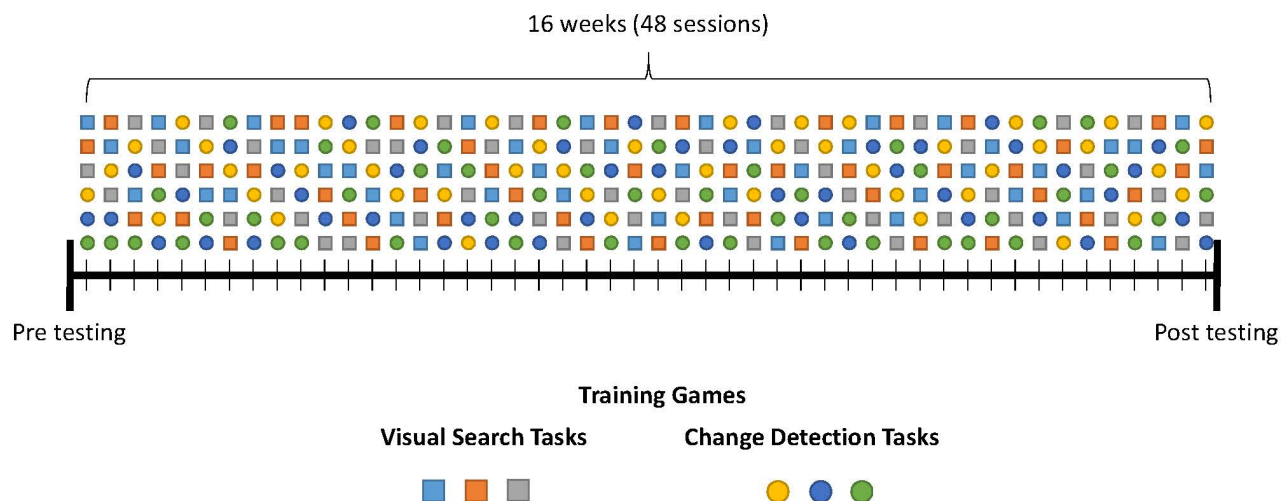


# IEEE Data - Statistical Analysis

Welcome to our sats notebook! We're going to be looking at some data from the IEEE brain bank. The data was taken over **48 trials** (16 weeks) with each subject taking part in a **baseline pre-test**, then multiple types of **training tests**, and a benchmark **post-test after** the training trials had been completed. The pre and post tests included the LSAT, as well as a figure series. The training tests included 3 visiospatial tasks and 3 change detection tasks.

## Timeline



```
In [1]: # just importing the modules we'll need
%matplotlib inline
import pandas as pd
import os
import numpy as np
import pandas as pd
import glob
from matplotlib import pyplot as plt
import matplotlib.cm as cm
import scipy.stats
from pylab import plot, show, savefig, xlim, figure, \
                    hold, ylim, legend, boxplot, setp, axes

# Let's import our data as a dataframe for easy statistical analysis (td for t
esting data)
td = pd.read_csv(os.path.join(os.getcwd(), 'PrePostData/TestingDataALL1.csv'))
```

Let's start just by taking a look at the data to see what we're dealing with

```
In [2]: td.head()
```

Out[2]:

	Subject	Age	Gender	PreFigureSeries	PreLSATReasoning	PreLetterComp_Accuracy
0	10	21	1	24	13	0.869565
1	11	20	2	20	15	0.962963
2	12	20	1	15	11	1.000000
3	13	21	2	18	10	0.958333
4	14	21	1	20	14	1.000000

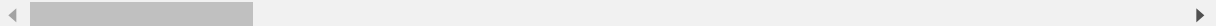
Gross, looks like it's arbitrarily sorted.

Let's just fix that really quick.

```
In [19]: # sort by Subject  
td.set_index('Subject').head()
```

Out[19]:

	Age	Gender	PreFigureSeries	PreLSATReasoning	PreLetterComp_Accuracy	P
Subject						
10	21	1	24	13	0.869565	2
11	20	2	20	15	0.962963	2
12	20	1	15	11	1.000000	2
13	21	2	18	10	0.958333	2
14	21	1	20	14	1.000000	2



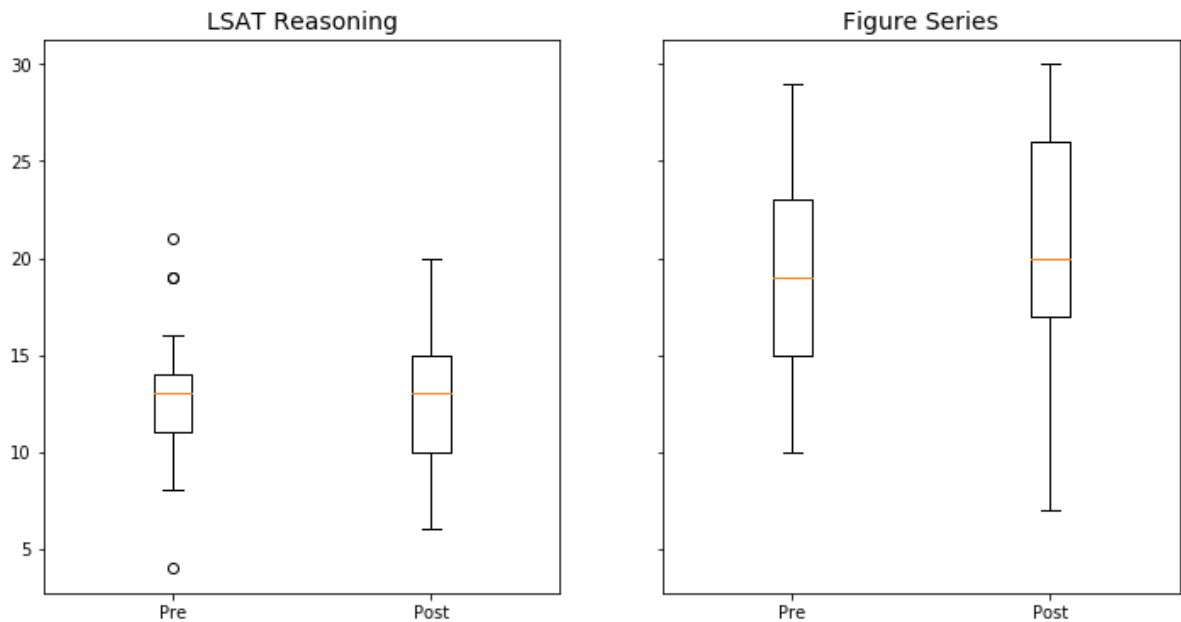
That's better. So it looks like we have two different pre/post tests to compare. The LSATReasoning, and the FigureSeries

## Analysis - Part 1

Let's plot both just to see what we're working with

```
In [4]: fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 6), sharey=True)
axes[0].boxplot([td['PreLSATReasoning'], td['PostLSATReasoning']], labels=["Pre", "Post"])
axes[0].set_title('LSAT Reasoning', fontsize=14)

axes[1].boxplot([td['PreFigureSeries'], td['PostFigureSeries']], labels=["Pre", "Post"])
axes[1].set_title('Figure Series', fontsize=14)
plt.show()
```



Hmm, doesn't look like there's much variation. Let's run some t-tests, though, just to be sure.

**Remember, a t-test is used to compare the means of two different samples in relation to the variations in the data.**

our null hypothesis is that the pre and post series are not statistically different, which would result in a p-value > 0.05

```
In [5]: scipy.stats.ttest_ind(td['PostFigureSeries'], td['PreFigureSeries']).pvalue
```

```
Out[5]: 0.45076787849712463
```

```
In [6]: scipy.stats.ttest_ind(td['PostLSATReasoning'], td['PreLSATReasoning']).pvalue
```

```
Out[6]: 0.93779903968594625
```

Our p-values are 0.45 and 0.937. Not very promising.

Conclusion:

**We cannot reject the null hypothesis**

## Analysis - Part 2

Let's do something else. Instead of looking at the pre/post tests, let's look at the training tests.

**We'll try to plot each subject's difficulty level over time to see if we can describe the trend**

```
In [7]: # import training data file names
os.path.join(os.getcwd(), 'Subjects/ChangeDetection')
path = os.path.join(os.getcwd(), 'Subjects/ChangeDetection')
filenames = glob.glob(path + "/*.csv")

# make a dataframe for each file
dfs = []
for filename in filenames:
    dfs.append(pd.read_csv(filename))

# Concatenate all data into one DataFrame -- All Subjects' change detection data
allSubjects_cd = pd.concat(dfs, ignore_index=True)
```

Let's check out what we have.

```
In [8]: allSubjects_cd.head()
```

Out[8]:

	Subject	Session	GamesPlayed	Version	AvgDifficulty	StartDifficulty	EndDifficulty	N
0	1	1	16.0	1	6.125000	3.0	8.0	3
1	1	2	10.0	1	7.700000	8.0	9.0	7
2	1	3	11.0	1	10.727273	9.0	12.0	9
3	1	4	10.0	1	13.000000	12.0	14.0	1
4	1	5	11.0	1	16.363636	14.0	20.0	1



Awesome. The dataframe holds each subject's 48 trials and their ratings for each one.

let's just start by restructuring the dataframe for easy accessibility.

```
In [9]: # filtering for only the first version
allSubjects_version1 = allSubjects_cd.loc[allSubjects_cd['Version']==1]

# create a new data frame for just the average difficulties of game 1, all sub
s
allSubjects_avgDifficulty = pd.DataFrame()

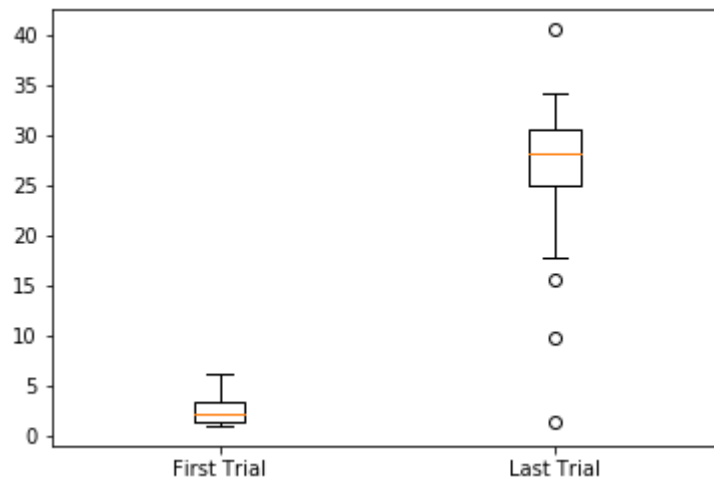
for subject in range(1,25):
    df = pd.DataFrame()
    oneSubject = allSubjects_version1.loc[allSubjects_version1['Subject']==sub
ject]
    oneSubject_avgDifficulty = oneSubject['AvgDifficulty'].values
    df['%d' % (subject)] = oneSubject_avgDifficulty.tolist()
    allSubjects_avgDifficulty = pd.concat([allSubjects_avgDifficulty, df], ign
ore_index=True, axis=1)
```

now rename these absurdly long variables for brevity over clarity

```
In [10]: ad = allSubjects_avgDifficulty
ad = allSubjects_avgDifficulty[np.logical_not(np.isnan(allSubjects_avgDifficul
ty))]
```

and really quickly, look at the boxplot of the first and last trial difficulty distribution

```
In [11]: fig = figure()
plt.boxplot([ad.iloc[0], ad.iloc[40]], labels=["First Trial", "Last Trial"])
plt.show()
```



plus a quick t-test, just to check.

```
In [12]: scipy.stats.ttest_ind(ad.iloc[0], ad.iloc[40]).pvalue
```

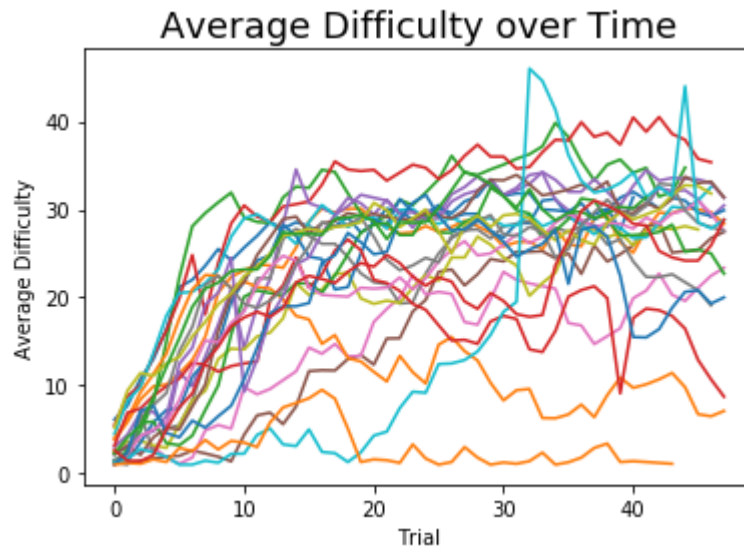
```
Out[12]: 1.1470033593210763e-17
```

awesome, that's what we like to see.

Now we know for sure that **there's a change between the first and last trials.**

Let's plot all the data, then see if we can fit a model to that trend.

```
In [13]: fig = figure()
for sub in range(0,24):
    plt.plot(ad.index, ad[sub])
plt.ylabel('Average Difficulty')
plt.xlabel('Trial')
plt.title('Average Difficulty over Time', fontsize=18)
plt.show()
```



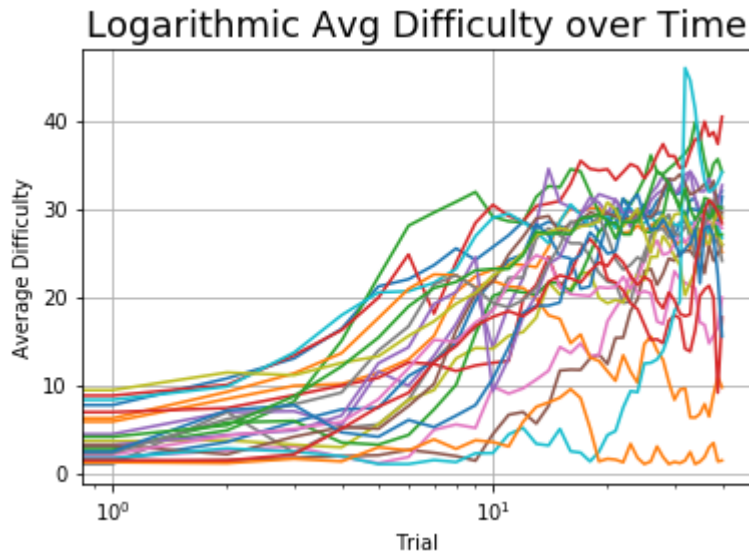
Yikes. that's gnarly. But it does look like there's a trend.

Let's try plotting it logarithmically to see if it gets more linear

It will be easier to perform a regression on it that way.



```
In [216]: fig = figure()
for sub in range(0,24):
    plt.plot(ad.index, ad[sub])
plt.xscale('log')
plt.ylabel('Average Difficulty')
plt.xlabel('Trial')
plt.title('Logarithmic Avg Difficulty over Time', fontsize=18)
plt.grid(True)
plt.show()
```



Cool. looks like the perfect data set for a multiple linear regression

## Mutiple Linear Regression

We're going to use the method called Ordinary Least Squares (OLS) which means we're trying to fit a regression line that would minimize the sum of the square of distance of each point from the regression line.

```
In [16]: import statsmodels.api as sm
```

```
C:\ProgramData\Miniconda3\lib\site-packages\statsmodels\compat\pandas.py:56:
FutureWarning: The pandas.core.datetools module is deprecated and will be rem
oved in a future version. Please use the pandas.tseries module instead.
    from pandas.core import datetools
```

```

In [203]: ad.dropna(inplace=True)
X = np.log10(ad.index+1) # independent variable
y = ad[0] # dependent variable
y = sm.add_constant(y)
# Note the difference in argument order
model = sm.OLS(X, y).fit()
predictions = model.predict(y) # make the predictions by the model

# Print out the statistics
model.summary()

```

Out[203]: OLS Regression Results

<b>Dep. Variable:</b>	y	<b>R-squared:</b>	0.860
<b>Model:</b>	OLS	<b>Adj. R-squared:</b>	0.857
<b>Method:</b>	Least Squares	<b>F-statistic:</b>	240.2
<b>Date:</b>	Wed, 20 Dec 2017	<b>Prob (F-statistic):</b>	2.92e-18
<b>Time:</b>	14:10:28	<b>Log-Likelihood:</b>	22.340
<b>No. Observations:</b>	41	<b>AIC:</b>	-40.68
<b>Df Residuals:</b>	39	<b>BIC:</b>	-37.25
<b>Df Model:</b>	1		
<b>Covariance Type:</b>	nonrobust		

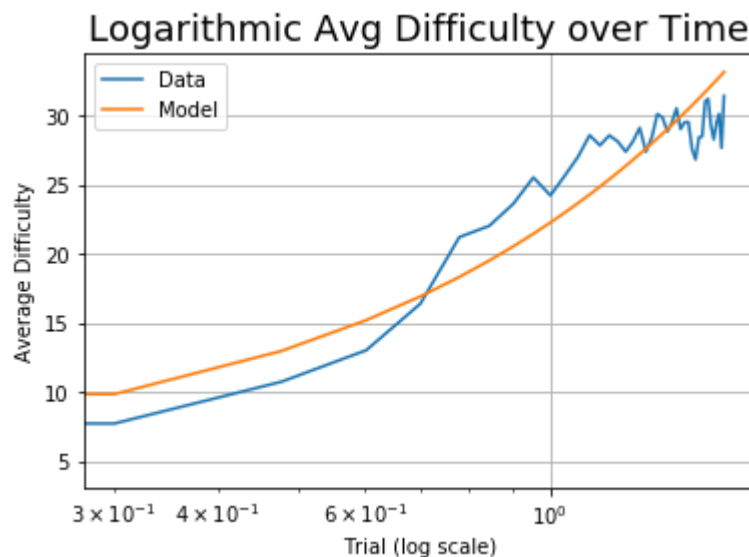
	coef	std err	t	P> t	[0.025	0.975]
<b>const</b>	-0.2522	0.097	-2.603	0.013	-0.448	-0.056
<b>0</b>	0.0563	0.004	15.498	0.000	0.049	0.064

<b>Omnibus:</b>	4.398	<b>Durbin-Watson:</b>	0.369
<b>Prob(Omnibus):</b>	0.111	<b>Jarque-Bera (JB):</b>	1.914
<b>Skew:</b>	0.151	<b>Prob(JB):</b>	0.384
<b>Kurtosis:</b>	1.986	<b>Cond. No.</b>	115.

That's a lot of nonsense. (Omnibus, seriously?). Basically, though, what we're looking at is a breakdown of the OLS analysis. Some are self explanatory -- like Method, and Model. The R number indicates the % of variance our model explains (82%). But what we care most about is the second table with the coef, std err, t, etc. The first column in that table gives us the **coefficients for the requisite linear equation (ours, for example, is  $y=16.7x+3.1$ )**. It also includes things like the 95% confidence interval and the p-values. We have an *almost* significant value for b, and a significant one for m.

Let's plot the regression on top of the data to see they fit.

```
In [205]: fig = figure()
data = plt.plot(np.log10(ad.index+1), ad[0], label='Data')
x = np.log10(ad.index+1)
y = (x +.2522)/0.0563
plt.plot(x, y, label='Model')
plt.xscale('log')
plt.ylabel('Average Difficulty')
plt.xlabel('Trial (log scale)')
plt.title('Logarithmic Avg Difficulty over Time', fontsize=18)
plt.legend()
plt.grid(True)
plt.show()
```



Hmm. Not bad. Now we'll include all 25 to see if we get better results.

```
In [208]: ad.dropna(inplace=True)
X = np.log10(ad.index+1) # independent variable
ad['mean'] = ad.mean(axis=1)
y = ad[['mean']] # dependent variable
y = sm.add_constant(y)
# Note the difference in argument order
model = sm.OLS(X, y).fit()
predictions = model.predict(y) # make the predictions by the model

# Print out the statistics
model.summary()
```

Out[208]: OLS Regression Results

<b>Dep. Variable:</b>	y	<b>R-squared:</b>	0.955
<b>Model:</b>	OLS	<b>Adj. R-squared:</b>	0.954
<b>Method:</b>	Least Squares	<b>F-statistic:</b>	824.5
<b>Date:</b>	Wed, 20 Dec 2017	<b>Prob (F-statistic):</b>	7.61e-28
<b>Time:</b>	14:32:00	<b>Log-Likelihood:</b>	45.487
<b>No. Observations:</b>	41	<b>AIC:</b>	-86.97
<b>Df Residuals:</b>	39	<b>BIC:</b>	-83.55
<b>Df Model:</b>	1		
<b>Covariance Type:</b>	nonrobust		

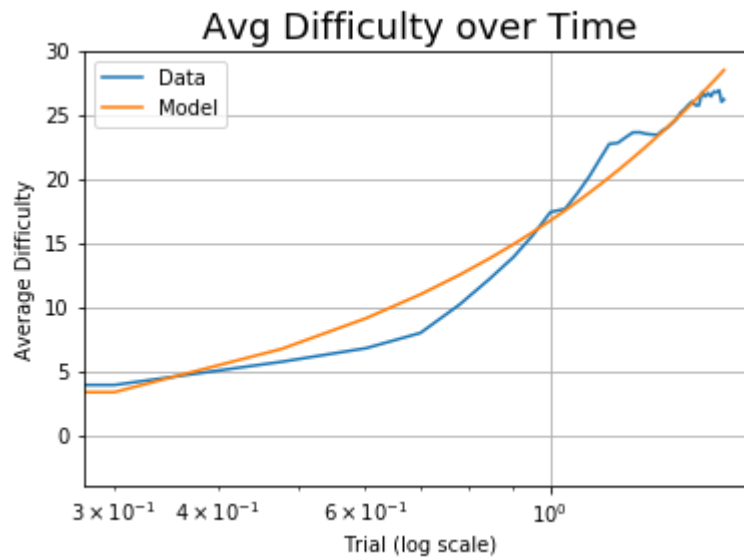
	coef	std err	t	P> t	[0.025	0.975]
<b>const</b>	0.1269	0.040	3.193	0.003	0.047	0.207
<b>mean</b>	0.0522	0.002	28.715	0.000	0.048	0.056

<b>Omnibus:</b>	6.761	<b>Durbin-Watson:</b>	0.347
<b>Prob(Omnibus):</b>	0.034	<b>Jarque-Bera (JB):</b>	5.841
<b>Skew:</b>	-0.648	<b>Prob(JB):</b>	0.0539
<b>Kurtosis:</b>	4.318	<b>Cond. No.</b>	68.2

You'll note here that I could have added multiple dependent variables (eg. ad[0, 1, 2...]) instead of taking the average of the data row-wise. I decided to do the mean instead because it yielded less degrees of freedom (meaning I had 2 coefficients instead of 23), and was nominally accurate for our needs (R-squared of 0.955 vs 0.995).

let's plot that one more time

```
In [212]: fig = figure()
data = plt.plot(np.log10(ad.index+1), ad['mean'], label='Data')
x = np.log10(ad.index+1)
y = (x -.1269)/0.0522
plt.plot(x, y, label='Model')
plt.xscale('log')
plt.ylabel('Average Difficulty')
plt.xlabel('Trial (log scale)')
plt.title('Avg Difficulty over Time', fontsize=18)
plt.legend()
plt.grid(True)
plt.show()
```



Hey, looks pretty good. And the **R-squared is .955** which is better than we can reasonably expect.

## Stats Note

***After some research, we decided to use a regression instead of ANOVA, simply because it's easier to manipulate and interpret on a graphical level. The statistical process for each is fairly similar, if not the same.***

Karen Grace-Martin from TheAnalysisFactor.com explains why:

*We can run this as either an ANOVA or a regression. In the ANOVA, the categorical variable is effect coded, which means that each category's mean is compared to the grand mean. In the regression, the categorical variable is dummy coded, which means that each category's intercept is compared to the reference group's intercept. Since the intercept is defined as the mean value when all other predictors = 0, and there are no other predictors, the two intercepts are just means.*

*So an ANOVA reports each mean and a p-value that says at least two are significantly different. A regression reports only one mean(as an intercept), and the differences between that one and all other means, but the p-values evaluate those specific comparisons.*

*It's all the same model, the same information, but presented in different ways. Understand what the model tells you in each way, and you are empowered.*

Source:<https://www.theanalysisfactor.com/why-anova-and-linear-regression-are-the-same-analysis/>  
(<https://www.theanalysisfactor.com/why-anova-and-linear-regression-are-the-same-analysis/>)

## Summary

We now have a fairly good model for the trend in accuracy for a specific training test. Because we didn't find much difference with the LSAT and Figure test, this analysis could be useful in the future for describing behavior over time, and serve as a benchmark with which to compare further experiments.

In [ ]: