Solution

December 10, 2017

0.1 Ibrahim AbuAlhaol, Ottawa, Canada

- DEC 10, 2017
- Q1: Using data in Bar.csv, How much faster does the treatment improves the dom time?
- Q2: Using data in Foo.csv, How much faster is H2 over H1? What percent improvement does H2 offer?
- Q2: Any other interesting things that jump out from the data?

0.1.1 Importing Libraries

```
In [4]: import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        sns.set_style("darkgrid")
        import numpy as np
        import statsmodels.formula.api as smf
        import statsmodels.stats.multicomp as multi
```

0.1.2 Functions

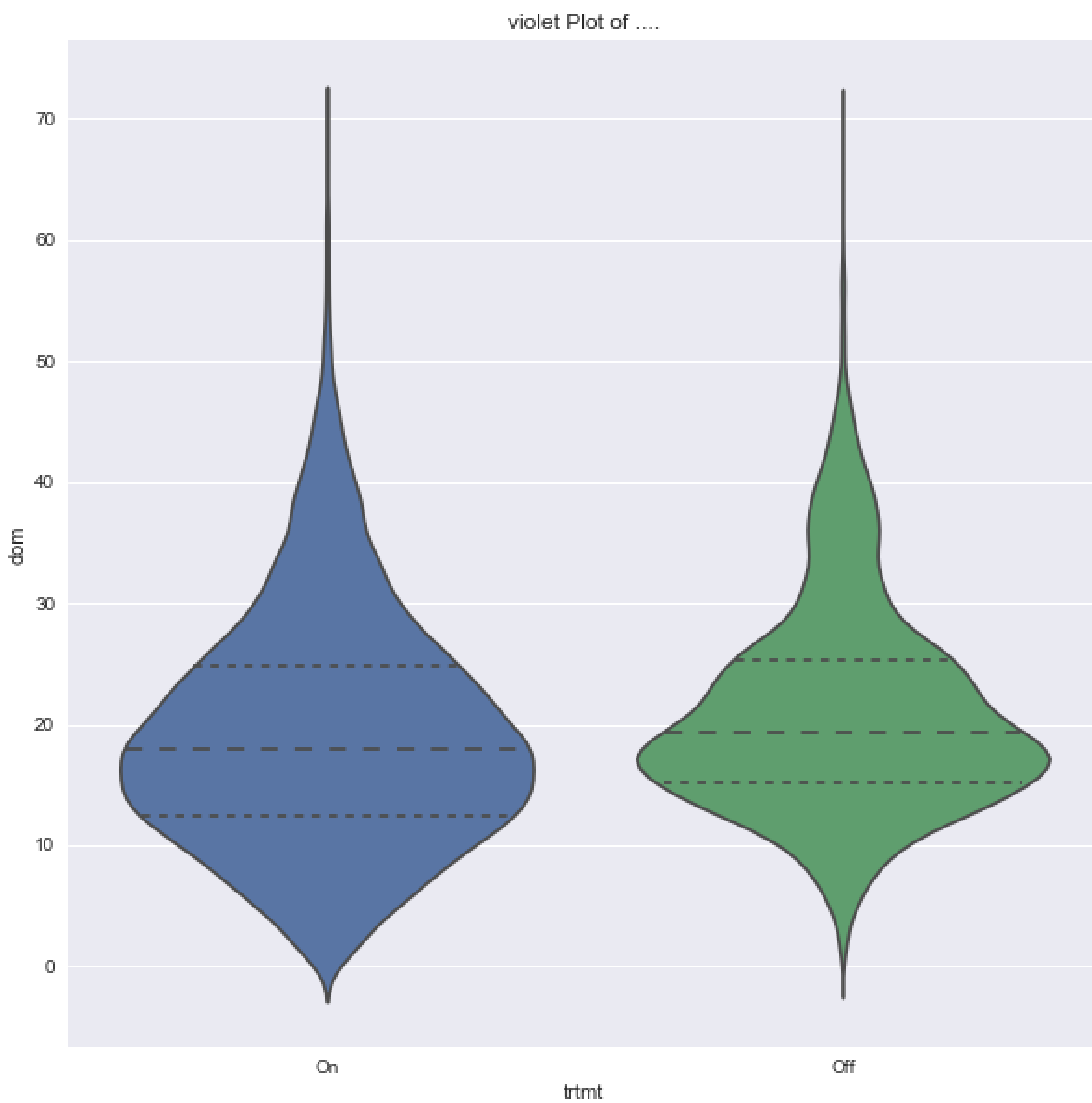
```
In [52]: def plot_corr(df,size=10):
             '''Function plots a graphical correlation matrix for each pair of columns in the 
             Input:
                 df: pandas DataFrame
                 size: vertical and horizontal size of the plot

             '''
             corr = df.corr()
             fig, ax = plt.subplots(figsize=(size, size))
             ax.matshow(corr)
             plt.xticks(range(len(corr.columns)), corr.columns);
             plt.yticks(range(len(corr.columns)), corr.columns);
             plt.show()

         def BoxPlot(Col1,Col2,Title='',size=10):
             fig, ax = plt.subplots(figsize=(size, size))
             sns.violinplot(x=Col1, y=Col2, showmeans=True,scale="width", inner="quartile")
             plt.title(Title)
```

```
plt.show()
```

0.1.3 Importing the CSV files

```
In [96]: BAR=pd.read_csv('data/bar.csv')
         FOO=pd.read_csv('data/foo.csv')
         DF_trtmt_dom=BAR[['trtmt','dom']]
         DF_FOO=FOO[['label','loadTime','fullyLoaded']]
```

```
In [54]: BoxPlot(BAR['trtmt'], BAR['dom'],'violet Plot of ....',10)
         DOM_TRM_ON_Mean=DF_trtmt_dom[DF_trtmt_dom['trtmt']=='On']['dom'].mean()
         DOM_TRM_OFF_Mean=DF_trtmt_dom[DF_trtmt_dom['trtmt']=='Off']['dom'].mean()

         print ('\nThe average dom time with treatement = {}'.format(DOM_TRM_ON_Mean) )
         print ('The average dom time without treatement = {} \n'.format(DOM_TRM_OFF_Mean) )
```



```
The average dom time with treatement = 19.30002315993797
The average dom time without treatement = 21.080772215776854
```

- The difference in the treatment mean does NOT prove the relationship between the treatment and the dom time.
- We now run Anova F-statistic test to identify if the Treatment (categorical predictor variable) associated or related with the dom time (quantitative target variable)?
- To prove the relationship we look for the Anova test F-statistic and Prob (F-statistic) values.

