Lambton College
School of Computer Studies

# AML-2203 Advanced Python AI and ML Tools

[Prepared By: Vahid Hadavi, PhD]

# Lecture1

1. Evaluate File and data set handling in Python (Recommended Module: scikit-learn, pandas)

## Major Topics:

- 1.1 Discuss various format for data sets
- 1.2 Use Python to import various JSON and CSV files
- 1.3 Use Python to generate a random data set
- 1.4 Use Python modules to manipulate imported data and save them on disk

# 1.1 Discuss various format for data sets

# File format

**What is a file format?**

- A file format is a standard way in which information is encoded for storage in a file.
- It specifies whether the file is a binary or ASCII file.
- it shows how the information is organized.
  - For example, comma-separated values (CSV) file format stores tabular data in plain text.

**Why we should understand different file formats?**

- Based on the application, different file formats are used.
- Choosing the optimal file format for storing data can improve the performance of your models in data processing.

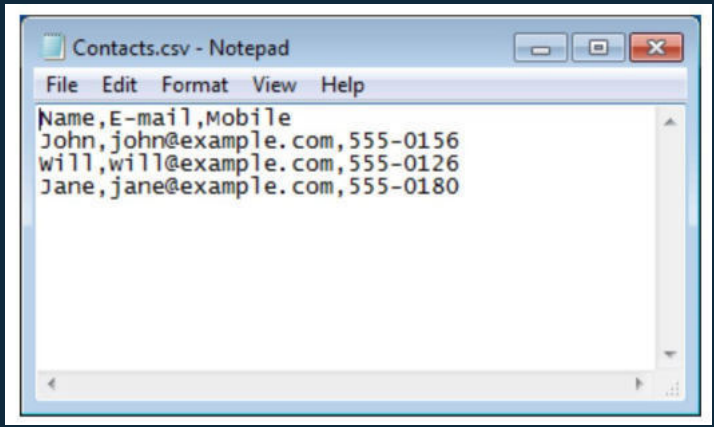
# File format

**Types**

- Comma-separated values (CSV)
- JSON
- ZIP
- Plain Text (txt)
- XLSX
- XML
- HTML
- Images
- Hierarchical Data Format
- PDF
- DOCX
- MP3
- MP4

# CSV

- CSV file format stores tabular data in plain text.
- Each line in CSV file represents an observation or commonly called a record.
- Each record may contain one or more fields which are separated by a comma.



# JSON

- JavaScript Object Notation.
- A text-based open standard designed for exchanging the data over web.
- It is primarily used for transmitting data between a web application and a server.

```
{
  "users": [
    {
      "userId": 1,
      "firstName": "Chris",
      "lastName": "Lee",
      "phoneNumber": "555-555-5555",
      "emailAddress": "clee@fileinfo.com"
    },
    {
      "userId": 2,
      "firstName": "Action",
      "lastName": "Jackson",
      "phoneNumber": "555-555-5556",
      "emailAddress": "ajackson@fileinfo.com"
    },
    {
      "userId": 3,
      "firstName": "Ross",
      "lastName": "Bing",
      "phoneNumber": "555-555-5557",
      "emailAddress": "rbing@fileinfo.com"
    },
```

# 1.2 Use Python to import various JSON and CSV files

# Loading CSV in python

- For loading the data you can use the “pandas” library in python.

```
import pandas as pd
pd.read_csv('train.csv')
```

- Some important arguments for read_csv method:
  - File path (required)
  - names (optional): lists of column names
  - sep (default “,”)
  - header (optional): row number to use as column names
  - index_col (optional): column to use as row numbers

# Loading JSON in python

- For loading the data you can use the “pandas” library in python.

```
import pandas as pd
pd.read_json('train.json')
```

- Some important arguments for read_json method:
  - File path (required)
  - typ (optional): {‘frame’, ‘series’} the type of object to recover

# 1.3 Use Python to generate a random data set

# Creating random data

- scikit-learn offers many methods for creating simulated data.
- Three most used methods to create random data in scikit-learn:
  - make-regression
    Designed to be used with linear regression
  - Make-classification
    Creating a simulated dataset for classification
  - Make-blobs
    Designed to work well with clustering techniques

# Make-regression example

- Creating 100 samples with 3 features.

```
features, target, coefficients = make_regression(n_samples = 100,
                                                 n_features = 3,
                                                 n_informative = 3,
                                                 n_targets = 1,
                                                 noise = 0.0,
                                                 coef = True,
                                                 random_state = 1)
```

- We can check the randomly created features and the corresponding targets.

```
print('Feature Matrix\n', features[:3])
print('Target Vector\n', target[:3])

Feature Matrix
 [[ 1.29322588 -0.61736206 -0.11044703]
 [-2.793085    0.36633201  1.93752881]
 [ 0.80186103 -0.18656977  0.0465673 ]]
Target Vector
 [-10.37865986  25.5124503   19.67705609]
```

# Make-classification example

- Creating 100 samples with 3 features for two classes.

```
features, target = make_classification(n_samples = 100,
                                       n_features = 3,
                                       n_informative = 3,
                                       n_redundant = 0,
                                       n_classes = 2,
                                       weights = [.25, .75],
                                       random_state = 1)
```

- We can check the randomly created features and the corresponding targets.

```
print('Feature Matrix\n', features[:3])
print('Target Vector\n', target[:3])

Feature Matrix
 [[ 1.06354768 -1.42632219  1.02163151]
 [ 0.23156977  1.49535261  0.33251578]

 [ 0.15972951  0.83533515 -0.40869554]]
Target Vector
 [1 0 0]
```

# 1.4 Use Python modules to manipulate imported data and save them on disk

# Manipulating Data

- We use the Titanic dataset on Kaggle.

```
# Load library
import pandas as pd

# Create URL
url = 'https://tinyurl.com/titanic-csv'

# Load data
dataframe = pd.read_csv(url)
```

- Using head() or tail() to check the first or last 5 (default) rows of the data frame

```
dataframe.head(2)
```

| | Name | PClass | Age | Sex | Survived | SexCode |
|---|---|---|---|---|---|---|
| **0** | Allen, Miss Elisabeth Walton | 1st | 29.0 | female | 1 | 1 |
| **1** | Allison, Miss Helen Loraine | 1st | 2.0 | female | 0 | 1 |

# Manipulating Data

- Using .shape to check the dimensions of the dataframe.

```
dataframe.shape
(1313, 6)
```

- Using describe() to get descriptive statistics for any numeric columns.

| | Age | Survived | SexCode |
|---|---|---|---|
| **count** | 756.000000 | 1313.000000 | 1313.000000 |
| **mean** | 30.397989 | 0.342727 | 0.351866 |
| **std** | 14.259049 | 0.474802 | 0.477734 |
| **min** | 0.170000 | 0.000000 | 0.000000 |
| **25%** | 21.000000 | 0.000000 | 0.000000 |
| **50%** | 28.000000 | 0.000000 | 0.000000 |
| **75%** | 39.000000 | 1.000000 | 1.000000 |
| **max** | 71.000000 | 1.000000 | 1.000000 |

# Manipulating Data

- Using .shape to check the dimensions of the dataframe.

```
dataframe.shape
(1313, 6)
```

- Using describe() to get descriptive statistics for any numeric columns.

| | Age | Survived | SexCode |
|---|---|---|---|
| **count** | 756.000000 | 1313.000000 | 1313.000000 |
| **mean** | 30.397989 | 0.342727 | 0.351866 |
| **std** | 14.259049 | 0.474802 | 0.477734 |
| **min** | 0.170000 | 0.000000 | 0.000000 |
| **25%** | 21.000000 | 0.000000 | 0.000000 |
| **50%** | 28.000000 | 0.000000 | 0.000000 |
| **75%** | 39.000000 | 1.000000 | 1.000000 |
| **max** | 71.000000 | 1.000000 | 1.000000 |

# Saving dataframe

- We can save dataframe using to_csv() and to to_json() functions on pandas.
- Desired File name must be given (file path is optional)

```
nme = ["aparna", "pankaj", "sudhir", "Geeku"]
deg = ["MBA", "BCA", "M.Tech", "MBA"]
scr = [90, 40, 80, 98]
```

```
dict = {'name': nme, 'degree': deg, 'score': scr}

df = pd.DataFrame(dict)
```

| | name | degree | score |
|---|---|---|---|
| 0 | aparna | MBA | 90 |
| 1 | pankaj | BCA | 40 |
| 2 | sudhir | M.Tech | 80 |
| 3 | Geeku | MBA | 98 |

```
df.to_csv('file1.csv')
```

Any questions so far?
Any comments?

Lambton College
School of Computer Studies

# AML-2203 Advanced Python AI and ML Tools

[Prepared By: Vahid Hadavi, PhD]

# Lecture2

2. Evaluate database and Python interactions

## Major Topics:

- 2.1 Use Python modules to connect to a relational database
- 2.2 Use Python modules to interact with a relational database
- 2.3 Explore CRUD operations using Python and relational database
- 2.4 Use Python modules to connect to a non-relational database
- 2.5 Explore CRUD operations using Python and non-relational database

# 2.1 Use Python modules to connect to a relational database

# Relational Databases

**What is SQL?**

- SQL stands for Structured Query Language.
- SQL database is a relational database.
- A relational database consists of two or more tables with columns and rows.
- Each row represents an entry, and each column sorts a very specific type of information, like a name, address, and phone number.

**What can SQL do?**

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database

# Python modules for SQL

- We first use create_engine to define a connection to a SQL database engine called SQLite.

```
from sqlalchemy import create_engine
```

```
database_connection = create_engine('sqlite:///sample.db')
```

- We can connect to other SQL databases such as MySQL.
- We use pandas' read_sql_query to query that database using SQL.
- We put the reults in a dataframe.

```
dataframe = pd.read_sql_query('SELECT * FROM data', database_connection)
```

# 2.2 Use Python modules to interact with a relational database

# Python modules for SQL

- We use pandas' read_sql_query to query that database using SQL.
- We put the results in a dataframe.

```
dataframe = pd.read_sql_query('SELECT * FROM data', database_connection)
```

- Next, we can manipulate the data in the dataframe using pandas methods.

# 2.3 Explore CRUD operations using Python and relational database

# CRUD

- CRUD operation:
  - C: Create
  - R: Read
  - U: Update
  - D: Delete.

# Create

- We need to first import sqlite3 and make connection to it
- We need to create a a cursor object by invoking the cursor() method on the created connection object.

```
import sqlite3
```

```
conn = sqlite3.connect('example.db')
```

```
cursor = conn.cursor()
```

- Statement:

```
CREATE TABLE database_name.table_name(
   column1 datatype PRIMARY KEY(one or more columns),
   column2 datatype,
   column3 datatype,
   .....
   columnN datatype
);
```

# Create

- Now we can execute the CREATE TABLE statement using the execute() method of the Cursor class

```
sql ='''CREATE TABLE EMPLOYEE(
   FIRST_NAME CHAR(20) NOT NULL,
   LAST_NAME CHAR(20),
   AGE INT,
   SEX CHAR(1),
   INCOME FLOAT
```

```
cursor.execute(sql)
```

```
conn.commit()
```

- It is important to call commit() method after every transaction that modifies data.

# Insert

- We can add new rows to an existing table of SQLite using the INSERT INTO statement.
- In this, we need to specify the name of the table, column names, and values (in the same order as column names).
- Statement:

```
INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)
VALUES (value1, value2, value3,...valueN);
```

# Insert

```
cursor.execute('''INSERT INTO EMPLOYEE(
   FIRST_NAME, LAST_NAME, AGE, SEX, INCOME) VALUES
   ('Ramya', 'Rama Priya', 27, 'F', 9000)''')
```

```
cursor.execute('''INSERT INTO EMPLOYEE(
   FIRST_NAME, LAST_NAME, AGE, SEX, INCOME) VALUES
   ('Sharukh', 'Sheik', 25, 'M', 8300)''')
```

```
cursor.execute('''INSERT INTO EMPLOYEE(
   FIRST_NAME, LAST_NAME, AGE, SEX, INCOME) VALUES
   ('Sarmista', 'Sharma', 26, 'F', 10000)''')
```

- We have to use commit() method after exciting the insert statements.

# Read

- We can retrieve the contents of an existing table using the SELECT statement.
- At this statement, we need to specify the name of the table and, it returns its contents in tabular format which is known as result set.
- Statement:

```
SELECT column1, column2, columnN FROM table_name;
```

- The sqlite3.Cursor class provides three methods namely fetchall(), fetchmany() and, fetchone() to fetch the selected rows:
  - The fetchall() method retrieves all the rows in the result set of a query and returns them as list of tuples. (If we execute this after retrieving few rows it returns the remaining ones).
  - The fetchone() method fetches the next row in the result of a query and returns it as a tuple
  - The fetchmany() method is similar to the fetchone() but, it retrieves the next set of rows in the result set of a query, instead of a single row.

# Read

```
cursor.execute('''SELECT * from EMPLOYEE''')
```

```
result = cursor.fetchall();
print(result)
```

```
[('Vinay', 'Battacharya', 20, 'M', 6000.0),
   ('Sharukh', 'Sheik', 25, 'M', 8300.0),
   ('Sarmista', 'Sharma', 26, 'F', 10000.0),
   ('Tripthi', 'Mishra', 24, 'F', 6000.0)
]
```

# Update

- UPDATE Operation on any database implies modifying the values of one or more records of a table, which are already available in the database.
- We can update the values of existing records in SQLite using the UPDATE statement.
- To update specific rows, you need to use the WHERE clause along with it
- Statement:

```
UPDATE table_name
SET column1 = value1, column2 = value2...., columnN = valueN
WHERE [condition];
```

```
sql = '''UPDATE EMPLOYEE SET AGE=AGE+1 WHERE SEX = 'M' '''
cursor.execute(sql)
```

# Delete

- To delete records from a SQLite table, you need to use the DELETE FROM statement.
- To remove specific records, you need to use WHERE clause along with it.
- Statement:

```
DELETE FROM table_name [WHERE Clause]
```

```
cursor.execute('''DELETE FROM EMPLOYEE WHERE AGE > 25''')
```

# 2.4 Use Python modules to connect to a non-relational database

# Non-Relational Databases

**What is NoSQL?**

- NoSQL stands for non-Structured Query Language.
- NoSQL database is a non-relational database.
- They are document databases, key-value databases, wide-column stores, and graph databases.
- NoSQL databases can store relationship data—they just store it differently than relational databases do.

**What can NoSQL do?**

- NoSQL can execute queries against a database
- NoSQL can retrieve data from a database
- NoSQL can insert records in a database
- NoSQL can update records in a database
- NoSQL can delete records from a database
- NoSQL can create new databases

# Python modules for NoSQL

- In order to connect to MongoDB, python uses a library known as pymongo.
- It needs to be installed using pip.
- After importing we can choose a client and connect to a database on it.

```
from pymongo import MongoClient
```

```
client = MongoClient()
```

```
db=client.test
```

- Next, we can have access to different collections on the connected database.

```
employee = db.employee
```

# 2.5 Explore CRUD operations using Python and non-relational database

# Insert

- To insert a record, or document as it is called in MongoDB, into a collection, we use the insert_one() method.
- The first parameter of the insert_one() method is a dictionary containing the name(s) and value(s) of each field in the document we want to insert.

```
employee_details = {
    'Name': 'Raj Kumar',
    'Address': 'Sears Streer, NZ',
    'Age': '42'
}
```

```
result = employee.insert_one(employee_details)
```

- To insert multiple documents into a collection in MongoDB, we use the insert_many() method.
- The first parameter of the insert_many() method is a list containing dictionaries with the data you want to insert

# Update

- You can update a record, or document as it is called in MongoDB, by using the update_one() method.
- The first parameter of the update_one() method is a query object defining which document to update (starting with $set).
- The second parameter is an object defining the new values of the document.

```
db.employee.update_one(
        {"Age":'42'},
        {
        "$set": {
            "Name":"Srinidhi",
            "Age":'35',
            "Address":"New Omsk, WC"
        }
        }
    )
```

# Delete

- To delete one document, we use the delete_one() method.
- The first parameter of the delete_one() method is a query object defining which document to delete.

```
db.employee.delete_one({"Age":'35'})
```

Any questions so far?
Any comments?

Lambton College
School of Computer Studies

# AML-2203 Advanced Python AI and ML Tools

[Prepared By: Vahid Hadavi, PhD]

# Lecture3

3. Evaluate vectors, matrices, and arrays in Python (Recommended Module: numpy)

## Major Topics:

- 3.1 Use Python modules to work with vectors
- 3.2 Use Python modules to work with matrices
- 3.3 Use Python modules to interact with elements of matrices
- 3.4 Use Python modules to interact with elements of arrays
- 3.5 Use Python modules to reshape arrays

# 3.1 Use Python modules to work with vectors

# Vectors

- We need to import numpy.

```
import numpy as np
```

- To create vector, we create one dimensional array.

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

- Selecting element in vectors.

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

```
vector[2]
```

```
3
```

# 3.2 Use Python modules to work with matrices

# Matrices

- We need to import numpy.

```
import numpy as np
```

- To create matrix, we create two-dimensional array (nested lists)

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

- Selecting element in matrices.

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

```
matrix[1,1]

5
```

# 3.3 Use Python modules to interact with elements of matrices

# Describing a matrix

- Describing the shape, size, and dimension of the matrix.

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

```
matrix.shape
```

```
(3, 4)
```

```
matrix.size
```

```
12
```

```
matrix.ndim
```

```
2
```

- Dimension of a matrix, vector and scalar is 2,1 and 0, respectively.

# Dot product, adding and subtrng

- Using dot method on numpy to calculate dot product.

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

```
np.dot(vector_a, vector_b)

32
```

- Using add and subtract methods on numpy.

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

```
matrix_b = np.array([[1, 3, 1],
                     [1, 3, 1],
                     [1, 3, 8]])
```

```
np.add(matrix_a, matrix_b)

array([[ 2,  4,  2],
       [ 2,  4,  2],
       [ 2,  4, 10]])
```

```
np.subtract(matrix_a, matrix_b)

array([[ 0, -2,  0],
       [ 0, -2,  0],
       [ 0, -2, -6]])
```

# 3.4 Use Python modules to interact with elements of arrays

# Applying Operations to Elements

- Directly manipulating elements.

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

```
matrix + 100

array([[101, 102, 103],
       [104, 105, 106],
       [107, 108, 109]])
```

- Using vectorize method to apply pre-defined functions.

```
add_100 = lambda i: i + 100
```

```
vectorized_add_100 = np.vectorize(add_100)
```

```
array([[101, 102, 103],
       [104, 105, 106],
       [107, 108, 109]])
```

# Finding Maximum and Minimum

- Using max and min methods.

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

```
np.max(matrix)

9
```

```
np.min(matrix)

1
```

- Using the axis parameter to apply the operation along a certain axis.

```
np.max(matrix, axis=0)

array([7, 8, 9])
```

```
np.max(matrix, axis=1)

array([3, 6, 9])
```

# Calculating statistical values

- Using mean, var and std methods on numpy.

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

```
np.mean(matrix)

5.0
```

```
np.var(matrix)

6.666666666666667
```

```
np.std(matrix)

2.5819888974716112
```

- Using the axis parameter to apply the operation along a certain axis.

```
np.mean(matrix, axis=0)

array([ 4.,  5.,  6.])
```

# Generating random values

- numPy offers a wide variety of means to generate random numbers.
- Using randint on numpy to generate random integers in a certain range.

```
np.random.randint(0, 11, 3)

array([3, 7, 9])
```

- Using random.normal on numpy to draw numbers from a normal distribution with mean 0 and standard deviation of 1.

```
np.random.normal(0.0, 1.0, 3)

array([-1.42232584,  1.52006949, -0.29139398])
```

# Generating random values

- Using random.logistic on numpy to draw numbers from a logistic distribution with mean 0 and scale of 1.

```
np.random.logistic(0.0, 1.0, 3)

array([-0.98118713, -0.08939902,  1.46416405])
```

- Using random.uniform on numpy  to draw numbers greater than or equal to a and less than b.

```
np.random.uniform(1.0, 2.0, 3)

array([ 1.47997717,  1.3927848 ,  1.83607876])
```

- We set the seed value to have repeatable outputs from random methods.

```
np.random.seed(0)
```

# 3.5 Use Python modules to reshape arrays

# Reshaping arrays

- Using reshape method on numpy.
- Size of the array before and after reshaping must match

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

```
matrix.reshape(2, 6)

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

- One useful argument in reshape is -1, which effectively means “as many as needed,”.

```
matrix.reshape(1, -1)

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

Any questions so far?
Any comments?

Lambton College
School of Computer Studies

# AML-2203 Advanced Python AI and ML Tools

[Prepared By: Vahid Hadavi, PhD]

# Lecture4

4. Evaluate Python features for data transformation (Recommended Module: pandas)

## Major Topics:

- 4.1 Discuss the importance of data transformation
- 4.2 Use Python modules to work with data frames
- 4.3 Navigate through data frames and parse individual or group elements
- 4.4 Use Python modules to adjust data frames
- 4.5 Use Python modules to apply a function over all elements in a column
- 4.6 Discuss various interactions with data and data frames in Python

# 4.1 Discuss the importance of data transformation

# Data transformation

- Data transformation (wrangling) is a key component of any data science workflow.
- Wrangling is a process where one transforms “raw” data for making it more suitable for analysis.
- It is usually the longest and most important step in a data science project.
- The most common data structure used to “wrangle” data is the data frame

# 4.2 Use Python modules to work with data frames

# Manipulating Data

- We use the Titanic dataset on Kaggle.

```
# Load library
import pandas as pd

# Create URL
url = 'https://tinyurl.com/titanic-csv'

# Load data
dataframe = pd.read_csv(url)
```

- Using head() or tail() to check the first or last 5 (default) rows of the data frame

```
dataframe.head(2)
```

| | Name | PClass | Age | Sex | Survived | SexCode |
|---|---|---|---|---|---|---|
| **0** | Allen, Miss Elisabeth Walton | 1st | 29.0 | female | 1 | 1 |
| **1** | Allison, Miss Helen Loraine | 1st | 2.0 | female | 0 | 1 |

# Manipulating Data

- Using .shape to check the dimensions of the dataframe.

```
dataframe.shape
(1313, 6)
```

- Using describe() to get descriptive statistics for any numeric columns.

| | Age | Survived | SexCode |
|---|---|---|---|
| **count** | 756.000000 | 1313.000000 | 1313.000000 |
| **mean** | 30.397989 | 0.342727 | 0.351866 |
| **std** | 14.259049 | 0.474802 | 0.477734 |
| **min** | 0.170000 | 0.000000 | 0.000000 |
| **25%** | 21.000000 | 0.000000 | 0.000000 |
| **50%** | 28.000000 | 0.000000 | 0.000000 |
| **75%** | 39.000000 | 1.000000 | 1.000000 |
| **max** | 71.000000 | 1.000000 | 1.000000 |

# 4.3 Navigate through data frames and parse individual or group elements

# Navigating dataframe

- Navigating dataframe using .iloc and .loc.

```
dataframe.iloc[1:4]
```

| | Name | PClass | Age | Sex | Survived | SexCode |
|---|---|---|---|---|---|---|
| 1 | Allison, Miss Helen Loraine | 1st | 2.0 | female | 0 | 1 |
| 2 | Allison, Mr Hudson Joshua Creighton | 1st | 30.0 | male | 0 | 0 |
| 3 | Allison, Mrs Hudson JC (Bessie Waldo Daniels) | 1st | 25.0 | female | 0 | 1 |

- loc is useful when the index of the DataFrame is a label (e.g., a string).

```
dataframe = dataframe.set_index(dataframe['Name'])
```

```
dataframe.loc['Allen, Miss Elisabeth Walton']

Name        Allen, Miss Elisabeth Walton
PClass                               1st
Age                                   29
Sex                             female
Survived                               1
SexCode                                1
```

# Navigating dataframe

- Selecting Rows Based on Conditionals.

```
dataframe[dataframe['Sex'] == 'female'].head(2)
```

| | Name | PClass | Age | Sex | Survived | SexCode |
|---|---|---|---|---|---|---|
| **0** | Allen, Miss Elisabeth Walton | 1st | 29.0 | female | 1 | 1 |
| **1** | Allison, Miss Helen Loraine | 1st | 2.0 | female | 0 | 1 |

- Multiple conditions.

```
dataframe[(dataframe['Sex'] == 'female') & (dataframe['Age'] >= 65)]
```

| | Name | PClass | Age | Sex | Survived | SexCode |
|---|---|---|---|---|---|---|
| **73** | Crosby, Mrs Edward Gifford (Catherine Elizabet... | 1st | 69.0 | female | 1 | 1 |

# 4.4 Use Python modules to adjust data frames

# Adjusting dataframe

- Using replace() to replace values.

```
dataframe['Sex'].replace(["female", "male"], ["Woman", "Man"]).head(5)

0    Woman
1    Woman
2      Man
3    Woman
4      Man
```

```
dataframe.replace(1, "One").head(2)
```

| | Name | PClass | Age | Sex | Survived | SexCode |
|---|---|---|---|---|---|---|
| **0** | Allen, Miss Elisabeth Walton | 1st | 29 | female | One | One |
| **1** | Allison, Miss Helen Loraine | 1st | 2 | female | 0 | One |

# Adjusting dataframe

- Using rename() to rename column names.

```
dataframe.rename(columns={'PClass': 'Passenger Class'}).head(2)
```

| | Name | Passenger Class | Age | Sex | Survived | SexCode |
|---|---|---|---|---|---|---|
| 0 | Allen, Miss Elisabeth Walton | 1st | 29.0 | female | 1 | 1 |
| 1 | Allison, Miss Helen Loraine | 1st | 2.0 | female | 0 | 1 |

- Finding the Minimum, Maximum, Sum, Average, and Count.

```
print('Maximum:', dataframe['Age'].max())
print('Minimum:', dataframe['Age'].min())
print('Mean:', dataframe['Age'].mean())
print('Sum:', dataframe['Age'].sum())
print('Count:', dataframe['Age'].count())

Maximum: 71.0
Minimum: 0.17
Mean: 30.397989417989415
```

# Adjusting dataframe

- Using isnull() to find missing values.

```
dataframe[dataframe['Age'].isnull()].head(2)
```

| | Name | PClass | Age | Sex | Survived | SexCode |
|---|---|---|---|---|---|---|
| **12** | Aubert, Mrs Leontine Pauline | 1st | NaN | female | 1 | 1 |
| **13** | Barkworth, Mr Algernon H | 1st | NaN | male | 1 | 0 |

- Using replace() to replace missing values. Numpy must be imported.

```
dataframe['Sex'] = dataframe['Sex'].replace('male', np.nan)
```

- Pandas fillna() function can be also used to address missing values in a dataframe.

# Adjusting dataframe

- Using drop() to delete columns and rows.
- Axis argument must be provided.
- axis = 1 for columns and axis = 0 for rows.

```
dataframe.drop('Age', axis=1).head(2)
```

| | Name | PClass | Sex | Survived | SexCode |
|---|---|---|---|---|---|
| 0 | Allen, Miss Elisabeth Walton | 1st | female | 1 | 1 |
| 1 | Allison, Miss Helen Loraine | 1st | female | 0 | 1 |

- For deleting rows, boolean conditions can be also used.

```
dataframe[dataframe['Sex'] != 'male'].head(2)
```

| | Name | PClass | Age | Sex | Survived | SexCode |
|---|---|---|---|---|---|---|
| 0 | Allen, Miss Elisabeth Walton | 1st | 29.0 | female | 1 | 1 |
| 1 | Allison, Miss Helen Loraine | 1st | 2.0 | female | 0 | 1 |

# Unique values

- Using unique() to find unique values.

```
dataframe['Sex'].unique()

array(['female', 'male'], dtype=object)
```

- Using value_counts() to show unique values with the number of times each value appears.

```
dataframe['Sex'].value_counts()
```

```
male      851
female    462
Name: Sex, dtype: int64
```

# Grouping

- Using groupby () to group individual rows according to some shared value.

```
dataframe.groupby('Sex').mean()
```

| Sex | Age | Survived | SexCode |
|---|---|---|---|
| **female** | 29.396424 | 0.666667 | 1.0 |
| **male** | 31.014338 | 0.166863 | 0.0 |

- For grouping to be useful, we need to group by something and then apply a function to each of those groups.

```
dataframe.groupby('Survived')['Name'].count()

Survived
0    863
1    450
Name: Name, dtype: int64
```

# Looping over dataframe

- Looping over a column.

```
for name in dataframe['Name'][0:2]:
    print(name.upper())
```

```
ALLEN, MISS ELISABETH WALTON
ALLISON, MISS HELEN LORAINE
```

```
[name.upper() for name in dataframe['Name'][0:2]]
```

```
['ALLEN, MISS ELISABETH WALTON', 'ALLISON, MISS HELEN LORAINE']
```

# 4.5 Use Python modules to apply a function over all elements in a column

# Applying a function on a column

- Using apply to apply some function over all elements in a column .

```
def uppercase(x):
    return x.upper()
```

```
dataframe['Name'].apply(uppercase)[0:2]

0    ALLEN, MISS ELISABETH WALTON
1     ALLISON, MISS HELEN LORAINE
Name: Name, dtype: object
```

# Applying a function on a group

- Using apply to apply some function to a group of rows.

```
dataframe.groupby('Sex').apply(lambda x: x.count())
```

| Sex | Name | PClass | Age | Sex | Survived | SexCode |
|---|---|---|---|---|---|---|
| **female** | 462 | 462 | 288 | 462 | 462 | 462 |
| **male** | 851 | 851 | 468 | 851 | 851 | 851 |

# 4.6 Discuss various interactions with data and data frames in Python

# Concatenating

- Using concat to concatenate two DataFrames.
- We can set axis=0 to concatenate along the row axis, and axis=1 to concatenate along the column axis

```
data_a = {'id': ['1', '2', '3'],
          'first': ['Alex', 'Amy', 'Allen'],
          'last': ['Anderson', 'Ackerman', 'Ali']}
dataframe_a = pd.DataFrame(data_a, columns = ['id', 'first', 'last'])
```

```
data_b = {'id': ['4', '5', '6'],
          'first': ['Billy', 'Brian', 'Bran'],
          'last': ['Bonder', 'Black', 'Balwner']}
dataframe_b = pd.DataFrame(data_b, columns = ['id', 'first', 'last'])
```

# Concatenating

```
pd.concat([dataframe_a, dataframe_b], axis=0)
```

| | id | first | last |
|---|---|---|---|
| **0** | 1 | Alex | Anderson |
| **1** | 2 | Amy | Ackerman |
| **2** | 3 | Allen | Ali |
| **0** | 4 | Billy | Bonder |
| **1** | 5 | Brian | Black |
| **2** | 6 | Bran | Balwner |

```
pd.concat([dataframe_a, dataframe_b], axis=1)
```

| | id | first | last | id | first | last |
|---|---|---|---|---|---|---|
| **0** | 1 | Alex | Anderson | 4 | Billy | Bonder |
| **1** | 2 | Amy | Ackerman | 5 | Brian | Black |
| **2** | 3 | Allen | Ali | 6 | Bran | Balwner |

# Merging

- Using merge to merge two DataFrames.
- We can use ‘how’ parameter to specify the type of merge; outer, inner, left and right.
- The default value for how is inner.

```
employee_data = {'employee_id': ['1', '2', '3', '4'],
                 'name': ['Amy Jones', 'Allen Keys', 'Alice Bees',
                 'Tim Horton']}
dataframe_employees = pd.DataFrame(employee_data, columns = ['employee_id',
                                                             'name'])
```

```
sales_data = {'employee_id': ['3', '4', '5', '6'],
              'total_sales': [23456, 2512, 2345, 1455]}
dataframe_sales = pd.DataFrame(sales_data, columns = ['employee_id',
                                                      'total_sales'])
```

# Merging

- We use ‘on’ parameter to specify the shared column if that column has a same name in both dataframes.
- If the shared column has two different names, we use ‘left_on’ , ‘right_on’ parameters

```
pd.merge(dataframe_employees, dataframe_sales, on='employee_id')
```

| | employee_id | name | total_sales |
|---|---|---|---|
| 0 | 3 | Alice Bees | 23456 |
| 1 | 4 | Tim Horton | 2512 |

```
pd.merge(dataframe_employees, dataframe_sales, on='employee_id', how='outer')
```

| | employee_id | name | total_sales |
|---|---|---|---|
| 0 | 1 | Amy Jones | NaN |
| 1 | 2 | Allen Keys | NaN |
| 2 | 3 | Alice Bees | 23456.0 |
| 3 | 4 | Tim Horton | 2512.0 |
| 4 | 5 | NaN | 2345.0 |
| 5 | 6 | NaN | 1455.0 |

# Merging

```
pd.merge(dataframe_employees,
         dataframe_sales,
         left_on='employee_id',
         right_on='employee_id')
```

| | employee_id | name | total_sales |
|---|---|---|---|
| 0 | 3 | Alice Bees | 23456 |
| 1 | 4 | Tim Horton | 2512 |

- If instead of merging on two columns we want to merge on the indexes of each dataframe, we can use right_index=True and left_index=True.

Any questions so far?
Any comments?

Lambton College
School of Computer Studies

# AML-2203 Advanced Python AI and ML Tools

[Prepared By: Vahid Hadavi, PhD]

# Lecture5

**5**. Analyze various ways to deal with numerical and categorical data in Python (Recommended Module: pandas, scikit-learn)

## Major Topics:

- 5.1 Discuss the differences between numerical and categorical data
- 5.2 Use Python modules to rescale a feature in numerical data sets
- 5.3 Discuss outliers and various ways to detect them
- 5.4 Use Python modules to discretize a feature
- 5.5 Use Python modules to cluster observations
- 5.6 Use Python modules to address missing values
- 5.7 Use Python modules to encode nominal categorical values
- 5.8 Use Python modules to encode ordinal categorical values

## 5.1 Discuss the differences between numerical and categorical data

# Categorical vs Numerical

**Categorical data**

- Categorical data is a collection of information that is divided into groups.
  - Examples are race, sex, age group, and educational level.
- They can  have a natural rank order.
  - Example is very bad, bad, good, very good.
- The distances between the categories are uneven or unknown.

**Numerical data**

- Numerical data is the measurement of something
  - Examples are size, monthly sales, or student scores.
- The natural way to represent these quantities is numerically.
- It is further divided into two subsets: discrete and continuous.
- Discrete Data can only take certain values.
  - Number of children
- Continuous Data can take any value (within a range).
  - A person’s height.

# 5.2 Use Python modules to rescale a feature in numerical data sets

# Feature scaling

- Feature Scaling is one of the important pre-processing.
- When the range of values are very distinct in each column, we need to scale them to the common level.
  - Example; we have different features in different units, e.g. kilometers, meters.
- Some algorithms are more affected by feature scaling.
  - Distance based algorithm, e.g. KNN and SVM.
- Tree-based algorithms, on the other hand, are fairly insensitive to the scale of the features.
- Two most used feature scaling techniques are:
  - Normalization
  - Standardization.

# Normalization

- Normalization is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1.

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

- $X_{max}$ and $X_{min}$ are the maximum and the minimum values of the feature respectively.
- Using scikit-learn's MinMaxScaler to rescale a feature array.
- We can specify the range using feature_range argument.

# Normalization

```
import numpy as np
from sklearn import preprocessing
```

```
feature = np.array([[-500.5],
                    [-100.1],
                    [0],
                    [100.1],
                    [900.9]])
```

```
minmax_scale = preprocessing.MinMaxScaler(feature_range=(0, 1))
```

```
scaled_feature = minmax_scale.fit_transform(feature)
```

```
array([[ 0.        ],
       [ 0.28571429],
       [ 0.35714286],
       [ 0.42857143],
       [ 1.        ]])
```

# Standardization

- Standardization is a scaling technique in which values are centered zero mean with the standard deviation of 1.

$$X' = \frac{X - \mu}{\sigma}$$

- $\mu$ is the mean of the feature values and $\sigma$ is the standard deviation of the feature values.
- Standardization is preferred over normalization is most cases.
- It is recommended to use standardization, unless there is a specific reason to use normalization (neural networks for example)
- Using scikit-learn's StandardScale to perform standardization.

# Standardization

```
import numpy as np
from sklearn import preprocessing
```

```
x = np.array([[-1000.1],
              [-200.2],
              [500.5],
              [600.6],
              [9000.9]])
```

```
scaler = preprocessing.StandardScaler()
```

```
standardized = scaler.fit_transform(x)
```

```
array([[-0.76058269],
       [-0.54177196],
       [-0.35009716],
       [-0.32271504],
       [ 1.97516685]])
```

## 5.3 Discuss outliers and various ways to detect them

# Outliers

- Outliers are extreme values that deviate from other observations on data.
- They may indicate a variability in a measurement, experimental errors or a novelty.
- If there are outliers, standardization might not be appropriate because the mean and variance might be highly influenced by the outliers.
- Detecting outliers is unfortunately more of an art than a science.
- There are different methods to detect outliers:
  - Interquartile Range (IQR)
  - Visualization techniques

# Outliers detection-example

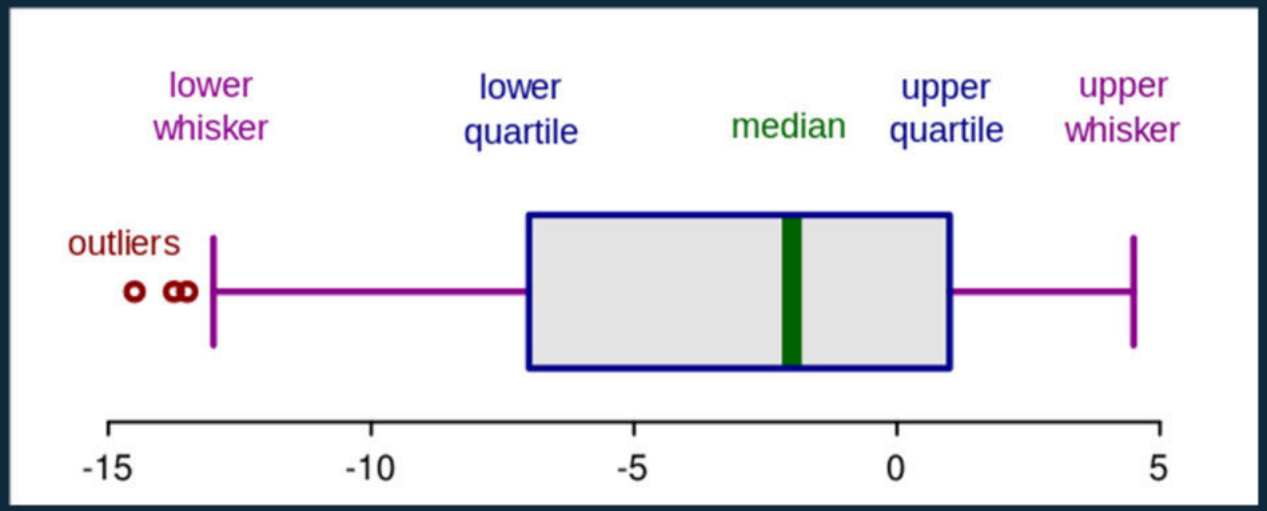
- We will be using a fictitious dataset of loan applications containing 600 observations and 6 variables:
  - Income, loan amount, term months, credit score, age and approval status.

```
df.describe()
```

| | Income | Loan_amount | Term_months | Credit_score | approval_status | Age |
|---|---|---|---|---|---|---|
| count | 600.000000 | 600.000000 | 600.00000 | 600.000000 | 600.000000 | 600.000000 |
| mean | 7210.720000 | 161.571667 | 367.10000 | 0.788333 | 0.686667 | 51.766667 |
| std | 8224.445086 | 93.467598 | 63.40892 | 0.408831 | 0.464236 | 21.240704 |
| min | 200.000000 | 10.000000 | 36.00000 | 0.000000 | 0.000000 | 0.000000 |
| 25% | 3832.500000 | 111.000000 | 384.00000 | 1.000000 | 0.000000 | 36.000000 |
| 50% | 5075.000000 | 140.000000 | 384.00000 | 1.000000 | 1.000000 | 51.000000 |
| 75% | 7641.500000 | 180.500000 | 384.00000 | 1.000000 | 1.000000 | 64.000000 |
| max | 108000.000000 | 778.000000 | 504.00000 | 1.000000 | 1.000000 | 200.000000 |

# Outliers with Box Plots and IQR

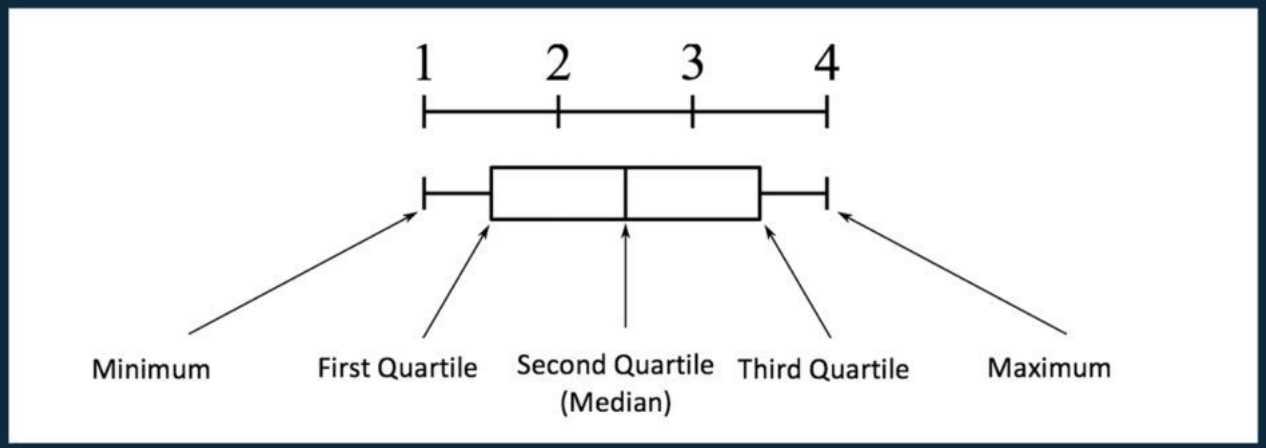
Box plots are a graphical depiction of numerical data through their quantiles. It is a very simple but effective way to visualize outliers. Think about the lower and upper whiskers as the boundaries of the data distribution. Any data points that show above or below the whiskers, can be considered outliers or anomalous.



# Outliers with Box Plots and IQR

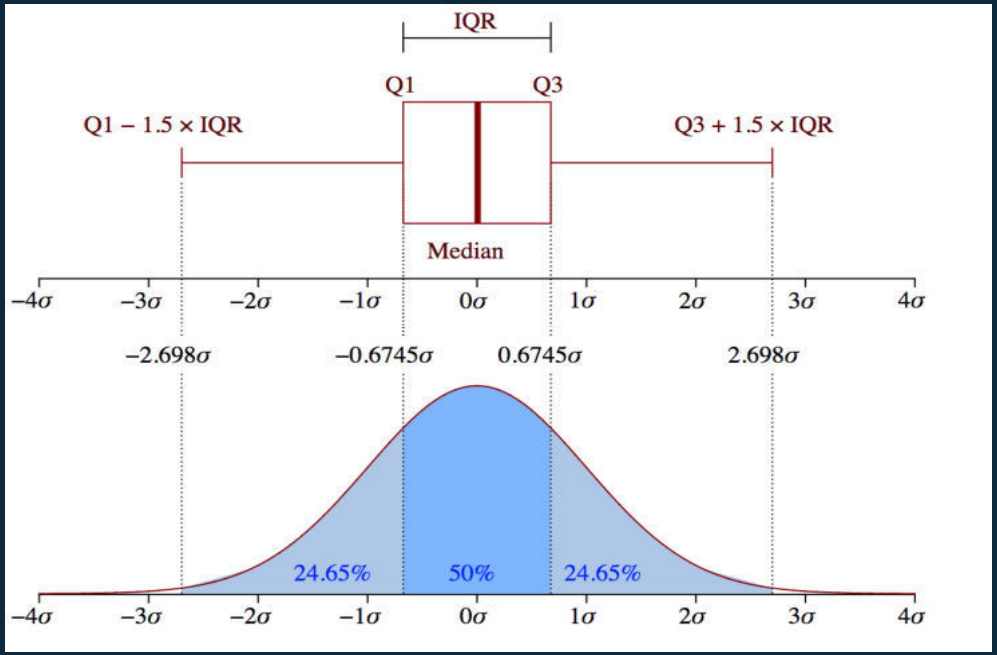
Boxplot Anatomy:

The concept of the Interquartile Range (IQR) is used to build the boxplot graphs. IQR is a concept in statistics that is used to measure the statistical dispersion and data variability by dividing the dataset into quartiles. In simple words, any dataset or any set of observations is divided into four defined intervals based upon the values of the data and how they compare to the entire dataset. A quartile is what divides the data into three points and four intervals.



# Outliers with Box Plots and IQR

Interquartile Range (IQR) is important because it is used to define the outliers. It is the difference between the third quartile and the first quartile (IQR = Q3 -Q1). Outliers in this case are defined as the observations that are below (Q1 – 1.5x IQR) or boxplot lower whisker or above (Q3 + 1.5x IQR) or boxplot upper whisker.



# Outliers detection-IQR

- The interquartile range (IQR) is a measure of statistical dispersion.
- It is calculated as the difference between the 75th and 25th percentiles.

```
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

- The IQR scores can be used to detect outliers for each feature.

```
Income            3809.0
Loan_amount         69.5
Term_months          0.0
Credit_score         0.0
approval_status      1.0
Age                 28.0
dtype: float64
```

# Outliers detection-IQR

- The data points which fall below Q1 – 1.5 IQR or above Q3 + 1.5 IQR are outliers.

```
print(df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR))
```

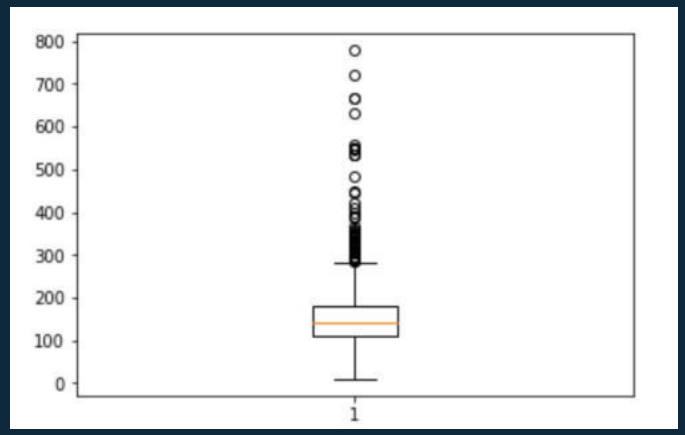
- The presence of outliers (Ture values) for the first 15 observations.

| | Income | Loan_amount | Term_months | Credit_score | approval_status | Age |
|---|---|---|---|---|---|---|
| 0 | False | False | False | False | False | False |
| 1 | False | False | True | False | False | False |
| 2 | False | False | False | False | False | False |
| 3 | False | False | False | False | False | False |
| 4 | False | False | False | False | False | False |
| 5 | False | False | False | False | False | False |
| 6 | False | False | False | False | False | False |
| 7 | False | False | False | False | False | False |
| 8 | False | False | False | False | False | False |
| 9 | False | False | False | False | False | False |
| 10 | False | False | False | False | False | False |
| 11 | False | False | False | False | False | False |
| 12 | False | False | True | True | False | False |
| 13 | False | False | False | False | False | False |
| 14 | False | False | False | False | False | False |

# Outliers detection-Visualization

- Using box plots for each feature.

