

Tutorial ggplot2

Laboratório de Macroevolução e Macroecologia

Maio/2015

Contents

1 O pacote ggplot2	2
1.1 Instalação	2
2 Como organizar os dados para o <i>ggplot2</i>	2
3 Estrutura básica de um gráfico	3
3.1 Gráficos unidimensionais (boxplots e histogramas)	3
3.2 Gráficos bidimensionais (gráficos de dispersão, modelos lineares, etc.)	6
3.2.1 Gráficos de dispersão	6
3.2.2 Adicionando linhas de tendência	11
4 Personalizando os gráficos	14
4.1 Alterar rótulos dos eixos	14
4.2 Alterar limites dos eixos	15
4.3 Alterar tamanho e direção das fontes dos eixos	16
4.4 Inserir anotações	17
4.5 Alterar paleta de cores	20
4.6 Separar gráficos em painéis	21
4.7 Alterar posição, título e tamanho da legenda	24
4.8 Alterar temas	29

1 O pacote *ggplot2*

O pacote *ggplot2* foi desenvolvido pelo Dr. Hadley Wickham (que desenvolveu diversos outros pacotes importantes para **R** como o *(d)plyr* por exemplo. Este pacote implementa uma nova maneira de criar gráficos a partir dos dados, trazendo o conceito de camadas (*layers*) para a sintaxe do **R**. Com esse novo conceito, surge também a necessidade de se organizar os dados de uma maneira que facilite a utilização da nova sintaxe. Para isso, os outros pacotes (como os já mencionados *(d)plyr*) foram desenvolvidos pelo Dr. Wickham de maneira a tornar a manipulação de dados mais ágil, intuitiva e já adequada para o uso conjunto com *ggplot2*.

1.1 Instalação

O pacote deve ser instalado como qualquer outro pacote no **R**:

```
install.packages("ggplot2")
```

É possível também instalar as versões mais recentes do pacote (mas que podem apresentar instabilidades ou serem incompletas) através da função *install_github* do pacote **devtools**. Este tipo de instalação usa os arquivos disponíveis no repositório público do *ggplot2* no github.

```
install.packages("devtools")
library(devtools)
install_github("hadley/ggplot2")
```

Após a instalação, carregue o pacote para utilização.

```
library(ggplot2)
```

2 Como organizar os dados para o *ggplot2*

É bastante comum usuários enfrentarem problemas nas primeiras vezes que usam o *ggplot2* pois ao organizarem os dados para serem importados no **R**, procuram o fazer da maneira mais sintética possível, agrupando variáveis por exemplo. A tabela a seguir é um exemplo de como **NÃO** se deve organizar os dados para serem usados com o *ggplot2* (para nenhuma situação na verdade).

```
##          Tratamento  Valor
## 1    Tratamento A - 2h 10.424
## 2    Tratamento A - 4h  9.663
## 3    Tratamento A - 8h  9.317
## 4    Tratamento A - 16h 10.112
## 5    Tratamento A - 24h  9.219
## 6    Tratamento B - 2h  9.569
## 7    Tratamento B - 4h  9.668
## 8    Tratamento B - 8h 10.177
## 9    Tratamento B - 16h 10.388
## 10   Tratamento B - 24h  8.774
```

Os objetos de entrada nas funções do *ggplot2* são da classe *data.frame*. No entanto, organizar o *data.frame* de maneira a otimizar o trabalho poupa bastante tempo e é de cabeceira. O *data.frame* a ser usado em um trabalho deve idealmente conter cada uma das variáveis em colunas separadas, e cada observação em linhas separadas. Assim, o exemplo anterior deveria ser idealmente organizado da seguinte forma:

```

##      Valor Tempo Tratamento
## 1  10.424     2 Tratamento A
## 2   9.663     4 Tratamento A
## 3   9.317     8 Tratamento A
## 4  10.112    16 Tratamento A
## 5   9.219    24 Tratamento A
## 6   9.569     2 Tratamento B
## 7   9.668     4 Tratamento B
## 8  10.177     8 Tratamento B
## 9  10.388    16 Tratamento B
## 10  8.774    24 Tratamento B

```

Alguns pacotes foram desenvolvidos com o propósito de facilitar a organização dos dados de maneira limpa e otimizada. Dentre esses pacotes, encontram-se (*dplyr*, *tidyverse*, *magrittr* dentre outros).

3 Estrutura básica de um gráfico

Como dito na introdução, um gráfico feito no *ggplot2* consiste de diversas camadas que são adicionadas a uma estrutura básica. Essa estrutura básica é criada usando as funções *ggplot* ou *qplot*. Essas funções possuem dois componentes essenciais: *data*, que como o nome diz recebe o *data.frame* a ser utilizado para a construção do gráfico, e *mapping*, que é o argumento que controla a estética do gráfico, ou seja, que controla quais variáveis serão plotadas nos eixos *x* e *y*, dentre outros detalhes que serão abordados posteriormente. Para o mapeamento dos dados, usaremos a função *aes*, na qual indicaremos qual variável deve ser posicionada em cada eixo, além de outros argumentos relativos a cor, tamanho, etc.

Diogo Melo, do Laboratório de Evolução de Mamíferos do IB-USP sintetizou uma regra geral para gráficos no *ggplot2*:

```

ggplot(data_frame_entrada, aes(x = coluna_eixo_x,
                               y = coluna_eixo_y,
                               group = coluna_agrupadora,
                               color = coluna_das_cores))
+ geom_tipo_do_grafico(opções que não dependem dos dados,
                       aes(opções que dependem))

```

Para exemplificar a construção de um gráfico, usaremos o conjunto de dados do próprio pacote *ggplot2* chamado **diamonds**.

```

data(diamonds)
head(diamonds)

##   carat      cut color clarity depth table price     x     y     z
## 1  0.23    Ideal    E    SI2  61.5     55   326  3.95 3.98 2.43
## 2  0.21  Premium    E    SI1  59.8     61   326  3.89 3.84 2.31
## 3  0.23      Good    E    VS1  56.9     65   327  4.05 4.07 2.31
## 4  0.29  Premium    I    VS2  62.4     58   334  4.20 4.23 2.63
## 5  0.31      Good    J    SI2  63.3     58   335  4.34 4.35 2.75
## 6  0.24  Very Good    J   VVS2  62.8     57   336  3.94 3.96 2.48

```

3.1 Gráficos unidimensionais (boxplots e histogramas)

Um bom modo de iniciar qualquer análise é fazendo uma análise exploratória dos dados. Para isso, alguns tipos específicos de gráficos como *boxplots* ou histogramas são maneiras simples de visualizar a estrutura

básica dos dados. Estes tipos de gráfico são ditos unidimensionais, pois representam apenas uma variável. Neste tópico, veremos como criar esses dois gráficos de maneira simples (complementos serão adicionados mais para frente no tutorial).

O primeiro passo para criarmos um gráfico é então gerar o objeto da classe **ggplot**. **DICA: armazene as camadas básicas do seu gráfico em um objeto, pois isso facilita a adição de novas camadas sem a necessidade de repetir o código inteiro.**

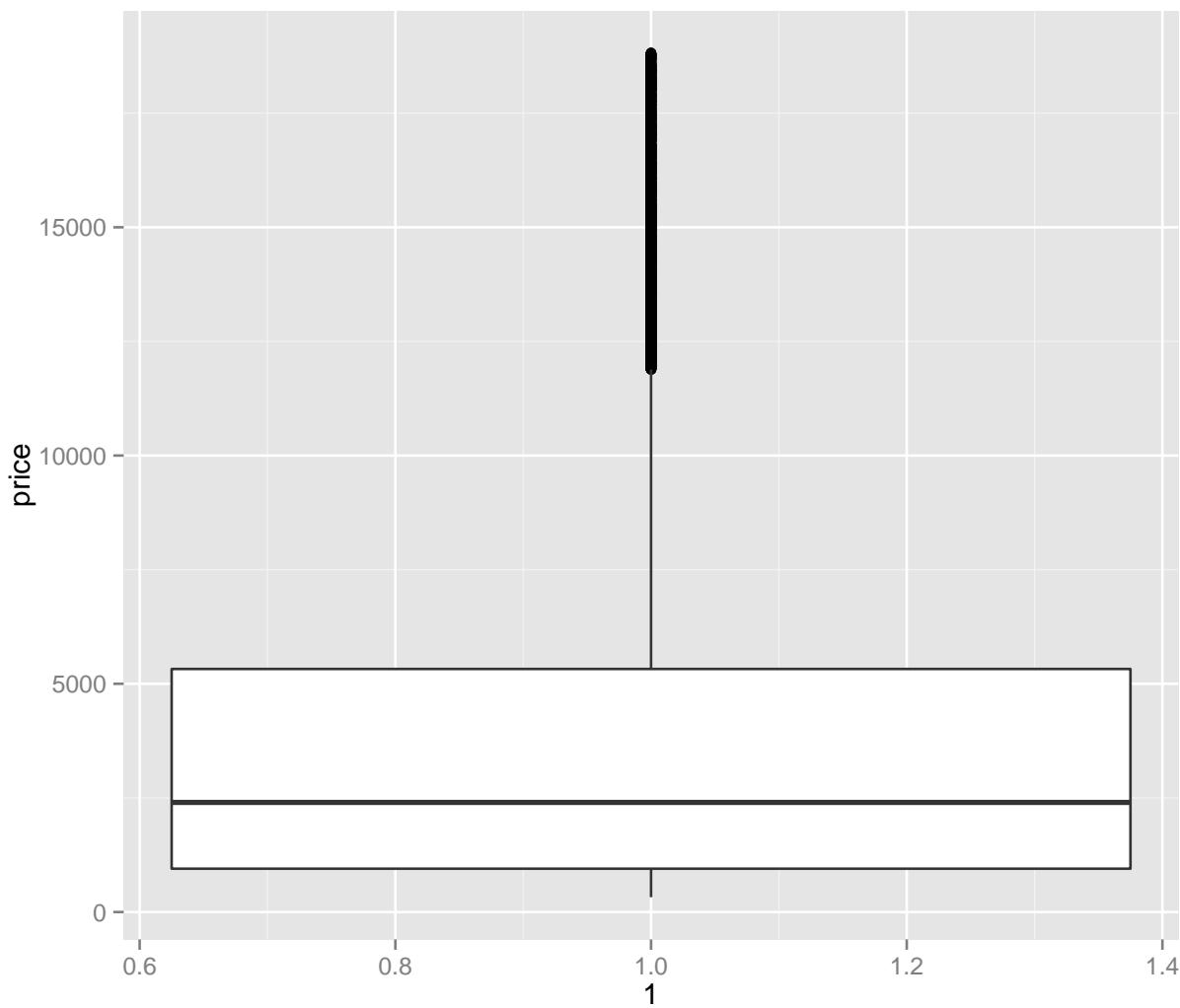
```
g1 <- ggplot(data=diamonds,aes(x=1,y=price))
```

No exemplo acima, indicamos que a variável *price* deve ser atribuída ao eixo *y*, e o eixo *x* recebe o valor 1 apenas para posicionar o gráfico. Além disso o código que acabamos de rodar serve apenas para criar um objeto da classe **ggplot**, porém ele ainda não produz nenhum gráfico. Ao chamarmos o objeto g1:

```
## Error: No layers in plot
```

Isso se deve ao fato de que nenhuma camada de gráfico foi adicionada, ou seja, não especificamos qual o tipo de gráfico desejamos produzir. A maioria das camadas iniciais podem ser criadas usando as funções da família **geom_**. No primeiro exemplo, vamos criar um boxplot para termos uma ideia de valores mínimos, máximos, média e posição dos quartis.

```
g1 + geom_boxplot()
```

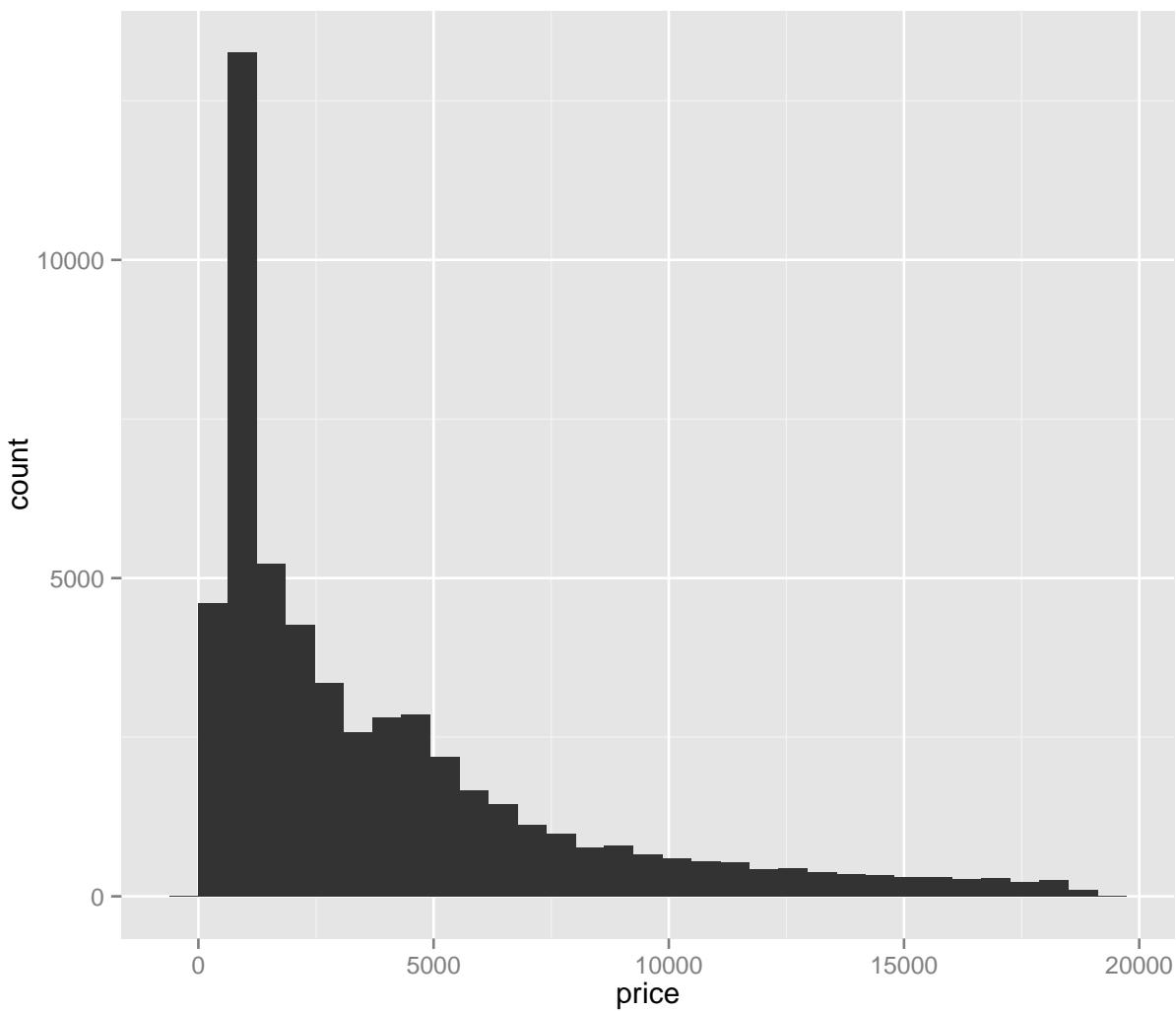


Note a sintaxe bastante intuitiva que usa o símbolo “+” para adicionar camadas. Isso torna o *ggplot2* um pacote bastante prático ~~depois de pegar o jeito~~ para criar gráficos informativos, bonitos e prontos para publicação.

Para criarmos um histograma simples, é igualmente fácil:

```
ggplot(diamonds,aes(x=price)) + geom_histogram()
```

```
## stat_bin: binwidth defaulted to range/30. Use 'binwidth = x' to adjust this.
```



Observe no entanto que para o histograma a variável de interesse está no eixo x , já que o eixo y contém a contagem de valores em cada intervalo.

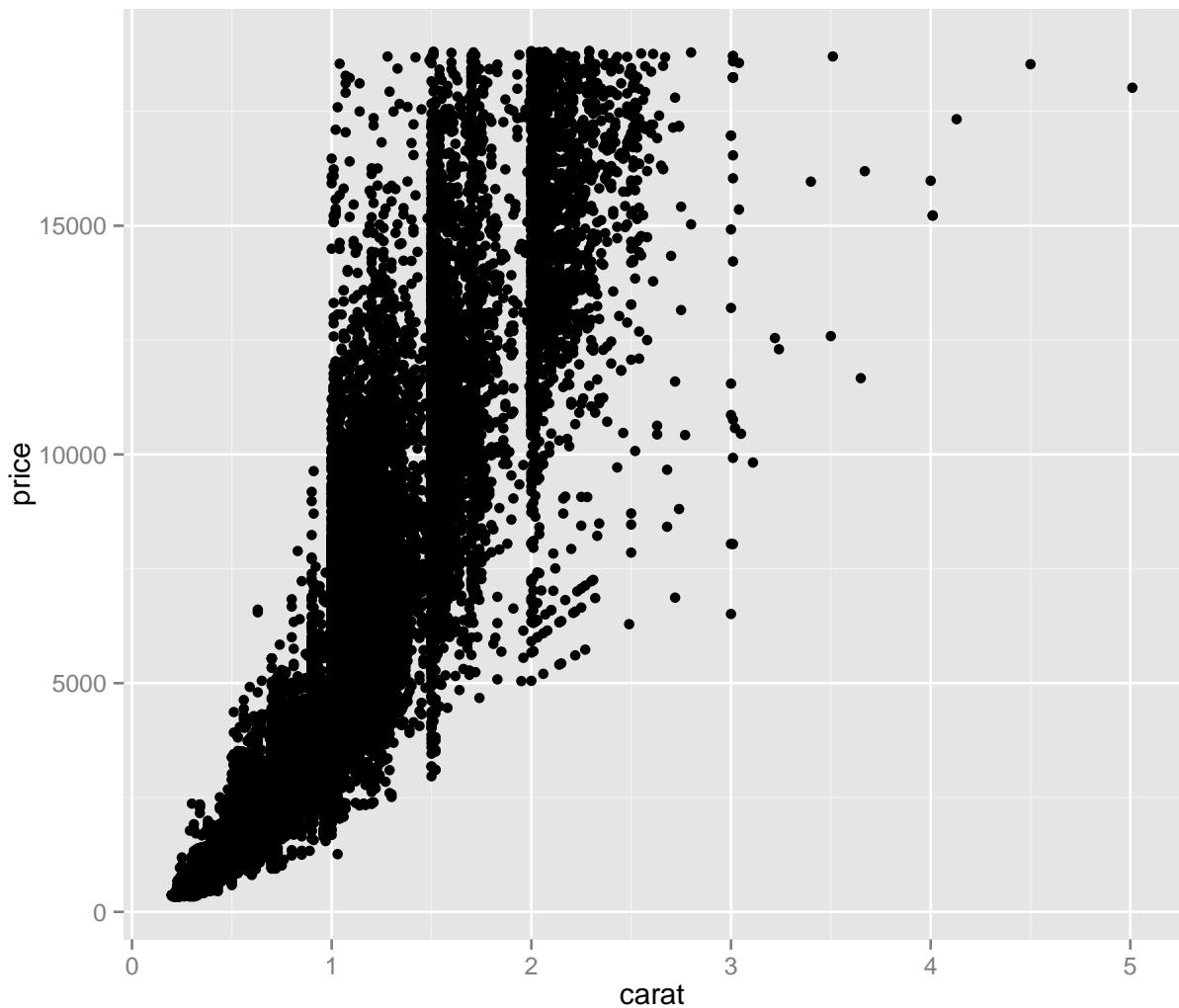
Os gráficos básicos podem não parecer tão bonitos assim à primeira vista; vamos aprender em seções posteriores como modificar a estética dos gráficos (cores, tamanho, bordas, plano de fundo, etc.).

3.2 Gráficos bidimensionais (gráficos de dispersão, modelos lineares, etc.)

3.2.1 Gráficos de dispersão

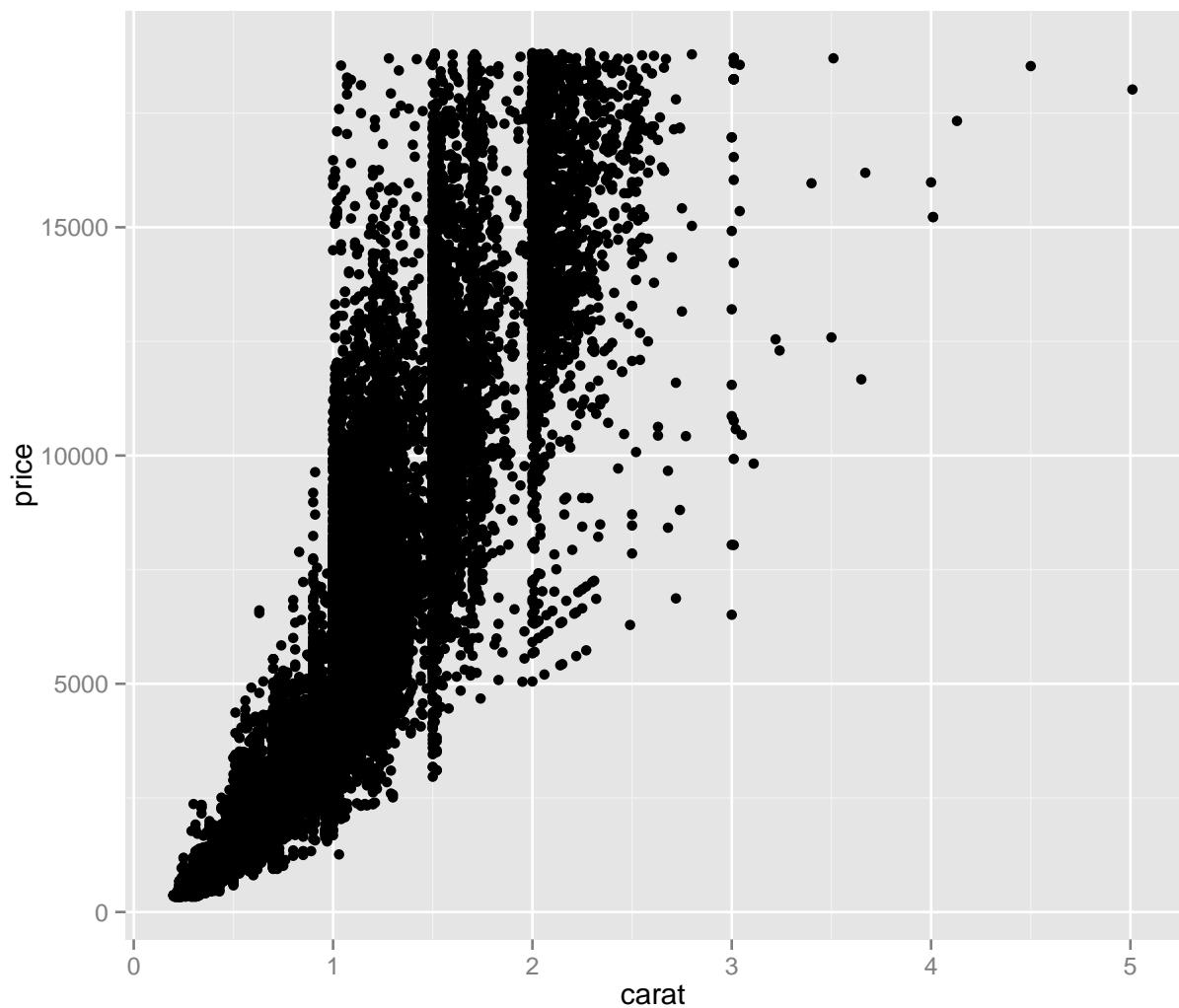
Em grande parte dos estudos em ecologia buscamos analisar como duas ou mais variáveis se comportam umas em relação às outras. Para isso, um bom ponto de partida é construir gráficos de dispersão (*scatterplots*). No exemplo a seguir construiremos um gráfico de dispersão entre as variáveis *carat* e *price*.

```
g2 <- ggplot(diamonds, aes(x=carat, y=price))
g2 + geom_point()
```



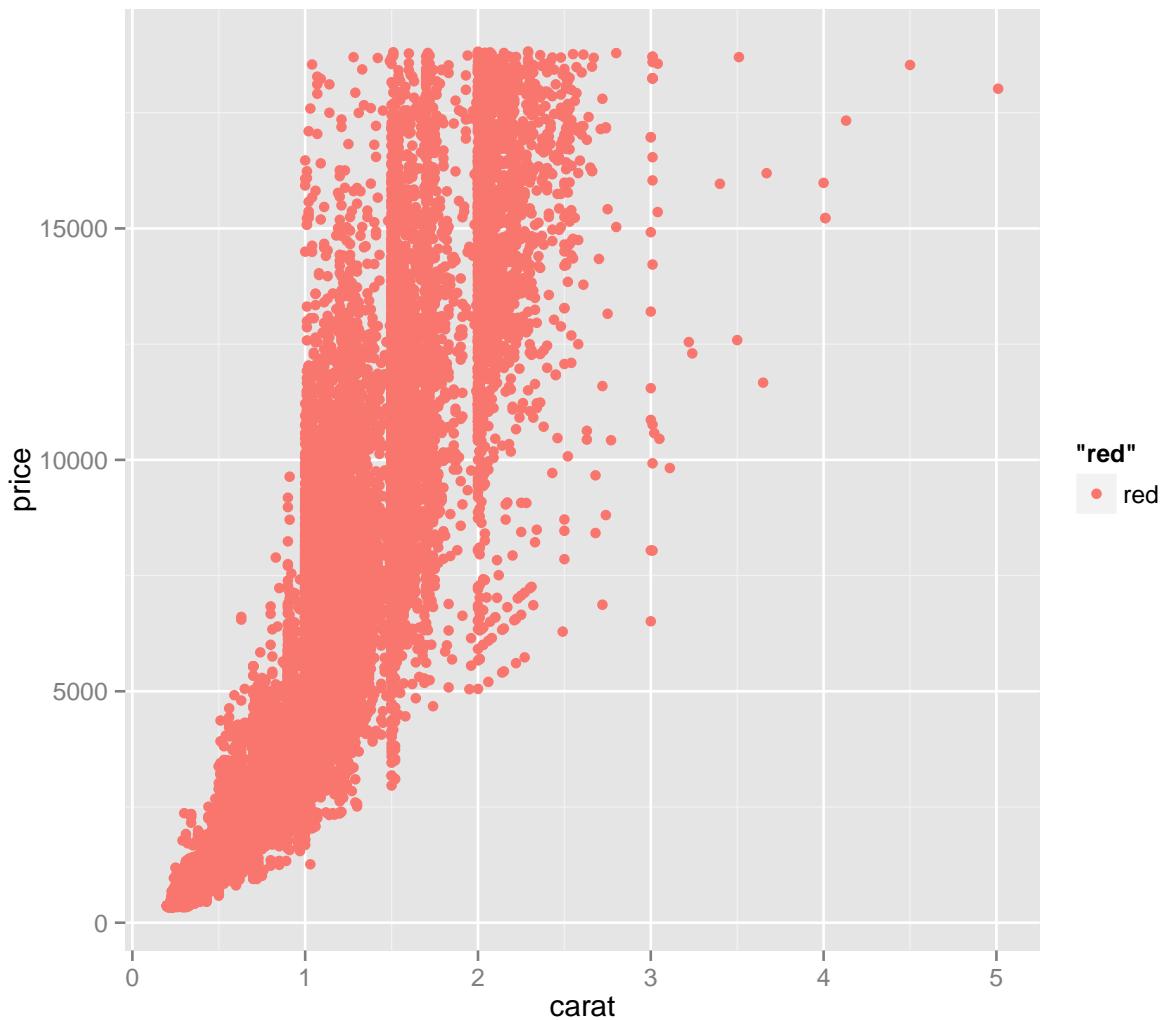
Note que no exemplo acima, estamos indicando para a função `ggplot` que as camadas que forem adicionadas de agora em diante devem conter os valores de `carat` no eixo `x` e de `price` no eixo `y`. Isso pode ser modificado posteriormente (como nos casos onde você quer por exemplo acrescentar uma outra variável no eixo `y`), e será exemplificado mais adiante. No entanto, a maioria (se não todas) das funções da família `geom_` também possuem os argumentos básicos `data` e `mapping`. Sendo assim, uma outra forma de criar o mesmo gráfico é:

```
ggplot() + geom_point(data=diamonds, mapping=aes(x=carat, y=price))
```



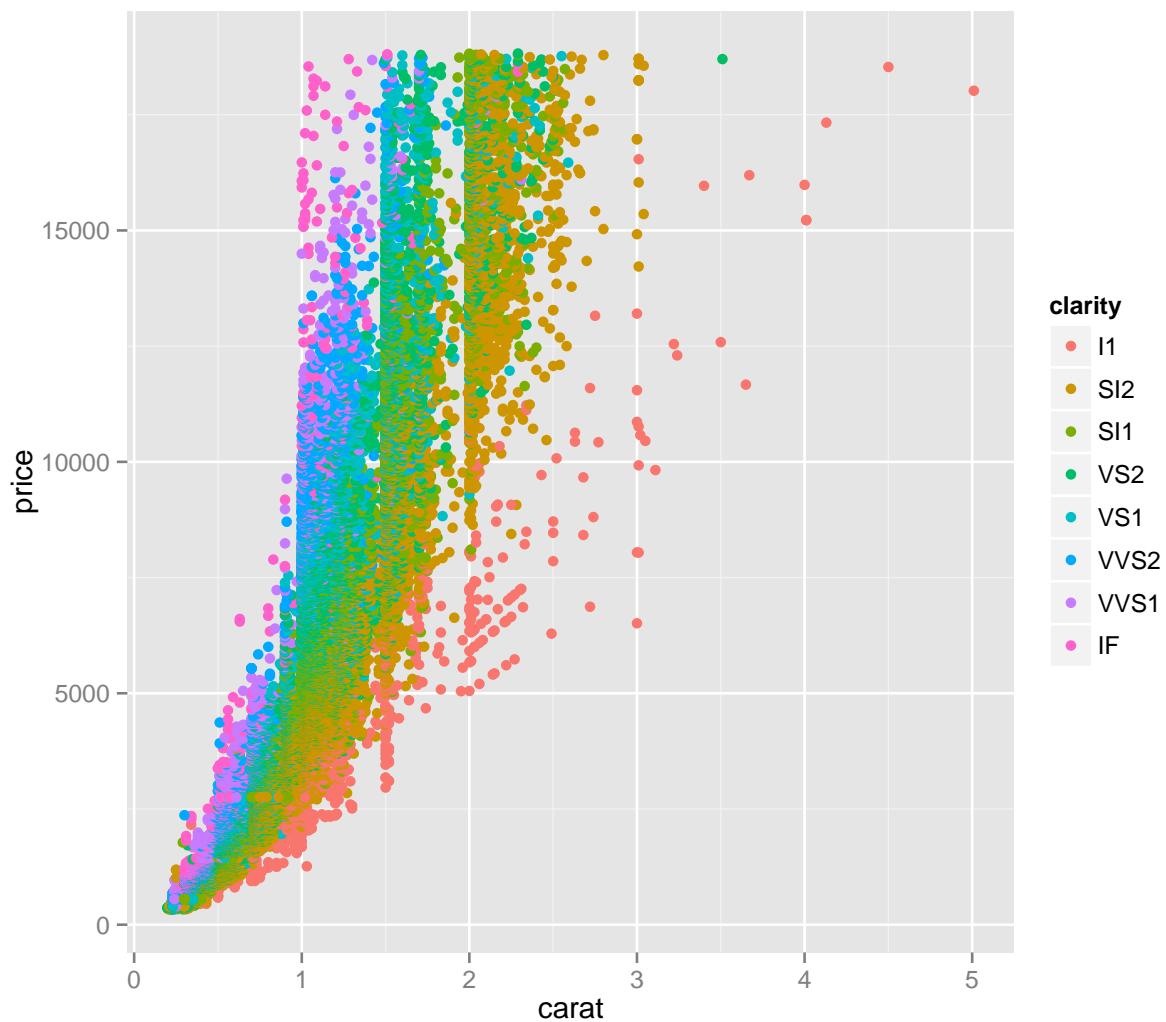
Caso a cor padrão seja feia não seja adequada, podemos indicar para o gráfico qual cor queremos usar para os pontos. Como cor é um dos componentes essenciais da parte estética de um gráfico, essa especificação deve entrar na função `aes`.

```
g2 + geom_point(mapping=aes(colour="red"))
```



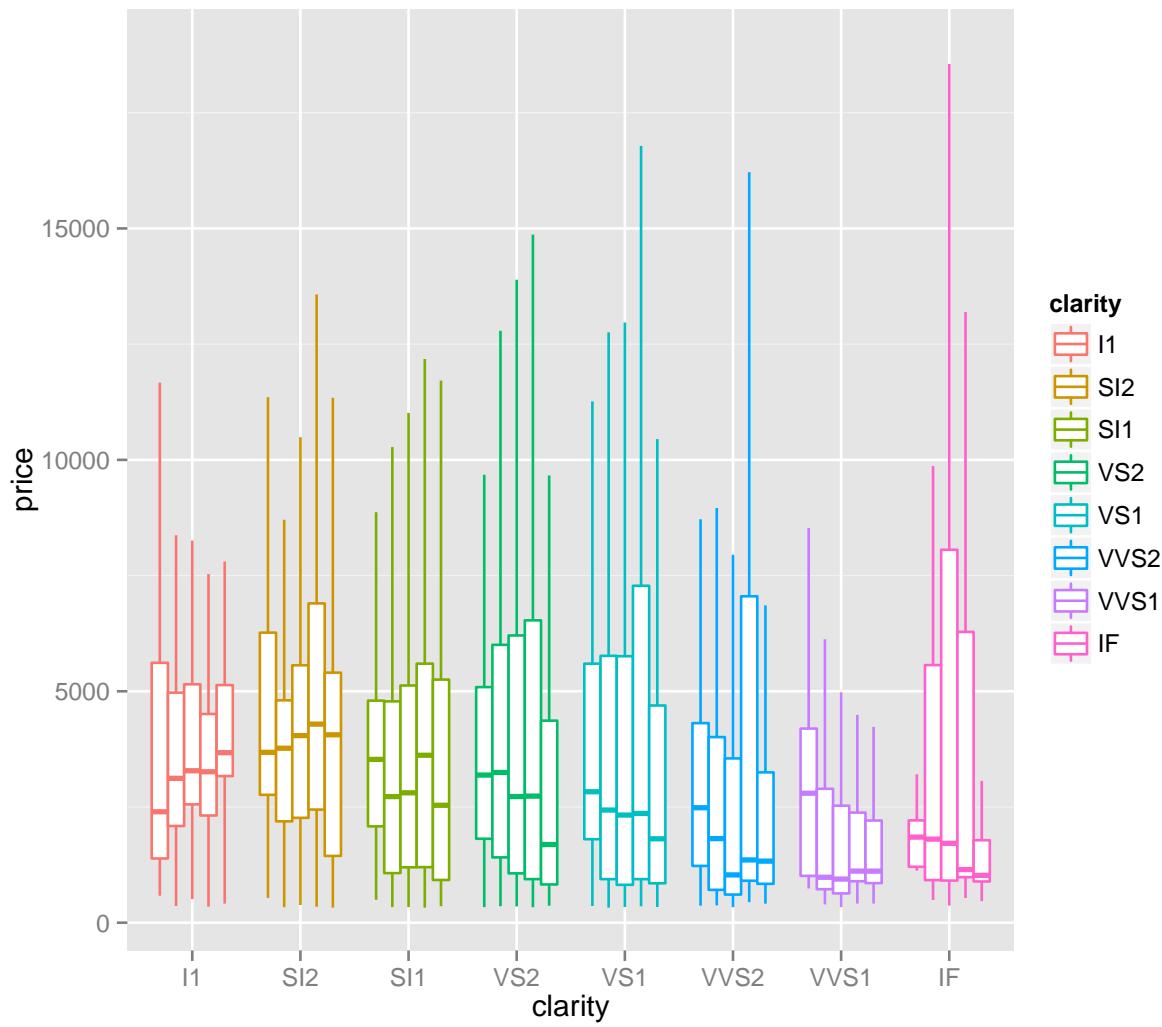
Ao analisarmos o conteúdo do objeto **diamonds** notamos que os valores de *carat* e *price* são divididos por classe de *cut*, *color* e *clarity*. Assim, podemos por exemplo representar o mesmo conjunto de dados separados por classe de *clarity* por exemplo. Se seu objeto contendo os dados estiver mal organizado, você levaria um tempo razoável para adicionar uma cada para cada classe cada uma com uma cor diferente. Porém, como já aprendemos a organizar nossos dados de maneira ideal, podemos facilmente usar a variável *clarity* para atribuir diferentes cores que representarão cada uma das classes da seguinte forma:

```
g2 + geom_point(mapping=aes(colour=clarity))
```



Essa separação em classes de *clarity* pode ser feita em outros tipos de gráfico também (os *outliers* foram removidos para fins de visualização):

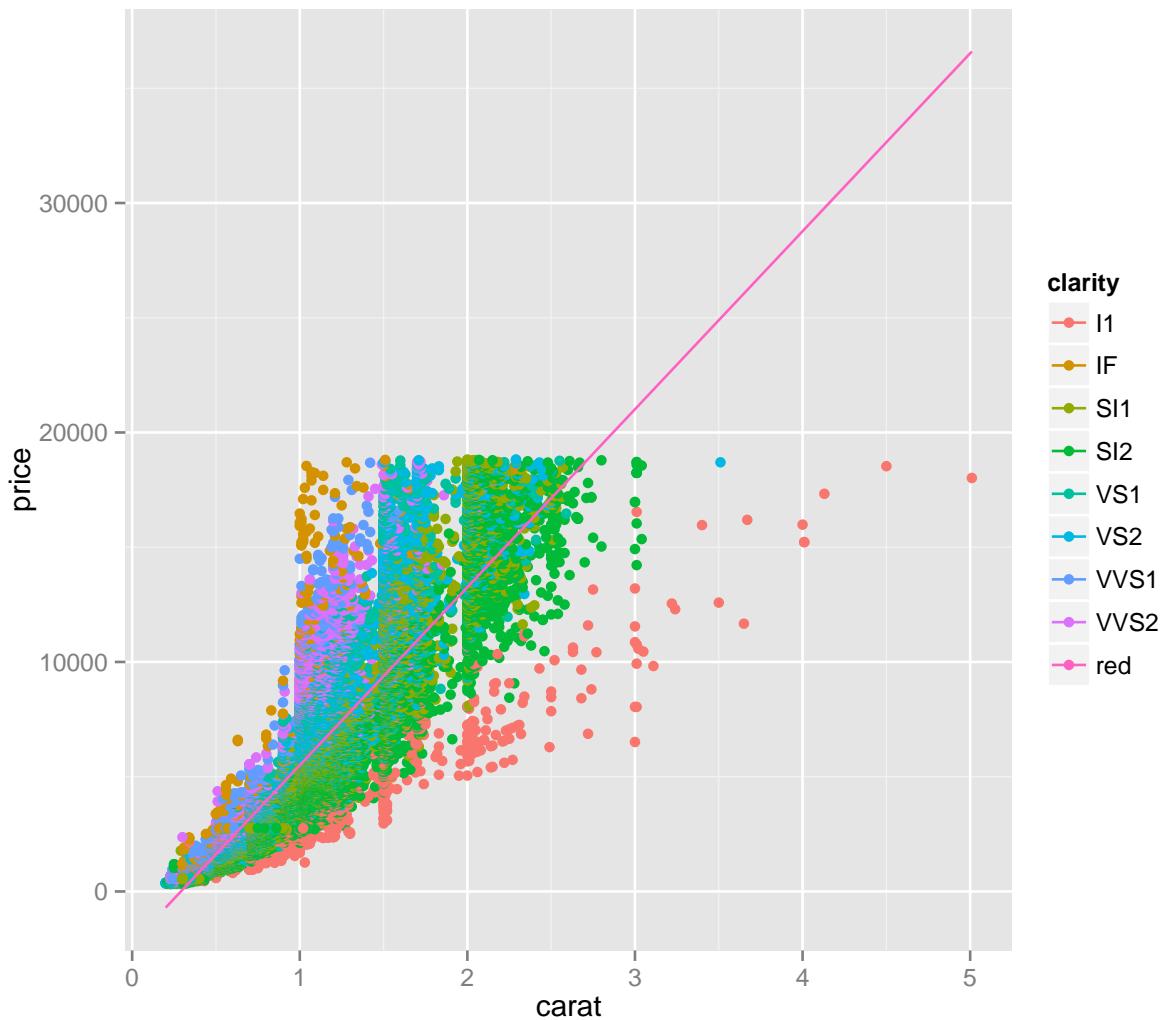
```
ggplot(data=diamonds,
       aes(x=clarity, y=price, group=interaction(clarity,cut), color=clarity)) +
  geom_boxplot(outlier.shape=NA)
```



3.2.2 Adicionando linhas de tendência

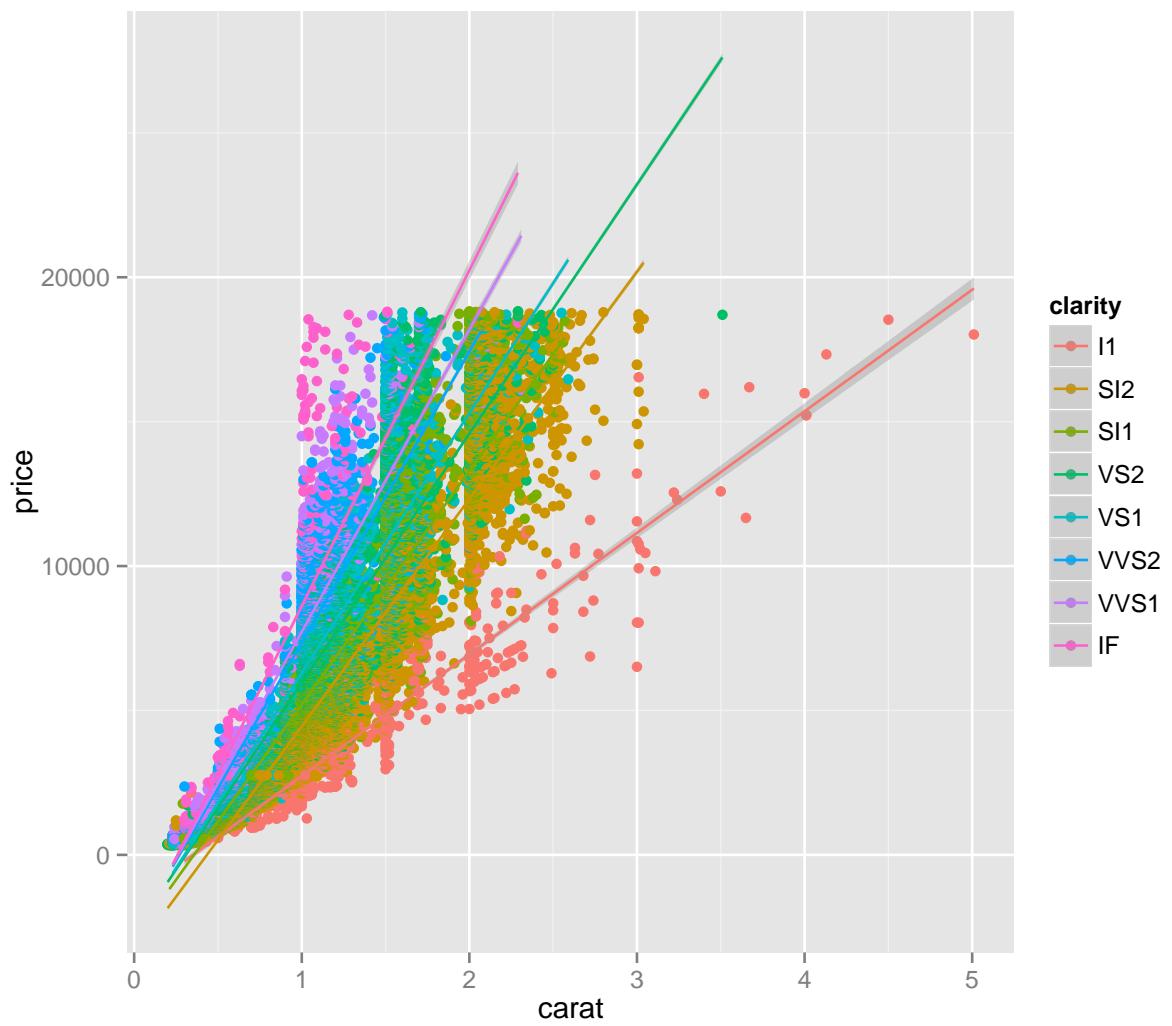
Os gráficos de dispersão que construímos até agora indicam que existe uma relação positiva entre quilates e preço. Nesse caso, podemos ajustar um modelo linear que indique como se dá essa correlação. Para isso, o pacote `ggplot2` possui funções específicas para algumas estatísticas comumente usadas. Aqui, usaremos a função `stat_smooth` para isso.

```
g2 + geom_point(aes(color=clarity)) + geom_smooth(method = "lm", se=FALSE, aes(color="red"))
```



A linha de regressão adicionada no gráfico anterior corresponde à correlação entre *carat* e *price* independentemente da classe de *clarity*. Porém, dependendo do caso pode ser mais interessante criar uma reta de correlação para cada classe de *clarity*. Fazemos isso da seguinte forma (adicionando também intervalos de confiança da média):

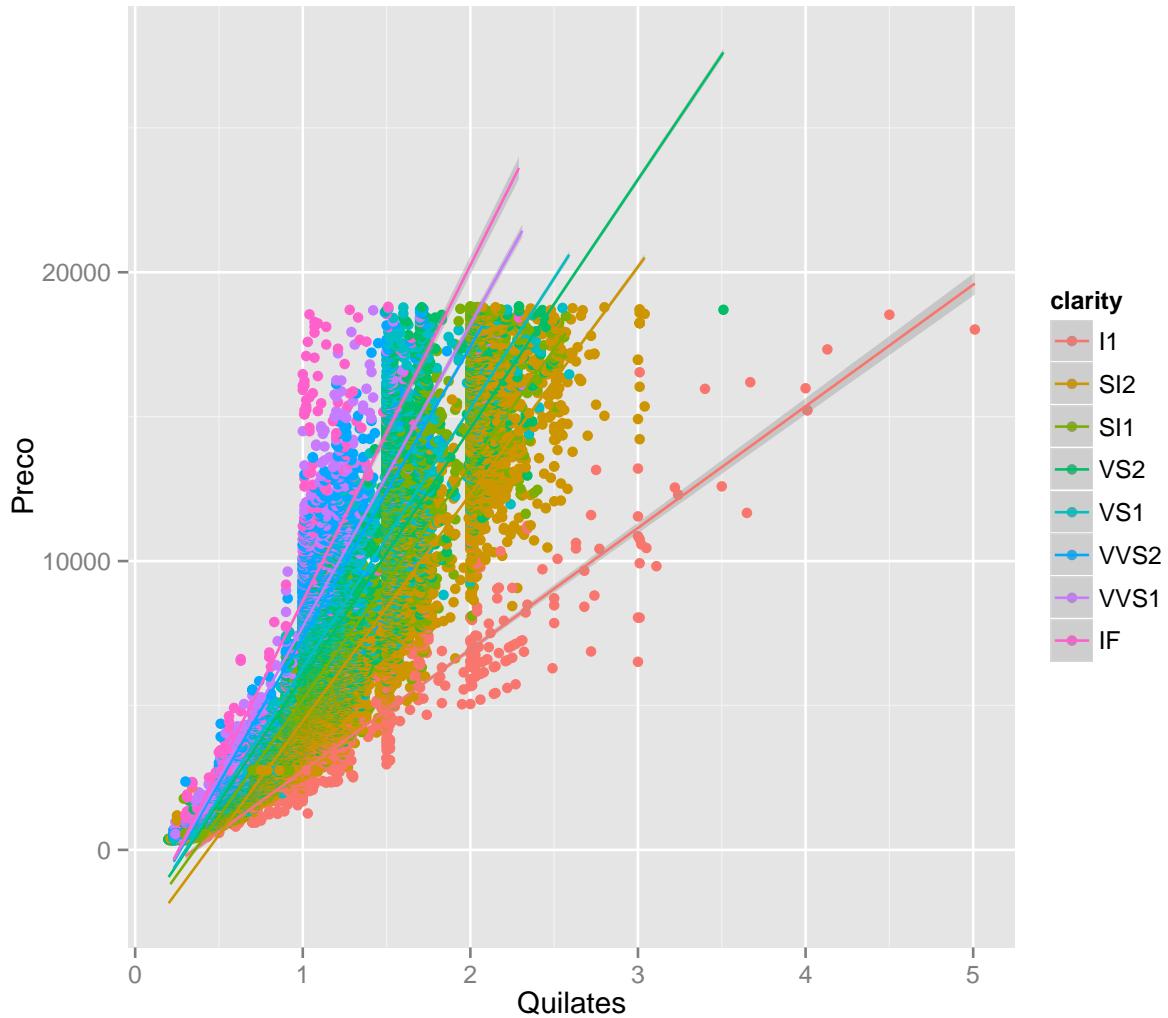
```
g3 <- g2 + geom_point(aes(color=clarity)) + geom_smooth(method = "lm", se=TRUE, aes(color=clarity))
g3
```



4 Personalizando os gráficos

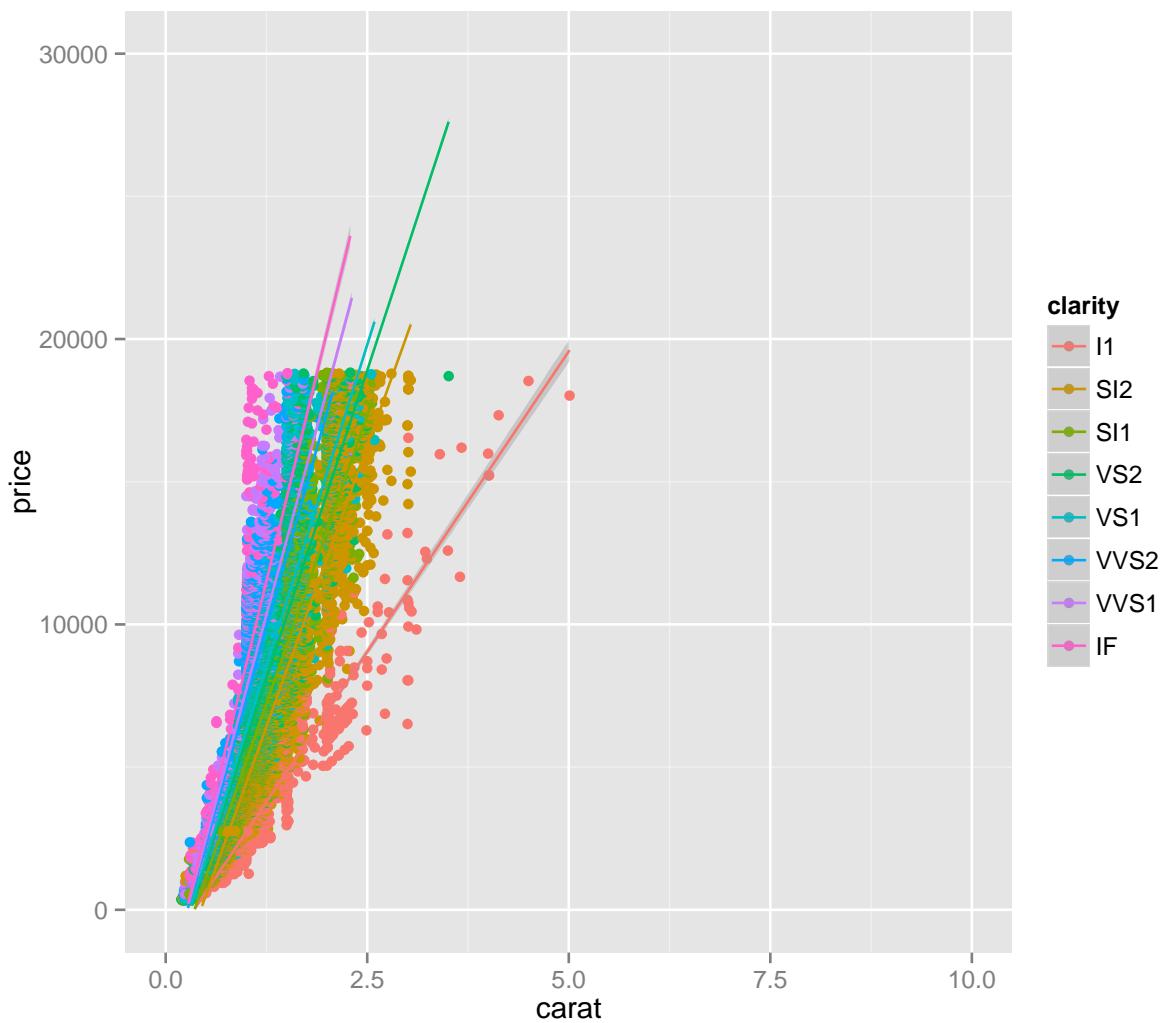
4.1 Alterar rótulos dos eixos

```
g3 + xlab("Quilates") + ylab("Preço")
```



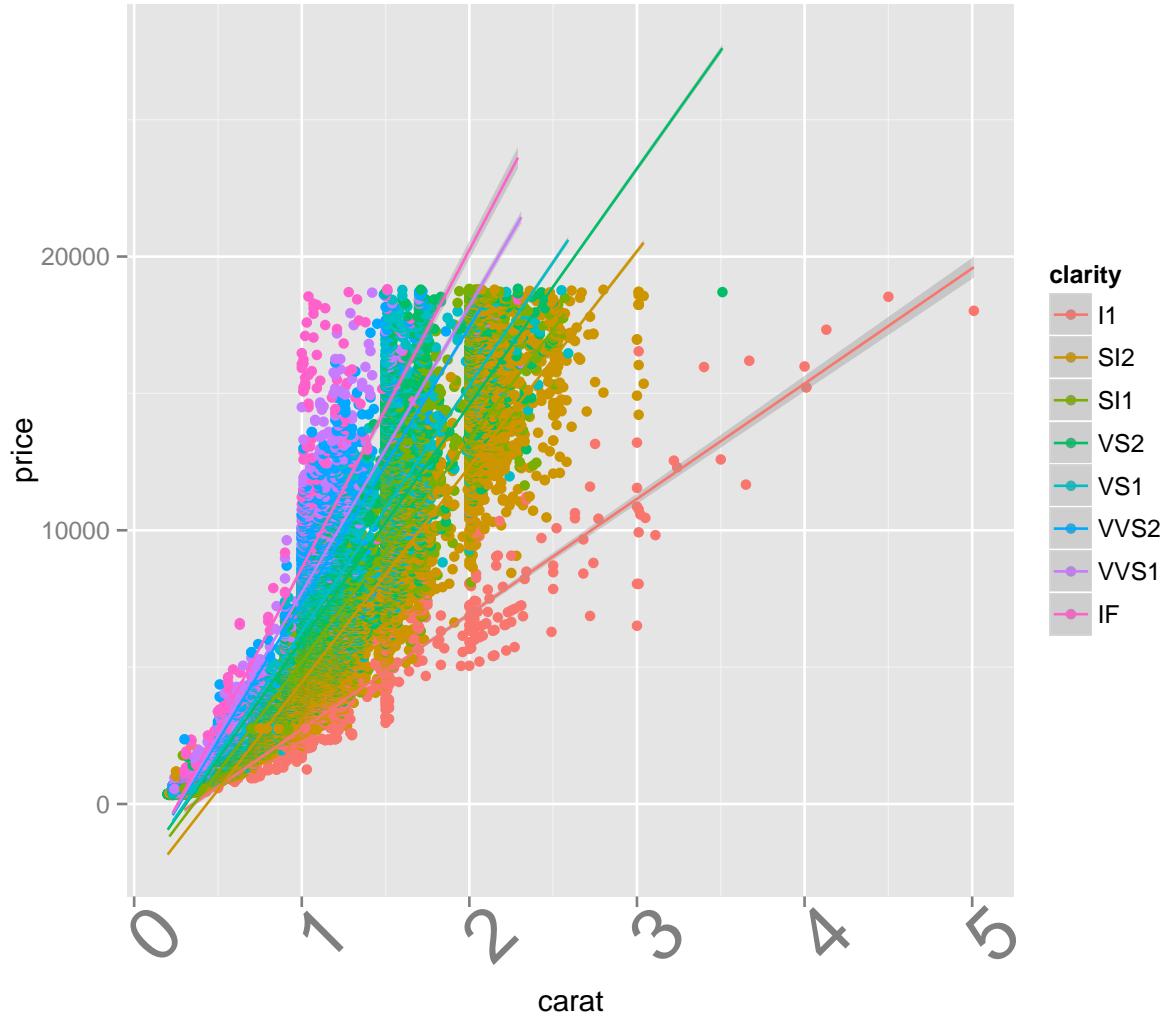
4.2 Alterar limites dos eixos

```
g3 + xlim(0,10) + ylim(0,30000)
```



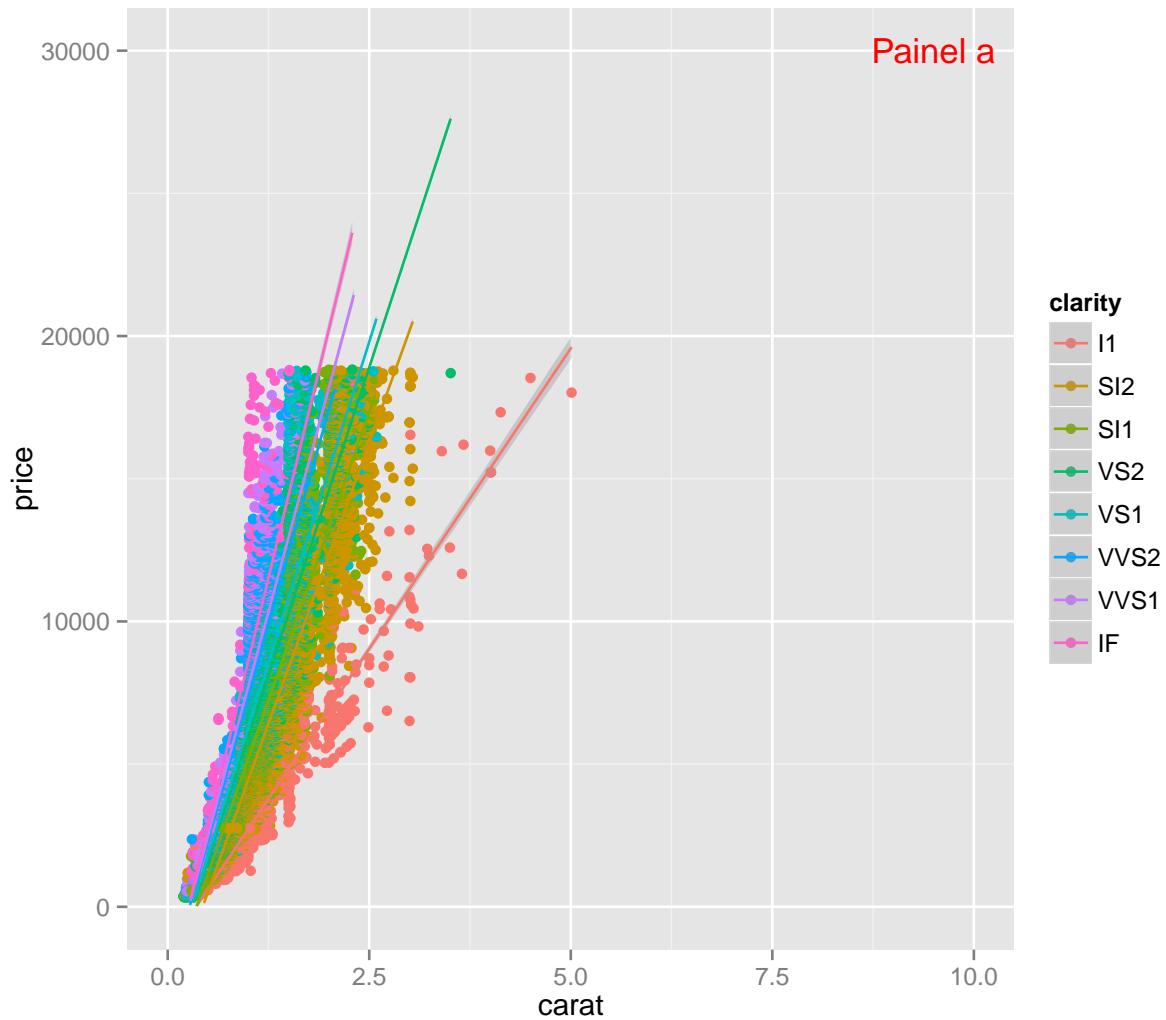
4.3 Alterar tamanho e direção das fontes dos eixos

```
g3 + theme(axis.text.x = element_text(angle=45,vjust=1,size=30))
```

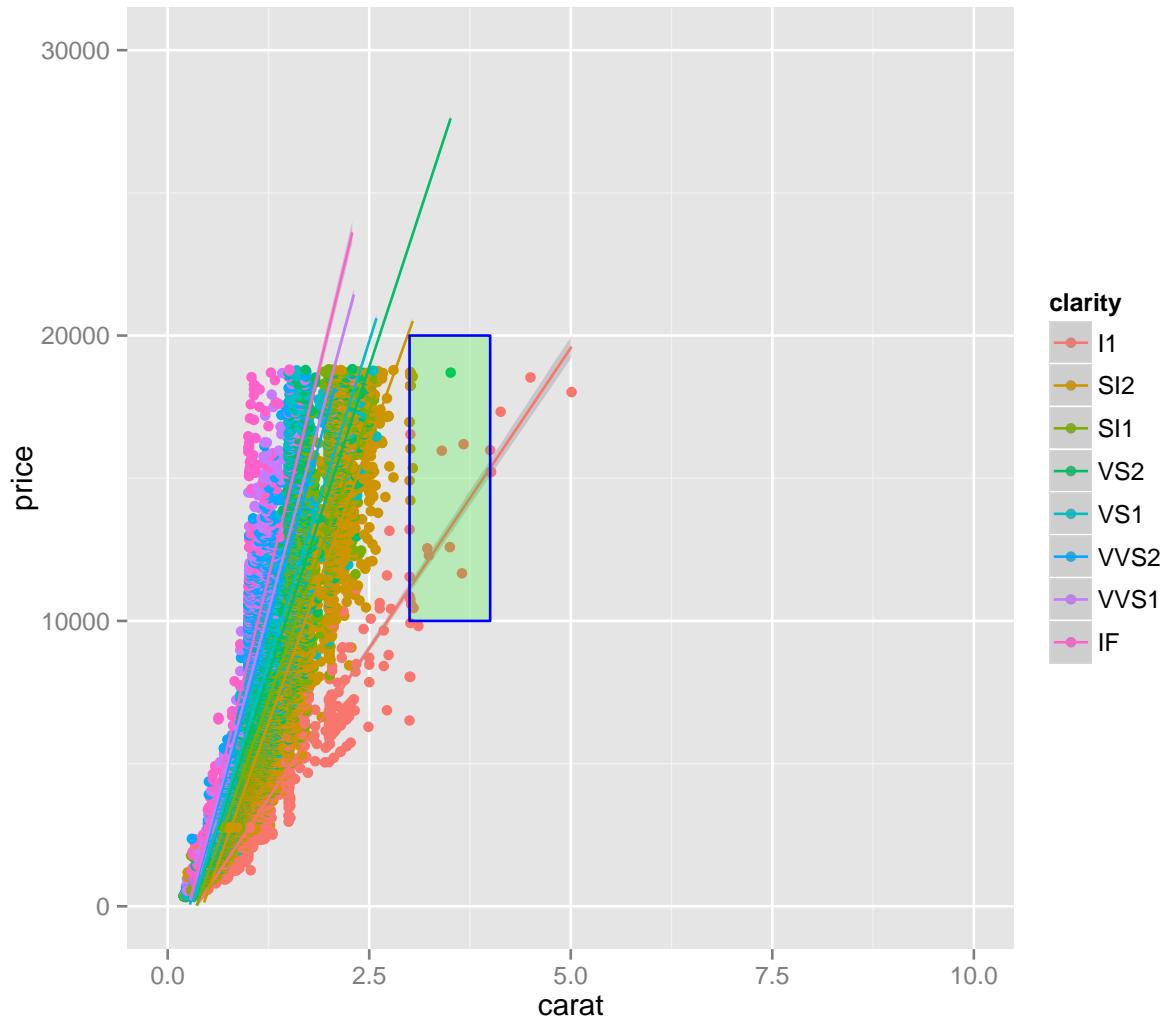


4.4 Inserir anotações

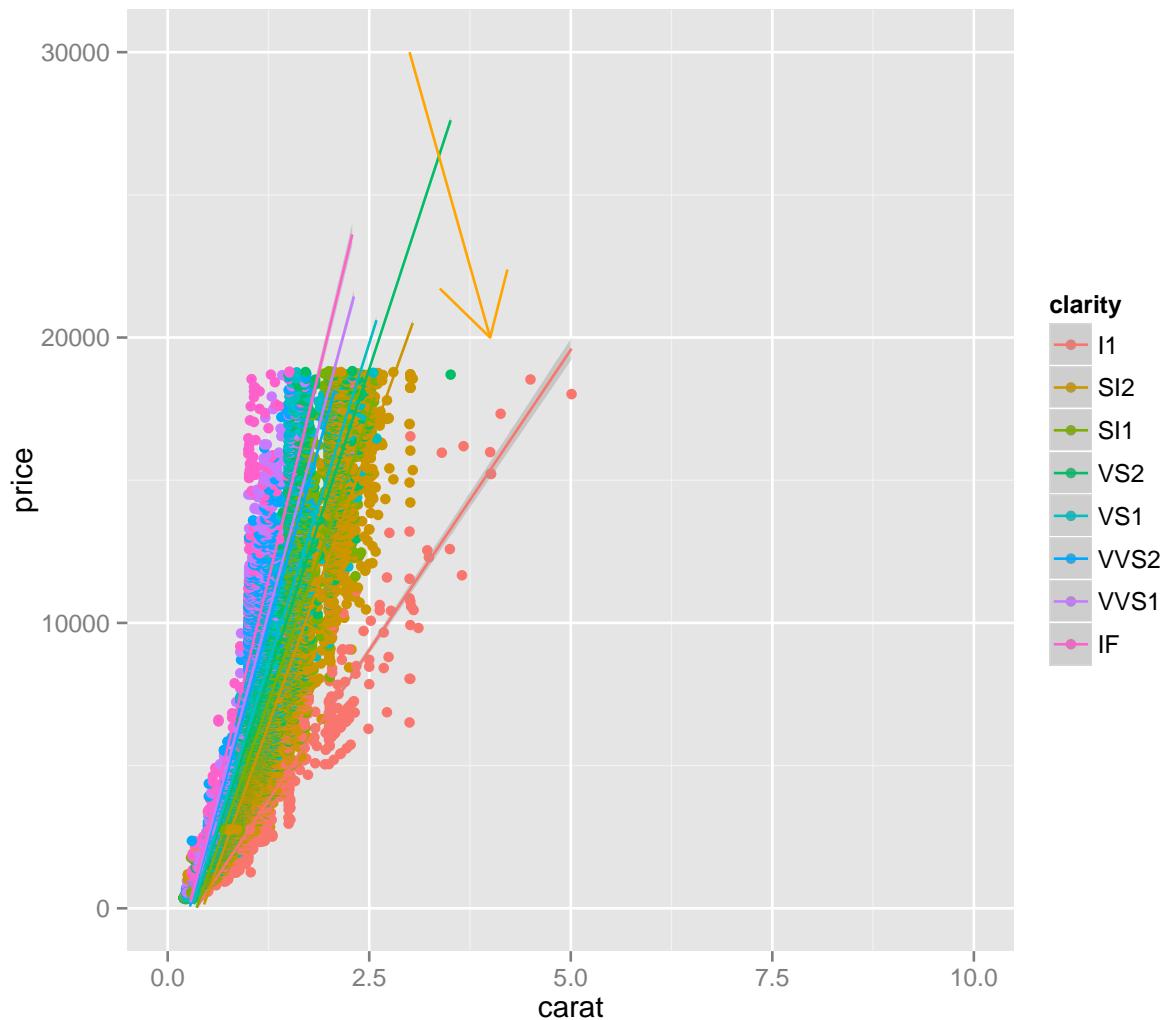
```
g3 + xlim(0,10) + ylim(0,30000) + annotate("text",x=9.5,y=30000,label= "Painel a",color="red")
```



```
g3 + xlim(0,10) + ylim(0,30000) +
  annotate("rect",xmin=3, xmax=4, ymin=10000,
          ymax=20000, alpha=.2, color="blue",fill="green")
```

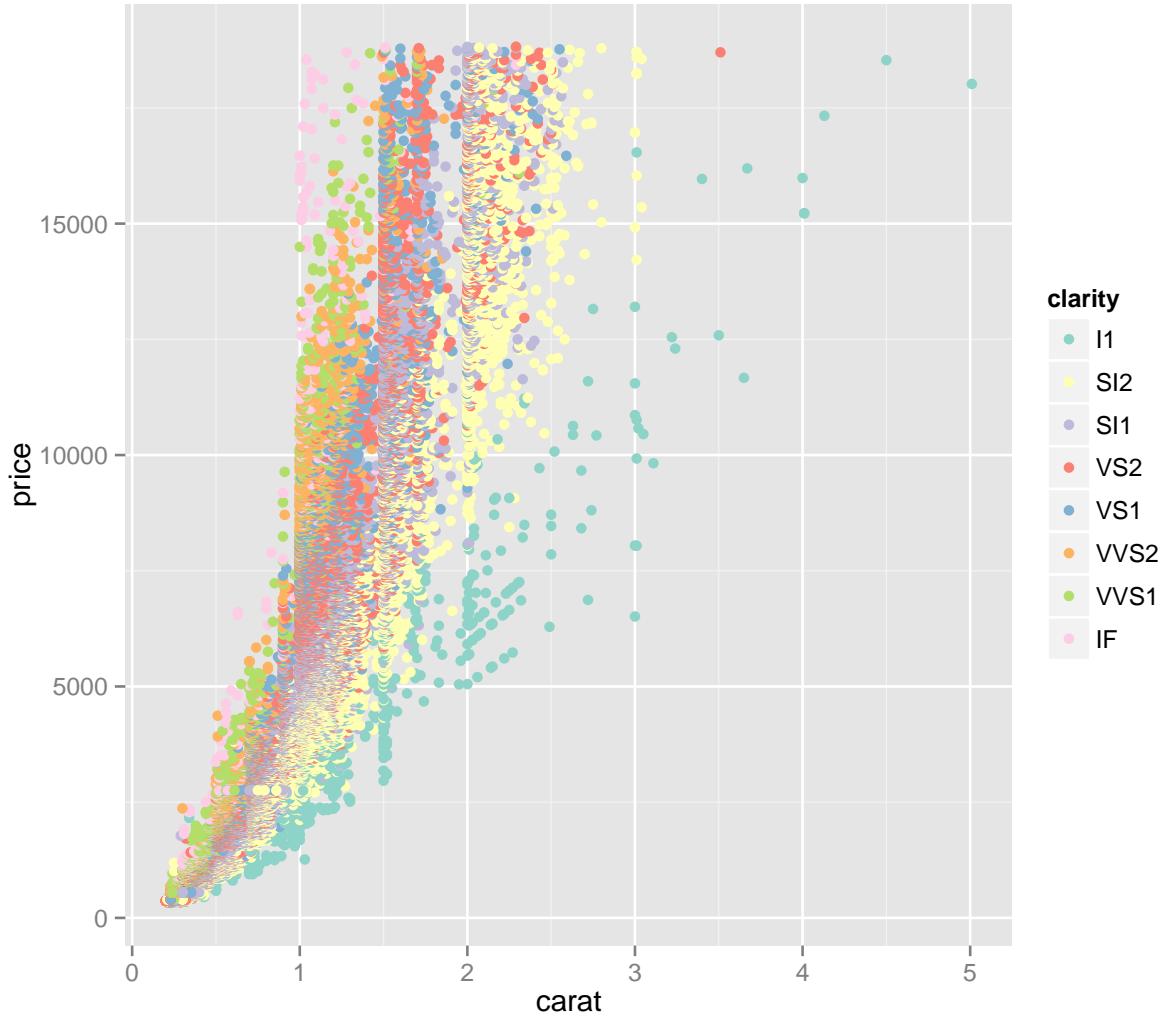


```
g3 + xlim(0,10) + ylim(0,30000) +
  annotate("segment", x=3, xend=4, y=30000, yend=20000,
           color="orange", arrow=arrow(length=unit(1,"cm")))
```



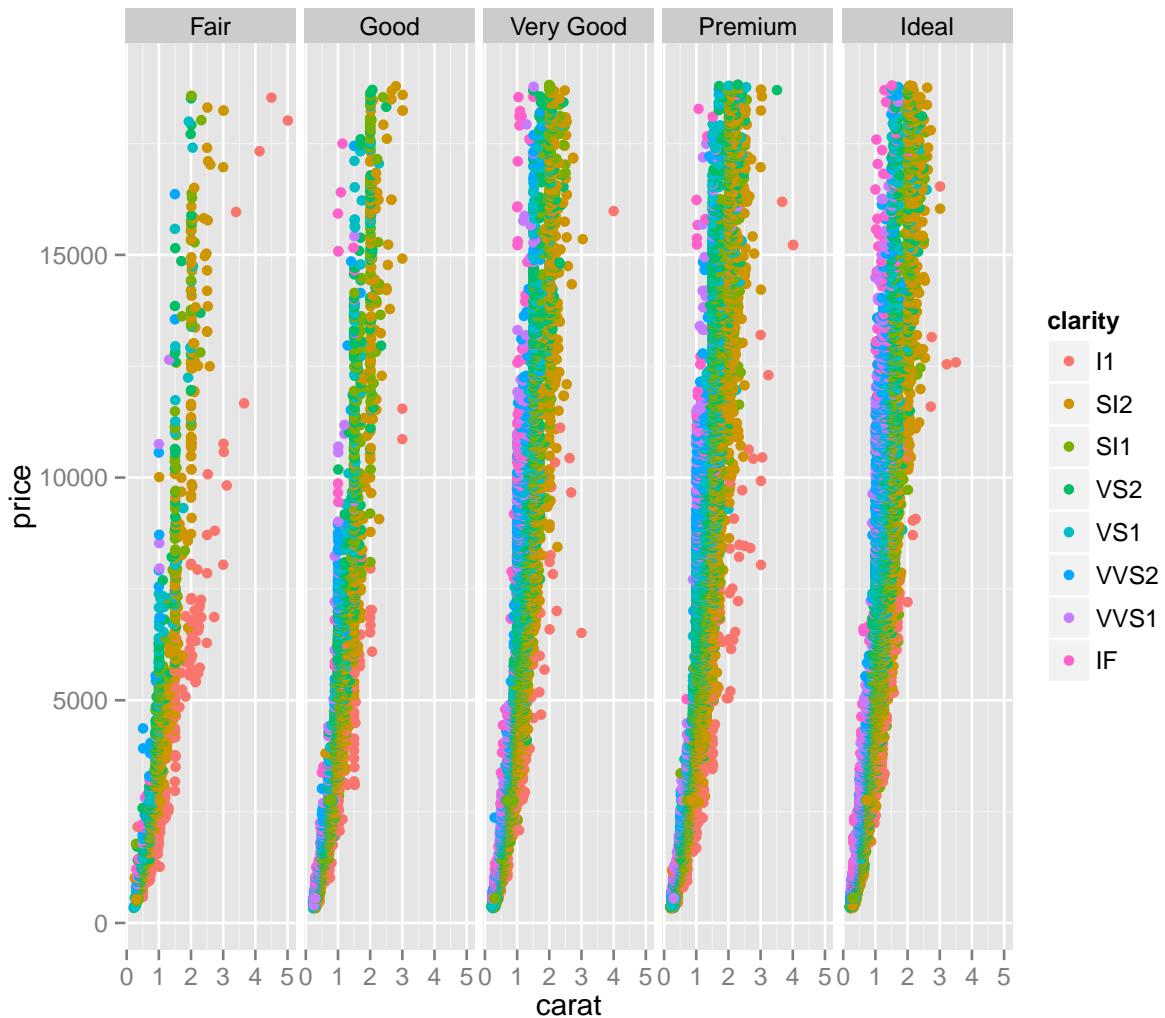
4.5 Alterar paleta de cores

```
g2 + geom_point(aes(color=clarity)) + scale_color_brewer(type="seq", palette="Set3")
```

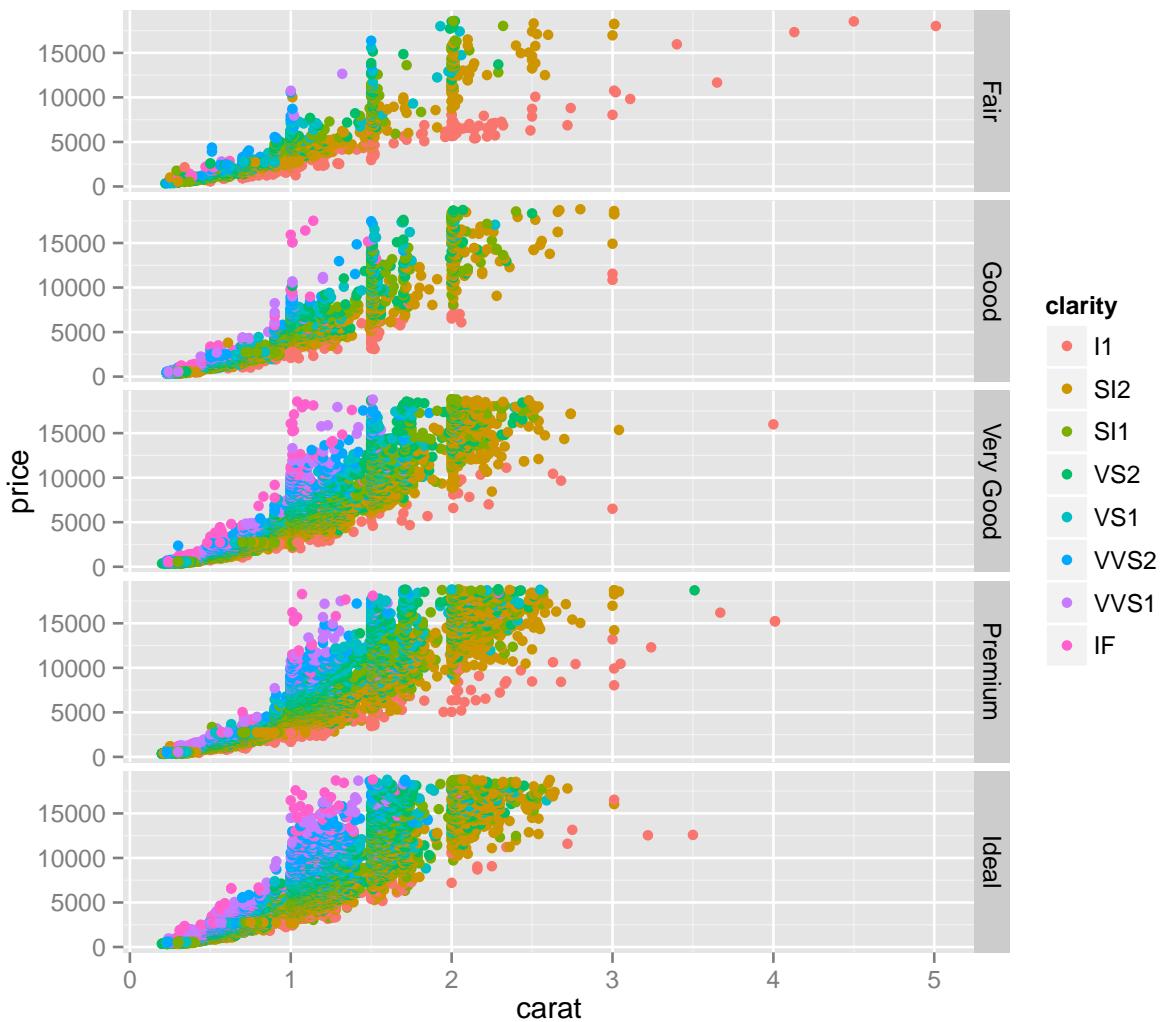


4.6 Separar gráficos em painéis

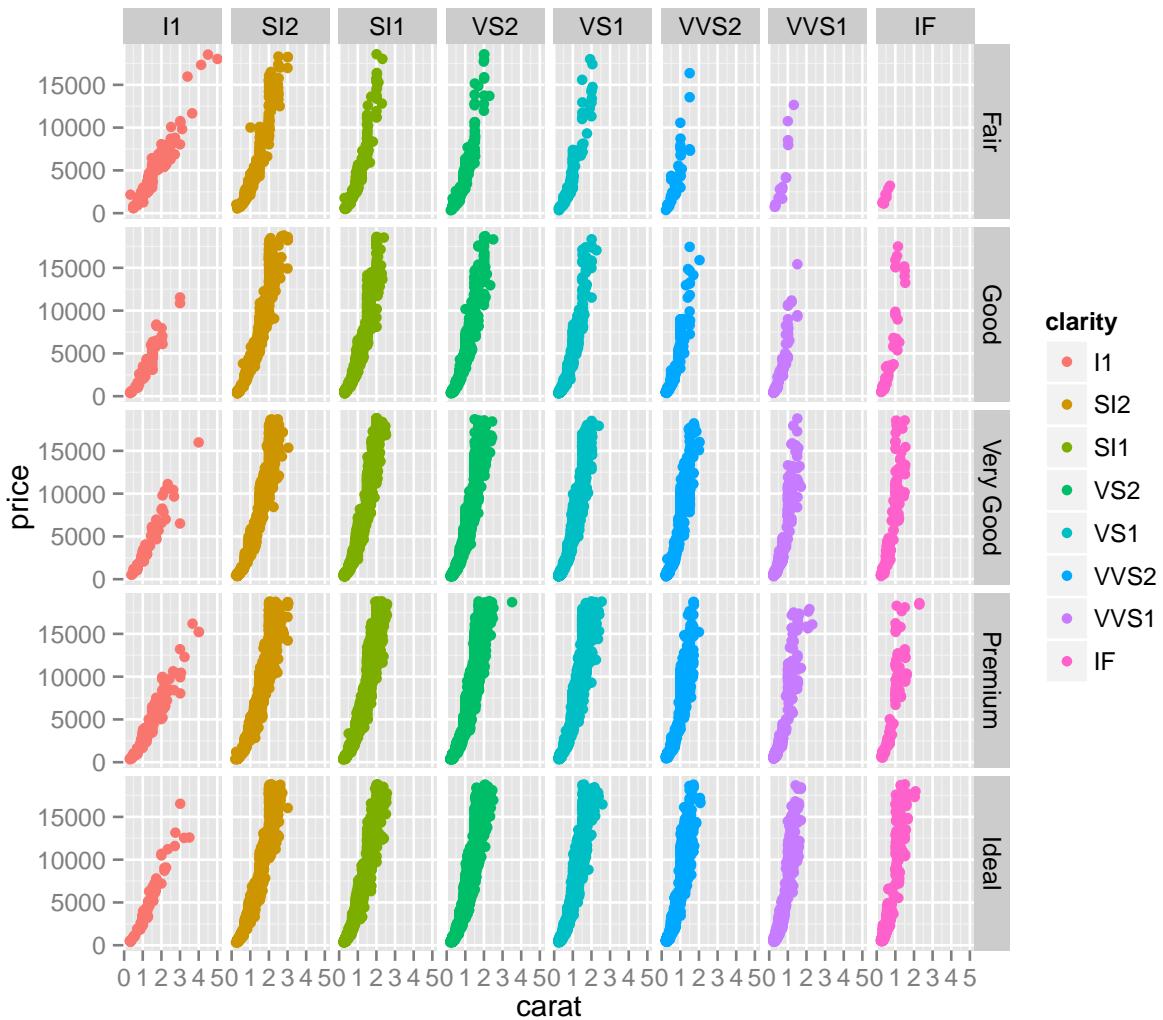
```
g2 + geom_point(aes(color=clarity)) + facet_grid(. ~ cut)
```



```
g2 + geom_point(aes(color=clarity)) + facet_grid(cut ~ .)
```

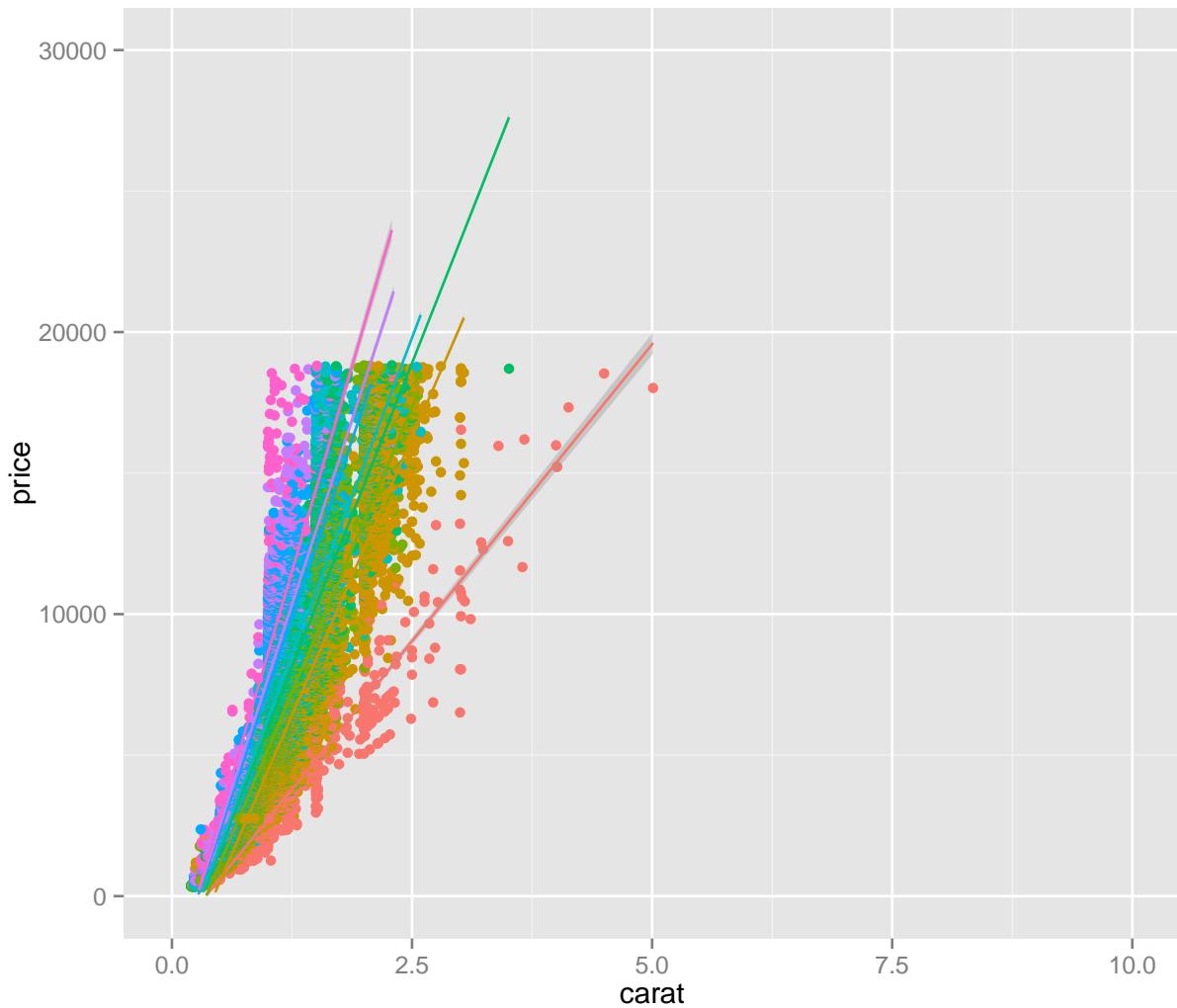


```
g2 + geom_point(aes(color=clarity)) + facet_grid(cut ~ clarity)
```

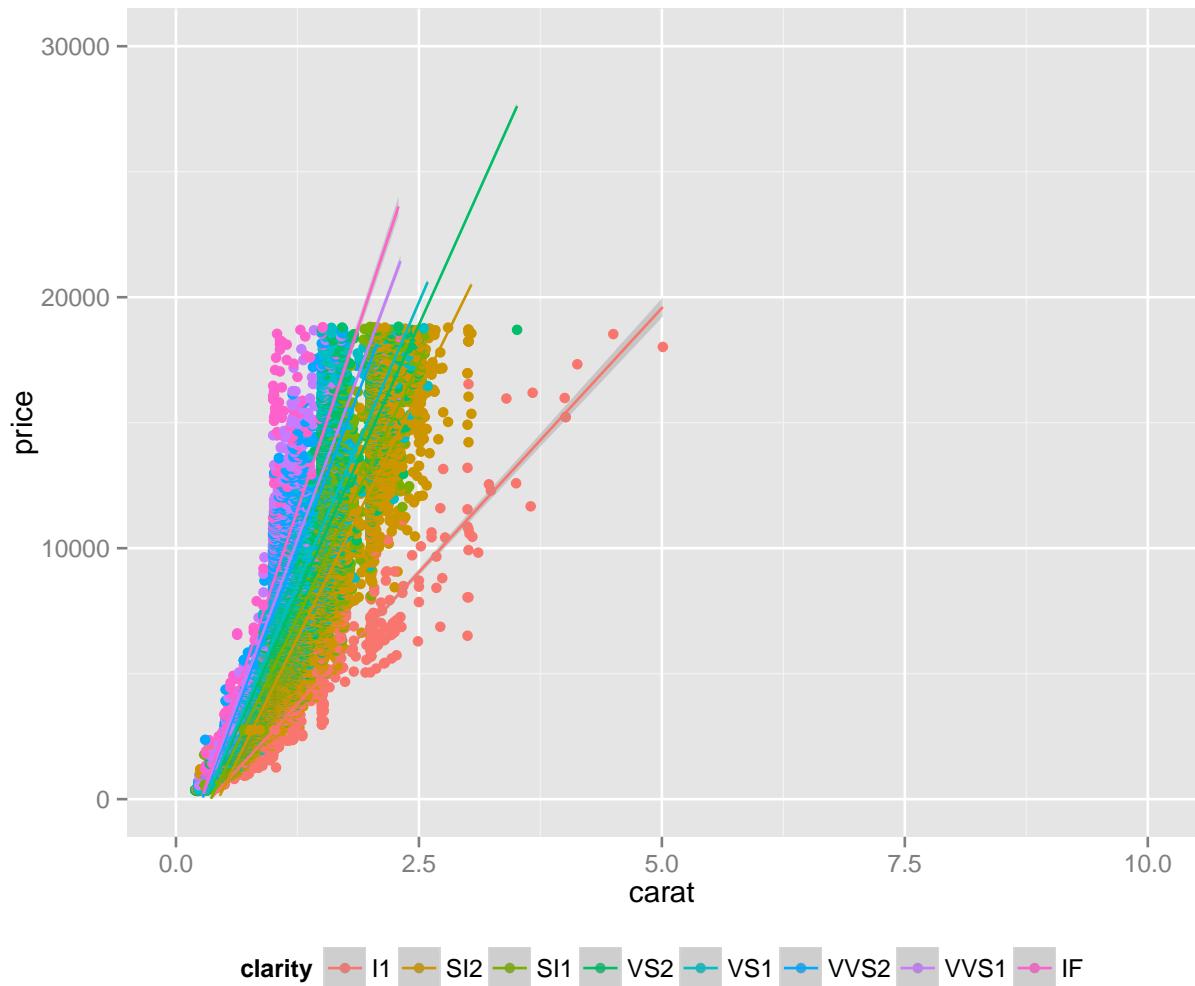


4.7 Alterar posição, título e tamanho da legenda

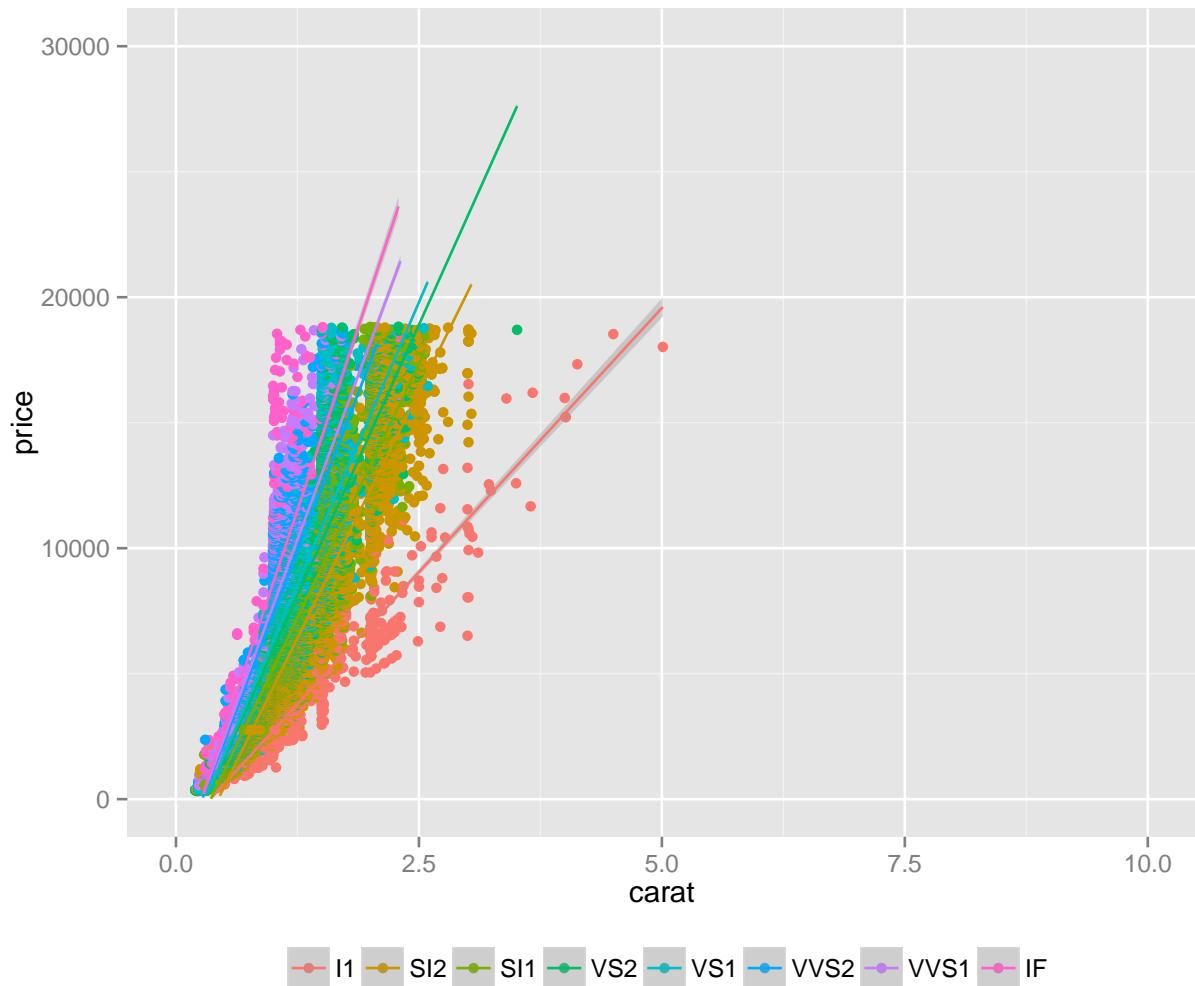
```
g3 + xlim(0,10) + ylim(0,30000) + theme(legend.position = "none")
```



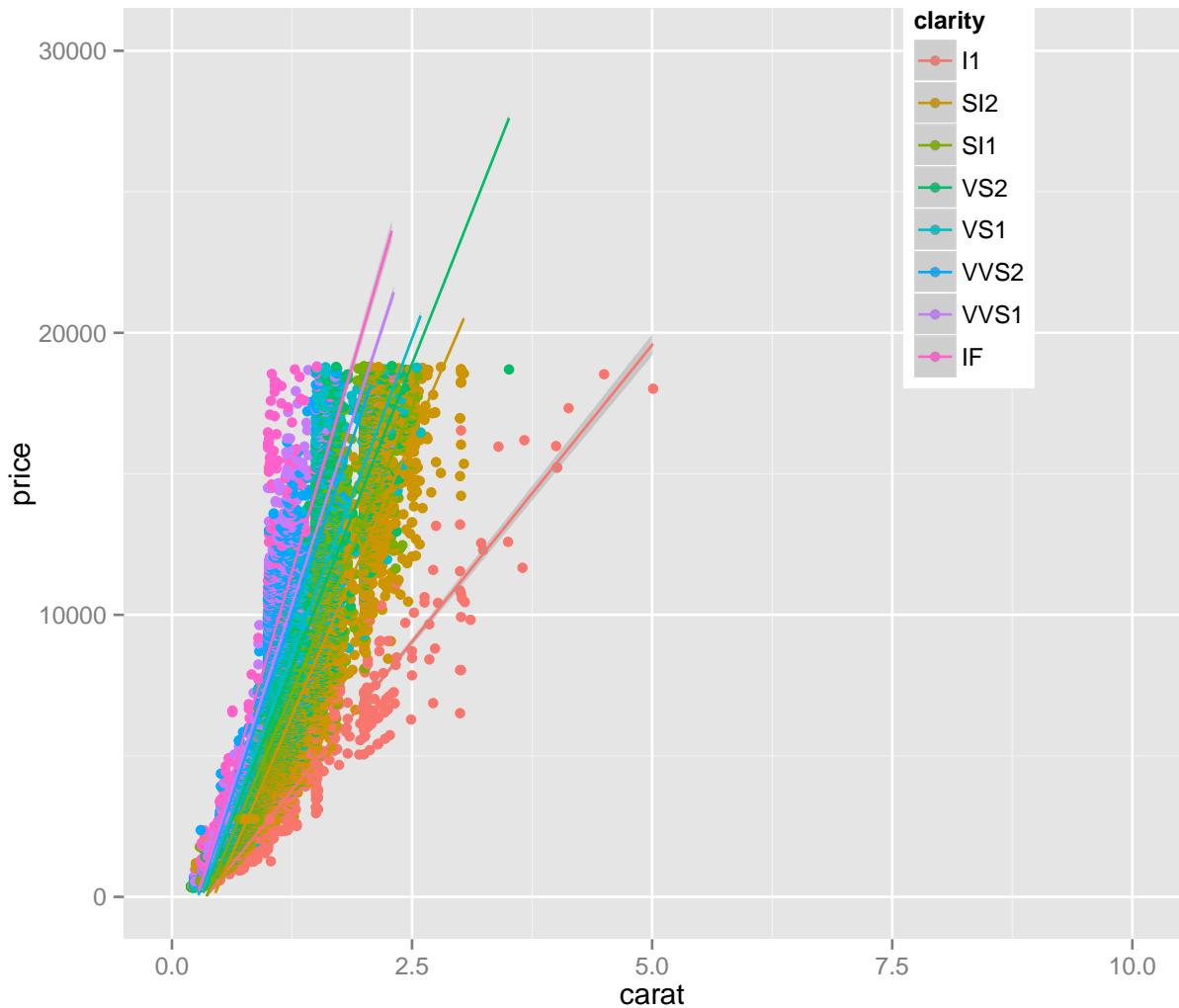
```
g3 + xlim(0,10) + ylim(0,30000) + theme(legend.position = "bottom")
```



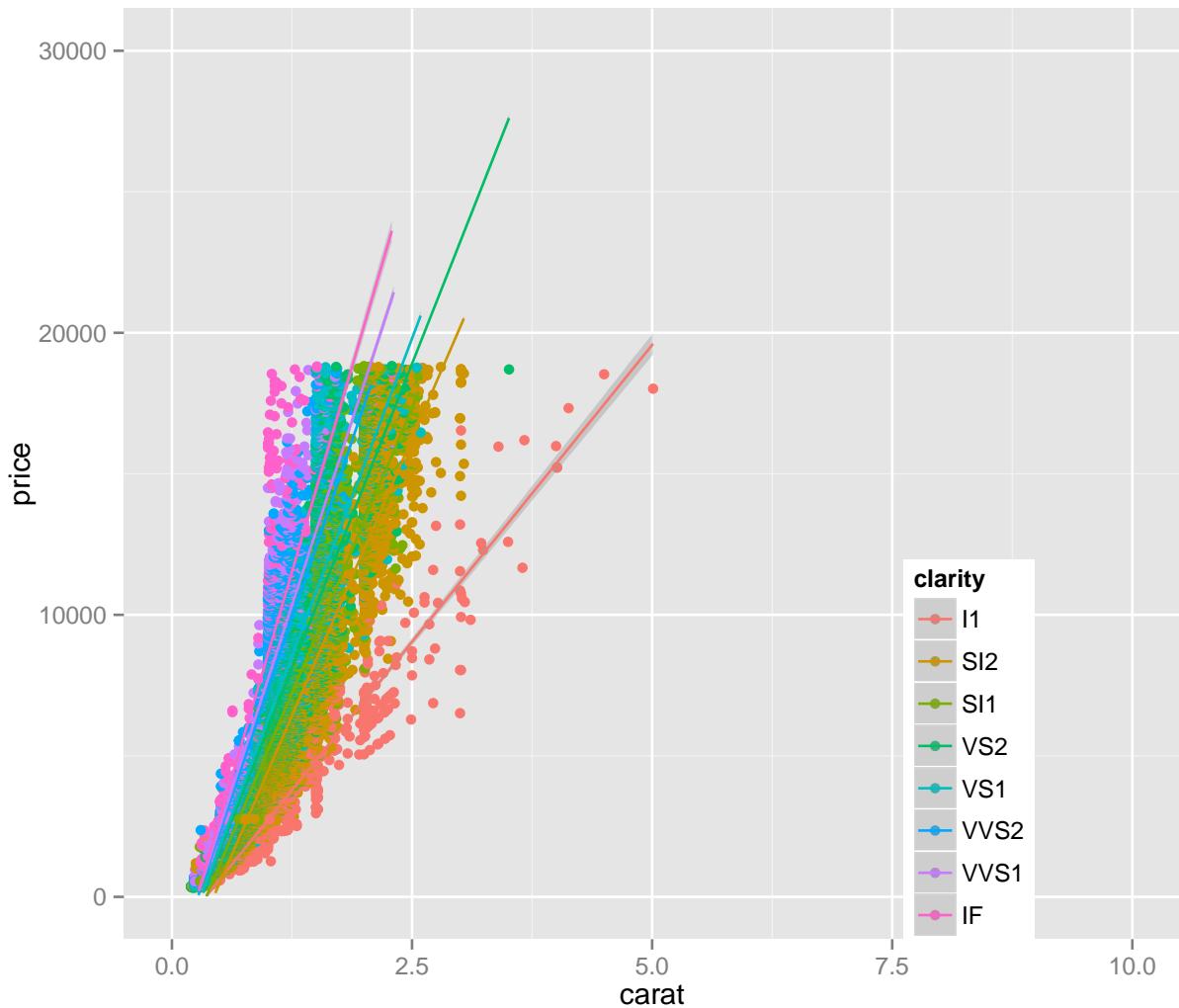
```
g3 + xlim(0,10) + ylim(0,30000) + theme(legend.position = "bottom", legend.title = element_blank())
```



```
g3 + xlim(0,10) + ylim(0,30000) + theme(legend.position = c(.8,.8))
```

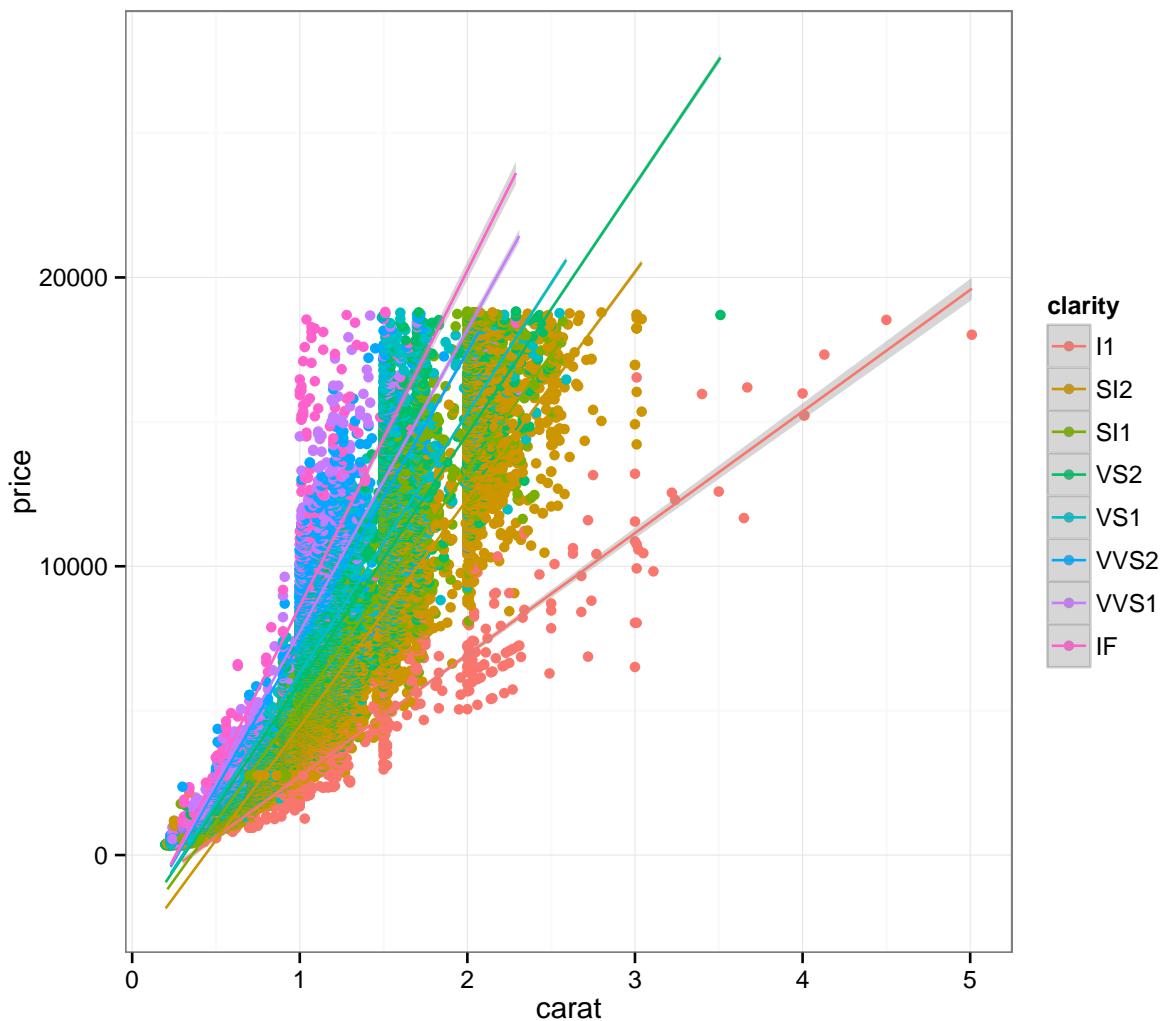


```
g3 + xlim(0,10) + ylim(0,30000) + theme(legend.position = c(.8,.2))
```



4.8 Alterar temas

```
g3 + theme_bw()
```



```
g3 + theme(panel.grid.major = element_line(colour = "red", linetype = "dotted"), panel.background = ele
```

