IE 588 Project Report

January 8, 2018

Mine Melodi Caliskan 2015705009

1 Party

1.1 Problem Description

Students living in Hisarustu, Istanbul have limited free time and they want to spend their Friday night at the one and only pub, La Liberta Plus, which is a very small. Therefore, it's very difficult to find a place to hang out with their friends. If the pub is too crowded when a student enters, she feels happy however if it is more than that and also if she decides to stay home she feels upset. Our task is to capture the effects of students types to overall happiness and themselves. It's important to remark that if all students use the same decision rule to decide whether to go or not, it's unavoidable to fail since in this sense the bar would be either full of people or full-empty. Therefore, there should be a group of decision rules and students must learn from their history in order to choose the following decision rule. Every students' action results from the other students' actions and also themselves. Our task is to answer the question of whether or not the equal happiness with higher than a certain degree of total happiness could be provided. Therefore it is interesting to analyze.

Modeling Questions

- How would overall happiness change depending on the type of the students?
- How would groups effect each others happiness?
- Which agent type results in better in the long-run?

Method Justification

We can't predict easily when to go to party since there is highly complex interaction effect, however components of the system are simple, therefore it's better to use ABM for this problem.

- **the puzzle component:** If everybody prefers the same strategy to decide whether go bar or not, all of them would make themselves unhappy since the bar will be full or empty and our happiness criteria would not be satisfied. So that, there should be multiple strategies and these strategies outputs will be affected with each other and these interactions are complex to define directly.
- **dynamic nature:** This model is dynamic since the decision strategies depend on historical data which changes over time. People will choose a strategy based on their experience in advance and their experience change over time and they depend on themselves (autocorrelation) and the other people.(cross-correlation)

- **importance of agent heterogeneity:** Every person has different behaviors and characteristic features.

1.2 Agent Types

Copies

- Shape: Star
- At each run randomly picks an agent type and copies behaivor of one-of them

Analyzers: * Shape: Human * Keeps History for 3 days * Has 3 strategies * Scores of stragies are kept for 3 days also.
Always: Always goes to party * Shape: Face Happy
Nevers: Never goes to party * Shape: X
Randomms: Goes party with probability %50 * Shape: Butterfly

1.2.1 Global Variables

- **number-of-people-at-bar:** the current number of people at the bar
- **outside-patches:** outside of the bar
- **party-patches :** inside of the bar
- **days-passed :** days since the beginning passed
- **total-total-happiness :** total happiness of all agents
- **total-agents:** total number of all type of agents
- **number-of-randomms:** number of agent type "randomms" which is determined by : total-agents- sum(number of other type of agents)
- **crowd-threshold:** threshold for the crowdness of the party of agents which is determined by: total-agents * crowd-percentage

Turtles (All) Variables

- happiness
- go?

1.2.2 Agent Variables

Agents are assumed to be have local information, i.e when they go to the party they can count the total number of agents perfectly. However, before entering the bar they dont know.
Copies

- **prediction-copy**

They are assumed to be to have contact of all agents and learn their decision to decide to go to party or not.
Analyzers

- **history-analyzer**
- *prediction-analyzer**
- **score-1 :** score of strategy-1

- **score-2 :** score of strategy-2
- **score-3 :** score of strategy-3
- **chosen :** corresponds to chosen strategies symbol as s1,s2,s3

Always * prediction-alway
Nevers

- **prediction-never**

Randomms

- **prediction-randomm**

1.3 Procedures

Setup

- total-agents are set to 100
- bar patches drew ask disk centered at the patch (0, 0) with color pink
- outside patches are in color blue
- turtles are created as follows: * number of user specified * for randomm type agents only, they are create with the number of left size of total agents 100.
- variables of turtles are initialized as follows: * for analyzers

```
;initialize history of 3 days' attendence randomly between 0 and number of total-agents
set history-analyzer (list (random total-agents)  (random total-agents) (random total-ag
show history-analyzer
set prediction-analyzer (random total-agents)
show prediction-analyzer
;initialize scores randomly (0 or 1) (failure or success) for 3 days
set score-1 (list (random 1) (random 1) (random 1))
set score-2 (list (random 1) (random 1) (random 1) )
set score-3  (list (random 1) (random 1) (random 1))]
```

- happiness of all turtles initialized randomly between 0,100

Go

```
to go
    ; scale agensts color with respect to their happiness levels
  ask turtles [ set color scale-color magenta happiness 0 100 ]
  go-analyzers
  go-always
  go-nevers
  go-randomms
  go-copies
```

```
    set number-of-people-at-bar count turtles-on party-patches

    update-happiness-always
    ;update-happiness-nevers (no need)
    update-happiness-randomms
    update-happiness-copies
    update-happiness-analyzers

    set days-passed days-passed + 1

    ifelse days-passed < 366 [ ; run for a year

      tick

    ]
      [ ask turtles [move-to one-of outside-patches] stop]

  end
```

go-always
Always go to party
go-randomms
goes to party with prob 0.5
go-analyzers
They store historical attendence values for 3 days and they use scores of strategies to decide which one to use to go to the party or not. Then they increase or decrease the score of the strategy with respect to success or fail. They also store scores of strategies for 3 days.
go-copies
randomly chooses and agent type and decides to go to party with the decision of one-of them
update-happiness
Update of happiness are the same except analyzers

- update for other turtles:

  to update-happiness ask “other-turtles” [if go? = True and happiness > 0 and happiness < 100 [ifelse number-of-people-at-bar > crowd-threshold [set happiness happiness - 1] [set happiness happiness + 1]]] end

- update for analyzers:

  to update-happiness-analyzers ask analyzers [if go? = True and happiness > 0 and happiness < 100 [ifelse number-of-people-at-bar > crowd-threshold [set happiness happiness - 1 ifelse chosen = “s1” ;update scores of strattegies as well as happiness level [set score-1 fput 0 but-last score-1] [ifelse chosen = “s2” [set score-2 fput 0 but-last score-2] [set score-3 fput 0 but-last score-3]]

```
    ]
    [set happiness happiness + 1
      ifelse chosen = "s1"
      [set score-1 fput 1  but-last score-1 show score-1] ; scores are also kept for 3 day
```

```
        [ifelse chosen = "s2"
           [set score-2 fput 1  but-last score-2 show score-2]
           [set score-3 fput 1 but-last  score-3 show score-3]]
         ;history is kept for 3 days
    ]] set history-analyzer fput number-of-people-at-bar but-last history-analyzer show "histo
```

end

update-scores
Strategies
Only analyzers use strategies

- strategy-1

to strategy-1
;predicts next attendence as last
set prediction-analyzer last history-analyzer
end

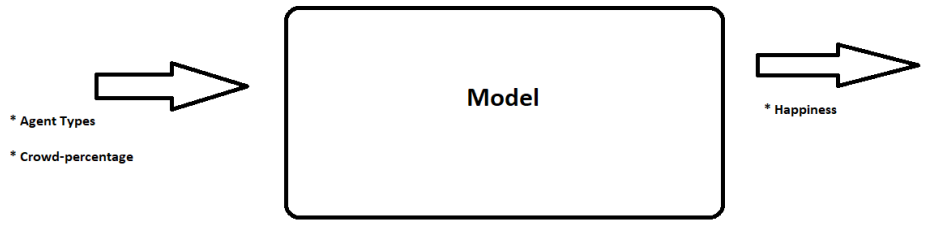
- strategy-2

to strategy-2
;predicts next attendence as first
set prediction-analyzer first history-analyzer
end

- strategy-3

to strategy-3
;predicts next attendence as total-agents - last
set prediction-analyzer (total-agents - last history-analyzer) end
update strategies
to update-strategy
ask analyzers [

```
;chooses strategy with maximum average score value

let m1 mean score-1
let m2 mean score-2
let m3 mean score-3
let max-score  max (list m1  m2 m3)
ifelse max-score = mean score-1
[set chosen "s1" strategy-1]
[ifelse max-score = mean score-2
  [set chosen "s2" strategy-2]
  [set chosen "s3" strategy-3]]

  ]
  end
```



model

1.4 Evaluation

Used parameters:

Experiment1:

```
["number-of-copies" 20]
["number-of-analyzers" 20]
["number-of-nevers" 20]
["crowd-percentage" 0.6]
["number-of-always" 20]
```

Experiment2:

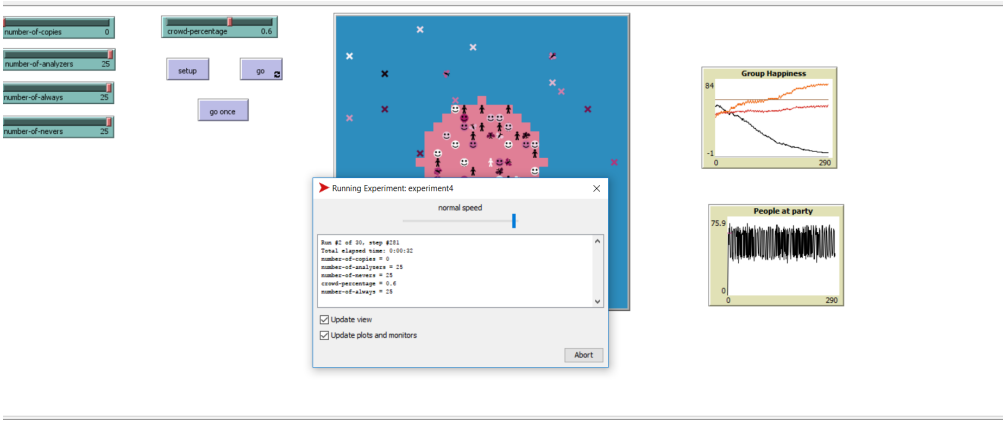
```
 ["number-of-copies" 20]
["number-of-analyzers" 20]
["number-of-nevers" 20]
["crowd-percentage" 0.3]
["number-of-always" 20]
```

Experiment3:

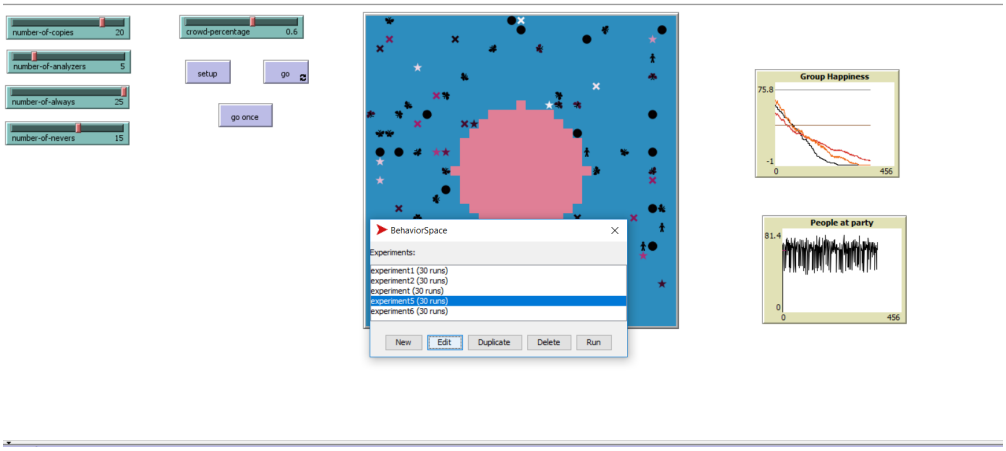
```
["number-of-copies" 25]
["number-of-analyzers" 25]
["number-of-nevers" 25]
["crowd-percentage" 0.6]
["number-of-always" 25]
```

Experiement4:

```
["number-of-copies" 20]
["number-of-analyzers" 5]
["number-of-nevers" 15]
["crowd-percentage" 0.6]
["number-of-always" 25]
```



exp4



exp5

Experiment5:

```
["number-of-copies" 50]
["number-of-analyzers" 50]
["number-of-nevers" 0]
["crowd-percentage" 0.6]
["number-of-always" 0]
```

Experiment6:

```
["number-of-copies" 1]
  ["number-of-analyzers" 25]
  ["number-of-nevers" 3]
  ["crowd-percentage" 0.7]
  ["number-of-always" 5]
```

```
In [1]: import pandas as pd
```

```
In [2]: df1 = pd.read_csv('experiment1-table.csv')
        df2 = pd.read_csv('experiment2-table.csv')
        df3=pd.read_csv('experiment3-table.csv')
        df4=pd.read_csv('experiment4-table.csv')
        df5=pd.read_csv('experiment5-table.csv')
        df6=pd.read_csv('experiment6-table.csv')
        df7=pd.read_csv('experiment8-table.csv')

In [3]: df=df1.append(df2)
        df=df.append(df3)
        df=df.append(df4)
        df=df.append(df5)
        df=df.append(df6)
        df=df.append(df7)

In [4]: df=df.reset_index()
        df=df.drop('index',axis=1)
        df=df.drop('[step]',axis=1)

In [5]: df.head()
```

Out[5]:

	number-of-copies	number-of-analyzers	number-of-nevers	crowd-percentage	\
0	20	20	20	0.6	
1	20	20	20	0.6	
2	20	20	20	0.6	
3	20	20	20	0.6	
4	20	20	20	0.6	

	number-of-always	analyzers-happiness	copies-happiness	always-happiness	\
0	20	0.0	41.60	49.25	
1	20	0.0	54.75	43.80	
2	20	0.0	51.80	52.75	
3	20	0.0	46.75	55.50	
4	20	0.0	52.40	56.35	

	nevers-happiness	randomms-happiness	all-happiness	avg-total
0	48.75	44.45	36.81	195.103562
1	48.85	58.80	41.24	217.143288
2	45.30	59.05	41.78	217.402329
3	42.25	44.75	37.85	200.613151
4	45.30	44.35	39.68	209.760411

```
In [6]: df.tail()
```

Out[6]:

	number-of-copies	number-of-analyzers	number-of-nevers	\
205	1	25	3	
206	1	25	3	
207	1	25	3	
208	1	25	3	

```
209                 1                   25                 3

     crowd-percentage  number-of-always  analyzers-happiness  \
205               0.7                 5                100.0
206               0.7                 5                100.0
207               0.7                 5                100.0
208               0.7                 5                100.0
209               0.7                 5                100.0

     copies-happiness  always-happiness  nevers-happiness  randomms-happiness  \
205              88.0             100.0         22.333333          100.000000
206              14.0             100.0         31.666667           95.454545
207              90.0             100.0         50.333333           98.484848
208              90.0             100.0         28.000000          100.000000
209              19.0             100.0         60.333333          100.000000

     all-happiness   avg-total
205          97.55  387.593051
206          94.09  318.078751
207          97.41  423.052259
208          97.74  402.147880
209          98.00  359.522413
```

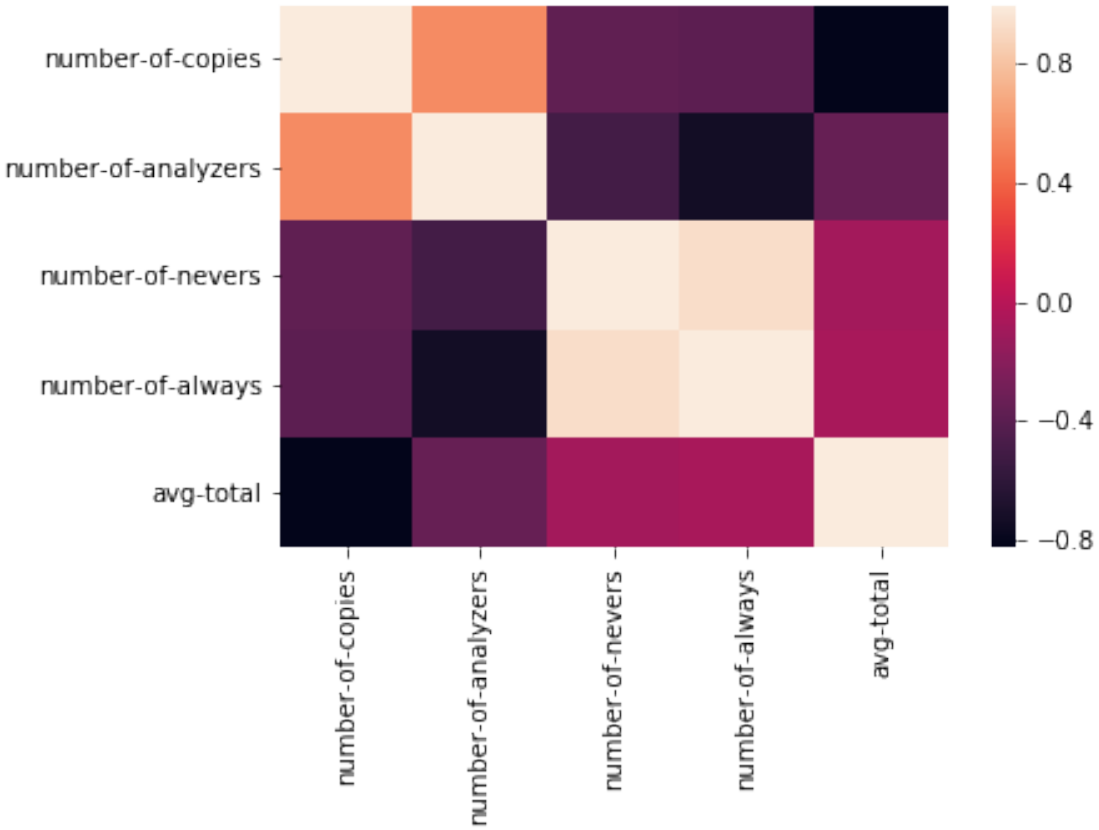
```
In [7]: df_all=df.drop('avg-total',axis=1)

In [8]: df_all=df.drop('analyzers-happiness',axis=1)

In [9]: df_all=df.drop('copies-happiness',axis=1)

In [10]: df_all=df.drop('nevers-happiness',axis=1)

In [11]: df_all=df.drop('randomms-happiness',axis=1)

In [12]: df_grouped=df_all.groupby(['number-of-copies','number-of-analyzers','number-of-nevers

In [13]: df_grouped=pd.DataFrame(df_grouped).reset_index()

In [14]: df_grouped.describe()
```

```
Out[14]:        number-of-copies  number-of-analyzers  number-of-nevers  \
         count          6.000000             6.000000          6.000000
         mean          19.333333            25.000000         14.666667
         std           18.348479            14.491377         10.893423
         min            0.000000             5.000000          0.000000
         25%            5.750000            21.250000          6.000000
         50%           20.000000            25.000000         17.500000
         75%           23.750000            25.000000         23.750000
         max           50.000000            50.000000         25.000000
```

```
       number-of-always   avg-total
count          6.000000    6.000000
mean          16.666667  187.829860
std           11.254629  102.178956
min            0.000000   59.211602
25%            8.750000  158.110380
50%           22.500000  168.553004
75%           25.000000  195.728544
max           25.000000  371.244985
```

In [15]:
```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

auto_df =df_grouped

# calculate the correlation matrix
corr = auto_df.corr()

# plot the heatmap
sns.heatmap(corr,
        xticklabels=corr.columns,
        yticklabels=corr.columns)

cmap = sns.diverging_palette(5, 250, as_cmap=True)

def magnify():
    return [dict(selector="th",
                 props=[("font-size", "7pt")]),
            dict(selector="td",
                 props=[('padding', "0em 0em")]),
            dict(selector="th:hover",
                 props=[("font-size", "12pt")]),
            dict(selector="tr:hover td:hover",
                 props=[('max-width', '200px'),
                        ('font-size', '12pt')])
]

corr.style.background_gradient(cmap, axis=1)\
    .set_properties(**{'max-width': '120px', 'font-size': '15pt'})\
    .set_caption("Hover to magify")\
    .set_precision(2)\
    .set_table_styles(magnify())
```

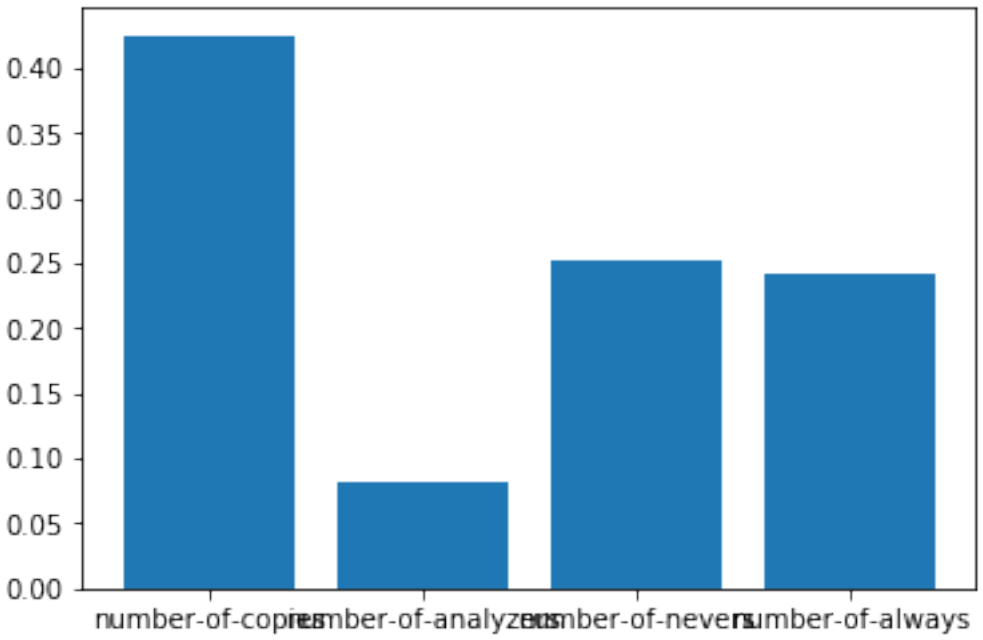
Out[15]: <pandas.io.formats.style.Styler at 0x17ea8a9c5f8>



We can see above that number-of-nevers and number-of-copies is highly correlated with avg-total happiness. number-of-copies randomly chooses an agent type and acts like them, so that it effects overall happiness as agent by theirself effects. And as we increase the number of nevers the others will be decreased so that the risk of unhappiness will be decreased.

```
In [16]: from sklearn.ensemble import RandomForestRegressor
         # split into input and output
         X = df_grouped[['number-of-copies','number-of-analyzers','number-of-nevers','number-o
         y = df_grouped['avg-total'].values
         # fit random forest model
         model = RandomForestRegressor(n_estimators=500, random_state=1)
         model.fit(X, y)
         # show importance scores
         print(model.feature_importances_)
         # plot importance scores
         names = df_grouped.columns.values[0:-1]
         ticks = [i for i in range(len(names))]
         plt.bar(ticks, model.feature_importances_)
         plt.xticks(ticks, names)
         plt.show()
```

```
[ 0.42555691  0.08160587  0.2515696   0.24126762]
```




```
In [17]: df.describe()
```

Out[17]:

	number-of-copies	number-of-analyzers	number-of-nevers
count	210.000000	210.000000	210.000000
mean	19.428571	24.285714	15.428571
std	15.546108	12.401354	9.416322
min	0.000000	5.000000	0.000000
25%	1.000000	20.000000	3.000000
50%	20.000000	25.000000	20.000000
75%	25.000000	25.000000	25.000000
max	50.000000	50.000000	25.000000

	crowd-percentage	number-of-always	analyzers-happiness
count	210.000000	210.000000	210.000000
mean	0.571429	17.142857	14.936619
std	0.116335	9.606047	34.503890
min	0.300000	0.000000	0.000000
25%	0.600000	5.000000	0.000000
50%	0.600000	20.000000	0.000000
75%	0.600000	25.000000	3.087500
max	0.700000	25.000000	100.000000

	copies-happiness	always-happiness	nevers-happiness

```
count        210.000000        210.000000        210.000000
mean          42.027857         41.582952         41.753254
std           21.369239         39.874557         19.033677
min            0.000000          0.000000          0.000000
25%           38.250000          0.000000         40.025000
50%           47.875000         44.910000         47.966667
75%           52.587500         87.510000         53.027500
max           92.000000        100.000000         76.333333

       randomms-happiness  all-happiness   avg-total
count          210.000000     210.000000  210.000000
mean            32.066977      41.680048  186.478839
std             36.873742      25.044487   89.340448
min              0.000000      14.440000   52.742192
25%              0.000000      22.097500  145.934970
50%             10.585714      36.345000  166.332098
75%             62.180000      50.647500  208.498705
max            100.000000      98.390000  423.052259
```

```
In [18]: df[['analyzers-happiness', 'copies-happiness','always-happiness','nevers-happiness','r
```

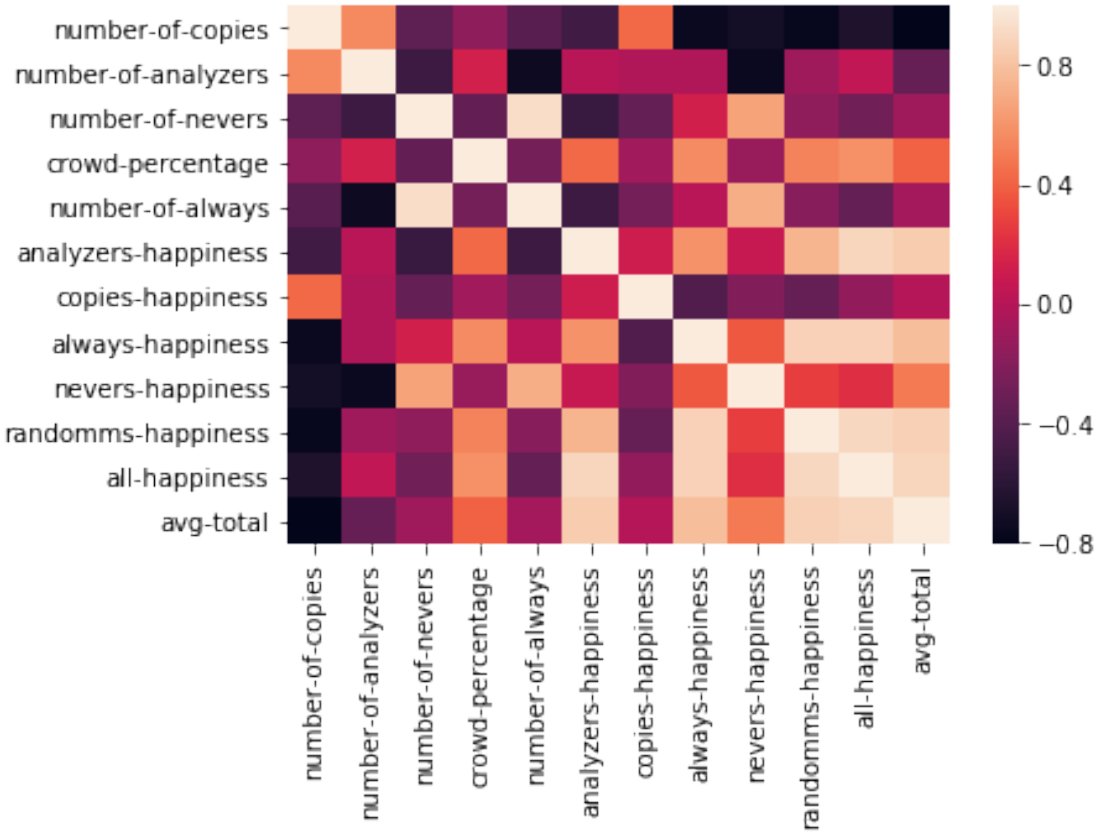
```
Out[18]:        analyzers-happiness  copies-happiness  always-happiness  \
count            210.000000        210.000000        210.000000
mean              14.936619         42.027857         41.582952
std               34.503890         21.369239         39.874557
min                0.000000          0.000000          0.000000
25%                0.000000         38.250000          0.000000
50%                0.000000         47.875000         44.910000
75%                3.087500         52.587500         87.510000
max              100.000000         92.000000        100.000000

       nevers-happiness  randomms-happiness
count        210.000000          210.000000
mean          41.753254           32.066977
std           19.033677           36.873742
min            0.000000            0.000000
25%           40.025000            0.000000
50%           47.966667           10.585714
75%           53.027500           62.180000
max           76.333333          100.000000
```

Here we can see that in our experiments average happiness of copies is the max. And it follows with "nevers" since they dont take any risk and stays as initialized happiness values. Analyzers has no advantage over others while their happiness depend on them.

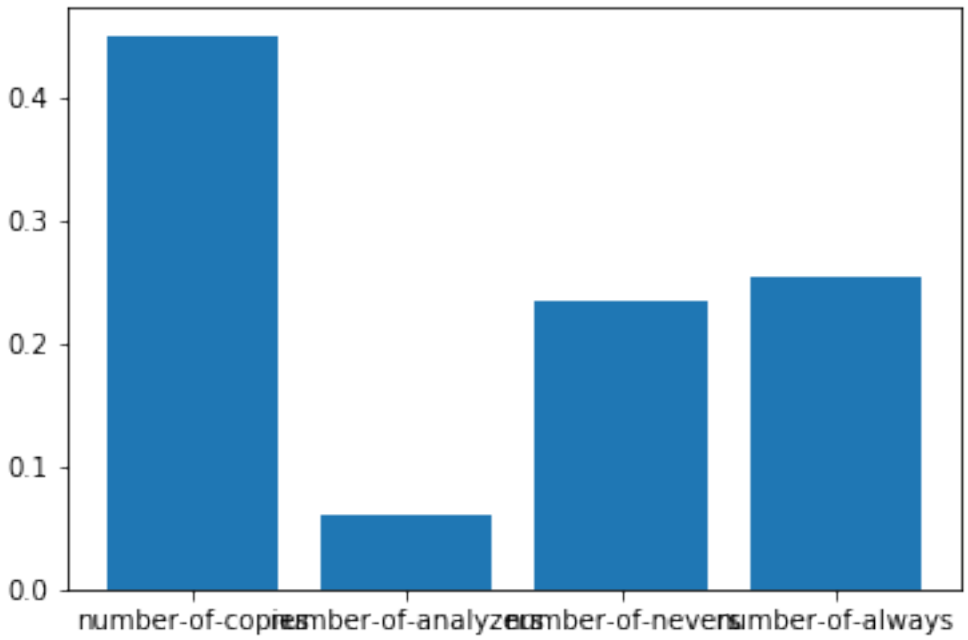
```
In [19]: import seaborn as sns
         import matplotlib.pyplot as plt
         %matplotlib inline
```

```
auto_df =df
# calculate the correlation matrix
corr = auto_df.corr()

# plot the heatmap
sns.heatmap(corr,
        xticklabels=corr.columns,
        yticklabels=corr.columns)

cmap = sns.diverging_palette(5, 250, as_cmap=True)

def magnify():
    return [dict(selector="th",
                 props=[("font-size", "7pt")]),
            dict(selector="td",
                 props=[('padding', "0em 0em")]),
            dict(selector="th:hover",
                 props=[("font-size", "12pt")]),
            dict(selector="tr:hover td:hover",
                 props=[('max-width', '200px'),
                        ('font-size', '12pt')])
]

corr.style.background_gradient(cmap, axis=1)\
    .set_properties(**{'max-width': '120px', 'font-size': '15pt'})\
    .set_caption("Hover to magify")\
    .set_precision(2)\
    .set_table_styles(magnify())
```

Out[19]: <pandas.io.formats.style.Styler at 0x17ea9cbb7f0>

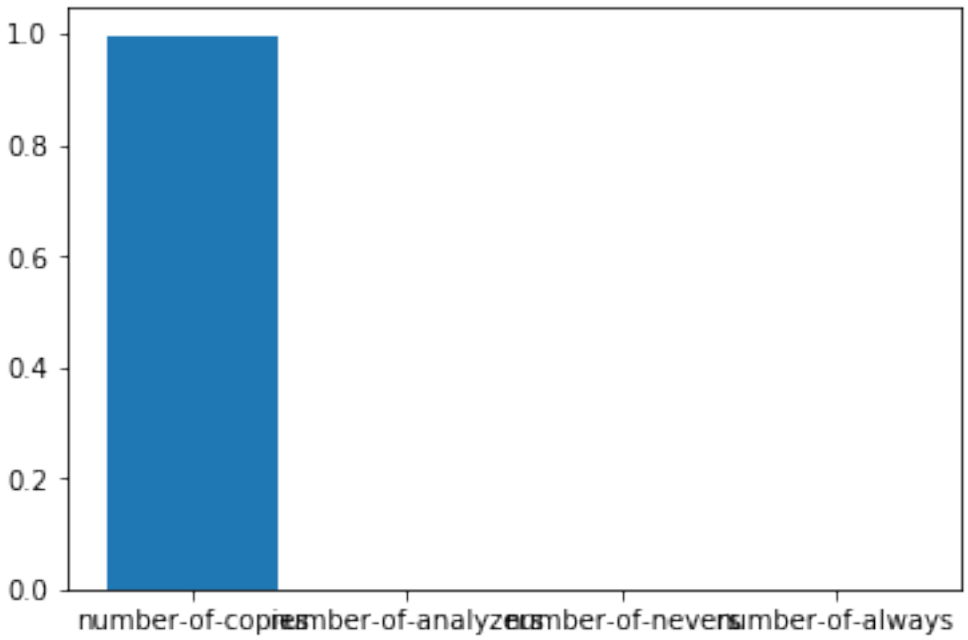



```
In [20]: from sklearn.ensemble import RandomForestRegressor
         #fit a randomforest to analyze the effect of agent types to total happiness
         X = df[['number-of-copies','number-of-analyzers','number-of-nevers','number-of-always
         y = df['analyzers-happiness'].values

         model = RandomForestRegressor(n_estimators=600, random_state=1)
         model.fit(X, y)
         # print and plot importance scores
         print(model.feature_importances_)

         types = df_grouped.columns.values[0:-1]
         ticks = [i for i in range(len(types))]
         plt.bar(ticks, model.feature_importances_)
         plt.xticks(ticks, types)
         plt.show()

[ 0.45059872  0.06036009  0.23534173  0.25369945]
```



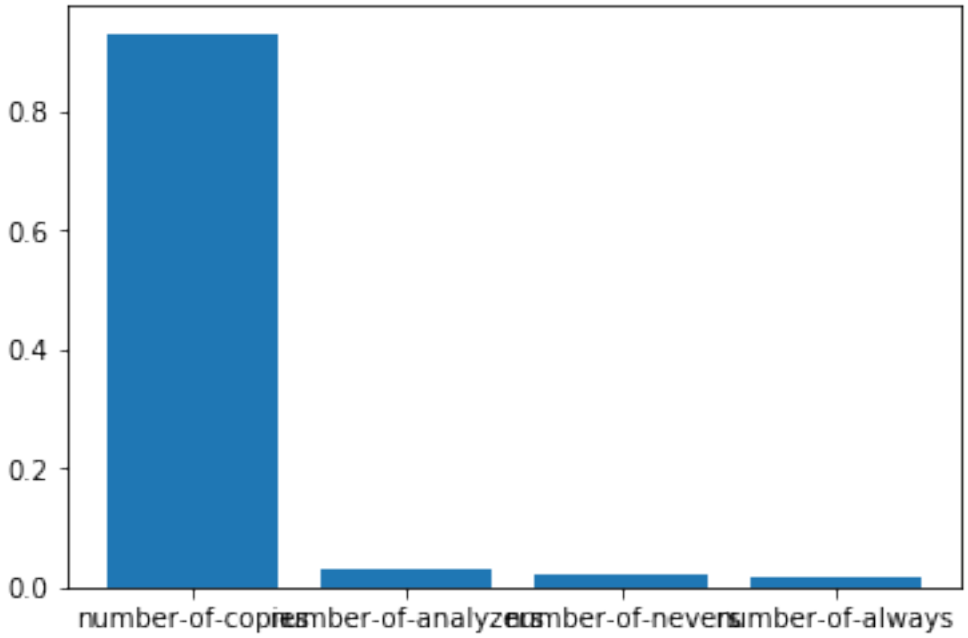

```
In [21]: from sklearn.ensemble import RandomForestRegressor
         #fit a randomforest to analyze the effect of agent types to total happiness
         X = df[['number-of-copies','number-of-analyzers','number-of-nevers','number-of-always
         y = df['copies-happiness'].values

         model = RandomForestRegressor(n_estimators=600, random_state=1)
         model.fit(X, y)
         # print and plot importance scores
         print(model.feature_importances_)

         types = df_grouped.columns.values[0:-1]
         ticks = [i for i in range(len(types))]
         plt.bar(ticks, model.feature_importances_)
         plt.xticks(ticks, types)
         plt.show()
```

```
[  9.97322180e-01   1.02867989e-03   1.00175327e-03   6.47387152e-04]
```



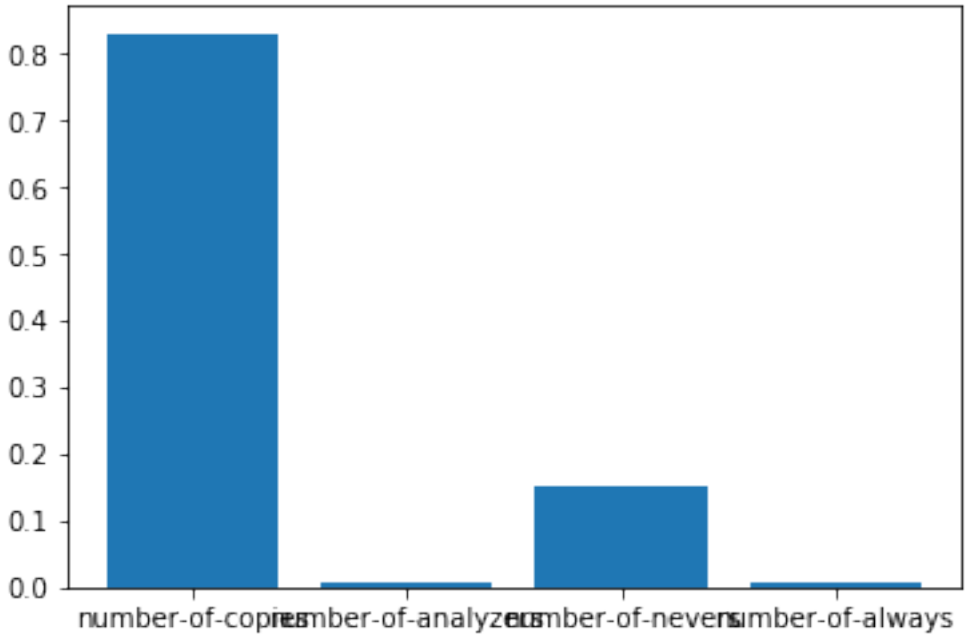
In [22]:
```
from sklearn.ensemble import RandomForestRegressor
# split into input and output
X = df[['number-of-copies','number-of-analyzers','number-of-nevers','number-of-always
y = df['randomms-happiness'].values

model = RandomForestRegressor(n_estimators=600, random_state=1)
model.fit(X, y)
# print and plot importance scores
print(model.feature_importances_)

types = df_grouped.columns.values[0:-1]
ticks = [i for i in range(len(types))]
plt.bar(ticks, model.feature_importances_)
plt.xticks(ticks, types)
plt.show()
```

```
[ 0.93157878  0.0304425   0.02123766  0.01674106]
```

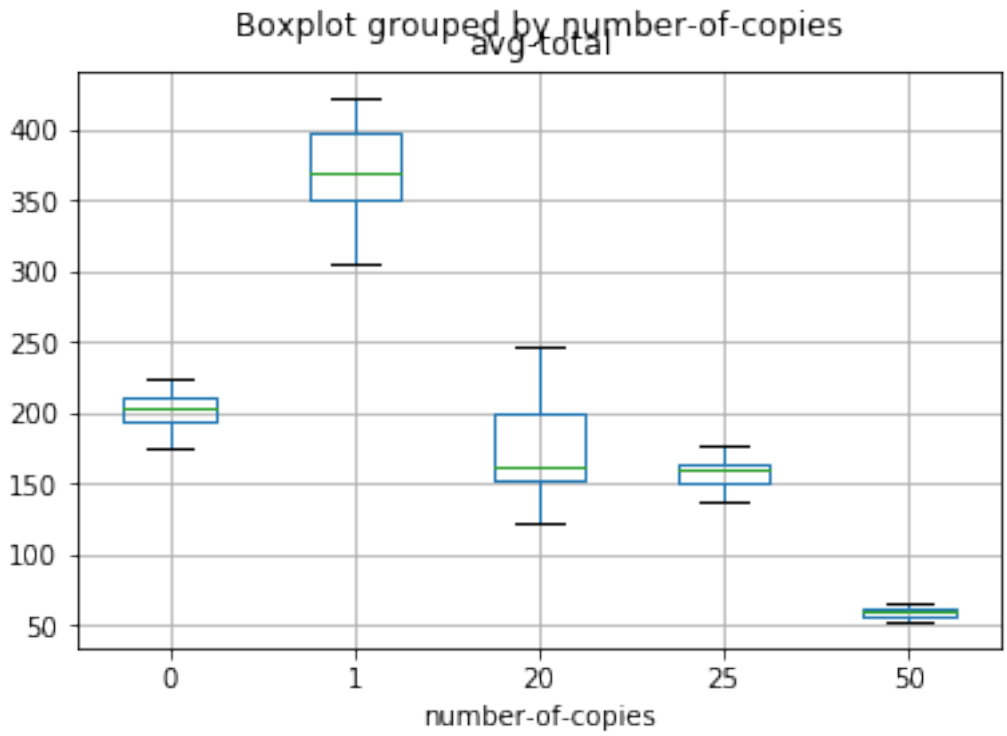


In [24]:
```
from sklearn.ensemble import RandomForestRegressor
#fit a randomforest to analyze the effect of agent types to total happiness
X = df[['number-of-copies','number-of-analyzers','number-of-nevers','number-of-always
y = df['always-happiness'].values

model = RandomForestRegressor(n_estimators=600, random_state=1)
model.fit(X, y)
# print and plot importance scores
print(model.feature_importances_)

types = df_grouped.columns.values[0:-1]
ticks = [i for i in range(len(types))]
plt.bar(ticks, model.feature_importances_)
plt.xticks(ticks, types)
plt.show()
```

```
[ 0.83106443  0.00726872  0.152901    0.00876586]
```



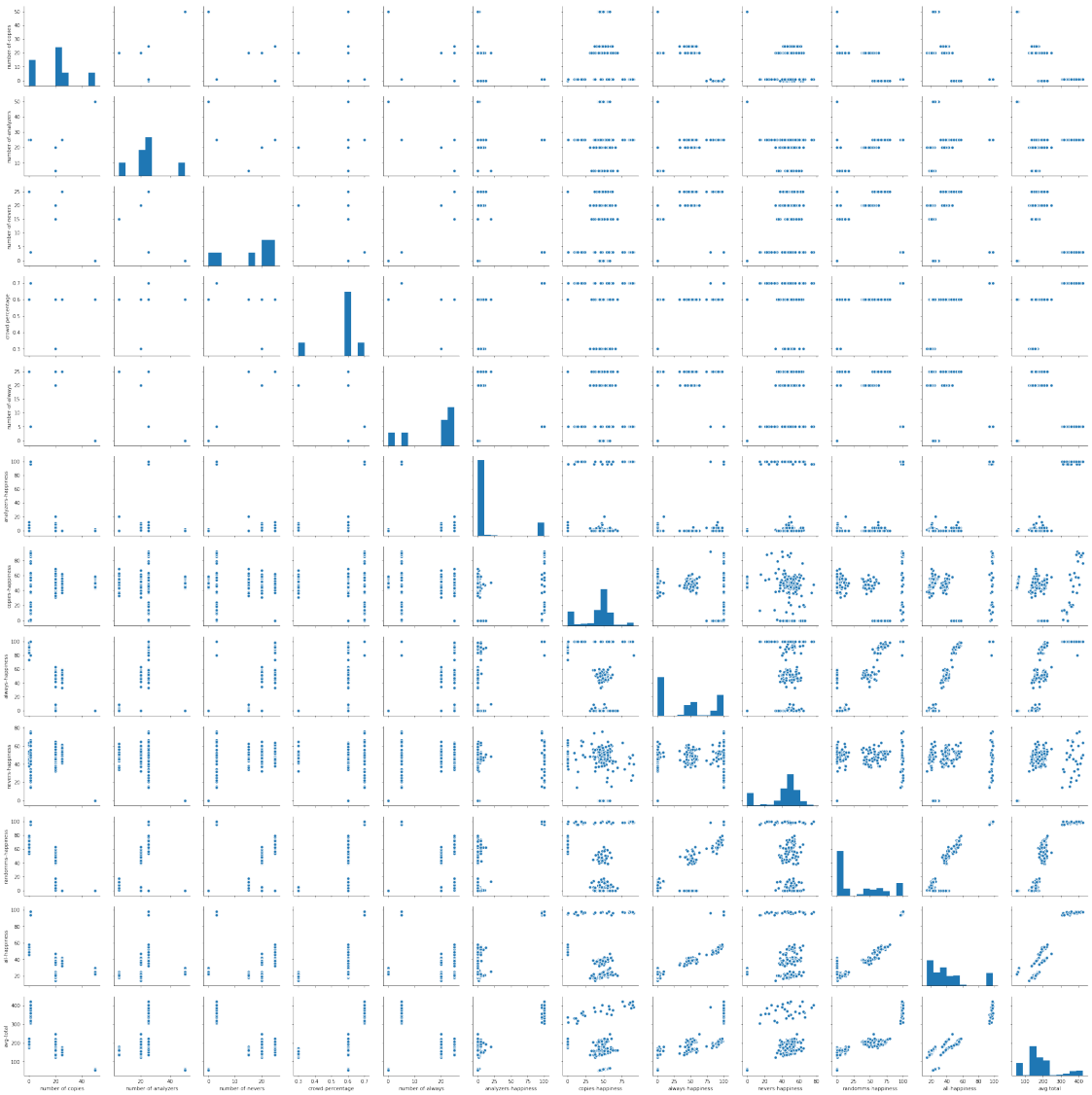

```
In [25]: df.boxplot(column='avg-total',by='number-of-copies')

C:\Users\Melodi\AppData\Local\Continuum\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:5
  return getattr(obj, method)(*args, **kwds)


Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x17eaa0d1da0>
```



- when number of copies : 0 avg-total happiness squeezed betwen 200-10 200+10
- when number of copies : 1 avg-total happiness between 400 and 3370
- when number of copies : 20 most of avg-total happiness greater than 160 and less than 200

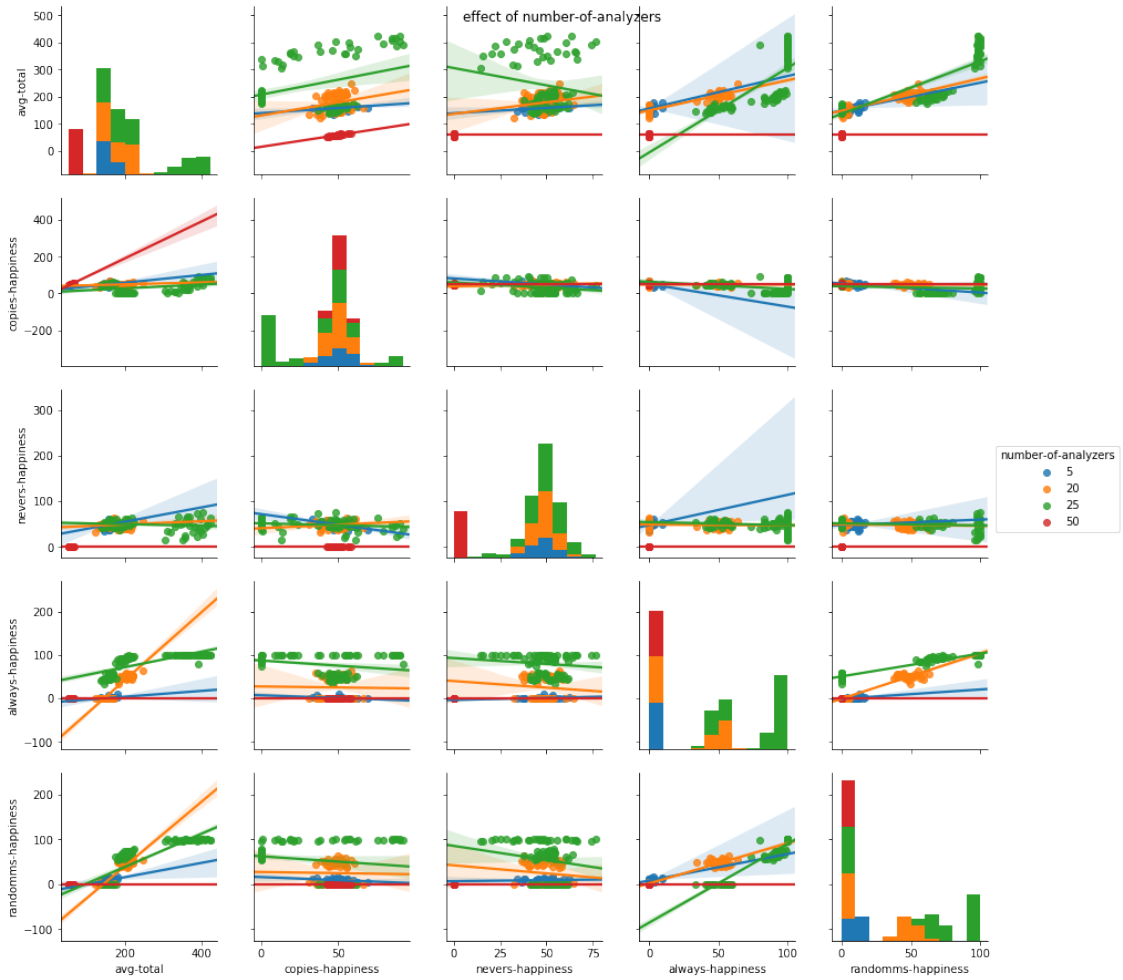
```
In [26]: sns.pairplot(df)
```

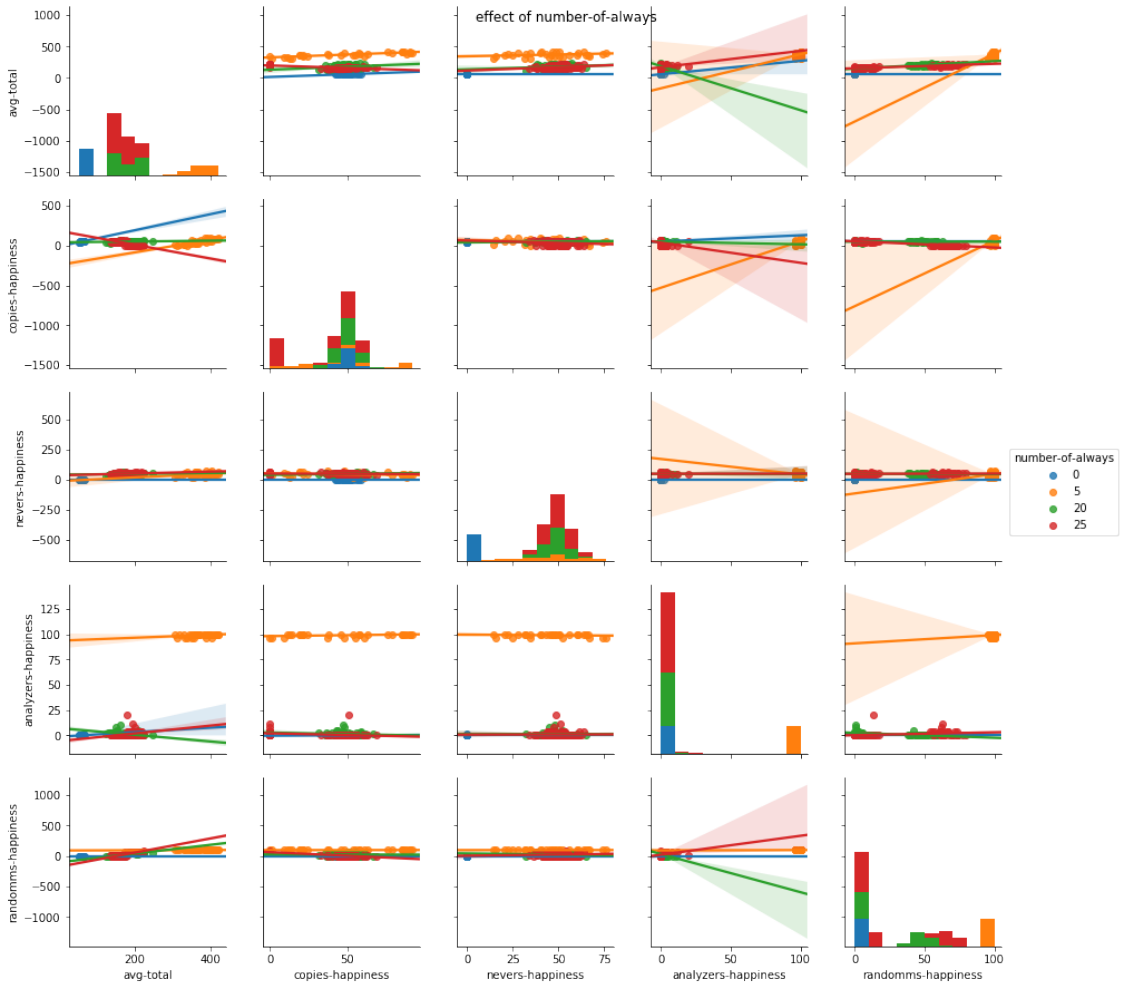
```
Out[26]: <seaborn.axisgrid.PairGrid at 0x17eaa481b38>
```



```
In [27]: sns.pairplot(df,vars= ['avg-total','copies-happiness','nevers-happiness','always-happi
         plt.suptitle('effect of number-of-analyzers')
         sns.pairplot(df,vars= ['avg-total','copies-happiness','nevers-happiness','analyzers-ha
         plt.suptitle('effect of number-of-always')
```

```
Out[27]: Text(0.5,0.98,'effect of number-of-always')
```


effect of number-of-analyzers
avg-total
copies-happiness
nevers-happiness
always-happiness
randomms-happiness
number-of-analyzers
5
20
25
50


effect of number-of-always
avg-total
copies-happiness
nevers-happiness
analyzers-happiness
randomms-happiness
number-of-always
0
5
20
25