Data analysis in python with pandas series courses

Useful websites:

https://github.com/justmarkham/pandas-videos # data file download

http://pandas.pydata.org/pandas-docs/stable/whatsnew.html #pandas release notes

https://www.youtube.com/user/dataschool # data school courses

1. website: pandas.pydata.org # tutorial

2. read file: pd.read_table(), pd.read_csv(), pd.read_excel()

```
user_cols = ['id', 'age', 'gender', 'occupation'] #select displayed columns
users = pd.read_table('file', header=None, names = user_cols)
```

3. select series:

```
columns: ufo['City'], ufo.City, ufo['Colors Reported']
ufo['Location'] = ufo.City + ',' + ufo.State #create a new column use the bracket
```

4. parentheses: type(movies), movies.duration.value_counts() # with parentheses

```
movies.dtypes, movies.shape #without parentheses
```

5. rename columns:

```
ufo.columns # show original columns
ufo.rename(columns = {'Colors Reported':'Colors_Reported'}, inplace = True) # dictionary
ufo_cols = ['City','Color','shape','state',time], ufo.columns=ufo.cols # type all columns
ufo.columns = ufo.columns.str.replace(' ','_') #less typing, use '_' replace space
```

6. remove columns:

```
ufo.drop('Colors reported', axis = 1,inplace=True) # axis=0 row, axis=1 column
ufo.drop(['City','State'],axis =1,inplace=True)
ufo.drop([0,1],axis=0, inplace=True) # drop rows
```

7. sort: movies.title.sort_values() movies['title'].sort_values(ascending=False) # default ascending

```
movies.sort_values('title')
movies.sort_values(['content_rating','duration'])
```

8. filter rows: movies[movies.duration >= 200]

```
movies.loc[movies.duration .>= 200, 'genre']
```

9. multiple filter: movies[(movies.duration >= 200) & (movies.genre == 'Drama')] # parentheses &, |

```
moivesgenre.isin(['Crime','Drama','Action']) # mutiple conditions
```

10. FAQ pd.read_csv(fid, usecols=['City','State']) pd.read_csv(fid, usecols = [0,4]) # read only two columns

```
pd.read_csv(fid, nrows = 3) #read first three rows
for index, row in ufo.iterrows(): print(index, row.City, row.State) # select individual entries
movies.select_dtypes(include=[np.number]) # drop non-numeric columns
```

11. axis: drinks.drop('continent', axis=1).head() #drop column

```
drinks.drop(2, axis =0).head() # drop row
```

drinks.mean(default axis=0) drinks.mean(axis = 'index') # calculate the mean of each column
drinks.mean(axis = 1) drinks.mean(axis = 'columns')

12. string: 'hello'.upper()→HELLO
orders.quality.str.upper() # 'quality' is a column
orders[orders.quality.str.contains('Chicken')] # select rows whose column contain chicken
GOOGLE 'API Reference' --- String handing---function # tutorial
orders.quality.str.replace('[','').str.replace(']','') # drop the bracket in items

13. data type: drinks['quality'] = drinks.quality.astype(float) # int64 to float64
pd.read_csv(fid, dtype = {'quality':float}) # dictionary

14. groupby: drinks.groupby('continent').beer_servings.mean() # all continents
drinks[drinks.continent=='Africa'].beer_servings.mean()
drinks.groupby('continent').beer_servings.agg(['count','mean','max','min'])
drinks.groupby('continent').mean()
drinks.groupby('continent').mean().plot(kind = 'bar') # visualization

15. explore series: movies.genre.describe() # one column
moives.genre.value_counts() # counts
moives.genre.value_counts(normalize=True) # normalization
moives.genre.unique() # all classes of gerne
moives.genre.nunique() # number of classes

16. missing: ufo.isnull() # shows the nan rows (Ture or False) ufo.notnull() # normal is Ture, missing is False
ufo.isnull().sum() #count the number of ture items of each column
ufo[ufo.City.isnull()]
ufo.dropna(how='any') # drop the row of any nan ufo.dropna(how='all')
ufo.dropna(subset=['City','Shape Reported'], how='any')
ufo['Shape Reported'].value_counts(dropna=False)
ufo['Shape Reported'].fillna(value='DISK', inplace=True)

17. index: drinks.index #show index drinks.columns
drinks.loc[23,'beer_servings']
drinks.set_index('country', inplace=True) # country column becomes the index
drinks.index.name = None # hide the index
drinks.reset_index(inplace=True)

18. index2: drinks.continent # Series index is same to DataFrame
drinks.continent.value_counts().sort_index() *.sort_values()
pd.concat([drinks,people], axis=1) # drinks (dataframe) and people (series) with the same index
pd.concat([weather1,weather2],axis=0,ignore_index=True)

