D_worksheet

February 19, 2024

1 Worksheet D

variationalform https://variationalform.github.io/

Just Enough: progress at pace https://variationalform.github.io/

https://github.com/variationalform

Simon Shaw https://www.brunel.ac.uk/people/simon-shaw.

This document uses python

and also makes use of LaTeX

in Markdown

1.1 What this is about:

This worksheet is based on the material in the notebook

- regress: polynomial and logistic regression.

Note that while the 'lecture' notebooks are prefixed with `1_`, `2_` and so on, to indicate the order in which they should be studied, the worksheets are prefixed with `A_`, `B_`, …

[1]:
```
# useful imports
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import linear_model
```

1.1.1 Exercise 1

A straight line has gradient $m = 2$ and y-intercept $c = 4$. Sketch it, and determine the value of x for which $y = 8$.

[2]:
```
# Answer here - create more cells as necessary
```

1.1.2 Exercise 2

A straight line with gradient m passes though the point (x_0, y_0). Show that $y - y_0 = m(x - x_0)$. This is called the *point slope* form.

[3]:
```
# Answer here - create more cells as necessary
```

1.1.3 Exercise 3

A line with gradient $m = 5$ passes through $(x, y) = (-1, 2)$. Find the equation of the line in the form $y = mx + c$.

[4]:
```
# Answer here - create more cells as necessary
```

1.1.4 Exercise 4

Recall Anscombe's data set. We used the following code to split it into its four sub-sets, and we also produced scatterplots, as shown below for the first data subset.

```
dfa = sns.load_dataset('anscombe')
print("The size of Anscombe's data set is:", dfa.shape)
dfa.dataset.unique()
dfa1 = dfa.loc[dfa['dataset'] == 'I']
dfa2 = dfa.loc[dfa['dataset'] == 'II']
dfa3 = dfa.loc[dfa['dataset'] == 'III']
dfa4 = dfa.loc[dfa['dataset'] == 'IV']
sns.scatterplot(data=dfa1, x="x", y="y")
dfa1.describe()
```

Implement linear regression for this fist dataset `dfa1`. Then implement ridge and LASSO regression. Plot your regreesion lines on the same plot and include the underlying data.

You might find the following useful:

```
dfreg = dfa1.sort_values('x', ascending = True).reset_index(drop=True)
```

After this you can either reassign `dfa1 = dfreg` or work directly with `dfreg`.

[5]:
```
# Answer here - create more cells as necessary
```

1.1.5 Exercise 5

Repeat Exercise 4 but with `dfa2`.

[6]:
```
# Answer here - create more cells as necessary
```

1.1.6 Exercise 6

Repeat Exercise 5 but with `dfa3`.

[7]:
```
# Answer here - create more cells as necessary
```

1.1.7 Exercise 7

Repeat Exercise 6 but with `dfa4`.

[8]:
```
# Answer here - create more cells as necessary
```

2 Outline Suggested Solutions

The following are suggestions for solutions of the above problems. Please have a go first though before looking at these.

2.0.1 Solution 1

The line passes through the vertical axis at $y = 4$ and climbs a vertical distance of 2 for every unit of horizontal distance. When $y = 8$, we have from $y = mx + c$ that $x = (y - c)/m = 2$.

2.0.2 Solution 2

At another (arbitrary but distinct) point (x, y) the gradient is $m = (y - y_0)/(x - x_0)$. The *point slope* form follows.

2.0.3 Solution 3

$y - y_0 = m(x - x_0)$ with $(x_0, y_0) = (-1, 2)$ and $m = 5$. Therefore

$$y = mx - mx_0 + y_0 = 5x - (5 \times -1 - 2) = 5x + 7.$$

2.0.4 Solution 4

An outline solution to Exercise 4 follows.

