In [1]:
```
import pandas as pd
from statsmodels.graphics.gofplots import qqplot
from sklearn import preprocessing
import copy
import math
from plotnine import *
from plotnine.data import *
import numpy as np
from scipy import stats
import seaborn as sns
from statsmodels.graphics.gofplots import qqplot
import matplotlib. pyplot as plt
from sklearn.preprocessing import StandardScaler
from combat.pycombat import pycombat
from numpy.random import seed
from numpy.random import randn
from scipy.stats import shapiro
from scipy.stats import normaltest
```

In [2]:
```
# Load the data with predictors
data_g = pd.read_excel("predictors.xlsx")
data_g.head()
```

Out[2]:

	SampleName	Plate	G2	G3	G4	G5	G6	G7	G8	G9	...	G28	G29	G30	G31	G32	G33
0	10300	15	2197756	855165	25259	1841249	968146	680418	326888	324603	...	809505	105068	222322	823629	78362	125237
1	10399	16	4852317	1604861	39573	2137690	911499	902705	460655	503297	...	2041372	240008	579176	2160481	186241	295631
2	10667	12	2457826	1596773	77197	2214044	1161974	1084932	963856	1153901	...	4197764	400858	1609727	2321895	301444	303239
3	10786	8	3402185	1294306	84166	3404739	1806373	1172932	583699	630740	...	1967106	214824	521280	1222482	300825	176710
4	10849	11	2571050	1241892	29052	2080235	1165996	939341	465035	577505	...	2395528	180610	874219	914291	134145	100390

5 rows × 38 columns

In [3]:
```
# remove wrong measurements
data_g = data_g[~data_g.SampleName.str.contains("st", na=False)]
```

In [4]:
```
# to transform predictors data we need to separate predictors from 2 fisrt descriptive colunmns
```

```
# Text_c - first 2 columns
Text_c = copy.deepcopy(data_g.iloc[:,0:2])
Text_c.head(5)
```

Out[4]:

	SampleName	Plate
0	10300	15
1	10399	16
2	10667	12
3	10786	8
4	10849	11

In [5]:

```
# Pred_c - columns with predictors
Pred_c = copy.deepcopy(data_g.iloc[:,2:38])
Pred_c.head(5)
```

Out[5]:

	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	...	G28	G29	G30	G31	G32	G33	
0	2197756	855165	25259	1841249	968146	680418	326888	324603	36343	1689348	...	809505	105068	222322	823629	78362	125237	7
1	4852317	1604861	39573	2137690	911499	902705	460655	503297	49946	1079935	...	2041372	240008	579176	2160481	186241	295631	22
2	2457826	1596773	77197	2214044	1161974	1084932	963856	1153901	78561	2452196	...	4197764	400858	1609727	2321895	301444	303239	37
3	3402185	1294306	84166	3404739	1806373	1172932	583699	630740	147203	2539198	...	1967106	214824	521280	1222482	300825	176710	15
4	2571050	1241892	29052	2080235	1165996	939341	465035	577505	54722	2157900	...	2395528	180610	874219	914291	134145	100390	16

5 rows × 36 columns

In [6]:

```
# I found that in previous papers the authors log transformed the data so I did this as well
temp_df = copy.deepcopy(Pred_c)
Pred_c_log10 = temp_df.iloc[:,0:36].applymap(lambda x: np.log10(x))
Pred_c_log10_text = pd.concat([Text_c, Pred_c_log10], axis=1, join='inner')
```

In [7]:

```
# As data are collected from different plates, we need to remove the effect of the plates - to reomve the batch effect
batch = Text_c.iloc[:,1]
batch
```

```
log_transposed = Pred_c_log10.transpose()
log_transposed.head(5)

# I used polycombat package to remove batch effect
data_g_1_log_batch_corr = pycombat(log_transposed,batch)

data_g_1_log_batch_corr = data_g_1_log_batch_corr.transpose()
data_g_1_log_batch_corr.head(5)
```

```
Found 16 batches.
Adjusting for 0 covariate(s) or covariate level(s).
Standardizing Data across genes.
Fitting L/S model and finding priors.
Finding parametric adjustments.
Adjusting the Data
C:\Users\evgeny\AppData\Local\Continuum\envs\tf2\lib\site-packages\numpy\core\_asarray.py:83: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray
```

Out[7]:

	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	...	G28	G29	G30	G31	G32
0	6.386773	6.015459	4.496841	6.334647	6.008568	5.894425	5.634879	5.666346	4.643783	6.319075	...	6.067696	5.190251	5.531833	6.068986	5.025978
1	6.634286	6.155799	4.496236	6.226663	5.879062	5.897634	5.564438	5.592347	4.671658	5.922881	...	6.143065	5.248149	5.566835	6.258787	5.181679
2	6.244373	6.133075	4.816569	6.232808	5.962691	5.978317	5.916264	5.990172	4.903062	6.309131	...	6.557189	5.566338	6.152671	6.282870	5.490366
3	6.557118	6.130495	4.936691	6.562325	6.289670	6.087507	5.769441	5.801973	5.156553	6.430226	...	6.323636	5.353423	5.747778	6.093832	5.449154
4	6.458688	6.134147	4.511775	6.365589	6.111720	6.012748	5.718986	5.782920	4.745351	6.377328	...	6.449561	5.317508	6.011964	6.037466	5.183945

5 rows × 36 columns

In [8]:

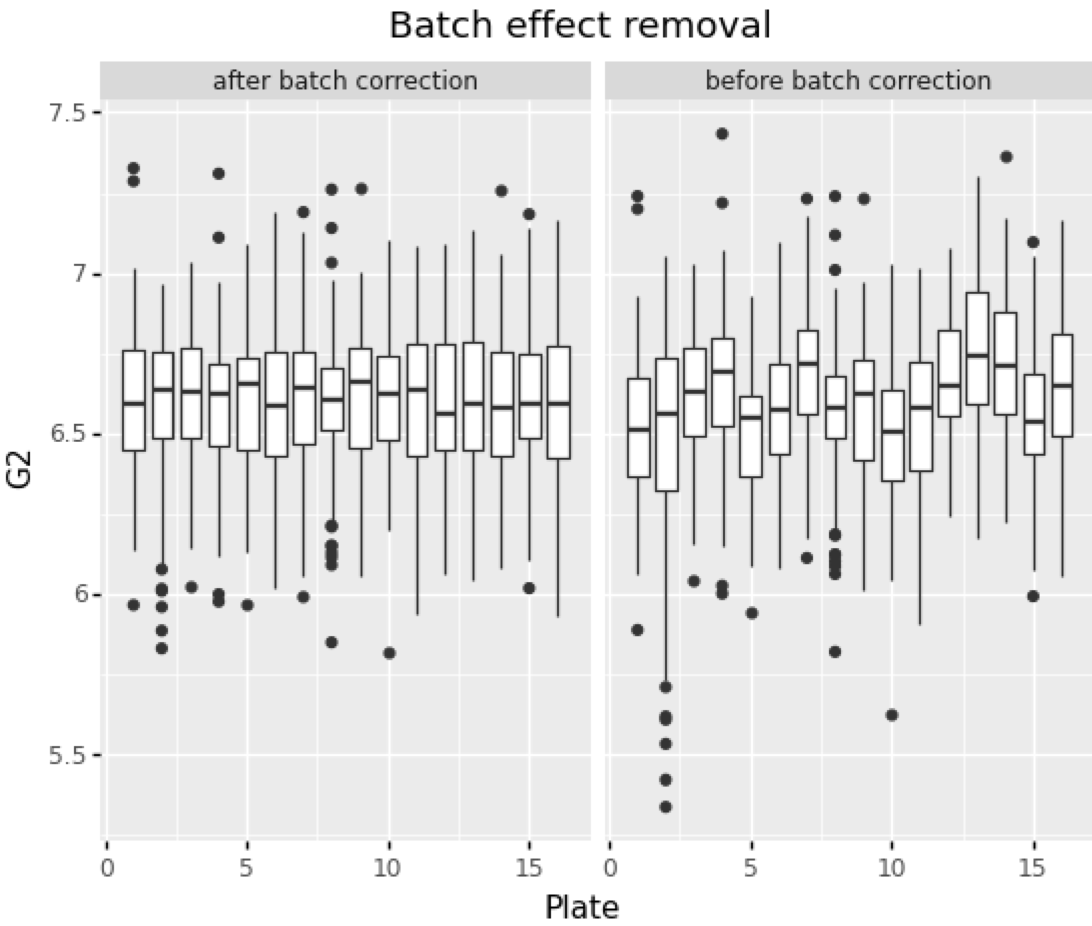
```
# Batch normalisation can sometimes give negative values so I checked for that
# check if there are negative values in predictor - we will see that there is no negative values among batch corrected data
d_neg_values = pd.DataFrame(np.zeros((1381, 36)))
for i in range(0,36):
    d_neg_values.iloc[:,i] = data_g_1_log_batch_corr.iloc[:,i].apply(lambda x: 1 if x < 0 else 0)
d_neg_values["sum"] = d_neg_values[list(d_neg_values.columns.values)].sum(axis=1)
d_neg_values["sum"].sum()
```

