EDA for features extracted by OpenFace

In [2]:

```
#import libraries
from graphutil import PlotGraphs
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
import numpy as np
import sweetviz as sv
import time
import plotly.io as pio
pio.templates.default = "none"
import warnings
import matplotlib.colors as mcolors
```

In [3]:

```
#read the data file
path = "C:\\Work\\606Capstone\\Video_chunks\\Final.csv"
df = pd.read_csv(path)
df
```

Out[3]:

	VideoName	mean_AU01	mean_AU02	mean_AU04	mean_AU05	mean_AU06	mean_AU07	mean_AU09
0	trial_lie_001_000	0.559135	0.513050	0.254950	0.413367	0.131757	0.119048	0.210417
1	trial_lie_001_001	0.546610	0.482385	0.292549	0.393281	0.123315	0.076923	0.182163
2	trial_lie_001_002	0.560753	0.483092	0.291333	0.393530	0.129316	0.094017	0.184086
3	trial_lie_001_003	0.532220	0.481838	0.321986	0.388174	0.154953	0.126984	0.256083
4	trial_lie_002_000	0.436142	0.412403	0.371331	0.332483	0.242381	0.111111	0.288108
...	...	...	...	...	...	...	...	...
732	trial_truth_060_000	0.496345	0.485515	0.535118	0.406868	0.251973	0.246032	0.375586
733	trial_truth_060_001	0.422170	0.468637	0.446201	0.508495	0.177772	0.228070	0.292094
734	trial_truth_060_002	0.515533	0.491938	0.438462	0.551434	0.202776	0.139344	0.341805
735	trial_truth_060_003	0.469542	0.475879	0.492550	0.457961	0.187666	0.245763	0.367703
736	trial_truth_060_004	0.532738	0.520756	0.499290	0.475737	0.249443	0.198198	0.384727

737 rows × 29 columns

In [4]:

```
#get the shape of the datafrme
df.shape
```

Out[4]:

```
(737, 29)
```

In [5]:

```
#Find number of non-null values and data type of each column in the dataframe
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 737 entries, 0 to 736
Data columns (total 29 columns):
```

```
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   VideoName       737 non-null    object 
 1   mean_AU01       737 non-null    float64
 2   mean_AU02       737 non-null    float64
 3   mean_AU04       737 non-null    float64
 4   mean_AU05       737 non-null    float64
 5   mean_AU06       737 non-null    float64
 6   mean_AU07       737 non-null    float64
 7   mean_AU09       737 non-null    float64
 8   mean_AU10       737 non-null    float64
 9   mean_AU11       737 non-null    float64
 10  mean_AU12       737 non-null    float64
 11  mean_AU14       737 non-null    float64
 12  mean_AU15       737 non-null    float64
 13  mean_AU17       737 non-null    float64
 14  mean_AU20       737 non-null    float64
 15  mean_AU23       737 non-null    float64
 16  mean_AU24       737 non-null    float64
 17  mean_AU25       737 non-null    float64
 18  mean_AU26       737 non-null    float64
 19  mean_AU28       737 non-null    float64
 20  mean_AU43       737 non-null    float64
 21  mean_anger      737 non-null    float64
 22  mean_disgust    737 non-null    float64
 23  mean_fear       737 non-null    float64
 24  mean_happiness  737 non-null    float64
 25  mean_sadness    737 non-null    float64
 26  mean_surprise   737 non-null    float64
 27  mean_neutral    737 non-null    float64
 28  Label           737 non-null    object 
dtypes: float64(27), object(2)
memory usage: 167.1+ KB
```

In [6]:

```
#get the summary of the dataset
df.describe()
```

Out[6]:

	mean_AU01	mean_AU02	mean_AU04	mean_AU05	mean_AU06	mean_AU07	mean_AU09	mean_AU10	mear
count	737.000000	737.000000	737.000000	737.000000	737.000000	737.000000	737.000000	737.000000	737.
mean	0.416655	0.467410	0.404436	0.418905	0.184911	0.205069	0.285994	0.224698	0.
std	0.074658	0.036116	0.083717	0.072502	0.069933	0.146853	0.084336	0.154125	0.
min	0.231443	0.359584	0.207955	0.250289	0.079295	0.000000	0.114577	0.007370	0.
25%	0.359575	0.445966	0.346847	0.363071	0.128320	0.092308	0.221705	0.098964	0.
50%	0.409721	0.470585	0.388437	0.410689	0.174261	0.170940	0.299751	0.196964	0.
75%	0.463986	0.492236	0.454445	0.464025	0.222434	0.290598	0.342762	0.325472	0.
max	0.633079	0.574856	0.702236	0.672131	0.499566	1.000000	0.587998	0.778519	0.

8 rows × 27 columns

Data Clean up and Sanity Checks

From the above analysis it is clear that all the columns have the correct data type and hence there is no need of type casting. We also observe that there are no null values in the dataset.

Let us check whether there are any columns that have same value for all the rows

In [7]:

```python
counter = 0
for this_column in df.columns:
    if (df[this_column].nunique()==1):
        print(this_column)
        counter = 1
if(counter != 1):
    print('There are no such columns having the same value for all the rows.')
```

```
There are no such columns having the same value for all the rows.
```

We have a few object type columns. Let us check if we can convert them to any other datatype so that our processing can be faster.

In [8]:

```python
#find number of unique values in various object type columns
count = 10
for col,col_type in df.dtypes.iteritems():
    if(col_type=='object'):
        print('\n',col,'has',df[col].nunique(),'unique entries; and the top unique values
        print(df[col].value_counts().head(count))
```

```
 VideoName has 737 unique entries; and the top unique values are
trial_lie_001_000      1
trial_truth_011_007    1
trial_truth_010_016    1
trial_truth_010_017    1
trial_truth_011_000    1
trial_truth_011_001    1
trial_truth_011_002    1
trial_truth_011_003    1
trial_truth_011_004    1
trial_truth_011_005    1
Name: VideoName, dtype: int64

 Label has 2 unique entries; and the top unique values are
lie      390
truth    347
Name: Label, dtype: int64
```

From the above report it is clear that we can convert the 'Label' to categorical for the ease of processing.

In [9]:

```python
#convert columns to categorical
df['Label'] = df['Label'].astype('category')
```

Let us now visualize the various statistics (statistical report) for all the variables

In [10]:

```python
#analyzing the dataset
advert_report = sv.analyze(df)
#display the report
advert_report.show_html('Advertising.html')
```

```
                                                  |
| [  0%]   00:00 ->…
Report Advertising.html was generated! NOTEBOOK/COLAB USERS: the web browser MAY not pop u
p, regardless, the report IS saved in your notebook/colab files.
```

Let us now find the correlation between the various parameters

In [11]:

```python
corr_df = df.corr()
corr_df
```

Out[11]:

	mean_AU01	mean_AU02	mean_AU04	mean_AU05	mean_AU06	mean_AU07	mean_AU09	mean_A

	mean_AU01	mean_AU02	mean_AU04	mean_AU05	mean_AU06	mean_AU07	mean_AU09	mean_A
mean_AU01	1.000000	0.623549	-0.134760	0.003546	-0.224585	-0.321679	-0.141198	-0.357
mean_AU02	0.623549	1.000000	-0.045739	0.012628	-0.224805	-0.184971	-0.285123	-0.289
mean_AU04	-0.134760	-0.045739	1.000000	-0.136145	-0.053624	0.129152	0.253420	0.009
mean_AU05	0.003546	0.012628	-0.136145	1.000000	0.128962	-0.089877	0.164110	0.246
mean_AU06	-0.224585	-0.224805	-0.053624	0.128962	1.000000	0.437918	0.622863	0.577
mean_AU07	-0.321679	-0.184971	0.129152	-0.089877	0.437918	1.000000	0.389165	0.609
mean_AU09	-0.141198	-0.285123	0.253420	0.164110	0.622863	0.389165	1.000000	0.504
mean_AU10	-0.357896	-0.289638	0.009066	0.246726	0.577232	0.609698	0.504748	1.000
mean_AU11	-0.106926	-0.163655	-0.005074	-0.063546	0.532240	0.470624	0.468098	0.424
mean_AU12	-0.047713	-0.081869	-0.142851	-0.185849	0.664737	0.332068	0.321225	0.259
mean_AU14	-0.243296	-0.141485	-0.270187	-0.518476	0.247625	0.323350	-0.041016	0.163
mean_AU15	-0.083336	-0.107643	0.082224	0.640656	0.345004	0.118133	0.501235	0.447
mean_AU17	-0.212892	-0.167989	0.045074	0.669024	0.352746	0.039647	0.493189	0.453
mean_AU20	0.129633	0.118576	0.067180	-0.308877	0.220133	0.147487	0.049445	0.011
mean_AU23	0.079253	0.099695	0.108898	-0.558911	-0.377478	-0.135659	-0.342669	-0.407
mean_AU24	-0.080191	-0.108724	-0.031451	-0.320092	-0.134166	-0.050337	-0.091620	-0.230
mean_AU25	0.240608	0.229006	0.111315	-0.209307	-0.034363	0.015298	-0.151745	-0.128
mean_AU26	0.414682	0.319830	0.046330	-0.090300	0.069555	0.053461	-0.040545	0.068
mean_AU28	-0.074213	-0.056023	0.150874	-0.035206	-0.014605	0.327295	0.133825	0.202
mean_AU43	0.073666	-0.046582	0.127624	-0.301851	0.318285	0.394951	0.525835	0.314
mean_anger	-0.289305	-0.381379	0.327600	-0.123930	0.254830	0.339028	0.413211	0.416
mean_disgust	0.068094	0.096655	-0.073442	-0.159170	0.095682	0.096717	-0.169386	0.093
mean_fear	0.214129	0.154068	-0.067734	0.321768	-0.037650	-0.207891	0.059892	-0.159
mean_happiness	0.058526	-0.042154	-0.077183	-0.224871	0.133678	0.025608	0.111418	-0.095
mean_sadness	0.133188	0.116979	-0.223406	-0.274227	-0.312189	-0.344980	-0.414844	-0.502
mean_surprise	0.368391	0.351784	-0.091121	0.372721	-0.047546	-0.123783	-0.012691	0.003
mean_neutral	-0.313835	-0.129473	0.062204	0.106257	-0.017802	0.114095	-0.026458	0.115

27 rows × 27 columns

Let us now find the top 10 correlated parameters

In [12]:

```
# convert dataframe to a series, exclude values equal to 1, and sort by descending order
max_values = corr_df.stack()[corr_df.stack()!=1].sort_values(ascending=False)

# create an empty set to keep track of row/column pairs that have already been included
included_pairs = set()

# iterate through the top 10 values and retrieve corresponding row and column names, while
i = 0
for idx in max_values.index:
    row_name, col_name = idx
```

```
    # check if row and column names are the same, skip if they are
    if row_name == col_name:
        continue
    # check if row/column pair has already been included, skip if it has
    if (row_name, col_name) in included_pairs or (col_name, row_name) in included_pairs:
        continue
    # add row/column pair to included_pairs set
    included_pairs.add((row_name, col_name))
    # retrieve value and print output
    value = corr_df.loc[row_name, col_name]
    print(f"Parameter1: {row_name}, Parameter2: {col_name}, Value: {value}")
    i += 1
    if i == 10:
        break
```

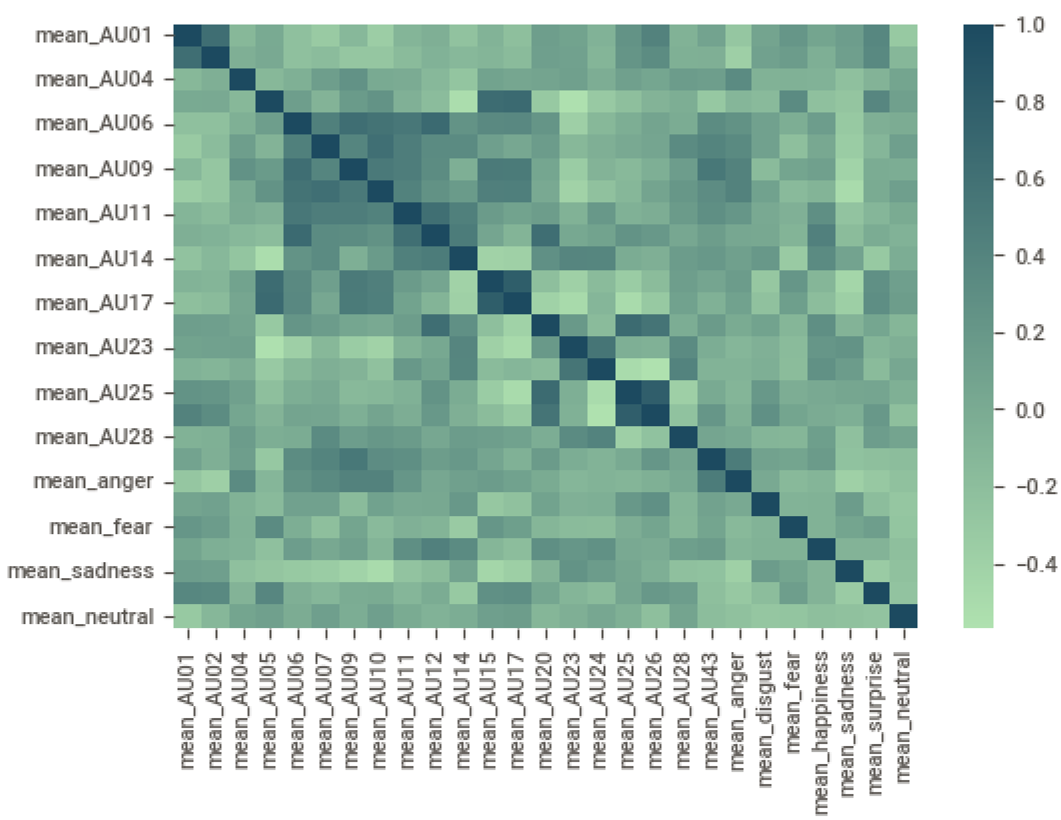
```
Parameter1: mean_AU17, Parameter2: mean_AU15, Value: 0.7969540520715238
Parameter1: mean_AU25, Parameter2: mean_AU26, Value: 0.7899169537982647
Parameter1: mean_AU17, Parameter2: mean_AU05, Value: 0.6690244086726005
Parameter1: mean_AU06, Parameter2: mean_AU12, Value: 0.6647365370757331
Parameter1: mean_AU25, Parameter2: mean_AU20, Value: 0.6558944138192253
Parameter1: mean_AU05, Parameter2: mean_AU15, Value: 0.6406557706297027
Parameter1: mean_AU01, Parameter2: mean_AU02, Value: 0.6235489281863267
Parameter1: mean_AU20, Parameter2: mean_AU12, Value: 0.623304329883933
Parameter1: mean_AU09, Parameter2: mean_AU06, Value: 0.6228628120988566
Parameter1: mean_AU11, Parameter2: mean_AU12, Value: 0.6190263141205835
```

In [13]:

```
colors = ["#AFE1AF", "#1c4a60"]
cmap = mcolors.LinearSegmentedColormap.from_list("", colors)
sns.heatmap(corr_df, cmap=cmap)
```

Out[13]:

```
<AxesSubplot:>
```



Visualizations

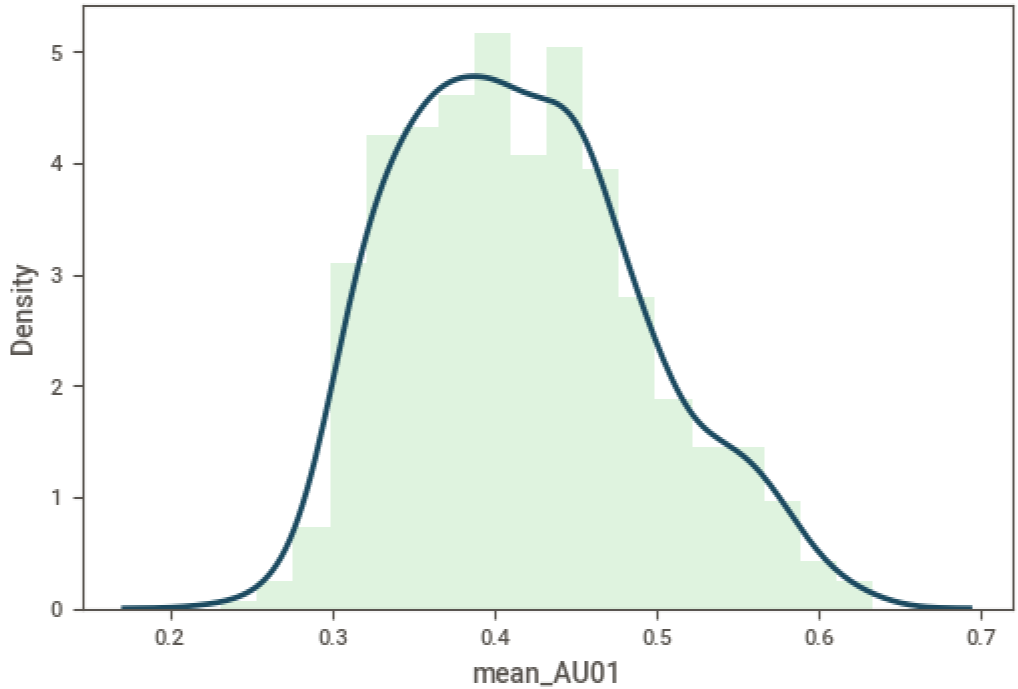
Let us find the distribution of each parameter

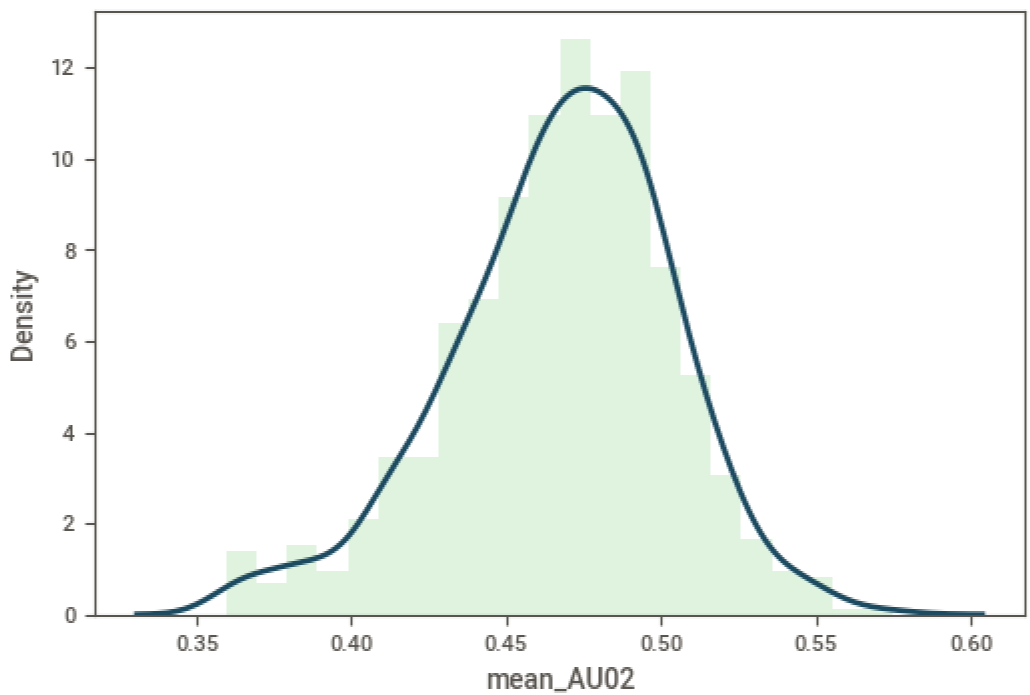
In [14]:

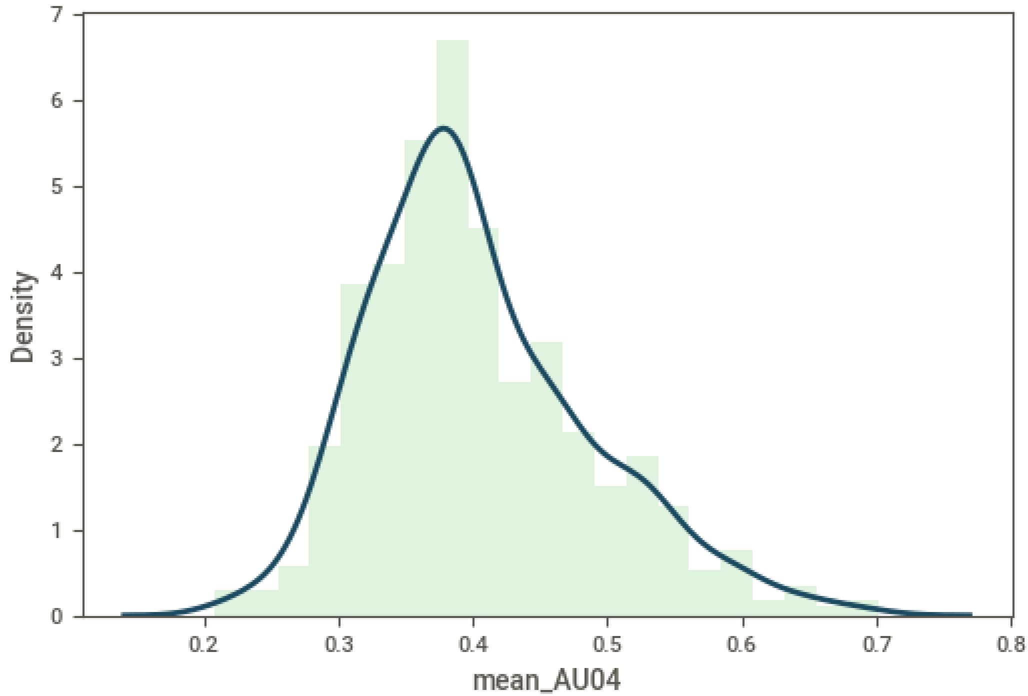
```
warnings.filterwarnings(action='ignore', category=FutureWarning)
```