```
plt.boxplot(df["Loan_amount"])
plt.show()
```

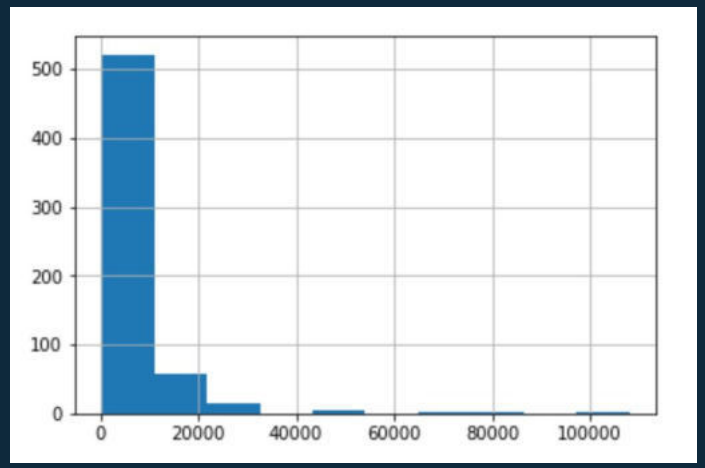



- The circles indicate the outliers

# Outliers detection-Visualization

- Using histograms for each feature.

```
df.Income.hist()
```

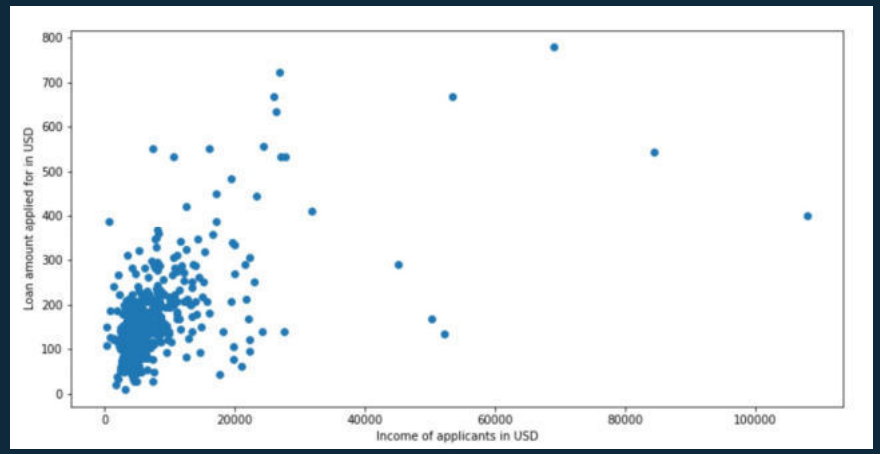



- The chart shows that the distribution is right-skewed, and there are extreme higher values at the right of the histogram.

# Outliers detection-Visualization

- Using scatterplots for each feature.

```
fig, ax = plt.subplots(figsize=(12,6))
ax.scatter(df['Income'], df['Loan_amount'])
ax.set_xlabel('Income of applicants in USD')
ax.set_ylabel('Loan amount applied for in USD')
plt.show()
```




- The chart indicates that most of the data points are clustered in the lower half of the plot.
- The points located to the extreme right of the x-axis or the y-axis indicate outliers.

# Outliers treatment

- Similar to detecting outliers, there is no hard-and-fast rule for handling them.
- We should consider what makes them an outlier:
  - Incorrect data, e.g. data from a broken sensor.
  - Genuine extreme values, e.g., a house [mansion] with 200 bathrooms
- In the second case, we should be more cautious in treating them.
- There are several techniques for outlier treatment:
  - Quantile-based Flooring and Capping
  - Trimming
  - Log Transformation

# Quantile-based Flooring and Capping

- We will do the flooring (e.g., the 10th percentile) for the lower values and capping (e.g., the 90th percentile) for the higher values.

```
print(df['Income'].quantile(0.10))
print(df['Income'].quantile(0.90))
```

```
2960.1
12681.0
```

```
df["Income"] = np.where(df["Income"] <2960.0, 2960.0,df['Income'])
df["Income"] = np.where(df["Income"] >12681.0, 12681.0,df['Income'])
```

# Trimming

- We completely remove data points that are outliers.

```
index = df[(df['Age'] >= 100)|(df['Age'] <= 18)].index
df.drop(index, inplace=True)
```

# Log transformation

- Transformation of the skewed variables may also help correct the distribution of the variables.
- These could be logarithmic, square root, or square transformations.
- The most common is the logarithmic transformation.

```
df["Log_Loanamt"] = df["Loan_amount"].map(lambda i: np.log(i) if i > 0 else
```

```
print(df['Loan_amount'].skew())
print(df['Log_Loanamt'].skew())
```

```
2.8146019248106815
-0.17792641310111373
```

# 5.4 Use Python modules to discretize a feature

# Discretization

- Discretization is the process through which we can transform continuous variables into a discrete form.
- We do this when we have reason to believe that a numerical feature should behave more like a categorical feature.
  - For example, a very little difference in the spending habits of 19 and 20-year-olds, but a significant difference between 20- and 21-year-olds.
- We do this by creating a set of contiguous intervals (or bins) that go across the range of our desired variable.
- Two methods cab used in python to do so;
  - scikit-learn's Binarizer for creating two bins
  - numPy's digitize for creating three or more bins

# scikit-learn's Binarizer

```
import numpy as np
from sklearn.preprocessing import Binarizer
```

```
age = np.array([[6],
                [12],
                [20],
                [36],
                [65]])
```

```
binarizer = Binarizer(18)
```

```
binarizer.fit_transform(age)

array([[0],
       [0],
       [1],
       [1],
       [1]])
```

# numPy's digitize

```
import numpy as np
```

```
age = np.array([[6],
                [12],
                [20],
                [36],
                [65]])
```

```
np.digitize(age, bins=[20,30,64])

array([[0],
       [0],
       [1],
       [2],
       [3]])
```

# 5.5 Use Python modules to cluster observations

# Clustering observations

- We can use clustering as a preprocessing step.
- We use unsupervised learning algorithms like k-means to cluster observations into groups.
- The end result is a categorical feature with similar observations being members of the same group.

# Clustering observations-example

```
import pandas as pd
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
```

```
clusterer = KMeans(3, random_state=0)
```

```
features, _ = make_blobs(n_samples = 50,
                         n_features = 2,
                         centers = 3,
                         random_state = 1)
```

```
clusterer.fit(features)
```

```
dataframe = pd.DataFrame(features, columns=["feature_1", "feature_2"])
```

```
dataframe["group"] = clusterer.predict(features)
```

```
dataframe.head(5)
```

| | feature_1 | feature_2 | group |
|---|---|---|---|
| 0 | −9.877554 | −3.336145 | 0 |
| 1 | −7.287210 | −8.353986 | 2 |
| 2 | −6.943061 | −7.023744 | 2 |
| 3 | −7.440167 | −8.791959 | 2 |
| 4 | −6.641388 | −8.075888 | 2 |

# 5.6 Use Python modules to address missing values

# Missing values

- Most machine learning algorithms cannot handle any missing values in the target and feature arrays.
- We cannot ignore missing values in our data and must address the issue during preprocessing.
- The simplest solution is to delete every observation that contains one or more missing values.
- Using dropna() method to drop any observation with missing values.

```
dataframe.dropna()
```

- We can use “inplace = True” to do the operation inplace in the dataframe.

# Imputing missing values

- There are two main strategies for replacing missing data with substitute values:
  - We can use machine learning to predict the values of the missing data
  - We can fill in all missing values with some average value

# Predicting missing values

- we can use machine learning to predict the values of the missing data.
- We treat the feature with missing values as a target vector and use the remaining subset of features to predict missing values.
- We can use a wide range of machine learning algorithms to impute values, a popular choice is KNN.

```
>>> import numpy as np
>>> from sklearn.impute import KNNImputer
>>> X = [[1, 2, np.nan], [3, 4, 3], [np.nan, 6, 5], [8, 8, 7]]
>>> imputer = KNNImputer(n_neighbors=2)
>>> imputer.fit_transform(X)
array([[1. , 2. , 4. ],
       [3. , 4. , 3. ],
       [5.5, 6. , 5. ],
       [8. , 8. , 7. ]])
```

# Filling missing values

- we can use scikit-learn's Imputer module to fill in missing values with the feature's mean, median, or most frequent value.
- We typically get worse results than KNN

```
>>> import numpy as np
>>> from sklearn.impute import SimpleImputer
>>> imp_mean = SimpleImputer(missing_values=np.nan, strategy='mean')
>>> imp_mean.fit([[7, 2, 3], [4, np.nan, 6], [10, 5, 9]])
SimpleImputer()
>>> X = [[np.nan, 2, 3], [4, np.nan, 6], [10, np.nan, 9]]
>>> print(imp_mean.transform(X))
[[ 7.   2.   3. ]
 [ 4.   3.5  6. ]
 [10.   3.5  9. ]]
```

# 5.7 Use Python modules to encode nominal categorical values

# Nominal variables

- Categorical information is often represented in data as a vector or column of strings.
- Most machine learning algorithms require inputs be numerical values.
- Sets of categories with no intrinsic ordering is called nominal.
  - Blue, Red, Green
  - Man, Woman
- Using scikit-learn's LabelBinarizer to one-hot encode.
- It creates a binary feature for each class in the original feature.
- It is recommended to drop one of the one-hot encoded features in the resulting
- matrix to avoid linear dependence.

# Nominal variables

```
from sklearn import preprocessing
lb = preprocessing.LabelBinarizer()
lb.fit_transform(['yes', 'no', 'no', 'yes'])
```

```
array([[1],
       [0],
       [0],
       [1]])
```

- We can use the classes_ method to output the classes.

```
lb.classes_
```

```
array(['no', 'yes'],
```

# Nominal variables

- We can also use pandas to one-hot encode the feature.

```
import pandas as pd
df = pd.DataFrame({'X': ['a', 'b', 'a'], 'Y': ['B', 'A', 'C'],'Z': [1, 2, 3]})
df
```

| | X | Y | Z |
|---|---|---|---|
| **0** | a | B | 1 |
| **1** | b | A | 2 |
| **2** | a | C | 3 |

```
pd.get_dummies(df, prefix=['colX', 'colY'])
```

| | Z | colX_a | colX_b | colY_A | colY_B | colY_C |
|---|---|---|---|---|---|---|
| **0** | 1 | 1 | 0 | 0 | 1 | 0 |
| **1** | 2 | 0 | 1 | 1 | 0 | 0 |
| **2** | 3 | 1 | 0 | 0 | 0 | 1 |

# 5.8 Use Python modules to encode ordinal categorical values

# Ordinal variables

- Ordinal variables have natural ordering.
  - Strongly Agree
  - Agree
  - Neutral
  - Disagree
  - Strongly Disagree
- We need to transform the ordinal classes into numerical values that maintain the notion of ordering.
- The most common approach is to create a dictionary that maps the string label of the class to a number and then apply that map to the feature.

# Ordinal variables

```
dataframe = pd.DataFrame({"Score": ["Low",
                                    "Low",
                                    "Medium",
                                    "Medium",
                                    "High",
                                    "Barely More Than Medium"]})
```

```
scale_mapper = {"Low":1,
                "Medium":2,
                "Barely More Than Medium": 3,
                "High":4}
```

```
dataframe["Score"].replace(scale_mapper)

0    1
1    1
2    2
3    2
4    4
5    3
Name: Score, dtype: int64
```

- scikit-learn's LabelEncoder can also be used, but using dictionaries gives us more control, as shown in the example.

Any questions so far?
Any comments?

**Lambton College**

School of Computer Studies

# AML-2203 Advanced Python AI and ML Tools

[Prepared By: Vahid Hadavi, PhD]

# Lecture6

**6**. Evaluate various ways to deal with textual data in Python (Recommended Module: NLTK, scikit-learn)

## Major Topics:

- 6.1 Discuss textual data and their characteristics
- 6.2 Use Python modules to clean textual data
- 6.3 Use Python modules to parse and clean HTML documents
- 6.4 Use Python modules to remove punctuation and stopwords
- 6.5 Use Python modules for textual data tokenization
- 6.6 Discuss the process of text mining and stemming
- 6.7 Discuss and use Python modules to encode textual data as bag of words
- 6.8 Discuss the concept of TF/IDF scoring in Python

# 6.1 Discuss textual data and their characteristics

# Textual data

- Text data is unstructured type data, since it does not have a predefined format.
  - Content of book
  - Tweet
- It is very complex to handle.
- Python has packages such as nlk or spacy to analyze text data.

# 6.2 Use Python modules to clean textual data

# Cleaning text

- Most text data needs to be cleaned before we can use it to build features.
- Most basic text cleaning tasks can be performed using Python's standard string operations.

```
text_data = ["   Interrobang. By Aishwarya Henriette     ",
             "Parking And Going. By Karl Gautier",
             "    Today Is The night. By Jarek Prakash   "]
```

- Striping white spaces:

```
strip_whitespace = [string.strip() for string in text_data]
```

```
strip_whitespace

['Interrobang. By Aishwarya Henriette',
 'Parking And Going. By Karl Gautier',
 'Today Is The night. By Jarek Prakash']
```

# Cleaning text

- Removing periods:

```
remove_periods = [string.replace(".", "") for string in strip_whitespace]
```

```
remove_periods

['Interrobang By Aishwarya Henriette',
 'Parking And Going By Karl Gautier',
 'Today Is The night By Jarek Prakash']
```

- We also create and apply a custom transformation function:

```
def capitalizer(string: str) -> str:
    return string.upper()
```

```
[capitalizer(string) for string in remove_periods]

['INTERROBANG BY AISHWARYA HENRIETTE',
 'PARKING AND GOING BY KARL GAUTIER',
 'TODAY IS THE NIGHT BY JAREK PRAKASH']
```

# Cleaning text

- We can use regular expression:

```
import re
```

```
def replace_letters_with_X(string: str) -> str:
    return re.sub(r"[a-zA-Z]", "X", string)
```

```
[replace_letters_with_X(string) for string in remove_periods

['XXXXXXXXXXX XX XXXXXXXXX XXXXXXXXX',
 'XXXXXXX XXX XXXXX XX XXXX XXXXXXX',
 'XXXXX XX XXX XXXXX XX XXXXX XXXXXXX']
```

# 6.3 Use Python modules to parse and clean HTML documents

# Beautiful soup

- Beautiful Soup is a powerful Python library designed for scraping HTML.
- We can use Beautiful soup to extract text data embedded in HTML .

# 6.4 Use Python modules to remove punctuation and stopwords

# Punctuation removal

- There are different ways to remove punctuations from a text.
- One of the best ones is to use regular expression.

```
text = "Hi! this is just an example, nothing else."
```

```
re.sub(r'[^\w\s]', '', text)
```

```
'Hi this is just an example nothing else'
```

# Stop Words

- Stop words are commonly-used word that are usually ignored because of their many occurrences.
- They do not add much meaning to a sentence.
- Most of these words are articles and prepositions, such as “the”, “a”, “in”, etc.
- These words can either end up taking too much space or eating up too much time.
- NLTK has a list of stop words in 16 different languages.

```
from nltk.corpus import stopwords
stopwords = stopwords.words('english')
```

- First 10 words in the stopwords list:

```
stopwords[:10]
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're"]
```

# Stop Words removal

```
text = 'A sentence is a textual unit consisting of words that are grammatically linked.'
```

- First, we tokenize the text.

```
from nltk.tokenize import word_tokenize
tokenized_word = word_tokenize(text)
tokenized_word
```

```
['A',
 'sentence',
 'is',
 'a',
 'textual',
 'unit',
 'consisting',
 'of',
 'words',
 'that',
 'are',
 'grammatically',
 'linked',
 '.']
```

# Stop Words removal

- Now we filter out the stopwords.

