**Numpy**

1. Creating arrays

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

m = np.array([[1,2,3],[4,5,6]]) -> 2\*3 array (m.shape->(2,3))

n= np.arange(0, 30, 2) -> returns evenly spaced values within a given interval.

n = n.reshape(3, 5) -> returns an array with the same data with a new shape(size cannot change).

n.resize(3, 4); changes the shape and size of array in-place.

o = np.linspace(0, 4, 9) -> returns evenly spaced numbers over a specified interval.

np.ones((3, 2)) np.ones((3, 2), int) np.zeros((2, 3)) np.eyes((2, 3)) np.diag(x)

np.array([1, 2, 3] \* 3) np.repeat([1, 2, 3], 3) -> Repeat elements of an array

np.vstack([x, y]) np.hstack([x, y]) -> Combine arrays

1. Operations

x+y x\*y x\*\*2

x.dot(y) -> dot product x.T -> transpose

z.dtype -> see data type of the elements in the array. z = z.astype('f') -> set data type

1. Math functions

a.sum() a.max()/a.min() a.mean() a.std() -> standard deviation

a.argmax()/a.argmin() -> index of max/min value

s = np.random.randint(0,100,1000) -> array of 1000 random int in [0,100)

1. Indexing/Slicing

s[0] s[-1] s[0:5] s[-4:] -> last 4 element s[:]=0 -> set all values to 0

s[2,2] s[3,3:6] s[:2,:-1] -> first two row, all columns except the last one

s[s>30] = 30 -> assigning all values in the array greater than 30 to the value of 30

When slicing arrays and change the value, the original array will also be changed!

r\_copy = r.copy() -> change of copy will not change the original array

1. Iterating over arrays

|  |  |  |  |
| --- | --- | --- | --- |
| for row in s:  print(row) | for i in range(len(test)):  print(test[i]) | for i, row in enumerate(test):  print('row', i, 'is', row) | for i, j in zip(test1, test2):  print(i,'+',j,'=',i+j)  -> iterate over multiple iterables. |

1. Distributions

np.random.binomial(1000, 0.5) np.random.uniform(0, 1) np.random.normal(0.75)

np.random.chisquare(2, size=10000)

Hypothesis testing (from scipy import stats)

T-test: stats.ttest\_ind(early['assignment1\_grade'], late['assignment1\_grade'])

-> Ttest\_indResult(statistic, pvalue)

pvalue larger than critical value α(0.05) -> cannot reject null hypothesis

**Pandas**

df = pd.read\_csv('olympics.csv')

**Series**

1. Creating Series by list:

animals = ['Tiger', 'Bear', None]

pd.Series(animals)

Or by dictionary:

sports = {'Golf': 'Scotland', 'Sumo': 'Japan', 'Taekwondo': 'South Korea'}

s = pd.Series(sports)

Or: s = pd.Series(['Tiger', 'Bear', 'Moose'], index=['India', 'America', 'Canada'])

1. Querying a Series

s.iloc[3] s.loc[‘Golf’]

1. Adding new value to Series

s.iloc[‘Animal’] = ‘Bears’

new = original.append(plus) -> the original Series doesn’t change, but create a new Series with original appending plus

Magic function in Jupter Notebook to calculate how low the code runs:

%%timeit -n 100 -> run 100 times and test running time on average

**DataFrame**

1. Create DataFrame by Series

df = pd.DataFrame([s1,s2,s3], index=['Store1', 'Store1', 'Store2'])

1. Querying a DataFrame

**Row-based**: loc[] -> base on label; iloc[] -> based on position

df.loc['Store 2'] df[‘Cost’] df[‘Cost’].count -> count the number of data exist (not NA)

df.loc[‘Store1’, ‘Cost’] or df.loc[‘Store 1’][‘Cost’]

**Column-based**: [‘Cost’]

multiple columns- -> df.loc[:,[‘Name’,‘Cost’]

**Boolean masking:**

df['Gold'] > 0 -> return boolean Series

1. only\_gold = df.where(df['Gold'] > 0) -> those fails the condition becomes NA

clean NA: only\_gold = only\_gold.dropna()

1. only\_gold = df[df['Gold'] > 0] -> automatically clean

only\_gold = df[(df['Gold'] > 0) | (df['Gold.1'] > 0)]

1. len(df[(df['Gold'] > 0) | (df['Gold.1'] > 0)])
2. Select columns needed
3. cost = df[‘Cost’] cost+=2 -> df will also change

df[‘Cost’]\*=0.8 -> change all values in a column

1. cols = ['SUMLEV','CTYNAME'] + ["POPESTIMATE20"+ str(x) for x in range(10,16)]

df = df[cols]

