IMPACT OF MUSIC STIMULI ON DECISION MAKING CODEBASE

April 27, 2020

1 IMPACT OF MUSIC STIMULI ON DECISION MAKING CODEBASE

1.1 Imports

In [1]:
```
from scipy import stats
import numpy as np
import pandas as pd
import random as rand
import researchpy as rp
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.formula.api import ols
import statsmodels.stats.multicomp
import hddm
```

```
/usr/local/lib/python3.6/dist-packages/IPython/parallel.py:13: ShimWarning: The `IPython.parallel` package ha
  "You should import from ipyparallel instead.", ShimWarning)
```

In [2]:
```
import warnings
warnings.filterwarnings("ignore")
```

1.2 Function for ANOVA Table

In [3]:
```
def anova_table(aov):
    aov['mean_sq'] = aov[:]['sum_sq']/aov[:]['df']
    aov['eta_sq'] = aov[:-1]['sum_sq']/sum(aov['sum_sq'])
    aov['omega_sq'] = (aov[:-1]['sum_sq']-(aov[:-1]['df']*aov['mean_sq'][-1]))/(sum(aov['sum_sq'])+aov['mea
    cols = ['sum_sq', 'mean_sq', 'df', 'F', 'PR(>F)', 'eta_sq', 'omega_sq']
    aov = aov[cols]
    return aov
```

1.3 Preparing Data from Response Sheet

In [4]: df = pd.read_csv("mmt_data.csv")

In [5]: df.head(10)

Out[5]:

	Id	Time-0	Mood-0	Score-0	Time-1	Mood-1	Score-1	Time-2	Mood-2	\
0	0	2.964578	U	4	3.527859	U	5	7.599937	U	
1	1	2.989358	P	4	6.896081	P	6	2.533781	P	
2	2	3.893769	U	5	8.155530	U	4	3.520698	U	
3	3	2.310259	N	3	12.299902	N	3	3.532279	N	
4	4	7.502648	U	3	5.708577	U	5	3.483882	U	
5	5	6.621638	P	6	5.308731	P	6	3.783082	P	
6	6	3.562238	U	3	6.023551	U	5	3.608170	U	
7	7	3.657962	N	2	5.286273	N	3	6.394205	N	
8	8	7.366485	U	2	5.527827	U	4	5.781769	U	
9	9	5.999600	P	4	5.628334	P	6	3.432613	P	

	Score-2	...	Score-36	Time-37	Mood-37	Score-37	Time-38	Mood-38	\
0	8	...	1	4.256945	N	4	9.689370	N	
1	6	...	5	6.215142	U	8	3.193628	U	
2	5	...	4	3.416484	P	8	8.364560	P	
3	4	...	4	10.218478	U	6	5.742848	U	
4	9	...	2	7.741418	N	5	15.042104	N	
5	6	...	3	6.932695	U	9	13.293433	U	
6	9	...	6	8.074441	P	6	7.216914	P	
7	5	...	1	7.167163	U	7	7.876160	U	
8	9	...	1	7.180183	N	6	5.355598	N	
9	8	...	5	7.490180	U	6	6.474940	U	

	Score-38	Time-39	Mood-39	Score-39
0	3	6.993097	N	3
1	9	7.430792	U	5
2	8	5.089959	P	6
3	9	5.764593	U	4
4	4	3.360752	N	3
5	6	5.256028	U	4
6	7	7.339705	P	6
7	7	8.695864	U	3
8	6	17.236559	N	4
9	7	2.903671	U	5

[10 rows x 121 columns]

In [6]: mood = np.array([[df['Mood-'+str(i)][x] for i in range(40)] for x in range(24)])

In [7]: score = np.array([[df['Score-'+str(i)][x] for i in range(40)] for x in range(24)])

In [8]: time = np.array([[df['Time-'+str(i)][x] for i in range(40)] for x in range(24)])

In [9]: ans = ["N","U","P","P","P","U","N","N","N","P","N","U","N","U","P","P","P","P","N","N","P","U"

1.4 Preparing Data for Section 5: ANOVA

In [10]:
```
## Per question analysis
st = [["U", "P", "U", "N"], ["P", "U", "N", "U"], ["U", "N", "U", "P"], ["N", "U", "P", "U"]]
p_val_int_o = []
p_val_int_c = []
p_val_ind_o = []
p_val_ind_t = []
p_val_ind_m = []
pdfr = []
model_1 = []
model_2 = []
mc_res1 = []
mc_res2 = []
res_1 = []
res_2 = []
nm = {"N":1, "U":5, "P":9};
for qs in range(40):
    a = np.asarray(score.T[qs], dtype=np.float64)
    #b = np.asarray([x for x in time.T[qs]], dtype=np.int)
    #b = [4 if v <= 4 else 7 if v <= 7 else 13 for v in b]
    b = [ans[qs]]*24 #[nm[ans[qs]]]*24
    c = st[int(qs/10)%4]*6
    d = pd.DataFrame([[b[i], c[i], a[i]] for i in range(24)], columns =['Stim', 'Mood', 'Score'])
    pdfr.append(d)
```