[9]:
```
dfa = sns.load_dataset('anscombe')
print("The size of Anscombe's data set is:", dfa.shape)
```

```
The size of Anscombe's data set is: (44, 3)
```

[10]:
```
dfa.dataset.unique()
```

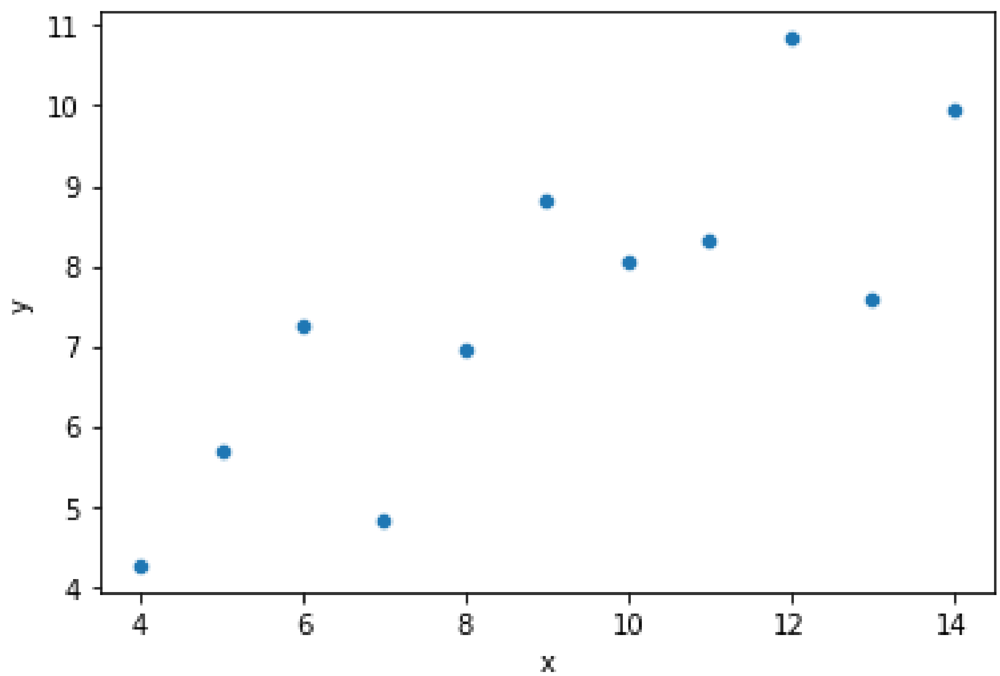
[10]:
```
array(['I', 'II', 'III', 'IV'], dtype=object)
```

[11]:
```
dfa1 = dfa.loc[dfa['dataset'] == 'I']
dfa2 = dfa.loc[dfa['dataset'] == 'II']
dfa3 = dfa.loc[dfa['dataset'] == 'III']
dfa4 = dfa.loc[dfa['dataset'] == 'IV']
```

[12]:
```
sns.scatterplot(data=dfa1, x="x", y="y")
dfa1.describe()
```

[12]:

	x	y
count	11.000000	11.000000
mean	9.000000	7.500909
std	3.316625	2.031568
min	4.000000	4.260000
25%	6.500000	6.315000
50%	9.000000	7.580000
75%	11.500000	8.570000
max	14.000000	10.840000



[13]:
```
dfa1.head()
```

[13]:

	dataset	x	y
0	I	10.0	8.04
1	I	8.0	6.95
2	I	13.0	7.58
3	I	9.0	8.81
4	I	11.0	8.33

[14]:
```
dfreg = dfa1.sort_values('x', ascending = True).reset_index(drop=True)
```

[15]:
```
dfreg.head()
```

[15]:

	dataset	x	y
0	I	4.0	4.26
1	I	5.0	5.68
2	I	6.0	7.24
3	I	7.0	4.82
4	I	8.0	6.95

[16]:

```
X_vals = dfreg.iloc[:,1].values.reshape(-1,1)
y_vals = dfreg.iloc[:,2].values.reshape(-1,1)
#print(X_vals,'\n', y_vals)
```

[17]:

```
# standard (usual) regression
reg_usual = linear_model.LinearRegression()
reg_usual.fit(X_vals, y_vals)
# Make predictions
y_pred_usual = reg_usual.predict(X_vals)
print('reg_usual_coef_ = ', reg_usual.coef_)
print('reg_usual_intercept_ = ', reg_usual.intercept_)
```

```
reg_usual_coef_ =  [[0.50009091]]
reg_usual_intercept_ =  [3.00009091]
```

[18]:

```
# ridge regression
reg_ridge = linear_model.Ridge(alpha=0.5)
reg_ridge.fit(X_vals, y_vals)
# Make predictions
y_pred_ridge = reg_ridge.predict(X_vals)
print('reg_ridge_coef_ = ', reg_ridge.coef_)
print('reg_ridge_intercept_ = ', reg_ridge.intercept_)
```

```
reg_ridge_coef_ =  [[0.49782805]]
reg_ridge_intercept_ =  [3.0204566]
```

