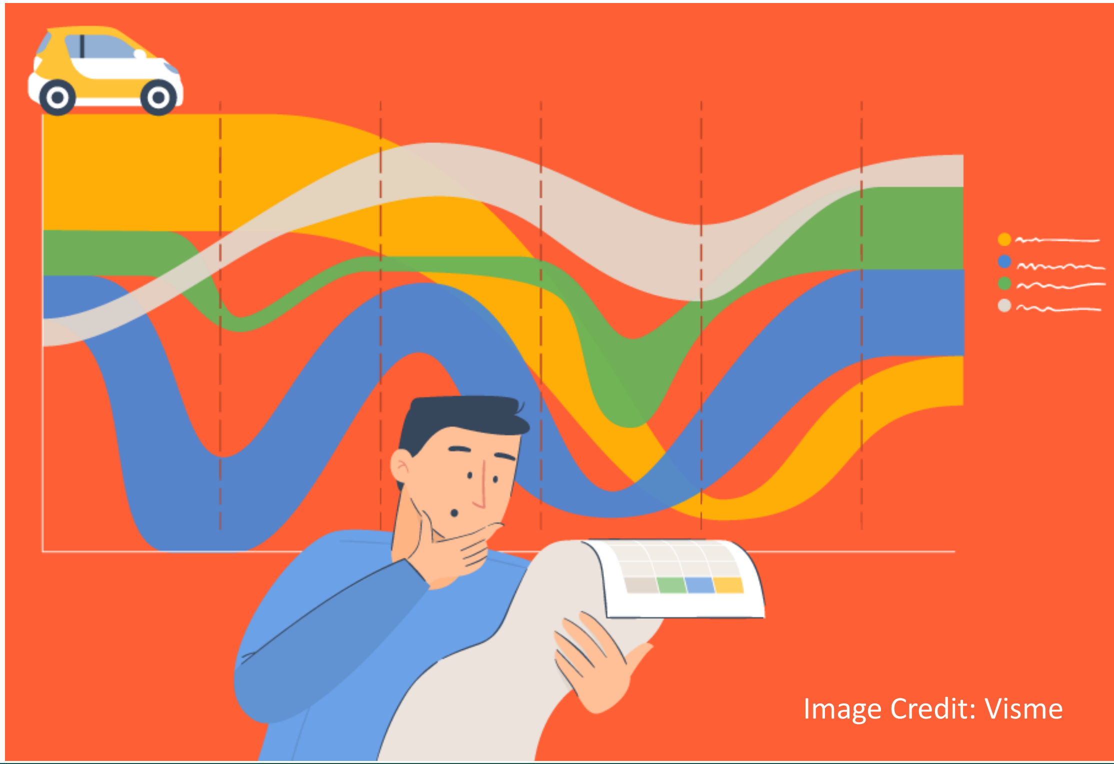


Image Credit: Visme

2. Data Cleaning and Transformation: Cleaning

Data cleaning consists of following steps:

1. Cleaning duplicate entries
2. Handling NaNs (missing values)
3. Fixing categorical errors
4. Removing outliers and unphysical data

2. Data Cleaning and Transformation: Handling Text and Categorical Attributes

- To handle text and categorical values, we can:
  - Create dummies
  - Use encoders
  - Create our own labels

Image Credit: UBC Learning commons

2. Data Cleaning and Transformation: Feature engineering

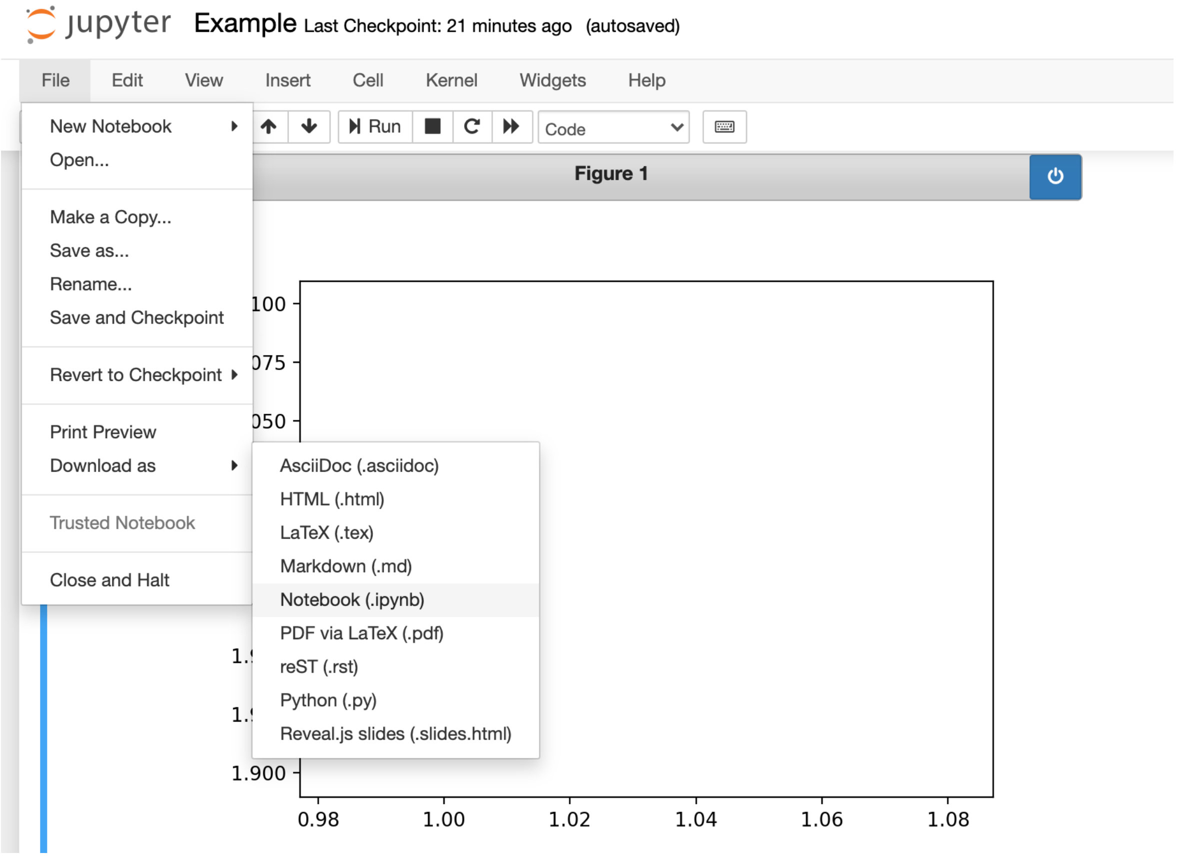
- Feature engineering is where your personality and creativity shines in Machine Learning.

- It depends on how well you understand the data and the associated tasks. This is called using “domain knowledge”.

- Feature engineering consists of the following tasks:
  - Deciding on new features
  - Creating features
  - Prioritizing certain features/categories
  - Interpreting ML models to decide what features are needed.

Introduction: Jupyter Notebook

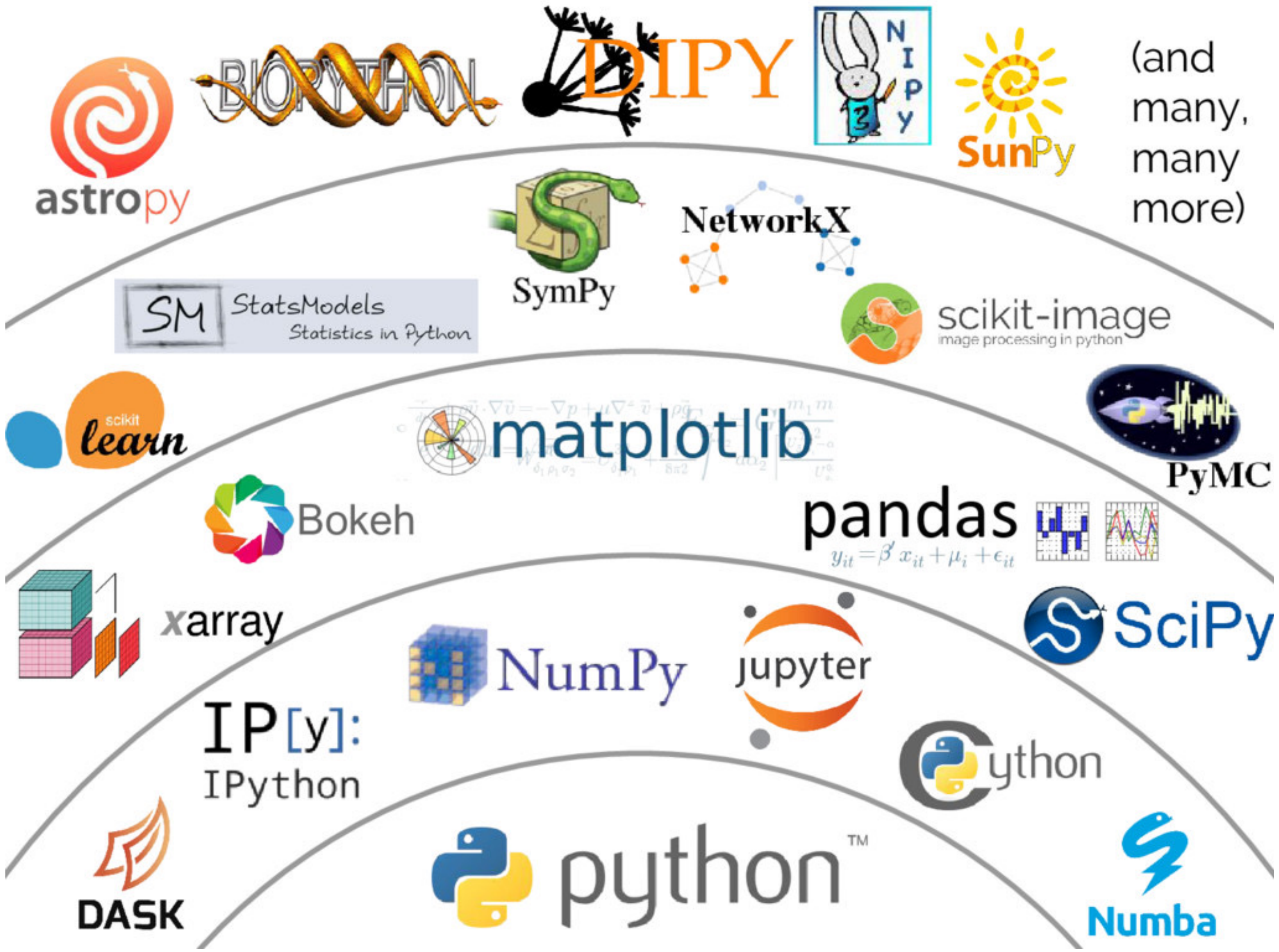
- The Jupyter Notebook is an open-source web application.
- You can create and share documents that contain live code, equations, visualizations and narrative text.
- It is a very powerful computational tool that will be sufficient for our class exercises.
- You can add titles, figures, and save Jupyter notebook in different formats.



