Table of Contents

[M1\_01\_Intro\_Python 4](#_Toc23761813)

[Functions 4](#_Toc23761814)

[Strings 4](#_Toc23761815)

[Indexing and Slicing 5](#_Toc23761816)

[Skip Slicing 5](#_Toc23761817)

[Immutable 5](#_Toc23761818)

[Operators + and \* 5](#_Toc23761819)

[Membership 5](#_Toc23761820)

[Iterating 5](#_Toc23761821)

[Other methods 5](#_Toc23761822)

[Formatting 5](#_Toc23761823)

[Out of range index 5](#_Toc23761824)

[Lists 5](#_Toc23761825)

[Indexing 5](#_Toc23761826)

[Slicing 5](#_Toc23761827)

[Skip Slicing 5](#_Toc23761828)

[Modifying 5](#_Toc23761829)

[Operators + and \* 6](#_Toc23761830)

[Removing items 6](#_Toc23761831)

[Other methods 6](#_Toc23761832)

[Membership 6](#_Toc23761833)

[Iterating 6](#_Toc23761834)

[Range() 6](#_Toc23761835)

[Construction(Stop) 6](#_Toc23761836)

[Construction(Start, Stop) 6](#_Toc23761837)

[Construction(Start, Stop, Step) 6](#_Toc23761838)

[Iterating using range 6](#_Toc23761839)

[Zip()/unzip 6](#_Toc23761840)

[Tuples 7](#_Toc23761841)

[Indexing and slicing 7](#_Toc23761842)

[Operator + and \* 7](#_Toc23761843)

[Count() and Index() 7](#_Toc23761844)

[Membership 7](#_Toc23761845)

[Iterating 7](#_Toc23761846)

[Other functions 7](#_Toc23761847)

[Comparison to lists 7](#_Toc23761848)

[List comprehensions 7](#_Toc23761849)

[Nested Lists 7](#_Toc23761850)

[Dictionary 7](#_Toc23761851)

[Construction 7](#_Toc23761852)

[Adding and modifying items 8](#_Toc23761853)

[Deleting items 8](#_Toc23761854)

[Membership 8](#_Toc23761855)

[Other functions 8](#_Toc23761856)

[Comprehension 8](#_Toc23761857)

[Sets 8](#_Toc23761858)

[Immutable 8](#_Toc23761859)

[No Indexing 8](#_Toc23761860)

[Adding elements 8](#_Toc23761861)

[Removing elements 8](#_Toc23761862)

[Membership 8](#_Toc23761863)

[Union 8](#_Toc23761864)

[Intersection 8](#_Toc23761865)

[Difference 8](#_Toc23761866)

[Symmetric Difference 9](#_Toc23761867)

[Comparison 9](#_Toc23761868)

[Iterating 9](#_Toc23761869)

[Functions 9](#_Toc23761870)

[With default arguments 9](#_Toc23761871)

[With variable arguments 9](#_Toc23761872)

[With keyword arguments 9](#_Toc23761873)

[Anonymous Lambda 9](#_Toc23761874)

[Indirect references 9](#_Toc23761875)

[M1\_02\_PythonModules 9](#_Toc23761876)

[Modules 9](#_Toc23761877)

[Packages 9](#_Toc23761878)

[Import individual modules 9](#_Toc23761879)

[Alternative method of importing submodules 9](#_Toc23761880)

[Import desired function or variable directly 9](#_Toc23761881)

[Import \* from Package 10](#_Toc23761882)

[Pandas lesson 01 basics 10](#_Toc23761883)

[Series 10](#_Toc23761884)

[Numpy 10](#_Toc23761885)

[Membership 10](#_Toc23761886)

[Series from Dictionary Data 10](#_Toc23761887)

[DataFrame 11](#_Toc23761888)

[Construction from dictionary of lists 11](#_Toc23761889)

[Retrieve Columns 11](#_Toc23761890)

[Retrieve Rows 11](#_Toc23761891)

[Add a column 11](#_Toc23761892)

[Delete a column 12](#_Toc23761893)

[Transpose the dataframe 12](#_Toc23761894)

[Construction using nested dictionaries 12](#_Toc23761895)

[Reindexing 12](#_Toc23761896)

[Forward fill NaN values 12](#_Toc23761897)

[Backward fill NaN values 12](#_Toc23761898)

[Dropping entries from an axis 13](#_Toc23761899)

[Indexing, Selection, & Filtering 13](#_Toc23761900)

[Selecting with loc and iloc 14](#_Toc23761901)

[Function application and mapping 14](#_Toc23761902)

[Create new rows using pd.Series 15](#_Toc23761903)

[Create new columns using pd.Series 16](#_Toc23761904)

[Sorting 16](#_Toc23761905)

[Sort by index 16](#_Toc23761906)

[Sort by values 16](#_Toc23761907)

[Axis Indices with Duplicate Labels 17](#_Toc23761908)

[Descriptive Statistics 17](#_Toc23761909)

[Sum 17](#_Toc23761910)

[idxmax, idxmin 17](#_Toc23761911)

[argmax, argmin (Series) 17](#_Toc23761912)

[Accumulations: cumsum, cumprod, cummin, cummax, diff, pct\_change, cov, corr 17](#_Toc23761913)

[Describe 17](#_Toc23761914)

[Matplotlib – plt.scatter 17](#_Toc23761915)

[Unique values & value counts 18](#_Toc23761916)

[Applying Functions (such as value\_counts) to dataframe 19](#_Toc23761917)

[Pandas\_Lesson02\_MultiLevelIndex 19](#_Toc23761918)

[Series MultiLevelIndex Construction from tuples 19](#_Toc23761919)

[Series MultiLevelIndex Construction from dotProduct 20](#_Toc23761920)

[Using tuples as atomic labels 20](#_Toc23761921)

[Pandas\_Lesson03\_MissingValues 21](#_Toc23761922)

[Pandas\_Lesson04\_MergeJoin 21](#_Toc23761923)

[Pandas\_Lesson05\_DataRetrieval 21](#_Toc23761924)

# M1\_01\_Intro\_Python

## Functions

## Strings

### Indexing and Slicing

### Skip Slicing

### Immutable

### Operators + and \*

### Membership

### Iterating

### Other methods

### Formatting

### Out of range index

## Lists

### Indexing

### Slicing

### Skip Slicing

### Modifying

### Operators + and \*

### Removing items

### Other methods

### Membership

### Iterating

## Range()

### Construction(Stop)

### Construction(Start, Stop)

### Construction(Start, Stop, Step)

### Iterating using range

## Zip()/unzip

**print(list(zip(skills, levels))) [('Python', 75), ('Java', 85), ('R', 60)]**

**for skill, level in zip(skills, levels): print(skill, '-->', level)**

**Python --> 75 Java --> 85 R --> 60**

**data = [('Python', 75), ('Java', 85), ('R', 60)]**

**skills, levels = zip(\*data)**

**print(skills)**

**print(levels)**

**('Python', 'Java', 'R')**

**(75, 85, 60)**

## Tuples

### Indexing and slicing

### Operator + and \*

### Count() and Index()

### Membership

### Iterating

### Other functions

### Comparison to lists

## List comprehensions

## Nested Lists

## Dictionary

### Construction

### Adding and modifying items

### Deleting items

### Membership

### Other functions

#### Keys

#### Values

#### Items

### Comprehension

## Sets

### Immutable

### No Indexing

### Adding elements

### Removing elements

### Membership

### Union

### Intersection

### Difference

### Symmetric Difference

### Comparison

### Iterating

## Functions

### With default arguments

### With variable arguments

### With keyword arguments

### Anonymous Lambda

### Indirect references

# M1\_02\_PythonModules

## Modules

## Packages

### Import individual modules

### Alternative method of importing submodules

### Import desired function or variable directly

## Import \* from Package

# Pandas lesson 01 basics

## Series

np.random.seed(123)

scores = np.random.randint(60, 90, 6)

a = pd.Series(scores)

a.values # attribute

a.index # attribute

a[[1, 4]] # extract multiple scores use a list

b = pd.Series(scores, index = ['Alice', 'Bob', 'Charlie', 'Dave', 'Ed', 'Fred'])

b['Bob']

b[['Bob', 'Ed']]

b.describe()

### Numpy

np.cumsum(b) # cumulative sum

np.average(b) # mean

b.std() # standard deviation

### Membership

## Series from Dictionary Data

d = pd.Series({'R': 60, 'Python': 75, 'Java': 50}, # Series from dictionary including indices

index=['Java', 'Python', 'R', 'C++'])

pd.isnull(d) # Determine if an entry is NaN

pd.notnull(d)

(c + d).dropna() # Remove all NaN items from the series

# DataFrame

## Construction from dictionary of lists

data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada', 'Nevada'], # key=str, value=list

'year': [2000, 2001, 2002, 2001, 2002, 2003],

'pop': [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}

df1 = pd.DataFrame(data) # Create dataframe from dictionary of lists

df1 = pd.DataFrame(data, columns = ['year', 'state', 'pop']) # Create dataframe and specify order of columns

df2 = pd.DataFrame(data, columns = ['year', 'state', 'pop', 'debt']) # Fills in debt column with NaN because it is not in data

df2.columns

df2.values

df2.index

## Retrieve Columns

df2['year']

df2.year

df2[['year', 'state']]

## Retrieve Rows

df2 = pd.DataFrame(data, columns = ['year', 'state', 'pop', 'debt'])

df2.iloc[2] # Extract SERIES at row 2 of the dataframe

df2.iloc[[2]] # Extract DATAFRAME at row df2.index = ['one', 'two', 'three', 'four', 'five', 'six'] # Set the dataframe index

df2.loc['two']

df2.loc[['two','five']]

df2['debt'] = 20.5 # Fills entire column

df2.shape

### Add a column

df2['east'] = df2.state == 'Ohio'

### Delete a column

del df2['east']

### Transpose the dataframe

df2.T

## Construction using nested dictionaries

pop = {'Nevada': {2001: 2.4, 2002: 2.9}, # key=str, value=dictionary(key=float, value=float)

'Ohio': {2000: 1.5, 2001: 1.7, 2002: 3.6}}

df3 = pd.DataFrame(pop)

## Reindexing

df1 = pd.Series([4.5, 7.2, -5.3, 3.6], index=['d', 'b', 'a', 'c'])

df2 = df1.reindex(['a', 'b', 'c', 'd', 'e']) # e gets filled with NaN

df3 = pd.Series(['blue', 'purple', 'yellow'], index=[0, 2, 4])

df3.reindex(np.arange(6)) # indices 1,3,5 get filled with NaN

### Forward fill NaN values

df3.reindex(np.arange(6), method='ffill') # blue, blue, purple, purple, yellow, yellow

### Backward fill NaN values

df3.reindex(np.arange(6), method='bfill') # NaN, yellow, yellow, purple, purple, blue

df4 = pd.DataFrame(np.arange(9).reshape((3, 3)),

index=['a', 'c', 'd'],

columns=['Ohio', 'Texas', 'California'])

df4.reindex(['a', 'b', 'c', 'd']) # Row ‘b’ gets filled with NaN values

df4.reindex(columns = ['Utah', 'Ohio', 'Texas']) # Column ‘Utah’ gets filled with NaN values

## Dropping entries from an axis

df1 = pd.Series(np.arange(5), index=['a', 'b', 'c', 'd', 'e'])

df1.drop('b')

df1.drop(['a', 'c'])

df2 = pd.DataFrame(np.arange(16).reshape((4, 4)),

index=['Ohio', 'Colorado', 'Utah', 'New York'],

columns=['one', 'two', 'three', 'four'])

df2.drop('Ohio') # default axis is ‘rows’

df2.drop(['Colorado', 'Ohio'])

df2.drop('two', axis='columns')

df2.drop(['two', 'four'], axis=1) # axis=’columns’

df2.drop(['two', 'four'], axis=1, inplace = True) # axis=1 -> column

## Indexing, Selection, & Filtering

df1 = pd.Series(np.arange(10,14), index=['a', 'b', 'c', 'd'])

df1['c']

df1[2]

df1[1:3] # exclusive endpoint WARNING

df1['b':'d'] # inclusive endpoint WARNING

df1[[3,1]]

df1[['d', 'b']]

## Selecting with loc and iloc

df2 = pd.DataFrame(np.arange(16).reshape((4, 4)),

index=['Ohio', 'Colorado', 'Utah', 'New York'],

columns=['one', 'two', 'three', 'four'])

df2.loc['Colorado']

df2.loc['Colorado', ['two', 'four']] # Row=colorado Column=['two', 'four']

df2.iloc[1, [1, 3]] # Row=1, Column=[1,3]

## Function application and mapping

df1 = pd.DataFrame(np.random.randn(4, 3), columns=list('abc'),

index=['Utah', 'Ohio', 'Texas', 'Oregon'])

np.abs(df1)

# WARNING: default is rows but this appears to work on the columns

# Selects each row, within the column, and then perfoms the lambda between the rows

df1.apply(lambda x: x.max() - x.min()) # ptp (peak-to-peak) function

### Create new rows using pd.Series

df1

|  | **a** | **b** | **c** |
| --- | --- | --- | --- |
| Utah | -1.737248 | -1.934010 | 0.779899 |
| Ohio | 1.374090 | 0.204451 | 0.217460 |
| Texas | 0.499522 | -0.218722 | 0.907912 |
| Oregon | 1.761883 | 0.354476 | 1.602138 |

df1.apply(

lambda x: pd.Series([x.min(), x.max()],

index = ['min', 'max']))

|  | **a** | **b** | **c** |
| --- | --- | --- | --- |
| min | -1.737248 | -1.934010 | 0.217460 |
| max | 1.761883 | 0.354476 | 1.602138 |

### Create new columns using pd.Series

df1.apply(

lambda x: pd.Series([x.min(), x.max()],

index = ['min', 'max']),

axis='columns')

| **min** | **max** |
| --- | --- |
| Utah | -1.934010 | 0.779899 |
| Ohio | 0.204451 | 1.374090 |
| Texas | -0.218722 | 0.907912 |
| Oregon | 0.354476 | 1.761883 |

## Sorting

### Sort by index

df.sort\_index() # For series, for dataframe default=’rows’

df.sort\_index(axis='columns')

df.sort\_index(axis='rows')

### Sort by values

df.sort\_values() # For series

df.sort\_values(by ='b', ascending = False)

## Axis Indices with Duplicate Labels

df1.index.is\_unique

## Descriptive Statistics

## Sum

df.sum() # Take the sum of each column and return the results as a series with column indices

# Default axis=’rows’

### idxmax, idxmin

index labels of maximum and minimum values

### argmax, argmin (Series)

index locations of maximum and minimum values for a Series

### Accumulations: cumsum, cumprod, cummin, cummax, diff, pct\_change, cov, corr

### Describe

## Matplotlib – plt.scatter

### Unique values & value counts

0 62

1 62

2 66

3 61

4 63

5 69

6 66

7 61

8 60

9 61

dtype: int64

a.unique()

array([62, 66, 61, 63, 69, 60])

a.value\_counts()

61 3

62 2

66 2

63 1

60 1

69 1

dtype: int64

pd.value\_counts(a.values, sort=False)

66 2

69 1

60 1

61 3

62 2

63 1

dtype: int64

pd.Index(a.unique()).get\_indexer(a)

array([0, 0, 1, 2, 3, 4, 1, 2, 5, 2])

## Applying Functions (such as value\_counts) to dataframe

df.apply(pd.value\_counts)

df2.apply(pd.value\_counts).dropna()

df2.apply(pd.value\_counts).fillna(0)

# Pandas\_Lesson02\_MultiLevelIndex

## Series MultiLevelIndex Construction from tuples

[(student, course) for student in ['Alice', 'Bob', 'Charlie', 'Dave'] \

for course in ['cs1', 'cs2']]

indices = pd.MultiIndex.from\_tuples(tuples, names = ['Student', 'Class'])

indices

MultiIndex([( 'Alice', 'cs1'),

( 'Alice', 'cs2'),

( 'Bob', 'cs1'),

( 'Bob', 'cs2'),

('Charlie', 'cs1'),

('Charlie', 'cs2'),

( 'Dave', 'cs1'),

( 'Dave', 'cs2')],

names=['Student', 'Class'])

np.random.seed(123)

s = pd.Series(np.random.randint(60,80,8), index = indices)

s

Student Class

Alice cs1 73

cs2 62

Bob cs1 62

cs2 66

Charlie cs1 77

cs2 79

Dave cs1 70

cs2 61

dtype: int64

## Series MultiLevelIndex Construction from dotProduct

data = [['Alice','Bob','Charlie','Dave'],

['cs1', 'cs2']]

indices = pd.MultiIndex.from\_product(data,

names=['Student', 'Class'])

indices

MultiIndex([( 'Alice', 'cs1'),

( 'Alice', 'cs2'),

( 'Bob', 'cs1'),

( 'Bob', 'cs2'),

('Charlie', 'cs1'),

('Charlie', 'cs2'),

( 'Dave', 'cs1'),

( 'Dave', 'cs2')],

names=['Student', 'Class'])

np.random.seed(123)

s = pd.Series(np.random.randint(60,80,8), index = indices)

s

## Using tuples as atomic labels

np.random.seed(123)

pd.Series(np.random.randint(60,80,8), index = tuples)