In [63]:
```python
model = smf.ols(formula='dom ~ trtmt', data=BAR)
results = model.fit()
FValue=results.fvalue
FProp=results.f_pvalue
R_sqr=results.rsquared
print ('\nF-statistic = {}'.format(FValue))
print ('Prob (F-statistic) = {}'.format(FProp))
print ('R-squred = {}\n'.format(R_sqr))
#print (results.summary())
```

```
F-statistic =272.1850607794322
Prob (F-statistic) =7.063553955518299e-61
R-squred =0.009053465451638432
```

- Given that the F-statistic is very large and the Prob (F-statistic) is very small, then, we can say that the change in dom time is related to the treatement but with very low R-squred value = 0.009.

0.1.4 The treatment improves the dom time by 1.78 (21.08 -19.30)

In [ ]:

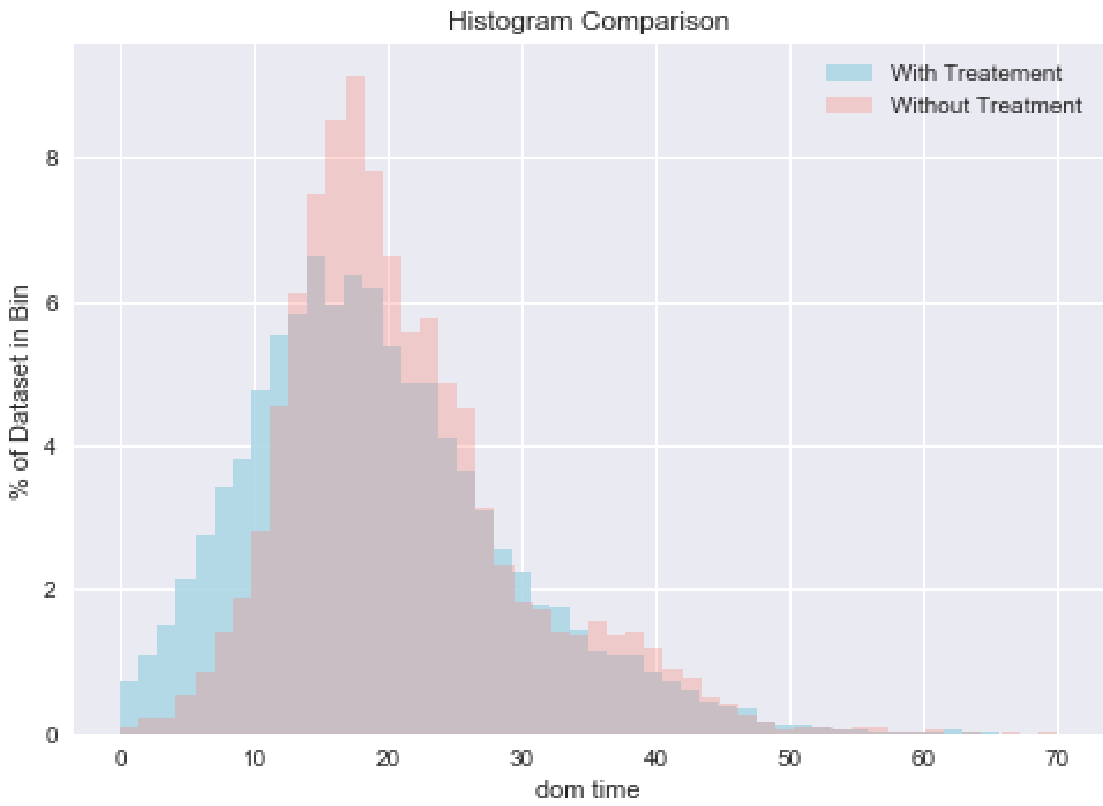
The t-statistic is a measure of the difference between the two sets expressed in units of standard error. Put differently, it's the size of the difference relative to the variance in the data. A high t value means there's probably a real difference between the two sets; you have "significance". The P-value is a measure of the probability of an observation lying at extreme t-values; so a low p-value also implies "significance." If you're looking for a "statistically significant" result, you want to see a very low p-value and a high t-statistic (well, a high absolute value of the t-statistic more precisely). In the real world, statisticians seem to put more weight on the p-value result.

Let's change things up so both A and B are just random, generated under the same parameters. So there's no "real" difference between the two:

In [68]:
```
import numpy as np
from scipy import stats


A=DF_trtmt_dom[DF_trtmt_dom['trtmt']=='On']['dom'].tolist()
B=DF_trtmt_dom[DF_trtmt_dom['trtmt']=='Off']['dom'].tolist()

stats.ttest_ind(A, B)
```

Out[68]:
```
Ttest_indResult(statistic=-16.498032027469961, pvalue=7.063553955755318e-61)
```

In [81]:
```
import matplotlib.pyplot as plt
import numpy as np
np.random.seed(1)

xweights = 100 * np.ones_like(A) / len(A)
yweights = 100 * np.ones_like(B) / len(B)

fig, ax = plt.subplots()
ax.hist(A, weights=xweights, color='lightblue', alpha=0.9, bins=50)
ax.hist(B, weights=yweights, color='salmon', alpha=0.3,bins=50)
ax.legend(['With Treatement', 'Without Treatment'])
ax.set(title='Histogram Comparison', ylabel='% of Dataset in Bin')
ax.margins(0.05)
ax.set_ylim(bottom=0)
plt.xlabel('dom time')
plt.show()
```