* If you have trouble downloading the file as a PDF via LaTeX,
you can always use Print >> Save as PDF option.

Introduction: Python libraries

- Python libraries are suites of functions and modules that help with common
  operations and data formats.
- There are standard and field specific libraries.
- In this course we will use standard libraries as well as Numpy, Matplotlib, Seaborn, Pandas, and Sklearn.
- You need to import libraries in the beginning of each code you write.



Introduction: Python data types and structures

- You can always find more information at: https://numpy.org/doc/stable/user/basics.types.html

Tip: type(variable) will return the variable type.

Data Type	Definition
Index [BI]	The order of a specific element
Boolean [BI]	Logical propositions like True or False
String [BI]	Character arrays in single or double quotes like 'Hello World!'
Integer [BI]	Positive or negative whole numbers like 3 or -512.
Float [BI]	Floating point real numbers like 3.14159 or -2.5.
Complex [NP]	Complex number (real and imaginary components) like -1.0 + 0.5j
Datetime object [DT]	Basic day and time data like t1 = dt.datetime(2021,3,29,13,20)

Data Structure	Definition
Array [NP]	Stores ordered numerical values [NumPy]
List [BI]	Stored collections of ordered data like ["apple", "banana", "cherry"]
Tuple [BI]	Like list but unchangeable.
Sequence [BI]	The indicator of a specific element like {"apple", "banana", "cherry"}
Dictionary [BI]	Stored data in key:value pairs like {"item" : "shoe", "color" : "red", "size" : 8}
Set [BI]	Stored collections of unordered data like {"apple", "banana", "cherry"}
Dataframe [PD]	A spreadsheet like dictionary objects [Pandas]

*BI: Built-in, NP: NumPy, DT: Datetime, PD: Pandas

Introduction: Numpy

You can always find more information at:

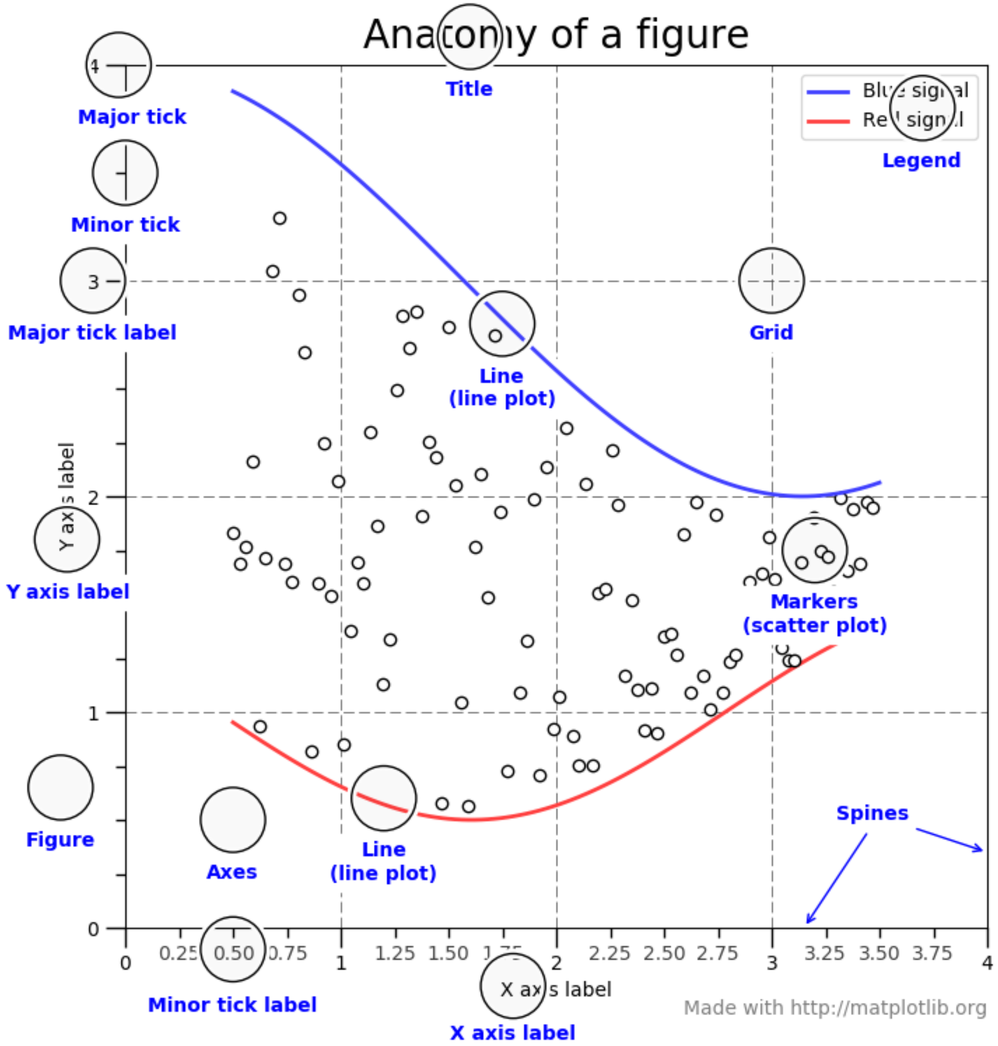
- https://numpy.org/doc/stable/user/quickstart.html
- https://scipy-lectures.org/intro/numpy/array_object.html

- NumPy is one of the most fundamental packages of Python.
- NumPy leverages the power of vectorization to conduct computations in minimal amount of time.
- To call the NumPy library: import numpy as np
- To create basic NumPy arrays: np.array(), np.zeros(), np.ones(), np.empty(), np.arange(), np.linspace()
- To specify shape or length: len(), np.shape(), .ndim, .shape, .size

Introduction: Matplotlib

You can always find more information at:
https://matplotlib.org/stable/index.html

- Matplotlib is a visualization library for creating static, animated, and interactive graphics with Python.
- Visualizing is a great way of conveying complex information.
- Python is object-oriented, this means all the components of a plot is an object, therefore it can be CUSTOMIZED!
- You need %matplotlib notebook or %matplotlib inline commands in Jupyter notebook for images to display.

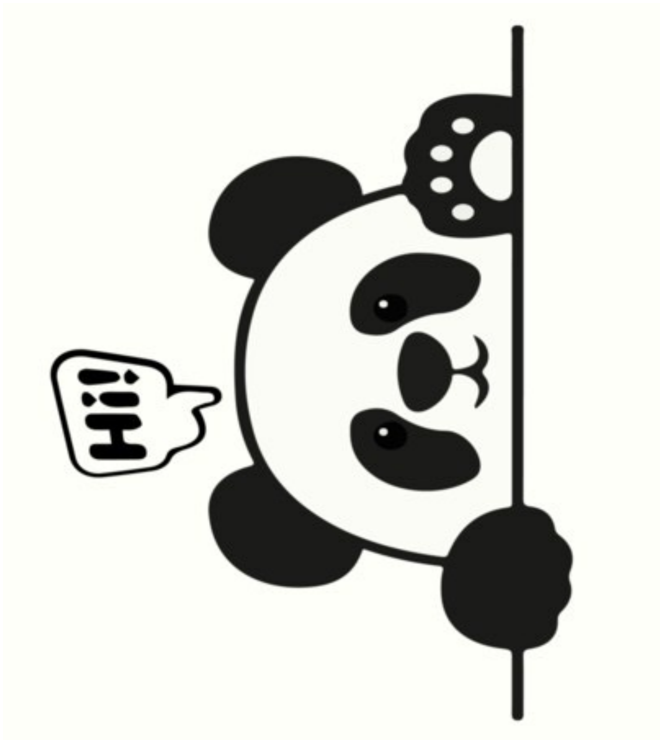


Introduction: Pandas

You can always find more information at: https://pandas.pydata.org/

Pandas is a fast and powerful data analysis and manipulation tool.

- The core of Pandas library is the Pandas dataframes.
- Dataframes are tabular, “excel”-like data structures.
- To call the Pandas library: import pandas as pd
- To create basic Pandas Dataframes: pd.DataFrame()
- A Python dictionary can be converted into Pandas dataframe.
- An entire dataset can be read into Pandas dataframe with one line of code.



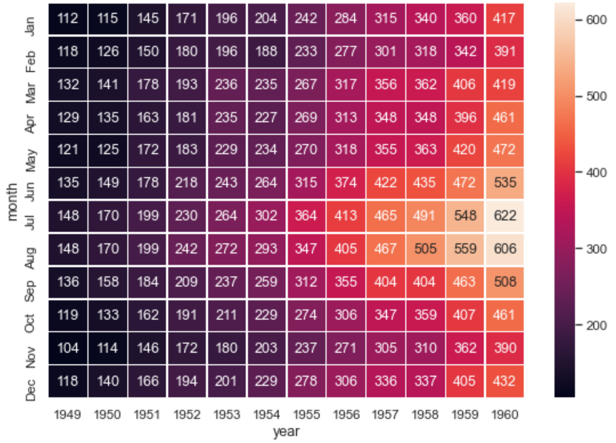
Pandas: Ways to load data sets

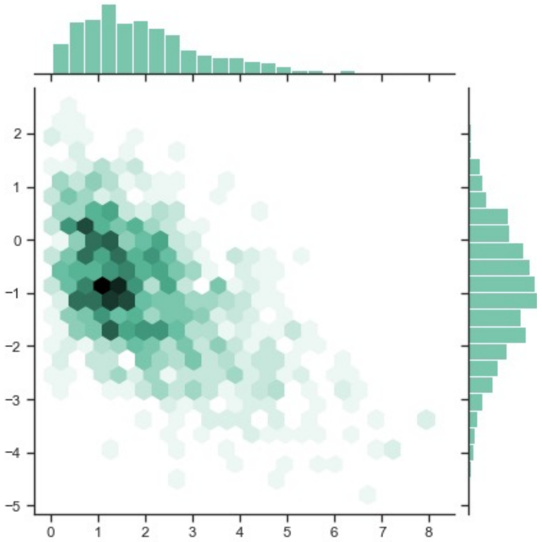
1. Read data manually
2. Load data from CSV file >> pd.read_csv()
3. Load data from Excel file >> pd.read_excel()
4. Load data from XML >> pd.read_xml()
5. Load data HTML >> pd.read_html()
6. Load data from JSON >> pd.read_json()
7. Load data from SQL >> pd.read_sql()
8. Load data from pickle >> pd.read_pickle()

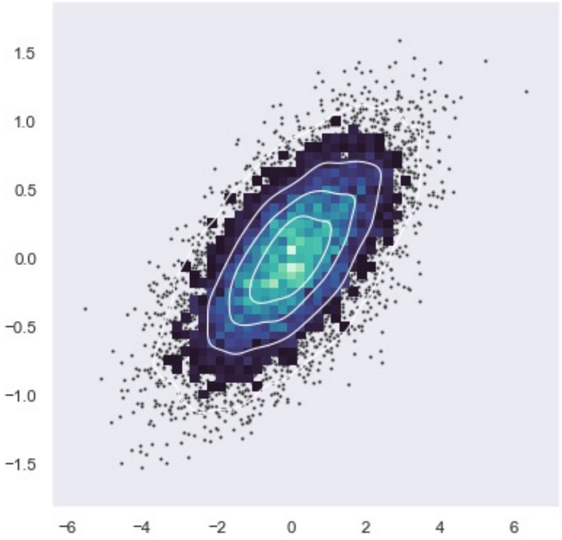
Introduction: Seaborn

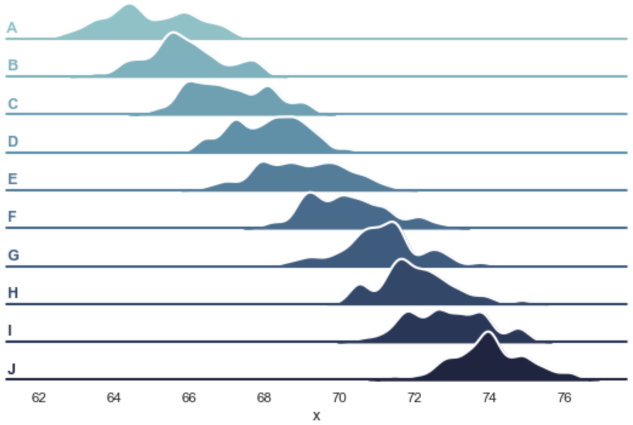
You can always find more information at:
https://seaborn.pydata.org/tutorial.html

- Seaborn is a high-level data visualization library for creating informative statistical graphics.
- Seaborn is based on matplotlib, so some functionality is interchangeable.
- Seaborn helps exploring and understanding the data better.



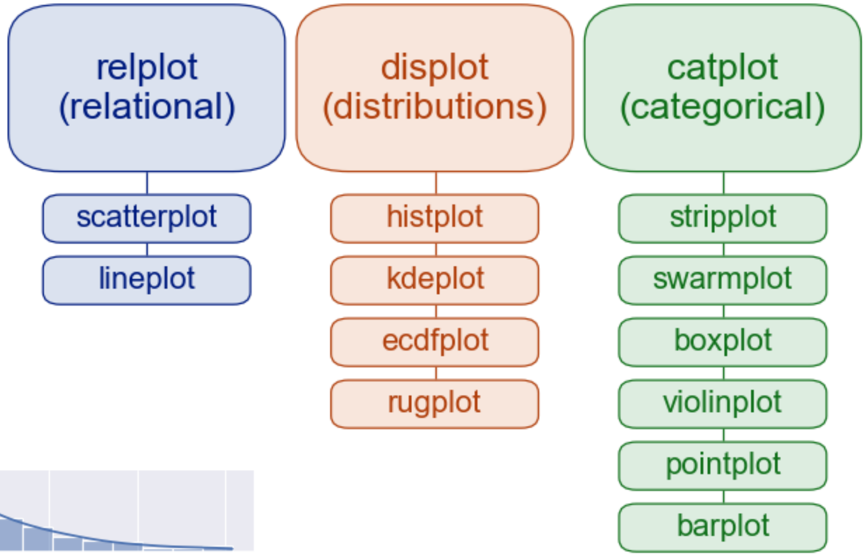


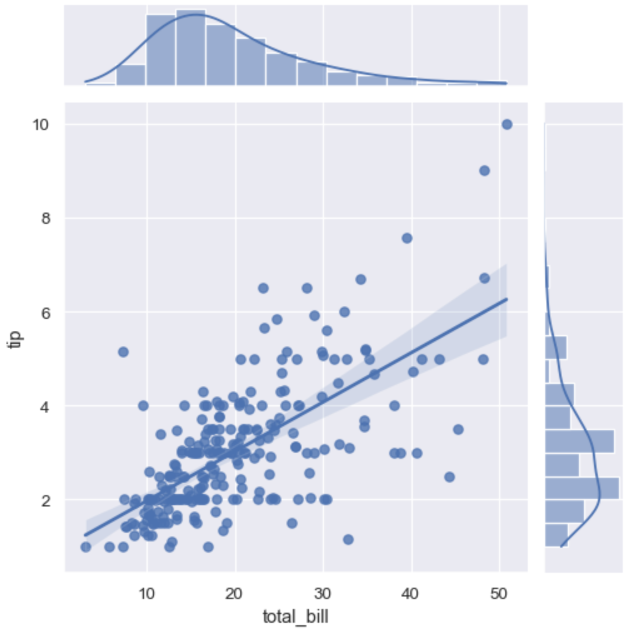


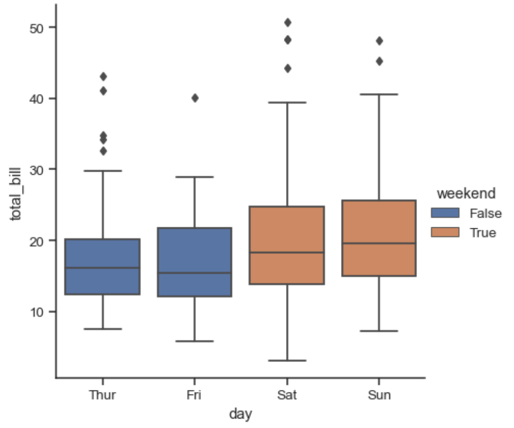


Introduction: Seaborn

- Regplot : To plot data and regression
- Lmplot: To plot data and regression with fit for more than one set
- Relplot: To plot relational plots
- Jointplot: To plot two variables with bivariate and univariate graphs
- Pairplot: To plot pairwise relationships
- Countplot: To plot counts of observations in each categorical bin using bars (histogram)
- Boxplot: To plot distributions with respect to categories
- Violinplot: To plot a combination of boxplot and kernel density estimate
- Heatmap: To plot rectangular data as a color-encoded matrix.







Introduction: Scikit-learn

You can always find more information at:
https://scikit-learn.org/stable/

- Scikit-learn is a code library for predictive and statistical analysis.
- It is built on NumPy, SciPy, and matplotlib.
- Sklearn is used in various stages of machine learning.
- To call the Scikit-Learn library: import sklearn

We will use sklearn in all steps of machine learning, but today we will focus on:

1. Data Cleaning
2. Data Transforming
3. Data Splitting
4. Model Validation

1. Sklearn for Data Cleaning: Missing Values

Most machine learning algorithms can not work with missing features.

>> Pandas dataframe has .dropna(), .drop(), or .fillna() options to get rid of missing values.

With scklearn we can use the following to take care of missing values:

1. Simple Imputer function: from sklearn.impute import SimpleImputer
   class sklearn.impute.SimpleImputer(**, missing_values=nan, strategy='mean',*
   fill_value=None, verbose=0, copy=True, add_indicator=False)
   - Missing values can be: int, float, str, np.nan, or None
   - Strategy for imputation can be: mean, median, most_frequent, or constant
   - fill_value only needed if strategy is constant.
2. Iterative Imputer function
3. Nearest-neighbor Imputer function

1. Sklearn for Data Cleaning: Missing Values

Most machine learning algorithms can not work with missing features.

>> Pandas dataframe has .dropna(), .drop(), or .fillna() options to get rid of missing values.

With scklearn we can use the following to take care of missing values:

1. Simple Imputer function
2. Iterative Imputer function: from sklearn.impute import IterativeImputer
   class sklearn.impute.IterativeImputer(*estimator=None, *, missing_values=nan, sample_posterior=False, max_iter=10, tol=0.001, n_nearest_features=None, initial_strategy='mean', imputation_order='ascending', skip_complete=False, min_value=-inf, max_value=inf, verbose=0 , random_state=None, add_indicator=False*)
   - Needs enabling of experimental module from sklearn.experimental import enable_iterative_imputer
3. Nearest-neighbor Imputer function

1. Sklearn for Data Cleaning: Missing Values

Most machine learning algorithms can not work with missing features.

>> Pandas dataframe has .dropna(), .drop(), or .fillna() options to get rid of missing values.

With sklearn we can use the following to take care of missing values:

1. Simple Imputer function
2. Iterative Imputer function
3. Nearest-neighbor Imputer function: from sklearn.impute import KNNImputer
   class sklearn.impute.KNNImputer(**, missing_values=nan*, *n_neighbors=5*, *weights='uniform'*, *metric='nan_euclidean'*, *copy=True*, *add_indicator=False*)
   - Missing values can be: int, float, str, np.nan, or None
   - n_neighbors = int
   - Weights can be ‘uniform’, ‘distance’, or callable.

2.1. Sklearn for Data Transforming: Handling text and categories

In many ML applications, you will encounter categorical or string type data, that needs to be transformed to numerical values.

With sklearn we can use the following functions to transform categorical data:

1. LabelEncoder: from sklearn.preprocessing import LabelEncoder

   classes_*ndarray of shape (n_classes,)*

   - n_classes: number of classes [0 to n]
2. Ordinal Encoder
3. OneHotEncoder

2.1. Sklearn for Data Transforming: Handling text and categories

In many ML applications, you will encounter categorical or string type data, that needs to be transformed to numerical values.

With sklearn we can use the following functions to transform categorical data:

1. LabelEncoder
2. Ordinal Encoder: from sklearn.preprocessing import OrdinalEncoder

   class sklearn.preprocessing.OrdinalEncoder(**, categories='auto', dtype=<class 'numpy.float64'>, handle_unknown='error', unknown_value=None*)

   - categories can be 'auto' or 'list'.
   - handle_unknown can be 'error' or 'use_encoded_value'.
3. OneHotEncoder

2.1. Sklearn for Data Transforming: Handling text and categories

In many ML applications, you will encounter categorical or string type data, that needs to be transformed to numerical values.

With sklearn we can use the following functions to transform categorical data:

1. LabelEncoder
2. Ordinal Encoder
3. OneHotEncoder : from sklearn.preprocessing import OneHotEncoder
   class sklearn.preprocessing.OneHotEncoder(**, categories='auto', drop=None, sparse=True, dtype=<class 'numpy.float64'>, handle_unknown='error'*)
   - categories can be ‘auto’ or ‘list’.
   - drop can be ‘first’, ‘if_binary’, or array-like
   - handle_unknown can be ‘error’ or ‘ignore’.