(Alice, cs1) 73

(Alice, cs2) 62

(Bob, cs1) 62

(Bob, cs2) 66

(Charlie, cs1) 77

(Charlie, cs2) 79

(Dave, cs1) 70

(Dave, cs2) 61

dtype: int64

## Use the dot product to create multi-level indices in a series

data = [['Alice','Bob','Charlie','Dave'],

['cs1', 'cs2']]

indices = pd.MultiIndex.from\_product(data

names=['Student', 'Class'])

indices

MultiIndex([( 'Alice', 'cs1'),

( 'Alice', 'cs2'),

( 'Bob', 'cs1'),

( 'Bob', 'cs2'),

('Charlie', 'cs1'),

('Charlie', 'cs2'),

( 'Dave', 'cs1'),

( 'Dave', 'cs2')],

names=['Student', 'Class'])

np.random.seed(123)

s = pd.Series(np.random.randint(60,80,8), index = indices)

s

Student Class

Alice cs1 73

cs2 62

Bob cs1 62

cs2 66

Charlie cs1 77

cs2 79

Dave cs1 70

cs2 61

dtype: int64

## Specifying the index directly to create a multi-level indexed series

np.random.seed(123)

data = [['Alice', 'Alice', 'Bob', 'Bob', 'Charlie', 'Charlie', 'Dave', 'Dave'],

['cs1', 'cs2', 'cs1', 'cs2', 'cs1', 'cs2', 'cs1', 'cs2']]

s = pd.Series(np.random.randint(60,80,8), index = data)

s

Alice cs1 73

cs2 62

Bob cs1 62

cs2 66

Charlie cs1 77

cs2 79

Dave cs1 70

cs2 61

dtype: int64

## Specifying the index directly to create a multi-level indexed dataframe

np.random.seed(123)

data = [['Alice', 'Alice', 'Bob', 'Bob', 'Charlie', 'Charlie', 'Dave', 'Dave'],

['cs1', 'cs2', 'cs1', 'cs2', 'cs1', 'cs2', 'cs1', 'cs2']]

df = pd.DataFrame(np.random.randint(60,80,(8, 4)), index = data,

columns = ['Quiz1', 'Quiz2', 'Quiz3', 'Quiz4'])

df.index.names = ['Student', 'Class']

df

|  | **Quiz1** | **Quiz2** | **Quiz3** | **Quiz4** |
| --- | --- | --- | --- | --- |
| **Student** | **Class** |  |  |  |  |
| Alice | cs1 | 73 | 62 | 62 | 66 |
| cs2 | 77 | 79 | 70 | 61 |
| Bob | cs1 | 60 | 77 | 75 | 69 |
| cs2 | 60 | 74 | 60 | 75 |
| Charlie | cs1 | 79 | 74 | 64 | 60 |
| cs2 | 76 | 64 | 77 | 63 |
| Dave | cs1 | 62 | 67 | 62 | 75 |
| cs2 | 76 | 67 | 69 | 63 |

## Indexing with MultiIndex

df.loc['Bob']

| **Quiz1** | **Quiz2** | **Quiz3** | **Quiz4** |
| --- | --- | --- | --- |
| **Class** |  |  |  |  |
| cs1 | 60 | 77 | 75 | 69 |
| cs2 | 60 | 74 | 60 | 75 |

df.loc['Bob','cs1']

Quiz1 60

Quiz2 77

Quiz3 75

Quiz4 69

Name: cs1, dtype: int64

df.loc[('Bob', 'cs1'), 'Quiz1']

60

df.loc['Bob':'Dave']

|  |  | **Quiz1** | **Quiz2** | **Quiz3** | **Quiz4** |
| --- | --- | --- | --- | --- | --- |
| **Student** | **Class** |  |  |  |  |
| Bob | cs1 | 60 | 77 | 75 | 69 |
| cs2 | 60 | 74 | 60 | 75 |
| Charlie | cs1 | 79 | 74 | 64 | 60 |
| cs2 | 76 | 64 | 77 | 63 |
| Dave | cs1 | 62 | 67 | 62 | 75 |
| cs2 | 76 | 67 | 69 | 63 |

df.loc[('Bob', 'cs2'):('Dave', 'cs1')]

|  |  | **Quiz1** | **Quiz2** | **Quiz3** | **Quiz4** |
| --- | --- | --- | --- | --- | --- |
| **Student** | **Class** |  |  |  |  |
| Bob | cs2 | 60 | 74 | 60 | 75 |
| Charlie | cs1 | 79 | 74 | 64 | 60 |
| cs2 | 76 | 64 | 77 | 63 |
| Dave | cs1 | 62 | 67 | 62 | 75 |

df.loc[ [('Bob', 'cs2'), ('Dave', 'cs1')] ]

|  |  | **Quiz1** | **Quiz2** | **Quiz3** | **Quiz4** |
| --- | --- | --- | --- | --- | --- |
| **Student** | **Class** |  |  |  |  |
| Bob | cs2 | 60 | 74 | 60 | 75 |
| Dave | cs1 | 62 | 67 | 62 |  |

## Unstacking a dataframe

df

|  | **Quiz1** | **Quiz2** | **Quiz3** | **Quiz4** |
| --- | --- | --- | --- | --- |
| **Student** | **Class** |  |  |  |  |
| Alice | cs1 | 73 | 62 | 62 | 66 |
| cs2 | 77 | 79 | 70 | 61 |
| Bob | cs1 | 60 | 77 | 75 | 69 |
| cs2 | 60 | 74 | 60 | 75 |
| Charlie | cs1 | 79 | 74 | 64 | 60 |
| cs2 | 76 | 64 | 77 | 63 |
| Dave | cs1 | 62 | 67 | 62 | 75 |
| cs2 | 76 | 67 | 69 | 63 |

df.columns

Index(['Quiz1', 'Quiz2', 'Quiz3', 'Quiz4'], dtype='object')

df.unstack() *# Note: second level index goes to the columns instead of the rows*

|  | **Quiz1** | | **Quiz2** | | **Quiz3** | | **Quiz4** | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Class** | **cs1** | **cs2** | **cs1** | **cs2** | **cs1** | **cs2** | **cs1** | **cs2** |
| **Student** |  |  |  |  |  |  |  |  |
| Alice | 73 | 77 | 62 | 79 | 62 | 70 | 66 | 61 |
| Bob | 60 | 60 | 77 | 74 | 75 | 60 | 69 | 75 |
| Charlie | 79 | 76 | 74 | 64 | 64 | 77 | 60 | 63 |
| Dave | 62 | 76 | 67 | 67 | 62 | 69 | 75 | 63 |

df.unstack()['Quiz1']

| **Class** | **cs1** | **cs2** |
| --- | --- | --- |
| **Student** |  |  |
| Alice | 73 | 77 |
| Bob | 60 | 60 |
| Charlie | 79 | 76 |
| Dave | 62 | 76 |

df.unstack()['Quiz1', 'cs1']

Student

Alice 73

Bob 60

Charlie 79

Dave 62

Name: (Quiz1, cs1), dtype: int64

## Unstacking a series

s

Alice cs1 73

cs2 62

Bob cs1 62

cs2 66

Charlie cs1 77

cs2 79

Dave cs1 70

cs2 61

dtype: int64

s[:, 'cs1']

Alice 73

Bob 62

Charlie 77

Dave 70

dtype: int64

s['Bob']

cs1 62

cs2 66

dtype: int64

s['Bob']['cs1']

62

s.unstack()

| **cs1** | **cs2** |
| --- | --- |
| Alice | 73 | 62 |
| Bob | 62 | 66 |
| Charlie | 77 | 79 |
| Dave | 70 | 6 |

s.unstack()['cs1']

Alice 73

Bob 62

Charlie 77

Dave 70

Name: cs1, dtype: int64

## Cross Section (xs)

df

|  | **Quiz1** | **Quiz2** | **Quiz3** | **Quiz4** |
| --- | --- | --- | --- | --- |
| **Student** | **Class** |  |  |  |  |
| Alice | cs1 | 73 | 62 | 62 | 66 |
| cs2 | 77 | 79 | 70 | 61 |
| Bob | cs1 | 60 | 77 | 75 | 69 |
| cs2 | 60 | 74 | 60 | 75 |
| Charlie | cs1 | 79 | 74 | 64 | 60 |
| cs2 | 76 | 64 | 77 | 63 |
| Dave | cs1 | 62 | 67 | 62 | 75 |
| cs2 | 76 | 67 | 69 | 63 |

df.xs('cs1', level=’Class’)

|  |
| --- |
|  |
| **Quiz1** | **Quiz2** | **Quiz3** | **Quiz4** |
| **Student** |  |  |  |  |
| Alice | 73 | 62 | 62 | 66 |
| Bob | 60 | 77 | 75 | 69 |
| Charlie | 79 | 74 | 64 | 60 |
| Dave | 62 | 67 | 62 |  |

