Python Tutorial 5: Pandas

1. Creating dataframes  
   a. From an ndarray

b. From a dictionary

2. Exploring your dataframe

a. info()

b. describe()

c. value\_counts()

d. head, tail, sample

e. other descriptive statistics

3. Indexing, subsetting and slicing

First Think To Do is do info()

Well hell

getting values df.loc[by label]

getting values df.loc[0]

index to rows is the column names

df.iloc[0].index

df["a"] + df["b"]

### adding columns

referring to a series

df[" "]

df.loc["Data1":"Date2", 'col1':'col2']

Selecting by row label (index) and column label with loc

Selecting by positional index with .iloc()

df.iloc[2:4, 2:4]

df['state'] = ['Alabama', 'Alaska', 'Arizona'] \* 10

df.set\_index()

df.reset\_index()

# Multiindex

df = df.set\_index(['state', 'index'])

df.head()

df.loc['Alabama'].head(10)

df.loc[('Alabama', '2015-01-31-)]

##### read files

pd.read\_ and hit tab

df.shape

df.columns

# boolean masks

mask = df['chlorides'] <= 0.9

type(mask)

# df[(df['a'] == 1) & (df['b'] <= 0.5)]

# df[(df['a'] == 1) & (df['b'] <= 0.5)]['column A', 'column B']

# df.sort\_values('column')

# df.groupby('column C')

g = df.groupby('column D')

g.max()

g.max()['column D']

df2 = df.groupby(['pH', 'quality']).size()

# computing math (create new column with division)

df.drop (axis=1)

df.fillna(0, inplace = True)

df.dropna() will drop all rows with any negative

# Merge

(inner, outer, etc)

# Concatenation

# Value counts with multiple columns

df[['Outlook','Result']].apply(lambda x: x.value\_counts())

pd.DataFrame( [df['Outlook'].value\_counts(),

df['Result'].value\_counts()] ).T #glue 'em together'

# crosstab etc

# plotting values

df.[['col A', 'col\_b']].plot(kind='box')

df.hist(['col A', 'col\_b']], bins=5)

df.plot('Temperature', 'Humidity', kind='scatter');

# we can iterate through the groups with the same Outlook

groups=df.groupby('Outlook')

for name, group in groups:

print(name)

# we can plot multiple times on the same plot, so let's use this group iteration

# to make three overlapping scatter plots:

fig, ax = plt.subplots()

ax.margins(0.05)

for name, group in groups:

ax.plot(group.Temperature, group.Humidity, marker='o', linestyle='', ms=12, label=name)

ax.legend(numpoints=1, loc='lower right')

plt.show()

time\_index = pd.date\_range('2015-01-01', '2017-01-01', freq='m')

df = df.DataFrame(np.random.randn(30,5), index=time\_index)

list\_months = ['Jan', 'Feb', 'Mar' ...]

df['Month'] = list\_months \* 5

#### set index

df.info() # will return list of column names, their type (int, obect or float)

df.describe() # will return count and distribution information about the data for entire dataset

df['A'].describe() # will return count and distrib. info for column A

df.head() # prints the firtst 5 rows of the dataframe

df.head(20) #prints out the first 20 lines of code

df.tail() #prints out the last 5 lines (can also be used to get more)

# using ATTRIBUTES of DataFrame (versus methods)

df.shape #returns a tuple with number of rows and number of columns in the dataframe

all\_this\_col = df['this\_col'].unique

print("this\_col has the following unique entries: ", all\_this\_col)

df['categorical\_column'].value\_counts()

# to look at two columns and compare counts

pd.DataFrame( [df['col\_1'].value\_counts(),

df['col\_2'].value\_counts()]).T #Note the .T glues them together

# to look at bivariate relationships between 2 variables

pd.crosstab(df['col\_1'], df['col\_2'])

# to get percentages in these relationships apply lambda function

# if you want row-wise percentages

pd.crosstab(df['col\_1'], df['col\_2']).apply(lambda r: r/r.sum(), axis=1)

### if you want column-wise percentages

pd.crosstab(df['col\_1'], df['col\_2']).apply(lambda c: c/c.sum(), axis=0)

### CHANGING COLUMN NAMES

### Change one column name

df2 = df.rename(columns= {'old\_col\_name\_1': 'new\_col\_name\_1'}) #this changes one column name and is a dictionary

### Changing all column names

### can create a dictionary to change the names in one fell swoop

### to get column names in the dataframe, this is an attribute

old\_names = df.columns #This is an attribute and not a method

new\_names = ['col\_1\_new\_name', 'col\_2\_new\_name', 'col\_3\_new\_name'] #length must equal the same length as old\_names

name\_change\_dict = dict(zip(old\_names, new\_names)) #this creates a ditcionary

df2 = df.rename(columns = name\_change\_dict) #passing the dictionary into the rename() method

### doing this inplace

df.rename(columns = name\_change\_dict, inplace = True) #renames and keeps changes in the existing dataframe

## ADDING COLUMNS BY CALCULATION

df['new\_column\_calculated'] = df['col\_name\_1'] / df['col\_name\_2']

### NOTE: can use +, -, #, /, \*\* or any operation here since the series in

### each column is basically an np.array

### For example could use np.sin(df['col\_name\_1])

### complex example from class

df['new\_col\_abs\_val\_a\_to\_power\_b'] abs(df['a'])\*\*df['b']

#another example from the readings

df['E'] = np.sqrt(df['A'])

### can use AND as'&', OR as '|' and NOT as '~'

mask = df['col\_name\_1'] == 2011 & df['col\_name\_2'] == 'String\_in\_col\_2'

mask = df['col\_name\_1'] == 2011 | df['col\_name\_2'] == 'String\_in\_col\_2'

df\_2 = df(mask) # creating a new dataframe

df(mask, inplace=True) # returns the original dataframe (we will losse information here)

### or the shorthand one step approach

df\_2 = df[df['col\_name\_1'] == 2011 & df['col\_name\_2'] == 'String\_in\_col\_2']

df\_2 = df[df['col\_name\_1'] == 2011 | df['col\_name\_2'] == 'String\_in\_col\_2']

### example from class

df[(df['chlorides'] >= 0.04) & (df['chlorides'] < 0.08)]

#### .unique()

#### transpose

##### slicing df[0:3]

### using isin() method for filtering

df2 = df.copy()

df2['E'] = ['one', 'one', 'two', 'three', 'four', 'three']

df2

df2[df2['E']].isin(['two', 'four'])

df.apply(np.cumsum)

np.where()