```
from nltk.corpus import stopwords
stopwords = stopwords.words('english')
filtered_tokens = []
for token in tokenized_word:
    if token not in stopwords:
        filtered_tokens.append(token)
filtered_tokens
```

```
['sentence',
 'textual',
 'unit',
 'consisting',
 'words',
 'grammatically',
 'linked',
 '.']
```

‘a’, ‘is’, ‘that’, ‘are’ and ‘of were removed.

# 6.5 Use Python modules for textual data tokenization

# Sentence splitting

- Sentence splitting (tokenization) is the process of splitting text into individual sentences.
- sent_tokenize on NLTK  is used to split text into sentences.
- It needs the text name and the language of the text (English as default) as input.

```
samplestring = 'Let’s make this our sample paragraph. It even knows that the period in Mr. Jones is not the end. Try it out!'
from nltk.tokenize import sent_tokenize
tokenized_sent = sent_tokenize(samplestring)
print(tokenized_sent)
```

```
['Let’s make this our sample paragraph.', 'It even knows that the period in Mr. Jones is not the end.', 'Try it out!']
```

- It recognizes that the “.” in Mr. is not a period ending a sentence.

# Word tokenizing

- A token is the smallest text unit a machine can process.
- Every chunk of text needs to be tokenized before you running natural language programs on it.
- Python split() can be used to tokenize a text.

```
msg = 'Hey everyone! The party starts in 10mins. Be there ASAP!'
print(msg.split())
```

```
['Hey', 'everyone!', 'The', 'party', 'starts', 'in', '10mins.', 'Be', 'there', 'ASAP!']
```

- Word_tokenize on NLTK can be used to tokenize a text.
- It needs the text name and the language of the text (English as default) as input.

```
from nltk.tokenize import word_tokenize
tokenized_word = word_tokenize(msg)
print(tokenized_word)
```

```
['Hey', 'everyone', '!', 'The', 'party', 'starts', 'in', '10mins', '.', 'Be', 'there', 'ASAP', '!']
```

- Their outputs are slightly different. What are the differences?

# 6.6 Discuss the process of text mining and stemming

# Stemming

- Stemming is a text normalization technique used to extract the base form of the words by removing affixes from them.
- Stemming algorithms work by cutting off the end or the beginning of the word, considering a list of common prefixes and suffixes that can be found in an inflected word.
  - Example: Jump-*ing*, Jump-*ed*, Jump-*s* → **Jump**
- There are different algorithms in python for stemming.
- The most common stemming algorithm in English is Porter stemmer.
- It needs a word or a list of tokens as input.

```
from nltk.stem import PorterStemmer
porter = PorterStemmer()
porter.stem('running')
```

```
'run'
```

- There are other algorithms such as Snowball Stemmer that works well with other languages.

# Stemming drawbacks

```
from nltk.stem import PorterStemmer
porter = PorterStemmer()
porter.stem('ate')
```

```
'ate'
```

- We expected to see “eat”.

```
from nltk.stem import PorterStemmer
porter = PorterStemmer()
print(porter.stem('university'), porter.stem('universal'))
```

```
univers univers
```

- “university” and “universal” are two different words with completely different meanings. Stemming returned a same root for both

# Lemmatization

- Lemmatization is a more advanced text normalization technique.
- It considers context and part of speech to determine the lemma.
- A lemma (plural lemmas or lemmata) is the canonical form, dictionary form of a set of words.
- Python NLTK provides WordNet Lemmatizer that uses the WordNet Database to lookup lemmas of words.
- Wordnet is database like a web of synonyms or a thesaurus.
- Because lemmatization involves deriving the meaning of a word from something like a dictionary, it's very time consuming.
- You need to provide the context in which you want to lemmatize that is the parts-of-speech (POS).
- You need to provide the word and pos (contect) as the input.
- pos constants:
  - ADJ, ADJ_SAT, ADV, NOUN, VERB = 'a', 's', 'r', 'n', 'v'

# Lemmatization

```
from nltk.stem import WordNetLemmatizer
lem = WordNetLemmatizer()
print(lem.lemmatize("running", pos = 'n'))
print(lem.lemmatize("running", pos = 'v'))
```

```
running
run
```

```
from nltk.stem import WordNetLemmatizer
lem = WordNetLemmatizer()
print(lem.lemmatize("better", pos = 'a'))
print(lem.lemmatize("ate", pos = 'v'))
```

```
good
eat
```

# Part of Speech (POS)

**What is Part-of-speech (POS) tagging?**

- It is a process of converting a sentence to a list of tuples, where each tuple is having a form (word, tag)).
- The tag is a part-of-speech and indicates whether the word is a noun, adjective, verb, and so on..
- NLTK uses the Penn Treebank parts for speech tags. Some examples:

| Tag | Part of speech |
|---|---|
| NNP | Proper noun, singular |
| NN | Noun, singular or mass |
| RB | Adverb |
| VBD | Verb, past tense |
| VBG | Verb, gerund or present participle |
| JJ | Adjective |
| PRP | Personal pronoun |

# Part of Speech (POS)

```python
import nltk
text = "Can you please buy me an Arizona Ice Tea? It is $0.99."
```

```python
print("parts of the speech:", nltk.pos_tag(tokens))
```

```
parts of the speech: [('Can', 'MD'), ('you', 'PRP'), ('please', 'VB'), ('buy', 'VB'), ('me', 'PRP'), ('an', 'DT'), ('Arizona',
'NNP'), ('Ice', 'NNP'), ('Tea', 'NNP'), ('?', '.'), ('It', 'PRP'), ('is', 'VBZ'), ('$', '$'), ('0.99', 'CD'), ('.', '.')]
```

# 6.7 Discuss and use Python modules to encode textual data as bag of words

# Bag-of-Words

- Machine learning algorithms cannot work with raw text directly; the text must be converted into numbers. Specifically, vectors of numbers.
- This is called feature extraction or feature encoding.
- Bag-of-Words is a popular and simple method of feature extraction with text.
- A bag-of-words is a representation of text that describes the occurrence of words within a document.
- It involves two things:
  - A vocabulary of known words.
    Constructing a document corpus which consists of all the unique words in the whole text.
  - A measure of the presence of known words.
    Counting the number of occurrences for each unique word.

# Bag-of-Words

- We use scikit-learn's CountVectorizer to implement bag-of-words.

```
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
```

```
text_data = np.array(['I love Brazil. Brazil!',
                      'Sweden is best',
                      'Germany beats both'])
```

```
count = CountVectorizer()
bag_of_words = count.fit_transform(text_data)
```

- The output is a sparse matrix

```
bag_of_words

<3x8 sparse matrix of type '<class 'numpy.int64'>'
    with 8 stored elements in Compressed Sparse Row format>
```

# Bag-of-Words

- We can use toarray to view a matrix of word counts for each observation.

```
bag_of_words.toarray()

array([[0, 0, 0, 2, 0, 0, 1, 0],
       [0, 1, 0, 0, 0, 1, 0, 1],
       [1, 0, 1, 0, 1, 0, 0, 0]], dtype=int64)
```

- We can use the vocabulary_ method to view the word associated with each feature

```
count.get_feature_names()

['beats', 'best', 'both', 'brazil', 'germany', 'is', 'love', 'sweden']
```

# Bag-of-Words

- We can use .toarray to see the feature matrix.

```
bag_of_words.toarray()

array([[0, 0, 0, 2, 0, 0, 1, 0],
       [0, 1, 0, 0, 0, 1, 0, 1],
       [1, 0, 1, 0, 1, 0, 0, 0]], dtype=int64)
```

- We can use the vocabulary_ method to view the word associated with each feature

```
count.get_feature_names()

['beats', 'best', 'both', 'brazil', 'germany', 'is', 'love', 'sweden']
```

- One useful argument in CountVecotrizer is ngram_range.
- We can set a range for ngram_range to consider bigrams, trigrams, etc. in our analysis.

# 6.8 Discuss the concept of TF/IDF scoring in Python

# TF-IDF

- A problem with scoring word frequency (bag-of-words) is that highly frequent words start to dominate in the document.
- Highly frequent words may not contain as much “informational content” to the model as rarer but perhaps domain specific words.
- One approach is to rescale the frequency of words by how often they appear in all documents.
- the scores for frequent words like “the” that are also frequent across all documents are penalized.
- Term Frequency (TF) is a scoring of the frequency of the word in the current document.
- Inverse Document Frequency (IDF) is a scoring of how rare the word is across documents

# TF-IDF

- We use scikit-learn's TfidfVectorizer to implement bag-of-words.

```
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
text_data = np.array(['I love Brazil. Brazil!',
                      'Sweden is best',
                      'Germany beats both'])
```

```
tfidf = TfidfVectorizer()
feature_matrix = tfidf.fit_transform(text_data)
```

- The output is a sparse matrix

# TF-IDF

- We can use .toarray to see the feature matrix.

```
array([[ 0.        ,  0.        ,  0.        ,  0.89442719,  0.        ,
         0.        ,  0.4472136 ,  0.        ],
       [ 0.        ,  0.57735027,  0.        ,  0.        ,  0.        ,
         0.57735027,  0.        ,  0.57735027],
       [ 0.57735027,  0.        ,  0.57735027,  0.        ,  0.57735027,
         0.        ,  0.        ,  0.        ]])
```

- We can use the vocabulary_ method to view the word associated with each feature

```
tfidf.vocabulary_

{'beats': 0,
 'best': 1,
 'both': 2,
 'brazil': 3,
 'germany': 4,
 'is': 5,
 'love': 6,
 'sweden': 7}
```

Any questions so far?
Any comments?

**Lambton College**

School of Computer Studies

# AML-2203 Advanced Python AI and ML Tools

[Prepared By: Vahid Hadavi, PhD]

# Lecture7

**7**. Evaluate Python modules to work with images (Recommended Module: OpenCV, matplotlib)

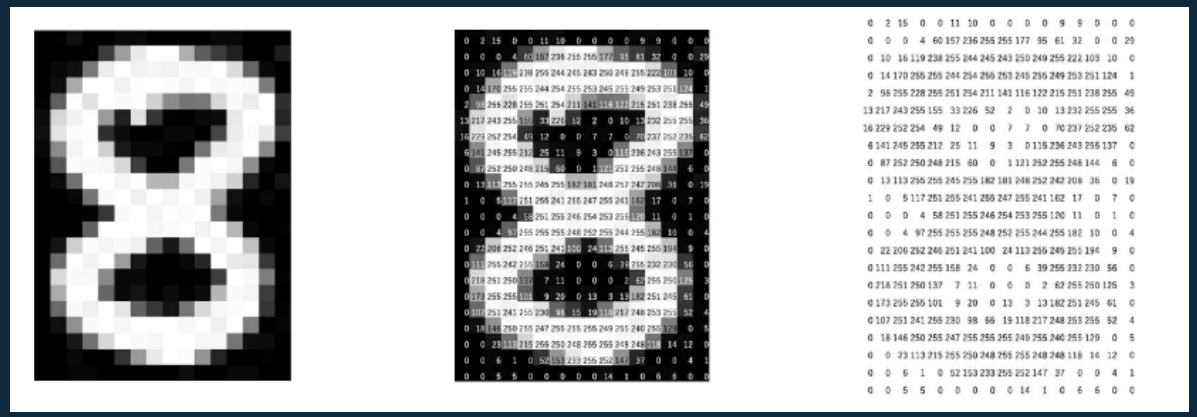
## Major Topics:

- 7.1 Discuss image data and their characteristics
- 7.2 Use Python modules to load images
- 7.3 Use Python modules to resize image
- 7.4 Use Python modules to crop images
- 7.5 Use Python modules to save images

# 7.1 Discuss image data and their characteristics

# Image data

- Handling images, especially image classification is one of the common areas of machine learning.
- Machines store images in the form of a matrix (or matrices for colored images) of numbers, and do the computation on it.
- The size of this matrix depends on the number of pixels we have in any given image.



- A colored image is typically composed of multiple colors (multiple matrices).
- Almost all colors can be generated from three primary colors – red, green and blue (a matrix for each color)

# 7.2 Use Python modules to load images

# Loading images

- OpenCV is the most popular and documented library for handling images.
- We use cv2.imread() to load the images.
- It needs the path to image and the flag
- Flag: It specifies the way in which image should be read.

  - cv2.IMREAD_COLOR: loading a color image.
  - cv2.IMREAD_GRAYSCALE: loading an image in grayscale mode.
  - cv2.IMREAD_UNCHANGED: loading an image as such including alpha channel.

- OpenCV works with BGR ordering (blue, green and red) while most of the other libraries (e.g. matplotlib) work with RGB (red, green and blue).
- We need to convert BGR to RGB using cv2.cvtColor() after loading the image.

# Loading images-grayscale

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
```

```
image = cv2.imread("images/plane.jpg", cv2.IMREAD_GRAYSCALE)
```

```
plt.imshow(image, cmap="gray"), plt.axis("off")
plt.show()
```

# Loading images-color

```
image_bgr = cv2.imread("images/plane.jpg", cv2.IMREAD_COLOR)
```

```
image_rgb = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2RGB)
```

```
plt.imshow(image_rgb), plt.axis("off")
plt.show()
```

# 7.3 Use Python modules to resize image

# Resizing images

- Resizing images is a common task in image preprocessing:
  - Images come in all shapes and sizes.
  - To be usable as features, images must have the same dimensions.

- Cost of resizing:
  - Images are matrices of information and when we reduce its size, we are reducing the size of that matrix and the information it contains

- We resize images using .resize().

```
image = cv2.imread("images/plane_256x256.jpg", cv2.IMREAD_GRAYSCALE)
```

```
image_50x50 = cv2.resize(image, (50, 50))
```

# 7.4 Use Python modules to crop images

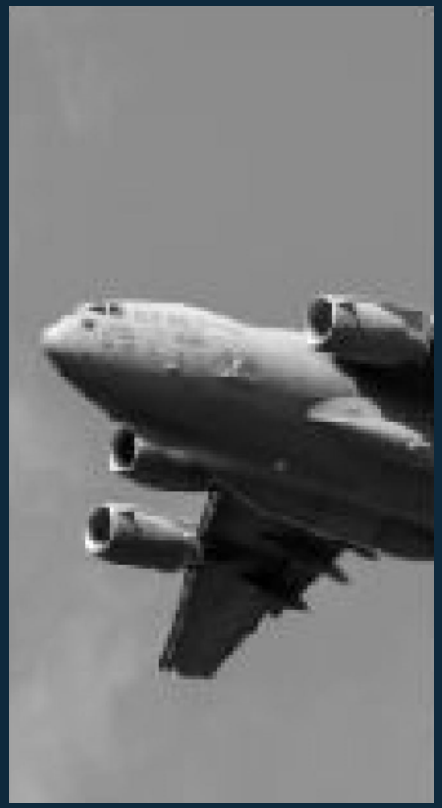
# Cropping images

- We can crop an image by selecting the columns and rows we want to keep.
- Cropping can be useful if we know that we only want to keep a certain part of every image.

```
image = cv2.imread("images/plane_256x256.jpg", cv2.IMREAD_GRAYSCALE)
```

```
image_cropped = image[:,:128]
```

```
plt.imshow(image_cropped, cmap="gray"), plt.axis("off")
plt.show()
```



# 7.5 Use Python modules to save images

# Saving images

- OpenCV's imwrite saves images to the filepath specified.
- The format of the image is defined by the filename's extension (.jpg, .png, etc.).

```
image = cv2.imread("images/plane_256x256.jpg", cv2.IMREAD_GRAYSCALE)
```

```
cv2.imwrite("images/plane_new.jpg", image)
```

Any questions so far?
Any comments?