2.2. Sklearn for Data Transforming: Feature Scaler

ML algorithms don’t perform well when the input numerical attributes have very different scales.

With sklearn we can use the following scaler functions to transform the data:

1. MinMaxScaler: from sklearn.preprocessing import MinMaxScaler
   class sklearn.preprocessing.MinMaxScaler(*feature_range=0, 1*, *, *copy=True*, *clip=False*)
   - feature_range: desired range of transformed data, tuple. (0,1)
2. StandardScaler: from sklearn.preprocessing import StandardScaler
   class sklearn.preprocessing.StandardScaler(*, *copy=True*, *with_mean=True*, *with_std=True*)
   - with_mean: Boolean, if true center data before scaling.
3. MaxAbsScaler: from sklearn.preprocessing import MaxAbsScaler
   class sklearn.preprocessing.MaxAbsScaler(*, *copy=True*)

3. Sklearn for Data Splitting

Now it is time to split the data into two categories: Train and test.

(for Neural Networks a third set of validation set is necessary).

With sklearn we can use the following functions to split the data set in to training and test sets:

1. Time series: from sklearn.model_selection import TimeSeriesSplit
   class sklearn.model_selection.TimeSeriesSplit(*n_splits=5*, ***, *max_train_size=None*, *test_size=None*, *gap=0*)
   - n_splits = integer, number of splits.
2. Train test split: from sklearn.model_selection import train_test_split
   sklearn.model_selection.train_test_split(**arrays*, *test_size=None*, *train_size=None*, *random_state=None*, *shuffle=True*, *stratify=None*)
   - arrays, X and y.
   - test_size: float between 0.0 to 1.0
   - train_size: float between 0.0 to 1.0

There are other splitting functions in sklearn, which are beyond the scope of this class.

What is error?

Most commonly used error descriptors:

- Mean absolute error

$$\text{mae } error = \frac{1}{N}\sum_{i=1}^{N}|actual\ values\ - predictions|$$

- Mean absolute percentage error

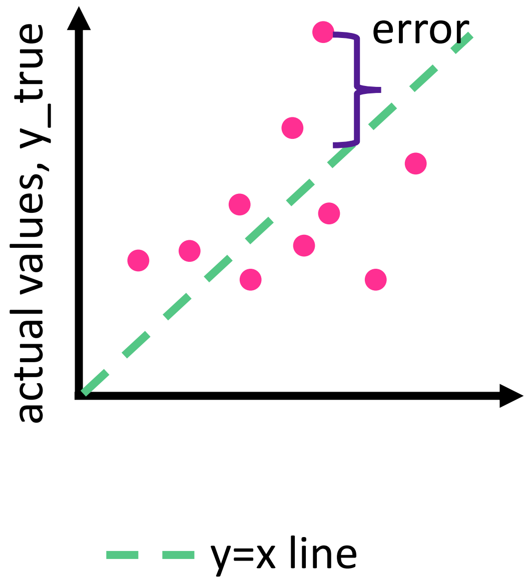
$$\text{mape } error = \frac{100\ \%}{N}\sum_{i=1}^{N}\left|\frac{actual\ values\ - predictions}{actual\ values}\right|$$

- Mean squared error

$$\text{mse } error = \frac{1}{N}\sum_{i=1}^{N}(actual\ values\ - predictions)^2$$

- Root mean squared error

$$\text{rmse } error = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(actual\ values\ - predictions)^2}$$



What is norm?

- Norm is the total size/length of a matrix in matrix space (including vectors).
- $||x||_p = \sqrt[p]{\sum_i |x_i|^p}$
- Most commonly used l-norms are:$l_0, l_1, l_2, and\ l_\infty$
- MAE → L1
- MSE → L2

4. Sklearn for Error Analysis

Most commonly used error descriptors:

- Mean absolute error: from sklearn.metrics import mean_absolute_error
  sklearn.metrics.mean_absolute_error(*y_true, y_pred, *, sample_weight=None, multioutput='uniform_average'*)

- Mean absolute percentage error: from sklearn.metrics import mean_absolute_percentage_error sklearn.metrics.mean_absolute_percentage_error(y_true, y_pred, *, sample_weight=None, multioutput='uniform_average')
- Mean squared error: from sklearn.metrics import mean_squared_error

sklearn.metrics.mean_squared_error(*y_true, y_pred, *, sample_weight=None, multioutput='uniform_average', squared=False*)

- Root mean squared error: from sklearn.metrics import mean_squared_error

sklearn.metrics.mean_squared_error(*y_true, y_pred, *, sample_weight=None, multioutput='uniform_average', squared=True*)