### Using slice() as a wildcard

df.loc[(slice(None), 'cs1'), :] *# slice(None) means choose all items in ‘Student’ column*

|  | **Quiz1** | **Quiz2** | **Quiz3** | **Quiz4** |
| --- | --- | --- | --- | --- |
| **Student** | **Class** |  |  |  |  |
| Alice | cs1 | 73 | 62 | 62 | 66 |
| Bob | cs1 | 60 | 77 | 75 | 69 |
| Charlie | cs1 | 79 | 74 | 64 | 60 |
| Dave | cs1 | 62 | 67 | 62 | 7 |

df.loc[(slice('Alice','Charlie'), slice(None)), :] *# Slice(None) means choose all items in ‘Class’*

|  | **Quiz1** | **Quiz2** | **Quiz3** | **Quiz4** |
| --- | --- | --- | --- | --- |
| **Student** | **Class** |  |  |  |  |
| Alice | cs1 | 73 | 62 | 62 | 66 |
| cs2 | 77 | 79 | 70 | 61 |
| Bob | cs1 | 60 | 77 | 75 | 69 |
| cs2 | 60 | 74 | 60 | 75 |
| Charlie | cs1 | 79 | 74 | 64 | 60 |
| cs2 | 76 | 64 | 77 | 63 |

merge, join, aggregation (everything done today)

# Pandas\_Lesson03\_MissingValues

# \* Pandas\_Lesson04\_MergeJoin

# \* Pandas\_Lesson05\_DataRetrieval

# \* Pandas\_Lesson06\_Aggregation

# \* Pandas\_Lesson07\_PivotTable

# \* TimeSeries01

# \* CaseStudy\_Marketing01\_ConversionRates

National Electronic Injury Surveillance System (NEISS).

A [cluster](https://ccsg.isr.umich.edu/index.php/resources/advanced-glossary/cluster) of elements sampled at the first stage of selection.

Synonyms: primary sampling unit, (PSU), PSU

<https://www.slideshare.net/USCPSC/neiss-national-electronic-injury-surveillance-system>

**Selecting k best features - sklearn\_preprocessing\_featureSelection01.pdf**

Research Sklearn\_preprocessing\_featureScaling01.ipynb

sklearn\_preprocessing\_labelEncoder01.ipynb - categorical

LDA keeps information about classes, PCA drops class information

Github.com Git%%Hub%%Rules

<https://github.com/mrcorbett/MET_CS677_DataScienceWithPython>

540 git clone --bare . MET\_CS677.git

541 mv MET\_CS677.git/ MET\_CS677\_DataScienceWithPython.git

548 git remote show origin

557 git remote add origin https://github.com/mrcorbett/MET\_CS677\_DataScienceWithPython

558 git checkout -b InitialBranch

559 git status -s

560 git add .

562 git status -s

563 git commit -am "Initial Commit"

[InitialBranch b44c6cd] second commit

1 file changed, 0 insertions(+), 0 deletions(-)

Martins-MacBook-Pro:MET CS677 - Data Science with Python mcorbett$ git config --global user.email marty.corbett@gmail.com

Martins-MacBook-Pro:MET CS677 - Data Science with Python mcorbett$ git branch

\* InitialBranch

Martins-MacBook-Pro:MET CS677 - Data Science with Python mcorbett$ git push origin InitialBranch

Username for 'https://github.com': marty.corbett@gmail.com

Password for 'https://marty.corbett@gmail.com@github.com':

Enumerating objects: 1415, done.

Counting objects: 100% (1415/1415), done.

Delta compression using up to 8 threads

Compressing objects: 100% (1388/1388), done

Writing objects: 55% (782/1415), 308.85 MiB | 2.08 MiB/s

<https://thomas-cokelaer.info/blog/2018/02/git-how-to-remove-a-big-file-wrongly-committed/>

**git filter-branch** --tree-filter 'rm -rf path/to/your/file' HEAD

**git push**