**Lambton College**

School of Computer Studies

# AML-2203 Advanced Python AI and ML Tools

[Prepared By: Vahid Hadavi, PhD]

# Lecture8

**8**. Describe the concept of dimensionality reduction (Recommended Module: scikit-learn)

## Major Topics:

- 8.1 Discuss the need for dimensionality reduction
- 8.2 Explore various approaches to dimensionality reduction
- 8.3 Use Python modules to achieve dimensionality reduction by using feature extraction

# 8.1 Discuss the need for dimensionality reduction

# Dimensionality reduction

- The performance of machine learning algorithms can degrade with too many features.
- This phenomenon is often referred to as “The Curse of Dimensionality.”.
- This can be more problematic when the number of observations is not enough compared to the number of features, to train the model.
- It is often desirable to reduce the number of input features.
- We reduce the number of features with only a small loss in our data’s ability to generate high-quality predictions.
- The process of reducing the dimensionality of the feature space is called dimensionality reduction.
- It addresses multicollinearity by removing redundant features.
  - E.g. you have two variables; ‘time spent on treadmill in minutes’ and ‘calories burnt’. These variables are highly correlated.

# 8.2 Explore various approaches to dimensionality reduction

# Dimensionality reduction techniques

**Feature selection**

- Only keeping the most relevant variables from the original dataset.
- Dropping less useful features.
  - Removing features with missing values
  - Removing features with low variance
  - Removing highly correlated features

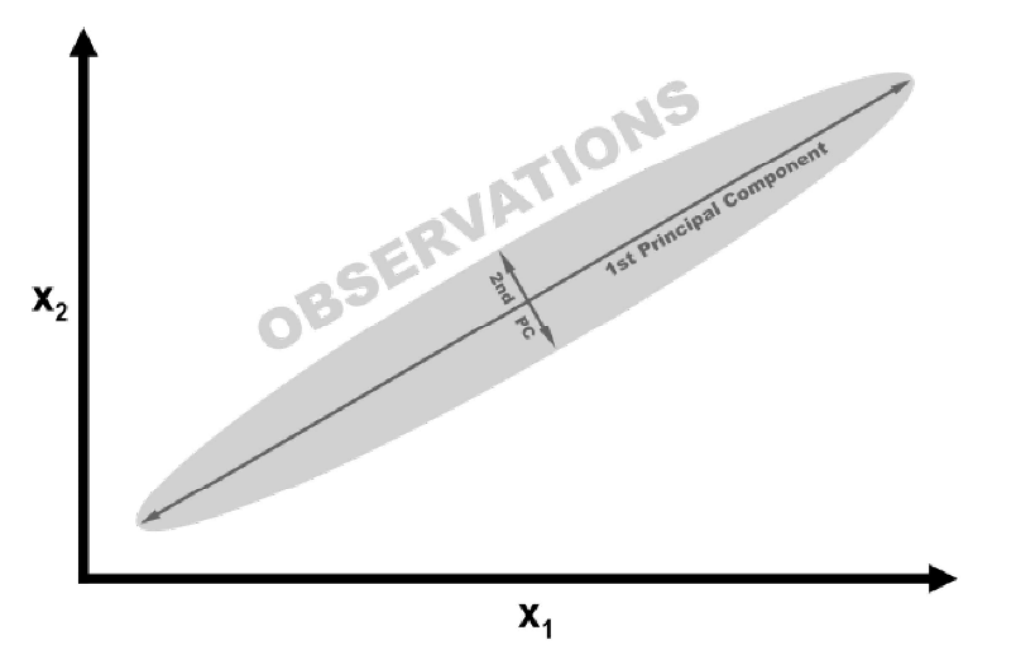
**Feature extraction**

- Creating a new, smaller set of features that stills captures most of the useful information.
- One downside of the feature extraction techniques is that the new features we generate will not be interpretable by humans.

# 8.3 Use Python modules to achieve dimensionality reduction by using feature extraction

# Principal component analysis (PCA)

- PCA is a popular linear dimensionality reduction technique.
- It is an unsupervised technique.
- PCA projects observations onto the principal components of the feature matrix that retain the most variance.



# Principal component analysis (PCA)

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn import datasets
```

```
digits = datasets.load_digits()
```

```
features = StandardScaler().fit_transform(digits.data)
```

```
pca = PCA(n_components=0.99, whiten=True)
```

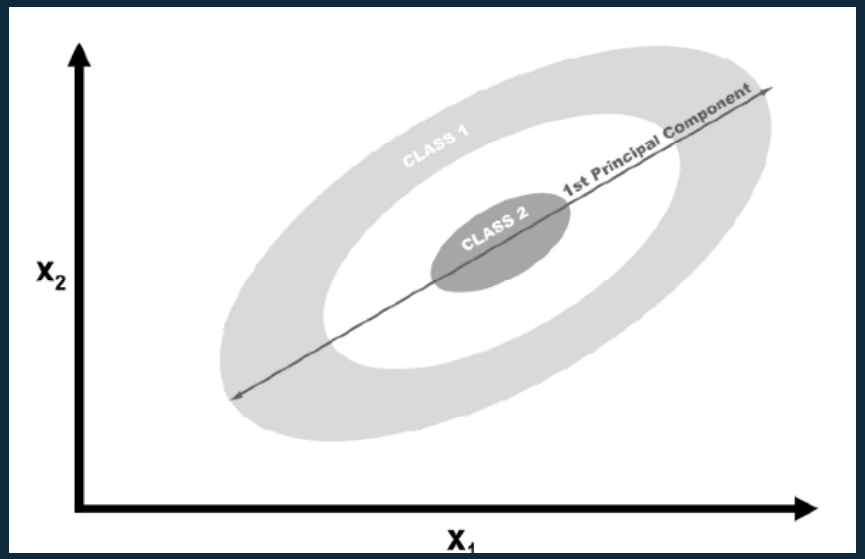
```
features_pca = pca.fit_transform(features)
```

```
print("Original number of features:", features.shape[1])
print("Reduced number of features:", features_pca.shape[1])
```

```
Original number of features: 64
Reduced number of features: 54
```

# Principal component analysis (PCA)

- Data is linearly separable if we can draw a straight line or hyperplane between different classes
- When our data is not linearly separable data, the standard PCA cannot be used.




- We want a transformation that both reduce the dimensions and make the data linearly separable.
- Kernel PCA can do both.

# Kernel PCA

- Kernels allow us to project the linearly inseparable data into a higher dimension
- where it is linearly separable.
- There are several kernels available to use:
  - Radial basis function (rbf)
  - Polynomial (poly)
  - Sigmoid

- Downside: there are several hyperparameters that we need to tune to achieve useful projection.
  - E.g. gamma in rbf

# Kernel PCA

```python
from sklearn.decomposition import PCA, KernelPCA
from sklearn.datasets import make_circles
```

```python
# Create linearly inseparable data
features, _ = make_circles(n_samples=1000, random_state=1, noise=0.1, factor=0.1)
```

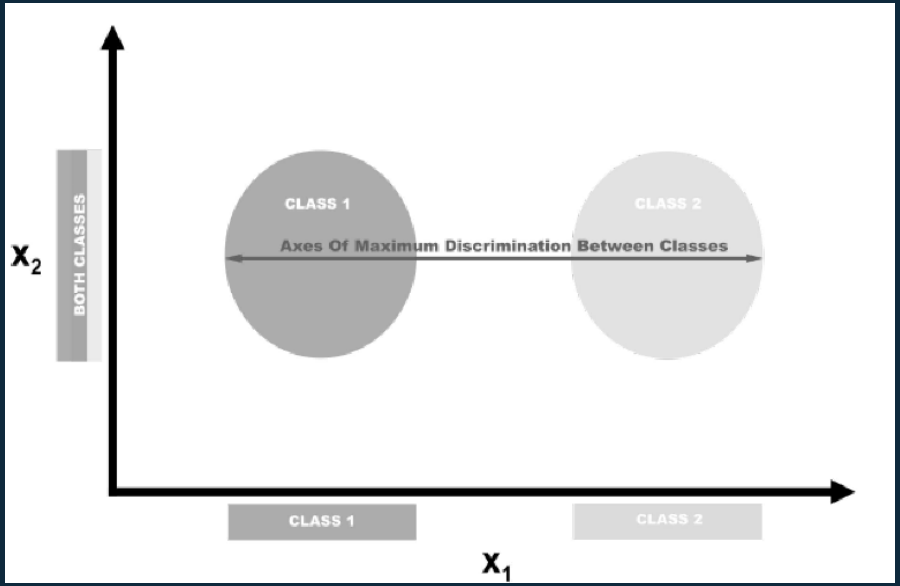
```python
kpca = KernelPCA(kernel="rbf", gamma=15, n_components=1)
features_kpca = kpca.fit_transform(features)
```

```python
print("Original number of features:", features.shape[1])
print("Reduced number of features:", features_kpca.shape[1])
```

```
Original number of features: 2
Reduced number of features: 1
```

# Linear Discriminant Analysis (LDA)

- LDA is a linear dimensionality reduction technique.
- LDA works similarly to PCA; it projects our feature space onto a lower-dimensional space.
- It is a supervised technique.
- In LDA, we have the additional goal of maximizing the differences between classes.
- It basically looks at what type of features subspace gives more discrimination to separate the data.



# Linear Discriminant Analysis (LDA)

```
from sklearn import datasets
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

```
iris = datasets.load_iris()
features = iris.data
target = iris.target
```

```
lda = LinearDiscriminantAnalysis(n_components=1)
features_lda = lda.fit(features, target).transform(features)
```

```
print("Original number of features:", features.shape[1])
print("Reduced number of features:", features_lda.shape[1])
```

```
Original number of features: 4
Reduced number of features: 1
```

```
lda.explained_variance_ratio_

array([ 0.99147248])
```

Any questions so far?
Any comments?

**Lambton College**

School of Computer Studies

# AML-2203 Advanced Python AI and ML Tools

[Prepared By: Vahid Hadavi, PhD]

# Lecture9

**8**. Implement linear regression in Python (Recommended Module: scikit-learn)

## Major Topics:

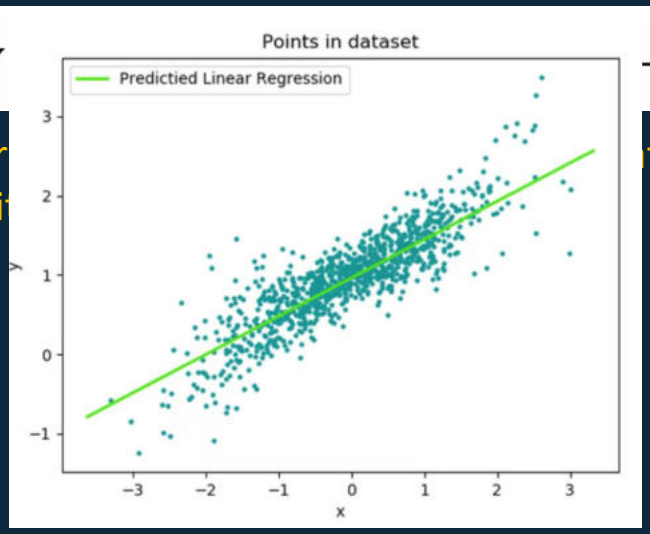
- 9.1 Discuss the importance of linear regression
- 9.2 Use Python modules to implement linear regression

# 9.1 Discuss the importance of linear regression

# Linear Regression

- Linear regression is one of the simplest supervised learning algorithms.
- Its core assumption is that the relationship between predictor & target variable is linear.
- It remains widely used across various domains such as Biology, Social Sciences, Finance, Marketing.
- It is usually used as a baseline model in regression problems.
- The mathematical representation:

$$Y \qquad\qquad + \epsilon$$




Where *Y* is our ture, *β*is are the coefficients identified by fi

# Linear Regression

- In standard linear regression the model trains to minimize the sum of squared error between the true and prediction values.

$$RSS = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

- Linear regression finds the weights to minimize the residual.

# 9.2 Use Python modules to implement linear regression

# Import and fitting a linear model

- We import LinearRegression from sklearn.model_selection package.

```
from sklearn.linear_model import LinearRegression
from sklearn.datasets import load_boston
```

- We import the first two columns of features from Boston housing database.

```
boston = load_boston()
features = boston.data[:,0:2]
target = boston.target
```

- We instantiate an object of linear regression and then train it by the data.

```
regression = LinearRegression()
```

```
model = regression.fit(features, target)
```

# Import and fitting a linear model

- We can check the feature coefficients and the bias term.

```
model.intercept_

22.46681692105723
```

```
model.coef_

array([-0.34977589,  0.11642402])
```

- We can use .predict() to use the model for predicting new observations.

# Interaction features

- Sometimes a feature's effect on our target variable is at least partially dependent on another feature.

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \hat{\beta}_3 x_1 x_2 + \epsilon$$

- We can add interaction features using polynomial features on scikit-learn.

```
from sklearn.preprocessing import PolynomialFeatures
```

```
boston = load_boston()
features = boston.data[:,0:2]
target = boston.target
```

```
interaction = PolynomialFeatures(
    degree=3, include_bias=False, interaction_only=True)
features_interaction = interaction.fit_transform(features)
```

# Interaction features

- We set interaction_only = Ture  to have only the interaction terms, not the polynomial features (those raised to an exponent).
- The degree parameter determines the maximum number of features to create interaction
- terms from.
- By default, PolynomialFeatures will add a feature containing ones called a bias.
- We can prevent that with include_bias=False.
- Later we use the new set of features (including interaction features) to train our model.

```
regression = LinearRegression()
```

```
model = regression.fit(features_interaction, target)
```

# polynomial features

- Sometimes we want to model a non-linear relationship
- Polynomial regression is an extension of linear regression to allow us to model nonlinear relationships.

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_1^2 + \ldots + \hat{\beta}_d x_i^d + \epsilon$$

where *d* is the degree of the polynomial

- This is still a weighted linear combination of features, so it is still a linear model.
- Beware of polynomial feature expansion of with high degree that can lead to complex models with overfitting.
- We use polynomial feature on scikit-learn to add new polynomial features.

```
polynomial = PolynomialFeatures(degree=3, include_bias=False)
features_polynomial = polynomial.fit_transform(features)
```

- Note that we didn't set the interaction = True in this example.

# Ridge Regularization

- To battle overfitting, regularization is used.
- Overfitting happens when the model is trying too hard to capture the noise in the training dataset
- This technique discourages learning a more complex or flexible model, to avoid the risk of overfitting.
- This technique attempts to minimize RSS and some penalty for the total size of the coefficient values

$$\mathrm{RSS}+\alpha \sum_{j=1}^{p} \hat{\beta}_j^2$$

- The influence of regularization term can be controlled by α.
- Higher α means more regularization and simpler models.
- When the number of features are so many, and they have small contribution in the model, ridge regularization should be used.

# Ridge Regularization

- The implementation in python is similar to linear regression.

```
from sklearn.linear_model import Ridge
```

```
regression = Ridge(alpha=0.5)
```

- scikit-learn includes a RidgeCV method that allows us to select the ideal value for α.

```
from sklearn.linear_model import RidgeCV
```

```
regr_cv = RidgeCV(alphas=[0.1, 1.0, 10.0])
```

# Lasso Regularization

- Lasso regularization minimizes the sum of the absolute values of the coefficients .

$$\mathrm{RSS} + \alpha \sum_{j=1}^{p} \left|\hat{\beta}_j\right|$$

- Lasso can shrink th coefficients of a model to zero for the least influential features.
- The influence of regularization term can be controlled by α.
- When there are only a few features with medium to large contribution, Lasso regularization is recommended.

# Ridge Regularization

- The implementation in python is similar to linear regression.

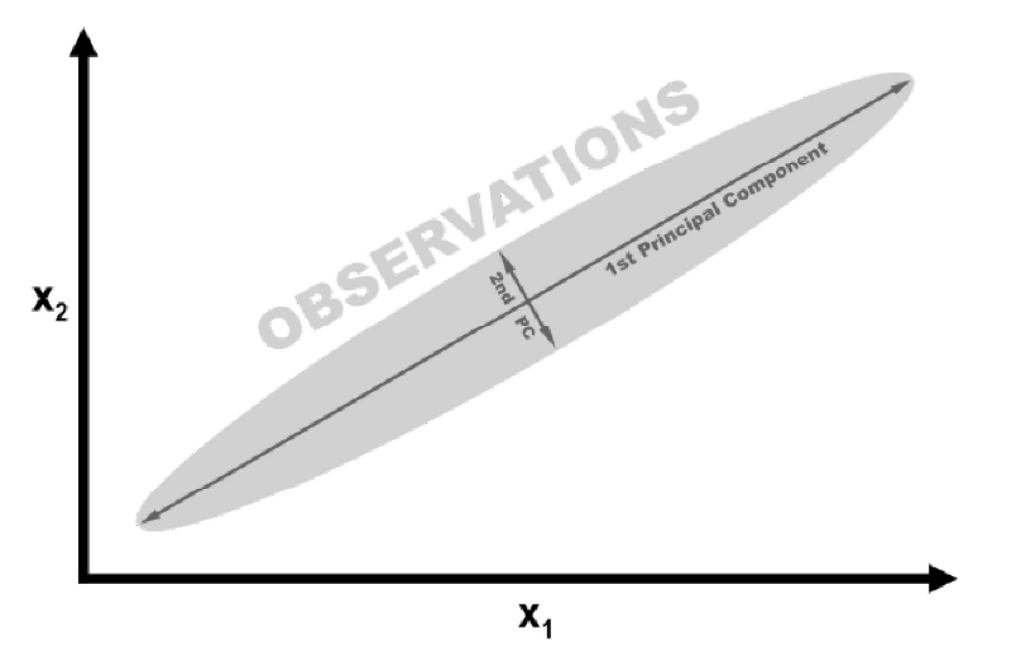
```
from sklearn.linear_model import Lasso
```

```
regression = Lasso(alpha=0.5)
```

# 8.3 Python modules to achieve dimensionality reduction by using feature extraction

# Principal component analysis (PCA)

- PCA is a popular linear dimensionality reduction technique.
- It is an unsupervised technique.
- PCA projects observations onto the principal components of the feature matrix that retain the most variance.



# Principal component analysis (PCA)

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn import datasets
```

```
digits = datasets.load_digits()
```

```
features = StandardScaler().fit_transform(digits.data)
```

```
pca = PCA(n_components=0.99, whiten=True)
```

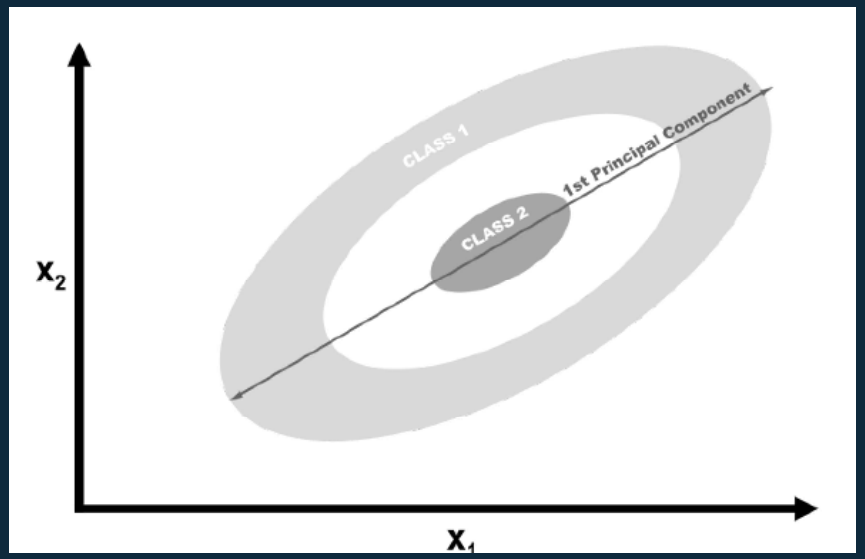
```
features_pca = pca.fit_transform(features)
```

```
print("Original number of features:", features.shape[1])
print("Reduced number of features:", features_pca.shape[1])
```

```
Original number of features: 64
Reduced number of features: 54
```

# Principal component analysis (PCA)

- Data is linearly separable if we can draw a straight line or hyperplane between different classes
- When our data is not linearly separable data, the standard PCA cannot be used.




- We want a transformation that both reduce the dimensions and make the data linearly separable.
- Kernel PCA can do both.

# Kernel PCA

- Kernels allow us to project the linearly inseparable data into a higher dimension
- where it is linearly separable.
- There are several kernels available to use:
  - Radial basis function (rbf)
  - Polynomial (poly)
  - Sigmoid

- Downside: there are several hyperparameters that we need to tune to achieve useful projection.
  - E.g. gamma in rbf

# Kernel PCA

```python
from sklearn.decomposition import PCA, KernelPCA
from sklearn.datasets import make_circles
```

```python
# Create linearly inseparable data
features, _ = make_circles(n_samples=1000, random_state=1, noise=0.1, factor=0.1)
```

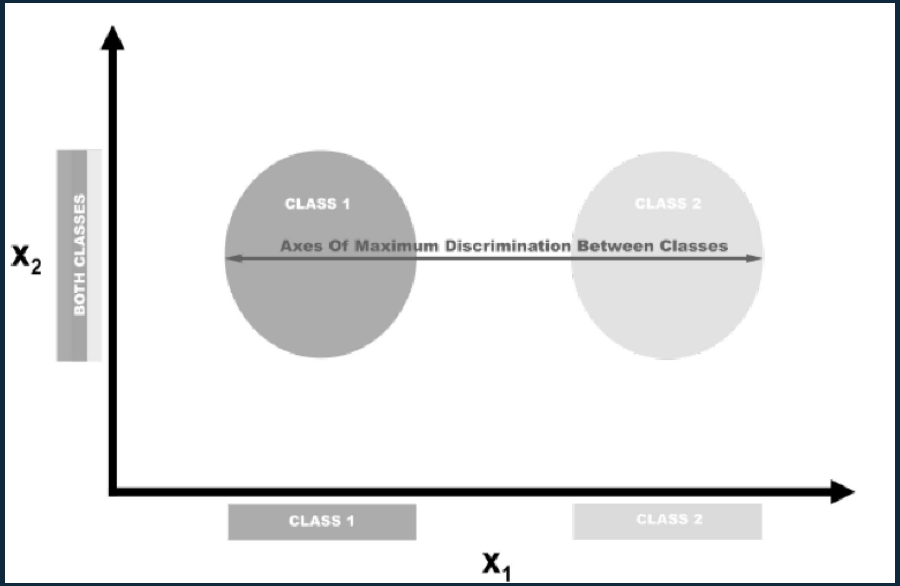
```python
kpca = KernelPCA(kernel="rbf", gamma=15, n_components=1)
features_kpca = kpca.fit_transform(features)
```

```python
print("Original number of features:", features.shape[1])
print("Reduced number of features:", features_kpca.shape[1])
```

```
Original number of features: 2
Reduced number of features: 1
```

# Linear Discriminant Analysis (LDA)

- LDA is a linear dimensionality reduction technique.
- LDA works similarly to PCA; it projects our feature space onto a lower-dimensional space.
- It is a supervised technique.
- In LDA, we have the additional goal of maximizing the differences between classes.
- It basically looks at what type of features subspace gives more discrimination to separate the data.



# Linear Discriminant Analysis (LDA)

```
from sklearn import datasets
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

```
iris = datasets.load_iris()
features = iris.data
target = iris.target
```

```
lda = LinearDiscriminantAnalysis(n_components=1)
features_lda = lda.fit(features, target).transform(features)
```

```
print("Original number of features:", features.shape[1])
print("Reduced number of features:", features_lda.shape[1])
```

```
Original number of features: 4
Reduced number of features: 1
```

```
lda.explained_variance_ratio_

array([ 0.99147248])
```

Any questions so far?
Any comments?

**Lambton College**

School of Computer Studies

# AML-2203 Advanced Python AI and ML Tools

[Prepared By: Vahid Hadavi, PhD]

# Lecture10

**10**. Discuss the use of Python in visualizing data (Recommended Module: (Matplotlib, Plotly)

## Major Topics:

- 10.1 Use Python modules to visualize data sets
- 10.2 Use Python modules to visualize linear regression

# 10.1 Use Python modules to visualize data sets

# Data Exploration

- Data exploration is the initial step in data analysis.
- In this step, users explore a large data set in an unstructured way to uncover initial patterns, characteristics, and points of interest.
- This process is not meant to reveal every bit of information a dataset holds, but rather to help create a broad picture of important trends and major points to study in greater detail.
- data exploration uses visualization because it creates a more straightforward view of data sets than simply examining thousands of individual numbers or names.

# Data Visualization

- Data visualization is the graphical representation of information and data.
- Visual data is very easy to understand compared to data in any other form.
- Advantages of data visualization:
  - Complex data can be easily understood
  - A simple visual representation of outliers, target audiences, and future markets can be created
  - Storytelling can be done using dashboards and animations
  - Data can be explored through interactive visualizations

# Data Visualization

- Most common types:
  - Bar Chart
  - Line Chart
  - Scatterplot
  - Pie Chart
  - Heat Map
  - Histogram
  - Box Plot

# Data visualization tools

- Non-coding tools:
  - Datawrapper
  - Tableau
  - Plotly

- Coding tools:
  - Python
  - MATLAB
  - R

# Data visualization with python

- Python packages:
  - Matplotlib

    The first and the most commonly used library for data visualization. It is simple yet very powerful.

  - Seaborn

    It is a popular data visualization library that is built on top of Matplotlib. Seaborn's default styles and color palettes are much more sophisticated than Matplotlib.

  - Plotly

    Plotly is widely known as an online platform for data visualization, very few people know that it can be accessed from a Python notebook.

# Matplotlib

- Matplotlib is a plotting library for the Python programming language.
- Matplotlib is probably the most popular plotting library for Python.
- John Hunter began developing Matplotlib in 2003.
- It aimed to emulate the commands of the MATLAB software.

# Matplotlib

- Plots in Matplotlib have a hierarchical structure that nests Python objects to create a tree-like structure.
- Each plot is encapsulated in a Figure object.
- Figure is the top-level container of the visualization.
- It can have multiple axes, which are basically individual plots inside this top-level container.
- Python objects that control axes, tick marks, legend, title, textboxes, the grid, and many other objects.
- pyplot contains a simpler interface for creating visualizations, which allows the users to plot the data without explicitly configuring the Figure and Axes themselves.
- They are implicitly and automatically configured to achieve the desired output.
- It is handy to use the alias plt to reference the imported submodule, as follows:

```
import matplotlib.pyplot as plt
```

# Pyplot basics; creating Figure

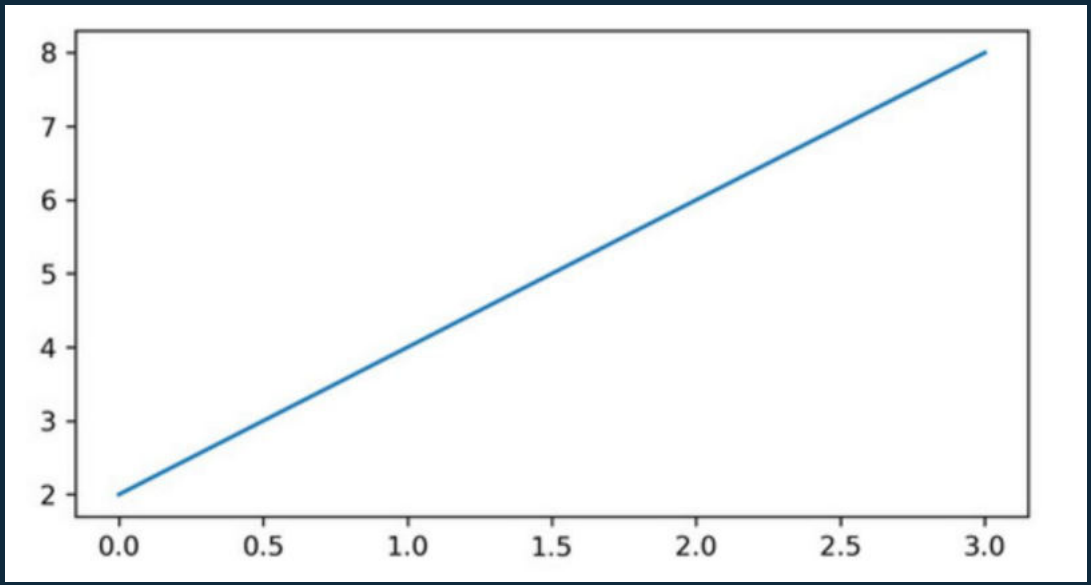
- We use plt.figure() to create a new Figure.
- This function returns a Figure instance, but it is also passed to the backend.
- Every Figure-related command that follows is applied to the current Figure and does not need to know the Figure instance.
- Figures that are not used anymore should be closed by explicitly calling plt.close(), which also cleans up memory efficiently.
- By default, the Figure has a width of 6.4 inches and a height of 4.8 inches with a dpi of 100.
- To change the default values of the Figure, we can use the parameters figsize and dpi

```
plt.figure(figsize=(10, 5)) #To change the width and the height
plt.figure(dpi=300) #To change the dpi
```

# Pyplot basics; plotting

- With plt.plot([x], y, [fmt]), we can plot data points as lines and/or markers.
- By default, if you do not provide a format string, the data points will be connected with straight, solid lines.
- plt.show() is used to display a Figure or multiple Figures.

```
plt.plot([0, 1, 2, 3], [2, 4, 6, 8])
```



# Pyplot basics; plotting

- We can use Line2D properties as keyword arguments for more control on the appearance.

```
plt.plot([2, 4, 6, 8], color='blue', marker='o', linestyle='dashed', linewidth=2, markersize=12)
```

- Refer to the following link to see the list of available properties for Line2D: https://matplotlib.org/3.2.1/api/_as_gen/matplotlib.lines.Line2D.html#matplotlib.lines.Line2D

# Seaborn

- Unlike Matplotlib, Seaborn is not a standalone Python library.
- It is built on top of Matplotlib and provides a higher-level abstraction to make visually appealing statistical visualizations.
- The most prominent features of Seaborn are as follows:
  - Beautiful out of the box plots with different themes
  - Built-in color palettes that can be used to reveal patterns in the dataset
  - Dataset-oriented interface
  - A high-level abstraction that still allows for complex visualizations

# Seaborn

- It uses fewer syntax and has easily interesting default themes.
- While Matplotlib is highly customizable and powerful, Seaborn avoids a ton of boilerplate by providing default themes which are commonly used.
- A neat feature of Seaborn is the ability to integrate with DataFrames from the pandas library.
- Seaborn works with the dataset as a whole and is much more intuitive than Matplotlib.
- In Seaborn, replot() is the entry API with ‘kind’ parameter to specify the type of plot which could be line, bar, or many of the other types (default scatter).

