**Functions**

1. Making numpy array as matrices and vectors

import numpy as np

vector = np.array([10, 20, 30])

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

1. Dimension of array

vector = numpy.array([1, 2, 3, 4])

print(vector.shape)

1. Reading a dataset in Numpy

world\_alcohol = np.genfromtxt("world\_alcohol.csv", delimiter=",")

print(type(world\_alcohol))

world\_alcohol = np.genfromtxt("world\_alcohol.csv", delimiter=",", dtype="U75", skip\_header=1)

print(world\_alcohol)

world\_alcohol.dtype

1. Selecting all rows and specific columns

countries = world\_alcohol[:,2]

alcohol\_consumption = world\_alcohol[:,4]

# slice of one column

first\_two\_columns = world\_alcohol[:,0:2]

first\_ten\_years = world\_alcohol[0:10,0]

first\_ten\_rows = world\_alcohol[0:10,:]

# Slice of both row and column

first\_twenty\_regions = world\_alcohol[0:20,1:3]

1. Computatio with Numpy

years\_1984 = (world\_alcohol[:,0] == "1984")

countries\_canada = (world\_alcohol[:,2] == "Canada")

#Extracting rows with conditions and multiple conditions

country\_is\_algeria = world\_alcohol[:,2] == "Algeria"

country\_algeria = world\_alcohol[country\_is\_algeria,:]

is\_algeria\_and\_1986 = (world\_alcohol[:,0] == "1986") & (world\_alcohol[:,2] == "Algeria")

rows\_with\_algeria\_and\_1986 = world\_alcohol[is\_algeria\_and\_1986,:]

#Changing row values with other values

world\_alcohol[:,0][world\_alcohol[:,0] == '1986'] = '2014'

world\_alcohol[:,3][world\_alcohol[:,3] == 'Wine'] = 'Grog'

## Filling null values

is\_value\_empty = world\_alcohol[:,4] == ''

world\_alcohol[is\_value\_empty, 4] = '0'

#We can convert the data type of an array with the [astype()](http://docs.scipy.org/doc/numpy-1.10.1/reference/generated/numpy.ndarray.astype.html) method.

alcohol\_consumption = world\_alcohol[:,4]

alcohol\_consumption = alcohol\_consumption.astype(float)

#Data row operations

total\_alcohol = alcohol\_consumption.sum()

average\_alcohol = alcohol\_consumption.mean()

## All Operations

is\_canada\_1986 = (world\_alcohol[:,2] == "Canada") & (world\_alcohol[:,0] == '1986')

canada\_1986 = world\_alcohol[is\_canada\_1986,:]

canada\_alcohol = canada\_1986[:,4]

empty\_strings = canada\_alcohol == ''

canada\_alcohol[empty\_strings] = "0"

canada\_alcohol = canada\_alcohol.astype(float)

total\_canadian\_drinking = canada\_alcohol.sum()

totals = {}

is\_year = world\_alcohol[:,0] == "1989"

year = world\_alcohol[is\_year,:]

for country in countries:

is\_country = year[:,2] == country

country\_consumption = year[is\_country,:]

alcohol\_column = country\_consumption[:,4]

is\_empty = alcohol\_column == ''

alcohol\_column[is\_empty] = "0"

alcohol\_column = alcohol\_column.astype(float)

totals[country] = alcohol\_column.sum()

print(totals)

highest\_value = 0

highest\_key = None

for country in totals:

consumption = totals[country]

if highest\_value < consumption:

highest\_value = consumption

highest\_key = country

**PANDAS**

1. Data manipulation with pandas

import pandas

food\_info = pandas.read\_csv("food\_info.csv")

#list of all column names

col\_names = food\_info.columns.tolist()

print(col\_names)

print(food\_info.head(3))

1. Operating on Columns

div\_1000 = food\_info["Iron\_(mg)"] / 1000

add\_100 = food\_info["Iron\_(mg)"] + 100

sub\_100 = food\_info["Iron\_(mg)"] - 100

mult\_2 = food\_info["Iron\_(mg)"]\*2

sodium\_grams = food\_info["Sodium\_(mg)"] / 1000

sugar\_milligrams = food\_info["Sugar\_Tot\_(g)"] \* 1000

water\_energy = food\_info["Water\_(g)"] \* food\_info["Energ\_Kcal"]

print(water\_energy[0:5])

# The largest value in the "Energ\_Kcal" column.

max\_calories = food\_info["Energ\_Kcal"].max()

# Divide the values in "Energ\_Kcal" by the largest value to normalize.

normalized\_calories = food\_info["Energ\_Kcal"] / max\_calories

1. Add processed column to the DataFrame as a new column instead

food\_info["Normalized\_Protein"] = normalized\_protein

food\_info["Normalized\_Fat"] = normalized\_fat

1. DataFrame objects have a [sort\_values() method](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.sort_values.html) that we can use to sort the entire DataFrame.

food\_info.sort\_values("Sodium\_(mg)")

# Sorts the DataFrame in-place, rather than returning a new DataFrame.

food\_info.sort\_values("Sodium\_(mg)", inplace=True)

# Sorts by descending order, rather than ascending.

food\_info.sort\_values("Sodium\_(mg)", inplace=True, ascending=False)

1. Missing values in dataset

age = titanic\_survival["age"]

print(age.loc[10:20])

age\_is\_null = pd.isnull(age)

age\_null\_true = age[age\_is\_null]

age\_null\_count = len(age\_null\_true)

print(age\_null\_count)

age\_is\_null = pd.isnull(titanic\_survival["age"])

good\_ages = titanic\_survival["age"][age\_is\_null == False]

correct\_mean\_age = sum(good\_ages) / len(good\_ages)

1. Rows without missing data

age\_is\_null = pd.isnull(titanic\_survival["age"])

age\_true = age\_is\_null==False

true\_rows = titanic\_survival[age\_true]

print(len(true\_rows))

good\_ages = titanic\_survival["age"][age\_is\_null == False]

correct\_mean\_age = sum(good\_ages) / len(good\_ages)

#operations without missing data with series.mean()

correct\_mean\_age = titanic\_survival["age"].mean()

correct\_mean\_fare = titanic\_survival["fare"].mean()

1. # Operations on unique data row valus in column

passenger\_classes = [1, 2, 3]

fares\_by\_class = {}

for this\_class in passenger\_classes:

pclass\_rows = titanic\_survival[titanic\_survival["pclass"] == this\_class]

pclass\_fares = pclass\_rows["fare"]

fare\_for\_class = pclass\_fares.mean()

fares\_by\_class[this\_class] = fare\_for\_class

1. Pivot function for function on data rows by columns

passenger\_survival = titanic\_survival.pivot\_table(index="pclass", values="survived")

passenger\_age = titanic\_survival.pivot\_table(index="pclass", values="age")

print(passenger\_age)

#If we pass a list of column names to the valuesparameter instead of a single value, we can perform calculations on multiple columns at once.

import numpy as np

port\_stats = titanic\_survival.pivot\_table(index="embarked", values=["fare","survived"], aggfunc=numpy.sum)

print(port\_stats)

1. Dropping missing rows and columns

drop\_na\_rows = titanic\_survival.dropna(axis=0)

drop\_na\_columns = titanic\_survival.dropna(axis=1)

new\_titanic\_survival = titanic\_survival.dropna(axis=0,subset=["age", "sex"])

1. Extracting row labels with and without integer

first\_five\_rows = new\_titanic\_survival.iloc[0:5]# this gives first five rows

first\_ten\_rows = new\_titanic\_survival.iloc[0:10]

row\_index\_25 = new\_titanic\_survival.loc[25]# this gives row with 25 label

row\_position\_fifth = new\_titanic\_survival.iloc[4]

first\_row\_first\_column = new\_titanic\_survival.iloc[0,0]

all\_rows\_first\_three\_columns = new\_titanic\_survival.iloc[:,0:3]

row\_index\_83\_age = new\_titanic\_survival.loc[83,"age"]

row\_index\_766\_pclass = new\_titanic\_survival.loc[766,"pclass"]

row\_index\_1100\_age = new\_titanic\_survival.loc[1100, "age"]

row\_index\_25\_survived = new\_titanic\_survival.loc[25, "survived"]

five\_rows\_three\_cols = new\_titanic\_survival.iloc[0:5,0:3]

1. Resetting the index after sorting of data

titanic\_reindexed = new\_titanic\_survival.reset\_index(drop=True)#true will drop the original index

print(titanic\_reindexed.iloc[0:5,:])

1. Dataframe column

def hundredth\_row(column):

hundredth\_item = column.iloc[99]

return hundredth\_item

hundredth\_row\_var = titanic\_survival.apply(hundredth\_row)

def not\_null\_count(column):

column\_null = pd.isnull(column)

null = column[column\_null]

return len(null)

column\_null\_count = titanic\_survival.apply(not\_null\_count)

#By passing in the axis=1 argument, we can use the DataFrame.apply() method to iterate over rows instead of columns.

def is\_minor(row):

if row["age"] < 18:

return True

else:

return False

minors = titanic\_survival.apply(is\_minor, axis=1)

import pandas as pd

def generate\_age\_label(row):

age = row["age"]

if pd.isnull(age):

return "unknown"

elif age < 18:

return "minor"

else:

return "adult"

age\_group\_survival = titanic\_survival.pivot\_table(index="age\_labels", values="survived")

1. Finding unique values in a column

# Unique values in Major\_category column.

print(all\_ages['Major\_category'].unique())

aa\_cat\_counts = dict()

rg\_cat\_counts = dict()

def calculate\_major\_cat\_totals(df):

cats = df['Major\_category'].unique()

counts\_dictionary = dict()

for c in cats:

major\_df = df[df["Major\_category"] == c]

total = major\_df["Total"].sum()

counts\_dictionary[c] = total

return counts\_dictionary

aa\_cat\_counts = calculate\_major\_cat\_totals(all\_ages)

low\_wage\_percent = 0.0

low\_wage\_jobs\_sum = recent\_grads['Low\_wage\_jobs'].sum()

recent\_grads\_sum = recent\_grads['Total'].sum()

low\_wage\_proportion = low\_wage\_jobs\_sum / recent\_grads\_sum

print(low\_wage\_proportion)

majors = recent\_grads['Major'].unique()

rg\_lower\_count = 0

for m in majors:

recent\_grads\_row = recent\_grads[recent\_grads['Major'] == m]

all\_ages\_row = all\_ages[all\_ages['Major'] == m]

rg\_unemp\_rate = recent\_grads\_row.iloc[0]['Unemployment\_rate']

aa\_unemp\_rate = all\_ages\_row.iloc[0]['Unemployment\_rate']

if rg\_unemp\_rate < aa\_unemp\_rate:

rg\_lower\_count += 1

print(rg\_lower\_count)

1. Slicing the column in pandas series

fandango = pd.read\_csv('fandango\_score\_comparison.csv')

series\_film = fandango['FILM']

print(series\_film[0:5])

series\_rt = fandango['RottenTomatoes']

print(series\_rt[0:5])

1. Using string indexes to find variables (series operation)

# Import the Series object from pandas

from pandas import Series

film\_names = series\_film.values

rt\_scores = series\_rt.values

series\_custom = Series(rt\_scores , index=film\_names)

series\_custom[['Minions (2015)', 'Leviathan (2014)']]

1. Reindexing in pandasfor series

original\_index = series\_custom.index

sorted\_index = sorted(original\_index)

sorted\_by\_index = series\_custom.reindex(sorted\_index)

1. Vectorized operation

criteria\_one = series\_custom > 50

criteria\_two = series\_custom < 75

both\_criteria = series\_custom[criteria\_one & criteria\_two]

1. Pandas dataframe operations

import pandas as pd

fandango = pd.read\_csv('fandango\_score\_comparison.csv')

#print(fandango.head(2))

print(fandango.index)

fandango = pd.read\_csv('fandango\_score\_comparison.csv')

first\_row = 0

last\_row = fandango.shape[0] - 1

first\_last = fandango.iloc[[first\_row, last\_row]]

1. Sorting dataframe in pandas

fandango = pd.read\_csv('fandango\_score\_comparison.csv')

fandango\_films = fandango.set\_index('FILM', drop=False)

print(fandango\_films.index)

movies = ["The Lazarus Effect (2015)", "Gett: The Trial of Viviane Amsalem (2015)", "Mr. Holmes (2015)"]

best\_movies\_ever = fandango\_films.loc[movies]

1. Using apply() function and using only float columns

import numpy as np

# returns the data types as a Series

types = fandango\_films.dtypes

# filter data types to just floats, index attributes returns just column names

float\_columns = types[types.values == 'float64'].index

# use bracket notation to filter columns to just float columns

float\_df = fandango\_films[float\_columns]

# `x` is a Series object representing a column

deviations = float\_df.apply(lambda x: np.std(x))

print(deviations)

#applying apply on dataframes

rt\_mt\_user = float\_df[['RT\_user\_norm', 'Metacritic\_user\_nom']]

rt\_mt\_deviations = rt\_mt\_user.apply(lambda x: np.std(x), axis=1)

print(rt\_mt\_deviations[0:5])

rt\_mt\_user = float\_df[['RT\_user\_norm', 'Metacritic\_user\_nom']]

rt\_mt\_means = rt\_mt\_user.apply(lambda x: np.mean(x), axis=1)

print(rt\_mt\_means[0:5])

1. Extracting values from long string

def age\_count(age\_str):

if pd.isnull(age\_str):

return None

age\_str = age\_str.split(" ")[0]

age\_str = age\_str.replace("+", "")

return int(age\_str)

data["int\_age"] = data["Age"].apply(age\_count)

print(data["int\_age"].describe())

def earn\_count(earnings):

if pd.isnull(earnings):

return None

earnings = earnings.split(" ")[0]

if earnings == "Prefer":

return None

earnings = earnings.replace("$", "")

earnings = earnings.replace(",", "")

return int(earnings)

data["int\_income"] = data["How much total combined money did all members of your HOUSEHOLD earn last year?"].apply(earn\_count)

print(data["int\_income"].describe())