Open and prepare the data set

```python
data = pd.read_csv('countries of the world.csv')
data
```

	Country	Region	Population	Area (sq. mi.)	Pop. Density (per sq. mi.)	Coastline (coast/area ratio)	Net migration	Infant mortality (per 1000 births)	GDP ($ per capita)	Literacy (%)	Phones (per 1000)	Arable (%)	Crops (%)	Other (%)	Climate	Birthra
0	Afghanistan	ASIA (EX. NEAR EAST)	31056997	647500	48.0	0.00	23.06	163.07	700	36.0	3.2	12.13	0.22	87.65	1.0	46.
1	Albania	EASTERN EUROPE	3581655	28748	124.6	1.26	-4.93	21.52	4500	86.5	71.2	21.09	4.42	74.49	3.0	15.
2	Algeria	NORTHERN AFRICA	32930091	2381740	13.8	0.04	-0.39	31.00	6000	70.0	78.1	3.22	0.25	96.53	1.0	17.
3	American Samoa	OCEANIA	57794	199	290.4	58.29	-20.71	9.27	8000	97.0	259.5	10.00	15.00	75.00	2.0	22.

Test the correlation between GDP and Phones and draw the variables for linear regression

```python
print('GDP - Phones')
print('\nComparing linear regression values:')
print('-'*47)
print('{:15} {:>15} {:>15}'.format('Statistic', 'From Scipy', 'From Scratch'))
print('-'*47)
for name, sc_val, sp_val in [("alpha", intercept_phone, alpha_phone),
                             ("beta", slope_phone, beta_phone),
                             ("r-squared", r_squared_value_phone, r_v_phone**2)]:
    print('{:15} {:15.2f} {:15.2f}'.format(name, sc_val, sp_val))
print('-'*47)

from scipy.stats import pearsonr
print('\nCorrelation from scipy: {:.3f}'.format(pearsonr(data[GDP], data[Phones])[0]))
print('Correlation from scratch: {:.3f}\n'.format(correlation(data[GDP], data[Phones])))

```

```
GDP - Phones

Comparing linear regression values:
-----------------------------------------------
Statistic            From Scipy    From Scratch
-----------------------------------------------
alpha                     47.91           47.92
beta                       0.02            0.02
r-squared                  0.72            0.72
-----------------------------------------------

Correlation from scipy: 0.849
Correlation from scratch: 0.849
```

Illustrate the linear regression graph using variables found above

```python
plt.scatter(data[GDP], data[Phones])
_ = plt.xlim(0,40000)
_ = plt.ylim(0,1000)

_ = plt.xlabel('GDP')
_ = plt.ylabel('# of Phones (per 1000 people)')
_ = plt.plot([0,40000], [predict(alpha_phone, beta_phone, 0), predict(alpha_phone, beta_phone, 40000)], 'r',
             [0,40000], [predict(alpha_phone, beta_phone, 0)+2*error_phone,
                         predict(alpha_phone, beta_phone, 40000)+2*error_phone], 'r--',
             [0,40000], [predict(alpha_phone, beta_phone, 0)-2*error_phone,
                         predict(alpha_phone, beta_phone, 40000)-2*error_phone], 'r--')
```

Compare correlation values using the heat-map

```python
sns.heatmap(data=data[[GDP,Phones,Migration,Infant,Birthrate]].corr(), annot=True, cmap="YlGnBu")
label = ["GDP","Phones","Migration","Infant","Birthrate"]
index = np.arange(len(label))
plt.xticks(index+0.5, label)
plt.yticks(index+0.5, label)
plt.show()
```

Split the data into Training data and Test data

```python
X = data[[GDP,Phones,Birthrate,Deathrate,Infant,Migration]]
Y = data[[Region]]
print(X.shape)
print(Y.shape)
```

```
(203, 6)
(203, 1)
```

```python
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=1)
print("Train X data:", X_train.shape)
print("Train Y data:", Y_train.shape)
print("Test X data:", X_test.shape)
print("Test Y data:", Y_test.shape)

```

```
('Train X data:', (142, 6))
('Train Y data:', (142, 1))
('Test X data:', (61, 6))
('Test Y data:', (61, 1))
```

Fit the data to decision tree model and check the accuracy

```python
from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier(random_state=1)
model.fit(X_train, Y_train)

print("Training set Accuracy: {:.3f}".format(model.score(X_train, Y_train)))
print("Test set Accuracy: {:.3f}".format(model.score(X_test, Y_test)))

```

```
Training set Accuracy: 1.000
Test set Accuracy: 0.607
```

Fit the data to logistic regression model and check the accuracy

```python
logreg = LogisticRegression()
_ = logreg.fit(X_train, Y_train)

print("Test set Accuracy: {:.3f}".format(logreg.score(X_test, Y_test)))
```

```
Test set Accuracy: 0.607
```

Declare ‘key’ for connecting with the targets(regions)

```python
region_groups = data.groupby('Region')
key=', '.join(['{}={}'.format(i,name) for i,name in enumerate(region_group)])
key
```

```
'0=AFRICA, 1=ASIA, 2=EAST ASIA, 3=EASTERN EUROPE, 4=LATIN AMER. & CARIB, 5=NORTHERN AMERICA, 6=OCEANIA, 7=WESTERN EUR
OPE'
```

Estimate intercept/coefficients/splitter and illustrate classification report and confusion matrix

```python
print('Intercept:\n{}\n'.format(logreg.intercept_))
print('Coefficients:\n{}\n'.format(logreg.coef_))

# We can predict the type of new organisms given measurements
print('\nPredicted type of first five organisms from test split: \n{}'.format(logreg.predict(X_test)[:8]))
print('\nActual type of first five organisms from test split:\n{}\n'.format(Y_test[:8]))

print(classification_report(Y_test, logreg.predict(X_test)))
print('Confusion matrix ({}):\n'.format(key))
_ = plt.matshow(confusion_matrix(Y_test, logreg.predict(X_test)), cmap=plt.cm.PuRd, interpolation='nearest')
_ = plt.colorbar()
_ = plt.ylabel('true label')
_ = plt.xlabel('predicted label')
```

```
Intercept:
[-0.02621052  0.03265841  0.1010144   0.13584803  0.04020111 -0.05868455
  0.01594544 -0.11620712]

Coefficients:
[[-4.07840105e-04 -4.97840778e-03  7.75371758e-03  2.64356527e-01
  -2.66124205e-02  7.56774974e-03]
 [ 2.51369699e-05 -1.60289657e-03  1.25401738e-02 -4.53743798e-01
   4.43585181e-02  9.20866284e-02]
 [ 4.20370288e-05  4.52984310e-03 -7.75586040e-02 -4.45196698e-01
  -3.44661932e-01  2.28582846e-02]
 [-6.66338607e-05  5.62148281e-04 -2.37455844e-01  1.29190659e-01
   3.37149195e-02 -9.99127859e-02]
 [-1.22118551e-04  3.65034917e-03  5.25809278e-02 -1.13588531e-01
  -3.78716764e-02  4.59362839e-03]
 [-3.53968248e-05  2.19909912e-02 -6.49615786e-01 -1.00838028e+00
  -3.64758019e-02 -4.94222049e-01]
 [ 3.74936819e-05 -3.47181166e-03  1.15178405e-01 -2.22156870e-01
  -1.73015559e-01 -1.79861142e-01]
 [ 1.30056128e-04  2.69258705e-03 -2.77112282e-01  4.81408617e-01
  -1.01865524e+00 -2.33406671e-01]]
```

```
Predicted type of first five organisms from test split:
['LATIN AMER. & CARIB' 'AFRICA' 'AFRICA' 'AFRICA' 'AFRICA' 'AFRICA' 'ASIA'
 'LATIN AMER. & CARIB']

Actual type of first five organisms from test split:
                  Region
174  LATIN AMER. & CARIB
175               AFRICA
18        EASTERN EUROPE
59                AFRICA
143              OCEANIA
31                  ASIA
28                  ASIA
40   LATIN AMER. & CARIB
```

Find the best parameter for logistic regression and print the classification report and the confusion matrix with new parameters

```python
from sklearn.model_selection import GridSearchCV

# Perform grid search
param_grid = [
    {'C': [1, 10, 100, 1000, 1e4, 1e5, 1e6, 1e7], 'penalty': ['l1', 'l2']}
]
logreg = GridSearchCV(LogisticRegression(), param_grid)
logreg.fit(X_train, Y_train)

# Print grid search results
print('Grid search mean and stdev:\n')
#for mean, std, params in zip(means, stds, tree.cv_results_['params']):
for mean_score, scores, params in zip (logreg.cv_results_['mean_score_time'],logreg.cv_results_['mean_test_score'],
    print("{:0.3f} (+/-{:0.03f}) for {}".format(
            mean_score, scores.std() * 2, params))

# Print best params
print('\nBest parameters:', logreg.best_params_)

print('\nClassification report ({}):\n'.format(key))
print(classification_report(Y_test, logreg.predict(X_test)))
print('Confusion matrix ({}):\n'.format(key))
_ = plt.matshow(confusion_matrix(Y_test, logreg.predict(X_test)), cmap=plt.cm.PuRd, interpolation='nearest')
_ = plt.colorbar()
_ = plt.ylabel('true label')
_ = plt.xlabel('predicted label')
```

```
0.002 (+/-0.000) for {'penalty': 'l1', 'C': 1}
0.002 (+/-0.000) for {'penalty': 'l2', 'C': 1}
0.002 (+/-0.000) for {'penalty': 'l1', 'C': 10}
0.002 (+/-0.000) for {'penalty': 'l2', 'C': 10}
0.001 (+/-0.000) for {'penalty': 'l1', 'C': 100}
0.002 (+/-0.000) for {'penalty': 'l2', 'C': 100}
0.002 (+/-0.000) for {'penalty': 'l1', 'C': 1000}
0.003 (+/-0.000) for {'penalty': 'l2', 'C': 1000}
0.002 (+/-0.000) for {'penalty': 'l1', 'C': 10000.0}
0.003 (+/-0.000) for {'penalty': 'l2', 'C': 10000.0}
0.002 (+/-0.000) for {'penalty': 'l1', 'C': 100000.0}
0.001 (+/-0.000) for {'penalty': 'l2', 'C': 100000.0}
0.001 (+/-0.000) for {'penalty': 'l1', 'C': 1000000.0}
0.001 (+/-0.000) for {'penalty': 'l2', 'C': 1000000.0}
0.002 (+/-0.000) for {'penalty': 'l1', 'C': 10000000.0}
0.002 (+/-0.000) for {'penalty': 'l2', 'C': 10000000.0}
('\nBest parameters:', {'penalty': 'l1', 'C': 100000.0})
```

```
                     precision    recall  f1-score   support

             AFRICA       1.00      0.79      0.88        19
               ASIA       0.56      0.62      0.59         8
          EAST ASIA       0.33      0.50      0.40         2
     EASTERN EUROPE       0.73      0.80      0.76        10
LATIN AMER. & CARIB       0.55      1.00      0.71         6
   NORTHERN AMERICA       0.00      0.00      0.00         2
            OCEANIA       0.00      0.00      0.00         4
     WESTERN EUROPE       0.58      0.70      0.64        10

          micro avg       0.69      0.69      0.69        61
          macro avg       0.47      0.55      0.50        61
       weighted avg       0.66      0.69      0.66        61
```

Test how the accuracy changes according to different parameters

```python
train_result = []
test_result = []
model_criterion = []
model_max_depth = []
parameter_min_leaf = []

insert_criterion = ['gini', 'entropy']
max_depth = 5
list_min_leaf = [i for i in range(1,10)]
for i in insert_criterion:
    for n in list_min_leaf:
        tree = DecisionTreeClassifier(criterion=i, max_depth=max_depth, min_samples_leaf=n, random_state=1)
        tree.fit(X_train, Y_train)
        train_result.append(tree.score(X_train,Y_train))
        test_result.append(tree.score(X_test,Y_test))
        model_criterion.append(i)
        model_max_depth.append(max_depth)
        parameter_min_leaf.append(n)

result = pd.DataFrame()
result["Criterion"] = model_criterion
result["Depth"] = max_depth
result["MinLeafSize"] = parameter_min_leaf
result["TrainAccuracy"] = train_result
result["TestAccuracy"] = test_result
result

```

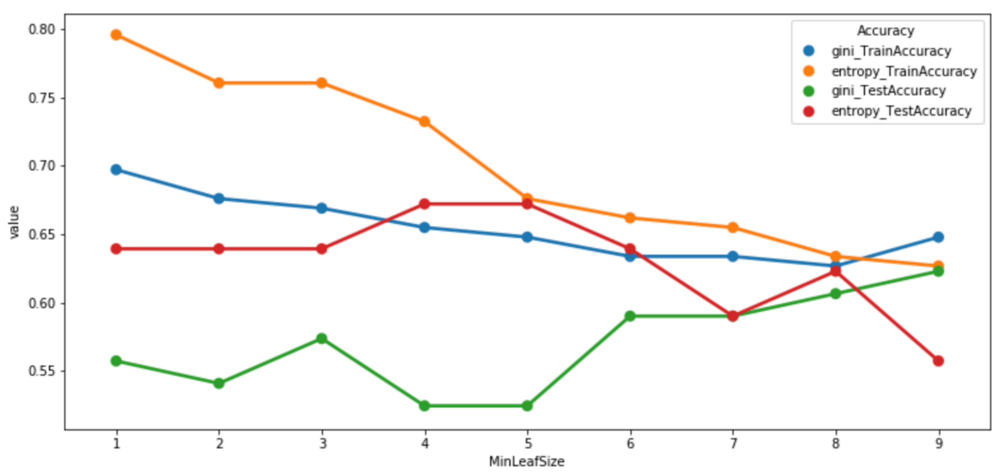
	Criterion	Depth	MinLeafSize	TrainAccuracy	TestAccuracy
0	gini	5	1	0.697183	0.557377
1	gini	5	2	0.676056	0.540984
2	gini	5	3	0.669014	0.573770
3	gini	5	4	0.654930	0.524590
4	gini	5	5	0.647887	0.524590
5	gini	5	6	0.633803	0.590164
6	gini	5	7	0.633803	0.590164
7	gini	5	8	0.626761	0.606557
8	gini	5	9	0.647887	0.622951
9	entropy	5	1	0.795775	0.639344
10	entropy	5	2	0.760563	0.639344
11	entropy	5	3	0.760563	0.639344
12	entropy	5	4	0.732394	0.672131
13	entropy	5	5	0.676056	0.672131
14	entropy	5	6	0.661972	0.639344
15	entropy	5	7	0.654930	0.590164
16	entropy	5	8	0.633803	0.622951
17	entropy	5	9	0.626761	0.557377

Visualise the test result and determine the best parameter

```python
plt.figure(figsize=(13,6))
result_melt = pd.melt(result, id_vars=['Criterion', 'Depth', 'MinLeafSize'])
result_melt['Accuracy'] = result_melt['Criterion']+'_'+result_melt['variable']
sns.pointplot(data=result_melt, x='MinLeafSize', y='value', hue='Accuracy')

```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1a462d90>
```



Fit the new model with the best parameter that chosen according to the test above and see how the accuracy has changed

```
model_new = DecisionTreeClassifier(criterion='entropy', max_depth=5, min_samples_leaf=5, random_state=1)
model_new.fit(X_train,Y_train)
print("Training set Accuracy: {:.3f}".format(model_new.score(X_train, Y_train)))
print("Test set Accuracy: {:.3f}\n".format(model_new.score(X_test, Y_test)))

from sklearn.metrics import classification_report
print(classification_report(Y_test, model_new.predict(X_test)))
```

```
Training set Accuracy: 0.676
Test set Accuracy: 0.672

                     precision    recall  f1-score   support

             AFRICA       0.84      0.84      0.84        19
               ASIA       0.40      0.75      0.52         8
          EAST ASIA       0.50      0.50      0.50         2
     EASTERN EUROPE       1.00      0.60      0.75        10
LATIN AMER. & CARIB       0.57      0.67      0.62         6
   NORTHERN AMERICA       0.00      0.00      0.00         2
            OCEANIA       0.00      0.00      0.00         4
     WESTERN EUROPE       0.67      0.80      0.73        10

          micro avg       0.67      0.67      0.67        61
          macro avg       0.50      0.52      0.49        61
       weighted avg       0.66      0.67      0.65        61
```

McNemar test to find out if there a big different in the error rate between logistic regression model and decision tree model and determine if my null hypothesis should be rejected or not

```
from scipy.stats import chi2
def mcnemar(x, y):
    n1 = np.sum(x < y)
    n2 = np.sum(x > y)
    stat = (np.abs(n1-n2)-1)**2 / (n1+n2)
    df = 1
    pval = chi2.sf(stat,1)
    return stat, pval

# Calculate whether each test prediction is correct
l_yn = np.array([int(p==t) for p,t in zip(logreg.predict(X_test), Y_test.values)])
t_yn = np.array([int(p==t) for p,t in zip(model_new.predict(X_test), Y_test.values)])

print(l_yn)
print(t_yn)

print('Can we reject H0?', 'Yes' if mcnemar(l_yn, t_yn)[1]<0.05 else 'No')
```

Draw the decision tree with final model

```
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier, export_graphviz

export_graphviz(model_new, out_file="tree999.dot", class_names=data[Region],
                feature_names=["GDP","Phones","Birthrate","Deathrate","Infant","Migration"],
                impurity=False, filled=True)
import os
os.environ["PATH"] += os.pathsep + '/anaconda2/pkgs/graphviz-2.40.1-hefbbd9a_2/lib/graphviz'
import graphviz
with open("tree999.dot") as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)

```