In [115]:

```python
# how to stack data using pandas concatenate
df1 = pd.DataFrame(np.random.randint(0,100,size=(100, 4)), columns=list('ABCD'))

```

In [118]:

```python
df2 = pd.DataFrame(np.random.randint(0,100,size=(150, 4)), columns=list('ABCD'))
df3 = pd.DataFrame(np.random.randint(0,100,size=(110, 4)), columns=list('ABCD'))

```

In [120]:

```python
frames = [df1, df2, df3]

df123 = pd.concat(frames)
print(df123.shape)
```

```
(360, 4)
```

In [62]:

```python

```

In [121]:

```python
#titanic data
#http://web.stanford.edu/class/archive/cs/cs109/cs109.1166/problem12.html
# put the data in the directory of workshop, which is in the working directory
```

In [56]:

```python
import numpy as np
import pandas as pd
import os
```

In [122]:

```python
titanic_train = pd.read_csv("workshop/titanic.csv")
char_cabin = titanic_train["Pclass"].astype(str)
# Convert cabin to str -- categorical data
```

In [ ]:

```python

```

In [125]:

```python
my_tab = pd.crosstab(titanic_train["Survived"],  # Make a crosstab
                              columns="count")      # Name the count column

my_tab
```

Out[125]:

col_0	count
Survived	
0	545
1	342

In [126]:

```
my_tab = pd.crosstab(titanic_train["Survived"],  # Make a crosstab
                              columns="count", margins = True)      # Name the count column

my_tab
```

Out[126]:

col_0	count	All
Survived		
0	545	545
1	342	342
All	887	887

In [70]:

```
my_tab = pd.crosstab(titanic_train["Sex"],  # Make a crosstab
                              columns="count")      # Name the count column

my_tab
```

Out[70]:

col_0	count
Sex	
female	314
male	573

In [ ]:

```

```

In [127]:

```
# Table of survival vs. sex
survived_sex = pd.crosstab(titanic_train["Survived"],
                           columns=titanic_train["Sex"])

survived_sex.index= ["died","survived"]

survived_sex
```

Out[127]:

Sex	female	male
died	81	464
survived	233	109

In [ ]:

```

```

In [128]:

```
# Table of survival vs passenger class
survived_class = pd.crosstab(titanic_train["Survived"],
                             columns=titanic_train["Pclass"])

survived_class.columns = ["class1","class2","class3"]
survived_class.index= ["died","survived"]

survived_class
```

Out[128]:

	class1	class2	class3
died	80	97	368
survived	136	87	119

In [129]:

```
# Table of survival vs passenger class
survived_class = pd.crosstab(titanic_train["Survived"],
                             columns=titanic_train["Pclass"], margins =
True)

survived_class.columns = ["class1","class2","class3", "rows_total"]
survived_class.index= ["died","survived", "cols_total"]

survived_class
```

Out[129]:

	class1	class2	class3	rows_total
died	80	97	368	545
survived	136	87	119	342
cols_total	216	184	487	887

In [131]:

```
# simple analysis for survial by class
print( survived_class.iloc[0, :] / survived_class.iloc[2, :] )
```

```
class1        0.370370
class2        0.527174
class3        0.755647
rows_total    0.614431
dtype: float64
```

In [85]:

```
# observed values --
observed = survived_class.iloc[0:2,0:3]   # Get table without totals
for later use
observed
```

Out[85]:

	class1	class2	class3
died	80	97	368
survived	136	87	119

In [ ]:

```python

```

In [107]:

```python
#expected
expected2 =  np.outer(survived_class["rows_total"][0:2],
                     survived_class.iloc[2,  0:3] )/ 487
expected2 = pd.DataFrame(expected2)

expected2.columns = ["class1","class2","class3"]
expected2.index = ["died","survived"]

expected2

```

Out[107]:

	class1	class2	class3
died	241.724846	205.913758	545.0
survived	151.687885	129.215606	342.0

In [108]:

```python
chi_squared_stat = (((observed-expected2)**2)/expected2).sum().sum()

print(chi_squared_stat)
```

```
384.11430821033935
```

In [111]:

```python
crit = stats.chi2.ppf(q = 0.95, # Find the critical value for 95% confidence*
                      df = 5)   # *

print("Critical value")
print(crit)

p_value = 1 - stats.chi2.cdf(x=chi_squared_stat,  # Find the p-value
                             df=8)
print("P value")
print(p_value)
```

```
Critical value
11.0704976935
P value
0.0
```

In [ ]:

```python

```

In [132]:

```
#independence test
import numpy as np
import pandas as pd
import scipy.stats as stats
np.random.seed(10)

# Sample data randomly at fixed probabilities
voter_race = np.random.choice(a=
["asian","black","hispanic","other","white"],
                              p = [0.05, 0.15 ,0.25, 0.05, 0.5],
                              size=1000)

# Sample data randomly at fixed probabilities
voter_party = np.random.choice(a=
["democrat","independent","republican"],
                              p = [0.4, 0.2, 0.4],
                              size=1000)

voters = pd.DataFrame({"race":voter_race,
                       "party":voter_party})

voter_tab = pd.crosstab(voters.race, voters.party, margins = True)

voter_tab.columns =
["democrat","independent","republican","row_totals"]

voter_tab.index =
["asian","black","hispanic","other","white","col_totals"]

observed = voter_tab.iloc[0:5,0:3]   # Get table without totals for
later use
voter_tab
```

Out[132]:

	democrat	independent	republican	row_totals
asian	21	7	32	60
black	65	25	64	154
hispanic	107	50	94	251
other	15	8	15	38
white	189	96	212	497
col_totals	397	186	417	1000

In [102]:

```
voter_tab.loc["col_totals"][0:3]
```

Out[102]:

```
democrat       397
independent    186
republican     417
Name: col_totals, dtype: int64
```

In [89]:

```
expected =  np.outer(voter_tab["row_totals"][0:5],
                     voter_tab.loc["col_totals"][0:3]) / 1000

expected = pd.DataFrame(expected)

expected.columns = ["democrat","independent","republican"]
expected.index = ["asian","black","hispanic","other","white"]

expected
```

Out[89]:

	democrat	independent	republican
asian	23.820	11.160	25.020
black	61.138	28.644	64.218
hispanic	99.647	46.686	104.667
other	15.086	7.068	15.846
white	197.309	92.442	207.249

In [52]:

```
chi_squared_stat = (((observed-expected)**2)/expected).sum().sum()

print(chi_squared_stat)
```

```
7.169321280162059
```

In [53]:

```
crit = stats.chi2.ppf(q = 0.95, # Find the critical value for 95% confidence*
                      df = 8)   # *

print("Critical value")
print(crit)

p_value = 1 - stats.chi2.cdf(x=chi_squared_stat,  # Find the p-value
                             df=8)
print("P value")
print(p_value)
```

```
Critical value
15.5073130559
P value
0.518479392949
```

In [ ]:

```
#Given the high p-value, the test result does not detect a significant relationship between the variables.
```

In [ ]:

```

```

In [ ]:

```

```

###iris dataset

https://gist.githubusercontent.com/curran/a08a1080b88344b0c8a7/raw/d546eaee765268bf2f487608c
(https://gist.githubusercontent.com/curran/a08a1080b88344b0c8a7/raw/d546eaee765268bf2f487608

In [133]:

```
import pandas as pd
filepath = 
'https://gist.githubusercontent.com/curran/a08a1080b88344b0c8a7/raw/d546eaee765268bf2f487608c537c05e22e4b221/iris.csv'
iris = pd.read_csv(filepath)
```

In [15]:

```
print(iris.head())
```

```
   sepal_length  sepal_width  petal_length  petal_width species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
```

In [16]:

```
iris.columns
```

Out[16]:

```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')
```

In [17]:

```
iris_groups = iris.groupby("species")
```

In [18]:

```
iris.describe()
```

Out[18]:

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

In [19]:
```python
iris_groups.describe()
```

Out[19]:

	petal_length								petal_width		...	sepal_length		se
	count	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**	**count**	**mean**	**...**	**75%**	**max**	**c**
species														
setosa	50.0	1.464	0.173511	1.0	1.4	1.50	1.575	1.9	50.0	0.244	...	5.2	5.8	
versicolor	50.0	4.260	0.469911	3.0	4.0	4.35	4.600	5.1	50.0	1.326	...	6.3	7.0	
virginica	50.0	5.552	0.551895	4.5	5.1	5.55	5.875	6.9	50.0	2.026	...	6.9	7.9	

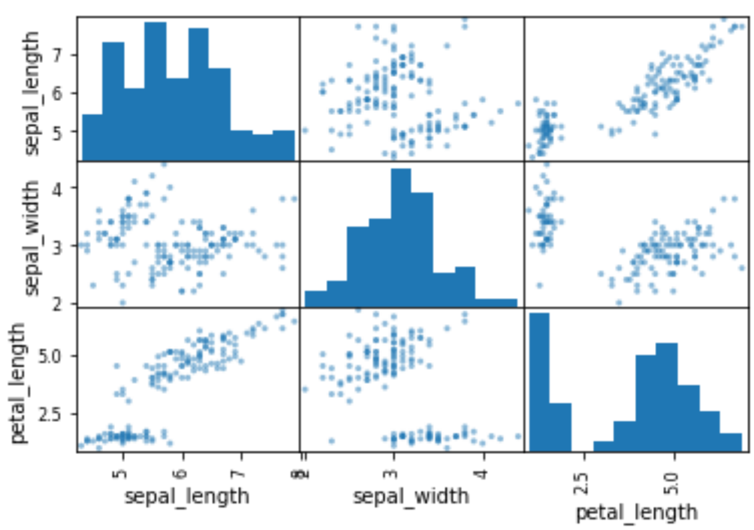
3 rows × 32 columns

In [20]:
```python
import os
os.getcwd()
```

Out[20]: '/Users/byunglee'

In [135]:
```python
%matplotlib inline
from pandas.plotting import scatter_matrix
scatter_matrix(irisData[['sepal_length', 'sepal_width',
'petal_length']])
```

Out[135]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x1a14517e10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a146232e8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a145a23c8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x1a146bb4a8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a146f54a8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a146f54e0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x1a14754f98>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a1478ef98>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a147d2048>]],
dtype=object)
```



In [ ]:
```python

```

In [138]:

```
irisData.drop(['species'], axis=1).corr(method='spearman')
```

Out[138]:

	sepal_length	sepal_width	petal_length	petal_width
sepal_length	1.000000	-0.159457	0.881386	0.834421
sepal_width	-0.159457	1.000000	-0.303421	-0.277511
petal_length	0.881386	-0.303421	1.000000	0.936003
petal_width	0.834421	-0.277511	0.936003	1.000000

In [22]:

```
#t-test
# 1-sample t-test: testing the value of a population mean
# scipy.stats.ttest_1samp() tests if the population mean of data is likely to be equal to a given value

from scipy import stats
stats.ttest_1samp(irisData['sepal_length'], 3)
```

Out[22]:

```
Ttest_1sampResult(statistic=42.054104134903668, pvalue=1.4781832542930679
e-84)
```

In [48]:

```
#2-sample t-test: testing for difference across populations
setosaSL = irisData[irisData['species'] == 'setosa']['sepal_length']

versicolorSL = irisData[irisData['species'] == 'versicolor']
['sepal_length']

#print(setosaSL)
#print(versicolorSL)
stats.ttest_ind(setosaSL, versicolorSL)
```

Out[48]:

```
Ttest_indResult(statistic=-10.520986267549111, pvalue=8.985235037487078 9e
-18)
```

In [ ]:

```

```

In [49]:

```
#  Regression: including multiple factors
import pandas as pd
filepath =
'https://gist.githubusercontent.com/curran/a08a1080b88344b0c8a7/raw/d5
46eaee765268bf2f487608c537c05e22e4b221/iris.csv'
irisData = pd.read_csv(filepath)
print(irisData.columns)
```

```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')
```

In [12]:

```
#simple regression
from statsmodels.formula.api import ols
model = ols("petal_width ~ sepal_width", irisData).fit()
print(model.summary())

```

```
                            OLS Regression Results
==============================================================================
=====
Dep. Variable:            petal_width   R-squared:
0.127
Model:                            OLS   Adj. R-squared:
0.121
Method:                 Least Squares   F-statistic:
21.55
Date:                Wed, 20 Jun 2018   Prob (F-statistic):           7.5
2e-06
Time:                        16:42:53   Log-Likelihood:                -1
61.60
No. Observations:                 150   AIC:
327.2
Df Residuals:                     148   BIC:
333.2
Df Model:                           1

Covariance Type:            nonrobust

==============================================================================
======
                  coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
------
Intercept       3.1152      0.417      7.472      0.000       2.291
3.939
sepal_width    -0.6275      0.135     -4.643      0.000      -0.895
-0.360
==============================================================================
=====
Omnibus:                       14.660   Durbin-Watson:
0.523
Prob(Omnibus):                  0.001   Jarque-Bera (JB):
6.402
Skew:                           0.266   Prob(JB):
0.0407
Kurtosis:                       2.140   Cond. No.
24.3
==============================================================================
=====

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is co
rrectly specified.
```

In [13]:

```
#multiple regression
from statsmodels.formula.api import ols
model = ols("petal_width ~ sepal_width + petal_length", irisData).fit()
print(model.summary())
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:            petal_width   R-squared:                       0.930
Model:                            OLS   Adj. R-squared:                  0.929
Method:                 Least Squares   F-statistic:                     972.6
Date:                Wed, 20 Jun 2018   Prob (F-statistic):           1.72e-85
Time:                        16:43:43   Log-Likelihood:                 27.367
No. Observations:                 150   AIC:                            -48.73
Df Residuals:                     147   BIC:                            -39.70
Df Model:                           2                                         
Covariance Type:            nonrobust                                         
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
Intercept       -0.7221      0.151     -4.776      0.000      -1.021      -0.423
sepal_width      0.1033      0.042      2.436      0.016       0.019       0.187
petal_length     0.4271      0.010     40.978      0.000       0.406       0.448
==============================================================================
Omnibus:                        3.971   Durbin-Watson:                   1.538
Prob(Omnibus):                  0.137   Jarque-Bera (JB):                3.764
Skew:                           0.244   Prob(JB):                        0.152
Kurtosis:                       3.604   Cond. No.                         48.2
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [ ]:

```

```

In [8]:

```
# Anova
irisData.columns
```

Out[8]:

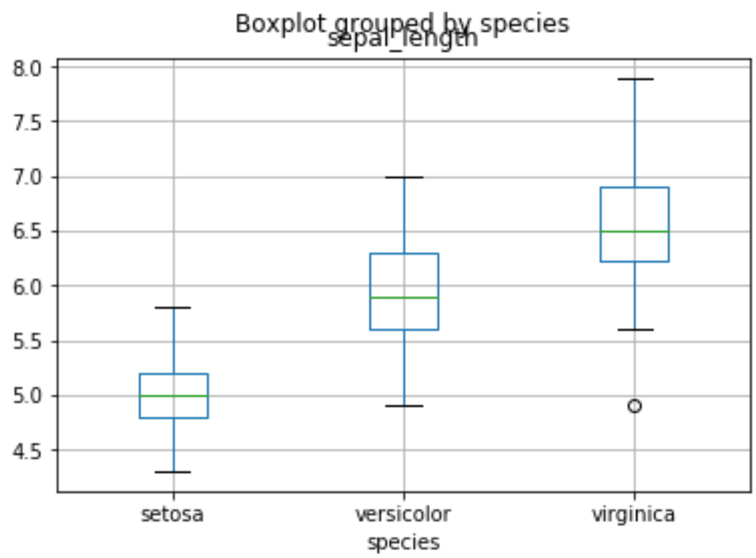
```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')
```

In [11]:

```
%matplotlib inline
irisData.boxplot("sepal_length", by = "species")
## what it means top, middle, bottom
```

Out[11]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1c8417b8>
```



In [12]:

```
#https://www.r-bloggers.com/anova-%E2%80%93-type-iiiiii-ss-explained/
#type 1, 2, 3 sums of squares


# compare  R and Python
# https://dpaniukov.github.io/2016/10/25/You-and-Your-R-Doing-
Statistics-in-Python.html
```

In [15]:

```
import statsmodels.api as sm
from statsmodels.formula.api import ols

mod = ols('sepal_length ~ species',
                data=irisData).fit()

aov_table = sm.stats.anova_lm(mod, typ=2)
print (aov_table)

```

```
             sum_sq     df           F        PR(>F)
species    63.212133    2.0  119.264502  1.669669e-31
Residual   38.956200  147.0         NaN           NaN
```

In [18]:

```
esq_sm = aov_table['sum_sq'][0]/(aov_table['sum_sq'][0]+aov_table['sum_sq'][1])
print(esq_sm)
```

```
0.618705730738
```

In [ ]:

```

```

In [19]:

```
# four different ways of ANOVA
#https://www.marsja.se/four-ways-to-conduct-one-way-anovas-using-python/
```

In [ ]:

```

```

In [ ]:

```
#============
```

In [3]:

```
#from sklearn.datasets import load_iris
from sklearn.decomposition import FactorAnalysis
#iris = load_iris()
#X, y = iris.data, iris.target
X = irisData.iloc[:, 0:4]
factor = FactorAnalysis(n_components=4, random_state=101).fit(X)

```

In [7]:

```
import pandas as pd
print(pd.DataFrame(factor.components_,columns=X.columns))
```

```
   sepal_length  sepal_width  petal_length  petal_width
0      0.707227    -0.153147      1.653151     0.701569
1      0.114676     0.159763     -0.045604    -0.014052
2     -0.000000     0.000000      0.000000     0.000000
3     -0.000000     0.000000      0.000000    -0.000000
```

In [11]:

```
from sklearn.decomposition import PCA
import pandas as pd
pca = PCA().fit(X)
print( "Explained variance by component: %s"  % pca.explained_variance_ratio_)
print("")
print (pd.DataFrame(pca.components_,columns=X.columns))
```

```
Explained variance by component: [ 0.92461621  0.05301557  0.01718514  0.00518309]

   sepal_length  sepal_width  petal_length  petal_width
0      0.361590    -0.082269      0.856572     0.358844
1      0.656540     0.729712     -0.175767    -0.074706
2     -0.580997     0.596418      0.072524     0.549061
3      0.317255    -0.324094     -0.479719     0.751121
```

#more advanced one including varimax rotation https://pypi.org/project/factor-analyzer/ (https://pypi.org/project/factor-analyzer/)

researchgate.net

Varimax Rotation and Thereafter: Tutorial on PCA Using Linear Algebra, Visualization, and Python Programming for R and Q analysis

```
In [ ]:
```