1. Drop data

df.drop('Store 1') -> result is a new DataFrame, original one is intact

df.drop(df[df['Quantity'] == 0].index)

del df['Name'] -> affect the original DataFrame

1. Add new data
2. Add a new column

df['Location'] = 0 -> Add new column, set default data

df[‘Date’] = ['December 1’, None, 'mid-May'] -> the length should be the same as rows

1. Add new data

df.append(pd.Series(data={'Item': 'Bird Seed', 'Cost': 3.00}, name=('Store 2','Kevyn')))

df = df.reset\_index() df['Date'] = pd.Series({0: 'December 1', 2: 'mid-May'}) -> put missing values as NA

1. Replace value

df.loc[country,'HighRenew'] = 1

df = df.replace(['very bad', 'bad', 'poor', 'good', 'very good'], [1, 2, 3, 4, 5])

1. Read DataFram from csv

df = pd.read\_csv('olympics.csv', index\_col=0, skiprows=2) -> use the first column as index, skip the first 2 rows

rename index/column title:

1. df.rename(columns={0: 'Grades'}, inplace=True)
2. df.rename(columns={col:'Gold' + col[4:]}, inplace=True)
3. Indexing

df = df.set\_index(‘Gold’) df = df.set\_index(['STNAME', 'CTYNAME']) -> dual index

df.index.names = (['Location','Name'])

querying a dual index DataFrame: df.loc['Michigan', 'Washtenaw County']

df.loc[ [('Michigan', 'Washtenaw County'), ('Michigan', 'Wayne County')] ]

df = df.reset\_index() -> reset index starting at 0

df['SUMLEV'].unique() -> return arrays of all unique values in a column

1. Missing values

df.fillna(-1) -> fill all N/A values with -1

df.fillna(ffil)/df.fillna(bfil) -> forward/backward filling, need to **sort** DataFrame by index first: df = df.sort\_index() -> sort by index or df = df.sort(‘Cost’) -> sort by column

1. Merging

df = pd.merge(staff\_df, student\_df, how='outer', left\_index=True, right\_index=True)

how = ‘outer’ -> union how = ‘inner’ -> intersection how = ‘left’/’right’

df = pd.merge(staff\_df, student\_df, how='outer', left\_on='Name', right\_on='Name') -> merge based on column

pd.merge(staff\_df, student\_df, how='inner', left\_on=['First Name','Last Name'], right\_on=['First Name','Last Name']) -> multiple columns

1. Pandas Idiom

(df.where(df['SUMLEV']==50)

.dropna()

.set\_index(['STNAME','CTYNAME'])

.rename(columns={'ESTIMATESBASE2010': 'Estimates Base 2010'}))

1. Apply functions

rows = ['POPESTIMATE2010',

'POPESTIMATE2011',

'POPESTIMATE2012']

df.apply(lambda x: np.max(x[rows]), axis=1)

1. groupby
2. df = df.groupby(['Continent']).size(as\_index=’True’) -> set ‘Continent’ as index of output
3. df.groupby('STNAME').agg(sum) -> apply function in each group (all column)

df.groupby('STNAME').agg({'CENSUS2010POP': np.average}) -> apply in specific column

1. (df.set\_index('STNAME').groupby(level=0)['POPESTIMATE2010','POPESTIMATE2011'].

.agg({'avg': np.average, 'sum': np.sum}))

1. Category data

grades = df['Grades'].astype('category', categories=['D', 'D+', 'C-', 'C', 'C+', 'B-', 'B', 'B+', 'A-', 'A', 'A+'], ordered=True) -> cast series to category with order

pd.cut(df[‘avg’], 3, labels=['Small', 'Medium', 'Large']) -> bin data into 3 bins with labels

1. Pivot table

df.pivot\_table(values='(kW)', index='YEAR', columns='Make', aggfunc=[np.mean,np.min], margins=True)

1. Datetime

a.Time stamp: pd.Timestamp('9/1/2016 10:05AM')

b. Period: pd.Period('1/2016')

c. DatetimeIndex: t1 = pd.Series(list('abc'), [pd.Timestamp('2016-09-01'), pd.Timestamp('2016-09-02'), pd.Timestamp('2016-09-03')])

d. PeriodIndex: t2 = pd.Series(list('def'), [pd.Period('2016-09'), pd.Period('2016-10'), pd.Period('2016-11')])

e. Convert to Datetime: ts3.index = pd.to\_datetime(ts3.index)

f. Time deltas: pd.Timestamp('9/3/2016')-pd.Timestamp('9/1/2016')

pd.Timestamp('9/2/2016 8:10AM') + pd.Timedelta('12D 3H')

g. Create DatetimeIndex: dates = pd.date\_range('10-01-2016', periods=9, freq='2W-SUN')

Check weeday: df.index.weekday\_name

Compute difference: df.diff()