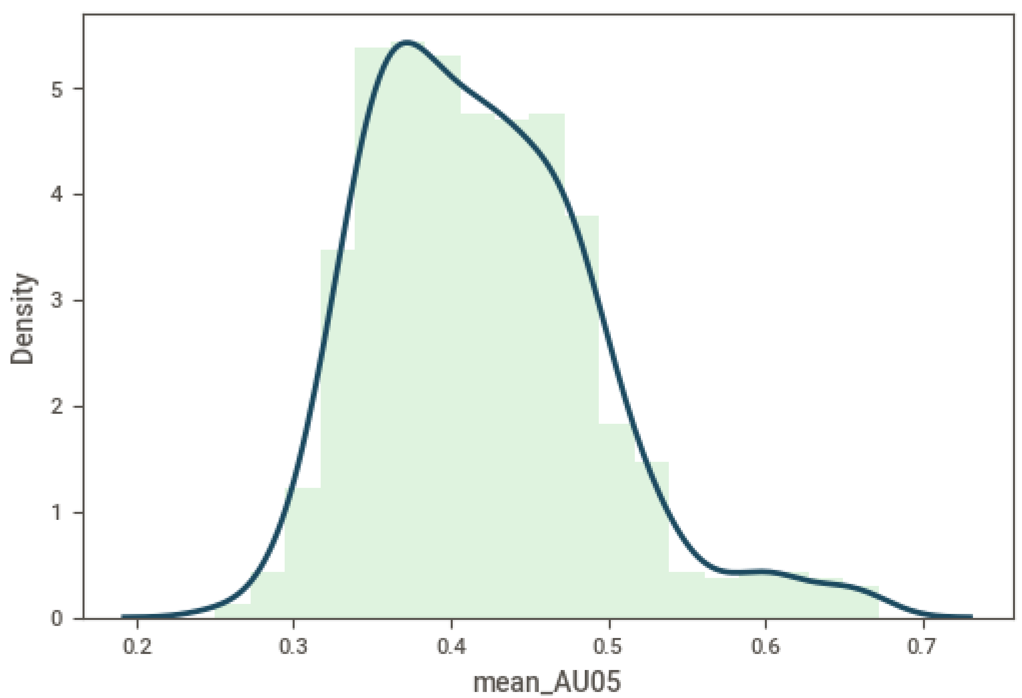
```
df1 = df.select_dtypes(exclude=['object', 'category'])
for column in df1.columns:
    plt.figure()
    sns.distplot(df1[column], color="#AFE1AF", kde_kws={"color": "#1c4a60", "linewidth": 2
plt.show()
```

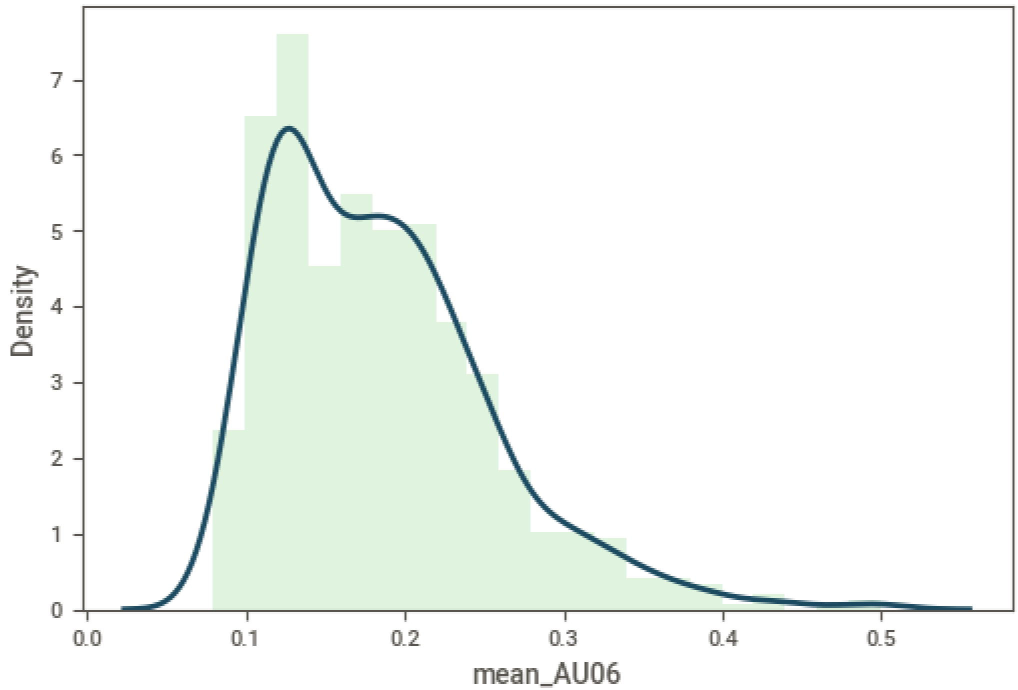
```
C:\Users\hinal\AppData\Local\Temp/ipykernel_11780/821257640.py:4: RuntimeWarning: More tha
n 20 figures have been opened. Figures created through the pyplot interface (`matplotlib.p
yplot.figure`) are retained until explicitly closed and may consume too much memory. (To c
ontrol this warning, see the rcParam `figure.max_open_warning`).
  plt.figure()
```



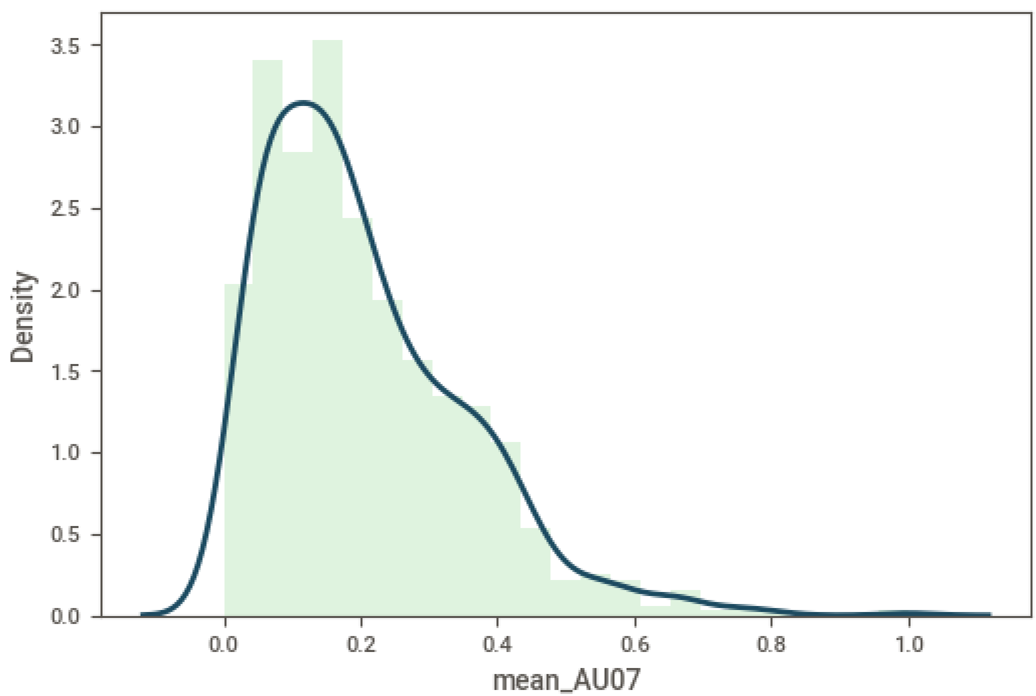



Density
0
1
2
3
4
5
6
7
0.2
0.3
0.4
0.5
0.6
0.7
0.8
mean_AU04

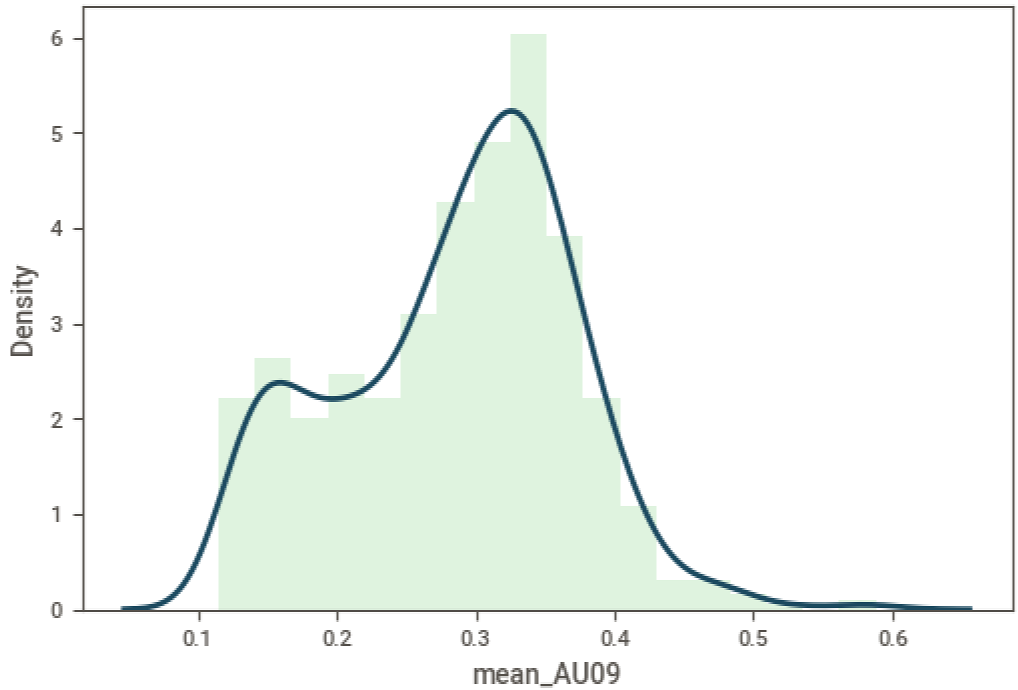

Density
0
1
2
3
4
5
0.2
0.3
0.4
0.5
0.6
0.7
mean_AU05



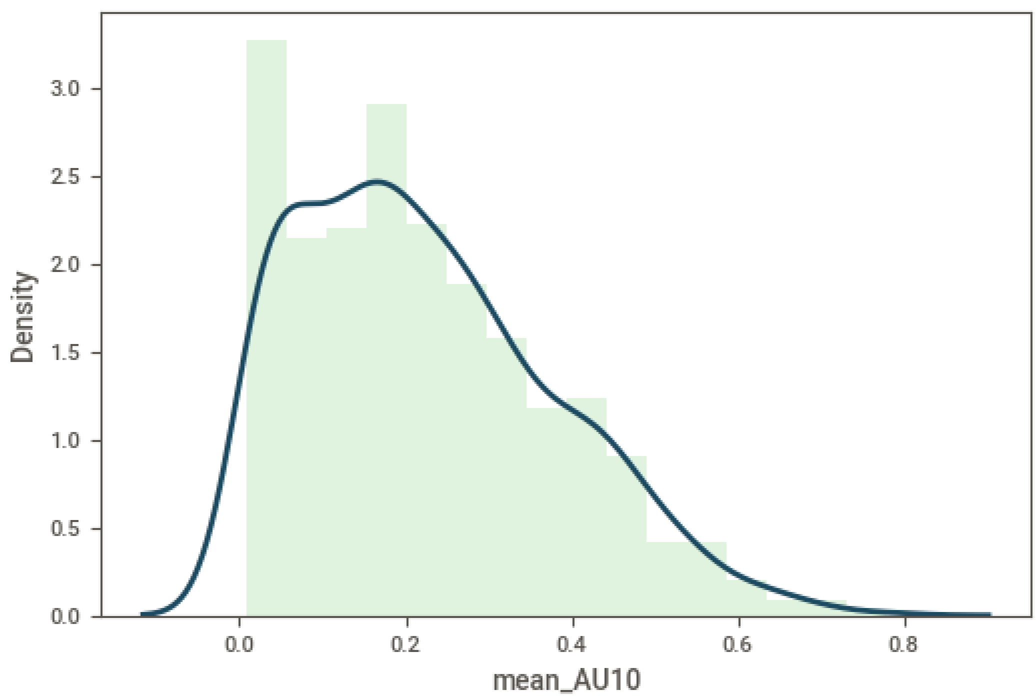
Density
mean_AU06
0.0
0.1
0.2
0.3
0.4
0.5
0
1
2
3
4
5
6
7



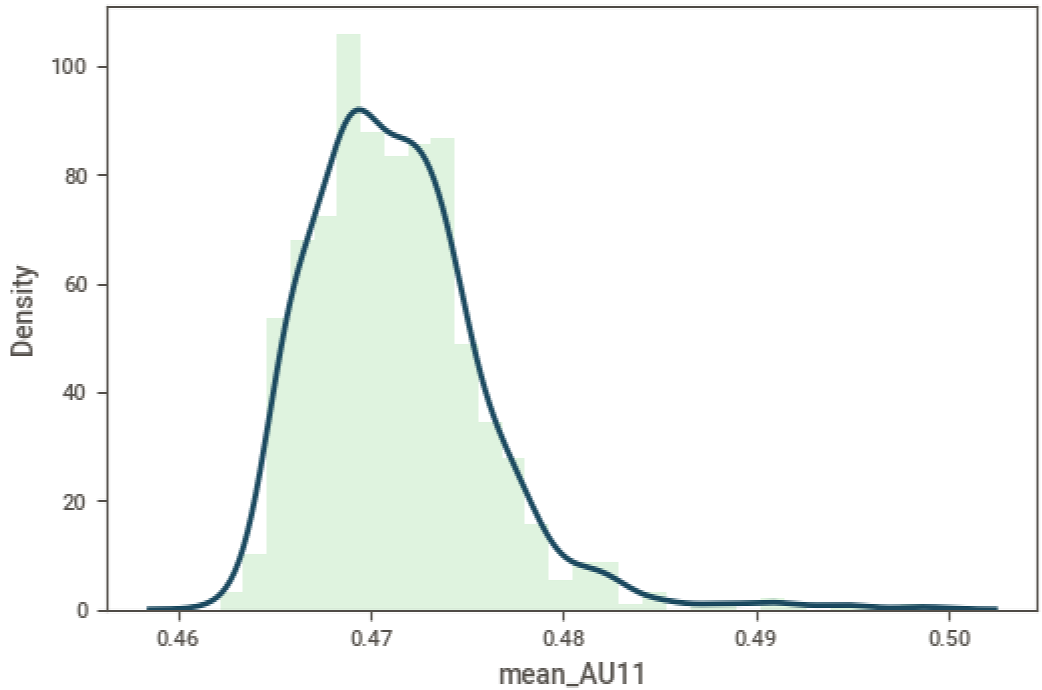
Density
mean_AU07
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5

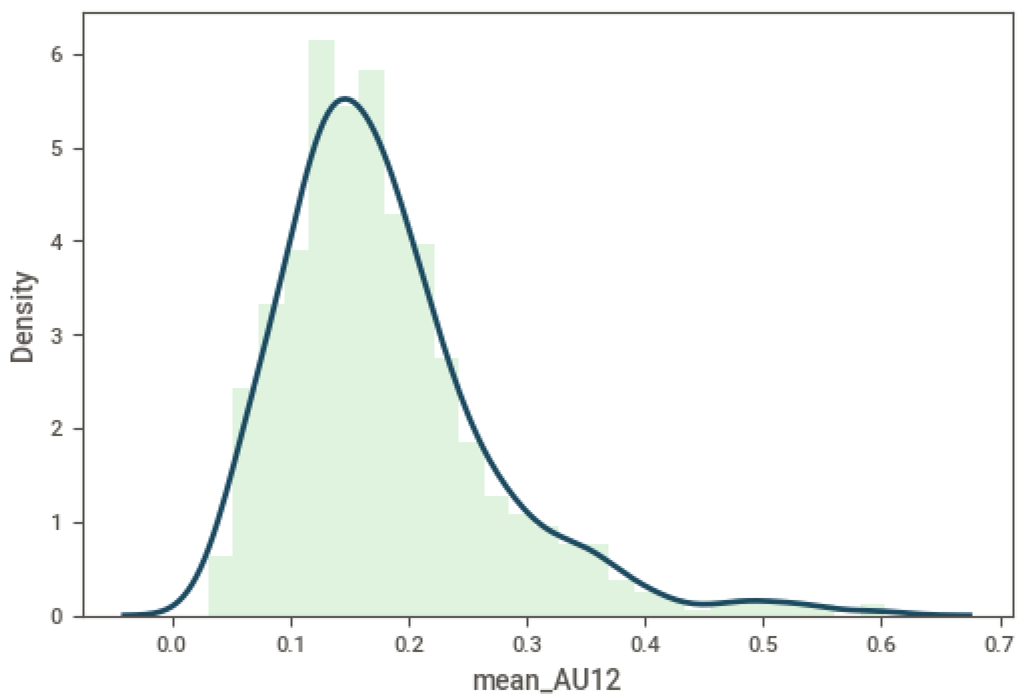


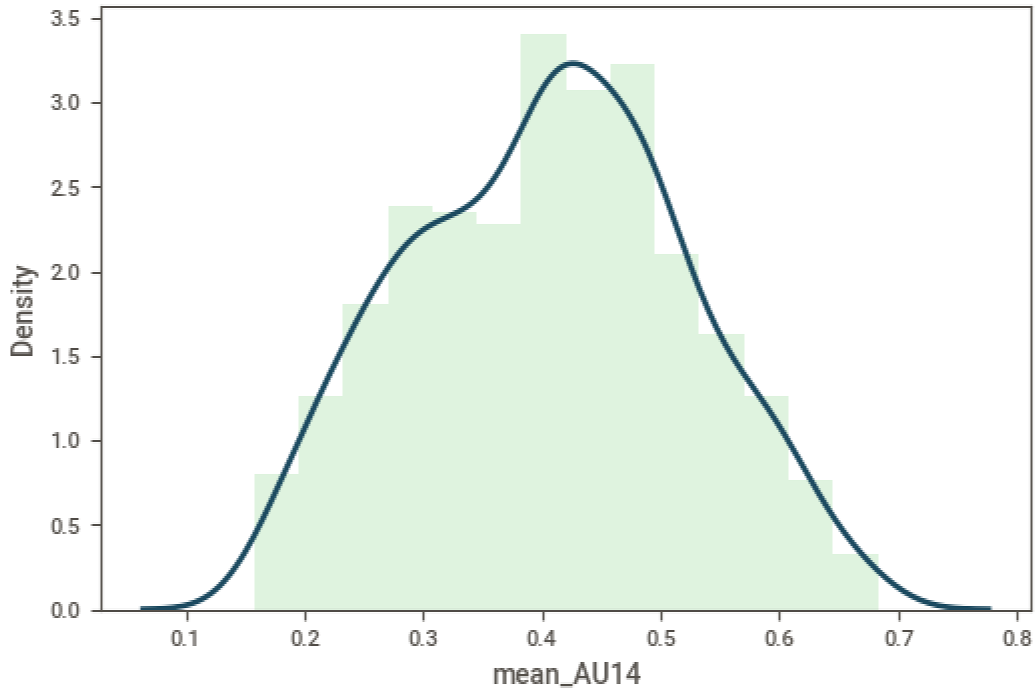
Density
mean_AU09
0
1
2
3
4
5
6
0.1
0.2
0.3
0.4
0.5
0.6

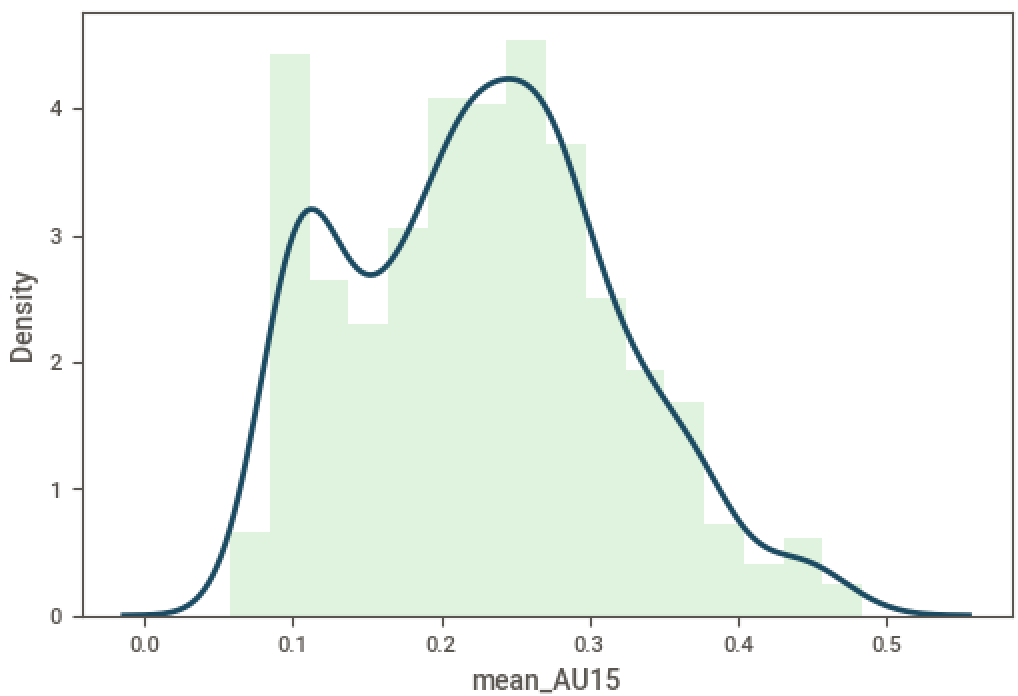


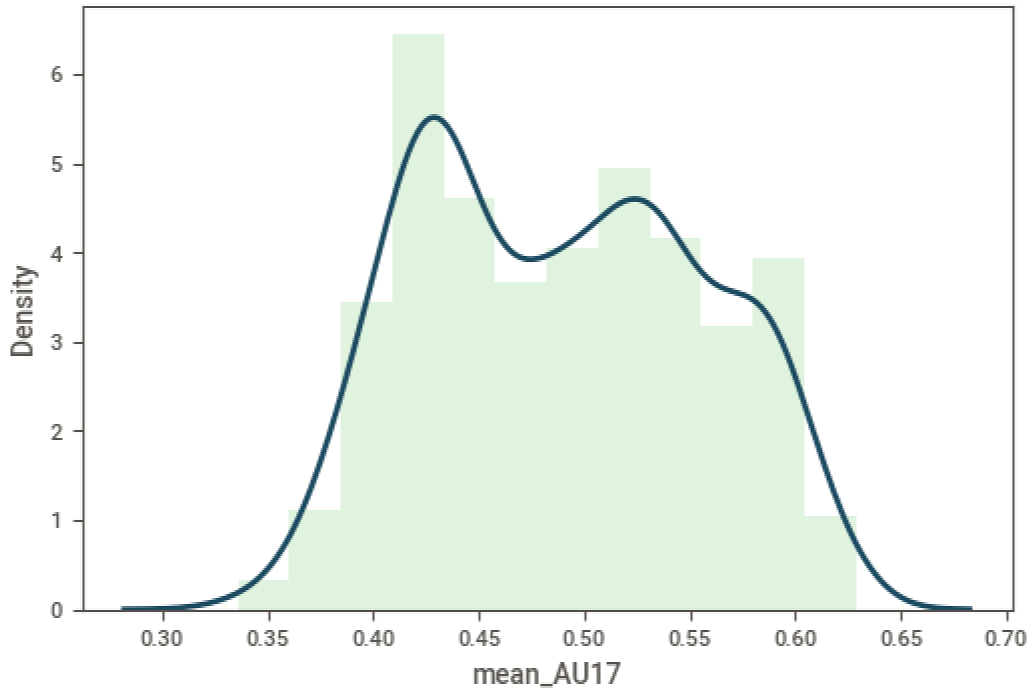
Density
mean_AU10
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0.0
0.2
0.4
0.6
0.8