[19]:

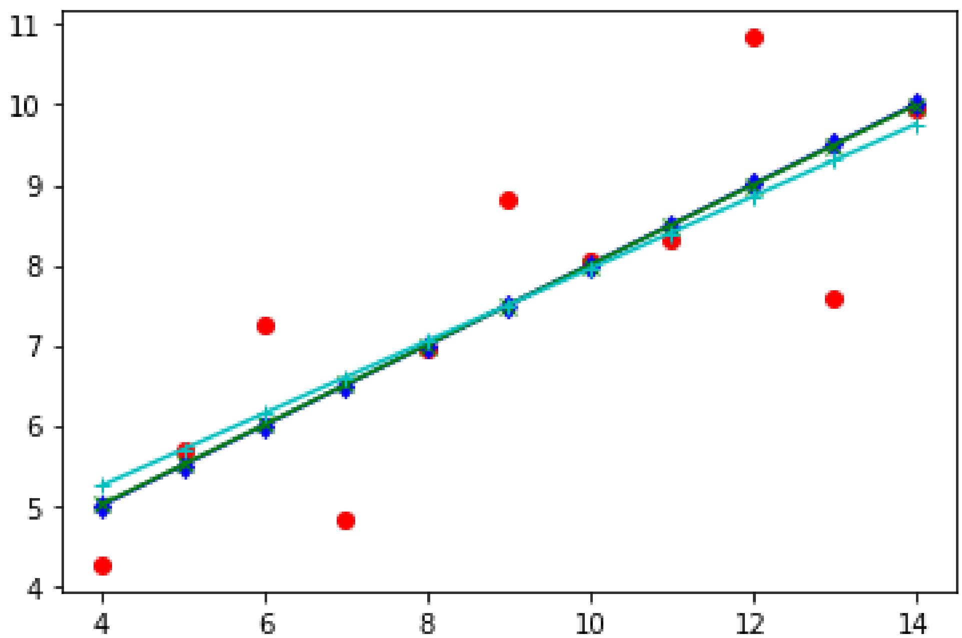
```
# LASSO regression
reg_lasso = linear_model.Lasso(alpha=0.5)
reg_lasso.fit(X_vals, y_vals)
# Make predictions
y_pred_lasso = reg_lasso.predict(X_vals)
print('reg_lasso_coef_ = ', reg_lasso.coef_)
print('reg_lasso_intercept_ = ', reg_lasso.intercept_)
```

```
reg_lasso_coef_ =  [0.45009091]
reg_lasso_intercept_ =  [3.45009091]
```

[20]:

```
plt.plot(X_vals,y_vals,'.r',marker='o')
plt.plot(X_vals,y_pred_usual,'b',marker='d')
plt.plot(X_vals,y_pred_ridge,'g',marker='x')
plt.plot(X_vals,y_pred_lasso,'c',marker='+')
```

[20]: [<matplotlib.lines.Line2D at 0x7f97e16f95f8>]



2.0.5 Solution 5

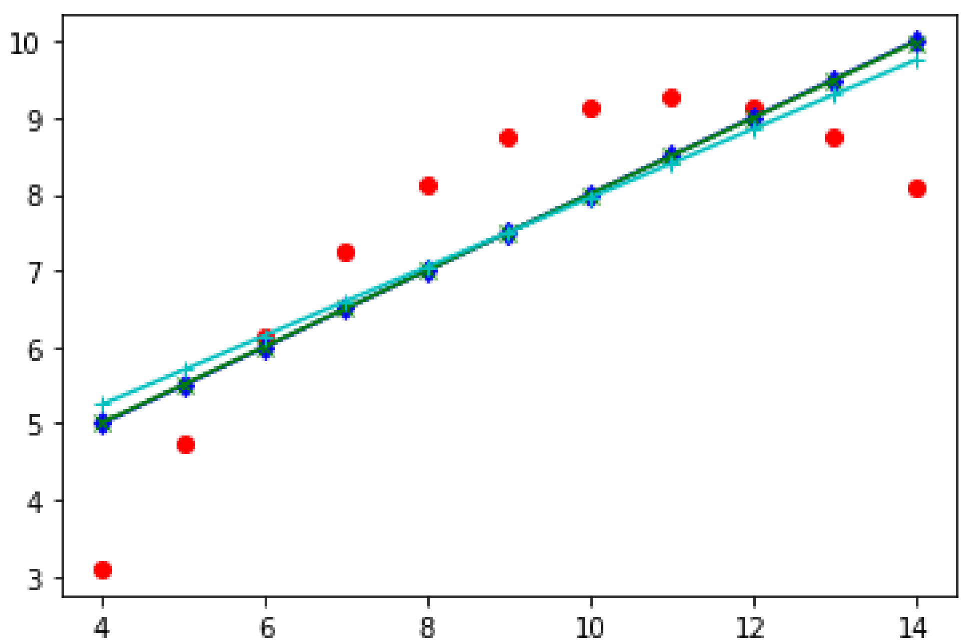
An outline solution to Exercise 5 follows.