# Seaborn

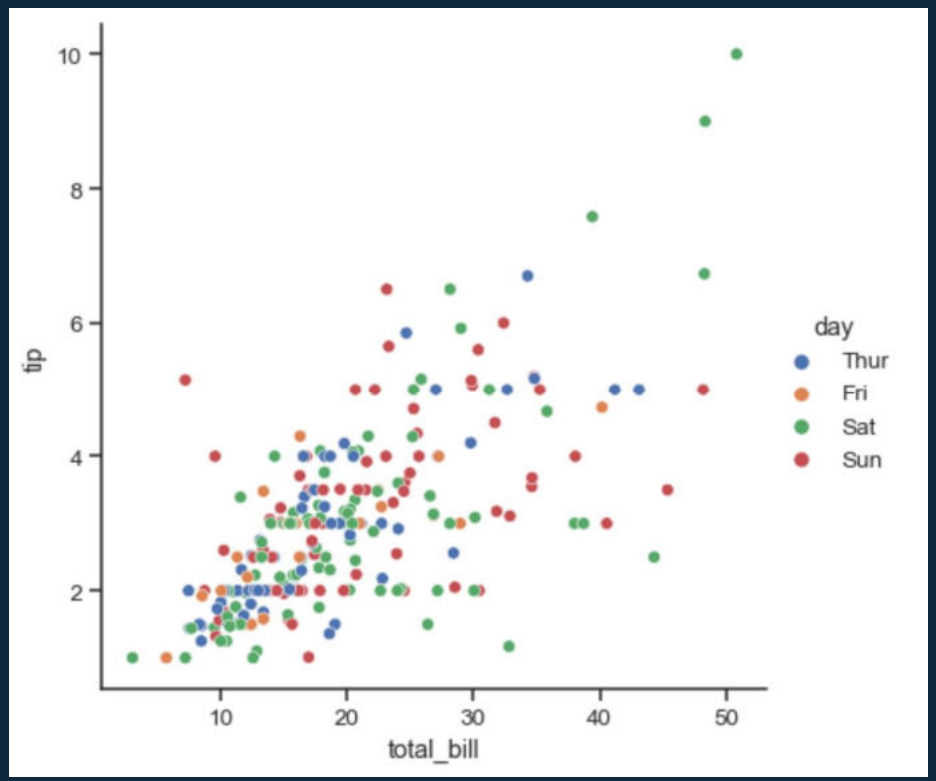
- Seaborn needs to imported.

```
tips = sns.load_dataset("tips")
tips.head()
```

| | total_bill | tip | sex | smoker | day | time | size |
|---|---|---|---|---|---|---|---|
| **0** | 16.99 | 1.01 | Female | No | Sun | Dinner | 2 |
| **1** | 10.34 | 1.66 | Male | No | Sun | Dinner | 3 |
| **2** | 21.01 | 3.50 | Male | No | Sun | Dinner | 3 |
| **3** | 23.68 | 3.31 | Male | No | Sun | Dinner | 2 |
| **4** | 24.59 | 3.61 | Female | No | Sun | Dinner | 4 |

# Seaborn

```
sns.relplot(data=tips, x="total_bill", y="tip", hue="day")
```



# Plotly

- Plotly is a plotting ecosystem that allows you to make plots in Python, as well as JavaScript and R.
- By using matplotlib or seaborn, we can only create static image files (jpeg or png or any static image format).
- What makes Plotly different is that it supports JavaScript, so it will respond to mouse events.
- Plotly in python, by itself creates interactive plots (as .html files).
- Users can interact with these plots (by zooming in, selecting particular area within plot for detailed analysis and hover-across the plot, etc..).
- Plotly has three different Python APIs:
  - An object-oriented API that feels similar to Matplotlib
  - A data-driven API that specifies plots by constructing dictionaries of JSON-like data
  - A "Plotly Express" API that gives you high-level plotting functions similar to Seaborn.

# Plotly exxpress

- Plotly Express is a new high-level Python visualization library.
- It is a wrapper for Plotly.py that exposes a simple syntax for complex charts.
- It is pretty easy to use, and doesn't require connecting your file to Plotly or specifying that you want to work with Plotly offline.
- In simple terms, we can assume it as an IDE for plotly.
- We need to import plot express as follows:

```
import plotly express as px
```

- Plotly express has built-in datasets.

```
iris_data = px.data.iris()
```

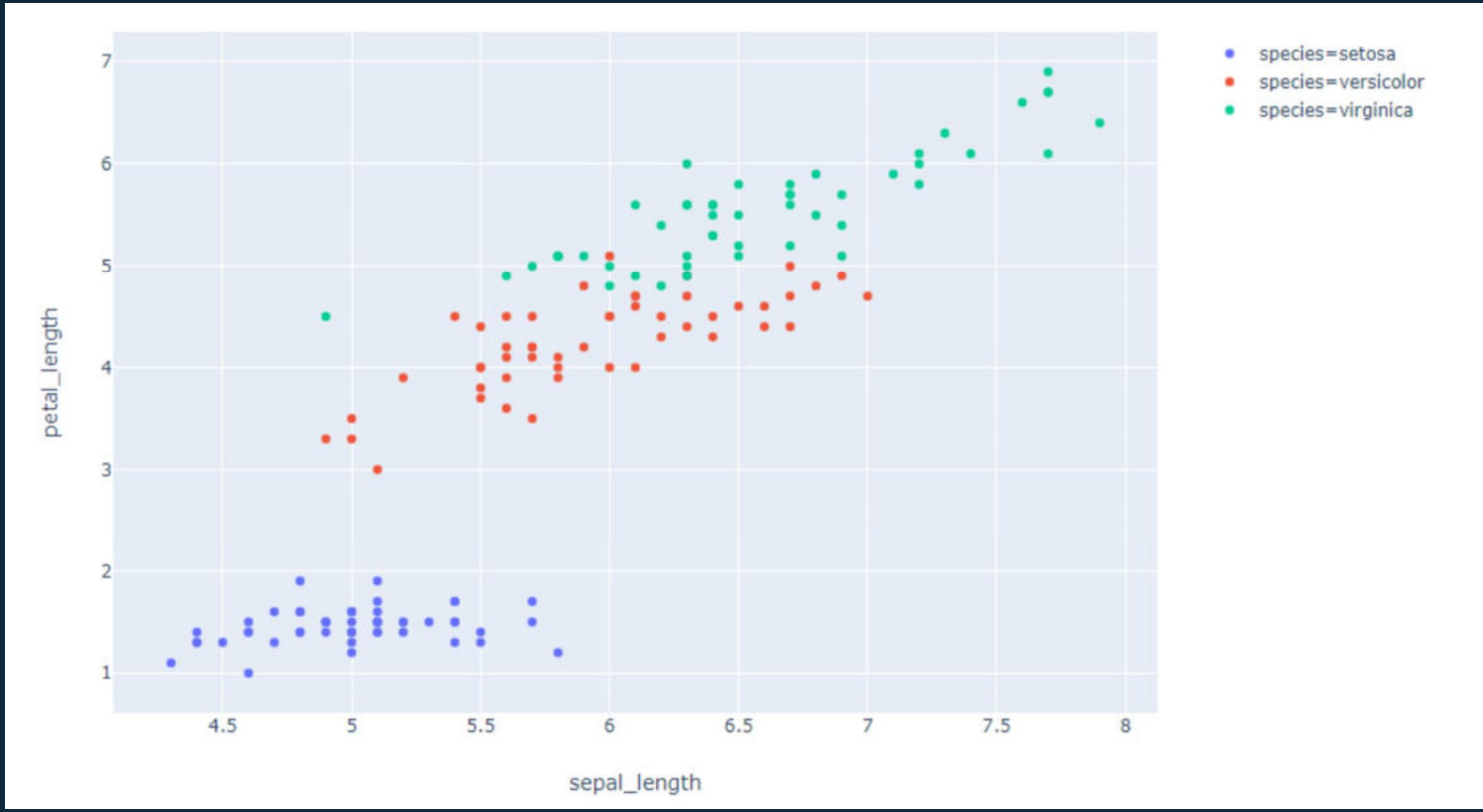
# Plotly exxpress

```
iris_data.head()
```

| | sepal_length | sepal_width | petal_length | petal_width | species | species_id |
|---|---|---|---|---|---|---|
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | setosa | 1 |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | setosa | 1 |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | setosa | 1 |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | setosa | 1 |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 | setosa | 1 |

```
figure = px.scatter(iris_data,x='sepal_length',y='petal_length',color='species')
figure.show()
```

# Plotly exxpress



# 10.2 Use Python modules to visualize linear regression

# Linear regression with seaborn

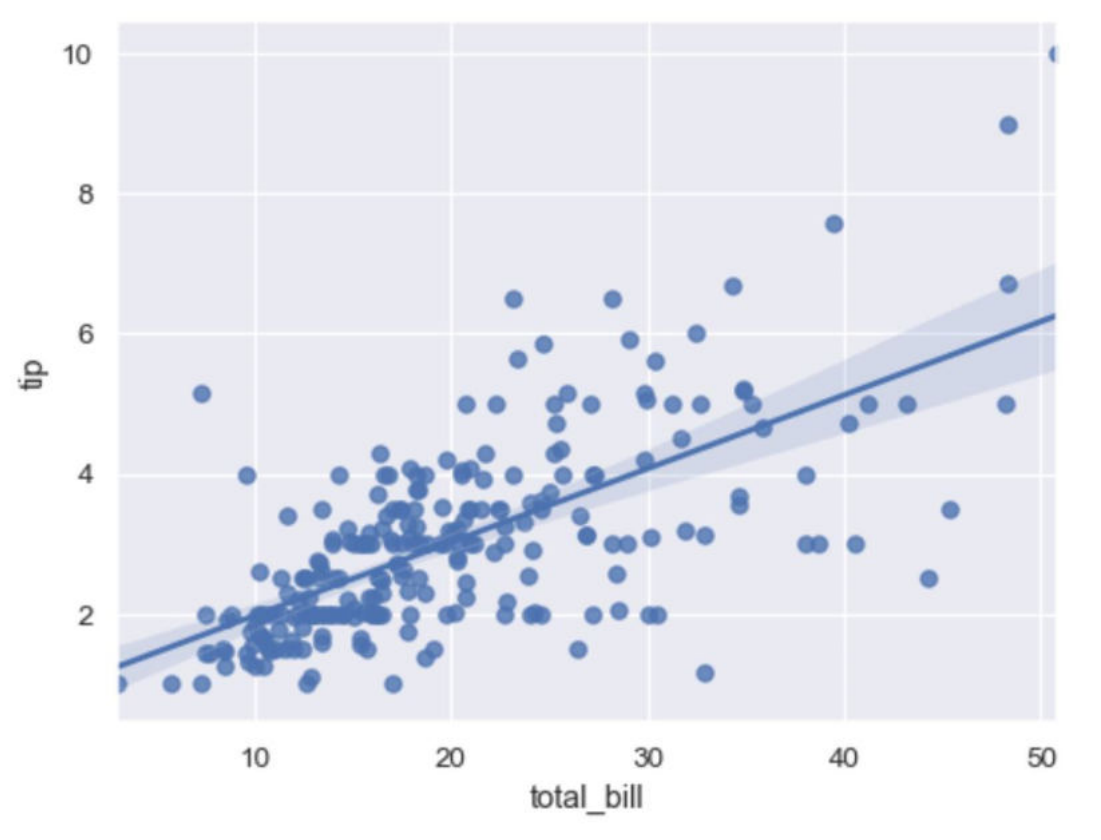
- The main function in seaborn used to visualize a linear relationship is regplot().
- regplot() accepts the x and y variables in a variety of formats including simple numpy arrays, pandas Series objects, or as references to variables in a pandas DataFrame object passed to data.

```
tips = sns.load_dataset("tips")
```

| | total_bill | tip | sex | smoker | day | time | size |
|---|---|---|---|---|---|---|---|
| 0 | 16.99 | 1.01 | Female | No | Sun | Dinner | 2 |
| 1 | 10.34 | 1.66 | Male | No | Sun | Dinner | 3 |
| 2 | 21.01 | 3.50 | Male | No | Sun | Dinner | 3 |
| 3 | 23.68 | 3.31 | Male | No | Sun | Dinner | 2 |
| 4 | 24.59 | 3.61 | Female | No | Sun | Dinner | 4 |

# Linear regression with seaborn

```
sns.regplot(x="total_bill", y="tip", data=tips)
```



# Linear regression with plotly

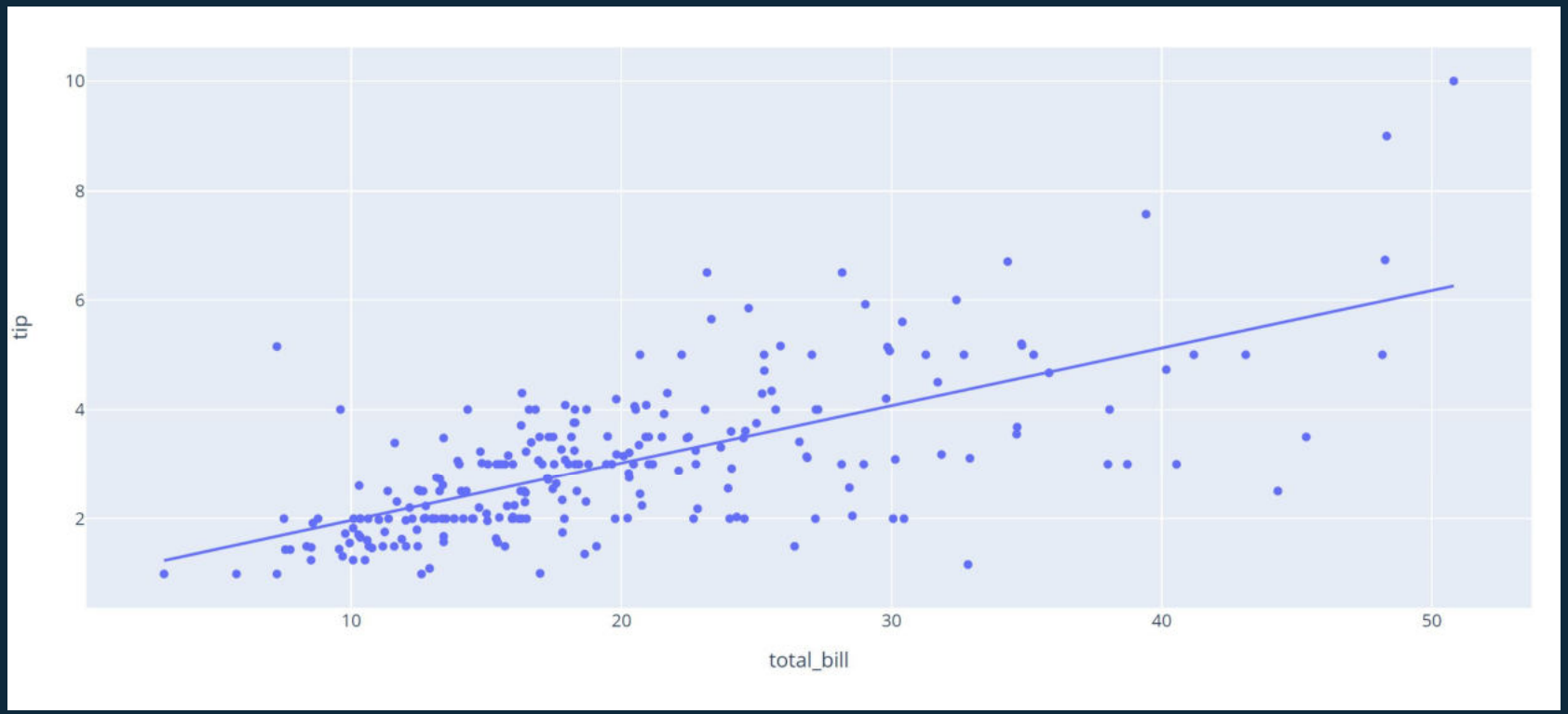
- We can use ‘trendline’ parameter in scatter plot on plotly express to fit a line on our data.
- In order to do so, you will need to install statsmodels and its dependencies.
- There are different options to select from for trendline.
- Ordinary least square (ols) is the most common option for trendline.

```
df = px.data.tips()
```

| | total_bill | tip | sex | smoker | day | time | size |
|---|---|---|---|---|---|---|---|
| **0** | 16.99 | 1.01 | Female | No | Sun | Dinner | 2 |
| **1** | 10.34 | 1.66 | Male | No | Sun | Dinner | 3 |
| **2** | 21.01 | 3.50 | Male | No | Sun | Dinner | 3 |
| **3** | 23.68 | 3.31 | Male | No | Sun | Dinner | 2 |
| **4** | 24.59 | 3.61 | Female | No | Sun | Dinner | 4 |

# Linear regression with plotly

```
fig = px.scatter(df, x="total_bill", y="tip", trendline="ols")
fig.show()
```



Any questions so far?
Any comments?