Density
mean_AU11
0
20
40
60
80
100
0.46
0.47
0.48
0.49
0.50

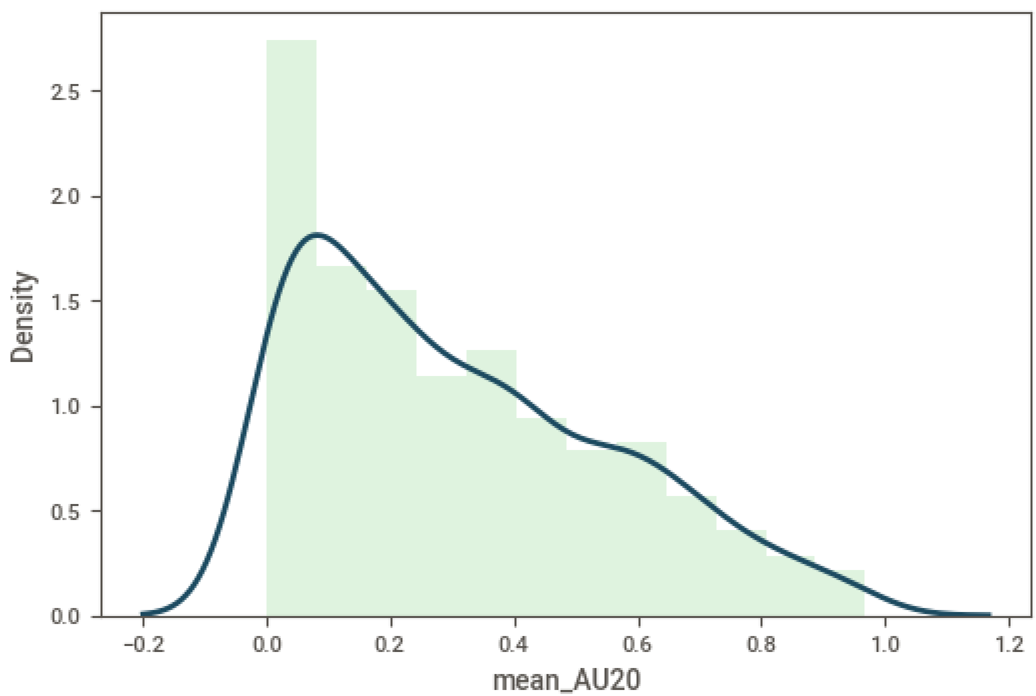

Density
mean_AU12
0
1
2
3
4
5
6
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7


Density
mean_AU14
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8

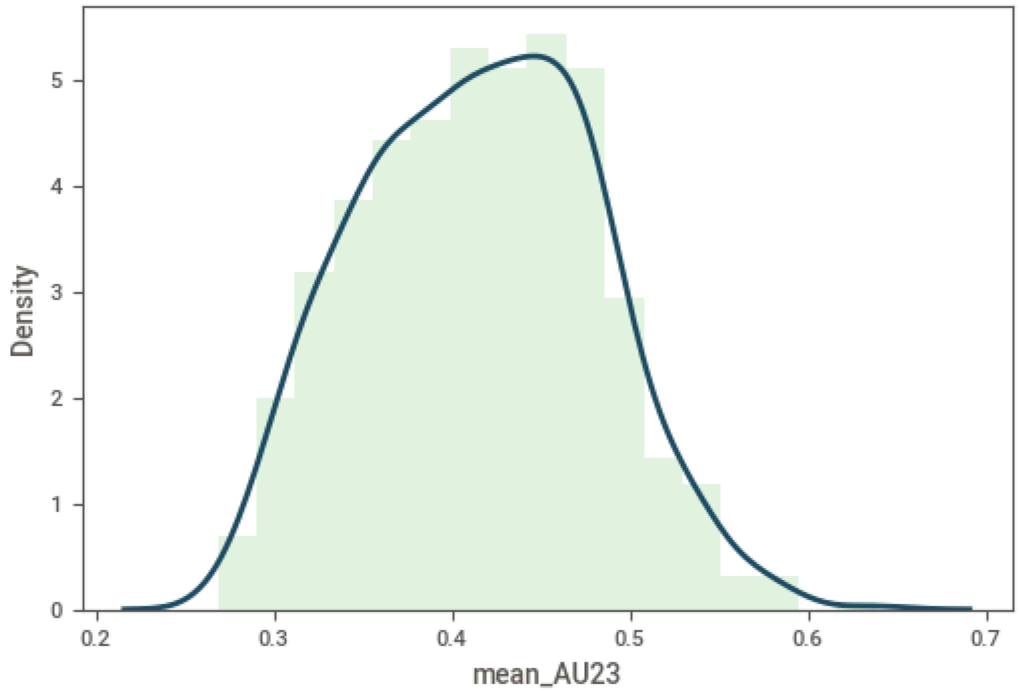

Density
mean_AU15
0
1
2
3
4
0.0
0.1
0.2
0.3
0.4
0.5

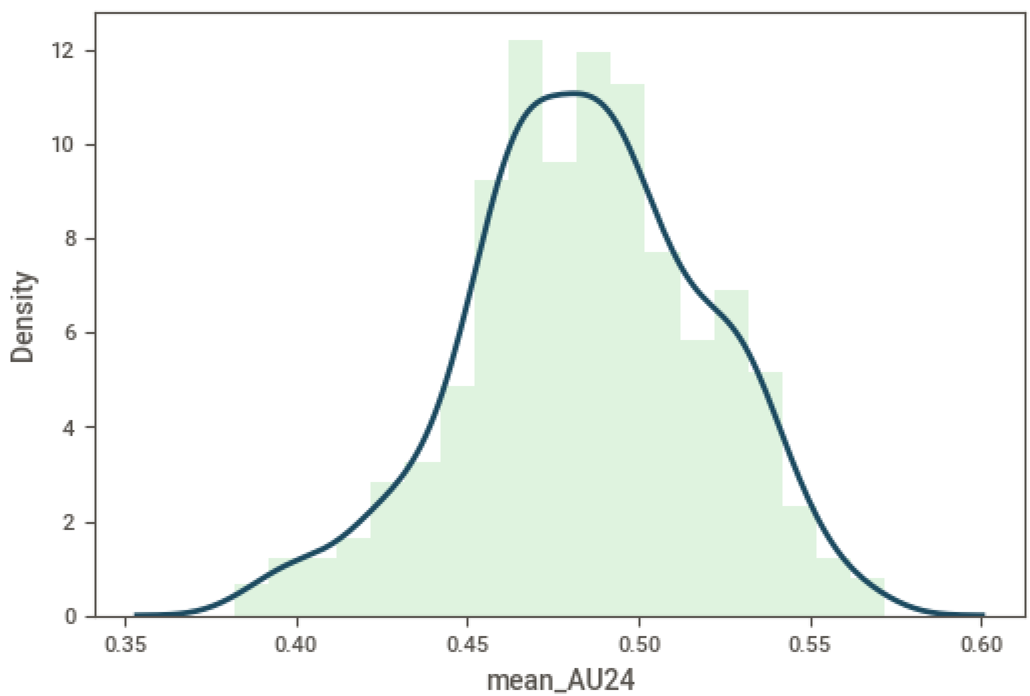


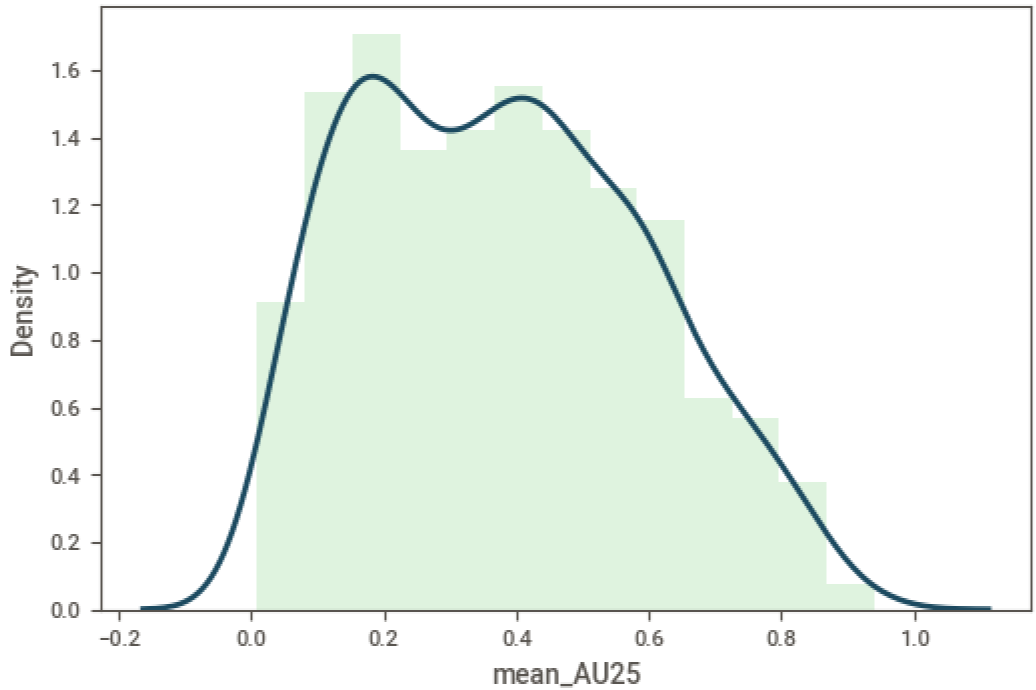
Density
mean_AU17



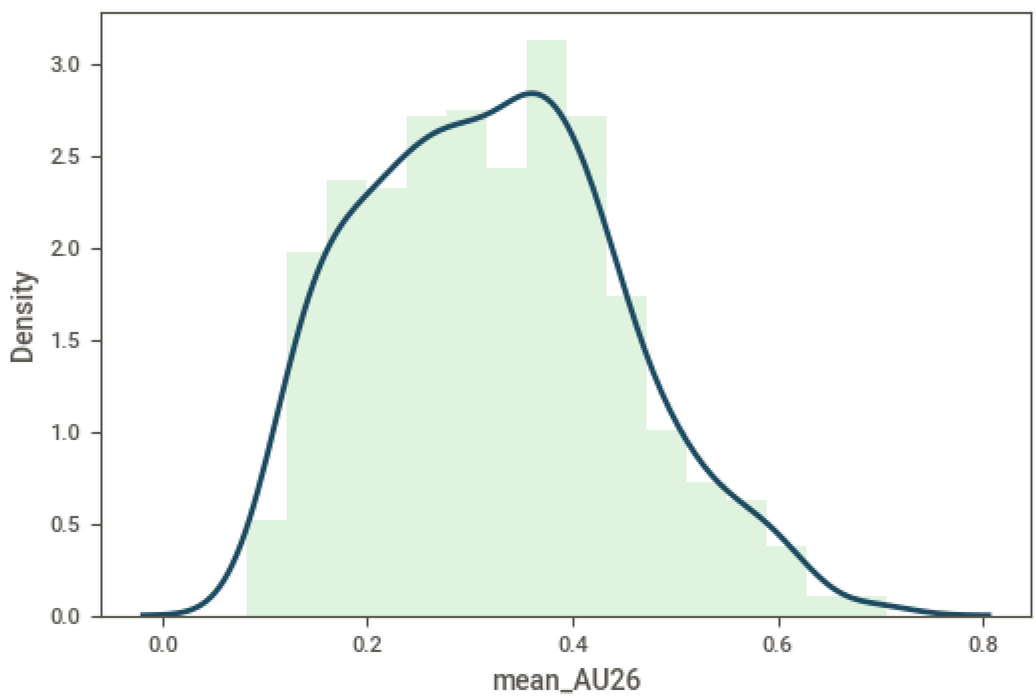
Density
mean_AU20


Density
mean_AU23

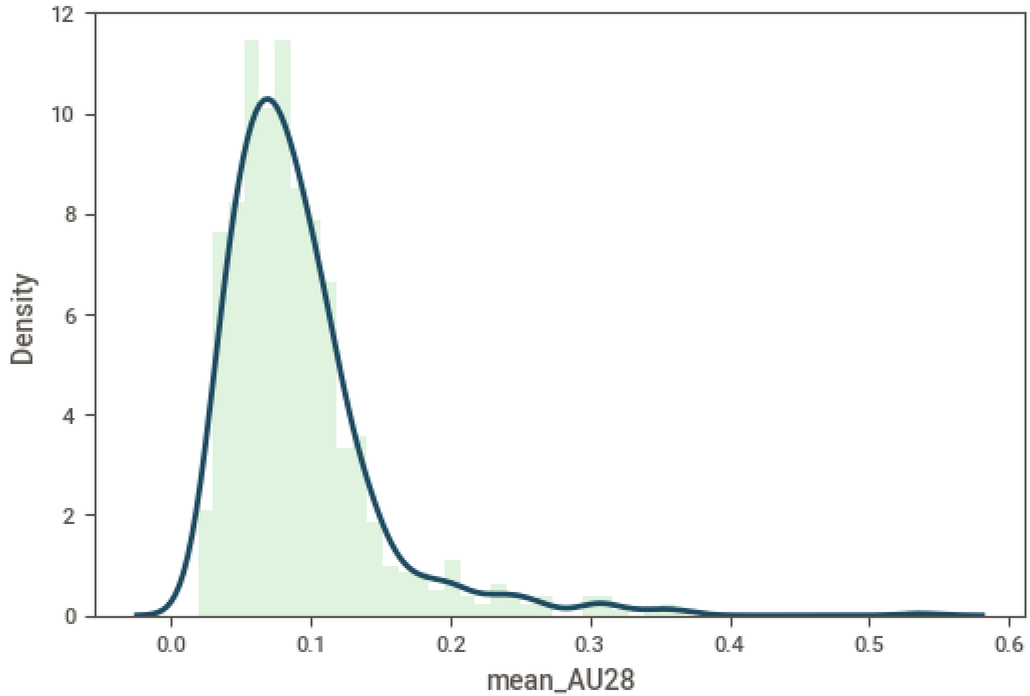

Density
mean_AU24



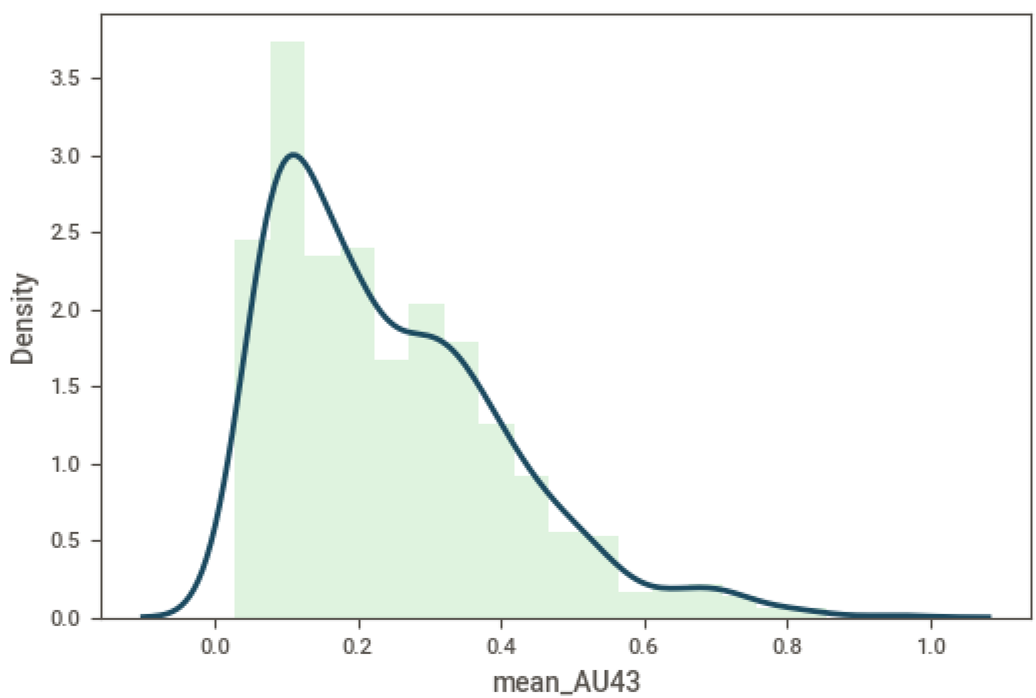
Density
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
mean_AU25



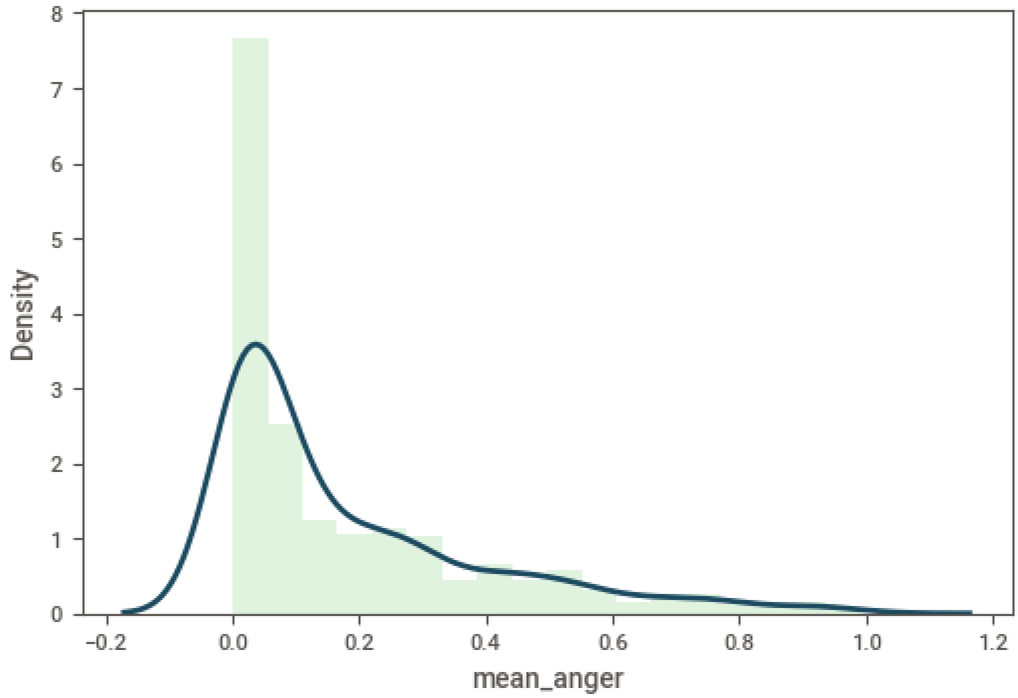
Density
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8
mean_AU26

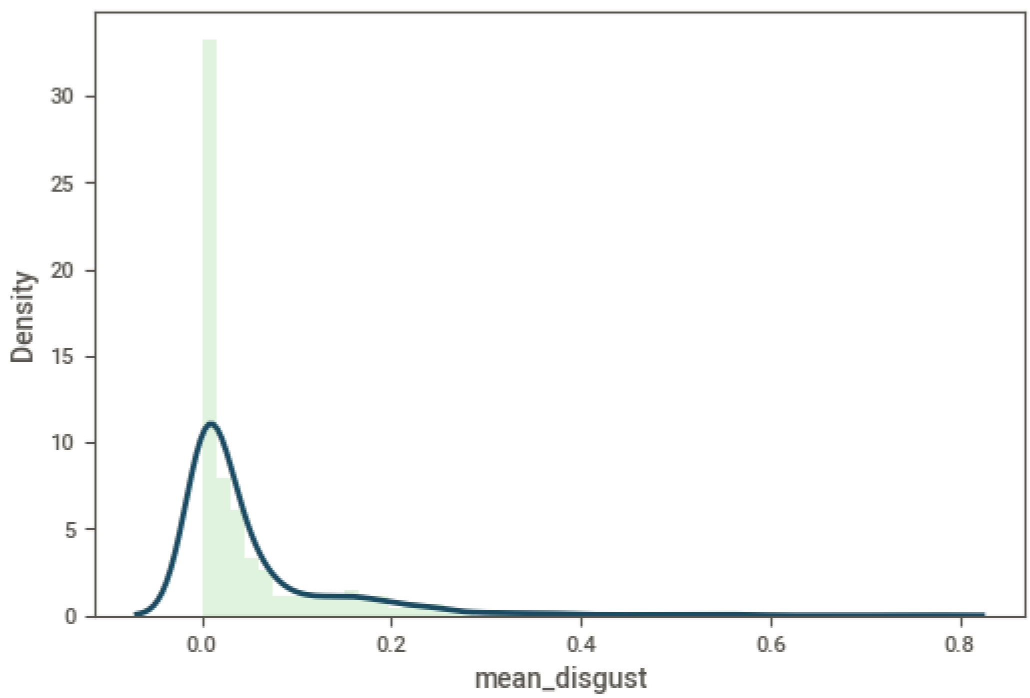


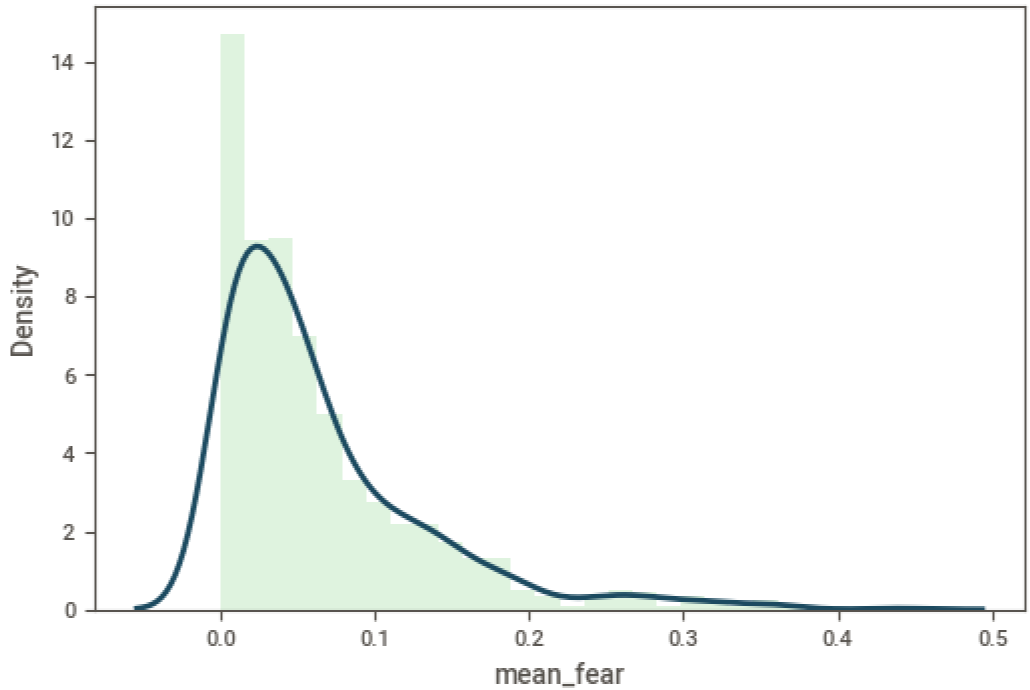
Density
mean_AU28
12
10
8
6
4
2
0
0.0
0.1
0.2
0.3
0.4
0.5
0.6



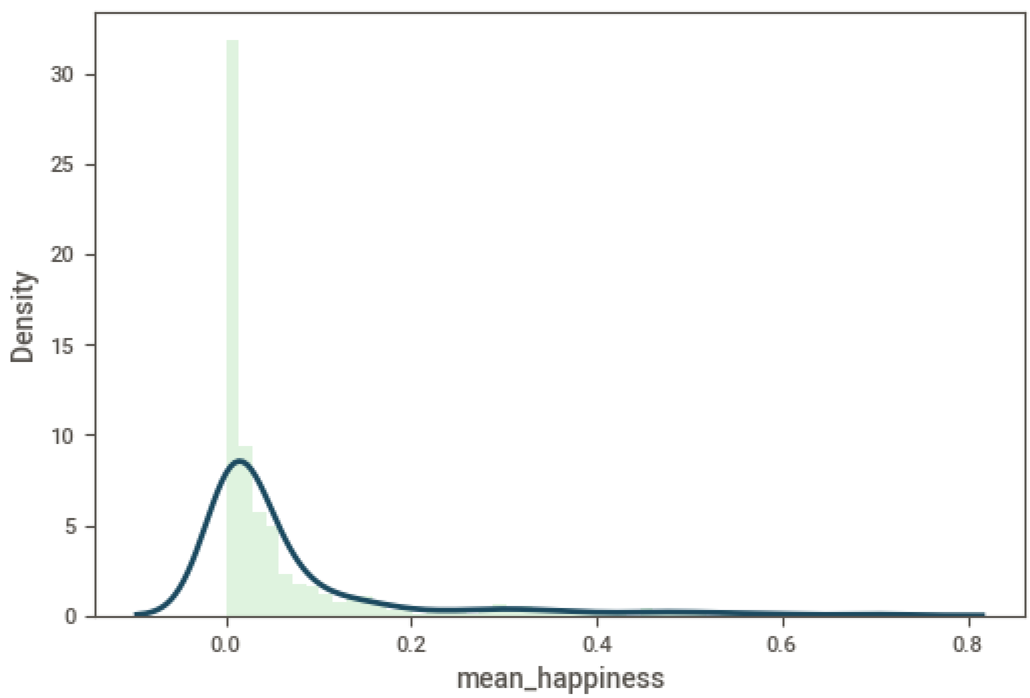
Density
mean_AU43
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0


Density
mean_anger
0
1
2
3
4
5
6
7
8
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2

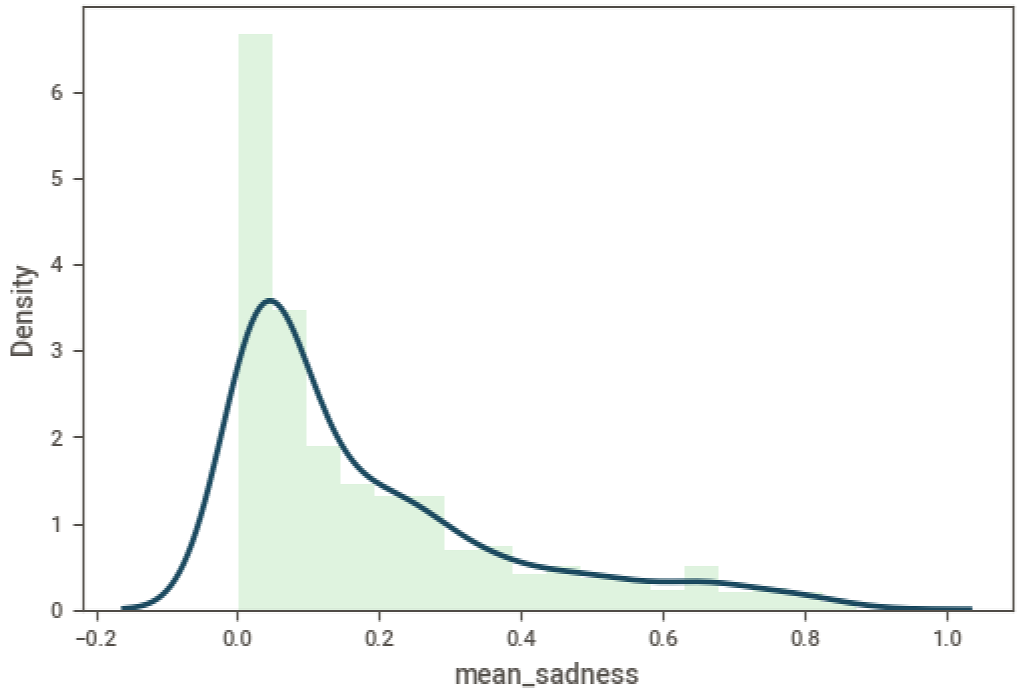

Density
mean_disgust
0
5
10
15
20
25
30
0.0
0.2
0.4
0.6
0.8

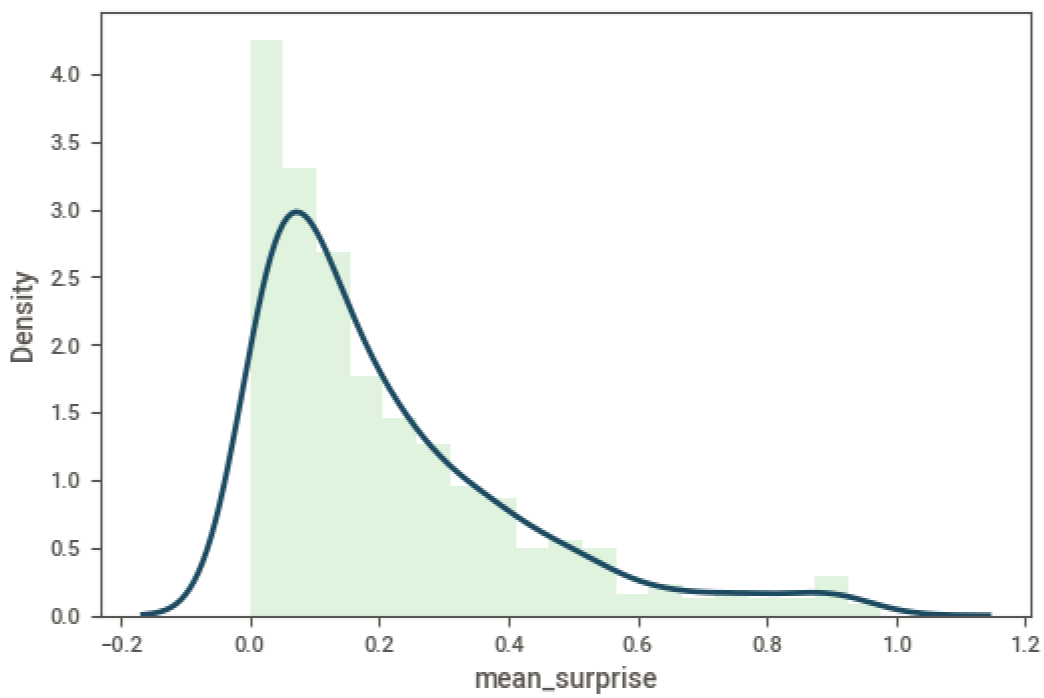


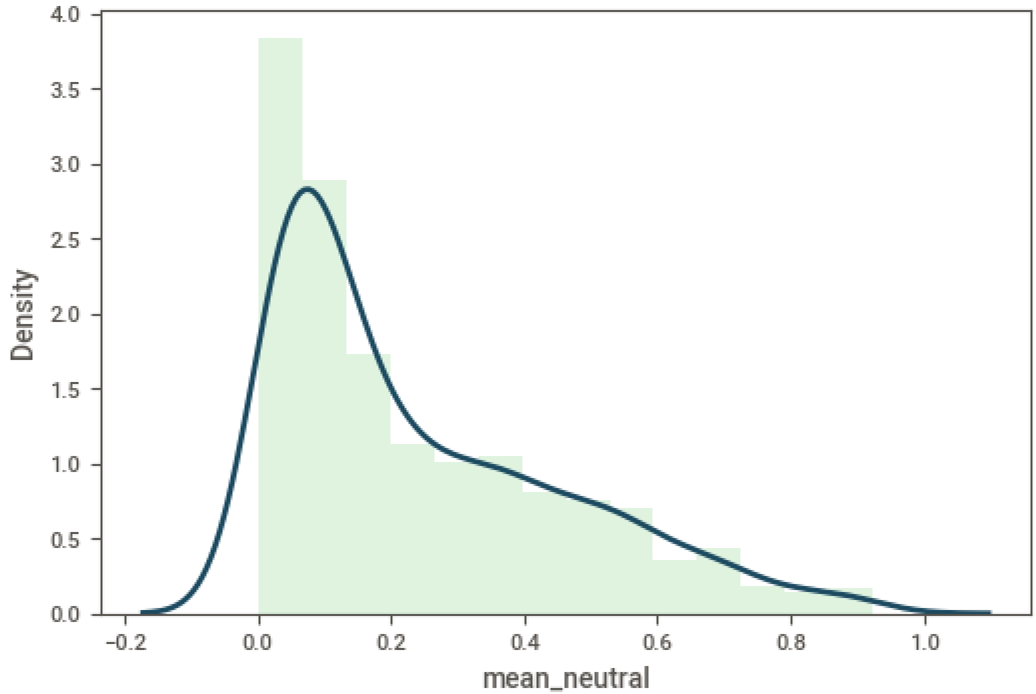
Density
mean_fear



Density
mean_happiness

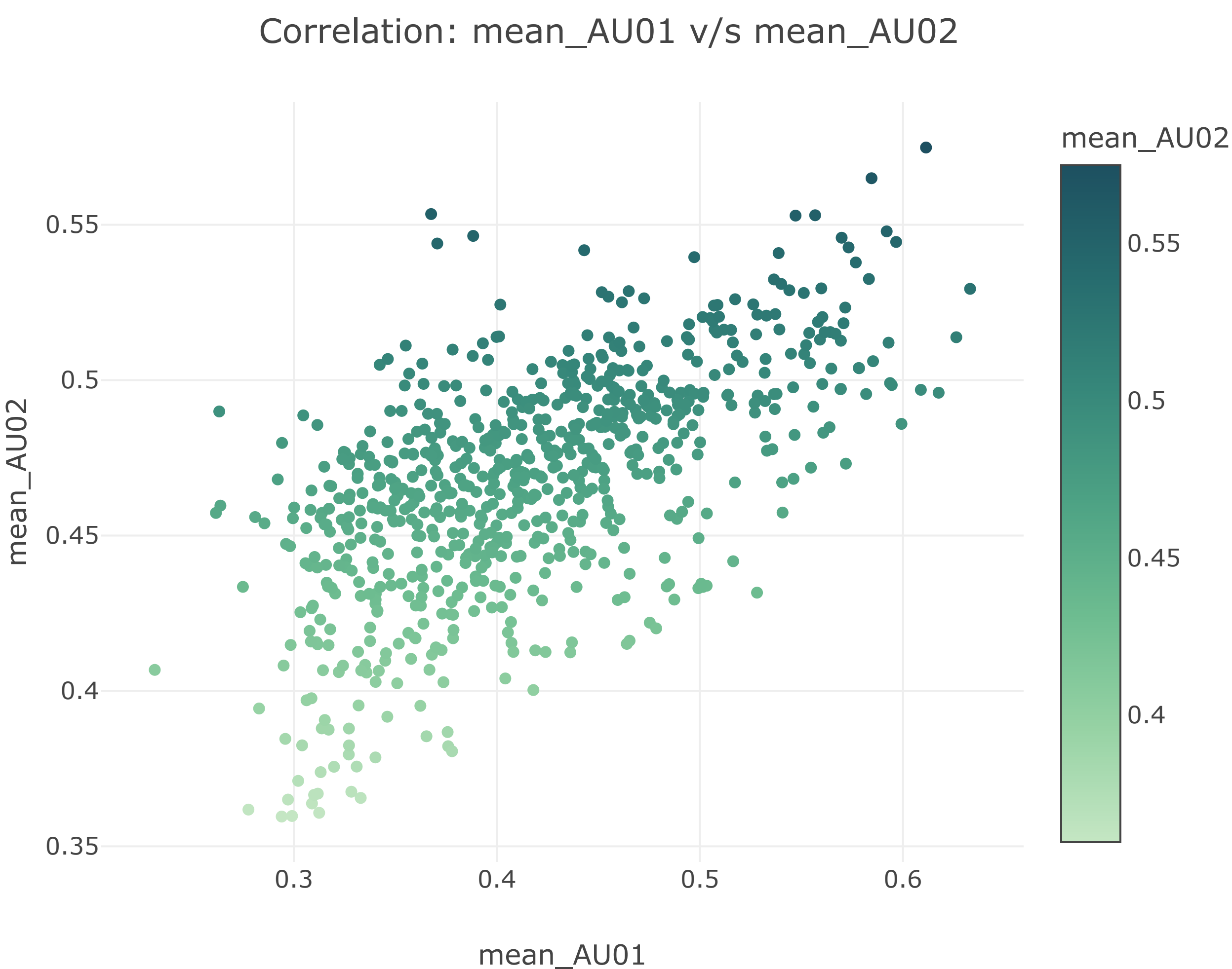

Density
mean_sadness


Density
mean_surprise



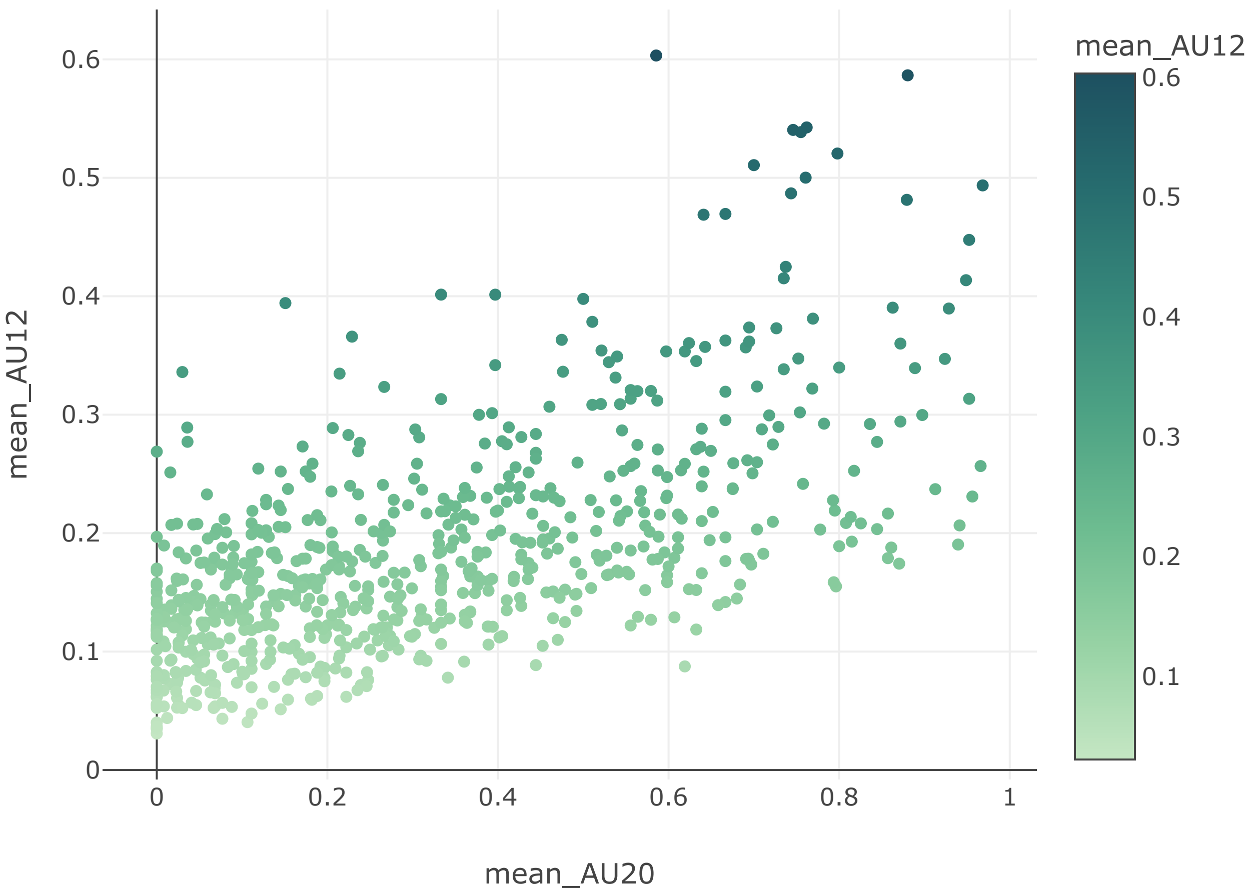
Let us find the correlation graphs for the most correlated parameters

In [15]:
```
for value in included_pairs:
    PlotGraphs.ScaterPlot(df,[df.columns.get_loc(value[0]),df.columns.get_loc(value[1])],'
```