In [11]:
```
d = pd.concat(pdfr, axis=0)
```

1.5 Calculating Results for Section 5: ANOVA

In [13]:
```
model = ols('Score ~ C(Mood)*C(Stim)', d).fit()
model_1.append(model)
p_val_int_o.append(model.f_pvalue)
res1 = sm.stats.anova_lm(model, typ= 2)
res_1.append(res1)
p_val_int_c.append(res1['PR(>F)'][2])
model2 = ols('Score ~ C(Mood) + C(Stim)', d).fit()
model_2.append(model2)
p_val_ind_o.append(model2.f_pvalue)
res2 = sm.stats.anova_lm(model2, typ= 2)
res_2.append(res2)
p_val_ind_m.append(res2['PR(>F)'][0])
p_val_ind_t.append(res2['PR(>F)'][1])
mc1 = statsmodels.stats.multicomp.MultiComparison(d['Score'], d['Stim'])
mc1_results = mc1.tukeyhsd()
mc2 = statsmodels.stats.multicomp.MultiComparison(d['Score'], d['Mood'])
mc2_results = mc2.tukeyhsd()
mc_res1.append(mc1_results)
mc_res2.append(mc2_results)
```

```
In [92]: model = ols('Score ~ C(Mood)',d[d['Stim'] == 'N']).fit()
         print(f"Overall model F({model.df_model: .0f},{model.df_resid: .0f}) = {model.fvalue: .3f}, p = {model.f
         anova_table(sm.stats.anova_lm(model, typ= 2))

Overall model F( 2, 381) =  65.889, p =  0.0000
```

Out[92]:

	sum_sq	mean_sq	df	F	PR(>F)	eta_sq	omega_sq
C(Mood)	236.427083	118.213542	2.0	65.889161	2.658327e-25	0.256989	0.252596
Residual	683.562500	1.794127	381.0	NaN	NaN	NaN	NaN

```
In [93]: model = ols('Score ~ C(Mood)',d[d['Stim'] == 'P']).fit()
         print(f"Overall model F({model.df_model: .0f},{model.df_resid: .0f}) = {model.fvalue: .3f}, p = {model.f
         anova_table(sm.stats.anova_lm(model, typ= 2))

Overall model F( 2, 381) =  87.527, p =  0.0000
```

Out[93]:

	sum_sq	mean_sq	df	F	PR(>F)	eta_sq	omega_sq
C(Mood)	316.463542	158.231771	2.0	87.526716	5.266352e-32	0.314814	0.310659
Residual	688.776042	1.807811	381.0	NaN	NaN	NaN	NaN

```
In [94]: model = ols('Score ~ C(Mood)', d[d['Stim'] == 'U']).fit()
         print(f"Overall model F({model.df_model: .0f},{model.df_resid: .0f}) = {model.fvalue: .3f}, p = {model.f
         anova_table(sm.stats.anova_lm(model, typ= 2))

Overall model F( 2, 189) =  99.016, p =  0.0000
```

Out[94]:

	sum_sq	mean_sq	df	F	PR(>F)	eta_sq	omega_sq
C(Mood)	210.125000	105.062500	2.0	99.015894	3.833392e-30	0.511668	0.505195
Residual	200.541667	1.061067	189.0	NaN	NaN	NaN	NaN

```
In [20]: model_1[0].summary()
```

Out[20]: <class 'statsmodels.iolib.summary.Summary'>

```
"""
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Score   R-squared:                       0.682
Model:                            OLS   Adj. R-squared:                  0.679
Method:                 Least Squares   F-statistic:                     255.1
Date:                Thu, 23 Apr 2020   Prob (F-statistic):          1.25e-230
Time:                        03:34:07   Log-Likelihood:                -1599.2
No. Observations:                 960   AIC:                             3216.
Df Residuals:                     951   BIC:                             3260.
Df Model:                           8
Covariance Type:            nonrobust
==============================================================================
                                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept                      2.1354      0.131     16.269      0.000       1.878       2.393
C(Mood)[T.P]                   2.2083      0.186     11.897      0.000       1.844       2.573
C(Mood)[T.U]                   0.9479      0.161      5.897      0.000       0.632       1.263
C(Stim)[T.P]                   3.1563      0.186     17.003      0.000       2.792       3.521
C(Stim)[T.U]                   1.2813      0.227      5.636      0.000       0.835       1.727
C(Mood)[T.P]:C(Stim)[T.P]      0.0729      0.263      0.278      0.781      -0.442       0.588
C(Mood)[T.U]:C(Stim)[T.P]      1.0260      0.227      4.513      0.000       0.580       1.472
C(Mood)[T.P]:C(Stim)[T.U]      0.7500      0.322      2.333      0.020       0.119       1.381
C(Mood)[T.U]:C(Stim)[T.U]      0.5729      0.278      2.058      0.040       0.026       1.119
==============================================================================
Omnibus:                      227.175   Durbin-Watson:                   1.925
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               41.768
Skew:                           0.030   Prob(JB):                     8.52e-10
Kurtosis:                       1.980   Cond. No.                         15.8
==============================================================================

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

In [40]:
```
model = model_1[0]
print(f"Overall model F({model.df_model: .0f},{model.df_resid: .0f}) = {model.fvalue: .3f}, p = {model.f
```

```
Overall model F( 8, 951) =  255.073, p =  0.0000
```

In [22]:
```
model = model_2[0]
print(f"Overall model F({model.df_model: .0f},{model.df_resid: .0f}) = {model.fvalue: .3f}, p = {model.f
```

```
Overall model F( 4, 955) =  484.565, p =  0.0000
```

In [16]:
```
statsmodels.stats.multicomp.MultiComparison(d['Score'], d['Stim']).tukeyhsd().summary()
```

Out[16]: <class 'statsmodels.iolib.table.SimpleTable'>

In [17]:
```
statsmodels.stats.multicomp.MultiComparison(d['Score'], d['Mood']).tukeyhsd().summary()
```

Out[17]: <class 'statsmodels.iolib.table.SimpleTable'>

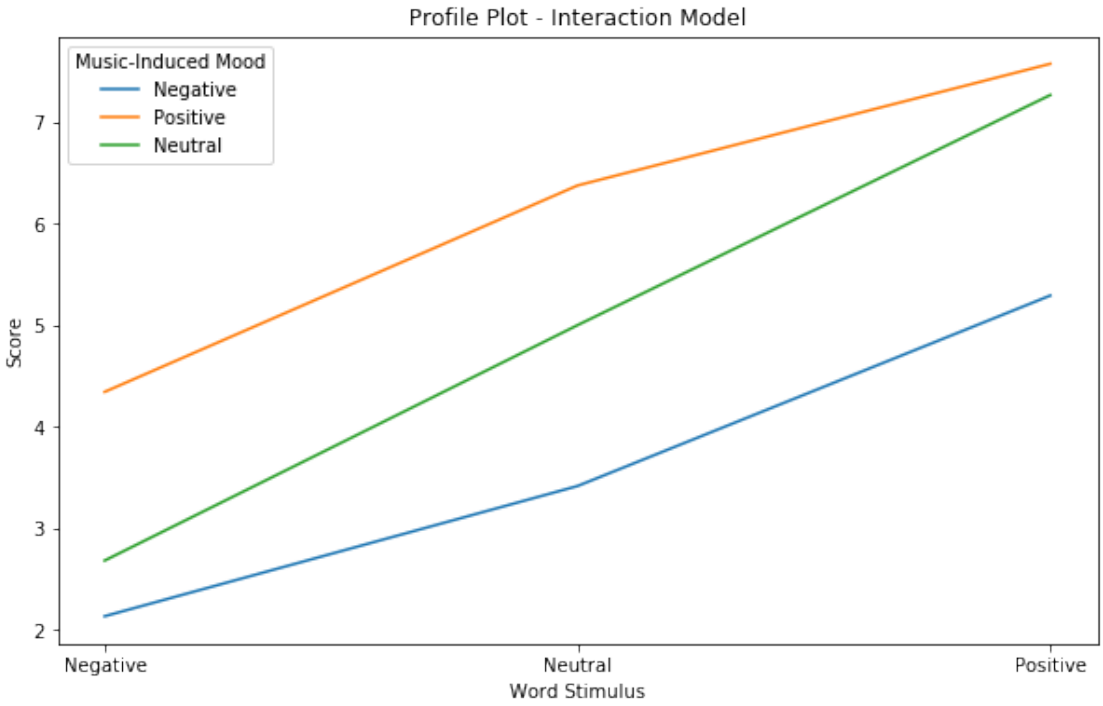
In [70]:
```
rp.summary_cont(d.groupby(['Stim', 'Mood']))['Score']
```

Out[70]:

Stim	Mood	N	Mean	SD	SE	95% Conf.	Interval
N	N	96	2.135417	1.110901	0.113381	1.913190	2.357643
	P	96	4.343750	1.375179	0.140354	4.068657	4.618843
	U	192	3.083333	1.422826	0.102684	2.882073	3.284593
P	N	96	5.291667	1.457588	0.148764	5.000088	5.583245
	P	96	7.572917	1.102579	0.112531	7.352355	7.793478
	U	192	7.265625	1.394554	0.100643	7.068364	7.462886
U	N	48	3.416667	0.918679	0.132600	3.156771	3.676563
	P	48	6.375000	1.002656	0.144721	6.091347	6.658653
	U	96	4.937500	1.093642	0.111619	4.718726	5.156274

In [19]:
```
y1 = [x for x in rp.summary_cont(d.groupby(['Mood', 'Stim']))['Score']['Mean'][0:3]]
y1[1], y1[2] = y1[2], y1[1]