Out[8]: 0

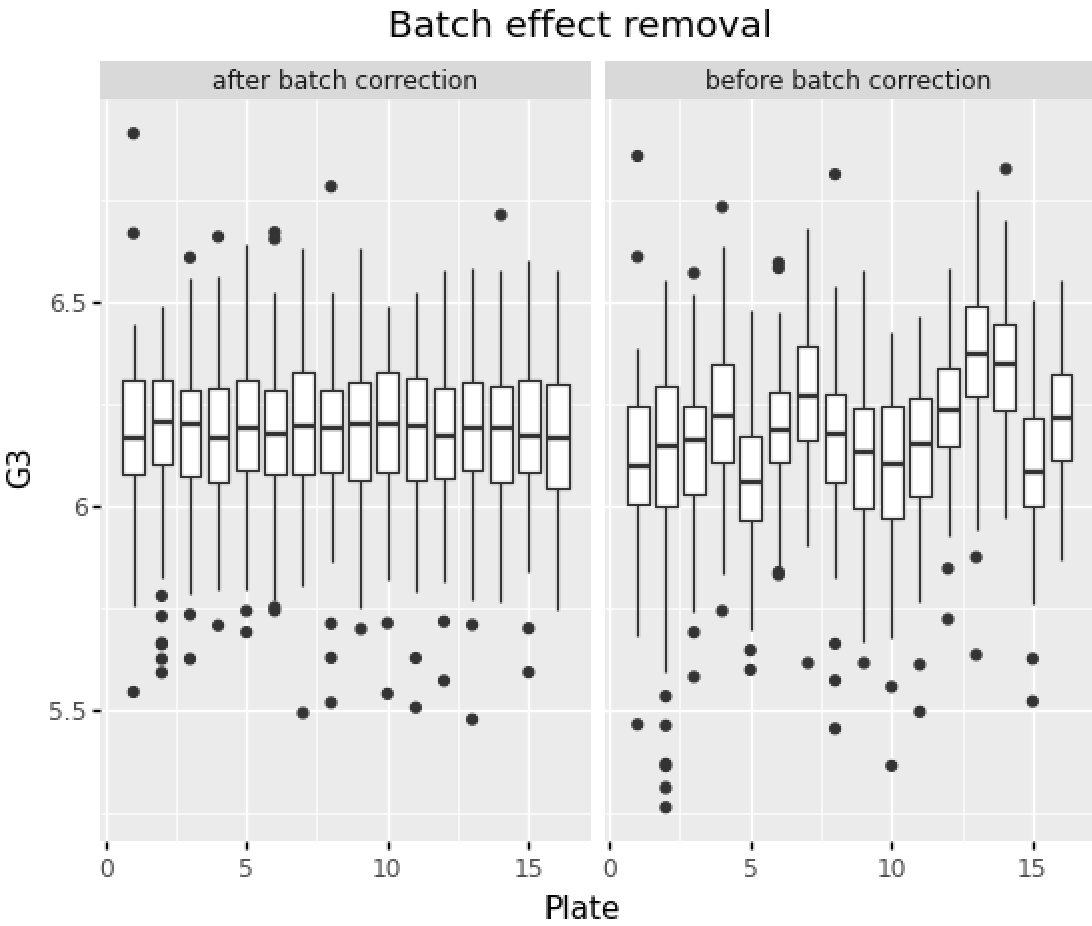
In [11]:

```
# concatenate batch corrected data with SampleName, repeat and Plate columns
```

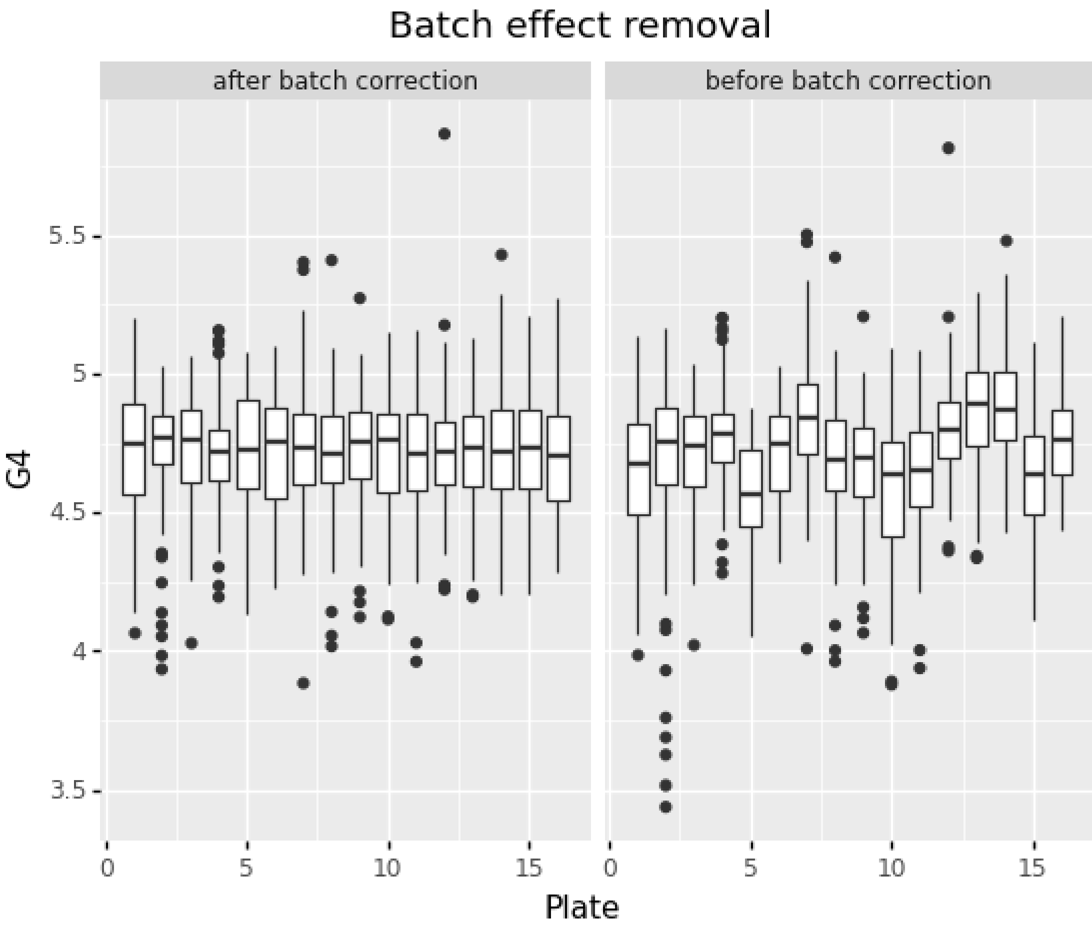
```
data_g_1_log_batch_corr_text = pd.concat([Text_c, data_g_1_log_batch_corr], axis=1, join='inner')
data_g_1_log_batch_corr_text.head(5)

Pred_c_log10_text['df'] = 'before batch correction'
data_g_1_log_batch_corr_text['df'] = 'after batch correction'
visual_df_log = pd.concat([Pred_c_log10_text, data_g_1_log_batch_corr_text], axis=0)

# set the list with predictor names to iterate over them
P_col_list = ['G2','G3','G4','G5','G6','G7','G8','G9','G10',
 'G11','G12','G13','G14','G15','G16','G17','G18','G19','G20','G21','G22','G23',
 'G24','G25','G26','G27','G28','G29','G30','G31','G32','G33','G34','G35',
 'G36','G37']


# Plot the results before and after batch correction (for the first 4 predictors)
for i in range(0,4):
    g = ggplot(visual_df_log, aes(x='Plate', y=P_col_list[i]))\
    + geom_boxplot(aes(group = 'Plate')) + labs(title="Batch effect removal")\
    + facet_wrap('~df')
    print(g)

# You see that that batch normalisation removed the effect of plates for each individual predictor
```



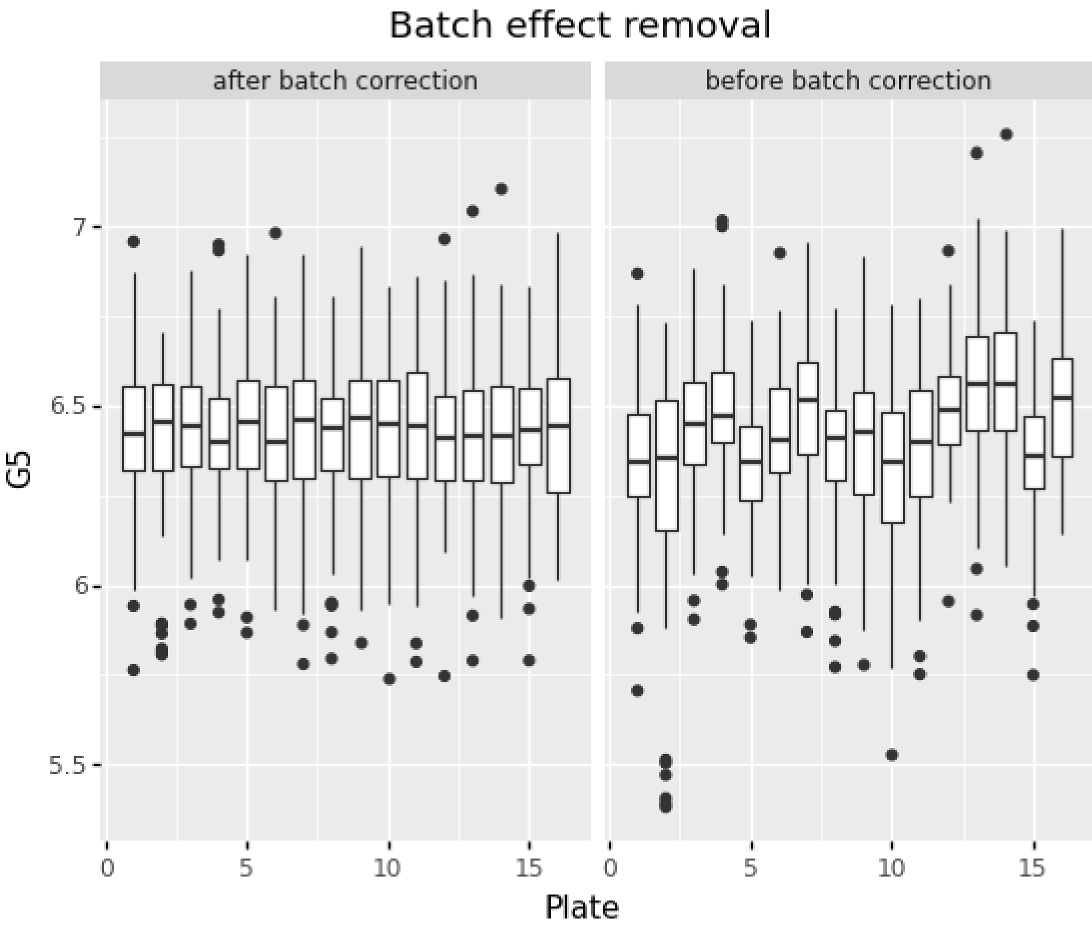

```
<ggplot: (-9223371893429551788)>
```




```
<ggplot: (-9223371893429503572)>
```



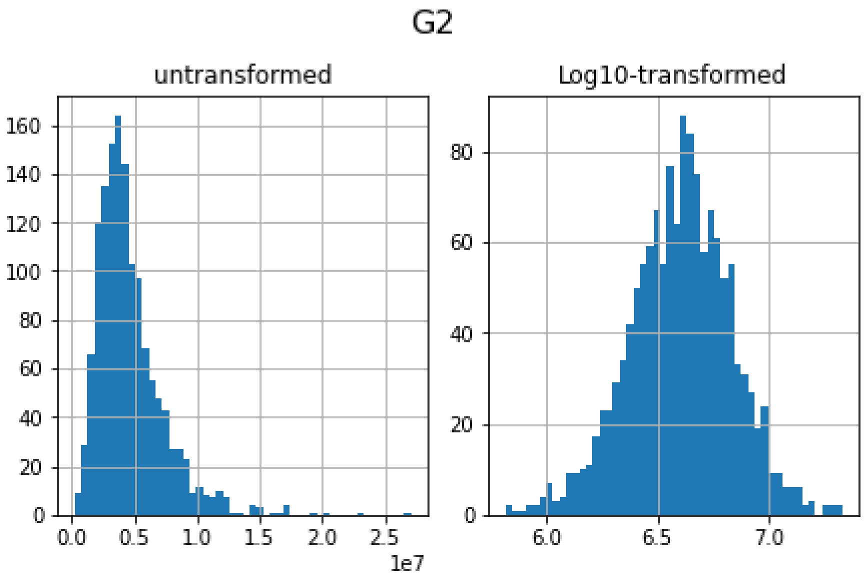
<ggplot: (-9223371893427239872)>

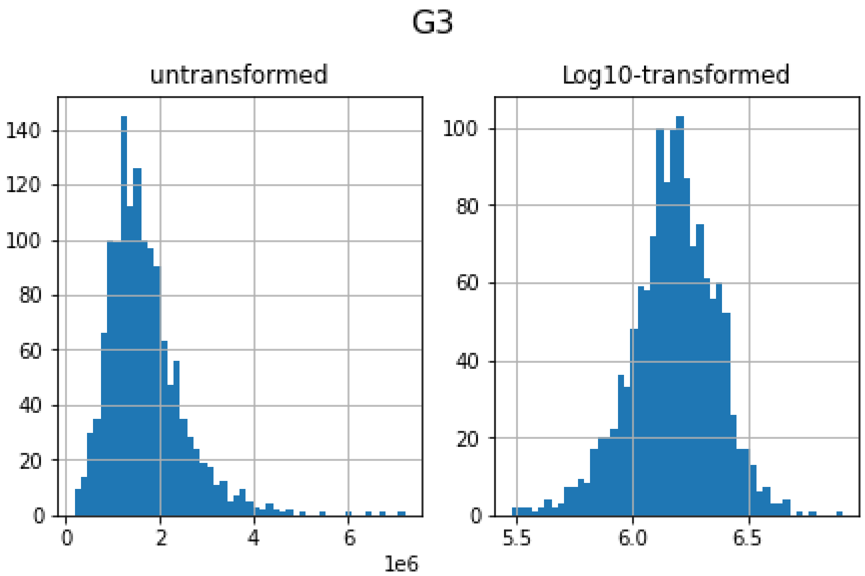


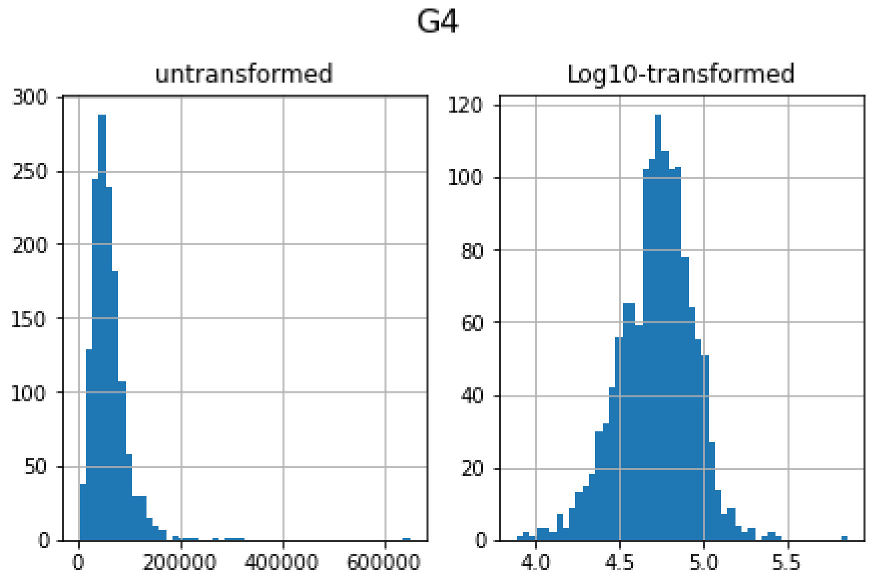
<ggplot: (-9223371893429359780)>

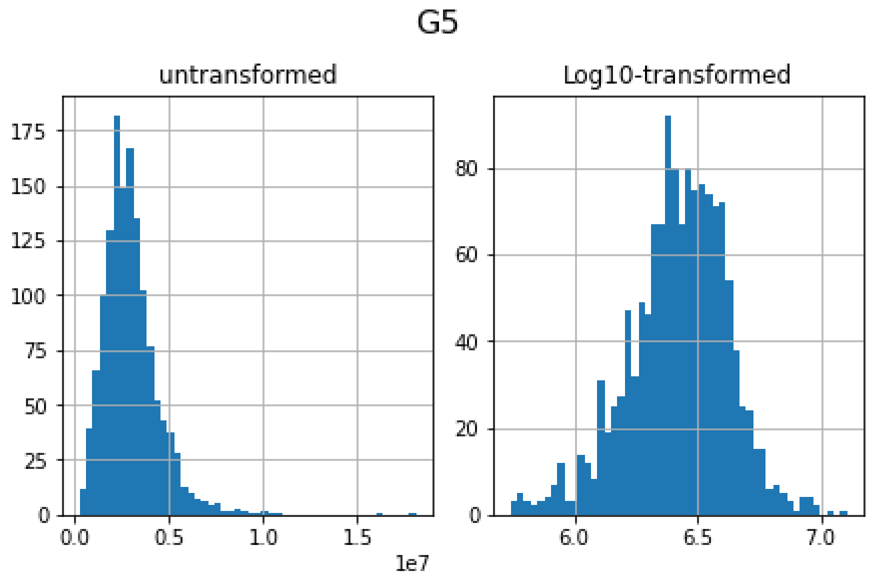
In [12]:
```
# See the effect of log transformation
# most of the data are left scewed. Log tranformation will be applied to improve distributions

# Compare untrasformed (left) and trasnformed (right) distributions  (for the first 4 predictors)
for i in range(0,4):
    fig, ax = plt.subplots(1, 2, constrained_layout=True)
    fig.suptitle(P_col_list[i], fontsize=16)
    Pred_c.iloc[:,i].hist(bins= 50, ax = ax[0])
    ax[0].set_title('untransformed')

    data_g_1_log_batch_corr.iloc[:,i].hist(bins= 50, ax = ax[1])
```

```
    ax[1].set_title('Log10-transformed')
    plt.show()
```









In [13]:
```
### averaging  over duplicates and triplicates
data_log_batch_corr_dup = copy.deepcopy(data_g_1_log_batch_corr_text)
```

In [14]:
```
# some Sample names are replicates and have "D" after the number.
# Here is a function to remove "D"
```

```
### Remove the last char if x is str
def f2(x):
    if isinstance(x,str) == True:
        a = int(x[:-1])
    else: a = x
    return a
data_log_batch_corr_dup['SampleName'] = data_log_batch_corr_dup['SampleName'].map(f2)
```

In [15]:

```
# create column "same" which will contain labels for rows with the same sampleName
data_log_batch_corr_dup = data_log_batch_corr_dup.sort_values(by=['SampleName'], ascending = [False])
data_log_batch_corr_dup['same'] = ''
new_c_order2 = ['SampleName', 'Plate','same',
 'G2','G3','G4','G5','G6','G7','G8','G9','G10',
 'G11','G12','G13','G14','G15','G16','G17','G18','G19','G20','G21','G22','G23',
 'G24','G25','G26','G27','G28','G29','G30','G31','G32','G33','G34','G35',
 'G36','G37']
data_log_batch_corr_dup = data_log_batch_corr_dup[new_c_order2]
data_log_batch_corr_dup.head()
```

Out[15]:

	SampleName	Plate	same	G2	G3	G4	G5	G6	G7	G8	...	G28	G29	G30	G31	G3
1380	971052	4		6.796732	6.411178	4.751047	6.729656	6.376118	6.261358	5.932363	...	6.596052	5.650274	6.060064	6.183815	5.55766
1257	395687	14		6.559553	6.137243	4.717146	6.392535	6.139847	5.988360	5.809461	...	6.438316	5.573447	6.030736	6.430253	5.57170
1256	395591	8		6.608899	6.195572	4.697063	6.595241	6.195734	6.021199	5.890090	...	6.731892	5.649014	6.113155	5.628393	5.49189
1255	395568	1		6.465452	6.049378	4.614538	6.423343	6.102366	5.983215	5.836081	...	6.688642	5.580329	6.037265	6.277707	5.39358
1254	395535	10		6.616280	6.187048	4.949364	6.566102	6.213090	5.871194	5.694943	...	6.477721	5.487084	5.907416	6.066836	5.30427

5 rows × 39 columns

In [16]:

```
# Fill columns "Same"
data_log_batch_corr_dup.iloc[0,2] = 0
c=0
for i in range(1, data_log_batch_corr_dup.shape[0]):
        if data_log_batch_corr_dup.iloc[i-1,0] == data_log_batch_corr_dup.iloc[i,0]:
            data_log_batch_corr_dup.iloc[i,2] = c
        else:
            c += 1
            data_log_batch_corr_dup.iloc[i,2] = c
```

In [17]:
```
# Remove unnecessary column with Plates
data_log_batch_corr_dup = data_log_batch_corr_dup.drop(data_log_batch_corr_dup.columns[[1]], axis=1)
# Average rows with the same Sample Name
dup = copy.deepcopy(data_log_batch_corr_dup)
dup2 = dup[dup.duplicated('SampleName', keep=False)]
dup_s = (dup.groupby((dup.same != dup.same.shift()).cumsum())
                  .mean()
                  .reset_index(drop=True))
```

In [18]:
```
# Scaling the data
s_c = copy.deepcopy(dup_s.iloc[:,1:37])
s_t = copy.deepcopy(dup_s.iloc[:,0])
scaler = StandardScaler()
scaled = scaler.fit_transform(s_c.to_numpy())
scaled_df = pd.DataFrame(data=scaled, columns=P_col_list)
# concatenate scaled data with SampleName, repeat and Plate columns
scaled_data = pd.concat([s_t, scaled_df], axis=1, join='inner')
scaled_data.head(5)

scaled_data.to_csv('predictors_processed.csv', index = False)
```

In [20]:
```
# Removing outliers
out_text = copy.deepcopy(scaled_data.iloc[:,0])
out_c = copy.deepcopy(scaled_data.iloc[:,1:37])
out_c.shape

# count outliers
out_matrix = pd.DataFrame(np.zeros((1262, 36)))
for i in range(0,36):
    q10 = np.quantile(out_c.iloc[:,i], 0.975)
    q90 = np.quantile(out_c.iloc[:,i], 0.025)
    out_matrix.iloc[:,i] = out_c.iloc[:,i].apply(lambda x: 1 if ((x >= q10) or (x <= q90)) else 0)

out_matrix["sum"] = out_matrix[list(out_matrix.columns.values)].sum(axis=1)
out_matrix["sum"].sum()
threashold = np.quantile(out_matrix["sum"], 0.95)

out = pd.concat([out_text, out_c], axis=1, join='inner')
out['sum_out'] = out_matrix['sum']

out_final = out[out['sum_out'] <= threashold ]
out_final.drop(['sum_out'], axis=1, inplace=True)
```

```
out_final.to_csv('predictors_processed_out_removed.csv', index = False)
```

```
C:\Users\evgeny\AppData\Local\Continuum\envs\tf2\lib\site-packages\pandas\core\frame.py:4170: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versu
s-a-copy
```

In [21]:

```
# Merge tables with patients data and processed predictors
data_p = pd.read_excel("patients.xlsx")
data_p.rename(columns={'Sample': 'SampleName'}, inplace=True)
# save table where predictor outliers were not removed
scaled_final_pat = pd.merge(scaled_data, data_p, on="SampleName")
scaled_final_pat.to_csv('predictors_processed_patients.csv', index = False)
# save table where predictor outliers were removed
out_final_pat = pd.merge(out_final, data_p, on="SampleName")
out_final_pat.to_csv('predictors_processed_out_removed_patients.csv', index = False)
```

In [22]:

```
# Assign labels to Age and Ethnicity columns for the dataset with removed outliers

out_patients = pd.read_csv('predictors_processed_out_removed_patients.csv')
out_patients['Ethnicity'] = out_patients['Ethnicity'].apply(lambda x: "Ethnic_group_1" if x == 1 else x)
out_patients['Ethnicity'] = out_patients['Ethnicity'].apply(lambda x: "Ethnic_group_2" if x == 2 else x)
out_patients['Ethnicity'] = out_patients['Ethnicity'].apply(lambda x: "Ethnic_group_3" if x == 3 else x)
out_patients.to_csv('R_patients_out.csv', index = False)
out_patients['Age'] = out_patients['Age'].apply(lambda x: 1000 if x < 65 else x)
out_patients['Age'] = out_patients['Age'].apply(lambda x: 1100 if x < 70 else x)
out_patients['Age'] = out_patients['Age'].apply(lambda x: 1200 if x < 75 else x)
out_patients['Age'] = out_patients['Age'].apply(lambda x: 1300 if x < 100 else x)
out_patients['Age'] = out_patients['Age'].apply(lambda x: "Age_65below" if x == 1000 else x)
out_patients['Age'] = out_patients['Age'].apply(lambda x: "Age_65_70" if x == 1100 else x)
out_patients['Age'] = out_patients['Age'].apply(lambda x: "Age_70_75" if x == 1200 else x)
out_patients['Age'] = out_patients['Age'].apply(lambda x: "Age_75plus" if x == 1300 else x)
out_patients.to_csv('R_patients_out_age_4groups.csv', index = False)
```

In [23]:

```
# Assign labels to Age and Ethnicity columns for the dataset with all samples
scaled_patients = pd.read_csv('predictors_processed_patients.csv')
scaled_patients['Ethnicity'] = scaled_patients['Ethnicity'].apply(lambda x: "Ethnic_group_1" if x == 1 else x)
scaled_patients['Ethnicity'] = scaled_patients['Ethnicity'].apply(lambda x: "Ethnic_group_2" if x == 2 else x)
scaled_patients['Ethnicity'] = scaled_patients['Ethnicity'].apply(lambda x: "Ethnic_group_3" if x == 3 else x)
scaled_patients.to_csv('R_patients_scaled.csv', index = False)
scaled_patients['Age'] = scaled_patients['Age'].apply(lambda x: 1000 if x < 65 else x)
scaled_patients['Age'] = scaled_patients['Age'].apply(lambda x: 1100 if x < 70 else x)
```

```
scaled_patients['Age'] = scaled_patients['Age'].apply(lambda x: 1200 if x < 75 else x)
scaled_patients['Age'] = scaled_patients['Age'].apply(lambda x: 1300 if x < 100 else x)
scaled_patients['Age'] = scaled_patients['Age'].apply(lambda x: "Age_65below" if x == 1000 else x)
scaled_patients['Age'] = scaled_patients['Age'].apply(lambda x: "Age_65_70" if x == 1100 else x)
scaled_patients['Age'] = scaled_patients['Age'].apply(lambda x: "Age_70_75" if x == 1200 else x)
scaled_patients['Age'] = scaled_patients['Age'].apply(lambda x: "Age_75plus" if x == 1300 else x)
scaled_patients.to_csv('R_patients_scaled_age_4groups.csv', index = False)
```