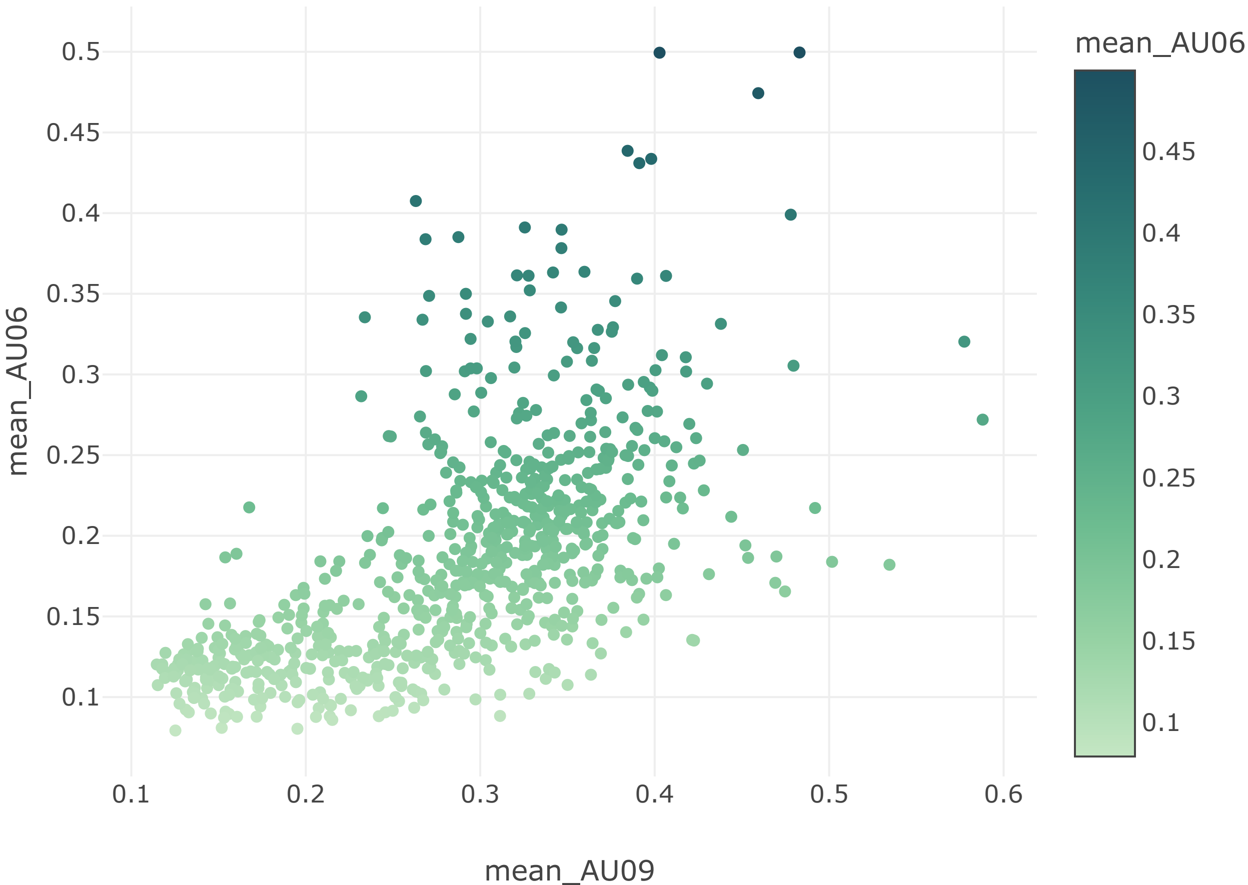


Correlation: mean_AU17 v/s mean_AU15

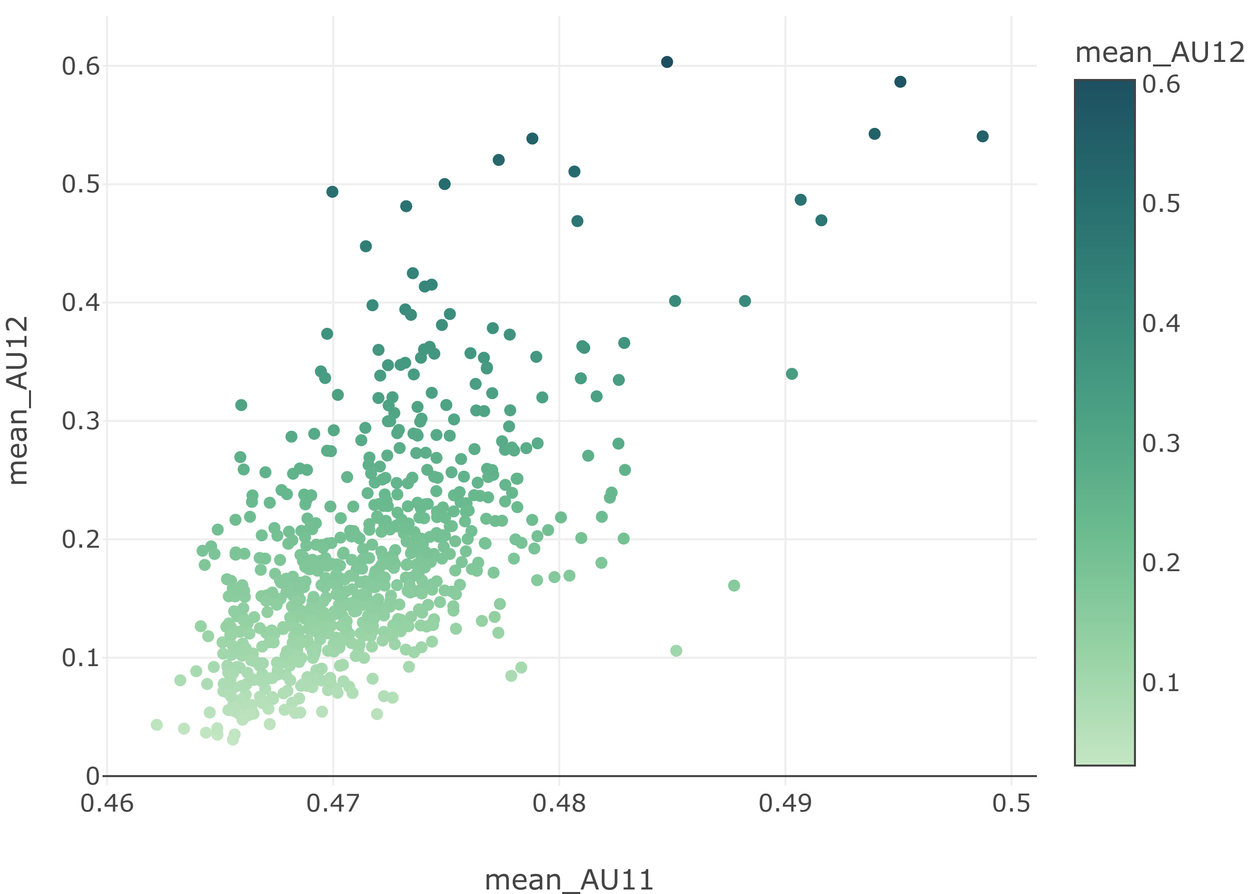
Correlation: mean_AU20 v/s mean_AU12



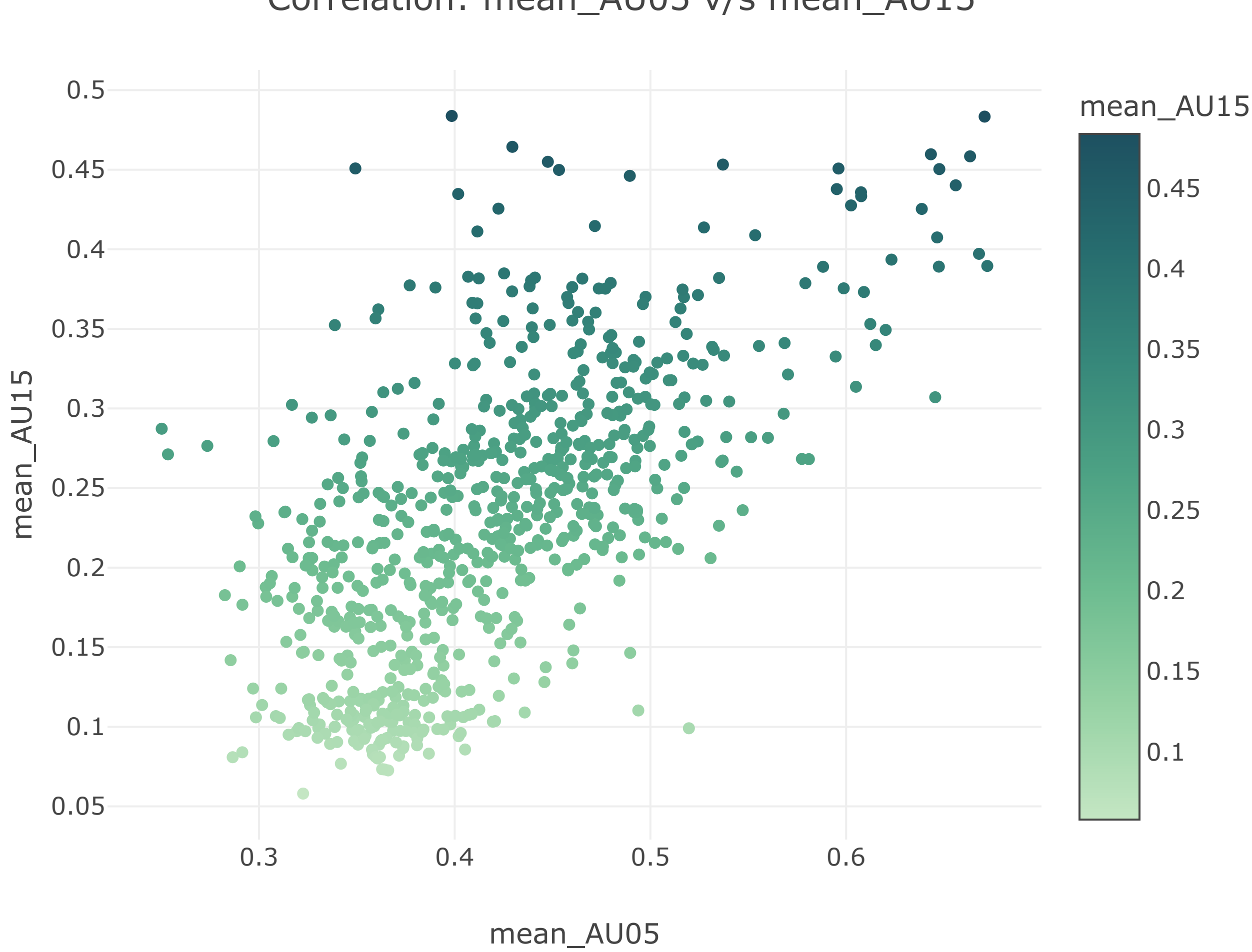
Correlation: mean_AU09 v/s mean_AU06



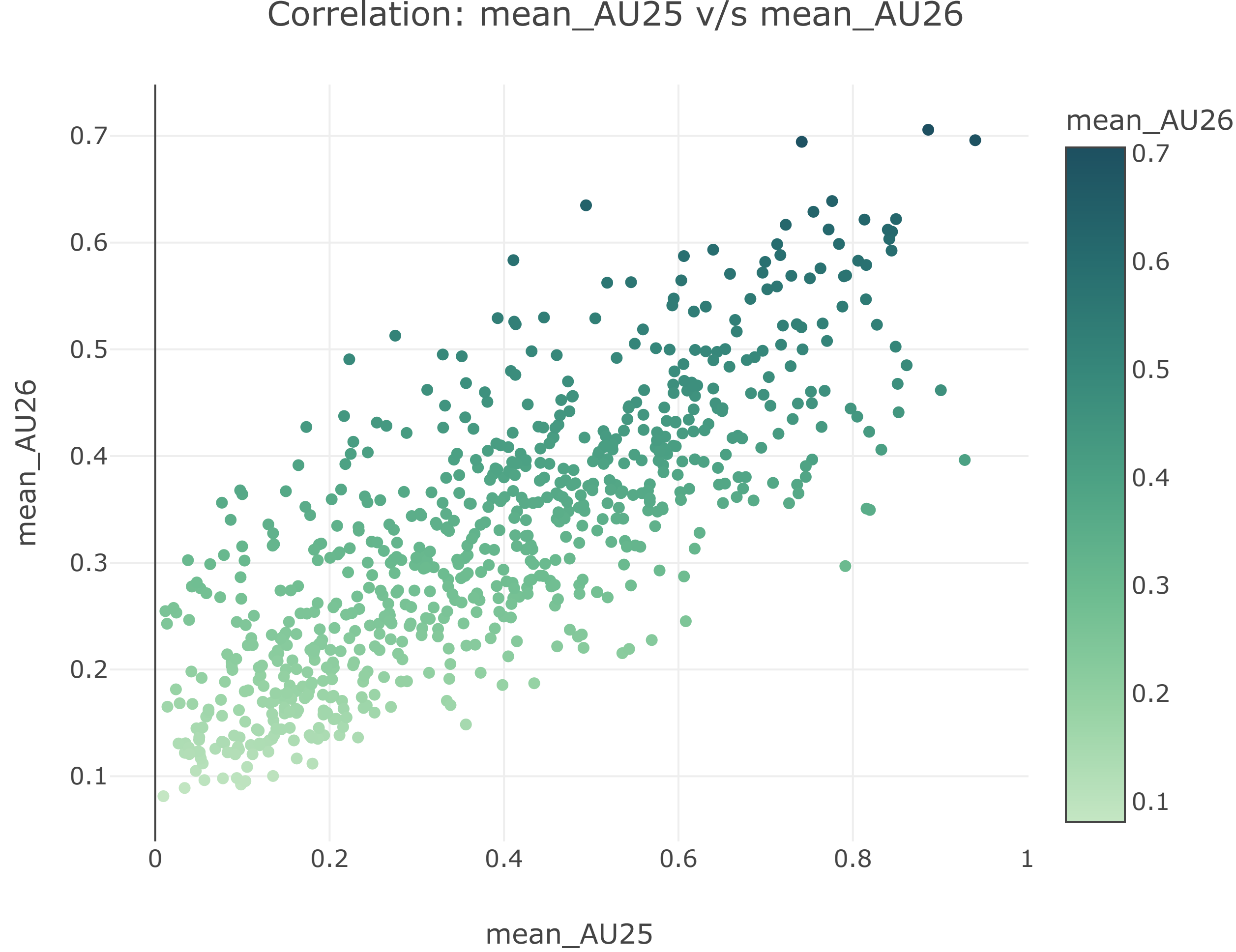
Correlation: mean_AU11 v/s mean_AU12

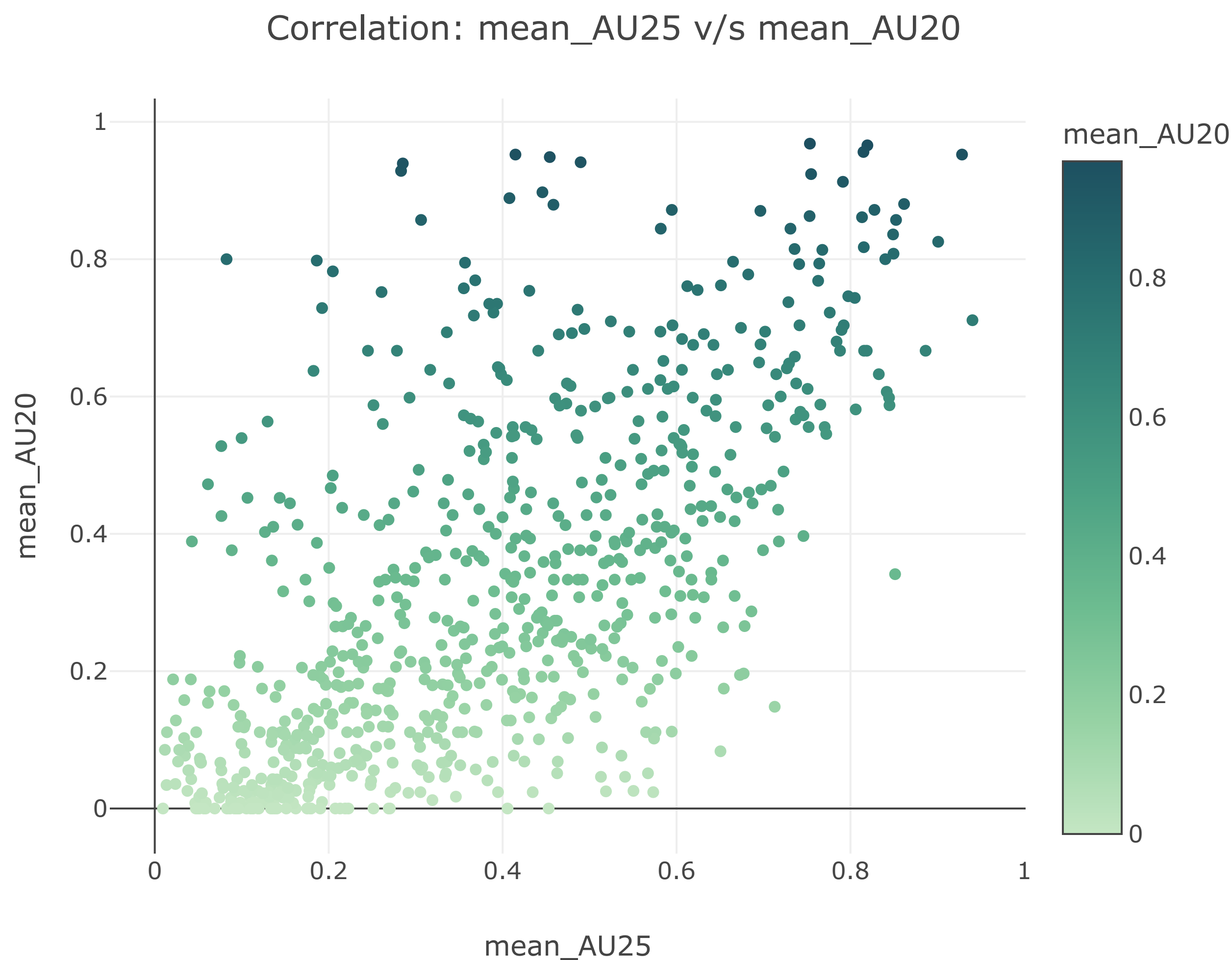


Correlation: mean_AU05 v/s mean_AU15


Correlation: mean_AU05 v/s mean_AU15
mean_AU15
mean_AU05
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0.3
0.4
0.5
0.6

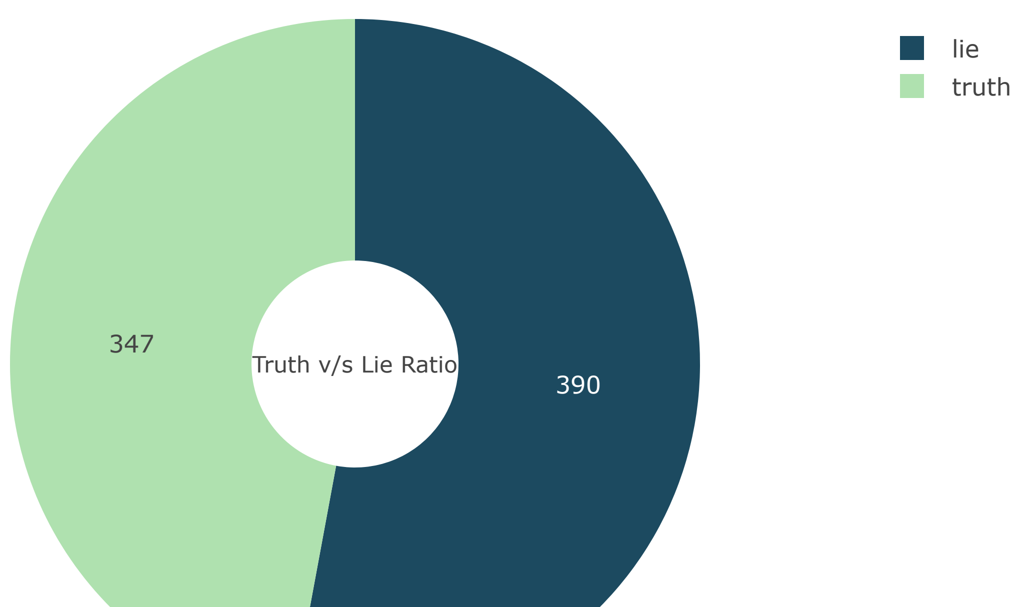

Correlation: mean_AU06 v/s mean_AU12
mean_AU12
mean_AU06
0.6
0.5
0.4
0.3
0.2
0.1
0
0.1
0.2
0.3
0.4
0.5


Correlation: mean_AU25 v/s mean_AU26
mean_AU26
mean_AU25
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
0.2
0.4
0.6
0.8
1


Correlation: mean_AU25 v/s mean_AU20
mean_AU20
mean_AU25
0
0.2
0.4
0.6
0.8
1

In [16]:

```
df1 = df \
      .groupby("Label") \
      .agg(count=("VideoName","count"),
           Label = ("Label","first")) \
      .sort_values(by="count",ascending=False)
PlotGraphs.DonutChart(df1,0,1,"Truth v/s Lie Ratio")
```



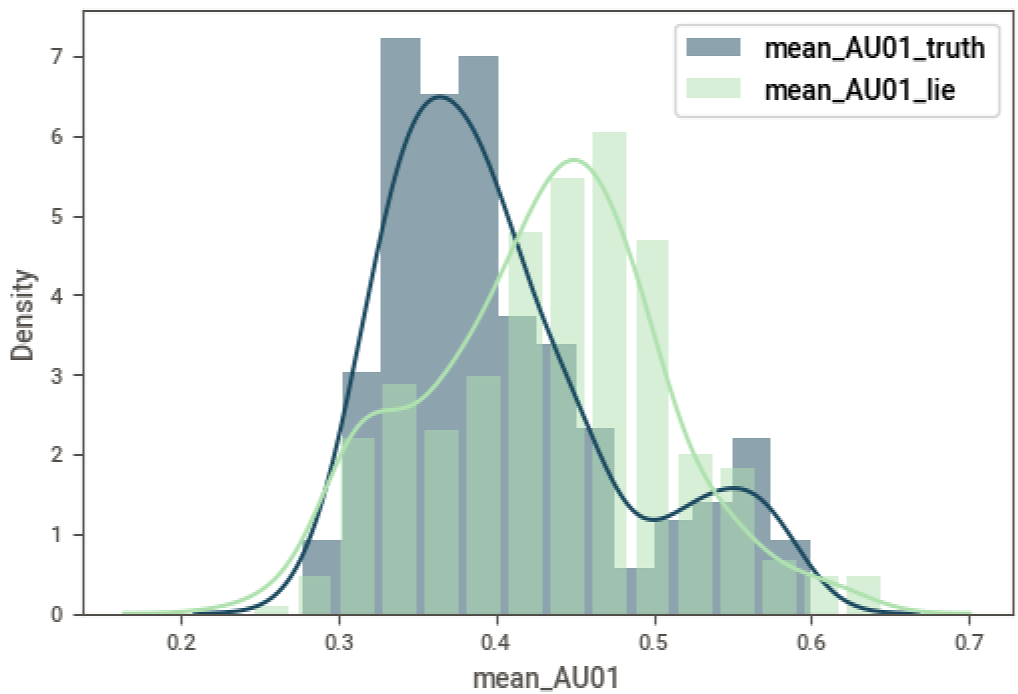
Let us now compare the overall distribution of data and the distribution of data for the lie labels.