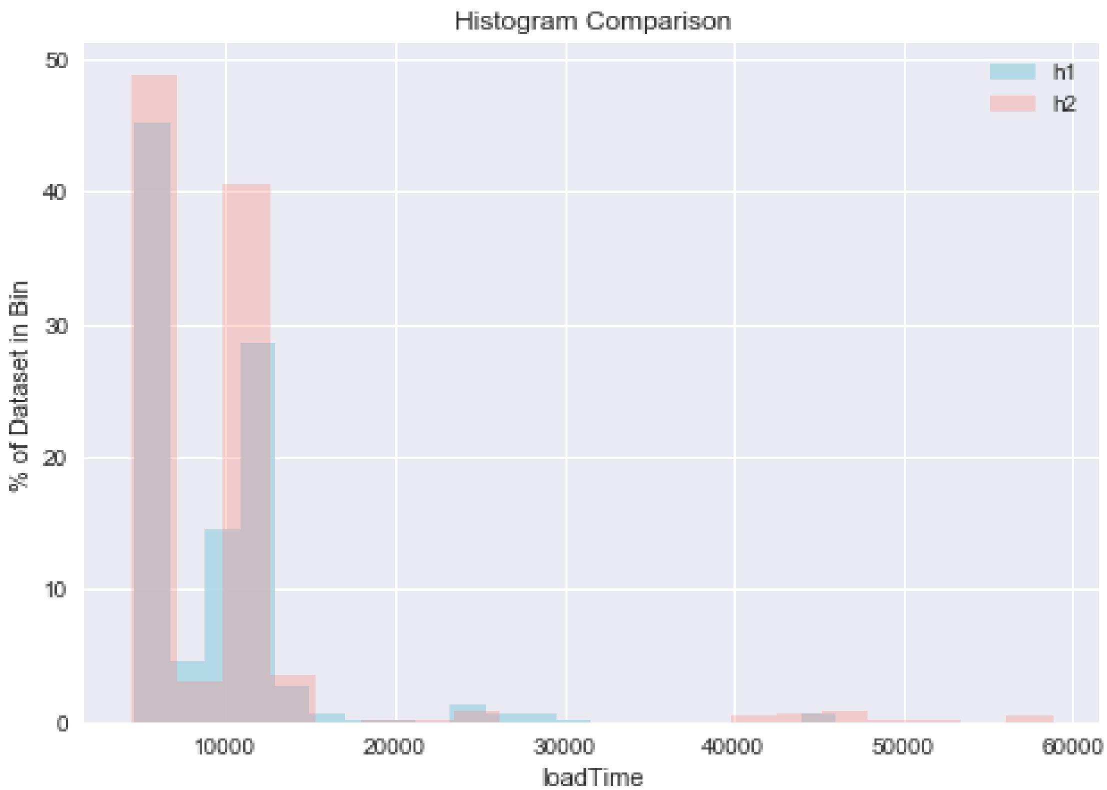

```
In [86]: FOO.columns

Out[86]: Index(['testId', 'label', 'location', 'completed_time_stamp', 'completed_time',
                'run_no', 'loadTime', 'fullyLoaded', 'bytesIn', 'bytesInDoc',
                'requests', 'requestsDoc', 'TTFB', 'firstPaint', 'render', 'SpeedIndex',
                'docTime', 'ttfb_first_req', 'ssl_first_req', 'dns_first_req'],
               dtype='object')

In [97]: DF_FOO['diff']=DF_FOO.apply(lambda row: row[2]-row[1],axis=1)

/Users/alhaol/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: SettingWithCopyWa
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  """Entry point for launching an IPython kernel.


In [99]: DF_FOO.nunique()

Out[99]: label            2
         loadTime       742
```

```
fullyLoaded    788
diff           682
dtype: int64
```

In [108]:
```python
import numpy as np
from scipy import stats


A=DF_FOO[DF_FOO['label']=='h1']['loadTime'].tolist()
B=DF_FOO[DF_FOO['label']=='h2']['loadTime'].tolist()

stats.ttest_ind(A, B)
```

Out[108]:
```
Ttest_indResult(statistic=-0.8736228043200106, pvalue=0.3825426361512243)
```

In [110]:
```python
xweights = 100 * np.ones_like(A) / len(A)
yweights = 100 * np.ones_like(B) / len(B)

fig, ax = plt.subplots()
ax.hist(A, weights=xweights, color='lightblue', alpha=0.9, bins=20)
ax.hist(B, weights=yweights, color='salmon', alpha=0.3,bins=20)
ax.legend(['h1', 'h2'])
ax.set(title='Histogram Comparison', ylabel='% of Dataset in Bin')
ax.margins(0.05)
ax.set_ylim(bottom=0)
plt.xlabel('loadTime')
plt.show()
```



In [111]:
```
import numpy as np
from scipy import stats


A=DF_FOO[DF_FOO['label']=='h1']['fullyLoaded'].tolist()
B=DF_FOO[DF_FOO['label']=='h2']['fullyLoaded'].tolist()

stats.ttest_ind(A, B)
```

Out[111]: Ttest_indResult(statistic=-0.81850317851539156, pvalue=0.41327347416106186)

In [ ]:
```
xweights = 100 * np.ones_like(A) / len(A)
yweights = 100 * np.ones_like(B) / len(B)

fig, ax = plt.subplots()
ax.hist(A, weights=xweights, color='lightblue', alpha=0.9, bins=20)
ax.hist(B, weights=yweights, color='salmon', alpha=0.3,bins=20)
ax.legend(['h1', 'h2'])
ax.set(title='Histogram Comparison', ylabel='% of Dataset in Bin')
ax.margins(0.05)
ax.set_ylim(bottom=0)
plt.xlabel('loadTime')
plt.show()
```

0.2 Unfortunitly there is no statistical evidence * low p-value to prove

0.3 that any of them is better, the performance cam from rando

In [ ]:

In [82]: FOO.info()

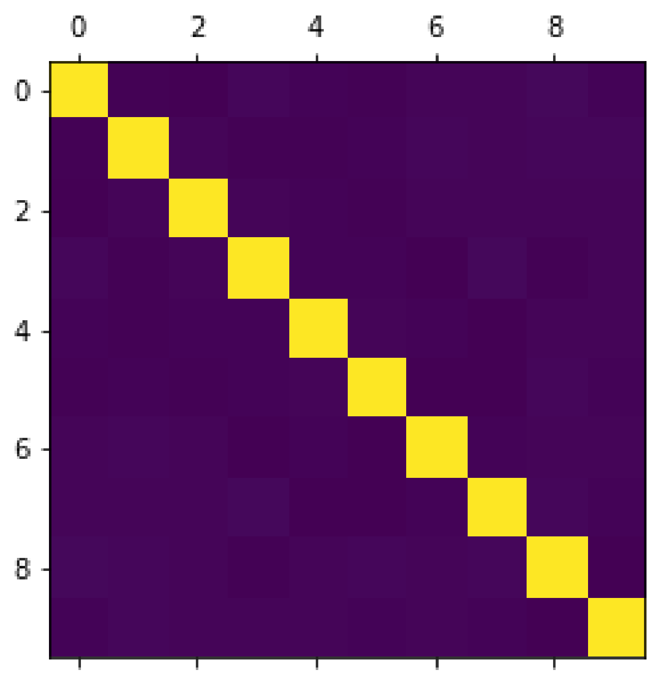
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 960 entries, 0 to 959
Data columns (total 20 columns):
testId                  960 non-null object
label                   960 non-null object
location                960 non-null object
completed_time_stamp    960 non-null int64
completed_time          960 non-null object
run_no                  960 non-null int64
loadTime                960 non-null int64
fullyLoaded             960 non-null int64
bytesIn                 960 non-null int64
bytesInDoc              960 non-null int64
requests                960 non-null int64
requestsDoc             960 non-null int64
TTFB                    960 non-null int64
firstPaint              960 non-null int64
render                  960 non-null int64
SpeedIndex              960 non-null int64
docTime                 960 non-null int64
ttfb_first_req          960 non-null int64
ssl_first_req           960 non-null int64
dns_first_req           960 non-null int64
dtypes: int64(16), object(4)
memory usage: 150.1+ KB
```

In [22]: BAR.info()

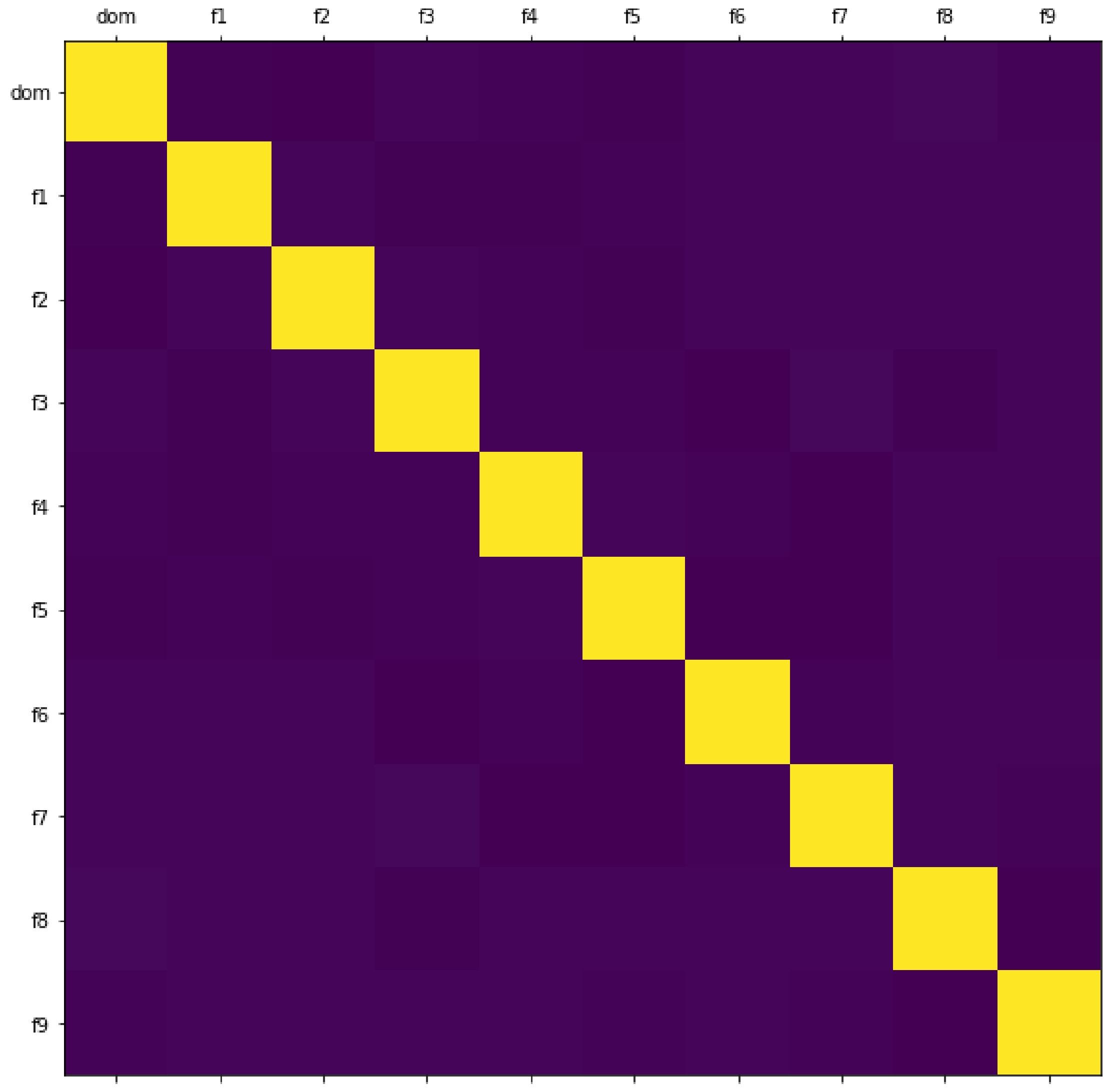
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29794 entries, 0 to 29793
Data columns (total 18 columns):
Unnamed: 0     29794 non-null int64
host           29794 non-null object
page           29794 non-null object
isp            29794 non-null object
browser        29794 non-null object
device         29794 non-null object
trtmt          29794 non-null object
dom            29794 non-null float64
queryString    29794 non-null object
```

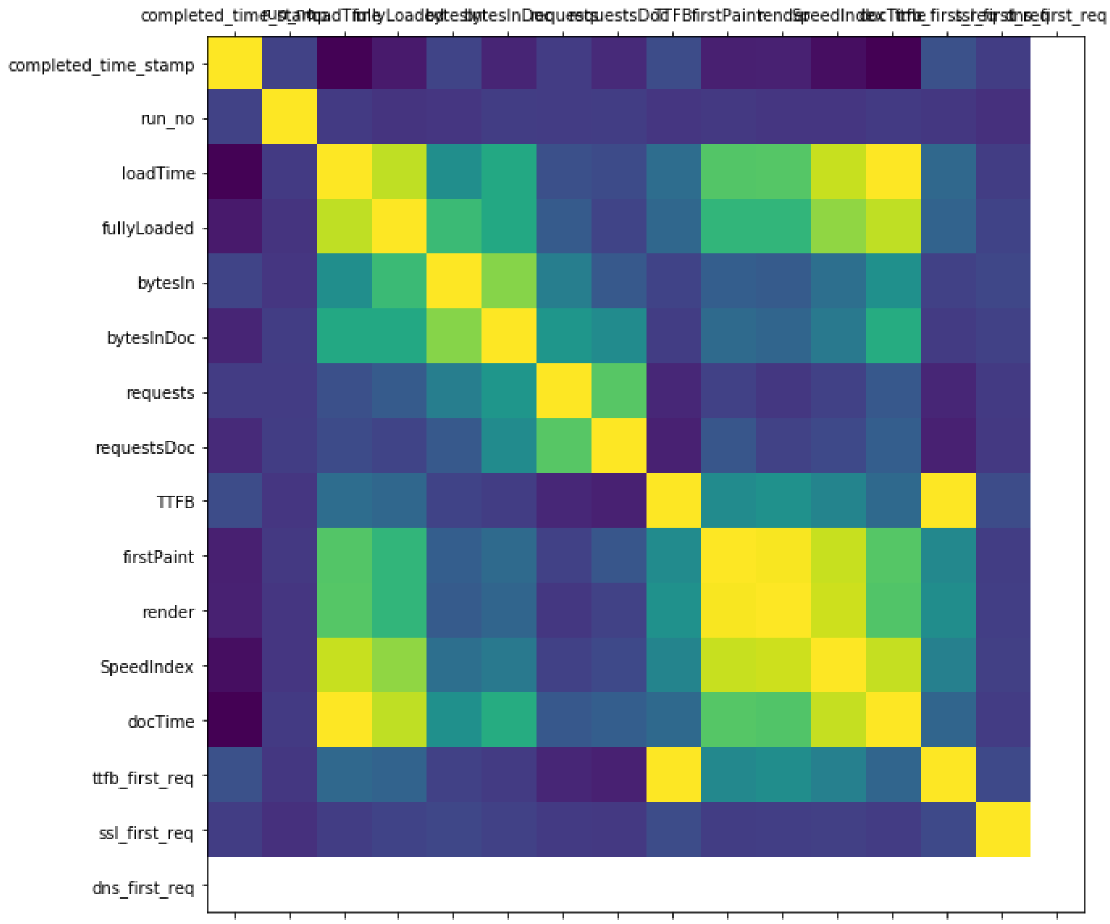
```
f1              29794 non-null float64
f2              29794 non-null float64
f3              29794 non-null float64
f4              29794 non-null float64
f5              29794 non-null float64
f6              29794 non-null float64
f7              29794 non-null float64
f8              29794 non-null float64
f9              29794 non-null float64
dtypes: float64(10), int64(1), object(7)
memory usage: 4.1+ MB
```

In [75]:
```
plt.matshow(BAR.corr())
plt.show()
```



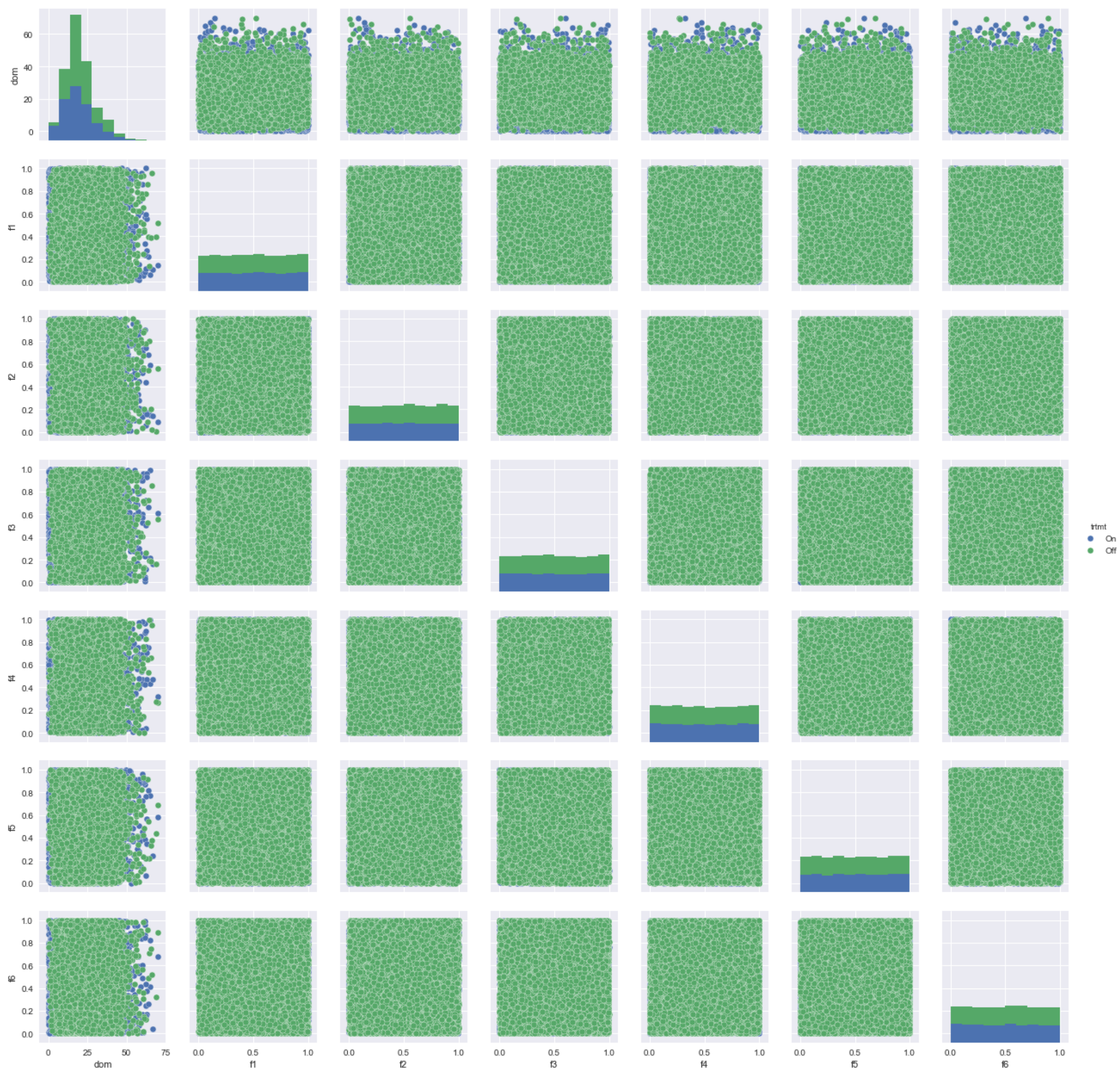
In [83]:
```
plot_corr(BAR,size=10)
plot_corr(FOO,size=10)
```