[21]:
```
dfreg = dfa2.sort_values('x', ascending = True).reset_index(drop=True)
X_vals = dfreg.iloc[:,1].values.reshape(-1,1)
y_vals = dfreg.iloc[:,2].values.reshape(-1,1)

# standard (usual) regression
reg_usual = linear_model.LinearRegression()
reg_usual.fit(X_vals, y_vals)
# Make predictions
y_pred_usual = reg_usual.predict(X_vals)
print('reg_usual_coef_ = ', reg_usual.coef_)
print('reg_usual_intercept_ = ', reg_usual.intercept_)

# ridge regression
reg_ridge = linear_model.Ridge(alpha=0.5)
reg_ridge.fit(X_vals, y_vals)
# Make predictions
y_pred_ridge = reg_ridge.predict(X_vals)
print('reg_ridge_coef_ = ', reg_ridge.coef_)
print('reg_ridge_intercept_ = ', reg_ridge.intercept_)
```

```
# LASSO regression
reg_lasso = linear_model.Lasso(alpha=0.5)
reg_lasso.fit(X_vals, y_vals)
# Make predictions
y_pred_lasso = reg_lasso.predict(X_vals)
print('reg_lasso_coef_ = ', reg_lasso.coef_)
print('reg_lasso_intercept_ = ', reg_lasso.intercept_)

plt.plot(X_vals,y_vals,'.r',marker='o')
plt.plot(X_vals,y_pred_usual,'b',marker='d')
plt.plot(X_vals,y_pred_ridge,'g',marker='x')
plt.plot(X_vals,y_pred_lasso,'c',marker='+')
```

```
reg_usual_coef_ =  [[0.5]]
reg_usual_intercept_ =  [3.00090909]
reg_ridge_coef_ =  [[0.49773756]]
reg_ridge_intercept_ =  [3.02127108]
reg_lasso_coef_ =  [0.45]
reg_lasso_intercept_ =  [3.45090909]
```

[21]:
```
[<matplotlib.lines.Line2D at 0x7f97e159eb38>]
```



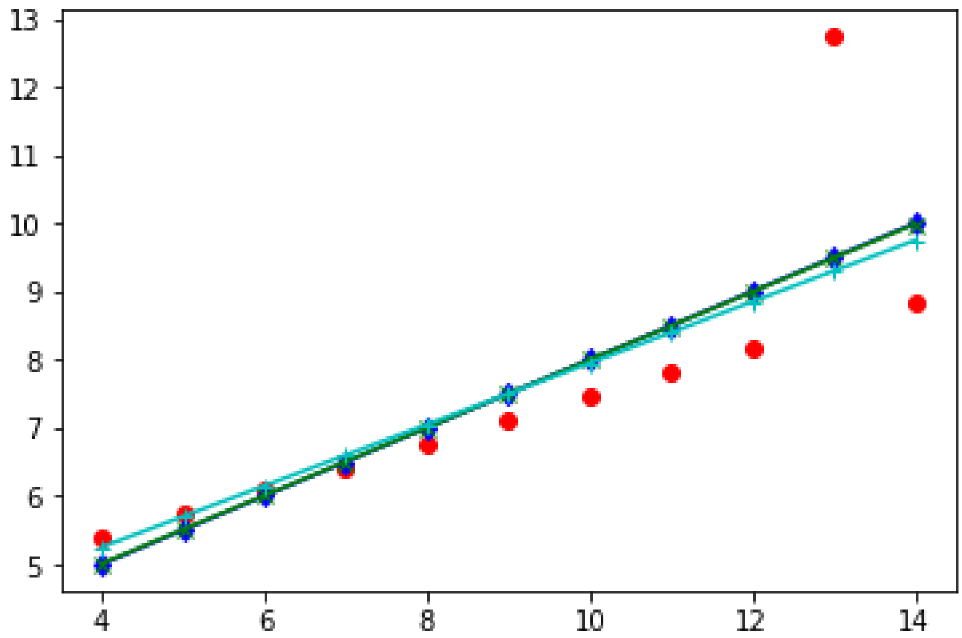
2.0.6 Solution 6

An outline solution to Exercise 6 follows.

[22]:
```
dfreg = dfa3.sort_values('x', ascending = True).reset_index(drop=True)
X_vals = dfreg.iloc[:,1].values.reshape(-1,1)
y_vals = dfreg.iloc[:,2].values.reshape(-1,1)

# standard (usual) regression
reg_usual = linear_model.LinearRegression()
reg_usual.fit(X_vals, y_vals)
# Make predictions
y_pred_usual = reg_usual.predict(X_vals)
print('reg_usual_coef_ = ', reg_usual.coef_)
print('reg_usual_intercept_ = ', reg_usual.intercept_)

# ridge regression
reg_ridge = linear_model.Ridge(alpha=0.5)
reg_ridge.fit(X_vals, y_vals)
# Make predictions
y_pred_ridge = reg_ridge.predict(X_vals)
print('reg_ridge_coef_ = ', reg_ridge.coef_)
print('reg_ridge_intercept_ = ', reg_ridge.intercept_)

# LASSO regression
reg_lasso = linear_model.Lasso(alpha=0.5)
reg_lasso.fit(X_vals, y_vals)
# Make predictions
y_pred_lasso = reg_lasso.predict(X_vals)
print('reg_lasso_coef_ = ', reg_lasso.coef_)
print('reg_lasso_intercept_ = ', reg_lasso.intercept_)

plt.plot(X_vals,y_vals,'.r',marker='o')
plt.plot(X_vals,y_pred_usual,'b',marker='d')
plt.plot(X_vals,y_pred_ridge,'g',marker='x')
plt.plot(X_vals,y_pred_lasso,'c',marker='+')
```

```
reg_usual_coef_ =  [[0.49972727]]
reg_usual_intercept_ =  [3.00245455]
reg_ridge_coef_ =  [[0.49746606]]
reg_ridge_intercept_ =  [3.02280543]
reg_lasso_coef_ =  [0.44972727]
reg_lasso_intercept_ =  [3.45245455]
```

[22]:
```
[<matplotlib.lines.Line2D at 0x7f97d1dd4668>]
```



2.0.7 Solution 7

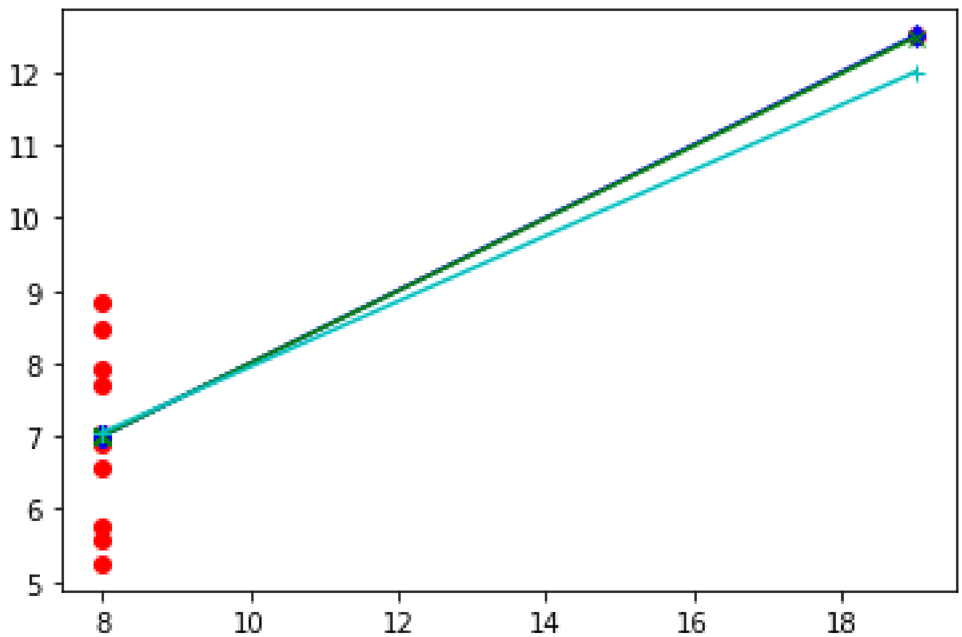
An outline solution to Exercise 7 follows.

[23]:
```
dfreg = dfa4.sort_values('x', ascending = True).reset_index(drop=True)
X_vals = dfreg.iloc[:,1].values.reshape(-1,1)
y_vals = dfreg.iloc[:,2].values.reshape(-1,1)

# standard (usual) regression
reg_usual = linear_model.LinearRegression()
reg_usual.fit(X_vals, y_vals)
# Make predictions
y_pred_usual = reg_usual.predict(X_vals)
print('reg_usual_coef_ = ', reg_usual.coef_)
print('reg_usual_intercept_ = ', reg_usual.intercept_)

# ridge regression
reg_ridge = linear_model.Ridge(alpha=0.5)
reg_ridge.fit(X_vals, y_vals)
# Make predictions
y_pred_ridge = reg_ridge.predict(X_vals)
print('reg_ridge_coef_ = ', reg_ridge.coef_)
print('reg_ridge_intercept_ = ', reg_ridge.intercept_)

# LASSO regression
reg_lasso = linear_model.Lasso(alpha=0.5)
```

```
reg_lasso.fit(X_vals, y_vals)
# Make predictions
y_pred_lasso = reg_lasso.predict(X_vals)
print('reg_lasso_coef_ = ', reg_lasso.coef_)
print('reg_lasso_intercept_ = ', reg_lasso.intercept_)

plt.plot(X_vals,y_vals,'.r',marker='o')
plt.plot(X_vals,y_pred_usual,'b',marker='d')
plt.plot(X_vals,y_pred_ridge,'g',marker='x')
plt.plot(X_vals,y_pred_lasso,'c',marker='+')
```

```
reg_usual_coef_ =  [[0.49990909]]
reg_usual_intercept_ =  [3.00172727]
reg_ridge_coef_ =  [[0.49764706]]
reg_ridge_intercept_ =  [3.02208556]
reg_lasso_coef_ =  [0.44990909]
reg_lasso_intercept_ =  [3.45172727]
```

[23]:
```
[<matplotlib.lines.Line2D at 0x7f97f0cfc048>]
```



2.1 Technical Notes, Production and Archiving

Ignore the material below. What follows is not relevant to the material being taught.

Production Workflow

- Finalise the notebook material above

- Clear and fresh run of entire notebook
- Create html slide show:
    - `jupyter nbconvert --to slides D_worksheet.ipynb`
- Set `OUTPUTTING=1` below
- Comment out the display of web-sourced diagrams
- Clear and fresh run of entire notebook
- Comment back in the display of web-sourced diagrams
- Clear all cell output
- Set `OUTPUTTING=0` below
- Save
- git add, commit and push to FML
- copy PDF, HTML etc to web site
    - git add, commit and push
- rebuild binder

Some of this originated from https://stackoverflow.com/questions/38540326/save-html-of-a-jupyter-notebook-from-within-the-notebook These lines create a back up of the notebook. They can be ignored. At some point this is better as a bash script outside of the notebook

[24]:
```
%%bash
NBROOTNAME='D_worksheet'
OUTPUTTING=1

if [ $OUTPUTTING -eq 1 ]; then
  jupyter nbconvert --to html $NBROOTNAME.ipynb
  cp $NBROOTNAME.html ../backups/$(date +"%m_%d_%Y-%H%M%S")_$NBROOTNAME.html
  mv -f $NBROOTNAME.html ./formats/html/

  jupyter nbconvert --to pdf $NBROOTNAME.ipynb
  cp $NBROOTNAME.pdf ../backups/$(date +"%m_%d_%Y-%H%M%S")_$NBROOTNAME.pdf
  mv -f $NBROOTNAME.pdf ./formats/pdf/

  jupyter nbconvert --to script $NBROOTNAME.ipynb
  cp $NBROOTNAME.py ../backups/$(date +"%m_%d_%Y-%H%M%S")_$NBROOTNAME.py
  mv -f $NBROOTNAME.py ./formats/py/
else
  echo 'Not Generating html, pdf and py output versions'
fi
```

```
Not Generating html, pdf and py output versions
```