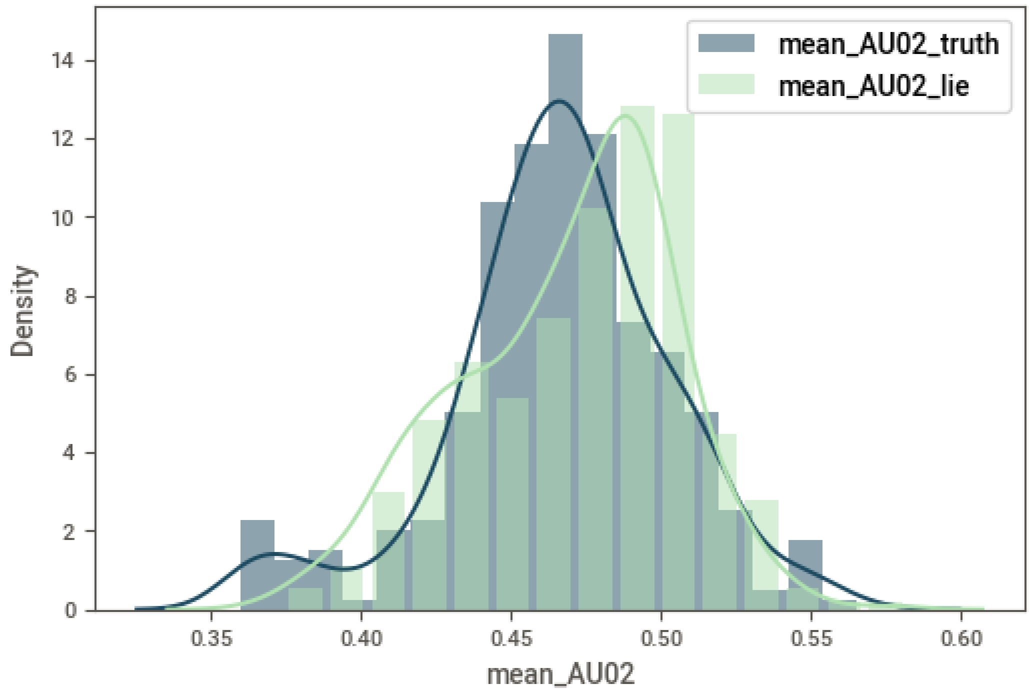
In [17]:
```
warnings.filterwarnings(action='ignore', category=FutureWarning)
df1 = df[df['Label'] == 'truth']
df1 = df1.select_dtypes(exclude=['object', 'category'])
df2 = df[df['Label'] == 'lie']
df2 = df2.select_dtypes(exclude=['object', 'category'])

for column in df1.columns:
    plt.figure()
    sns.distplot(df1[column], color='#1c4a60', label=column+"_truth", hist_kws={'alpha': (
    sns.distplot(df2[column], color='#AFE1AF', label=column+"_lie", hist_kws={'alpha': 0.5
    plt.legend()
plt.show()
```

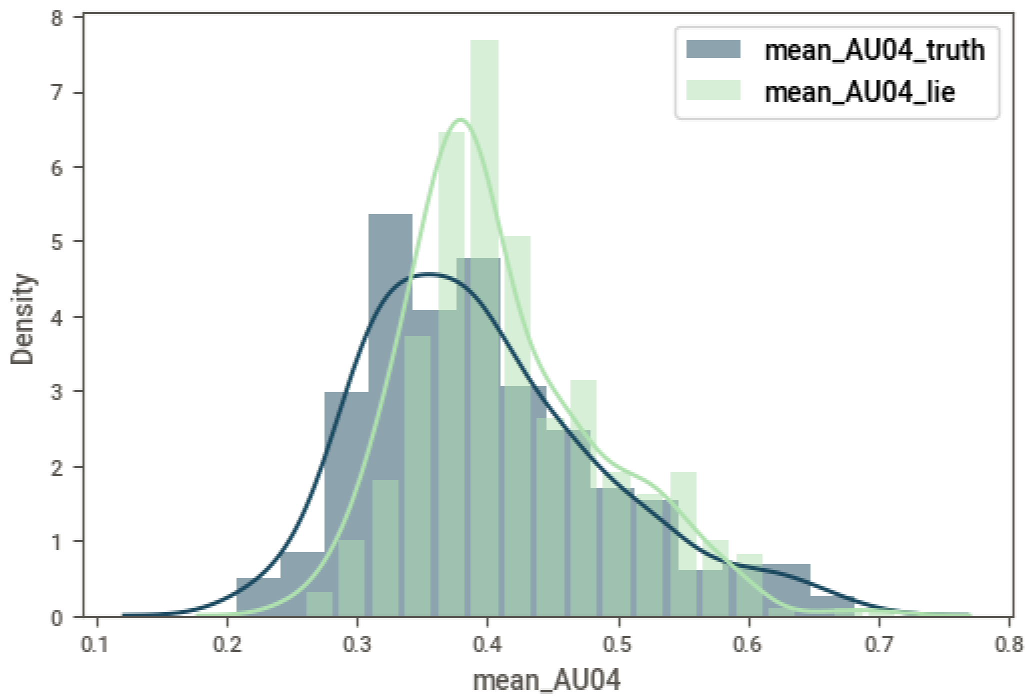
```
C:\Users\hinal\AppData\Local\Temp/ipykernel_11780/1882150525.py:8: RuntimeWarning:

More than 20 figures have been opened. Figures created through the pyplot interface (`matp
lotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memor
y. (To control this warning, see the rcParam `figure.max_open_warning`).
```

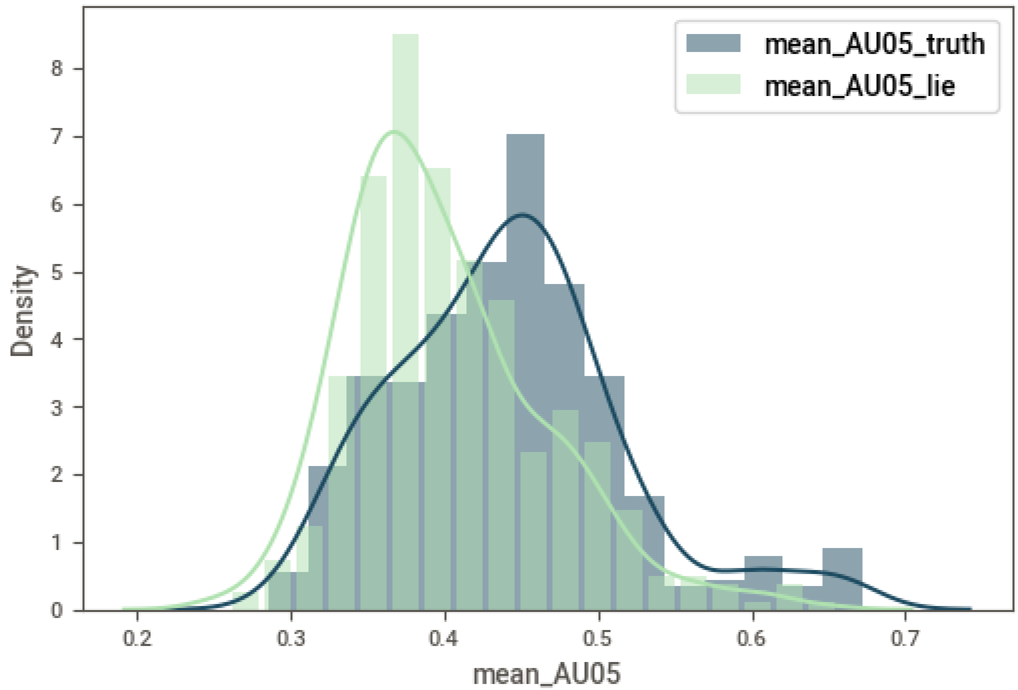




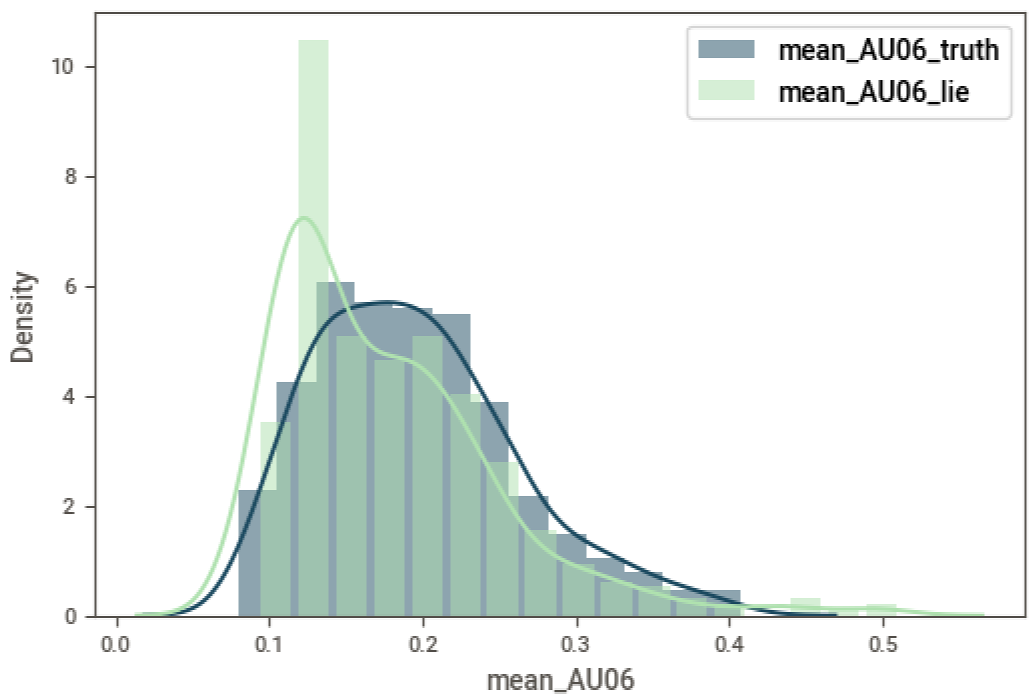
mean_AU02_truth
mean_AU02_lie
Density
mean_AU02



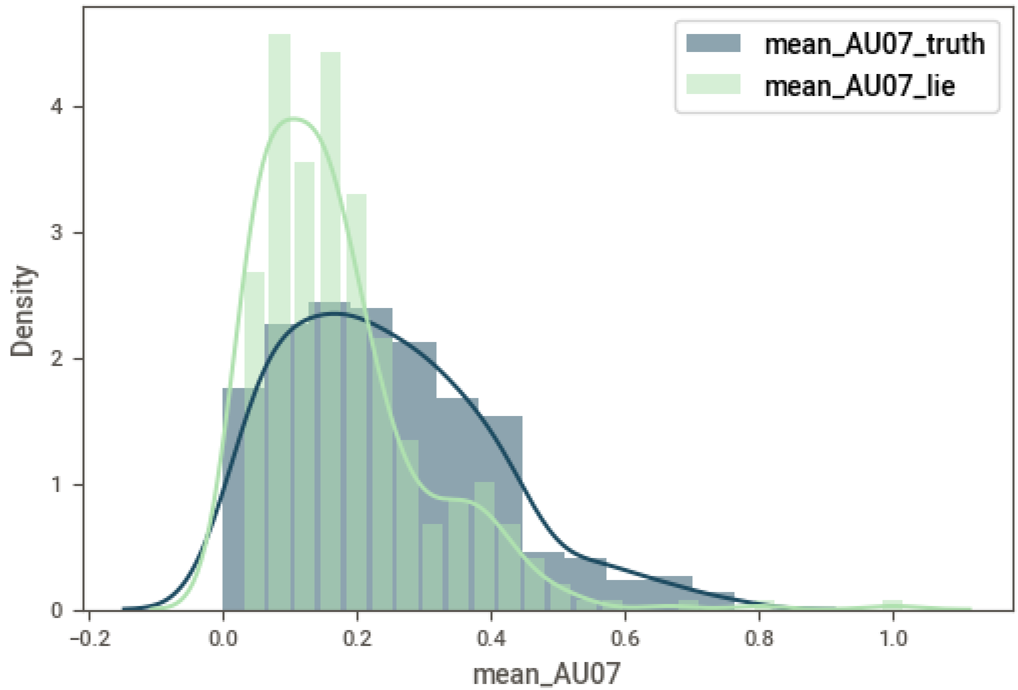
mean_AU04_truth
mean_AU04_lie
Density
mean_AU04

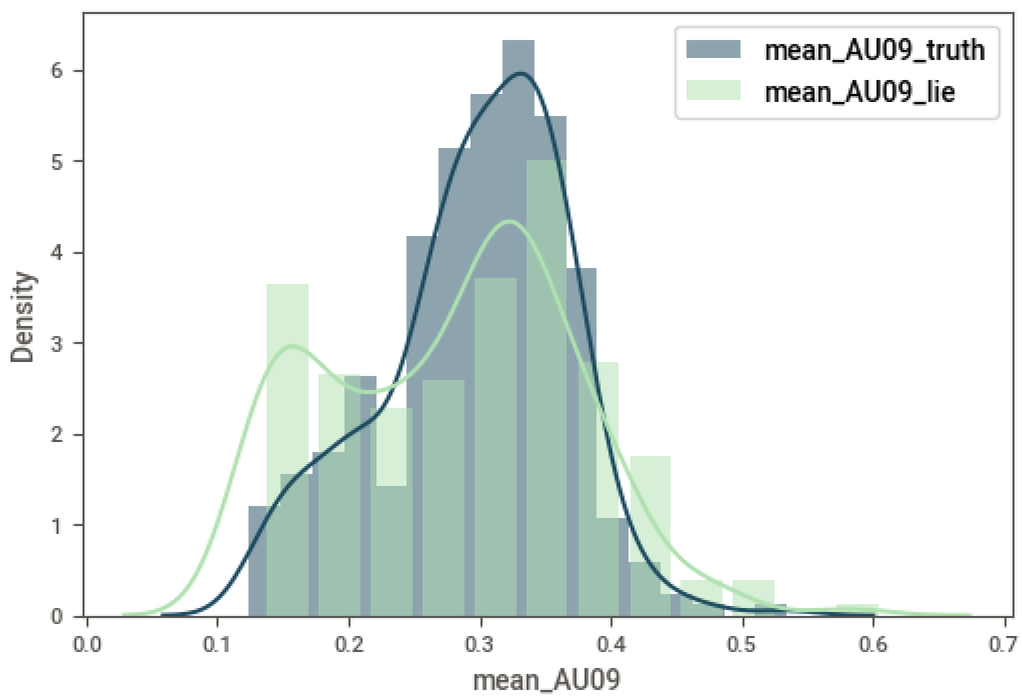


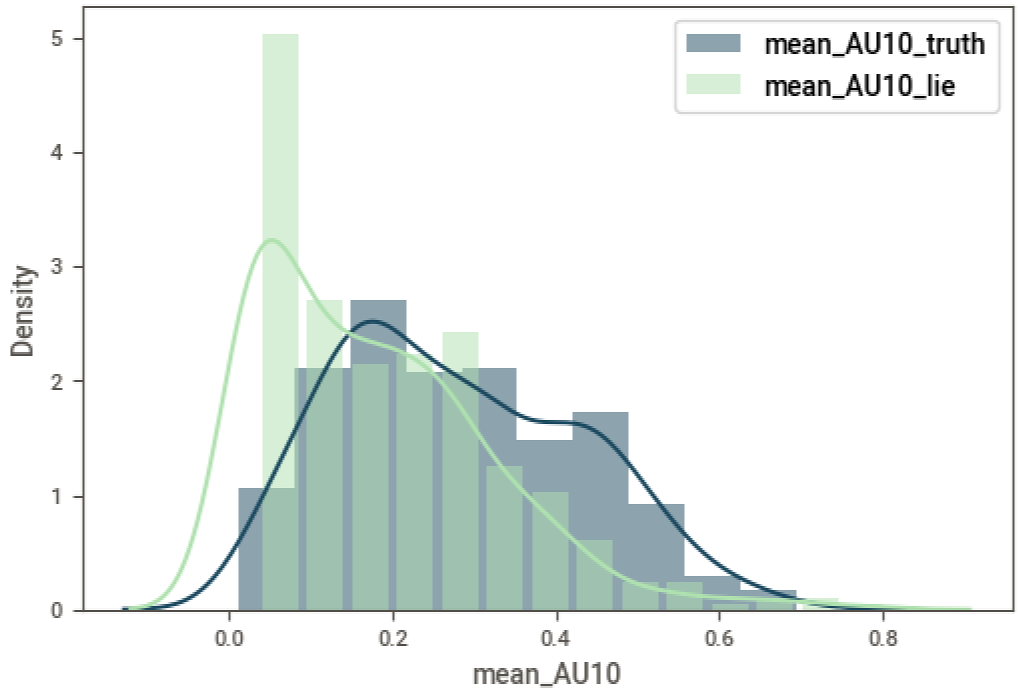
mean_AU05_truth
mean_AU05_lie
Density
mean_AU05



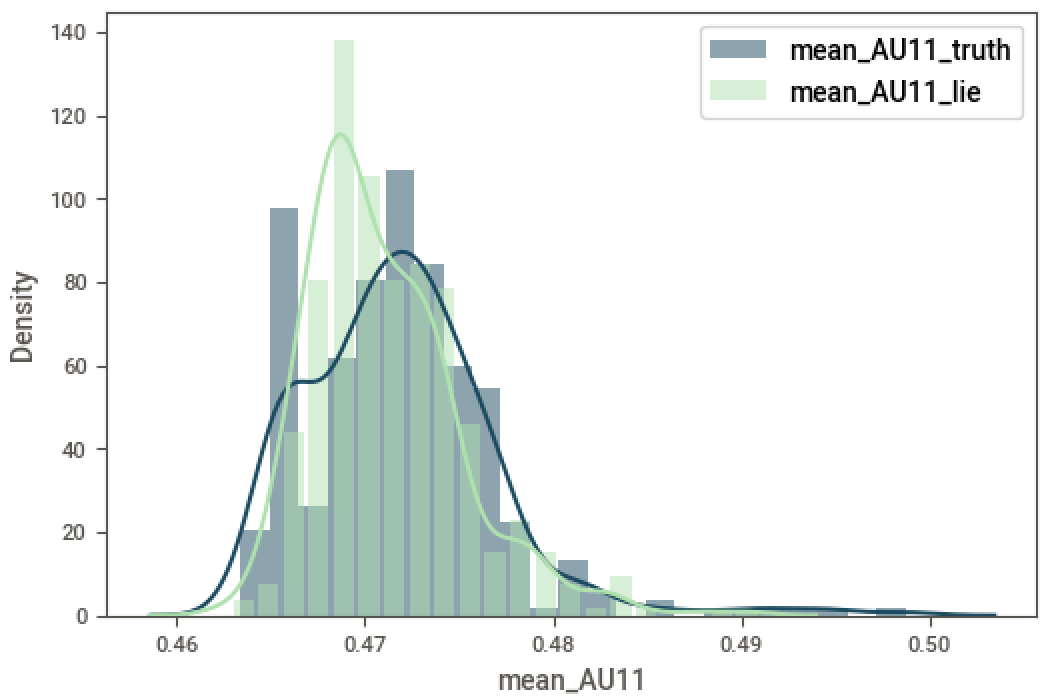
mean_AU06_truth
mean_AU06_lie
Density
mean_AU06


mean_AU07_truth
mean_AU07_lie
Density
mean_AU07
4
3
2
1
0
-0.2
0.0
0.2
0.4
0.6
0.8
1.0

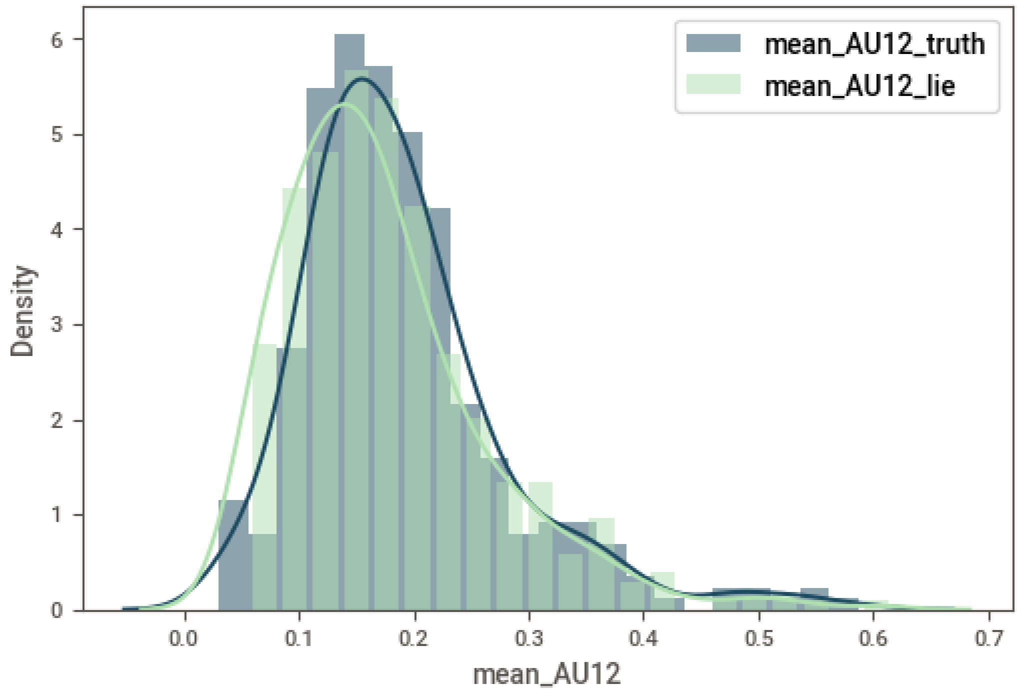

mean_AU09_truth
mean_AU09_lie
Density
mean_AU09
6
5
4
3
2
1
0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7

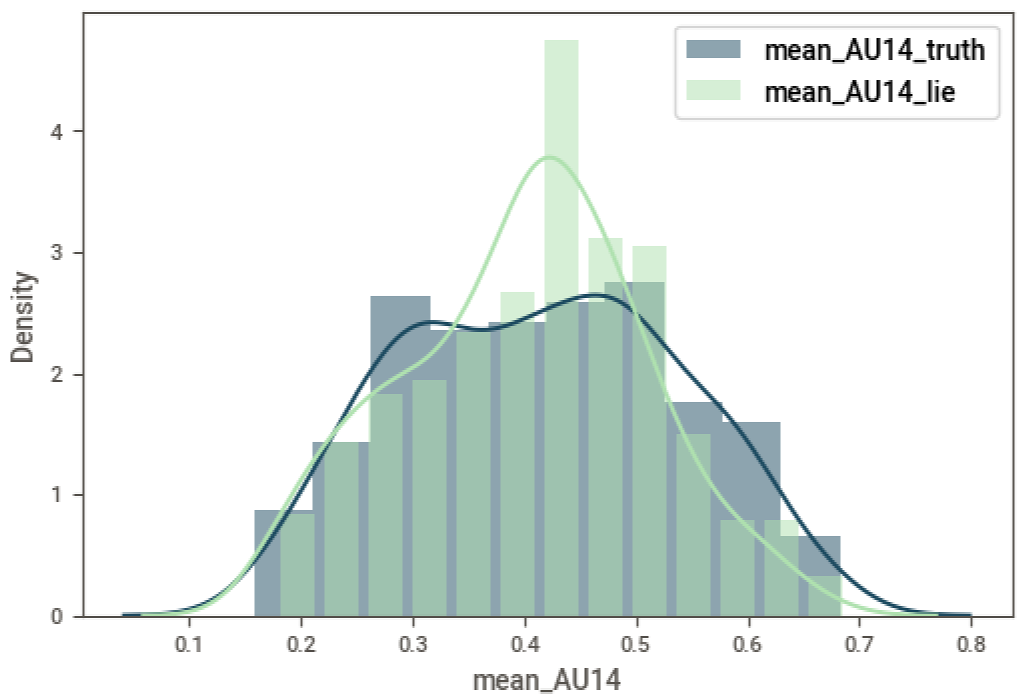


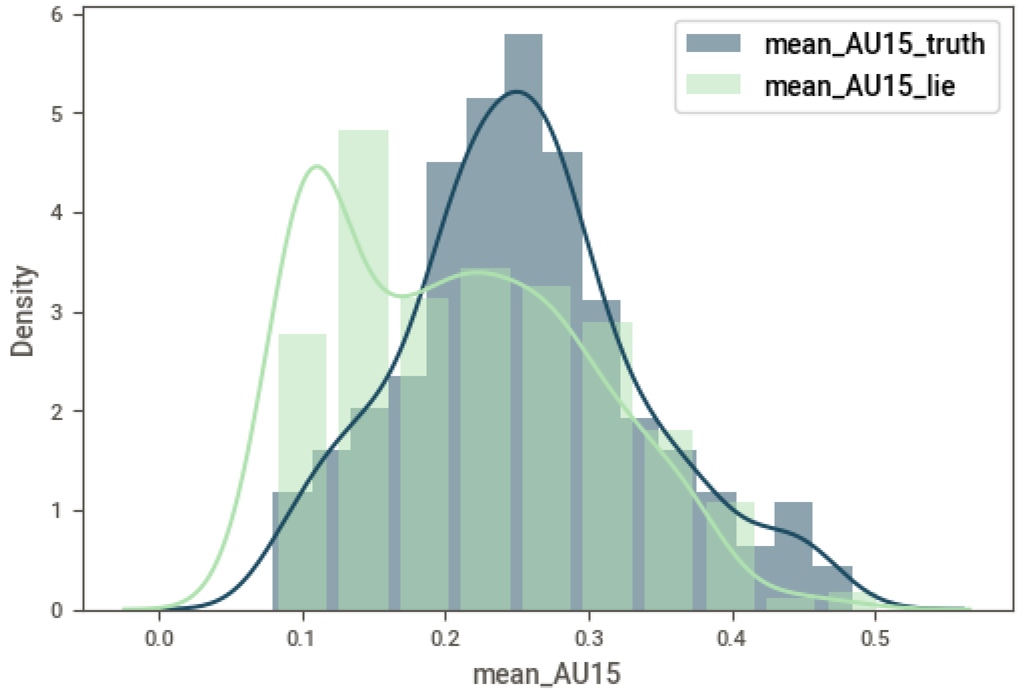
mean_AU10_truth
mean_AU10_lie
Density
mean_AU10

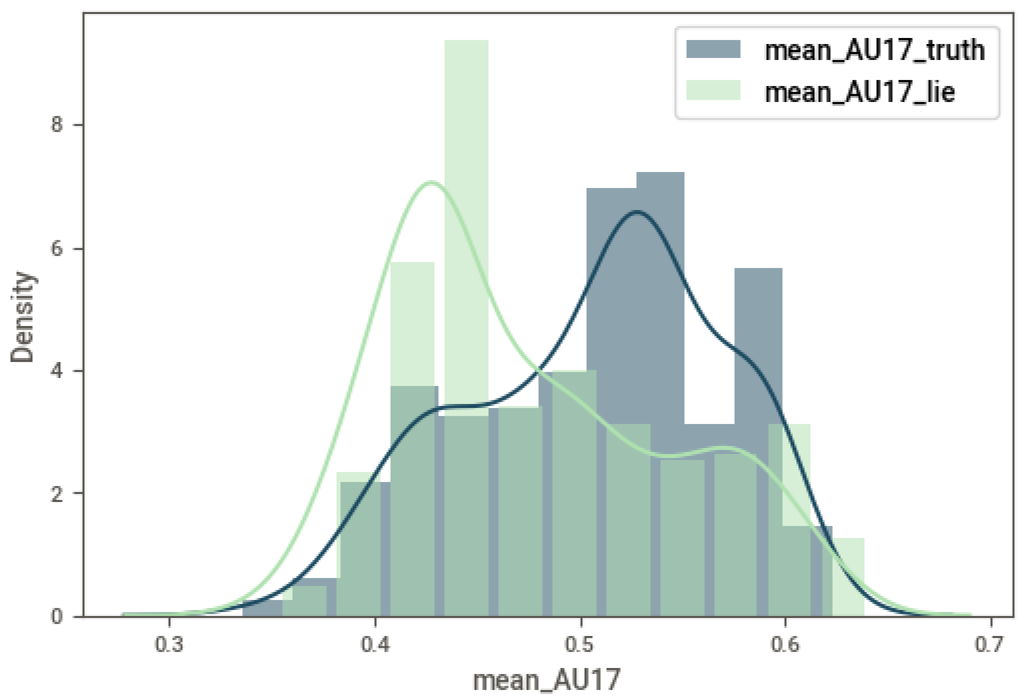


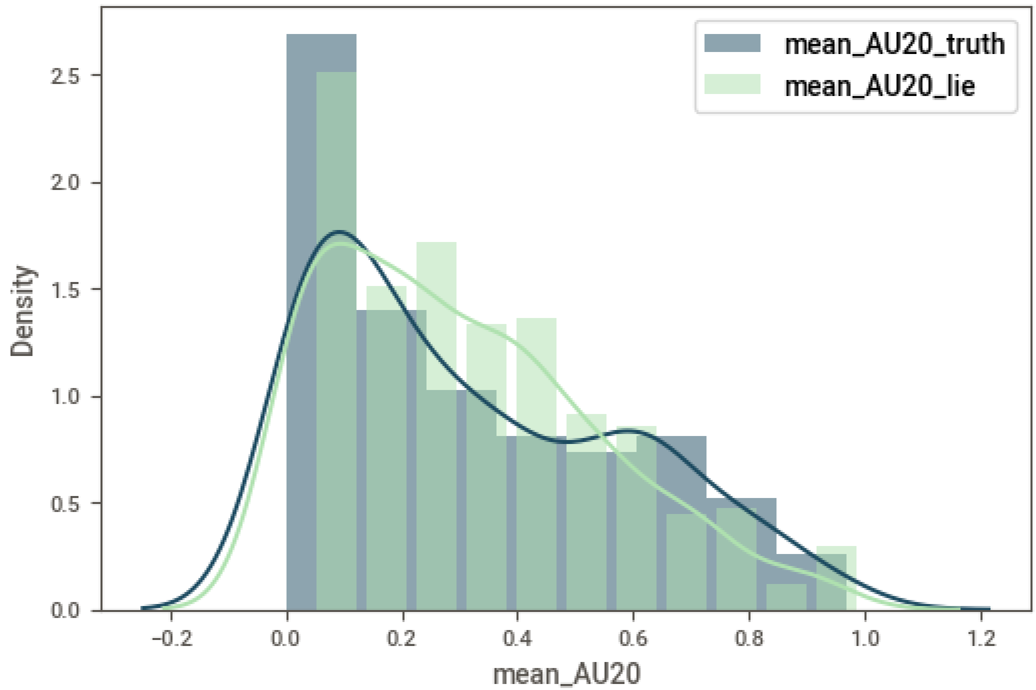
mean_AU11_truth
mean_AU11_lie
Density
mean_AU11

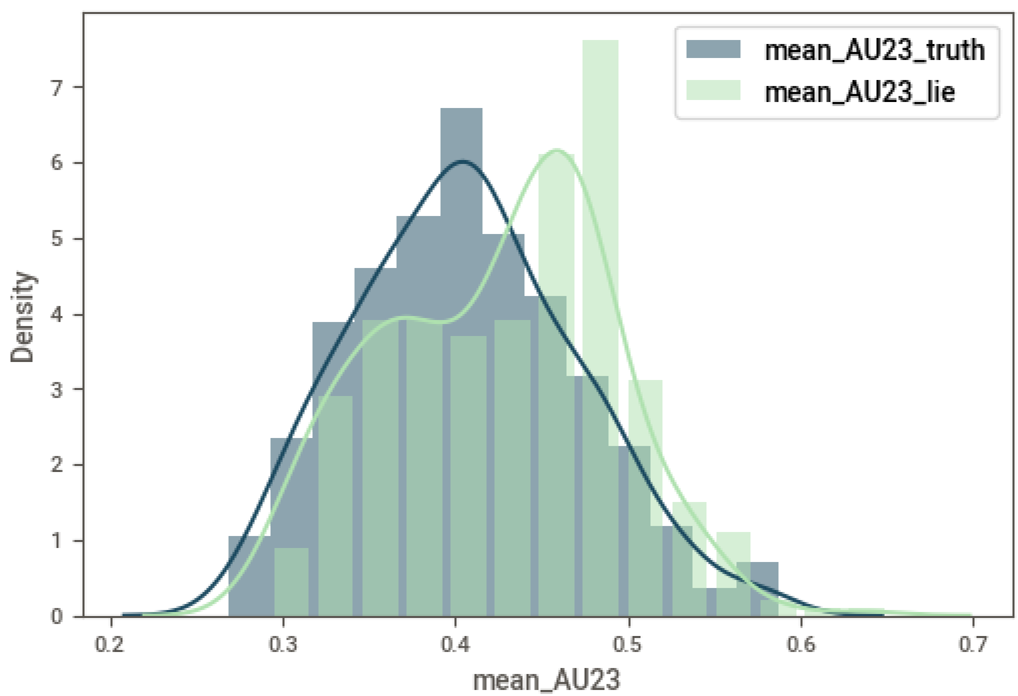

mean_AU12_truth
mean_AU12_lie
Density
mean_AU12
0
1
2
3
4
5
6
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7

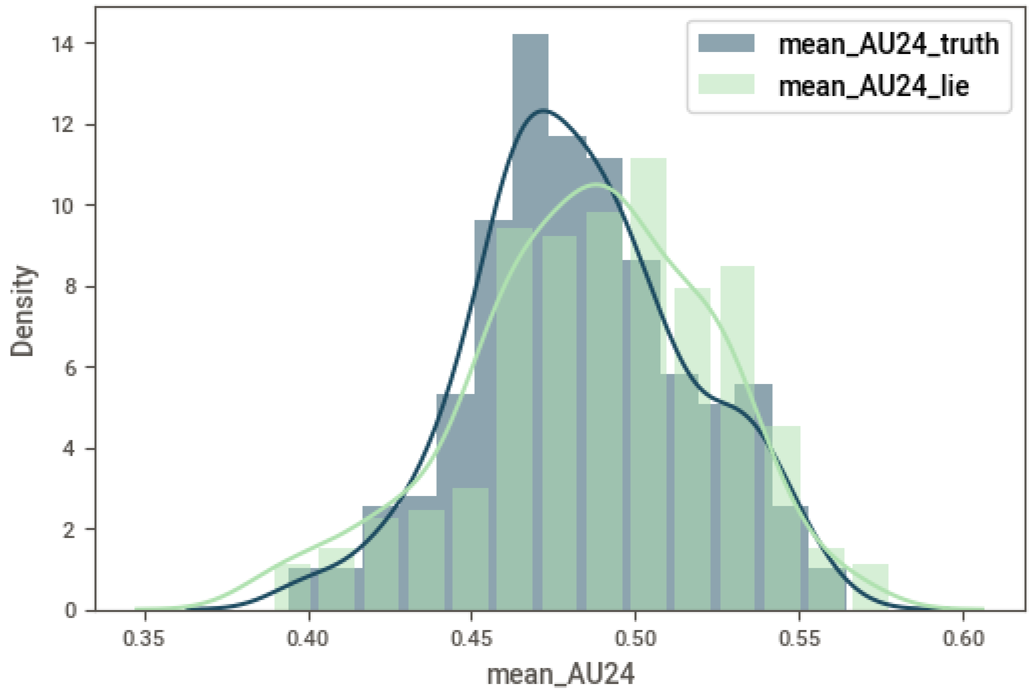

mean_AU14_truth
mean_AU14_lie
Density
mean_AU14
0
1
2
3
4
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8


mean_AU15_truth
mean_AU15_lie
Density
mean_AU15

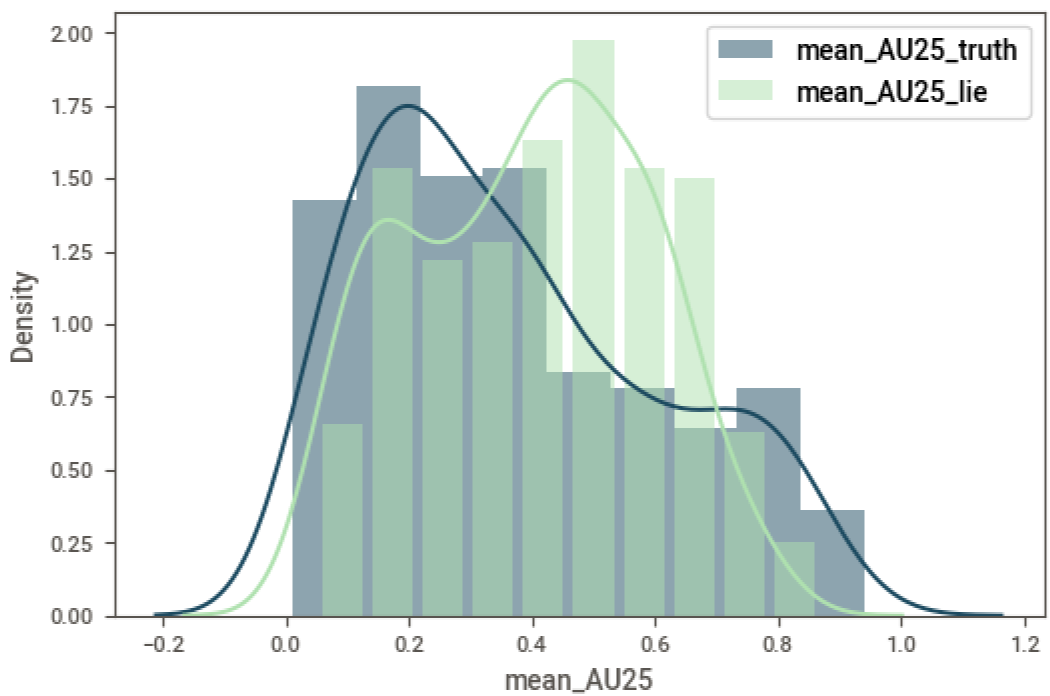

mean_AU17_truth
mean_AU17_lie
Density
mean_AU17


mean_AU20_truth
mean_AU20_lie
Density
mean_AU20

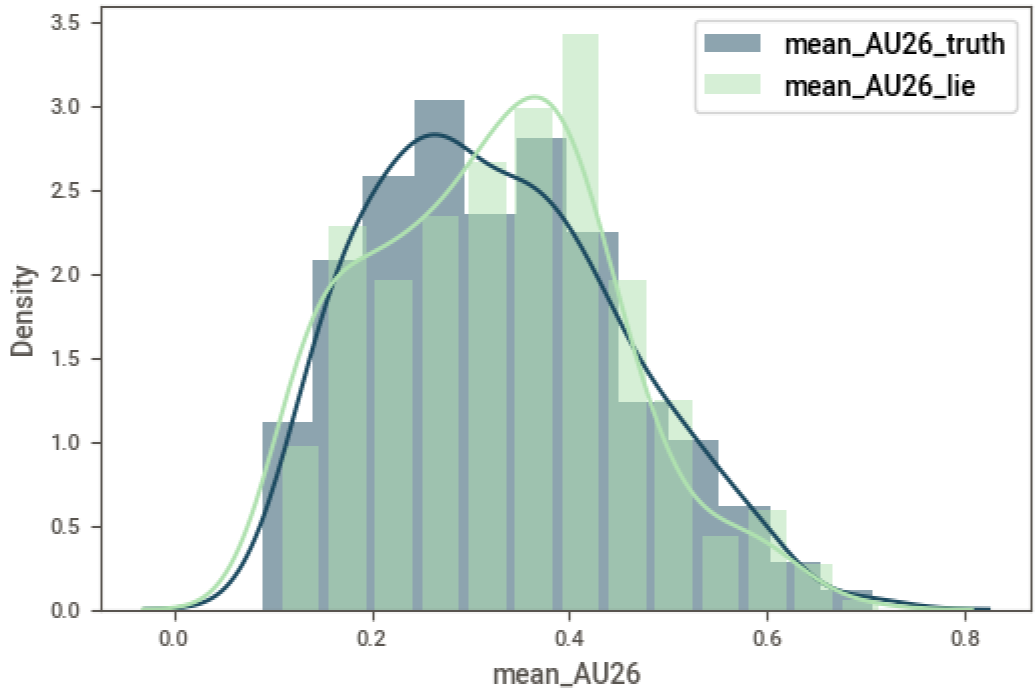

mean_AU23_truth
mean_AU23_lie
Density
mean_AU23

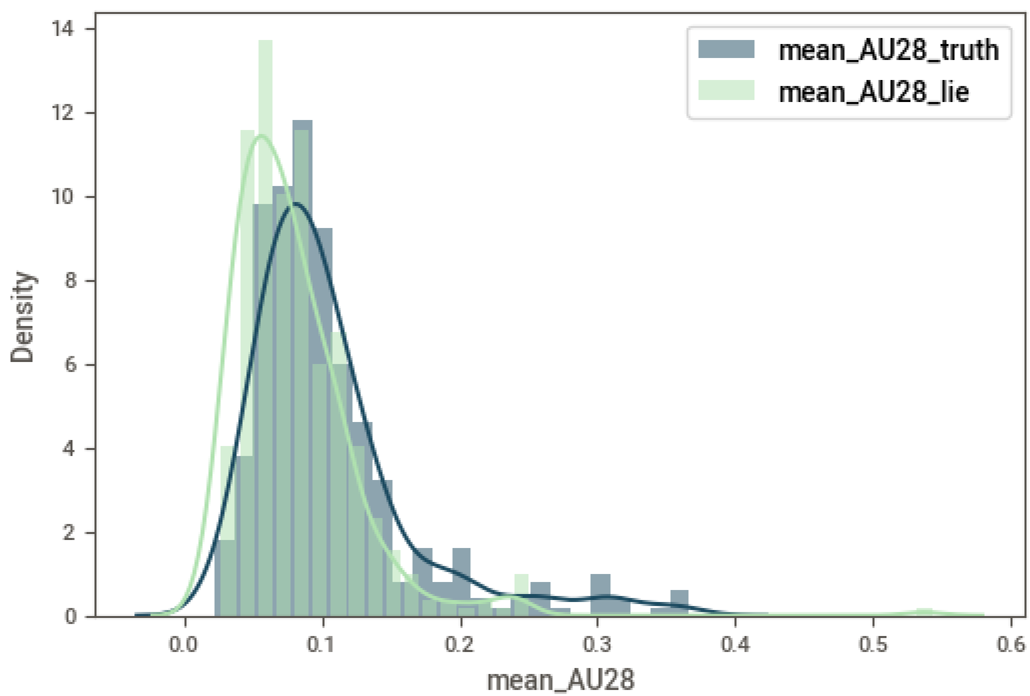


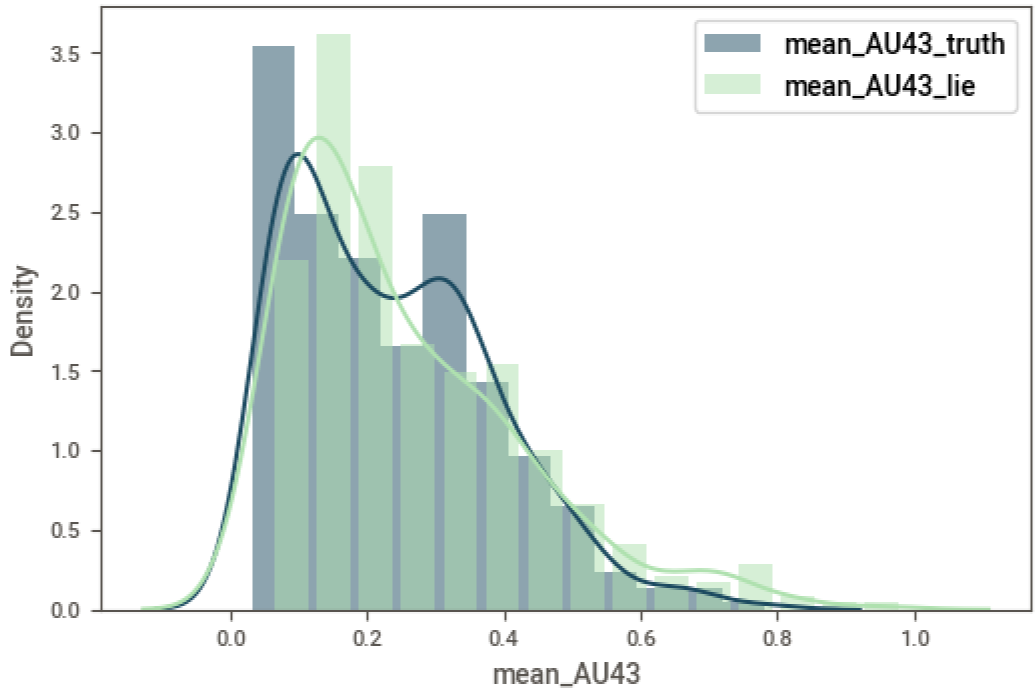
mean_AU24_truth
mean_AU24_lie
Density
mean_AU24
0
2
4
6
8
10
12
14
0.35
0.40
0.45
0.50
0.55
0.60