19. select multiple rows and columns:
ufo.loc[:,['City','State']] ufo.loc[:,'City':'State'] # by label
ufo.loc[[0:4],:] # inclusive 0 , inclusive 4
ufo.loc[ufo.City=='Oakland',:] #select special rows
ufo.iloc[:,[0,3]] # inclusive 0 , exclusive 3, same to list(range(0,3))
drinks.ix[1,'beer_servings'] drinks.ix['Albania':'Andorra',0:2]

```
ufo.ix[0:2,0:2]        # 3 rows and 2 columns, so it is not recommended.
# remember loc is inclusive and iloc is exclusive.
```

20. inplace:
```
ufo.drop('City',axis=1, inplace=True)   # default inplace=False
ufo.dropna(how='any', inplace=True)   # default inplace=False
ufo = ufo.set_index('Time')   or ufo.set_index('Time',inplace=True)
```

21. category:
```
drinks.info()
drinks.memory_usage(deep=True)
sorted(drinks.continent.unique())   # replace string with integer,step1          smaller and faster
drinks['continent']=drinks.continent.astype('category')       # replace string with integer,step2
drinks.continent.cat.codes      # replace string with integer,step3
drinks.memory_usage(deep=True)        # replace string with integer,step4
df['quality']= df.quality.astype('category',category=['good','excellent'],ordered=True)
df.loc[df.quality > 'good',:]
```

22. scikit-learn:
```
feature_cols = ['Pclass','Parch']
X = traindata.loc[:,feature_cols]
Y = traindata.Survived
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
logreg.fit(X,Y)
X_new = testdata.loc[:,feature_cols]
new_pred_class = logreg.predict(X_new)
pd.DateFrame({'PassengerID':test.PassengerID,'Survived':new_pred_class}).set_index('PassengerID').to_csv('1.csv')
train.to_pickle(train.pkl)              pd.read_pickle('train.pkl')
```

23. FAQ
```
GOOGLE      python drop     find: str.
ufo.sample(n=3,random_state=42)        ufo.sample(frac=0.75,random_state=99)       # randomly sample rose
ufo.loc[~ufo.index.isin(train.index),:]
```

24. dummy variables:
```
traindate['Sex_male'] = traindata.Sex.map({'female':0,'male':1})      # new column, another way to redefine
pd.get_dummies(traindata.Sex, prefix='Sex')       # prefix is used to rename
pd.get_dummies(traindata,columns=['Sex','Embarked'])
```

25. date/time:
```
ufo.Time.str.slice(-5,-3).astype(int)       # 4/18/1993 19:00 string
ufo['Time'] = pd.to_datetime(ufo.Time)
ufo.Time.dt.hour     # API Reference      .dt.*
ufo.loc[ufo.Time >= ts,:]
(ufo.Time.max()-ufo.Time.min()).days
ufo['Year']=ufo.Time.dt.year        ufo.Year.value_counts().sort_index().plot()
```

26. remove duplicate rows:
```
users.duplicated()
users.loc[users.duplicated(),:]
users.drop_duplicates(keep='first')
users.duplicated(subset=['age','sex'])
```

27. SettingWithCopyWarning:

```
top_movies = movies.loc[movies.star_rating>=9,:].copy()   # use copy() to avoid the warning
```

28. display:

```
pd.get_option('display.max_rows')              # the default rows that display
pd.set_option('display.max_rows',None)         #show all rows
pd.reset_option('display.max_rows')
pd.get_option('display.max_columns')
pd.get_option('display.max_colwidth')          #the length of displayed character
pd.set_option('display.max_colwidth',100)
pd.set_option('display.precision', 2)          #display the float number in 2
pd.set_option('display.float_format','{:,}'.format)       # 4,900.00
pd.describe_option()      #tutorial
pd.reset_option('all')
```

29. DataFrame:

```
pd.DataFrame({'student':np.arange(100,110,1),'test':np.random.randint(60,101,10)})
df = pd.DataFrame({'id':[101,102,103],'color':['red','blue','red']},index=list('abc'))
s = pd.Series(['round','square'],index=['c','b'],name='shape')
```

30. apply:

```
traindata['Sex_num'] = traindata.Sex.map({'female':0,'male':1})      # map works in Series
traindata['name_len']=traindata.Name.apply(len)
traindata.name.str.split(',').apply(lambda x: x[0])        # Series
drinks.loc[:,'beer_serving':'wine_servings'].apply(max,axis=1)  # DataFrame
drinks.loc[:,'beer_serving':'wine_servings'].applymap(float)      #all elements in the DataFrame
# three tips in the end of video
```

Numpy skills in Python For Data Analysis

https://docs.scipy.org/doc/numpy-dev/reference/

1. create:

```
arr = np.array([0,1,2,3], dtype = float)          np.arange(4) #[0->3]      np.arange(-5,5,0.01)
arr.shape  arr.ndim   arr.dtype  arr.astype(int)
np.zeros(10)   np.eye(5,dtype=int)        np.zeros((3,4))        np.ones((3,4)) #mention the parenthesis
np.array([[1,2,3],[4,5,6]])
```

2. select:

```
arr[:2,:3] # same to Matlab     arr[[0,2]] #row 0 and 2    arr[:,[0,2]]
data[data<0]=0
```