y2 = [x for x in rp.summary_cont(d.groupby(['Mood', 'Stim']))['Score']['Mean'][3:6]]
y2[1], y2[2] = y2[2], y2[1]
y3 = [x for x in rp.summary_cont(d.groupby(['Mood', 'Stim']))['Score']['Mean'][6:9]]
y3[1], y3[2] = y3[2], y3[1]
x  = ['Negative', 'Neutral', 'Positive']
lb  = ['Negative', 'Neutral', 'Positive']
```

In [23]:
```
plt.figure(figsize=(10,6))
plt.plot(x, y1, label=lb[0])
plt.plot(x, y2, label=lb[2])
plt.plot(x, y3, label=lb[1])
plt.xlabel('Word Stimulus')
plt.ylabel('Score')
plt.title('Profile Plot - Interaction Model')
plt.legend(title='Music-Induced Mood')
plt.savefig('6.png', bbox_inches='tight')
```



In [40]: model_2[0].summary()

Out[40]: <class 'statsmodels.iolib.summary.Summary'>

"""

OLS Regression Results

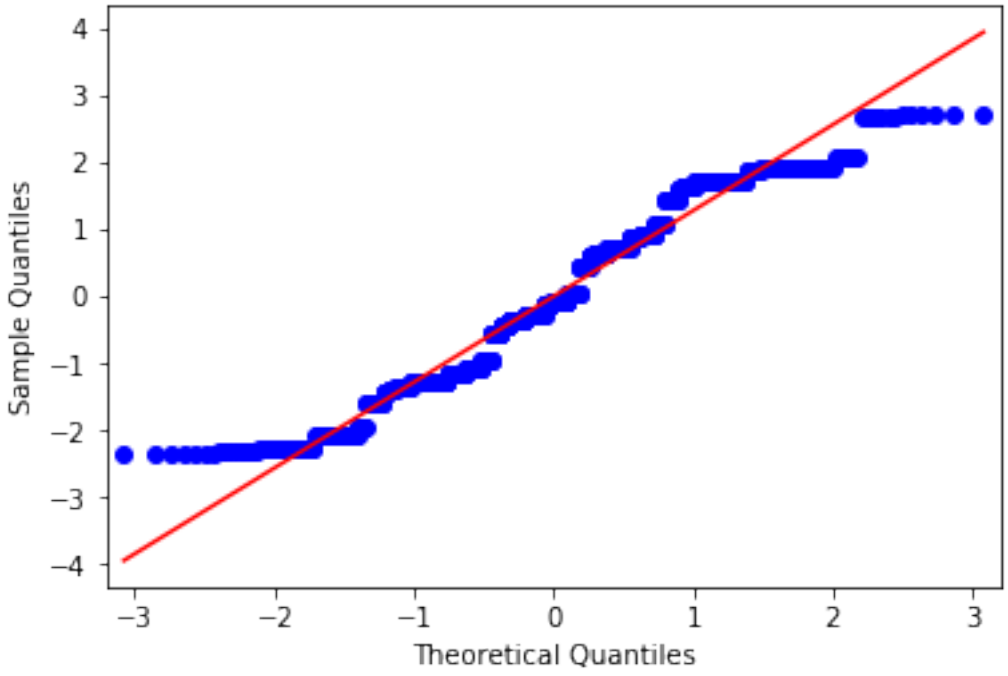
Dep. Variable:	Score	R-squared:	0.670
Model:	OLS	Adj. R-squared:	0.669
Method:	Least Squares	F-statistic:	484.6
Date:	Sat, 18 Apr 2020	Prob (F-statistic):	4.42e-228
Time:	09:25:00	Log-Likelihood:	-1617.2
No. Observations:	960	AIC:	3244.
Df Residuals:	955	BIC:	3269.
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.8281	0.099	18.469	0.000	1.634	2.022
C(Mood)[T.P]	2.3875	0.119	20.000	0.000	2.153	2.622
C(Mood)[T.U]	1.4729	0.103	14.247	0.000	1.270	1.676
C(Stim)[T.P]	3.6875	0.094	39.072	0.000	3.502	3.873
C(Stim)[T.U]	1.7552	0.116	15.185	0.000	1.528	1.982

```
Omnibus:                      169.136   Durbin-Watson:                   1.989
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               37.125
Skew:                           0.020   Prob(JB):                     8.68e-09
Kurtosis:                       2.037   Cond. No.                         4.88
==============================================================================

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

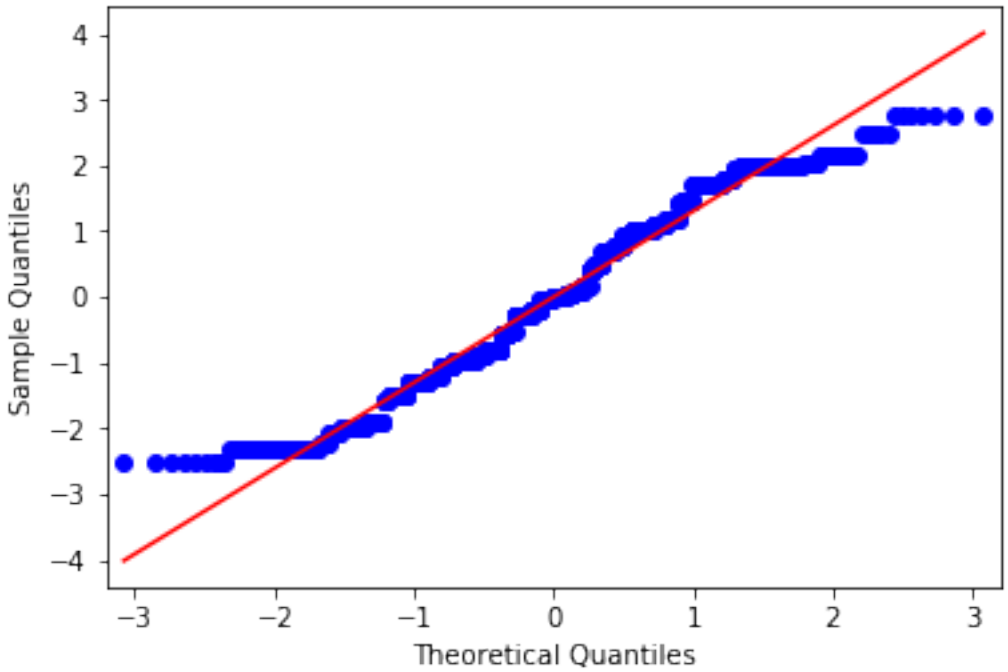
In [25]:
```
for i, j in enumerate(res_1):
    fig = sm.qqplot(model_1[i].resid, line='s')
    plt.savefig('7.png', bbox_inches='tight')
    print(anova_table(j))
```

```
                   sum_sq      mean_sq     df          F        PR(>F)  \
C(Mood)         702.722917   351.361458    2.0  212.441320  5.362712e-77
C(Stim)        2611.954167  1305.977083    2.0  789.624156  8.332225e-203
C(Mood):C(Stim)   60.292708    15.073177    4.0    9.113594  3.165397e-07
Residual        1572.880208     1.653922  951.0         NaN           NaN

                  eta_sq  omega_sq
C(Mood)         0.142026  0.141310
C(Stim)         0.527897  0.527052
C(Mood):C(Stim)  0.012186  0.010845
Residual              NaN       NaN
```



In [44]:
```
for i, j in enumerate(res_2):
    fig = sm.qqplot(model_2[i].resid, line='s')
    print(anova_table(j))
```

```
              sum_sq      mean_sq     df          F        PR(>F)  \
C(Mood)    702.722917   351.361458    2.0  205.459073   6.114036e-75
C(Stim)   2611.954167  1305.977083    2.0  763.671808  8.036252e-199
Residual  1633.172917     1.710129  955.0         NaN           NaN

             eta_sq  omega_sq
C(Mood)    0.142026  0.141286
C(Stim)    0.527897  0.527023
Residual        NaN       NaN
```



In [28]:
```
## Per Person analysis
for qs in range(24):
    a = np.asarray(score[qs], dtype=np.float64)
    b = np.asarray([x for x in time[qs]], dtype=np.int)
    b = [4 if v <= 4 else 7 if v <= 7 else 13 for v in b]
    c = [st[qs%4][int(i/10)] for i in range(40)]
    d = pd.DataFrame([[i, b[i], c[i], a[i]] for i in range(40)], columns =['Id', 'Time', 'Mood', 'Score'])
```

```
        model = ols('Score ~ C(Mood)*C(Time)', d).fit()
        if model.f_pvalue <= 0.05:
            print('Model1: ', model.f_pvalue)
        model2 = ols('Score ~ C(Mood)+C(Time)', d).fit()
        if model2.f_pvalue <= 0.05:
            print('Model2: ', model2.f_pvalue)
```

```
Model1:  0.0025446335253248614
Model2:  0.0009114689307813681
Model1:  0.011416604040595042
Model2:  0.016406207134407453
Model1:  0.054572930533523696
Model2:  0.04155753158211968
Model1:  0.02807264544645337
Model2:  0.013765201593679832
Model2:  0.05560146156246437
Model2:  0.0430038573757632
Model2:  0.042873774378730314
Model2:  0.05564950271266286
Model1:  0.011240705955709699
Model2:  0.0169049461964173
```

1.6 Preparing Data for Section 6: DDM

In [28]:
```
d1 = []
d2 = []
#d3 = []
st = [["U", "P", "U", "N"], ["P", "U", "N", "U"], ["U", "N", "U", "P"], ["N", "U", "P", "U"]]

for i in range(24):
    for qs in range(40):
        a = score[i][qs]
        if int(a) < 5:
            a = 0
        else:
            a = 1
        b = time[i][qs]
        c = st[i%4][int(qs/10)]
        d = ans[qs]
        if c == 'P':
            d1.append([i, b, d, a])
        elif c == 'U':
            if a and score[i][qs] != 5:
                d1.append([i, b, d, a])
            elif not a and score[i][qs] !=5:
                d2.append([i, b, d, a])
        elif c == 'N':
```

```
            d2.append([i, b, d, a])
```

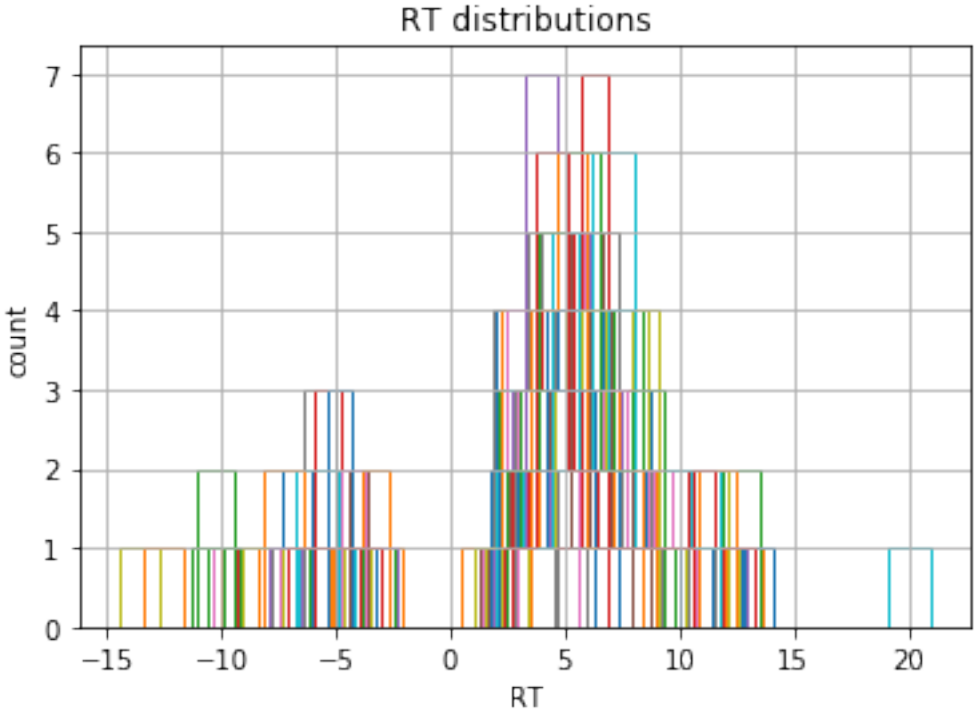
In [32]:
```
data1 = pd.DataFrame(d1, columns =['subj_idx', 'rt', 'stim', 'response']).to_csv('ddm_data1.csv')
data2 = pd.DataFrame(d2, columns =['subj_idx', 'rt', 'stim', 'response']).to_csv('ddm_data2.csv')
```

1.7 Calculating Results for Section 6: DDM

In [33]:
```
data1 = hddm.load_csv('ddm_data1.csv')
data2 = hddm.load_csv('ddm_data2.csv')
```

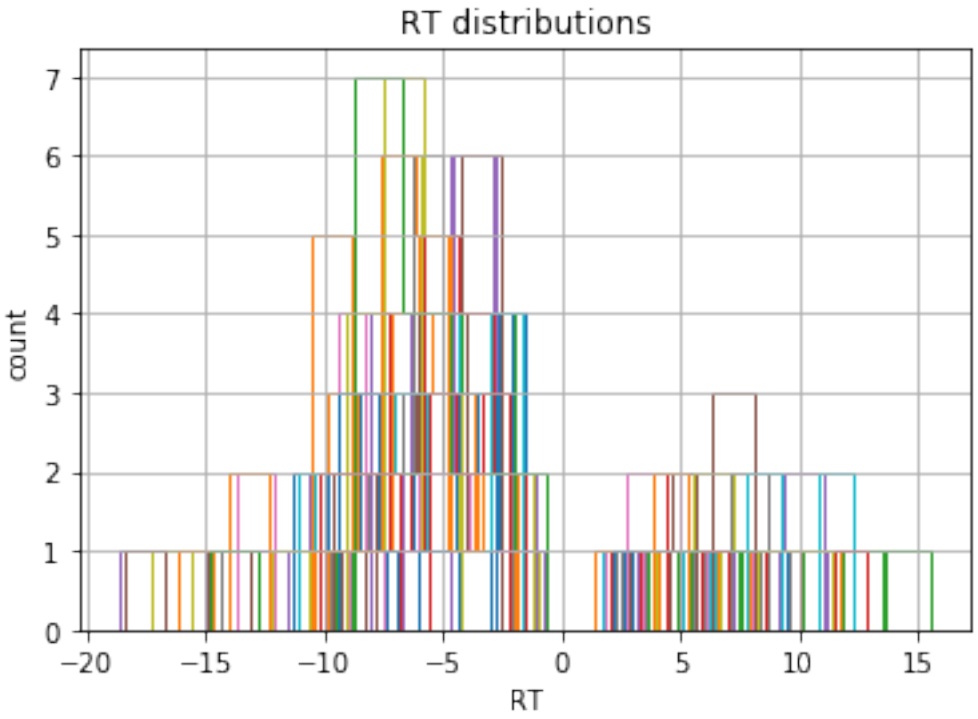
In [39]:
```
data1 = hddm.utils.flip_errors(data1)
```

```
fig = plt.figure()
ax = fig.add_subplot(111, xlabel='RT', ylabel='count', title='RT distributions')
for i, subj_data in data1.groupby('subj_idx'):
    subj_data.rt.hist(bins=15, histtype='step', ax=ax)
plt.savefig('2.png', bbox_inches='tight')
```



In [38]:
```
data2 = hddm.utils.flip_errors(data2)
```

```
fig = plt.figure()
ax = fig.add_subplot(111, xlabel='RT', ylabel='count', title='RT distributions')
for i, subj_data in data2.groupby('subj_idx'):
    subj_data.rt.hist(bins=15, histtype='step', ax=ax)
plt.savefig('3.png', bbox_inches='tight')
```



In [12]:
```
m1 = hddm.HDDM(data1, bias=True)
# find a good starting point which helps with the convergence.
m1.find_starting_values()
# start drawing 7000 samples and discarding 5000 as burn-in
m1.sample(4000, burn=100)
```

```
 [----------------100%----------------] 4001 of 4000 complete in 4414.7 sec
```

Out[12]: <pymc.MCMC.MCMC at 0x7f9398ffd198>

In [10]:
```
m2 = hddm.HDDM(data2, bias=True)
# find a good starting point which helps with the convergence.
m2.find_starting_values()
# start drawing 7000 samples and discarding 5000 as burn-in
m2.sample(4000, burn=100)
```

```
 [----------------100%----------------] 4000 of 4000 complete in 2412.0 sec
```

Out[10]: <pymc.MCMC.MCMC at 0x7f9398b0fef0>

In [ ]:
```
m2.gen_stats()
```

In [ ]:
```
m2.gen_stats().to_csv('ddm_data_bias.csv')
```

In [ ]:
```
data1 = pd.DataFrame(d1, columns =['Id', 'Time', 'Mood', 'Score'])#.to_csv('mmt_ddm_p_data.csv')
data2 = pd.DataFrame(d2, columns =['Id', 'Time', 'Mood', 'Score'])#.to_csv('mmt_ddm_u_data.csv')
data3 = pd.DataFrame(d3, columns =['Id', 'Time', 'Mood', 'Score'])#.to_csv('mmt_ddm_n_data.csv')
```

In [ ]:
```
mp = hddm.HDDM(data1, bias=True)
mp.find_starting_values()
mp.sample(2000, burn=20)

mu = hddm.HDDM(data2, bias=True)
mu.find_starting_values()
mu.sample(2000, burn=20)

mn = hddm.HDDM(data3, bias=True)
mn.find_starting_values()
mn.sample(2000, burn=20)
```

In [ ]:
```
a_valp = np.asarray([i for i in mp.gen_stats()[mp.gen_stats().index.isin(['a_subj.'+str(i) for i in range(24
v_valp = np.asarray([i for i in mp.gen_stats()[mp.gen_stats().index.isin(['v_subj.'+str(i) for i in range(24
t_valp = np.asarray([i for i in mp.gen_stats()[mp.gen_stats().index.isin(['t_subj.'+str(i) for i in range(24)
z_valp = np.asarray([i for i in mp.gen_stats()[mp.gen_stats().index.isin(['z_subj.'+str(i) for i in range(24)

a_valn = np.asarray([i for i in mn.gen_stats()[mn.gen_stats().index.isin(['a_subj.'+str(i) for i in range(24
v_valn = np.asarray([i for i in mn.gen_stats()[mn.gen_stats().index.isin(['v_subj.'+str(i) for i in range(24
t_valn = np.asarray([i for i in mn.gen_stats()[mn.gen_stats().index.isin(['t_subj.'+str(i) for i in range(24)
z_valn = np.asarray([i for i in mn.gen_stats()[mn.gen_stats().index.isin(['z_subj.'+str(i) for i in range(24)
```

In [ ]:
```
mp.gen_stats()[mp1.gen_stats().index.isin(['a', 'v', 't', 'z'])]
```

In [ ]:
```
mu.gen_stats()[mp1.gen_stats().index.isin(['a', 'v', 't', 'z'])]
```

In [ ]:
```
mn.gen_stats()[mp1.gen_stats().index.isin(['a', 'v', 't', 'z'])]
```

1.8 Calculating Table Data for Section 4: Initial Results

In [42]:
```
ans = ["N","U","P","P","P","U","N","N","N","P","N","U","N","U","P","P","P","P","N","N","P","U
x1 = np.asarray([[0],[0],[0]])
x2 = np.asarray([[0],[0],[0]])
x3 = np.asarray([[0],[0],[0]])
x4 = [[0,0,0], [0,0,0], [0,0,0]]
nm = {'N':0, 'U':1, 'P':2}

for ps in range(24):
    for qs in range(40):
        a = 0
        if df['Score-'+str(qs)][ps] > 5:
            a = 1
        else:
            a = 0
```

```
            if ans[qs] == "N":
                x1[nm[df['Mood-'+str(qs)][ps]]] += a
                x4[0][nm[df['Mood-'+str(qs)][ps]]] += 1
            elif ans[qs] == "U":
                x2[nm[df['Mood-'+str(qs)][ps]]] += a
                x4[1][nm[df['Mood-'+str(qs)][ps]]] += 1
            else:
                x3[nm[df['Mood-'+str(qs)][ps]]] += a
                x4[2][nm[df['Mood-'+str(qs)][ps]]] += 1
```

In [43]:
```
for i in range(3):
    print(x1[i]/x4[0][i])

for i in range(3):
    print(x2[i]/x4[1][i])

for i in range(3):
    print(x3[i]/x4[2][i])
```

```
[0.]
[0.]
[0.21875]
[0.]
[0.30208333]
[0.79166667]
[0.45833333]
[0.86979167]
[1.]
```

In [44]:
```
x11 = [[],[],[]]
x12 = [[],[],[]]
x13 = [[],[],[]]
x4 = [[0,0,0], [0,0,0], [0,0,0]]
nm = {'N':0, 'U':1, 'P':2}

for ps in range(24):
    for qs in range(40):

        if ans[qs] == "N":
            x11[nm[df['Mood-'+str(qs)][ps]]].append(df['Time-'+str(qs)][ps])
            #x4[0][nm[df['Mood-'+str(qs)][ps]]] += 1
        elif ans[qs] == "U":
            x12[nm[df['Mood-'+str(qs)][ps]]].append(df['Time-'+str(qs)][ps])
            #x4[1][nm[df['Mood-'+str(qs)][ps]]] += 1
        else:
            x13[nm[df['Mood-'+str(qs)][ps]]].append(df['Time-'+str(qs)][ps])
            #x4[2][nm[df['Mood-'+str(qs)][ps]]] += 1
```

```
In [47]: print([np.mean(np.array(x)) for x in x11])
         print([np.mean(np.array(x)) for x in x12])
         print([np.mean(np.array(x)) for x in x13])
```

```
[4.899853469264678, 6.683169223938701, 6.207500937388654]
[7.369645436448366, 6.349489046930901, 6.253775276923274]
[6.6896841351590055, 6.591271390190433, 5.114316703843104]
```

```
In [46]: print([np.std(np.array(x)) for x in x11])
         print([np.std(np.array(x)) for x in x12])
         print([np.std(np.array(x)) for x in x13])
```

```
[2.151572980610577, 3.1640598179128934, 2.5183553982292968]
[3.7504425877722647, 2.6602599088006924, 2.242990716503152]
[2.997048635806363, 2.804592641028399, 2.241320741543202]
```

```
In [48]: ans = ["N","U","P","P","P","U","N","N","N","P","N","U","N","U","P","P","P","P","N","N","P","U
         x1 = np.asarray([[0],[0],[0]])
         x2 = np.asarray([[0],[0],[0]])
         x3 = np.asarray([[0],[0],[0]])
         x4 = [[0,0,0], [0,0,0], [0,0,0]]
         nm = {'N':0, 'U':1, 'P':2}

         for ps in range(24):
             for qs in range(40):
                 a = 0
                 if df['Score-'+str(qs)][ps] > 5 and ans[qs] == "P":
                     a = 1
                 elif df['Score-'+str(qs)][ps] < 5 and ans[qs] == "N":
                     a = 1
                 elif df['Score-'+str(qs)][ps] == 5 and ans[qs] == "U":
                     a = 1


                 if ans[qs] == "N":
                     x1[nm[df['Mood-'+str(qs)][ps]]] += a
                     x4[0][nm[df['Mood-'+str(qs)][ps]]] += 1
                 elif ans[qs] == "U":
                     x2[nm[df['Mood-'+str(qs)][ps]]] += a
                     x4[1][nm[df['Mood-'+str(qs)][ps]]] += 1
                 else:
                     x3[nm[df['Mood-'+str(qs)][ps]]] += a
                     x4[2][nm[df['Mood-'+str(qs)][ps]]] += 1
```

```
In [50]: for i in range(3):
             print(x1[i]/x4[0][i])
```

```
for i in range(3):
    print(x2[i]/x4[1][i])

for i in range(3):
    print(x3[i]/x4[2][i])
```

```
[1.]
[0.765625]
[0.5625]
[0.14583333]
[0.34375]
[0.20833333]
[0.45833333]
[0.86979167]
[1.]
```