Python - Data structures

```
x = 36  # this is an integer
x = 3.14  # a decimal number
x = True  # Boolean
x = "This is a string"
[ ] # list, mutable sequence
( ) # tuple, immutable
{ } # dictionary, mapping
([ ]) #set
print x[0]  #print first element
top_teams.ix[['NYA','PIT']] #look up index
```

Python - basics

```
%time # tell you how long something takes to run
\ #break line
```

list methods:

```
l=[]
l.append(obj) #add object to end of list
l.count(obj) #returns int nbr of occurences of obj in list
l.index(obj) #returns index of first occurence
l.pop([index]) # returns item at specified index
l.remove(obj) #remove first occurance of obj in list
l.reverse() #reverse list
l.sort() #sort list

range() #creates range of integers
xrange () #creates an iterator object
for i, element in enumerate(l):
  print "Element no", i, "is", element #returns tuple; this is called
tuple unpacking; enumerate function lets you iterate both over the
index and the elements

lambda #functions on the fly
func = lambda x: 1 + .1 * (x - 4) ** 2 + 4 *
np.random.random(len(x))
```

Python- functions

If/else

```
if x > 4:
   print "x"
elif x == 4:
   print "y"
else:
   print "z"
```

For loop

```
l = []  # empty list
for i in range(10):
   l.append(i ** 2)
print l
```

while loop

Function:

```
def func (x):
if x > 4:
   print "x"
elif x == 4:
   print "y"
else:
   print "z"

func(4)

def f(x):
   return 3*x**2 - 2*x – 7
```

Python- slicing

```
print l # whole list
print l[2] #third element
print l[0:3] #elements 1 through 4
print l[:3]# first 3 elements
print l[3:] # elements after and including element 3
print l[-1] #last element in array
print l[-3:] #last three elements in array
print l[:] #same as l, but it creates a copy
print l[:-2]#everything but last two
print l[::2] #every second
print l[::-1] #reverse array
```

numpy

```
import numpy as np
np.log
arr = array([])
arr.shape #shape of an array
convolve(a,b) #linear convolution of two sequences
dot(arr1,arr2) #compute inner product of two arrays
vectorize() #turn scalar function into one accepting/returning
vectors
np.log(train_data['SalaryNormalized']).plot()
```

Pandas examples

```
#add date index
year_salary.index = pd.to_datetime(year_salary.index,
format="%Y")

#loop to display unique values with writing
for col in test_data.columns:
   print "%s has %s unique categories" % (col,
test_data[col].nunique())
```

pandas

```
import pandas as pd
df = pd.read_csv ("path/data.csv") #returns dataframe
df.head() #see top rows of dataframe
len(df) #number of rows
df.describe() #descriptive statistics
df.info() #general info
df.column.value_counts() #count of each value
df.column.unique() #same as value counts
df.column.nunique() # number of unique entries

df[['column']] #select/returns dataframe with column
df['column'] #returns series; same as df.column
df.iloc[label] #select row by label
df.inbox #return dataframe index

df.sort_values('column', ascending=False)
new_df = df.merge(other_df, on='column') #columns must be in
both dataframes
concat() #merge dataframe or series objects
df.drop()#delete row/column
df.dropna() #drop rows where data is missing

df.grouby() # split df by columns and create groupby object
df.mean(); df.median(); df.std(); df.sort()
df.min()/df.max() #return min/max of every column
df.T() #transpose dataframe
df.agg({'column':[mean,std]})
pd.pivot_table(df, values=['column2', 'column4'],
index=['column3'], columns=['column5']) #pivot table

df [['column1','column2','column3']][(df['column5'] >= 100) &
(df['column2']>10)].head() #where/and statement like SQL
df[['column1','column2','column3']].grouby('column2').mean()
#mean for each column grouped by column 2

df.applymap() #apply function to every element in dataframe
df.apply() #apply function along a given axis

df.plot(x='column1',y='column2', kind='scatter')
f = df.column.hist(bins = 20) #histogram df.hist(alpha =.5)
pd.scatter_matrix(data, figsize=(10,10))

df.to_csv('file.csv') #save to csv
read_csv('file.csv') #read csv into dataframe
df.to_excel('file.xlsx', sheet_name) #save to excel
read_excel('file.xlsx','sheet1', index_col = None, na_values
=['NA']) #read excel into dataframe
```

matplotlib

```
from matplotlib import pyplot as plt
%matplotlib inline #% = in juptyer notebook, don't export graphic to file but display in line in notebook
f = plt.plot(l)
plt.plot(x, y, 'r:') #red dotted line; b-- #blue – line
plt.title("title"), plt.xlabel("x-axis"), plt.ylabel("y-axis")
plt.xlim(0, 5), plt.ylim(0, 70) #plot limits
plt.plot(range(6), range(6), 'y*:', markersize=10, label="straight line")#* creates the star markers, s creates square ones
f = plt.legend() #use f to suppress standard title
plt.plot.kde #density

f = plt.scatter(df.column1, df.column2, linewidth=0, alpha=.1)
plt.subplot(n,x,y) #creates multiple plots; n- number of plots, x – number of horizontally displayed, y – vertically displayed
xticks([],[]) #same for yticks; set tick values, first array for values, second for labels
.plot(kind='barh') #horizontal bar plot

savefig('image.png') #save plot
```

Matplotlib examples

```
#horizontal bar plot
df.plot(kind='barh')

#pie chart
plt.pie(Contract_Time, labels=('permanent','contract'),startangle=80)
plt.axis('equal')
plt.show()

#overlaying histograms
bins = np.linspace(1, 100000, 30)
f = plt.hist(y, bins=bins)
f = plt.hist(lasso_2.predict(X), bins=bins)
```

numpy

```
import numpy as np
np.log
arr = array([])
arr.shape #shape of an array
convolve(a,b) #linear convolution of two sequences
dot(arr1,arr2) #compute inner product of two arrays
vectorize() #turn scalar function into one accepting/returning vectors
np.log(train_data['SalaryNormalized']).plot()
```

vincent

```
!pip install vincent
import Vincent as v
s = v.Bar(dataframe)
s.axis_titles(x='Team', y='Salary')
v.Scatter(year_salary)
f = vincent.StackedBar(team_year_salary).legend(title='Teams')
f.width, f.height = 800, 500
```

seaborn

```
!pip install seaborn
import seaborn as sns
sns.heatmap(data[data.columns[:10]]['column'], annot=True)
sb.lmplot('column1','column2', df) #create trendline
f = sb.lmplot(x = 'yd', y='sl', data=data, ci=95)
f = sb.lmplot(x = 'yr', y='sl', col='rk', row='sx', data=data)
sb.boxplot (x='column1',y='column2',data)#similar violinplot
sb.factorplot(x='yearID', y='HR', col='teamID', col_wrap=5, data=data)
```

Clean data – titanic example

```
#clean up Sex into boolean 0 and 1
train_data['Sex'] = train_data['Sex'].map( {'female': 0, 'male': 1} ).astype(int)

# All the missing Fares -> assume median of their respective class
if len(train_data.Fare[train_data.Fare.isnull() ]) > 0:
    median_fare = np.zeros(3)
    for f in range(0,3):                                  # loop 0 to 2
        median_fare[f] = train_data[train_data.Pclass == f+1 ]['Fare'].dropna().median()
    for f in range(0,3):                                  # loop 0 to 2
        train_data.loc[ (train_data.Fare.isnull()) & (train_data.Pclass == f+1 ), 'Fare'] = median_fare[f]

# All missing Embarked -> just make them embark from most common place
if len(train_data.Embarked[ train_data.Embarked.isnull() ]) > 0:
    train_data.Embarked[ train_data.Embarked.isnull() ] = train_data.Embarked.dropna().mode().values
Ports = list(enumerate(np.unique(train_data['Embarked'])))
Ports_dict = { name : i for i, name in Ports }
train_data.Embarked = train_data.Embarked.map( lambda x: Ports_dict[x]).astype(int)

# All the ages with no data -> make the median of all Ages
median_age = train_data['Age'].dropna().median()
if len(train_data.Age[ train_data.Age.isnull() ]) > 0:
    train_data.loc[ (train_data.Age.isnull()), 'Age'] = median_age

#substitute all NaN with 0
train_data = train_data.fillna(0)

# substitute all cabin numbers to 1, in other words we transformed this column into saying, if the person had a cabin or not
train_data.loc[train_data['Cabin'] > 0, 'Cabin'] = 1
```

Add columns/features – titanic example

```
#add column "married woman", if brackets in name
train_data['married woman'] = train_data.Name.map(lambda x: int(str(x).lower().find('(') > -1) if x is not None else None)

#add dependents column - if you were alone on the boat = 0
train_data['dependents'] = train_data.SibSp + train_data.Parch

#want to establish their richness level
train_data['richness'] = train_data.Fare/train_data.Pclass

#add log column of column
richness2 = np.log(np.sqrt(train_data[['richness']]))

#add log square root column
age2 = np.log(np.sqrt(train_data[['Age']]))
```

Model

```
Import model from sklearn
X = data[features]  # put our features in a separate matrix, this is the 'input'
y = data.species  # these are the labels to predict/ response
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=.7, random_state=42) #cross validation: 70% of data for training; random state 42 or 123
model = model()
model.fit(X_train, y_train)
train_accuracy = model.score(X_train, y_train)
test_accuracy = model.score(X_test, y_test)
y_prediction  = model.predict(x_test)
preds = pd.DataFrame(y_prediction, columns=['Salary'])
preds.to_csv('/Users/username/Desktop/file.csv')
```

Cross Validation

```
from sklearn.cross_validation import KFold
cv = KFold(len(data), n_folds=5, shuffle=True)
#Provides train/test indices to split data in train test sets. Split dataset into k consecutive folds

cross_scores = []
for k in xrange(1, 10):
   model = model()
   cross_score = cross_val_score(model, X, y, cv=5)
   cross_scores.append(cross_score.mean())
plt.plot(cross_scores)

#model performance
for metric in ["accuracy", "precision", "recall", "f1", "roc_auc"]:
   print "%-10s: %.4f" % (metric, cross_val_score(model, X, y, cv=10, scoring=metric).mean())

from sklearn.metrics import mean_absolute_error
mean_absolute_error(data.y, data.y_pred)
#MAE = np.mean(abs(data.y - data.y_pred))
mean_squared_error(data.y, data.y_pred)
#MSA = np.square(data.y_pred - data.y).mean()

print -cross_val_score(lasso_2, X, y, cv=10, scoring="mean_absolute_error").mean()
print -np.median(cross_val_score(lasso_2, X, y, cv=10, scoring="median_absolute_error"))

#compare models
models = [('RFC', RandomForestClassifier(n_estimators=200))
   ,('GBC',GradientBoostingClassifier(n_estimators=100))
   ,('DTC',DecisionTreeClassifier())
   ,('ABC',AdaBoostClassifier())]
results = {}
for model, clf in models:
   score = cross_val_score(clf, X2, Y_target).mean()
   results[model] = score
```

Linear Regression

regression (supervised and continuous): functional/linear relationship between input variable x and response variable y → fit the regression model to the dataset by minimizing the sum of the squared residuals (OLS algorithm = Ordinary Least Squares: distance of data point to best fit line)

```
from sklearn.linear_model import LinearRegression
model = LinearRegression() #create linear regression object
model.fit(x_train, y_train) #train model on train data
model.score(x_train, y_train) #check score

print ('Coefficient: \n', model.coef_)
print ('Intercept: \n', model.intercept_)
coefs = zip(model.coef_, X.columns)
model.__dict__
print "sl = %.1f + " % model.intercept_ + \
   " + ".join("%.1f %s" % coef for coef in coefs) #linear model
```

Regularization

```
from sklearn.linear_model import Ridge, Lasso, RidgeCV
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.pipeline import make_pipeline

lasso_model = Lasso(alpha=.01) # imposes L1 prior on coefficient (→ many coeffiecients become zero); the higher the alphoa the more you penalize coefficients
ridge_model = Ridge(alpha = 2.0)
#Ridge regression imposes L2 prior on coefficient → outliers to be less likely and coeffiecients to be small across the board
ridgeCV_model = RidgeCV  (alphas=[0.1, 1.0, 10.0]) #several alphas at once for comparison

model = make_pipeline(StandardScaler(), PolynomialFeatures(degree), test_model)
make_pipeline # chain multiple estimators into one
PolynomialFeatures(degree)#Generate a new feature matrix consisting of all polynomial combinations of the features with specified degree
StandardScaler()#Standardize features by removing the mean and scaling to unit variance
```

Categorical features

```
from patsy import dmatrices # patsy provides R formula syntax
y, X = dmatrices('y ~ column1 + column2 + column3', data=df, return_type='dataframe') #add columns for each category 0/1

+ #is not addition operator but separator between variables
: #adds the interaction of two variables.
* #adds the original terms as well as their interaction effect
C() #make categorical
```

Text features – converting text to vectors

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer, DictVectorizer

#most used words
words = pd.Series([word for line in data.column.values for word in line.lower().split()]).value_counts()
words.head(20)

#how many times a word is included
data['word'] = data.column.map(lambda x: x.find('word') > -1)
data.great.value_counts()

#specify stopwords
additional_stop_words = ['word','word']
my_stop_words = text.ENGLISH_STOP_WORDS.union(additional_stop_words)

cv = CountVectorizer(stop_words = my_stop_words, ngram_range=(1,2), max_features=5, min_df=.10, max_df=.95)
#only use what appears at least some times, but not too often
#class transforms an array-like (list, dataframe column, array) of strings into a matrix where each column represents a token (word or phrase) and each row represents the sample

tv = TfidfVectorizer(stop_words = my_stop_words, ngram_range=(1,2), max_features=5, min_df=.10, max_df=.95)
#Term Frequency is simply the number of times that a word appear in a sample

X, y = cv.fit_transform(data.text).todense(), data.score

#for each line see what words are present true/false
pd.concat([pd.DataFrame(X, columns=cv.get_feature_names()), data.text], axis=1).head()

 #for model check which words are positive vs. negative
coef = pd.Series(model.coef_, index=cv.get_feature_names())
coef.sort()
top = 15
fig, axes = plt.subplots(1, 2, figsize=(12, 4))
f = coef[:top].plot(kind='barh', ax=axes[0])
f = coef[-top:].plot(kind='barh', ax=axes[1])
print "Negative:", ", ".join(coef[:top].index)
print "Positive:", ", ".join(coef[-top:].index)

#
dv = DictVectorizer(sparse=False)
```

statsmodels

```
import statsmodels.formula.api as sm
# investigating results of a model & integrates with patsy
model = sm.ols(formula="y ~ column1+ column2", data=df).fit()
model.summary()
```

knn

knn (k nearest neighbors algorithm): Euclidean distance, can be used for both classification and regression; instance based learning; lazy learning; might need to rescale data to make dimensions comparable

```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=5)
```

loop to go through different n_neighbors options:

```
scores = []
for k in xrange(1,100):
   model = KNeighborsClassifier(n_neighbors=k)
   model.fit(X_train, y_train)
   score = model.score(X_test, y_test)
   scores.append(score)
plt.plot(scores)
```

Logistic Regression

dependent variable is categorical

```
from sklearn.linear_model import LogisticRegression
```

decision boundary:

```
x1, x2 = features
colors = list("rby")

# Plot the flowers with color labels
for spec in data.species.unique():
   data_spec = data[data.species == spec]
   plt.scatter(data_spec[x1], data_spec[x2], label=spec,
c=colors.pop(),
            linewidths=0, s=100, alpha=.4)

# draw the decision boundary
boundary_x1 = np.array([data[x1].min() - .3, data[x1].max() + .3])
boundary_x2 = -(model.intercept_ + model.coef_[0, 0] *
boundary_x1) / model.coef_[0, 1]
f = plt.plot(boundary_x1, boundary_x2, ':')
f = plt.legend(loc="upper left", bbox_to_anchor=(1,1))
f = plt.xlabel(x1), plt.ylabel(x2)
```

Naïve Bayes

*P(A|B) = (P(B|A)*P(A))/P(B) (you can swap conditional probabilities); can be easily updated in real time; optimize either P(A|B) or P(A|B)P(A) - update our beliefs based on new evidence; MLE (maximum likelihood estimator) finds the parameters that make the data most likely*

```
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
```

Decision Tree – Random Forest

non-parametric hierarchical; nodes represent questions ("test conditions") and edges are answers to these questions (edges lead from parent node to child node starting from root node and ending in leaf nodes = class labels) → binary yes/no answers; gild (grow) a decision tree = Hunt's algorithm (greedy recursive algorithm →goal highest possible purity: Entropy (information gain) coefficient, Gini coefficient, Misclassification Error)

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
model = DecisionTreeClassifier (criterion ='gini') # single tree
#here you can change the algorithm as entropy or gini
model = RandomForestClassifier(n_estimators=20, n_jobs=6)

#important features
model.feature_importances_
from sklearn.ensemble import ExtraTreesClassifier
etc_model = ExtraTreesClassifier()
etc_model.fit(X, Y_target)
df = pd.DataFrame(etc_model.feature_importances_, index =
X.columns.values, columns =['importance'])

#optimize model
%time
scores = []
for n in [1, 2, 3, 5, 10, 20, 50, 100, 200, 300]:
   for criterion in ['gini', 'entropy']:
      start_time = time.time()  # let's time it, to see how long
running a forest takes
      r_model = RandomForestClassifier(n_estimators=n,
criterion=criterion)
      accuracy = cross_val_score(r_model, X, Y_target).mean()  #
out of sample accuracy
      duration = time.time() - start_time
      scores.append(dict(n_estimators=n, criterion=criterion,
accuracy=accuracy, duration=duration))
scores = pd.DataFrame(scores)
```

Boosting

Boosting (convert weak learners to strong learners) - classifier for classification and regressor for regression
Ada Boost; Gradient Boosting; XGBoost

```
From sklearn.ensemble import GradientBoostingClassifier,
AdaBoostClassifier

model = AdaBoostClassifier(n_estimators=10)
%time cross_val_score(model, X, data.target).mean()

X_array = X.toarray()  # Covert to dense matrix (model won't
accept sparse matrices)
model = GradientBoostingClassifier(n_estimators=k)
```

SVM (Support Vector Machine)

decision boundary that makes the most sense based on analytic geometry; generalization error = margin (area around line without points) → margin depends only on subset of training data nearest to line → create line with largest margin (maximum margin hyperplane) → convex objective function - take derivative and equal to zero to find max

```
from sklearn import svm
model = svm.SVC() #create SVM classification object
```

k-means

```
From sklearn.closter import KMeans
k_means = KMeans (n_clusters=3, random_state = 0)
model.fit(X)
predicted = model.predict(x_test)
```

Content Filtering

fff

Collaborative Filtering

fff

terminal

cd #open directory (or get toHome)
mkdir # make directory
pwd # print working directory
ls # list what is in your directory
command + shift + H # Home directory
cp
mv
python
jupyter notebook
touch
echo
chmod
clear # clear terminal
control + c #kill
cat # top lines of file
less # preview file (use q to exit)
| # use output of one prog as input for other

tar -czvf msongs.tgz ~/Downloads/MSongsDB #zip file
tar -xzvf msongs.tgz #unzip file

df –h #see usage/free space

Github (terminal)

```
git init #create empty repository in directory
git status #check status of repository
git add filename.txt #add file to staging to start
tracking changes (Changes to be commited)
git commit –m "comment" #store staged changes with
message explaining them
git add '*.txt' #add all the text files to staging
git add . #add everything
git log #log of all changes you have made
git reset --hard origin/master #undo local comits
```

Python - sqlite3

import sqlite3
conn= sqlite3.connect('mxm_dataset.db') #connect to database
c = conn.cursor()   # from that connection, get a cursor to do queries

#list tables in database
q = "SELECT name FROM sqlite_master WHERE type='table' ORDER BY name"
res = c.execute(q)
print res.fetchall()

conn.close() #close the connection

# list column names
q = "SELECT sql FROM sqlite_master WHERE tbl_name = 'songs' AND type = 'table'"
res = c.execute(q)
print res.fetchall()[0][0]

write sqlite database as csv:
import pandas.io.sql as sql
table = sql.read_sql('select * from some_table', con)
table.to_csv('output.csv')

AWS – launch instance

Launch instance
Start Instance (wait for green light: running)
Connect to instance

In terminal: (not in remote server instance)
To run instance: ssh -i "key.pem" ubuntu@ec2-XXX-XX-XX-XXX.compute-1.amazonaws.com (copy this from the instance after clicking connect)

To add file to instance: scp –i "key.pem" file.csv ubuntu@ec2-XXX-XX-XX-XXX.compute-1.amazonaws.com:~

To save file from instance locally: scp -i "key.pem" -r ec2-user@ec2-52-201-215-27.compute-1.amazonaws.com:output_2.csv ~/Desktop

AWS – in instance

install anaconda:
wget http://repo.continuum.io/archive/Anaconda2-4.0.0-Linux-x86_64.sh
source ~/.bashrc
export PATH="/home/ubuntu/anaconda2/bin:$PATH"
bash Anaconda2-4.0.0-Linux-x86_64.sh

jupyter notebook in AWS:
screen -S notebook #in Amazon Server terminal create new terminal
jupyter notebook #in new terminal in Amazon server terminal
Control + A
Control + D
exit
ssh -i ~/Downloads/key.pem -L8889/localhost/8888 ubuntu@ec2-107-21-39-180.compute-1.amazonaws.com #make sure you have the up-to-date instance
→ in your local browser enter: localhost:8889

mount volume (non-empty w file system):
lsblk #view your available disk devices and their mount points
sudo file -s /dev/xvdf #check if there is a filesystem
sudo mkdir mount_point #create directory
sudo mount /dev/xvdf mount_point #mount

Web scraping

!pip install xmltodict
from bs4 import BeautifulSoup
import requests
import xmltodict
import os
import webbrowser

requests + beautiful soup:
```
def browse(soup):
  """ Browse current HTML. """
  if str(type(soup)) != "<class 'bs4.BeautifulSoup'>":
    soup = BeautifulSoup(soup.content, 'lxml')
  path = os.path.abspath('browse.html')
  url = 'file://' + path
  with open(path, 'w') as f:
    f.write(soup.encode('ascii','ignore'))
  return webbrowser.open(url)
res = requests.get(url)
res.text[:1000]
soup = BeautifulSoup(res.text)
```

Wikipedia tables:
!pip install html5lib
import html5lib
import pandas as pd
url ='https://en.wikipedia.org/wiki/XXX'
tables = pd.read_html(url)

requests:
search_string = raw_input()
headers = {'X-Requested-With': 'XMLHttpRequest'}
data = {'searchRequestJson': #enter from website'}
res = requests.post(url, data=data, headers=headers)
result_dict = xmltodict.parse(res.text)
df = pd.DataFrame(result_dict['result']['requisition'])
df.head()