Section 1: Data Operations using Python (3 points per problem)

Problem 1 - On the Football data, find the average difference between game predictions and actual outcomes, across all games. The HomeScore and AwayScore are the actual scores, while Prediction is an estimate of how much the home team will win or lose by. In arithmetic terms, Outcome = (HomeScore - AwayScore), and Difference = (Prediction - Outcome). Your Python program should print a single positive or negative number for the average difference.

```python
total = 0
for game in football:
    total += float(game['Prediction']) - (float(game['HomeScore']) - float(game['AwayScore']))
print total/len(football)
```

Problem 2 - On the Football data, find all pairs of teams where the two teams played each other in 1998 and 1999 in the same configuration (the same team was home and the same team was away), and in 1998 the home team won while in 1999 the away team won. Your Python program should print all such pairs of teams.

```python
for game1 in football:
    for game2 in football:
        if game1['Year'] == '1998' and game2['Year'] == '1999' 
        and game1['Home'] == game2['Home'] 
        and game1['Away'] == game2['Away'] 
        and float(game1['HomeScore']) > float(game1['AwayScore']) 
        and float(game2['HomeScore']) < float(game2['AwayScore']):
            print 'Team1: ', game1['Home'], '|', 'Team2: ', game1['Away']
```

Problem 3 - On the Schoolkids data, count how many 4th graders, 5th graders, and 6th graders think Looks are more important than Grades in determining popularity, separately for each of the three grades. Your Python program should print the three results, specifying which number goes with which grade.

```python
fourth = 0
fifth = 0
sixth = 0
for s in schoolkids:
    if s['Grade'] == '4' and s['Looks'] < s['Grades']: fourth += 1
    if s['Grade'] == '5' and s['Looks'] < s['Grades']: fifth += 1
    if s['Grade'] == '6' and s['Looks'] < s['Grades']: sixth += 1
print 'Fourth: ', fourth, ' Fifth: ', fifth, ' Sixth: ', sixth
```

Problem 4 - On the Schoolkids data, find the oldest students. For each one your Python program should print their gender, grade, age, and school.

```python
maxage = max([int(s['Age']) for s in schoolkids])
for s in schoolkids:
    if int(s['Age']) == maxage:
        print s['Gender'], '|', s['Grade'], '|', s['Age'], '|', s['School']
```
Extra Credit Problem EX1 - Find the average prediction for games where the home team won, and the average prediction for games where the away team won. Your Python program should print the two numbers, specifying which is for home wins and which is for away wins.

```python
Hpredictions = []
Apredictions = []
for game in football:
    if float(game['HomeScore']) > float(game['AwayScore']):
        Hpredictions.append(float(game['Prediction']))
    if float(game['HomeScore']) < float(game['AwayScore']):
        Apredictions.append(float(game['Prediction']))
print 'Average Prediction for Home wins:', sum(Hpredictions)/len(Hpredictions)
print 'Average Prediction for Away wins:', sum(Apredictions)/len(Apredictions)
```

Extra Credit Problem EX2 - What is the overall ranking of Grades, Sports, Looks, and Money in perceived popularity impact across all of the students? Your Python program should print the four factors in order of overall importance, i.e., the factor with the highest importance should be first.

```python
grades = 0
sports = 0
looks = 0
money = 0
for s in schoolkids:
    grades += int(s['Grades'])
sports += int(s['Sports'])
looks += int(s['Looks'])
money += int(s['Money'])
factors = [(Grades,grades),(Sports,sports),(Looks,looks),(Money,money)]
sorted_factors = sorted(factors,key=lambda x: x[1])
print sorted_factors[0][0], sorted_factors[1][0], sorted_factors[2][0], sorted_factors[3][0]
```

Section 2: Data Mining using SQL (3 points per problem)

Problem 1 - In class we developed SQL queries to compute frequent itemsets of two movies and three movies over a very small set of shopping data. Copy the two queries from the lecture notes into your notebook and modify them to compute frequent itemsets of two and three movies over the movie data using support threshold .03 (not 0.3!). Then write a third query that computes frequent itemsets of four movies over the movie data, still using support threshold .03. If your queries are correct, you should find 20 itemsets of two, 14 itemsets of three, and 3 itemsets of four.

```sql
select M1.movie, M2.movie
from Movies M1, Movies M2
where M1.uid = M2.uid
and M1.movie < M2.movie
group by M1.movie, M2.movie
having count(*) > (select count(distinct uid) * .03 from Movies)
```

```sql
select M1.movie, M2.movie, M3.movie
```
Problem 2 - Write SQL queries to compute association rules over the movie data that have one movie on the left-hand side and one movie on the right-hand side, with support threshold .03 and confidence threshold 0.5. If your approach is correct, you should find 10 association rules.

First compute frequent itemsets of one, with support

\[
\text{create table Frequent1(movie, numbaskets);} \n\text{insert into Frequent1} \n\text{select movie, count(*)} \n\text{from Movies} \n\text{group by movie} \n\text{having count(*) > (select count(distinct uid) * .03 from Movies)} \n\]

Next frequencies for all pairs

\[
\text{create table Pairs(movie1, movie2, numbaskets);} \n\text{insert into Pairs} \n\text{select M1.movie, M2.movie, count(*)} \n\text{from Movies M1, Movies M2} \n\text{where M1.uid = M2.uid} \n\text{and M1.movie <> M2.movie} \n\text{group by M1.movie, M2.movie} \n\]

Finally keep pairs with frequent LHS and sufficient confidence:

\[
\text{select Pairs.movie1, Pairs.movie2} \n\text{from Pairs, Frequent1} \n\text{where Pairs.movie1 = Frequent1.movie} \n\text{and (1.0*Pairs.numbaskets)/(1.0*Frequent1.numbaskets) > 0.5} \n\]

Extra Credit Problem - Write SQL queries to compute association rules over the movie data that have two movies on the left-hand side and one movie on the right-hand side, with support threshold .02 and confidence threshold 0.8. If your approach is correct, you should find 7 association rules.

First compute frequent itemsets of two, with support

\[
\text{create table Frequent2(movie1, movie2, numbaskets);} \n\text{insert into Frequent2} \n\text{select M1.movie, M2.movie, count(*)} \n\text{from Movies M1, Movies M2} \n\text{where M1.uid = M2.uid} \n\]
and M1.movie < M2.movie
  group by M1.movie, M2.movie
having count(*) > (select count(distinct uid) * .02 from Movies)

Next frequencies for all triples:

create table Triples(movie1, movie2, movie3, numbaskets);
insert into Triples
select M1.movie, M2.movie, M3.movie, count(*)
from Movies M1, Movies M2, Movies M3
where M1.uid = M2.uid and M2.uid = M3.uid
  and M1.movie <> M2.movie and M2.movie <> M3.movie and M1.movie<>M3.movie
  group by M1.movie, M2.movie, M3.movie

Finally pairs with frequent LHS and sufficient confidence:

select T.movie1, T.movie2, T.movie3
from Triples T, Frequent2 F
where T.movie1 = F.movie1 and T.movie2 = F.movie2
  and T.movie3 <> F.movie1 and T.movie3 <> F.movie2
  and (1.0*T.numbaskets)/(1.0*F.numbaskets) > 0.8

Section 3: Data Mining using Python (3 points per problem)

Problem 1 - In class we developed Python code to compute frequent itemsets of two movies
and three movies over a very small set of shopping data. Copy the code from the lecture notes
into your notebook and modify it to compute frequent itemsets of two and three movies over the
movie data using support threshold 0.03 (not 0.3!) Then extend the code to compute frequent
itemsets of four movies over the movie data, still using support threshold 0.03. If your code is
correct, you should find 20 itemsets of two, 14 itemsets of three, and 3 itemsets of four.

transactions={} # dictionary from TID to list of items
items={} # dictionary from item to list of TIDs
with open('Movies.csv', 'rU') as csvfile:
    data = csv.reader(csvfile)
    for row in data:
        if row[0] not in transactions: transactions[row[0]]= [row[1]]
        else: transactions[row[0]].append(row[1])
        if row[1] not in items: items[row[1]]= [row[0]]
        else: items[row[1]].append(row[0])

numtransactions = len(transactions)

# compute all pairs of items, alphabetical
pairs = []
for i1 in items:
    for i2 in items:
        if i1<i2: pairs.append([i1,i2,0])

# append number of transactions containing each pair
for p in pairs:
    for t in transactions:

# compute frequent itemsets of two
Problem 2 - Write Python code to compute association rules over the movie data that have one movie on the left-hand side and one movie on the right-hand side, with support threshold .03 and confidence threshold .5. If your approach is correct, you should find 10 association rules.
if len(items[i]) > numtransactions * .03:
   frequent1.append([i, len(items[i])])

# compute all pairs of items, including reverse
pairs = []
for i1 in items:
   for i2 in items:
      if i1 <> i2: pairs.append([i1,i2,0])

# append number of transactions containing each pair
for p in pairs:
   for t in transactions:

# determine which pairs represent association rules
rules = []
for p in pairs:
   for f in frequent1:
      if p[0] == f[0] and float(p[2])/float(f[1]) > 0.5:
         rules.append(p)
print 'ONE-ONE ASSOCIATION RULES:'
for r in rules: print '  ', r[0], '->', r[1]

Extra Credit Problem - Write Python code to compute association rules over the movie data
that have two movies on the left-hand side and one movie on the right-hand side, with support
threshold .02 and confidence threshold 0.8. If your approach is correct, you should find 7
association rules.

# compute frequent itemsets of two, alphabetical
frequent2 = []
for p in pairs:
   if p[0] < p[1] and float(p[2])/float(numtransactions) > .02:
      frequent2.append(p)

# compute all triples of items where first two are in frequent itemsets of two,
# nonalphabetical
triples = []
for f in frequent2:
   for i in items:
      if f[0] <> i and f[1] <> i: triples.append([f[0],f[1],i,0])

# append number of transactions containing each triple
for tr in triples:
   for t in transactions:
      if tr[0] in transactions[t] and tr[1] in transactions[t] \

# determine which triples represent association rules
rules = []
for tr in triples:
   for p in pairs:
      if tr[0] == p[0] and tr[1] == p[1] \
      and float(tr[3])/float(p[2]) > 0.8: rules.append(tr)
print 'TWO-ONE ASSOCIATION RULES:'


Section 4: Regression and Correlation in Google Sheets (2 points per problem)

In Assignment 1 Problem 13 you created a scatterplot using the football data where the x-axis was the Prediction and the y-axis was the actual outcome (HomeScore minus AwayScore). Most of you concluded that there was at least some correlation between the two measures.

**Problem 1** - To make that conclusion concrete, create the scatterplot again in Google Sheets, use the Trendline (Linear) feature to plot the linear regression line, and select the R$^2$ button to see what Google computes as the coefficient of determination.

[Scatterplot image with R$^2 = 0.195$]

**Problem 2** - Correlation between two measures shouldn’t be affected by which measure is on the x-axis and which on the y-axis. Repeat Problem 1 with the axes reversed. Did you get the same R$^2$ value? **YES**

[Scatterplot image with R$^2 = 0.195$]

Section 5: Plotting and Regression in Python (6 points)
For this problem you are given a set of (x,y) points in coordinate space, and three lines; we provide Python code that reads the points and lines from files. Your job is to write Python code that computes the sum of squares error (SSE) for each line with respect to the points, determines colors for the lines based on how closely they fit the points -- green for the closest fit (smallest SSE), orange for the middle, and red for the worst fit (largest SSE) -- then draws a plot showing the points and the colored lines.

# Part 1 (provided): Read files and set up data structures
import csv
x=[]
y=[]
with open('datapoints.csv', 'rU') as csvfile:
    data = csv.reader(csvfile)
    for row in data:
        x.append(float(row[0]))
        y.append(float(row[1]))
lines=[]
with open('lines.csv', 'rU') as csvfile:
    data = csv.reader(csvfile)
    for row in data:
        lines.append((row[0],row[1]))
line1 = {'a': float(lines[0][0]), 'b': float(lines[0][1])}
line2 = {'a': float(lines[1][0]), 'b': float(lines[1][1])}
line3 = {'a': float(lines[2][0]), 'b': float(lines[2][1])}

# Part 2 (students): Compute SSEs and set line colors
error1 = 0
for i in range(0,len(x)):
    error1 += (line1['a']*x[i] + line1['b'] - y[i])**2
error2 = 0
for i in range(0,len(x)):
    error2 += (line2['a']*x[i] + line2['b'] - y[i])**2
error3 = 0
for i in range(0,len(x)):
    error3 += (line3['a']*x[i] + line3['b'] - y[i])**2

color_order = ['green', 'orange', 'red']
line_error = {1: error1, 2: error2, 3: error3}
sorted_le = sorted(line_error, key=line_error.get)
color1 = color_order[sorted_le[0]-1]
color2 = color_order[sorted_le[1]-1]
color3 = color_order[sorted_le[2]-1]

# Part 3 (provided + student): Plot points and lines
%matplotlib notebook
import matplotlib.pyplot as plt
plt.plot(x,y,'o')
xmax = max(x)
plt.plot([0,xmax],[line1['b'], line1['a']*xmax + line1['b']], color = color1)
plt.plot([0,xmax],[line2['b'], line2['a']*xmax + line2['b']], color = color2)
plt.plot([0,xmax],[line3['b'], line3['a']*xmax + line3['b']], color = color3)
plt.show()