3. calculate:

```
arr.T #transpose      np.dot(arr.T,arr)      np.sqrt(arr)      np.exp(arr) #e^x
np.maximum(x,y)   abs, square, log, ceil, floor, isnan, isinf
arr.mean()       np.mean(arr)   arr.mean(axis=0/1) # 0 is for each row, 1 for each column
arr.cumsum()   arr.cumprod()
np.random.randint(low, high=None, size=None, dtype='l')# parameter: (b),[0,b];(a,b),[a,b];(a,b,lambda),[a,b]
np.random.randn()          np.random.randn(4,6)     # 4 by 6 normal distribution value
np.linspace(a,b,n)   #same to Matlab
np.sort(arr)
```

Machine learning with scikit-learn series course

1. ML: semi-automated extraction of knowledge from data
categories: supervised & unsupervised
www-bcf.usc.edu/~gareth/ISL/ # An introduction to statistical learning, PDF
work.caltech.edu/library/014.html # ML video library, VIDEO
kevin@dataschool.io #email dataschool.io # website @justmarkham #tritter

2. Ipython: Enter, Esc, M(markdown), H(keyboard shortcuts)
nbviewer.ipython.org #nbviewer
codecademy.com/en/tracks/python #codecademy
Dataquest.io/missions # Dataquest
developers.google.com/edu/python/ #Google's python class
pythonlearn.com #pyhtonlearn(slides and videos)

3. dataset: observation(sample, example, instance, record)
feature(predictor, attribute, independent variable, input, regressor)
response(target, outcome, label, dependent variable) type 'numpy.ndarray'
categories: classification & regression

4. four-step modeling pattern:

```
S1: from sklearn.neighbors import KNeighborsClassifier  #import classifier
S2: knn = KNeighborsClassifier(n_neighbors=1)  #instantiate the model
S3: knn.fit(x,y)  # fit the model
S4: knn.predict(x')  # predict the response  // kNN
import sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
logreg.fit(x,y)
logreg.predict(x')  // logistic regression
```

dataschool.io/15-hours-of-expert-machine-learning-videos/

5. comparing ML:

```
accuracy: from sklearn import metrics
print(metrics.accuracy_score(y,y_pred))  // calculate the prediction accuracy
from sklearn.cross_validation import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.4) // split the data into two parts
k_range = range(1,26)
scores = []
for i in range k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(x_train,y_train)
    y_pred = knn.predict(x_test)
    scores.append(metrics.accuracy-score(y_test,y_pred))
plt.plot(k_range, scores)
plt.xlabel('Value of k for KNN')
```

plt.ylabel('Testing Accuracy') // try k=1 through k=25, record accuracy

http://scott.fortmann-roe.com/ Accurately measuring model prediction error.pdf

Understanding the bias-variance tradeoff.pdf # downloaded in E

6. pandas, seaborn, scikit-learn: #Linear regression:

import seaborn as sns sns.pairplot(data, x_vars=['TV','Radio','Newspaper'],y_vars='Sales',size=6,kind='reg')

feature_cols = ['TV','Radio','Newspaper'] x = data[feature_cols] y = data['Sale'] # dataset

from sklearn.cross_validation import train_test_split x = df[['a','b','c']] y = df.y

x_train,x_test,y_train,y_test = train_test_split(x,y,random_state=1) # split

from sklearn.linear_model import LinearRegression

linreg = LinearRegression() linreg.fit(x_train,y_train) y_pred = linreg.predict(x_test) #predict

linreg.intercept_ linreg.coef_

from sklearn import metrics np.sqrt(metrics.mean_squared_error(y_test,y_pred))

Mean Absolute Error (MAE): $\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$, metrics.mean_absolute_error(y_test,y_pred)

Mean Squared Error (MSE): $\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$, metrics.mean_squared_error(y_test,y_pred)

Root Mean Squared Error (**RMSE**): $\sqrt{\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|}$ np.sqrt(metrics.mean_squared_error(y_test,y_pred))

7. cross_validation vs train_test_split:

from sklearn.cross_validation import KFold

kf = KFold(25,n_fold=5,shuffle=False) //simulate splitting a dataset of 25 observations into 5 folds

from sklearn.cross_validation import cross_val_score

knn = KNeighborsClassifier(n_neighbors=5)

scores=cross_val_score(knn, x,y,cv=10,scoring='accuracy') //10-fold cross-validation with k=5

8. best model parameters:

from sklearn.grid_search import GridSearchCV

k_range = range(1,31)

param_grid = dict(n-neighbors=k_range)

grid = GridSearchCV(knn,param_grid,cv=10,scoring='accuracy')

grid.grid_scores_

grid.best_score_ grid.best_params_ grid.best_estimator_

from sklearn.grid_search import RandomizedSearchCV

param_dist = dict(n_neighbors=k_range,weights=weight_options)

rand = RandomizedSearchCV(knn,param_dist,cv=10,scoring='accuracy',n_iter=10,random_state=5)

rand.fit(x,y)

rand.grid_scores_

9. Evaluating a classification model: