GGPLOT IN PYTHON

-By Sarah Masud

INTRODUCTION:

This ppt will cover the basic functions of ggplot in python. This will help beginners to understand what the functions mean and how to use them.

SELF HELP:

If you don’t remember the function or wish to know more about it, you can use the help function in python by simply typing the function name followed by a ?

EXAMPLE:

INPUT:

```
In [13]: import numpy as np
         np.sin?
```

OUTPUT:

```
Type:            ufunc
String form:     <ufunc 'sin'>
File:            ~/anaconda/lib/python2.7/site-packages/numpy/__init__.py
Docstring:
sin(x[, out])

Trigonometric sine, element-wise.

Parameters
----------
x : array_like
    Angle, in radians (:math:`2 \pi` rad equals 360 degrees).

Returns
-------
y : array_like
    The sine of each element of x.

See Also
--------
arcsin, sinh, cos

Notes
-----
The sine is one of the fundamental functions of trigonometry (the
```

PREREQUISITES INSTALLED:

- pip
- easy install
- matplotlib
- pandas
- numpy
- scipy
- statmodels

INSTALL ggplot UNDER PYTHON:

FOR LINUX

Method 1: pip install ggplot

Method 2: pip install git+git://github.com/yhat/ggplot.git

SOURCE OF DATA:

Big Diamonds data set is used through the presentation. You can download the data set from:
https://github.com/SolomonMg/diamonds-data

HOW TO OBTAIN THE CSV:

METHOD 1: Read the RDA file in R and writeback as CSV.

METHOD 2: Use rpy2.

WHAT IS ggplot?

Created by H. Wickman, ggplot provides an easy interface to generate state of art visualizations.

Written originally for R, its success enabled it be used for Python as well.

COMPONENTS OF ggplot:

- ggplot API- Used to implement the plots.
- Data- Uses data as Data Frames as in pandas.
- Aesthetics- How the axes and theme looks.
- Layer- what information is annotated on top of basic plot.

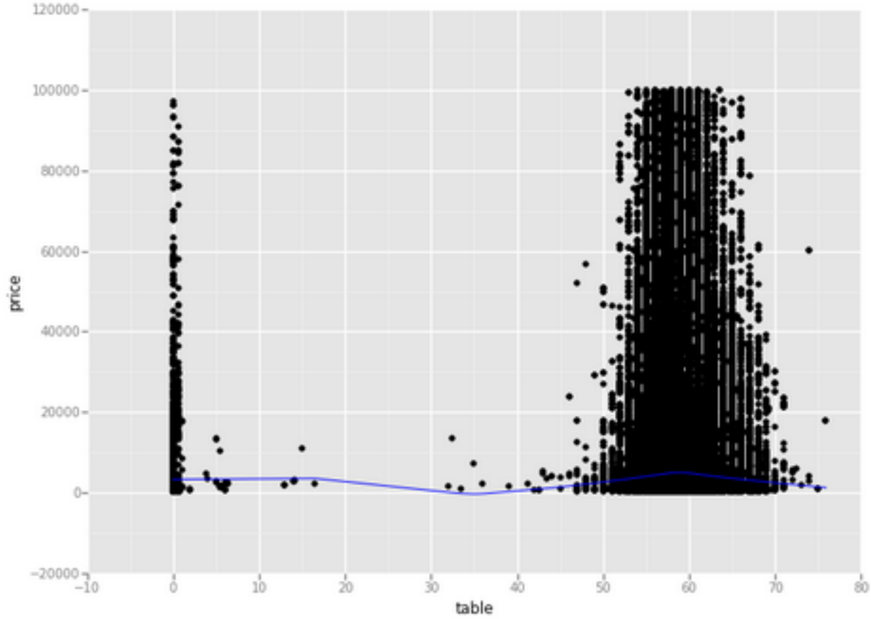
TIME TAKEN TO EXECUTE A FUNCTION:

```
from datetime import datetime
startTime = datetime.now()

#do something

print datetime.now() - startTime
```

```
from datetime import datetime
startTime = datetime.now()
ggplot(diamonds, aes('table', 'price')) + geom_point()+stat_smooth(colour='blue',se=False)
```




```
<ggplot: (8740400790369)>
```

```
print datetime.now() - startTime
```

```
0:00:35.785688
```

Source: http://stackoverflow.com/questions/6786990/find-out-time-it-took-for-a-python-script-to-complete-execution

INPUT THE DATA:

Import necessary packages.

```
import pandas as pd
import numpy as np
from ggplot import *
import matplotlib.pyplot as plt
%matplotlib inline
```

Read Data:

```
In [3]: diamonds=pd.read_csv('/home/sarah/diamonds.csv')
```

EXPLORE THE DATA:

1. **len()**- Number of rows in the dataset

```
In [16]: len(diamonds)

Out[16]: 598024
```

2. **column()**- What are names of the columns.

```
In [17]: diamonds.columns

Out[17]: Index([u'carat', u'cut', u'color', u'clarity', u'table', u'depth', u'cert',
                u'measurements', u'price', u'x', u'y', u'z'],
               dtype='object')
```

WHAT DOES THE COLUMNS CONTAIN:

1. **Carat**- Weight of the diamond (1 carat=0.2g)
2. **Cut**- Quality of cut
3. **Color**- Color of diamond (J-worst D-best)
4. **Clarity**- A measure of how clear the diamond is.
5. **Cert**- The level of certification granted.
6. **x**- Length in mm.
7. **y**- Breadth in mm.
8. **z**- Height in mm.
9. **Measurement**- Volume in terms of x*y*z.
10. **Table**- Width of top of diamond relative to widest point.
11. **Depth**- Numerically = (2*z) /(x+y)

3. **head()/tail()**- To know the first few & last few values, respectively.

NOTE: The dataset contains both quantitative(numeric) and qualitative fields.

In [3]:
```
diamonds.head()
```

Out[3]:

	carat	cut	color	clarity	table	depth	cert	measurements	price	x	y	z
0	0.25	V.Good	K	I1	59	63.7	GIA	3.96 x 3.95 x 2.52	NaN	3.96	3.95	2.52
1	0.23	Good	G	I1	61	58.1	GIA	4.00 x 4.05 x 2.30	NaN	4.00	4.05	2.30
2	0.34	Good	J	I2	58	58.7	GIA	4.56 x 4.53 x 2.67	NaN	4.56	4.53	2.67
3	0.21	V.Good	D	I1	60	60.6	GIA	3.80 x 3.82 x 2.31	NaN	3.80	3.82	2.31
4	0.31	V.Good	K	I1	59	62.2	EGL	4.35 x 4.26 x 2.68	NaN	4.35	4.26	2.68

In [4]:
```
diamonds.tail()
```

Out[4]:

	carat	cut	color	clarity	table	depth	cert	measurements	price	x	y	z
598019	3.02	Ideal	E	VVS2	58.0	59.8	HRD	9.43 x 9.51 x 5.66	99930	9.43	9.51	5.66
598020	5.01	V.Good	I	VVS2	63.5	61.5	IGI	10.78 x 10.89 x 6.68	99942	10.78	10.89	6.68
598021	3.43	Ideal	F	VS2	54.0	62.7	GIA	9.66 x 9.61 x 6.05	99960	9.66	9.61	6.05
598022	3.01	V.Good	E	VS1	58.0	62.9	GIA	9.15 x 9.19 x 5.77	99966	9.15	9.19	5.77
598023	4.13	Ideal	H	IF	56.0	62.5	IGI	10.27 x 10.19 x 6.4	99990	10.27	10.19	6.40

4. **Random selection**- To see data values at random.

In [24]:
```
rows = np.random.choice(diamonds.index.values, 0.00001*len(diamonds))
print(rows)
sampled_df = diamonds.ix[rows]
```

```
[186637  26608 267520 239842 177549]
```

In [6]:
```
sampled_df
```

Out[6]:

	carat	cut	color	clarity	table	depth	cert	measurements	price	x	y	z
396523	1.15	Ideal	E	SI2	59.0	59.8	GIA	6.81 x 6.85 x 4.08	6963	6.81	6.85	4.08
178386	0.70	V.Good	I	SI2	52.5	64.1	IGI	5.56 x 5.58 x 3.57	1499	5.56	5.58	3.57
576698	3.30	Ideal	L	VVS2	57.0	62.4	OTHER	9.47 x 9.52 x 5.92	29239	9.47	9.52	5.92
94151	0.50	V.Good	H	SI2	66.0	58.8	EGL USA	5.02 x 5.04 x 2.96	866	5.02	5.04	2.96
423366	1.00	Ideal	E	VVS2	57.0	63.3	GIA	3.99 x 6.29 x 6.32	8699	3.99	6.29	6.32

5. **Describe()**- Give the mathematical details of fields with numerical value.

```
In [7]: diamonds.describe()
```

Out[7]:

	carat	table	depth	price	x	y	z
count	598024.000000	598024.000000	598024.000000	597311.000000	596209.000000	596172.000000	595480.000000
mean	1.071297	57.631077	61.063683	8753.017974	5.990771	6.198671	4.033430
std	0.812696	4.996892	7.604342	13017.567760	1.530936	1.485891	1.240951
min	0.200000	0.000000	0.000000	300.000000	0.150000	1.000000	0.040000
25%	0.500000	56.000000	61.000000	1220.000000	4.740000	4.970000	3.120000
50%	0.900000	58.000000	62.100000	3503.000000	5.780000	6.050000	3.860000
75%	1.500000	59.000000	62.700000	11174.000000	6.970000	7.230000	4.610000
max	9.250000	75.900000	81.300000	99990.000000	13.890000	13.890000	13.180000

NOTE: The mean of x,y are approximately same. Do diamonds have proportionate length/breadth?

7. **unique()**- To know the unique(1 or many) values that make up the dataset.

```
In [8]: diamonds['clarity'].unique()

Out[8]: array(['I1', 'I2', 'SI2', 'SI1', 'VS2', 'VVS2', 'VS1', 'IF', 'VVS1'], dtype=object)

In [9]: diamonds['cert'].unique()

Out[9]: array(['GIA', 'EGL', 'IGI', 'EGL USA', 'OTHER', 'EGL Intl.', 'AGS', 'HRD',
               'EGL ISRAEL'], dtype=object)

In [33]: diamonds['cut'].unique()

Out[33]: array(['V.Good', 'Good', 'Ideal'], dtype=object)

In [34]: diamonds['color'].unique()

Out[34]: array(['G', 'K', 'J', 'H', 'F', 'I', 'D', 'E', 'L'], dtype=object)
```

PREPARING DATA:

1.Check for null values.

```
In [10]: len(diamonds)
Out[10]: 598024

In [11]: diamonds2=diamonds.dropna(how='any')

In [12]: len(diamonds2)
Out[12]: 593784

In [15]: print "No of Null value= "+ str (len(diamonds)-len(diamonds2))
No of Null value= 4240
```

2. Check for zero price values.

```
In [47]: df=diamonds[diamonds['price']==0]

In [48]: df
Out[48]:
```

	carat	cut	color	clarity	depth	table	price	x	y	z

```
In [ ]: #hence there are no diamonds with missing price value
```

3. Obtain clean data set by removing null values.

In [19]:
```
diamonds=diamonds.dropna(how='any')
```

In [20]:
```
len(diamonds)
```

Out[20]: 593784

In [17]:
```
diamonds.head(2)
```

Out[17]:

	carat	cut	color	clarity	table	depth	cert	measurements	price	x	y	z
493	0.24	V.Good	G	SI1	61	58.9	GIA	4.09 x 4.10 x 2.41	300	4.09	4.10	2.41
494	0.31	V.Good	K	SI2	59	60.2	GIA	4.40 x 4.42 x 2.65	300	4.40	4.42	2.65

In [18]:
```
diamonds.tail(2)
```

Out[18]:

	carat	cut	color	clarity	table	depth	cert	measurements	price	x	y	z
598022	3.01	V.Good	E	VS1	58	62.9	GIA	9.15 x 9.19 x 5.77	99966	9.15	9.19	5.77
598023	4.13	Ideal	H	IF	56	62.5	IGI	10.27 x 10.19 x 6.4	99990	10.27	10.19	6.40

EVALUATION OF DATA:

1. New Statistical Information:

In [30]:
```
diamonds.describe()
```

Out[30]:

	carat	table	depth	price	x	y	z
count	593784.000000	593784.000000	593784.000000	593784.000000	593784.000000	593784.000000	593784.000000
mean	1.072593	57.658755	61.091980	8755.808723	5.991952	6.200535	4.036075
std	0.813113	4.827985	7.487465	13022.108651	1.530444	1.485081	1.240932
min	0.200000	0.000000	0.000000	300.000000	0.150000	1.000000	0.040000
25%	0.500000	56.000000	61.000000	1218.000000	4.740000	4.970000	3.120000
50%	0.900000	58.000000	62.000000	3503.000000	5.780000	6.050000	3.860000
75%	1.500000	59.000000	62.700000	11186.000000	6.970000	7.230000	4.610000
max	9.250000	75.900000	81.300000	99990.000000	13.890000	13.890000	13.180000

2. Correlations

In [27]:
```
diamonds.corr()
```

Out[27]:

	carat	table	depth	price	x	y	z
carat	1.000000	0.037631	0.008883	0.856340	0.859864	0.960857	0.791658
table	0.037631	1.000000	0.423914	0.023266	0.028462	0.045617	0.031170
depth	0.008883	0.423914	1.000000	-0.002129	-0.003632	0.007346	0.031961
price	0.856340	0.023266	-0.002129	1.000000	0.719537	0.796746	0.645191
x	0.859864	0.028462	-0.003632	0.719537	1.000000	0.893783	0.482109
y	0.960857	0.045617	0.007346	0.796746	0.893783	1.000000	0.819880
z	0.791658	0.031170	0.031961	0.645191	0.482109	0.819880	1.000000

In [31]:
```
diamonds.corr()>0.8
```

Out[31]:

	carat	table	depth	price	x	y	z
carat	True	False	False	True	True	True	False
table	False	True	False	False	False	False	False
depth	False	False	True	False	False	False	False
price	True	False	False	True	False	False	False
x	True	False	False	False	True	True	False
y	True	False	False	False	True	True	True
z	False	False	False	False	False	True	True

3. The plot of density of diamond

```
In [31]: from scipy import stats

In [ ]: slope,intercept,r_value,p_value,std_err=stats.linregress(diamonds['x']*diamonds['y']*diamonds['z'],diamonds['carat'])

In [34]: slope

Out[34]: 0.0061625069995040081

In [35]: intercept

Out[35]: 0.0026328639805921483

In [38]: ggplot(diamonds, aes(y='carat',x='x*y*z')) + geom_point()+\
         scale_y_continuous(name="weight(carat)") + scale_x_continuous(name="volume(x*y*z)")+\
         geom_abline(intercept =0.0026328639805921483,  colour = "red",slope=0.0061625069995040081)
```

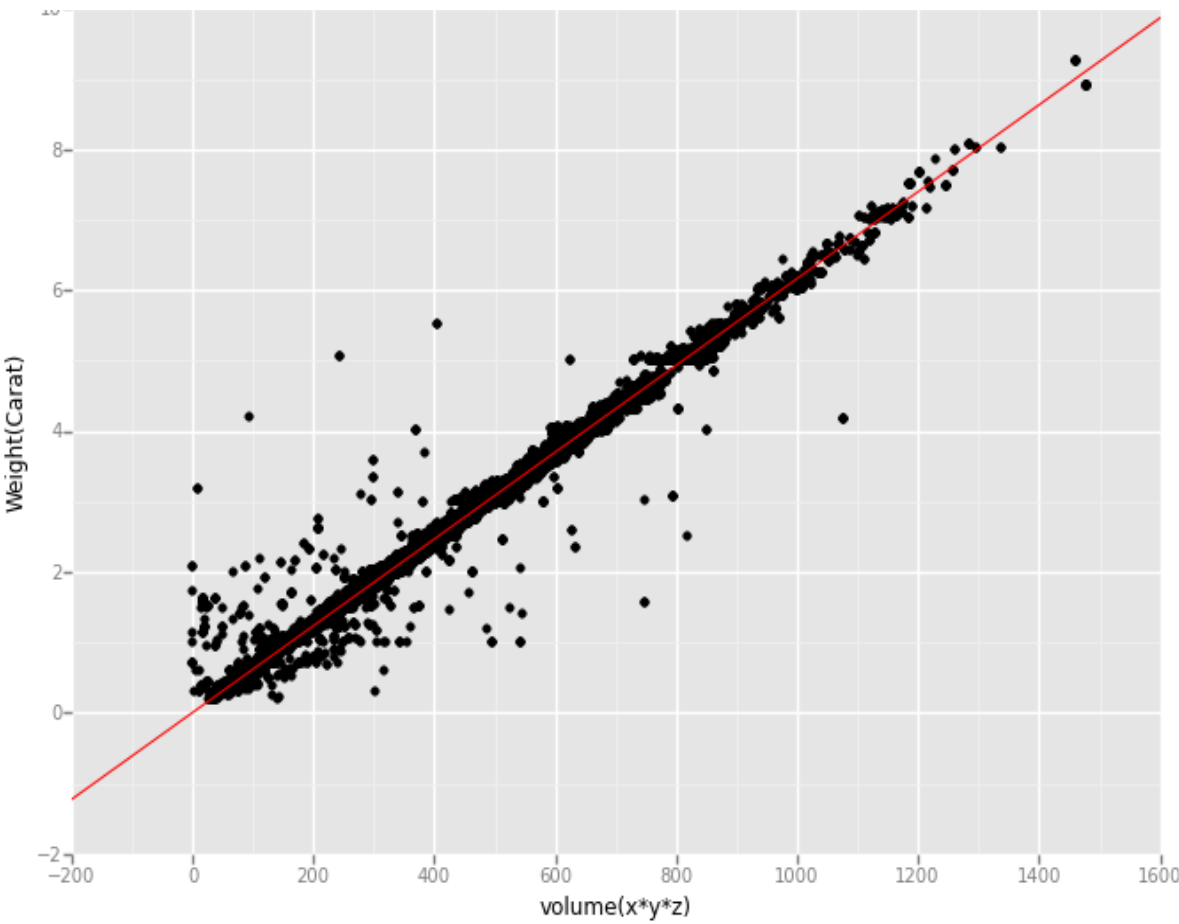
NOTE:
stat.lingress is used to calculate the components of the line of best fit of the form y=mx+c, where m=slope and c=y-intercept. The **r_value** is the regression coefficient, the **p_value** s a constant usually zero, while **std_err** is the error of estimation.

ggplot(dataset,aesthetics(y,x)- Gives us a blank coordinate system
geom_points- Plots the dataset on the blank plot.
scale_y/x_continous- Use to give name and range of the axis.
geom_abline- Draw a line of form y=mx+c.

OUTPUT:

HOW IT WORKS:

1. ggplot is invoked.
2. A blank coordinate system with labeled axes is put up.
3. The points are plotted.
4. The axis redefined and cropped.
5. The line draw as another layer on top of the points.



PRICE EVALUATION:

Diamonds are expensive!

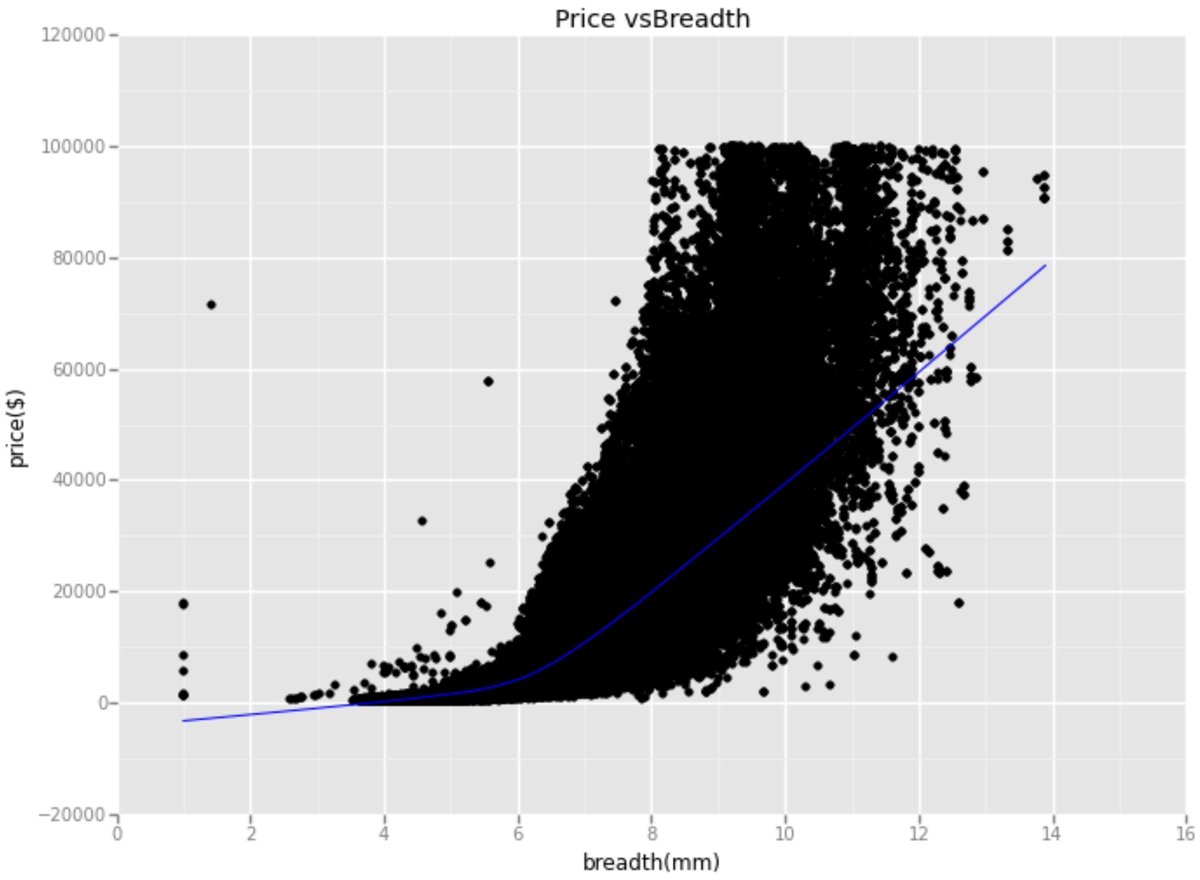
Let us try to map what factors make them so.

PRICE VS BREADTH

```
ggplot(diamonds, aes('y', 'price')) + geom_point() +\
labs(title="Price vsBreadth", x="breadth(mm)", y="price($)")+stat_smooth(colour='blue',se=False)
```

NOTE:
labs-use to label the graph and the axises.
x-lab and y-lab can also be separately used.

stats_smooth provides a mechanism to plot the line of regression and help determine the relation among the variants.

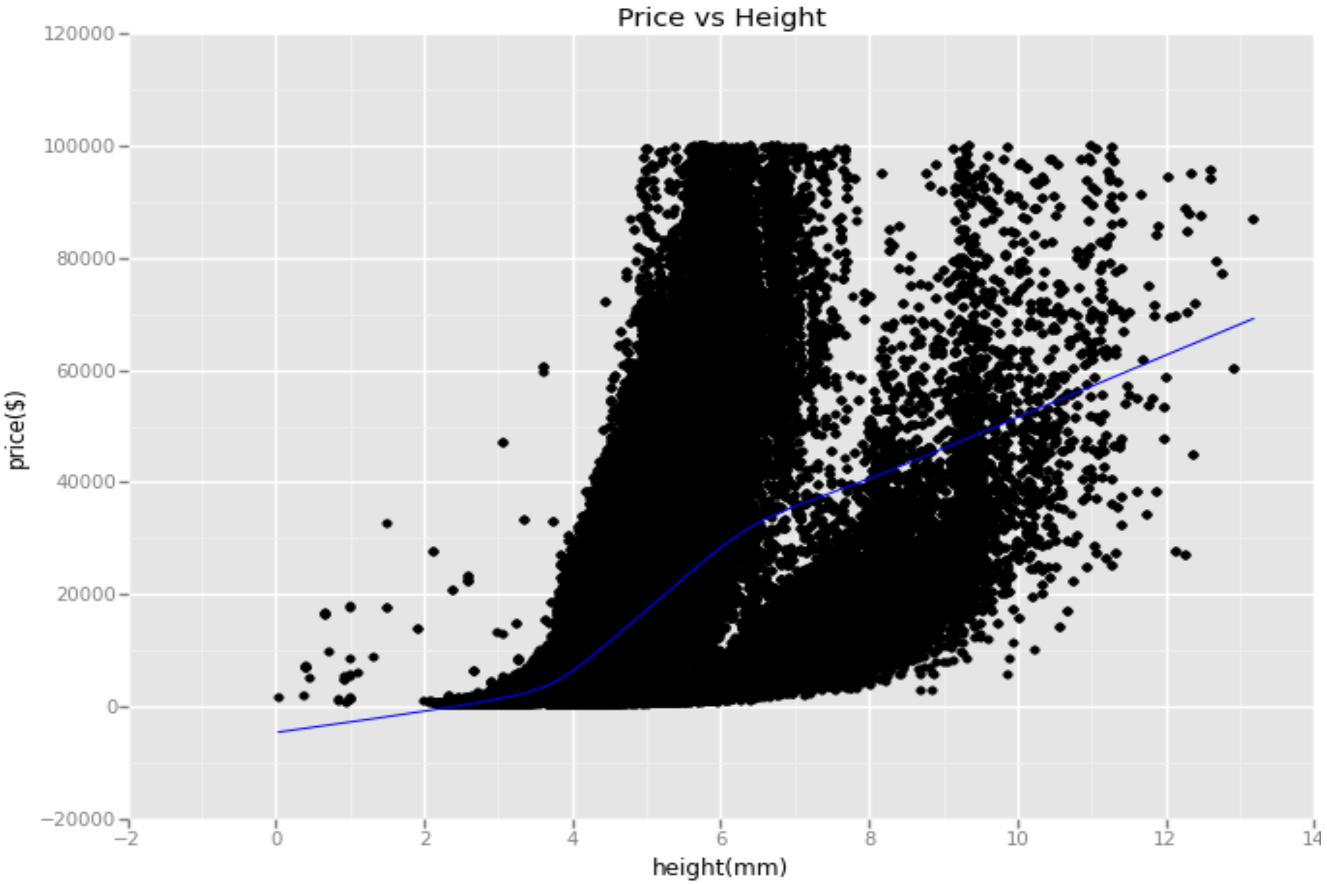


PRICE VS LENGTH

In [43]:
```
ggplot(diamonds, aes('x', 'price')) + geom_point() +\
labs(title="Price vs Length", x="length(mm)", y="price($)")+stat_smooth(colour='blue',se=False)
```

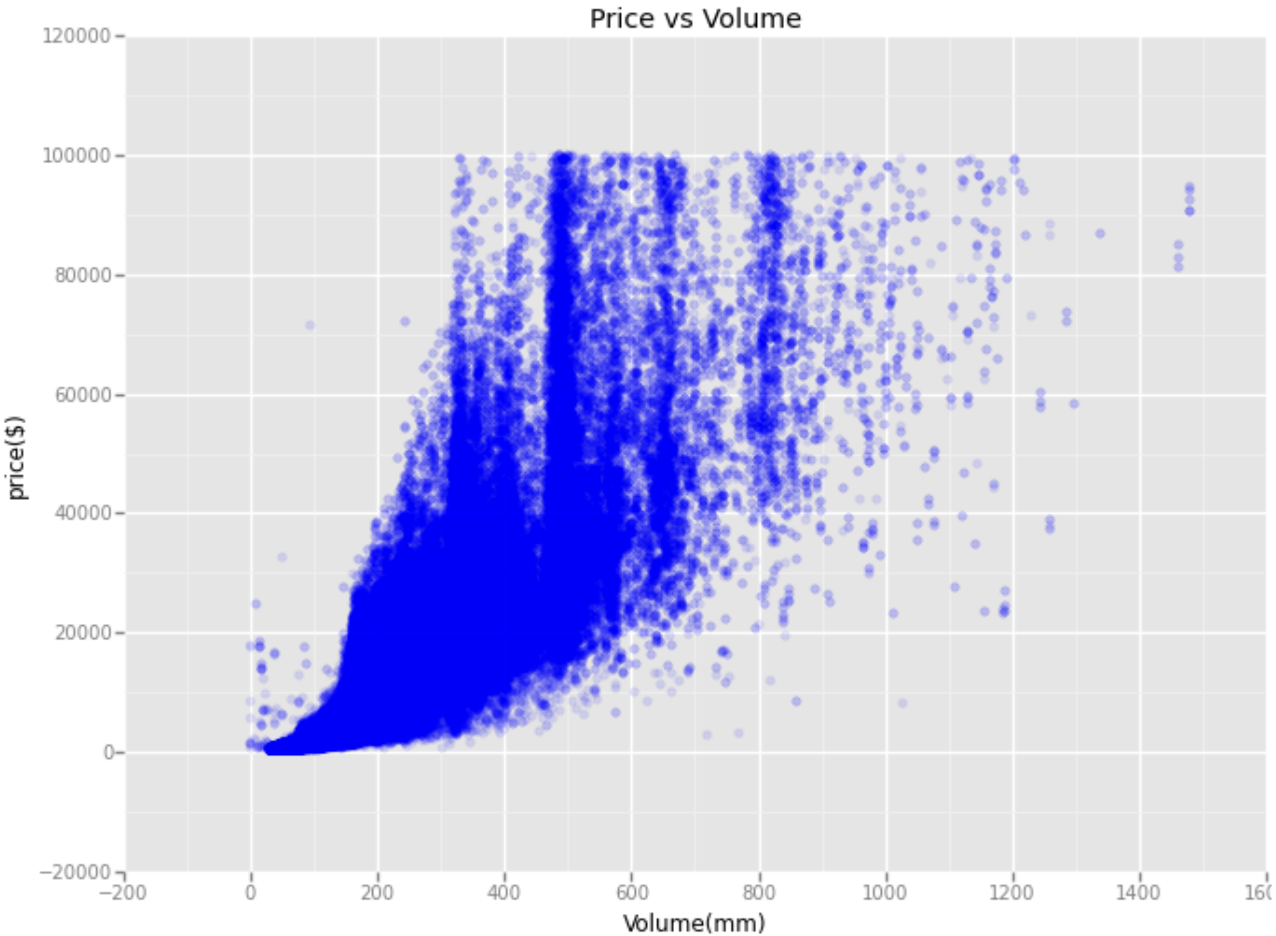
PRICE VS HEIGHT

```
ggplot(diamonds, aes('z', 'price')) + geom_point() +\
labs(title="Price vs Height", x="height(mm)", y="price($)")+stat_smooth(colour='blue',se=False)
```



PRICE VS VOLUME

```
ggplot(diamonds, aes('x*y*z', 'price')) + geom_jitter(alpha=0.1,color='blue') +\
labs(title="Price vs Volume", x="Volume(mm)", y="price($)")
```



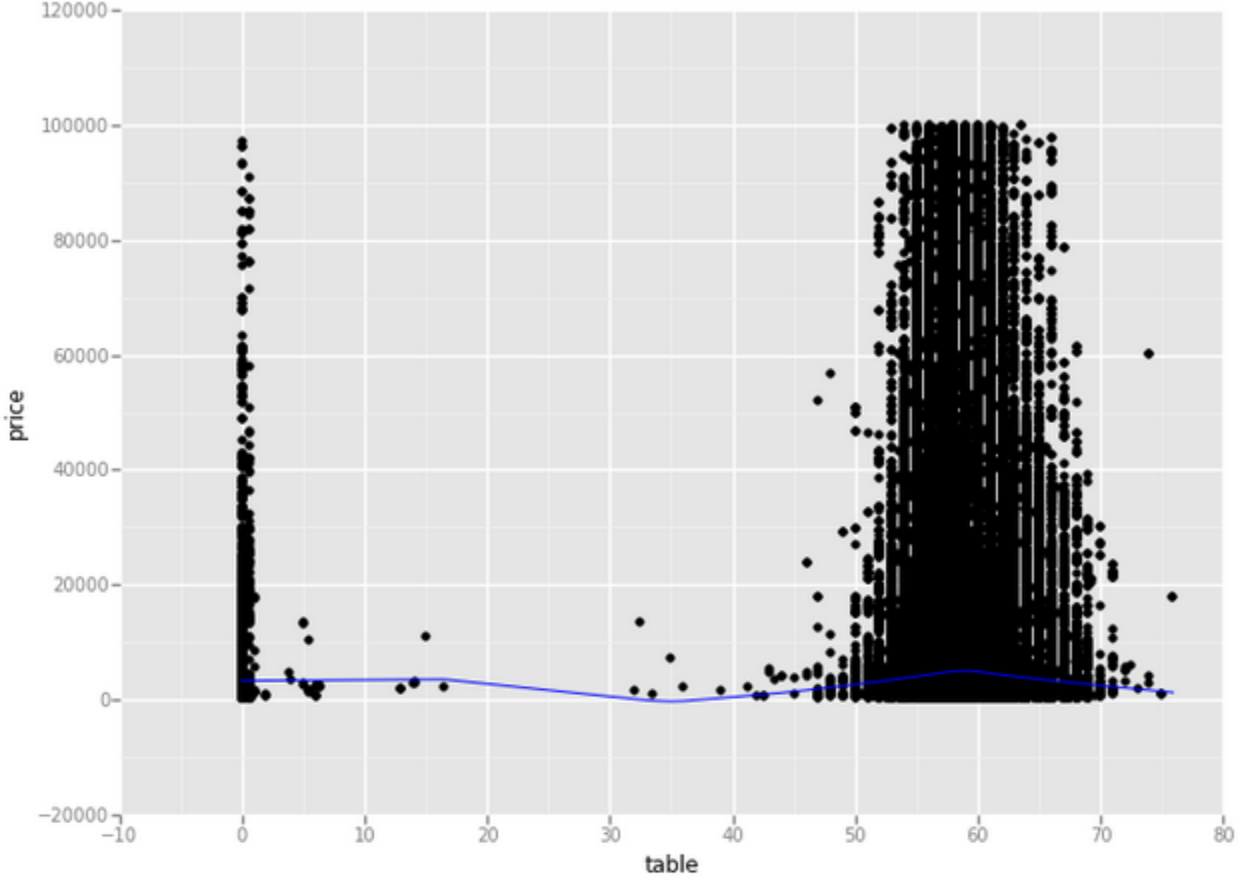
NOTE:

Geom-jitter

Over-plotting hides the number of points in each neighbourhood. We can reduce this problem by making the points more transparent.

PRICE VS TABLE

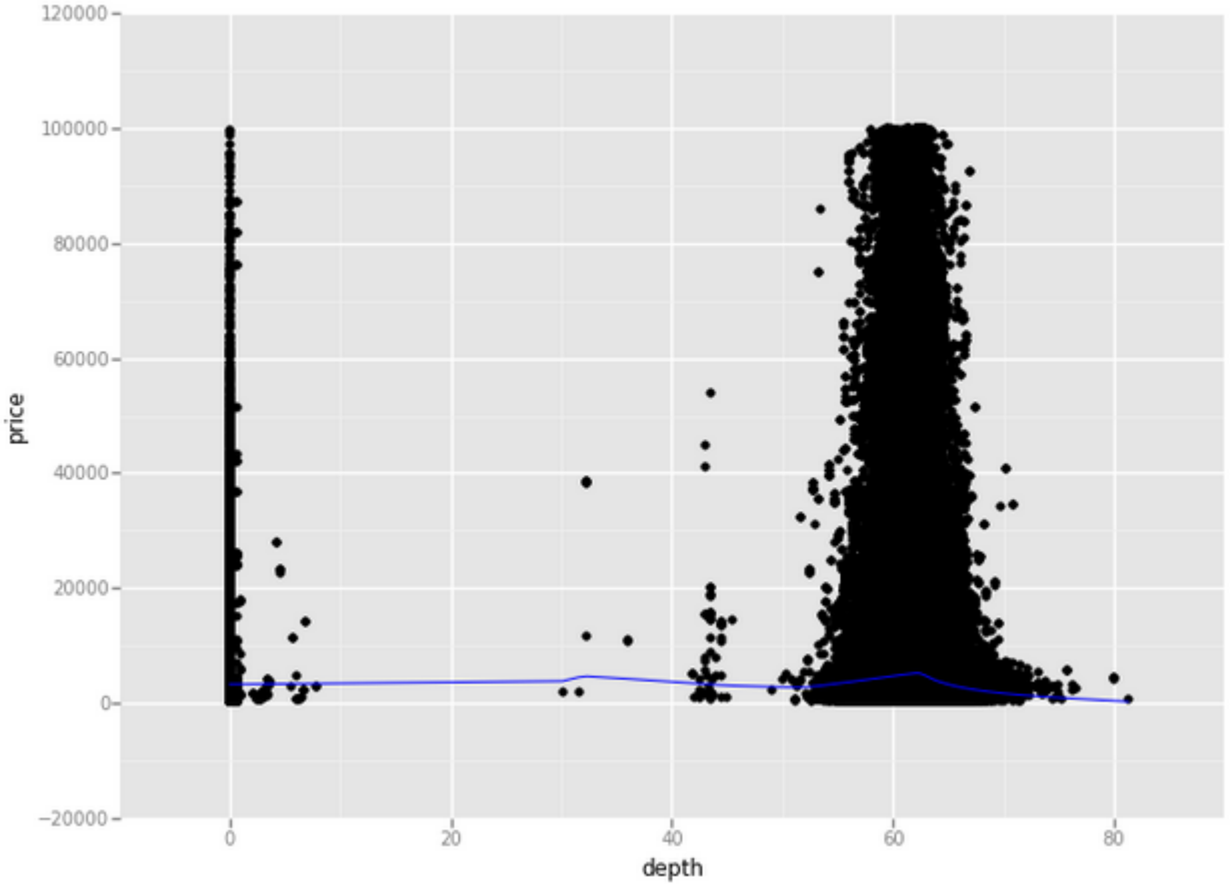
```
ggplot(diamonds, aes('table', 'price')) + geom_point()+stat_smooth(colour='blue',se=False)
```



NOTE:
A horizontal line of regression means that value of f(x) can be calculated without much consideration of the value of x.
Thus, price is not considerably affected by table and can be calculated without taking table into account.

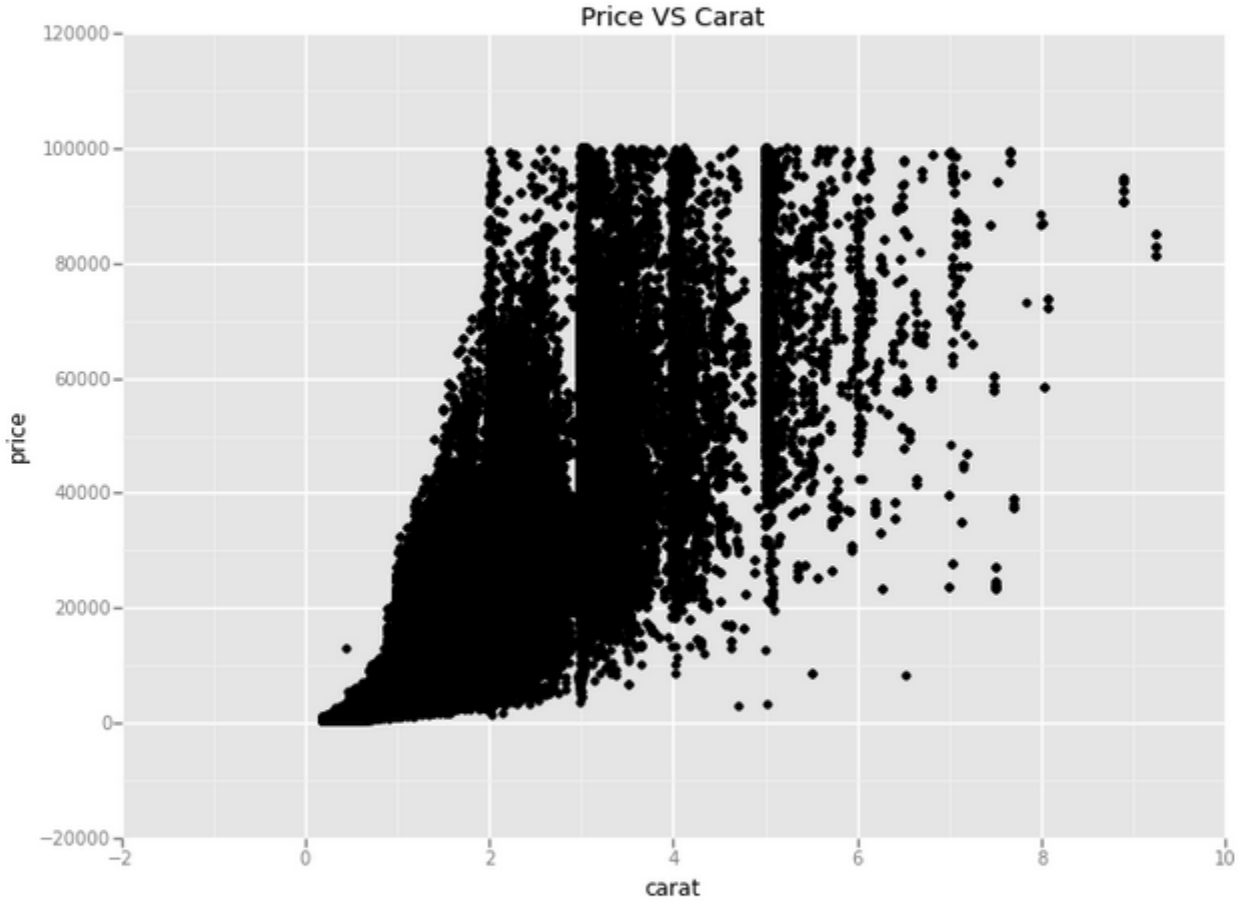
PRICE VS DEPTH

```
ggplot(diamonds, aes('depth', 'price')) + geom_point()+stat_smooth(colour='blue',se=False)
```



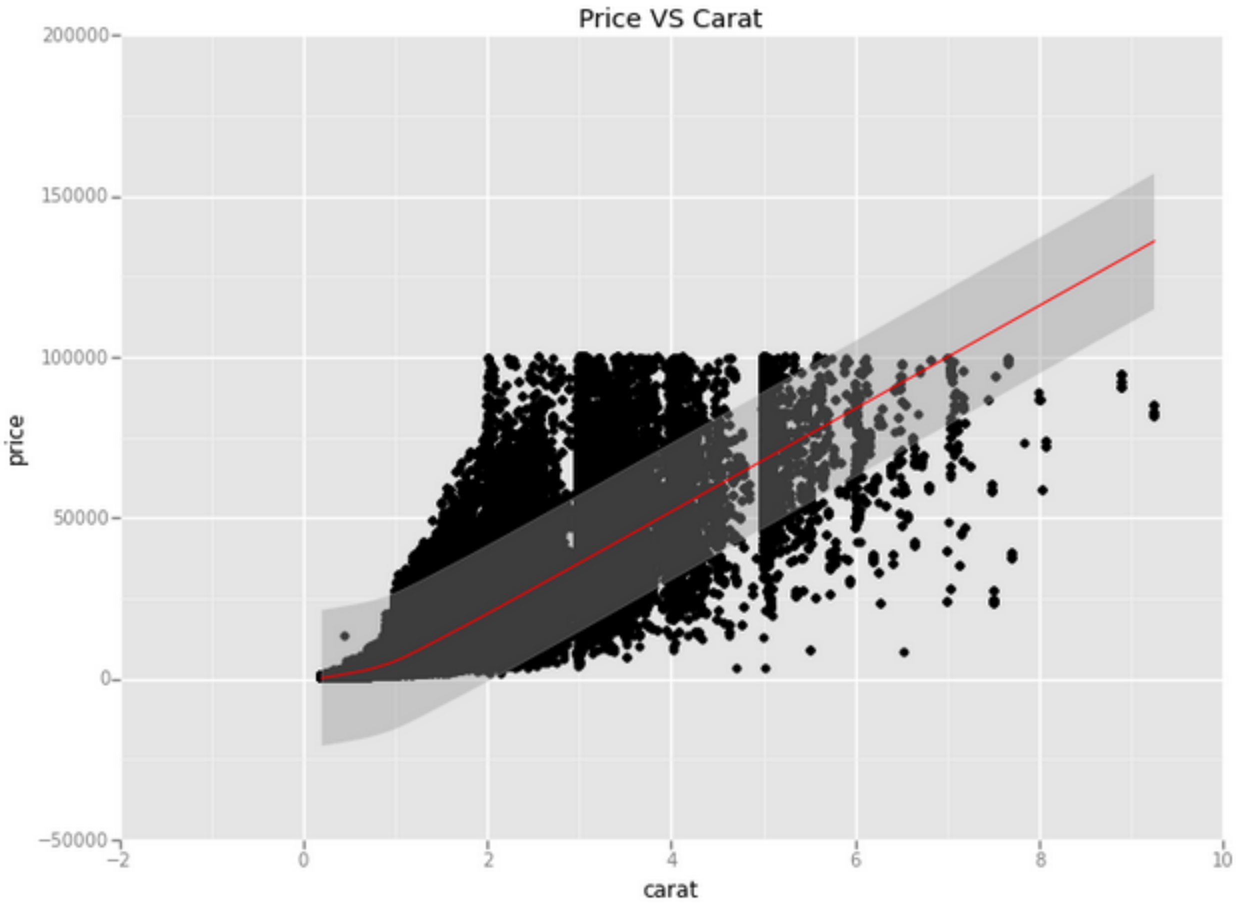
PRICE VS CARATS

```
ggplot(diamonds, aes('carat', 'price')) + geom_point()+ggtitle("Price VS Carat")
```



```
ggplot(diamonds, aes('carat', 'price')) + geom_point()+ggtitle("Price VS Carat")+stat_smooth(colour='red')
```

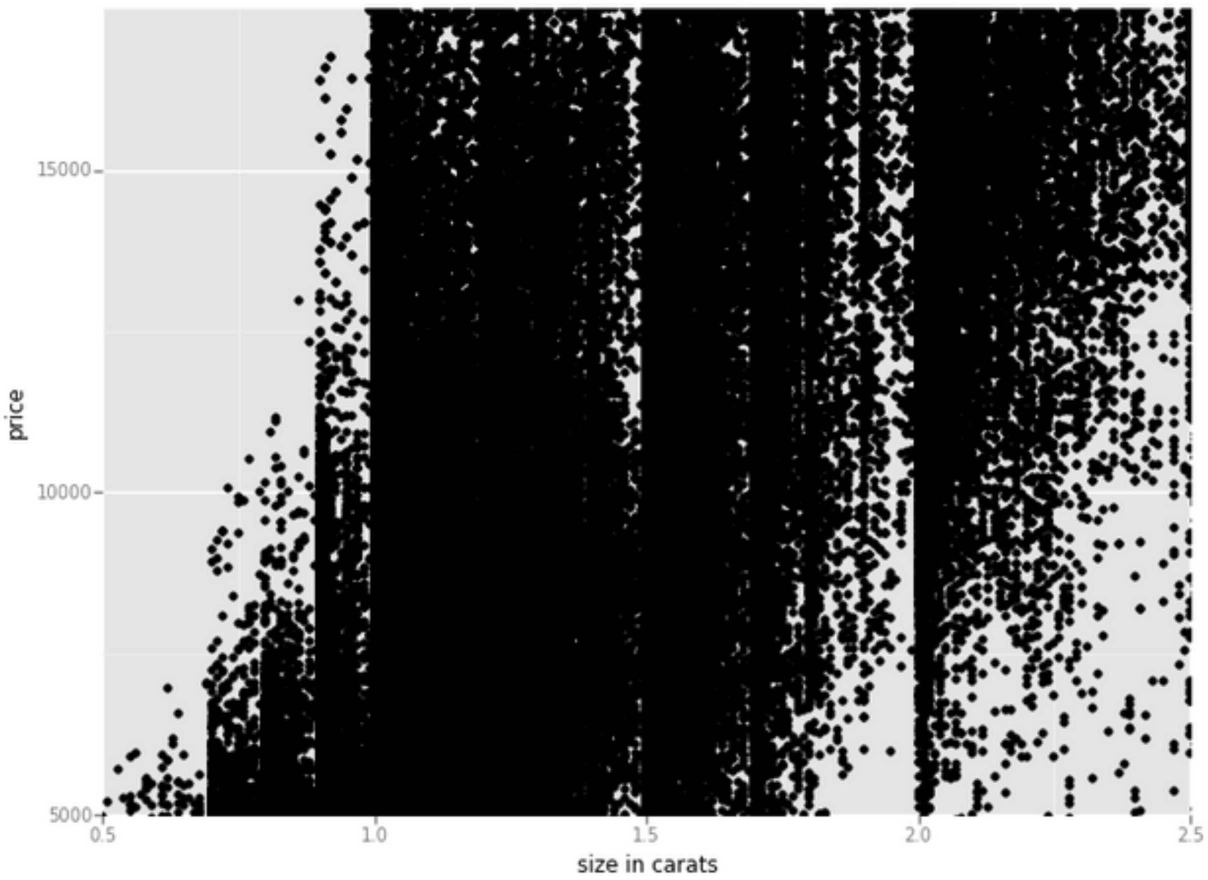
NOTE:
A quadratic line of regression signifies that value of price depends on the value of carat. But is only carat, lets see closely.



PRICE vs CARAT

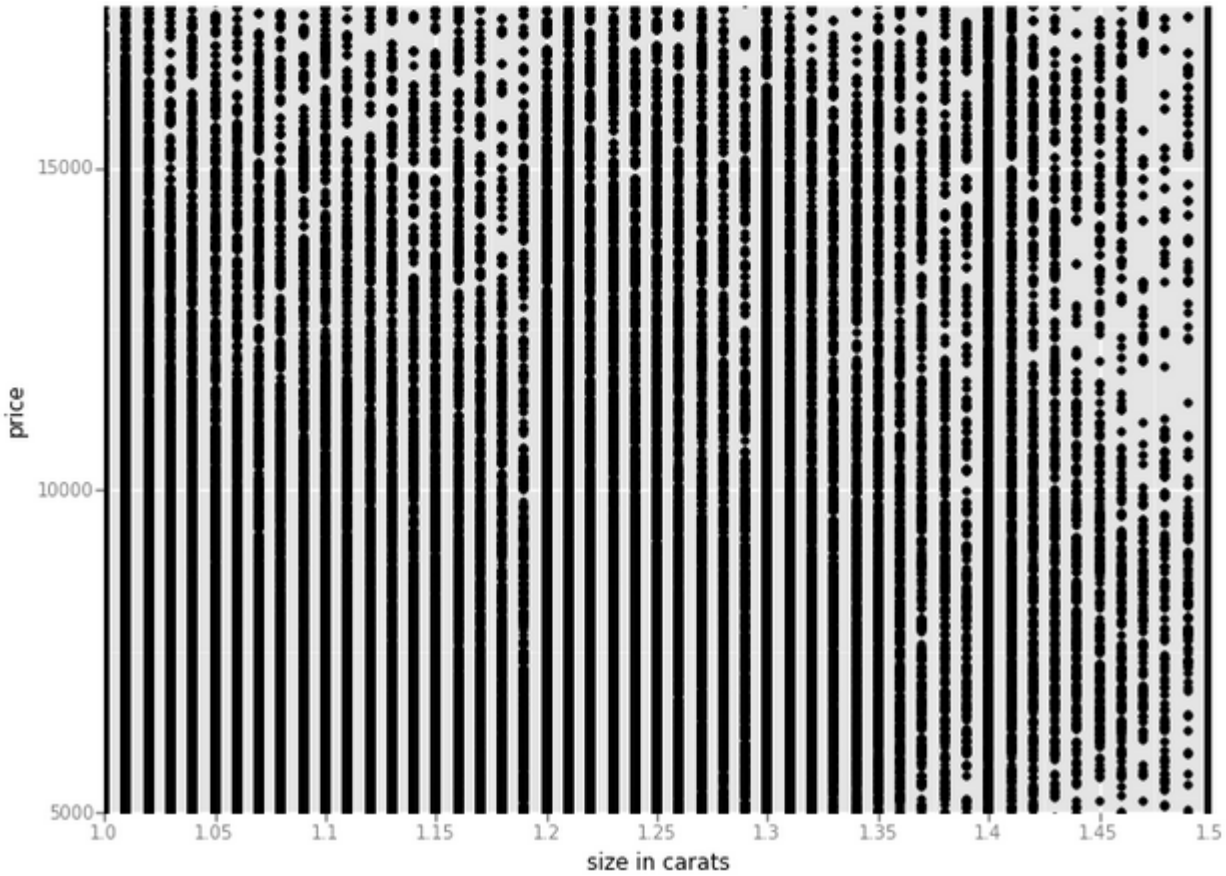
NOTE:
Since the original Price VS carat graph was not providing us accurate information, we narrow down the scale to a particular section. In the next slide we narrow it down further.

```
ggplot(diamonds, aes('carat', 'price')) + geom_point()+\
scale_y_continuous(limits=(5000,17500)) +\
    scale_x_continuous(name="size in carats", limits=(0.5,2.5))
```



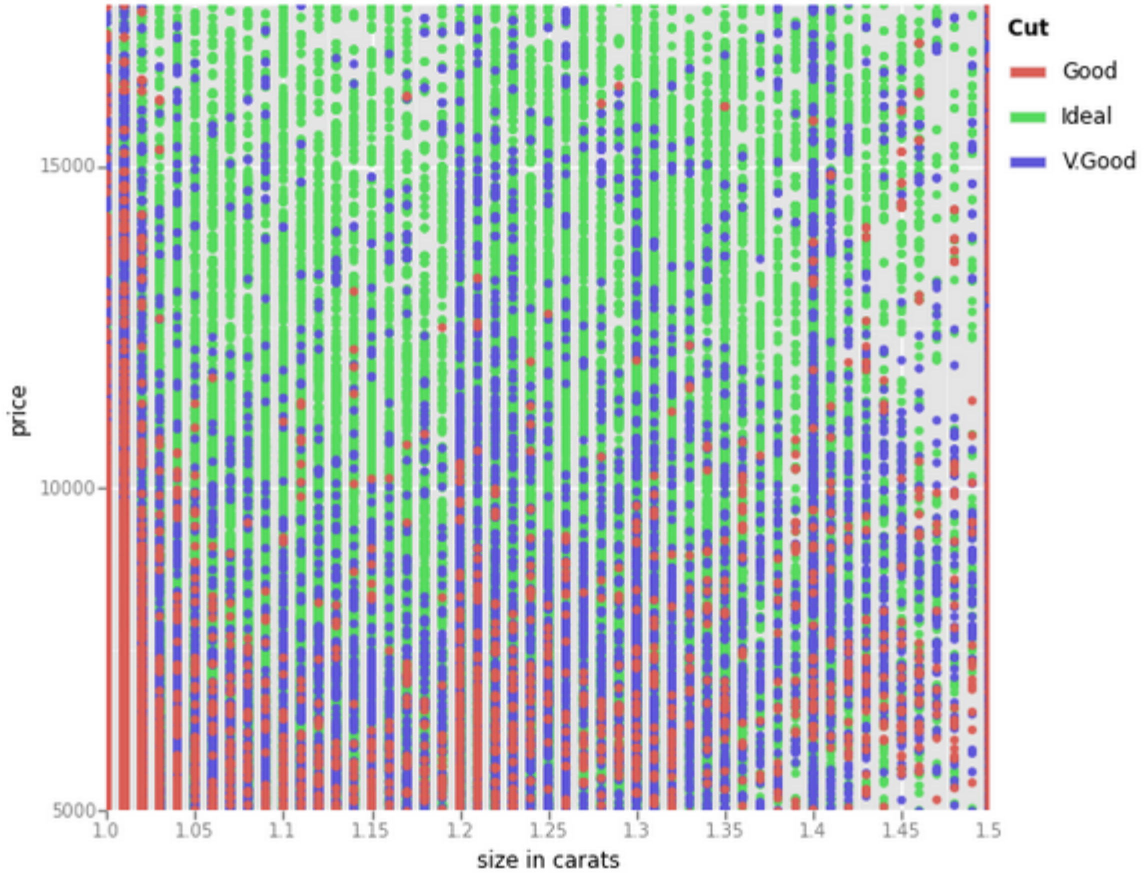
NOTE:
We see that the plot becomes vertical, i.e for the same value of carat we have varying price. Surely some other factor is controlling it.

```
ggplot(diamonds, aes('carat', 'price')) + geom_point()+\
scale_y_continuous(limits=(5000,17500)) +\
    scale_x_continuous(name="size in carats", limits=(1,1.5))
```



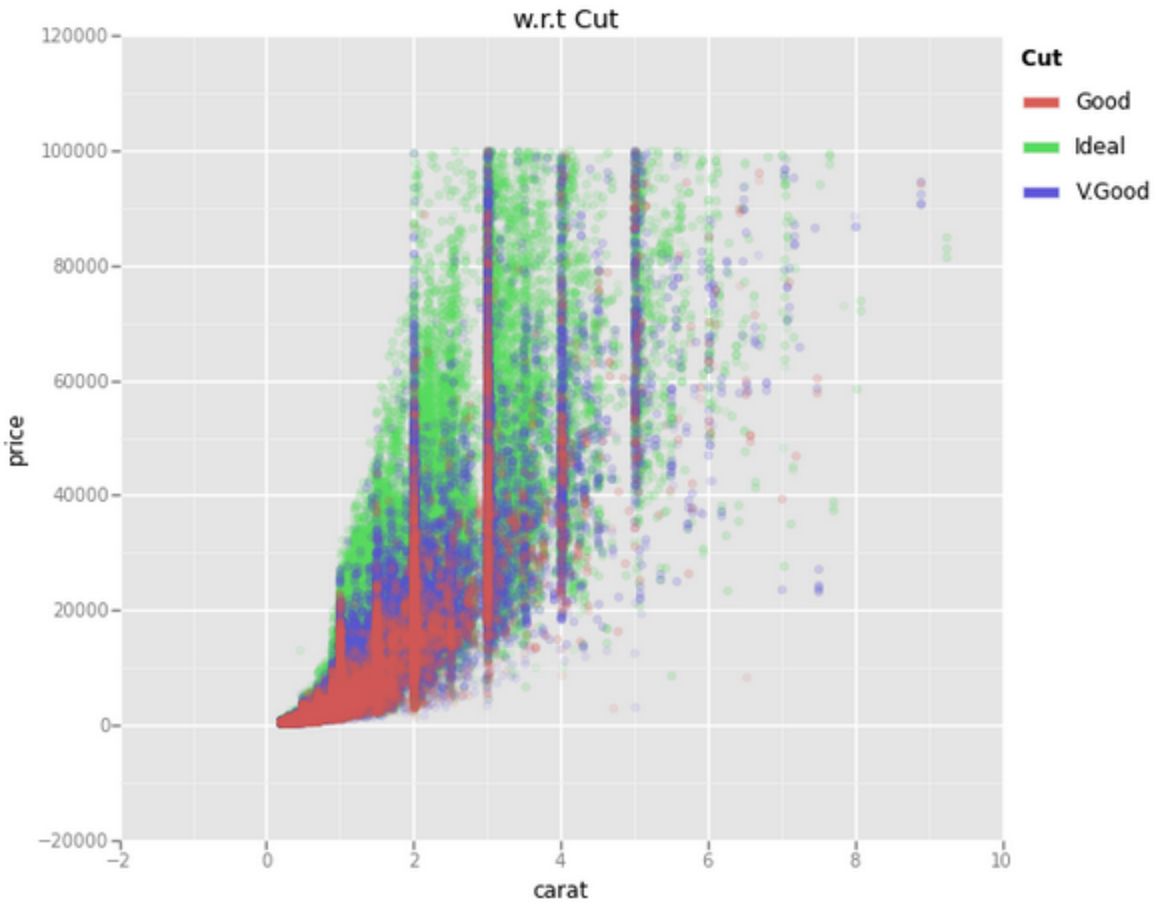
NOTE:
This is plotting the price with respect to the cut.
We see that for a given carat value the quality of changes the price.

In [12]:
```
ggplot(diamonds, aes('carat', 'price',color='cut')) + geom_point()+\
scale_y_continuous(limits=(5000,17500)) +\
    scale_x_continuous(name="size in carats", limits=(1,1.5))
```



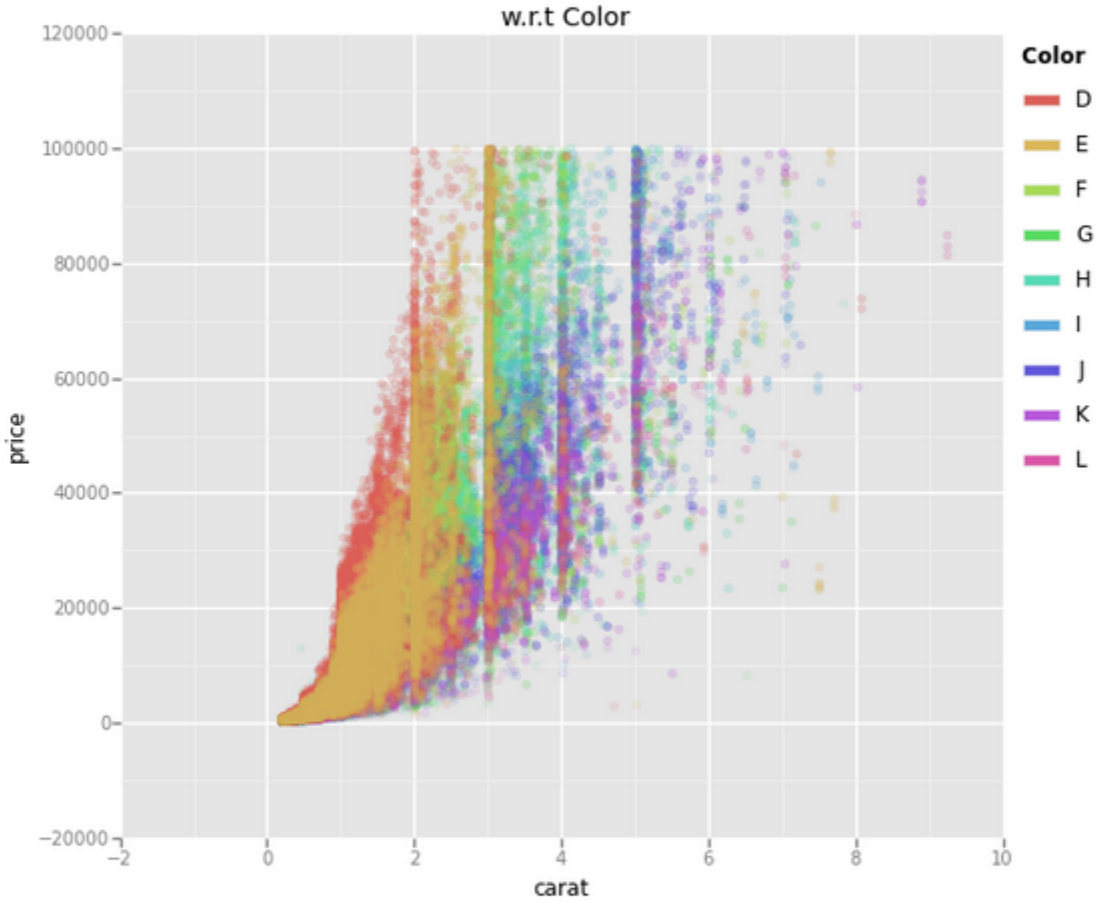
```
In [31]: ggplot(diamonds, aes('carat', 'price',color='cut')) + geom_jitter(alpha=0.1)+ggtitle("W.r.t Cut")
```

Differentiate price VS carat with respect to cut.



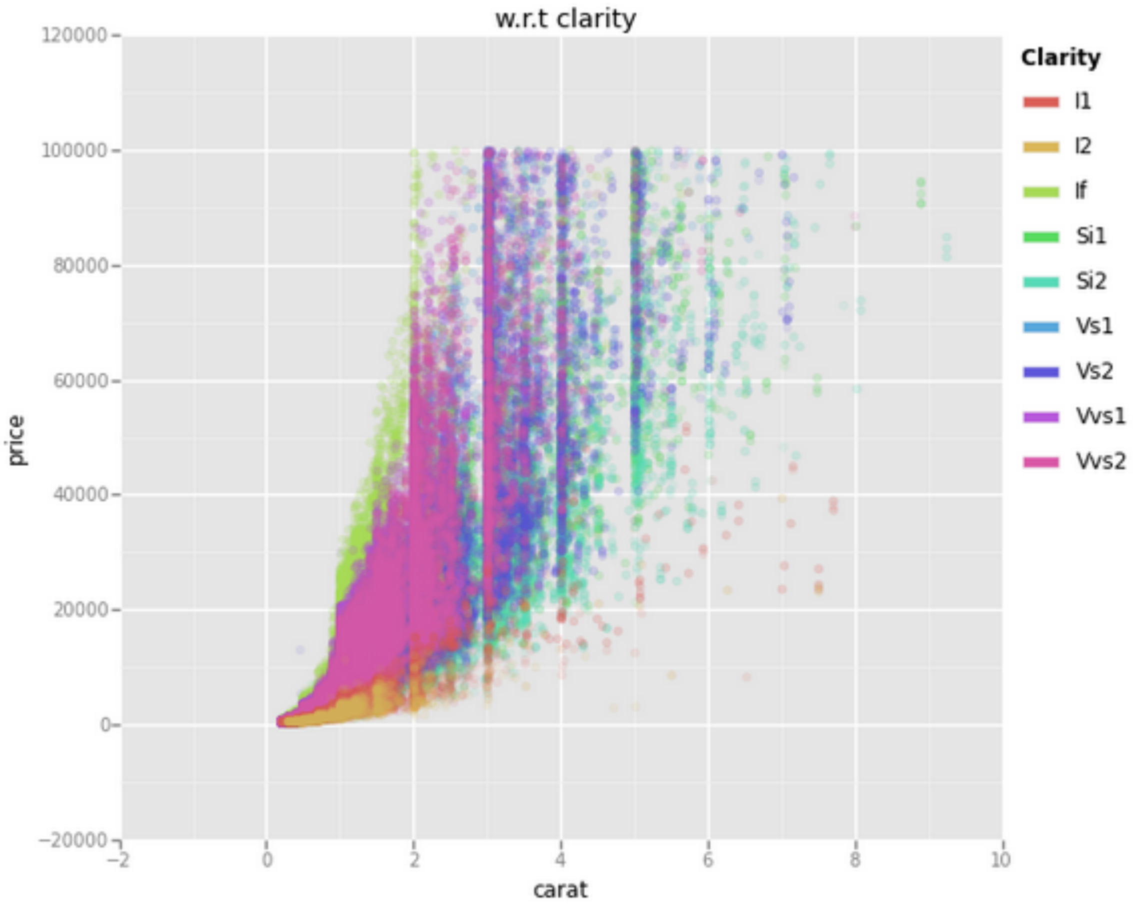
```
In [38]: ggplot(diamonds, aes('carat', 'price',color='color')) + geom_jitter(alpha=0.1)+ggtitle("W.r.t Color")
```

Differentiate price VS carat with respect to color.



```
In [37]: ggplot(diamonds, aes('carat', 'price',color='clarity')) + geom_jitter(alpha=0.1)+ggtitle("W.r.t Clarity")
```

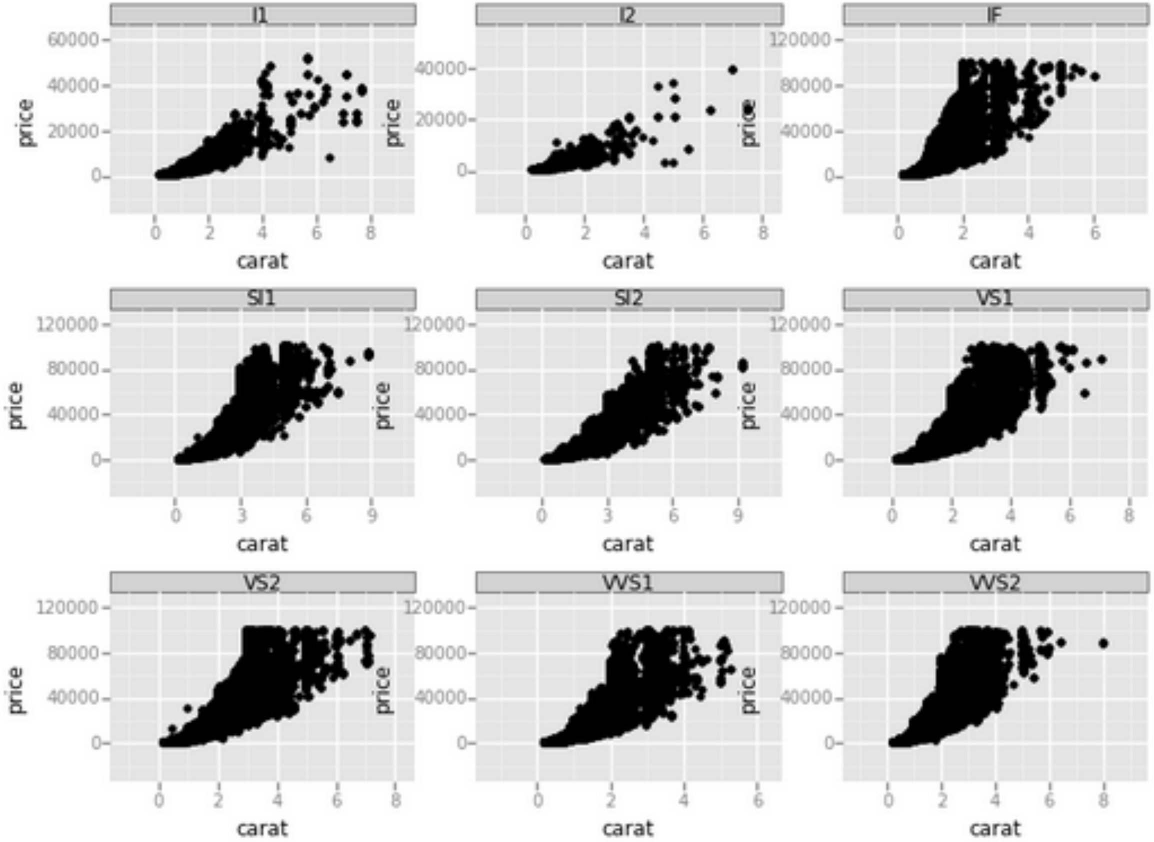
Differentiate price VS carat with respect to clarity.



FACETS

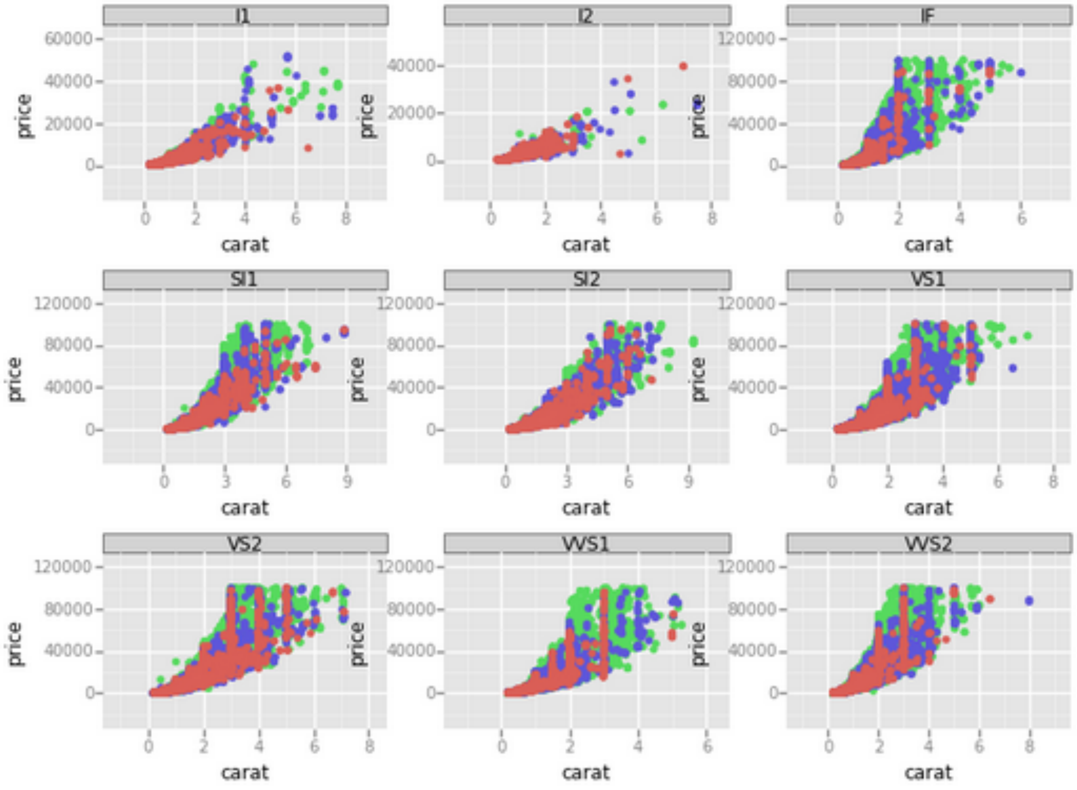
NOTE:
Facets- It features the same set of data with respect to a given factor. This helps us determine which value of factor affects f(x) the most,

```
In [22]: ggplot(aes(x='carat', y='price'), data=diamonds) + \
    geom_point() + facet_wrap("clarity")
```



FACETS

```
In [23]: ggplot(aes(x='carat', y='price', colour='cut'), data=diamonds) + \
             geom_point() + facet_wrap("clarity")
```



FURTHER SOURCES:

This presentation is a part of the larger pool of learning resources provided by DecisionStats.org

1. https://decisionstats.org
2. https://github.com/SolomonMg/diamonds-data
3. https://themessier.wordpress.com/2015/06/17/ggplot-in-python-part-1
4. http://nbviewer.ipython.org/gist/sara-02/d5a61234ef32e60bddda
5. http://nbviewer.ipython.org/gist/sara-02/d38da4a2023da169ac13
6. https://gist.github.com/sara-02/4eb520fd1b82521e8a11

CITATION:

1. http://ggplot.yhathq.com/
2. https://github.com/yhat/ggplot

FOR QUERIES:

info@decisionstats.org
sarahmasud02@gmail.com

THANK YOU