dom
f1
f2
f3
f4
f5
f6
f7
f8
f9




```
In [94]: BAR.columns.tolist()

Out[94]: ['host',
          'page',
          'isp',
          'browser',
          'device',
          'trtmt',
          'dom',
          'queryString',
          'f1',
          'f2',
          'f3',
          'f4',
          'f5',
          'f6',
          'f7',
          'f8',
          'f9']
```

In [103]:
```python
import seaborn as sns
sns.pairplot(BAR[['dom','f1','f2','f3','f4','f5','f6','trtmt']], hue='trtmt')
plt.show()
```

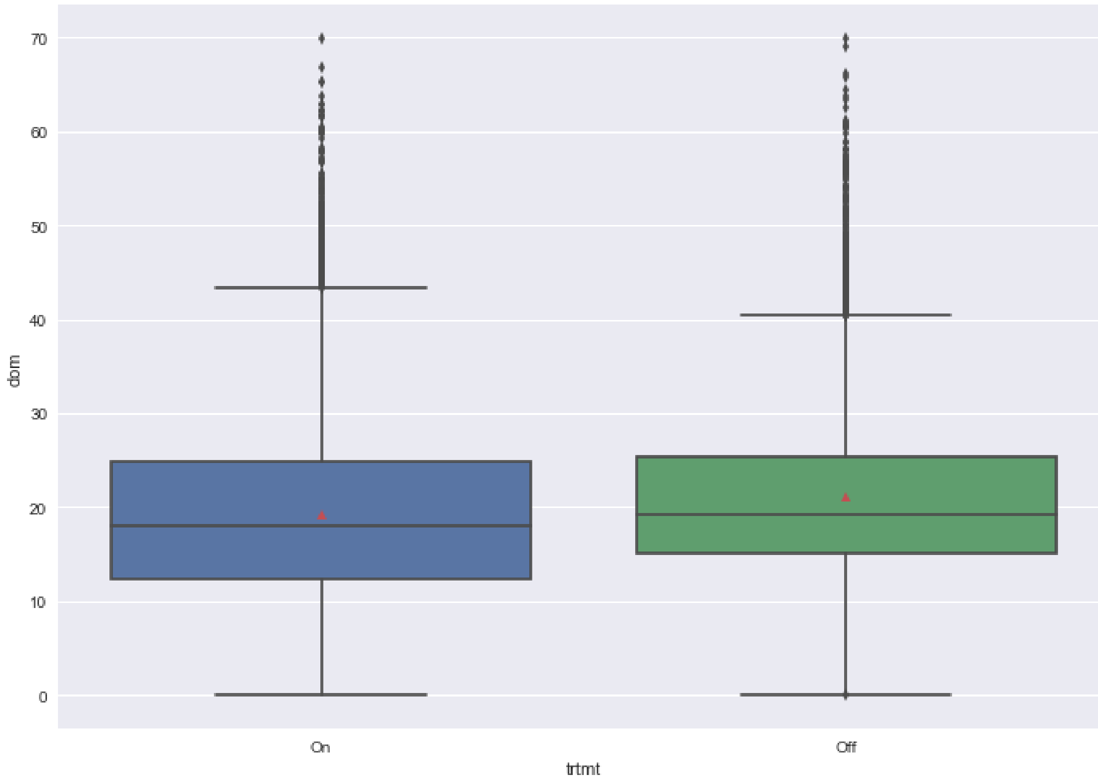


In [106]:
```python
BAR[['dom','f1','f2','f3','f4','f5','f6','trtmt']].head(3)
```

Out[106]:

	dom	f1	f2	f3	f4	f5	f6	trtmt
0	43.846811	0.205366	0.817999	0.533166	0.358433	0.069712	0.370256	On
1	34.078843	0.455522	0.793443	0.090781	0.609515	0.746002	0.511265	On
2	10.423054	0.195735	0.491308	0.147366	0.797300	0.409352	0.857075	On

In [110]:

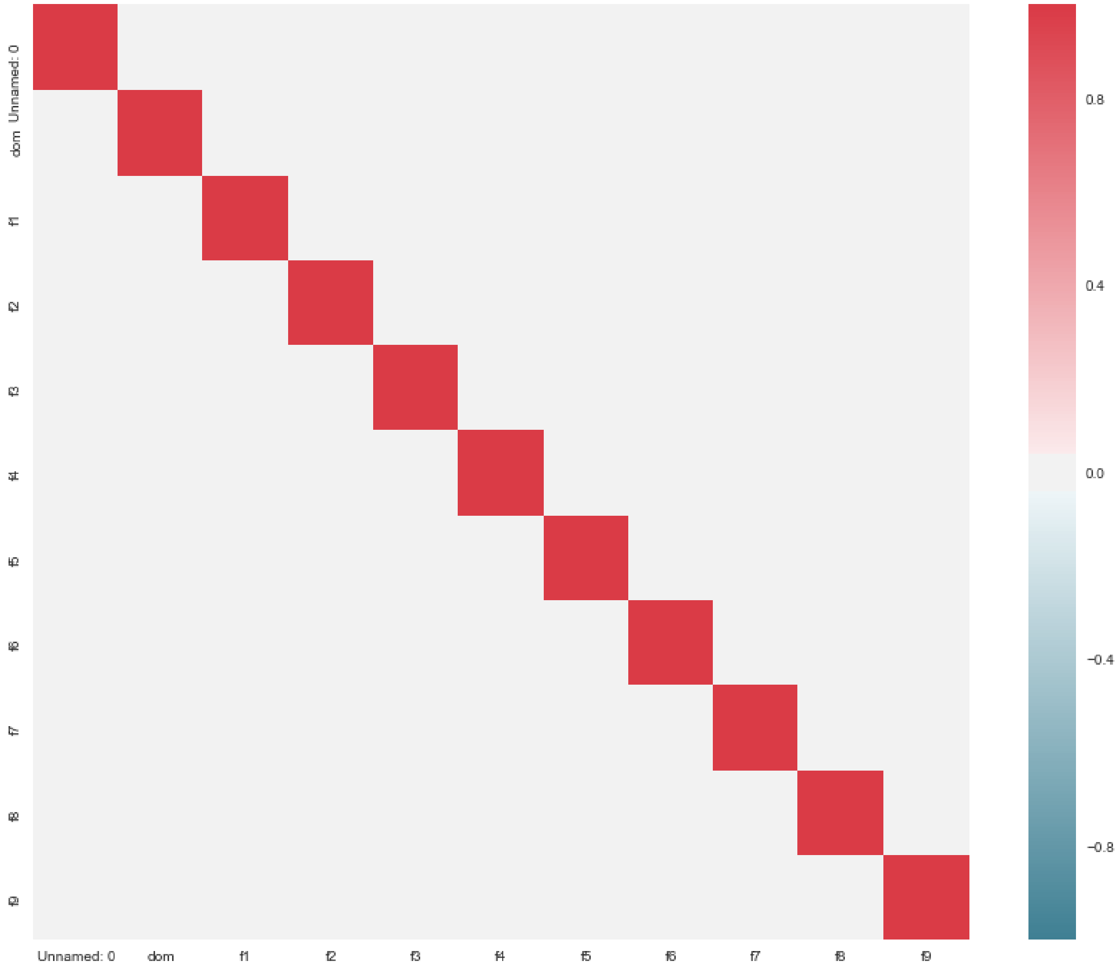



```
In [113]: #ANOVA F Test
          import numpy as np
          import pandas as pd
          import statsmodels.formula.api as smf
          import statsmodels.stats.multicomp as multi
          model = smf.ols(formula='dom ~ trtmt', data=BAR)
          results = model.fit()
          print (results.summary())
```

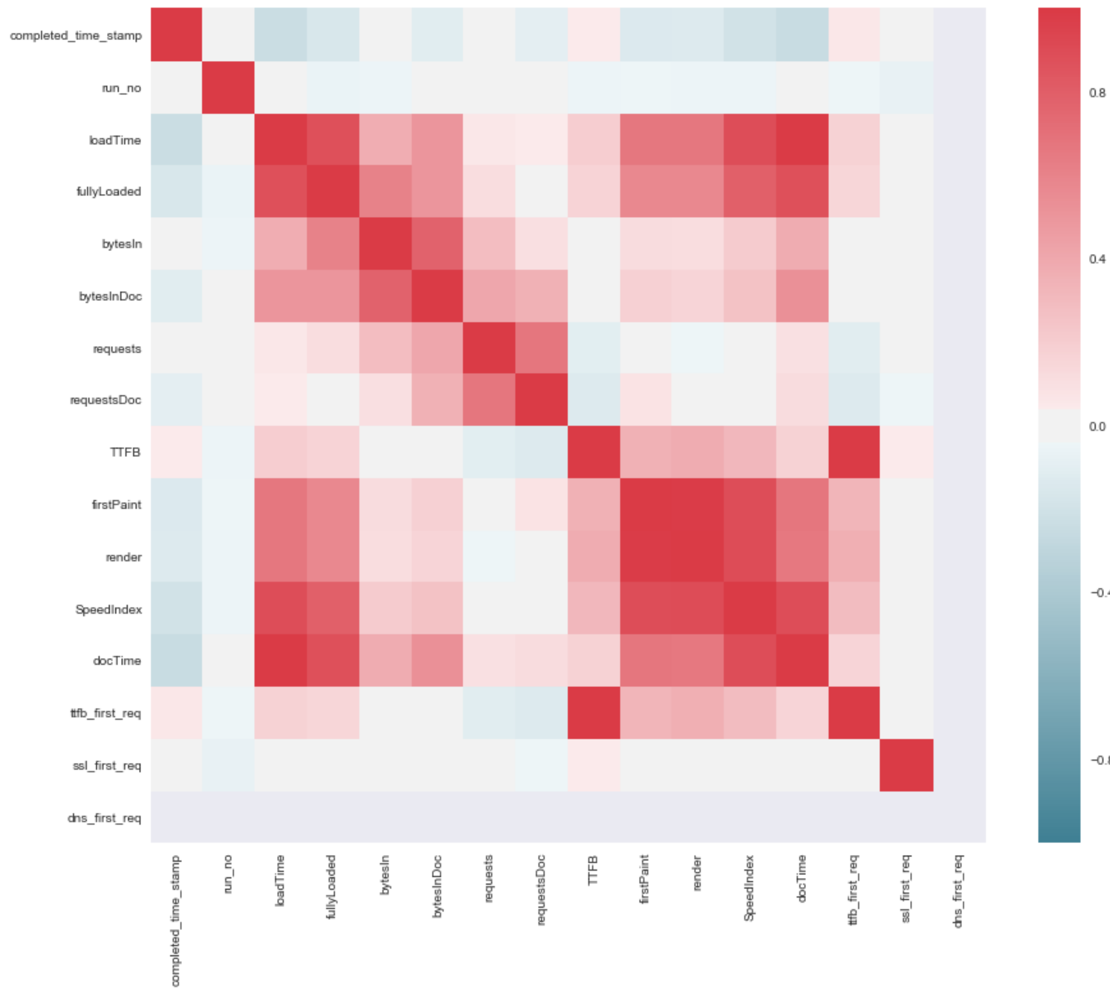
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    dom   R-squared:                       0.009
Model:                            OLS   Adj. R-squared:                  0.009
Method:                 Least Squares   F-statistic:                     272.2
Date:                Sat, 09 Dec 2017   Prob (F-statistic):           7.06e-61
Time:                        13:06:43   Log-Likelihood:            -1.0876e+05
No. Observations:               29794   AIC:                         2.175e+05
Df Residuals:                   29792   BIC:                         2.175e+05
Df Model:                           1
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
```

