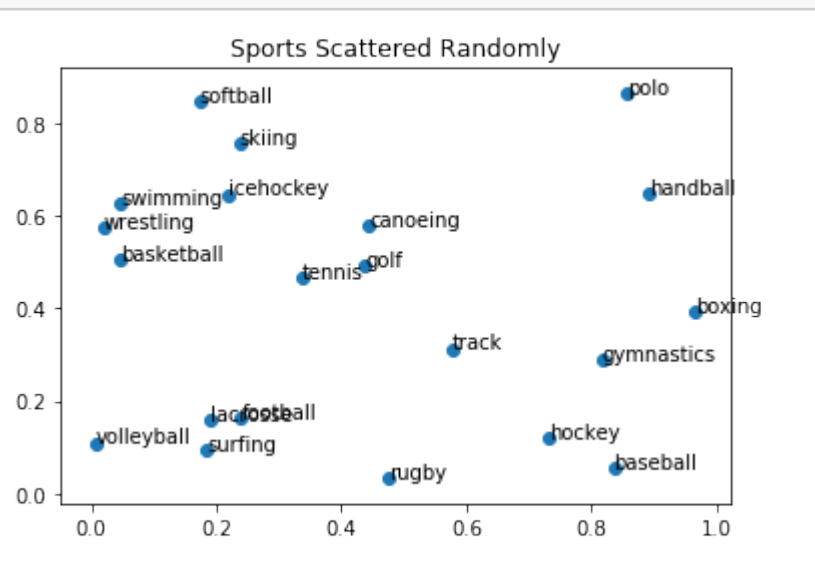
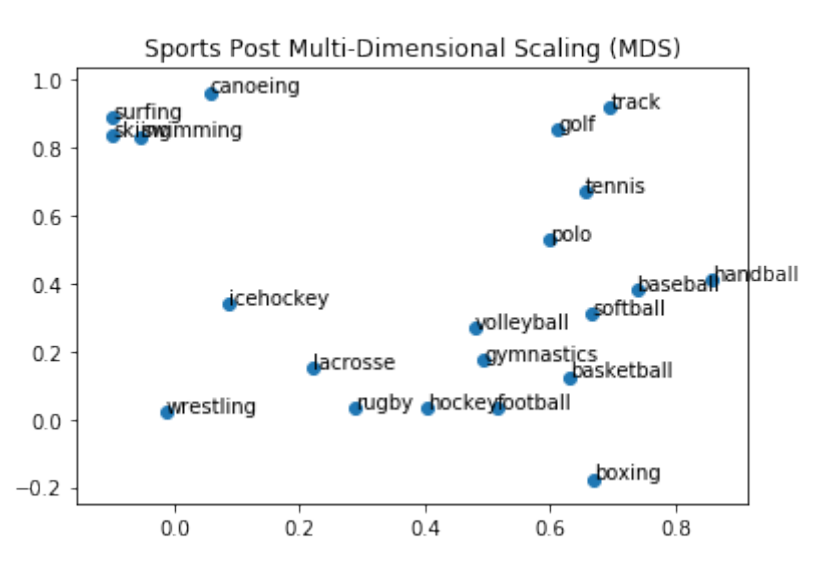
In [10]:
```python
#Question 2

import numpy as np
import matplotlib.pyplot as plt

#to compute partial derivatives
delta = 0.005
scaling = 0.01

#other needed variables
gradient = []
stress = 0

#read in similarities
psychsimilarities = np.loadtxt(open("Assignment5-similarities.csv", "rb"), delimiter=",", skiprows=1)

#assign random values to the starting points on MDS grid
mdsdistances = np.random.uniform(0, 1, size=(21,2))

#matrix ==> array of MDS locations (x,y coordinates)
#[(football)[0.03,93.2], (soccer)[4,6.78], (basketball)[3.3,4.1]]

#similarities ==> array from our data --> psychological distances
#* = startingsport
#[(football)[football-football, *football-soccer, football-bball, football-golf],
#(soccer)[football-soccer, soccer-soccer, *soccer-bball, soccer-golf]],
#(bball)[football-bball, bball-soccer, bball-bball, *bball-golf]


def stresscalc(mdsdistances, psychsimilarities):
    stress = 0
    mdsdistance = 0
    psychdistance = 0
    row = 0
    col = 1
    for row in range(0,20): #for each sport
        #compare sport to each remaining sport
        c = col
        for c in range(c,20):
            psychdistance = 1-psychsimilarities[row][c]
            mdsdistance = np.sqrt(np.square(mdsdistances[row][0] - mdsdistances[c][0]) + np.square(mdsdistances[
row][1] - mdsdistances[c][1]))
            stress += np.square(psychdistance-mdsdistance)
        col += 1
    return stress

#QUESTION 3
def findgradient(mdsdistances, psychsimilarities, delta):
    gradient= np.copy(mdsdistances)

    #Find the partial derivative by changing x and keeping y & changing y and keeping x constant.
    for i in np.arange(len(mdsdistances) -1): #for all coordinates in our grid

        #FIRST SOLVE FOR X
        coordinatesmodplus = np.copy(mdsdistances)
        coordinatesmodminus = np.copy(mdsdistances)

        coordinatesmodplus[i][0]+= delta #changing x value, and keeping y the same
        fpdx = stresscalc(coordinatesmodplus, psychsimilarities) #f(changed concept x, y remains)

        coordinatesmodminus[i][0]-=delta #changing x value, keeping y the same
        fmdx=stresscalc(coordinatesmodminus, psychsimilarities) #f(changed concept x, y remains)

        partialderivativex=(fpdx-fmdx)/(2*delta)

        gradient[i][0]=partialderivativex #update that part of the gradient with the partial derivative found fo
r element i of values

        #THEN SOLVE FOR Y
        coordinatesmodplus = np.copy(mdsdistances)
        coordinatesmodminus = np.copy(mdsdistances)

        coordinatesmodplus[i][1]+= delta #changing y value, and keeping x the same
        fpdy = stresscalc(coordinatesmodplus, psychsimilarities) #f(changed concept y, x remains)

        coordinatesmodminus[i][1]-=delta #changing y value, keeping x the same
        fmdy=stresscalc(coordinatesmodminus, psychsimilarities) #f(changed concept y, x remains)

        partialderivativey=(fpdy-fmdy)/(2*delta)

        gradient[i][1]=partialderivativey #update that part of the gradient with the partial derivative found fo
r element i of values

    return gradient

# def gradientcheck(gradient):
#     passed = []
#     for i in range(len(gradient)):
#         if gradient[i][0] <= 0.1:
#             passed.append(1)
#         if gradient[i][1] <= 0.1:
#             passed.append(1)
#     return sum(passed)

#QUESTION 4
names = ['football', 'baseball', 'basketball', 'tennis',
         'softball', 'canoeing', 'handball', 'rugby',
         'hockey', 'icehockey', 'swimming', 'track',
         'boxing', 'volleyball', 'lacrosse', 'skiing',
         'golf', 'polo', 'surfing', 'wrestling', 'gymnastics']

xpoints = []
ypoints = []
stressvals = []
iterations= 0

#PLOT RANDOM POINTS FIRST
for i in range(len(mdsdistances)):
    xpoints.append(mdsdistances[i][0])
    ypoints.append(mdsdistances[i][1])

fig, ax = plt.subplots()
ax.scatter(xpoints,ypoints)
ax.set_title("Sports Scattered Randomly")
for i, txt in enumerate(names):
    ax.annotate(txt, (xpoints[i], ypoints[i]))


gradient = findgradient(mdsdistances, psychsimilarities, delta)
stress = stresscalc(mdsdistances, psychsimilarities)
# checkval = gradientcheck(gradient)
while stress >= 10:
    for i in range(len(gradient)):
        mdsdistances[i][0] += (-scaling * gradient[i][0])
        mdsdistances[i][1] += (-scaling * gradient[i][1])
        xpoints[i] = (mdsdistances[i][0])
        ypoints[i] = (mdsdistances[i][1])
    gradient = findgradient(mdsdistances, psychsimilarities, delta)
    stress = stresscalc(mdsdistances, psychsimilarities)
    stressvals.append(stress)
    iterations += 1
#     checkval = gradientcheck(gradient)
#     print("checkval", checkval)

fig, ax = plt.subplots()
ax.scatter(xpoints,ypoints)
ax.set_title("Sports Post Multi-Dimensional Scaling (MDS)")
for i, txt in enumerate(names):
    ax.annotate(txt, (xpoints[i], ypoints[i]), fontsize = 10)
```





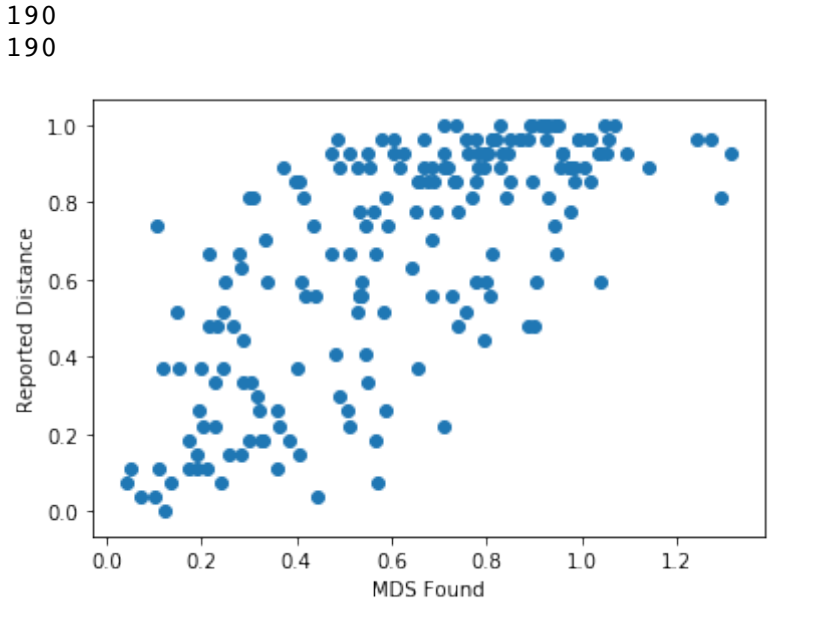
4.[20pts, HELP] Write the code that follows a gradient in order to find positions that minimize the stress – be sure to take small steps in the direction of the gradient (e.g. 0.01*gradient). Plot the sport names at the resulting coordinates. Do the results agree with your intuitions about how this domain might be organized? Why or why not?

The results are in line with what my intuition is about how sports might be organized. For example, all of the water-related (or snow-related) sports (i.e. canoeing, surfing, swimming, skiing) are clustered together. In addition, the "fighting" sports (i.e. wrestling, boxing) are clustered together, and the sports that require a ball are also clustered in the same region of the graph.

1. [5pts, SOLO] Make a scatter plot of the the pairwise distances MDS found vs. people's reported distances. Briefly describe what good and bad plots would look like and whether yours is good or bad.

In [15]:
```python
#QUESTION 5
psydist = []
mdsdist = []

col = 1
for row in range(0,20): #for each sport
    #compare sport to each remaining sport
    c = col
    for c in range(c,20):
        psydist.append(1-psychsimilarities[row][c])
    col += 1

col = 1
for row in range(0,20): #for each sport
    #compare sport to each remaining sport
    c = col
    for c in range(c,20):
        mdsdist.append(np.sqrt(np.square(mdsdistances[row][0] - mdsdistances[c][0]) + np.square(mdsdistances[row
][1] - mdsdistances[c][1])))
    col += 1

print(len(psydist))
print(len(mdsdist))
plt.scatter(mdsdist, psydist)
plt.xlabel("MDS Found")
plt.ylabel("Reported Distance")
plt.show()
```