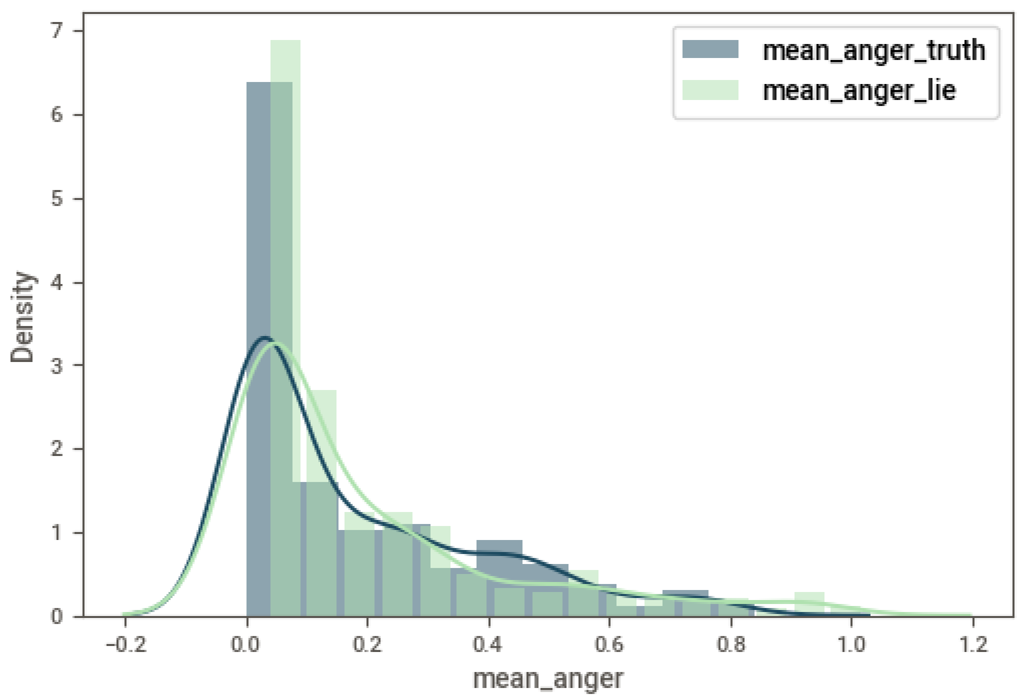


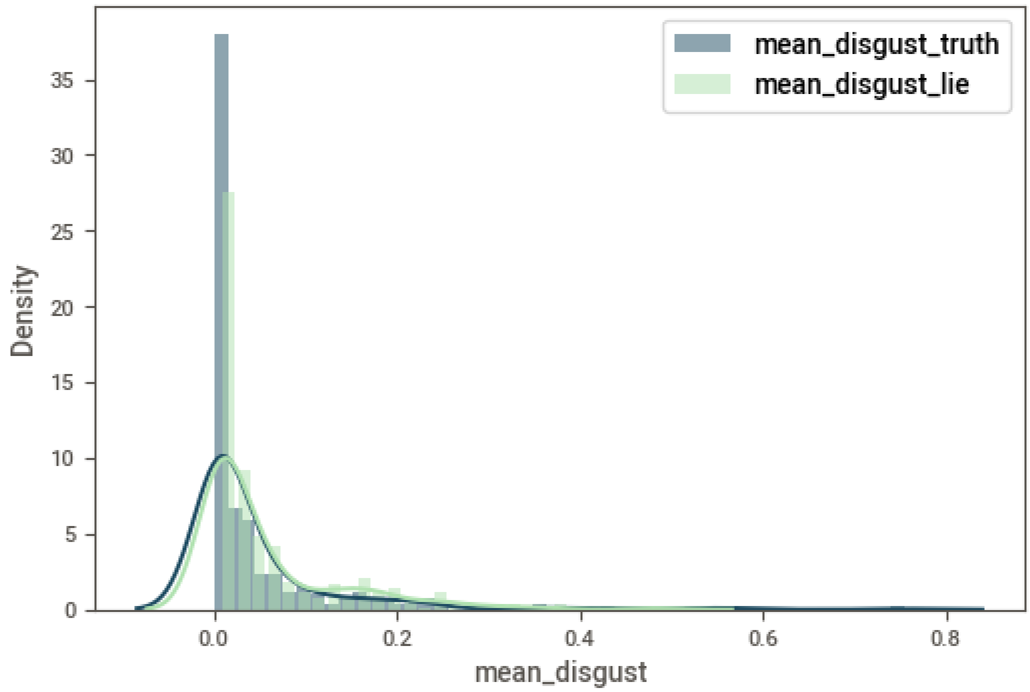
mean_AU25_truth
mean_AU25_lie
Density
mean_AU25
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2

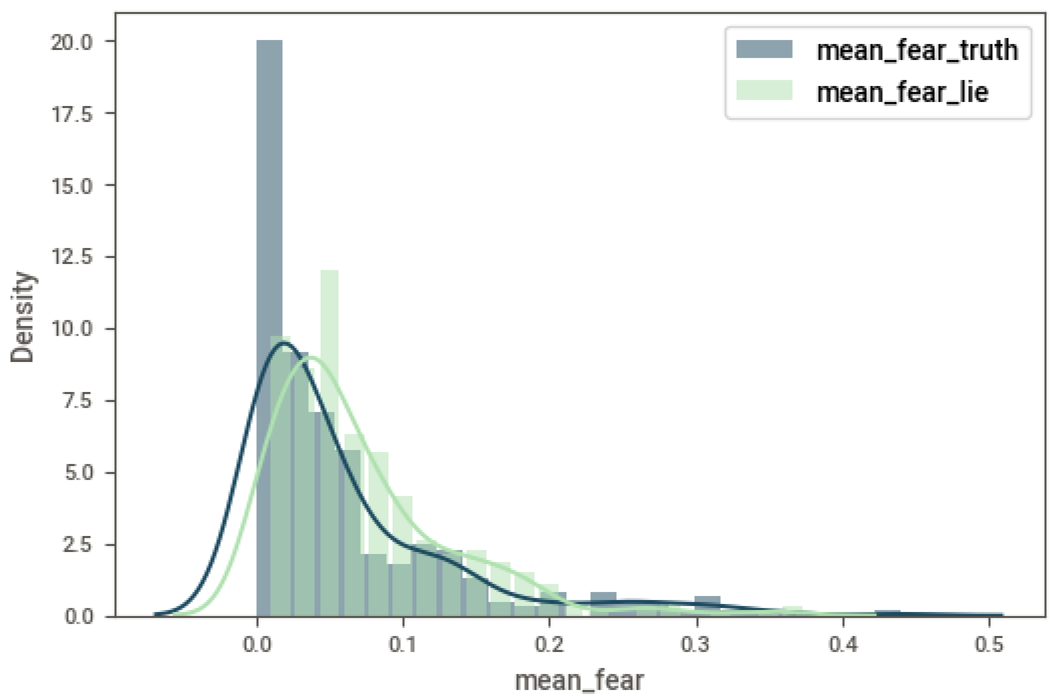

mean_AU26_truth
mean_AU26_lie
Density
mean_AU26
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0.2
0.4
0.6
0.8

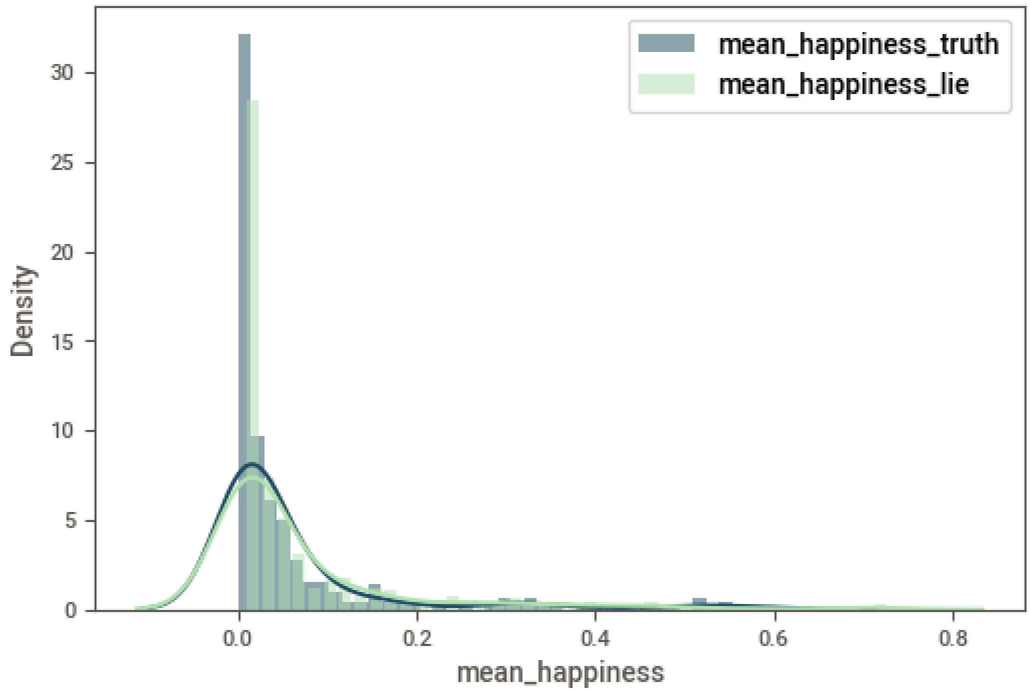

mean_AU28_truth
mean_AU28_lie
Density
mean_AU28
0
2
4
6
8
10
12
14
0.0
0.1
0.2
0.3
0.4
0.5
0.6

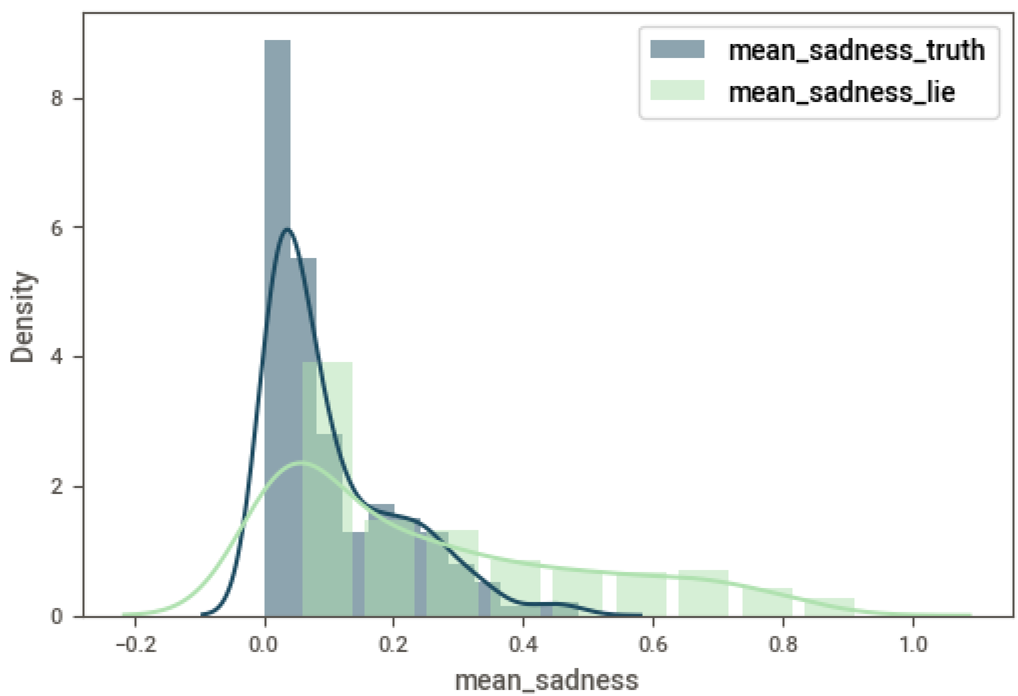

mean_AU43_truth
mean_AU43_lie
Density
mean_AU43

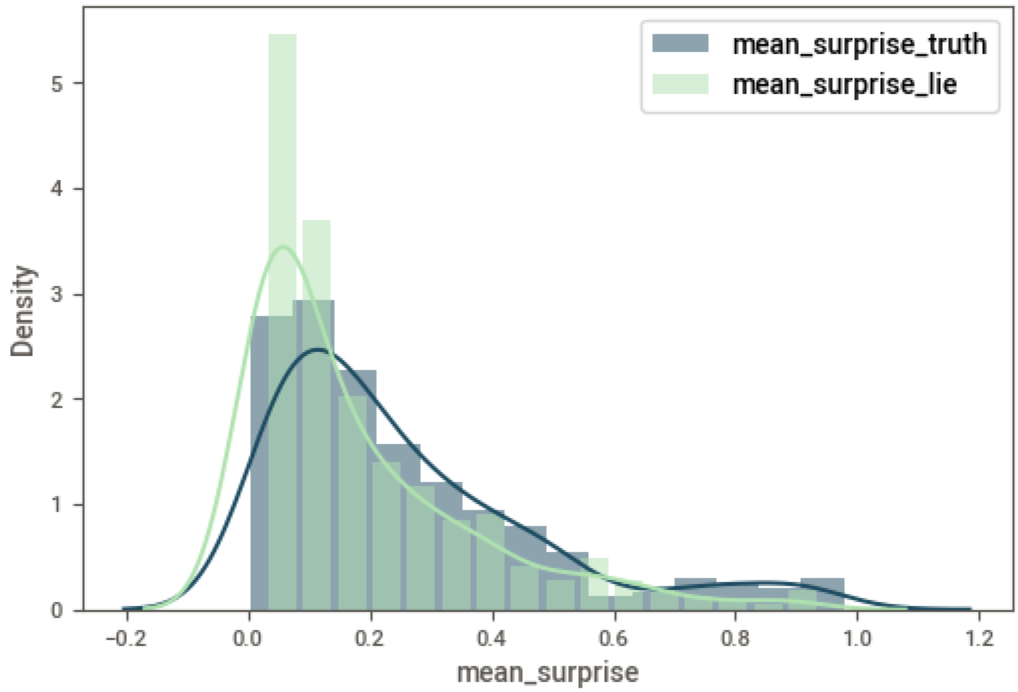

mean_anger_truth
mean_anger_lie
Density
mean_anger

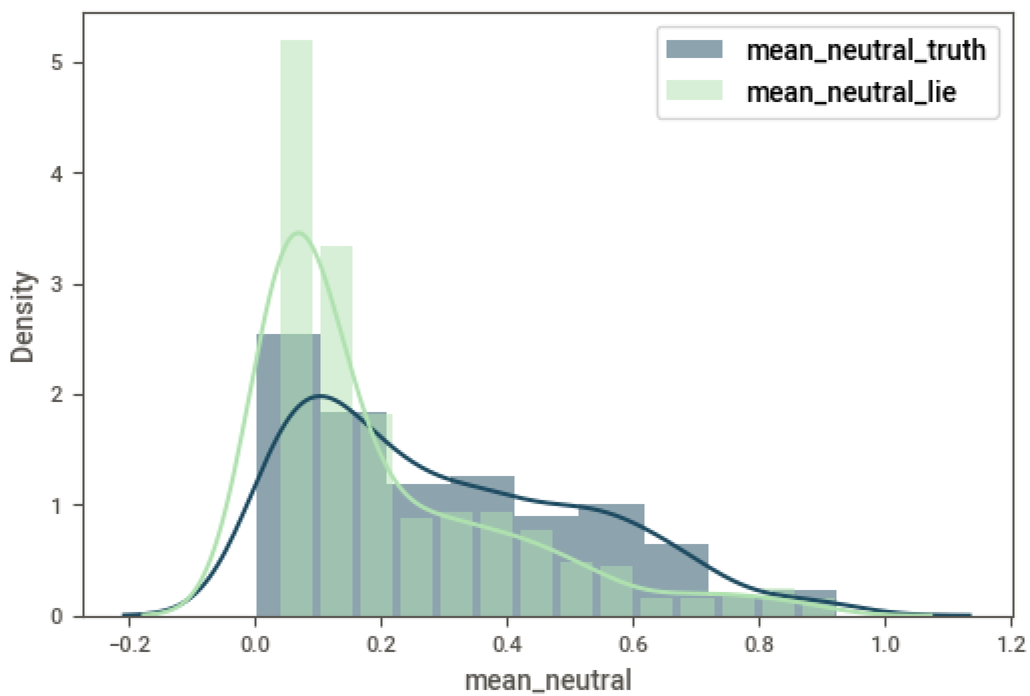

mean_disgust_truth
mean_disgust_lie
Density
mean_disgust
0
5
10
15
20
25
30
35
0.0
0.2
0.4
0.6
0.8


mean_fear_truth
mean_fear_lie
Density
mean_fear
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0.0
0.1
0.2
0.3
0.4
0.5


mean_happiness_truth
mean_happiness_lie
Density
0
5
10
15
20
25
30
0.0
0.2
0.4
0.6
0.8
mean_happiness


mean_sadness_truth
mean_sadness_lie
Density
0
2
4
6
8
-0.2
0.0
0.2
0.4
0.6
0.8
1.0
mean_sadness

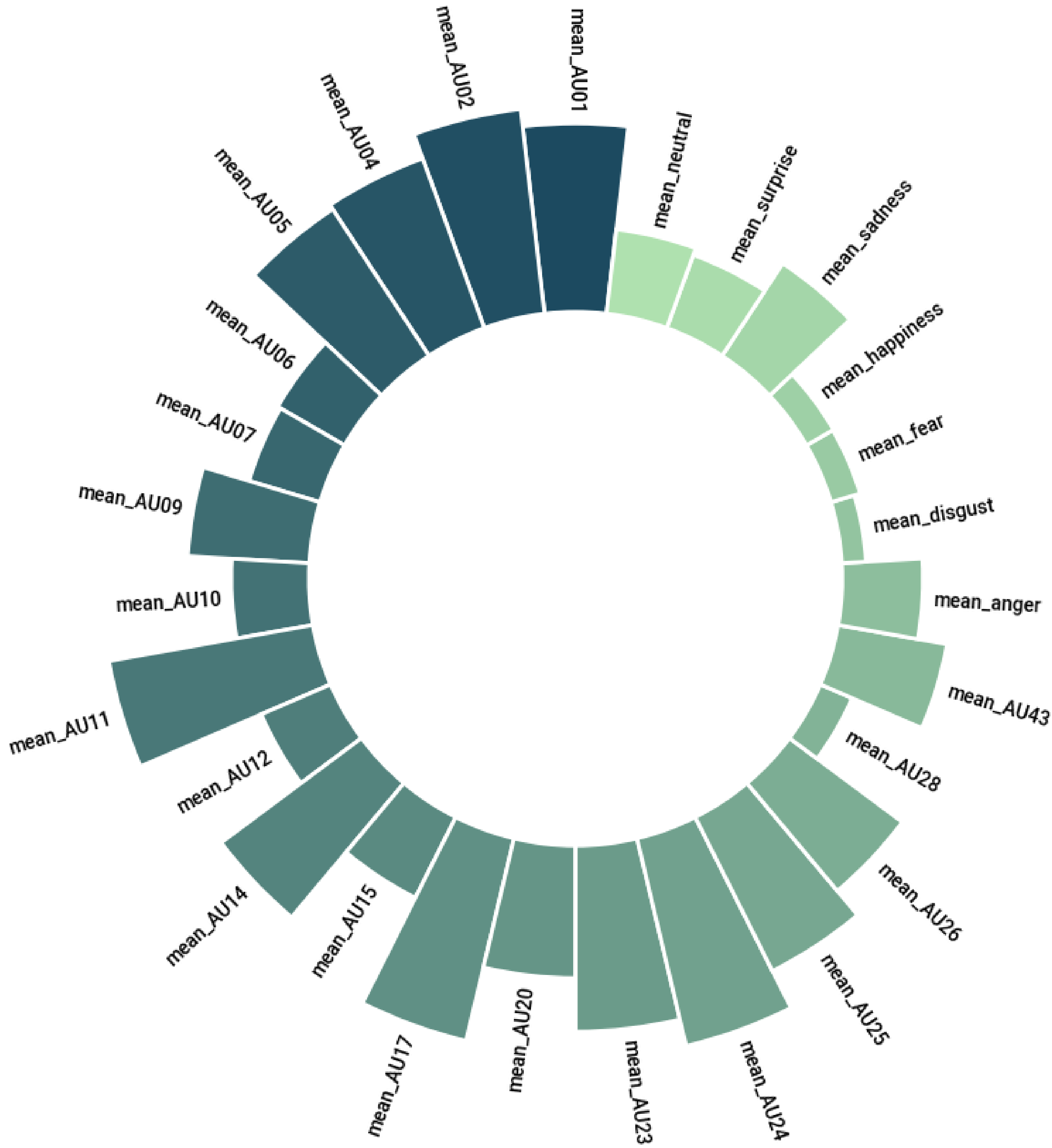




Let us now see the variations of various parameters when the label is lie.

In [18]:

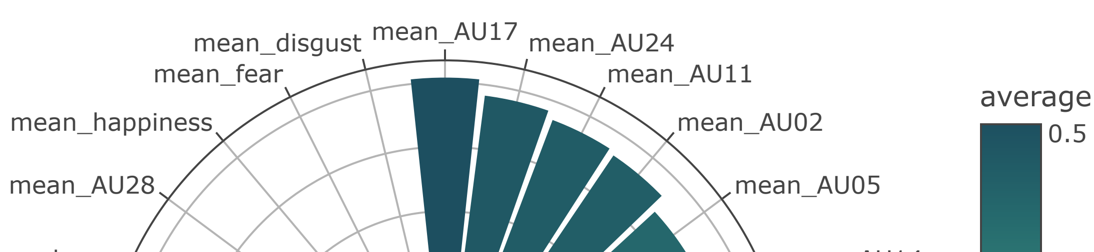
```
lie_data = df2.mean().to_frame(name='average')
lie_df = lie_data.reset_index()
PlotGraphs.CircularBarPlot(lie_df,1,0)
```

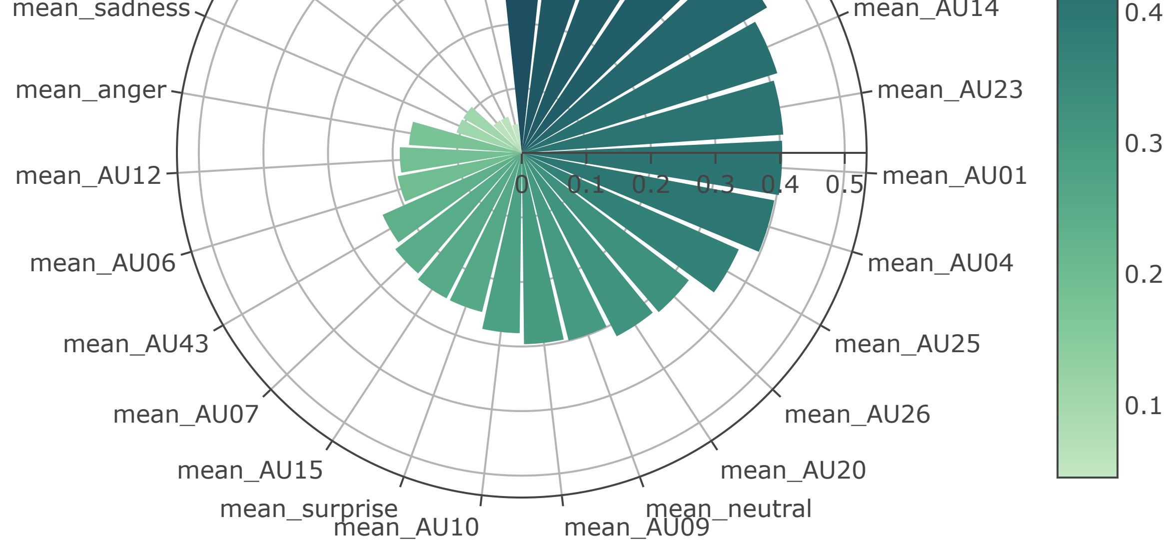


Let us now see the variations of various parameters when the label is lie.

In [19]:
```
tru_data = df1.mean().to_frame(name='average')
tru_df = tru_data.reset_index()
PlotGraphs.PolarBarPlot(tru_df,1,0,"True Label Distribution")
```

True Label Distribution





In [20]:

```
tru_df['Label'] = "Truth"
lie_df['Label'] = "Lie"
concatenated_df = pd.concat([tru_df, lie_df])
PlotGraphs.SliderBarPlot(concatenated_df,1,0,2,"Truth v/s Lie")
```