```
-------------------------------------------------------------------------------
Intercept      21.0808      0.076    277.446      0.000      20.932      21.230
trtmt[T.On]    -1.7807      0.108    -16.498      0.000      -1.992      -1.569
==============================================================================
Omnibus:                     3493.413   Durbin-Watson:                   1.982
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             5196.387
Skew:                           0.873   Prob(JB):                         0.00
Kurtosis:                       4.065   Cond. No.                         2.61
==============================================================================

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

In [116]:
```
import seaborn as sns

f, ax = plt.subplots(figsize=(15, 12))
corr = BAR.corr()
sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool), cmap=sns.diverging_palett
            square=True, ax=ax)
plt.show()
```




```
In [117]: import seaborn as sns

          f, ax = plt.subplots(figsize=(15, 12))
          corr = FOO.corr()
          sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool), cmap=sns.diverging_palette
                      square=True, ax=ax)
          plt.show()
```



Thank you for your interest in the Data Scientist position in our Ottawa office. When you have a chance please complete the attached exercise (ideally we would like the exercise completed and returned in 2 days but if you have other obligations please feel free to let me know). If you have any questions please do not hesitate to call or email me at any time. Regards, Nick Russo

Questions for attached Excercise Bar.csv How much faster does the treatment improves the dom time? Foo.csv: How much faster is H2 over H1? What percent improvement does H2 offer? Any other interesting things that jump out from the data.

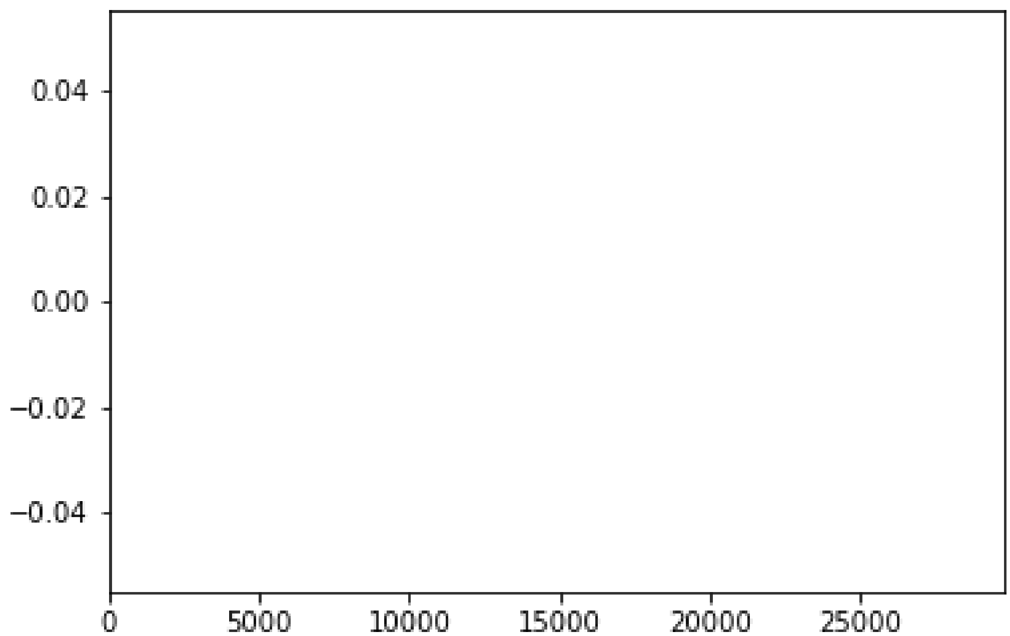
```
In [65]: DOM_TRM=BAR[['dom','trtmt']]
```

```
In [66]: import numpy as np

         DOM_TRM_ON=DOM_TRM[DOM_TRM['trtmt']=='On']['dom']
         DOM_TRM_off=DOM_TRM[DOM_TRM['trtmt']=='Off']['dom']
         DOM_TRM_off=pd.sub
```

```
  File "<ipython-input-66-d08cd9db6fd4>", line 4
```

```
    DOM_TRM_ON=DOM_TRM[DOM_TRM['trtmt']=='On']['dom']
              ^
IndentationError: expected an indented block
```

In [60]:
```python
import matplotlib.pyplot as plt
DOM=DOM_TRM_ON-DOM_TRM_off
DOM.plot()
plt.show()
```



In [54]:
```python
DOM_TRM_off
```

Out[54]: 15030

In [47]:
```python
FOO[FOO['label']=='h1']['fullyLoaded'].mean()
```

Out[47]: 9168.816666666668

In [48]:
```python
FOO[FOO['label']=='h2']['fullyLoaded'].mean()
```

Out[48]: 9506.891666666666

In [49]:
```
Waht to do:

    1) Correlation between Features
    2) PCA and visulization of the fature
    3) Random forest predition
    4) Correlation matrix
```