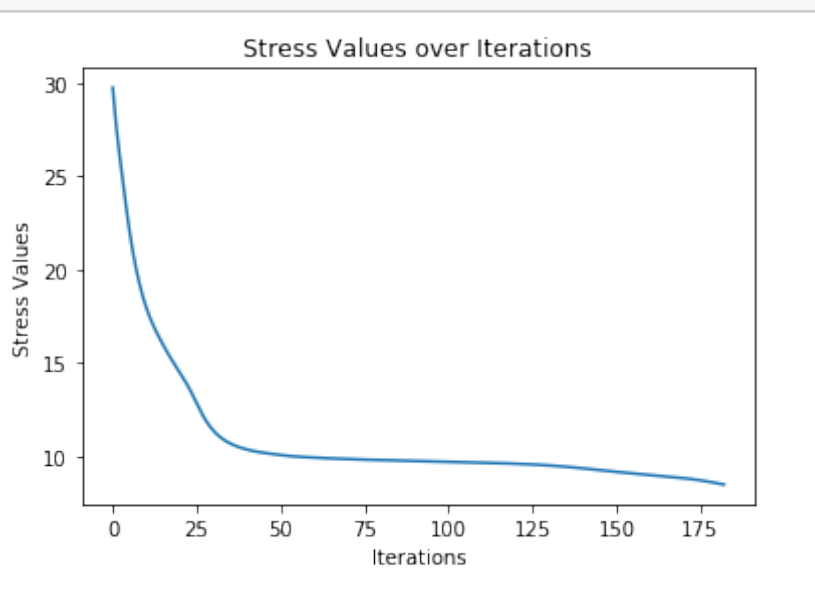
```
190
190
```



A good plot has a trend, meaning the x values and y values increase or decrease with some sort of consistent relationship. In this case, a good plot would have a positive correlation, meaning as people's reported distances increased, my MDS plots found should have also increased. This is somewhat represented in my graph but there are definitely some outliers. For example, one point was reported at a near 1 distance by people psychologically, but my MDS found it to be at ~0.4 distance value.

1. [5pts, SOLO] Plot the stress over iterations of your MDS. How should you use this plot in order to figure out how many iterations are needed?

In [54]:
```python
#QUESTION 6

plt.plot((np.arange(iterations)), stressvals)
plt.title("Stress Values over Iterations")
plt.xlabel("Iterations")
plt.ylabel("Stress Values")
plt.show()
```



This plot can be used to figure out how many iterations are needed by looking at where the stress values begin to plateau, which is around a stress value of ~9. By the looks of my graph, my stress values continue to decrease even after 175 iterations, meaning I should probably run my program for longer. However, for the purpose of this assignment and on piazza/from advice from GSI's, I was told a stress value of around 10 is sufficient.

1. [5pts, SOLO] Run the MDS code you wrote 10 times and show small plots, starting from random initial positions. Are they all the same or not? Why?

After running MDS 10 times it can be seen that the graphs are not all the same. This is due to a couple factors, including that the points are all starting from random initial positions. This means that their locations on the graph will vary depending on where they started that round of MDS. In addition, the method of gradient descent we are using does not account for local vs global minima of the stress function. If the gradient descent method finds a local minimum, this will also cause the resulting graph to look differently.

In [17]:
```python
#Question 7

import numpy as np
import matplotlib.pyplot as plt

#to compute partial derivatives
delta = 0.005
scaling = 0.01

#other needed variables
gradient = []
stress = 0

#read in similarities
psychsimilarities = np.loadtxt(open("Assignment5-similarities.csv", "rb"), delimiter=",", skiprows=1)

#assign random values to the starting points on MDS grid
mdsdistances = np.random.uniform(0, 1, size=(21,2))

def stresscalc(mdsdistances, psychsimilarities):
    stress = 0
    mdsdistance = 0
    psychdistance = 0
    row = 0
    col = 1
    for row in range(0,20): #for each sport
        #compare sport to each remaining sport
        c = col
        for c in range(c,20):
            psychdistance = 1-psychsimilarities[row][c]
            mdsdistance = np.sqrt(np.square(mdsdistances[row][0] - mdsdistances[c][0]) + np.square(mdsdistances[
row][1] - mdsdistances[c][1]))
            stress += np.square(psychdistance-mdsdistance)
        col += 1
    return stress

def findgradient(mdsdistances, psychsimilarities, delta):
    gradient= np.copy(mdsdistances)

    #Find the partial derivative by changing x and keeping y & changing y and keeping x constant.
    for i in np.arange(len(mdsdistances) -1): #for all coordinates in our grid

        #FIRST SOLVE FOR X
        coordinatesmodplus = np.copy(mdsdistances)
        coordinatesmodminus = np.copy(mdsdistances)

        coordinatesmodplus[i][0]+= delta #changing x value, and keeping y the same
        fpdx = stresscalc(coordinatesmodplus, psychsimilarities) #f(changed concept x, y remains)

        coordinatesmodminus[i][0]-=delta #changing x value, keeping y the same
        fmdx=stresscalc(coordinatesmodminus, psychsimilarities) #f(changed concept x, y remains)

        partialderivativex=(fpdx-fmdx)/(2*delta)

        gradient[i][0]=partialderivativex #update that part of the gradient with the partial derivative found fo
r element i of values

        #THEN SOLVE FOR Y
        coordinatesmodplus = np.copy(mdsdistances)
        coordinatesmodminus = np.copy(mdsdistances)

        coordinatesmodplus[i][1]+= delta #changing y value, and keeping x the same
        fpdy = stresscalc(coordinatesmodplus, psychsimilarities) #f(changed concept y, x remains)

        coordinatesmodminus[i][1]-=delta #changing y value, keeping x the same
        fmdy=stresscalc(coordinatesmodminus, psychsimilarities) #f(changed concept y, x remains)

        partialderivativey=(fpdy-fmdy)/(2*delta)

        gradient[i][1]=partialderivativey #update that part of the gradient with the partial derivative found fo
r element i of values

    return gradient

def gradientcheck(gradient):
    passed = []
    for i in range(len(gradient)):
        if gradient[i][0] <= 0.1:
            passed.append(1)
        if gradient[i][1] <= 0.1:
            passed.append(1)
    return sum(passed)

names = ['football', 'baseball', 'basketball', 'tennis',
         'softball', 'canoeing', 'handball', 'rugby',
         'hockey', 'icehockey', 'swimming', 'track',
         'boxing', 'volleyball', 'lacrosse', 'skiing',
         'golf', 'polo', 'surfing', 'wrestling', 'gymnastics']

xpoints = []
ypoints = []
stressvals = []
iterations= 0

#PLOT RANDOM POINTS FIRST
for i in range(len(mdsdistances)):
    xpoints.append(mdsdistances[i][0])
    ypoints.append(mdsdistances[i][1])

# fig, ax = plt.subplots()
# ax.scatter(xpoints,ypoints)
# for i, txt in enumerate(names):
#     ax.annotate(txt, (xpoints[i], ypoints[i]))
n = 0

while n < 10:
    mdsdistances = np.random.uniform(0, 1, size=(21,2))
    gradient = findgradient(mdsdistances, psychsimilarities, delta)
    stress = stresscalc(mdsdistances, psychsimilarities)
    checkval = gradientcheck(gradient)
    while stress >= 10:
        for i in range(len(gradient)):
            mdsdistances[i][0] += (-scaling * gradient[i][0])
            mdsdistances[i][1] += (-scaling * gradient[i][1])
            xpoints[i] = (mdsdistances[i][0])
            ypoints[i] = (mdsdistances[i][1])
        gradient = findgradient(mdsdistances, psychsimilarities, delta)
        stress = stresscalc(mdsdistances, psychsimilarities)
       # checkval = gradientcheck(gradient)

    fig, ax = plt.subplots()
    ax.scatter(xpoints,ypoints)
    for i, txt in enumerate(names):
        ax.annotate(txt, (xpoints[i], ypoints[i]), fontsize = 10)
    n += 1
```