Contents

Appendix to Basics in Exploratory Data Analysis Using R

This is a companion document to *Basics in Exploratory Data Analysis* focused entirely on the language syntax of R and the **LaTeX** markup used to generate the dynamic pdf. For this purpose, the R code used is largely retained while all the evaluation has been removed. Additionally, since it will be helpful to present this information in a format easier to lookup, a reference style list has been created to keep track of all the different functions and commands used not only in R, but also in **LaTeX** and to a smaller degree, other alternative markup languages to consider such as **Markdown** and a presentation specific tool for R, **slidify**.

To get the simpler cosmetic issues out of the way first, here are some **LaTeX** commands useful for creating publication grade articles.

1 LaTeXSyntax

1.1 Document level formatting

```
\documentclass[10pt]{article}: At the begining of document, to declare the document class.
                               We can pass addtional paramters into [], such as font size,
                               twocolumn, landscape, etc.
\addtolength{\oddsidemargin}{-.875in}: To change the default margins for the pages.
\addtolength{\textwidth}{1.75in}: To change the default width (or height) that the text
                                  goes out to.
```

1.2 Title page formatting

```
\begin{document}: To begin the environment called document. Used with \end{document}
\tableofcontents: To request table of contents.
\title{title name}: To create a title.
\date{}: To date the document. To avoid adding a date, leave it blank.
\maketitle: To request the title to be made.
```

1.3 Creating special elements

```
\parbox{15cm}: To create a paragraph box; the parameter inside specifies the width of the text.
\fbox{}: To create a framed box
\textbf{}: To create bold-faced content
\textit{}: To create italicized content
\texttt{}: To create typewriter font
\vspace{5mm}: To add a certain vertical space
\\: To force line break
\begin{minipage}[t]{0.4\textwidth}: To add a minipage with alignment and width options.
                                    Used with \end{minipage}
%: To create a comment
\iffalse...\fi: To comment out an entire section of code.
\verb|...|: To create inline verbatim text
```

1.4 Math notation

```
$$...$$: To enter and exit math mode
\Delta: To enter a capital Greek letter as math symbol
```

```
\sigma: To enter a lower-case Greek letter as math symbol
\theta_1: Underscore to create subscript
\theta^2: Hat to create superscript
\theta^{12}: Braces to specify the length of the sub- or super-script
\frac{}{}: To create a fraction with a horizontal dividing line
\sqrt{}: To enclose in square root
\int_a^b: To integrate from a to b
\times: Multiplication symbol
```

1.5 Lists and tables

1.5.1 An enumerated list

```
\begin{enumerate}
\item
\item
...
\end{enumerate}
```

1.5.2 A bullet point list

```
\begin{itemize}
\item
\item
...
\end{itemize}
```

1.5.3 Basic tables

To create the table below, the following code is used. Notice **(a)** How to specify the vertical and horizontal lines **(b)** How to specify the alignment of the table elements **(c)** How to span certain elements over multiple columns **(d)** How to differentiate between different elements of the same row **(e)** How to create a new row and **(f)** How to place the table in the center.

```
\begin{center}
\begin{tabular}{ | r | l | }
  \hline
  \multicolumn{2}{ | c | }{Examples of stochastic processes} \\
  \hline
  Random walk &
  $log(S_{t+\Delta t})=log(S_t)+\mu \Delta t+\sigma \sqrt{\Delta t}Z_t$\\
  Mean-Reverting O-U &
  $dX_t=(\theta_1-\theta_2) dt+\theta_3 dW_t$\\
  Cox-Ingersoll-Ross (CIR) &
  $dX_t=(\theta_1-\theta_2) dt+\theta_3 \sqrt{X_t} dW_t$\\
  \hline
\end{tabular}
\end{center}
```

Examples of stochastic processes	
Random walk	$log(S_{t+\Delta t})=log(S_t)+\mu \Delta t+\sigma \sqrt{\Delta t}Z_t$
Mean-Reverting O-U	$dX_t=(\theta_1-\theta_2) dt+\theta_3 dW_t$
Cox-Ingersoll-Ross (CIR)	$dX_t=(\theta_1-\theta_2) dt+\theta_3 \sqrt{X_t} dW_t$

It needs pointing out that while the majority of this document is written using the RStudio IDE. Its editor is poorly equipped for troubleshooting **LaTeX** syntax errors. One would be much better served to run some large chunks of **LaTeX** code using an editor like Texmaker, which provides readable error messages.

By this time, however, you might wonder by the look of complexity in the code above to achieve a rather simple table why we need to bother with **LaTeX**. It seems fair then, that we list the pros and cons.

Pros

1. Using Rnw scripts in R, we can produce high-quality text and graphics in pdf format
2. Lots of flexibility with all manners of formating options
3. Math notation looks especially professional with math mode
4. Helps achieve the goal to separate content and style
5. Table of contents, bibliography, etc. can be efficiently generated
6. A large number of packages to achieve goals without extensive coding

Cons

1. Large installation size
2. The number of packages has grown beyond manageability. It's hard to identify quickly the right package to use.
3. Each additional package means additional learning curve (looking up community help sites and/or pakcage readmes).

Additionally, since we're using RStudio, everytime we want to use new packages, we'll have to interupt the usual workflow for installation. Granted there exists **Markdown** for R to generate dynamic documents, it is more suited for `html` type presentation and therefore does not deal with page breaks. Even though we can translate from a **Markdown** file to a **LaTeX** file, the default *pandoc* function provided in the **knitr** package can creates unexpected "floats" especially when the document includes figures.

2 R Code Essentials

Before delving right in, there is another tutorial on the uses of two specific packges covered in *Basics* at this GitHub location. As the tutorial requires all the major tools mentioned in this Appendix, the code used to generate it is appended at the end to give an impression of how things look once they are brought together, despite the fact that some of it overlaps with what is already included in *Basics*.

Recall also that the full list of packages used were

- sde (stochastic differential equations)
- lubridate (for massaging dates)
- fGarch (for fitting Garch models)
- xlsx (for importing Excel files)
- evir (for working with Extreme Value distributions and plots)
- ggplot2 (for creating some pretty charts)
- mvtnorm (for multivariate normal)
- MASS (for fitting some common distributions)
- copula (for fitting copulas)
- gtools (used here for applying functions to running windows of data)
- tseries (for using functions/tests associated with time series)
- forecast (for forecasting)
- boot (for bootstraping)

It is not necessary and beyond the scope of this exercise to explain the details of each and every package above. The most straightforward and possibly more efficient way to get familiar with them is by using them, and when running into issues in usage, by searching for solutions available either in formal literature or on the boards of R user groups.

In the example-based text below, the aim is largely to show some rough working code encountered during some of the more frequent and basic tasks. This list is neither comprehensive nor represent the only methods that should be employed in like situations. They do provide, however, a nice sample of miscellanenous functions, techniques in data manipulation, and plotting, namely,

1. simulating simple stochastic processes
2. ploting histograms and densities of data
3. fitting a model using available functions in R packages
4. comparing model-generated data with empirical data
5. plotting forecasts based on time series models
6. doing diagnostic and hypothesis tests such as Q-Q plot, ADF
7. manipulating plots and changing plot layout
8. utilizing R objects by referring to their elements
9. utilizing ready-made statistical reports

2.1 Coding stochastic process

2.1.1 Random Walk

```
# Model:
# s     =log(S)
# r     =diff(s)
# mu    =mean(r)
# sigma =std(r)
# z     =(r-mu)/sigma
s_0=10
mu=0.05/250
sigma=0.2/sqrt(250)
set.seed(123)
x=matrix(s_0,ncol=100,nrow=10^3)    #store the 100 bm's in a matrix for plotting
for (t in 2: 10^3)                  #note how loop is done in R
  x[t,]=x[t-1,]+rnorm(100)*sigma+mu #note the subsetting syntax in R
plot(seq(0,1,le=1000),x[,1],"n",    #note the syntax of the plotting function
     ylim=range(x),xlab="",ylab="") #note the beginning of the string in function call
polygon(c(1:10^3,10^3:1)/10^3,c(apply(x,1,max),
                                rev(apply(x,1,min))),col="gold",bor=F)
polygon(c(1:10^3,10^3:1)/10^3,c(apply(x,1,quantile,.95),
                                rev(apply(x,1,quantile,.05))),col="brown",bor=F)
```

While this is a short chunck, there are quite a number of things to notice:

1. function `matrix(name, nrow=n, ncol=m)` to create blank $n \times m$ matrix
2. function `seq(begin, end, length=n)` to create a sequence of length n, equally spaced between beginning and end
3. fuction `apply(matrix, dimension, function)` to apply a certain function to the rows or columns of a matrix

4. function `polygon(x,y)` to draw polygons with vertices as coordinates x and y

   It is slightly tricky to figure out what the x and y vectors are in this case to trace the correct polygons. It may be easier to simply remember that we need to run `plot` once, even though it could be to plot nothing ("n"); that we need to provide two sequences of x with desired number of "steps" and "step size", the second of which is in reverse order; that finally the y coordinates are two sequences: one for the upper bound values and one for the lower bound, with the lower bound values in reverse order.

5. control stucture `for (i in 1:n){...}`, where we place the calculation for each loop inside the curly brackets

6. control structure `if (...){...} else {...}` is quite similar, where if what's inside () evaluates to true, the first curly bracket executes. This is sometimes followed by `else` or `elseif`.

7. function `plot(x, y, ,type='...', main='...', xlim=..., xlab='...', ...)` to create x-y plot

2.2 Coding basis stats

2.2.1 Histogram

The following code plots the histogram of historical Henry Hub returns data and compares it to that of a normal distribution.

```
require(xlsx)
setwd("C:\\Users\\Yangster\\Dropbox\\Private\\")
histRet= read.xlsx("HistPowerGasSpot_Values2.xlsx",1)
retHH=histRet[which(histRet$Hub=="HH"),]
attach(retHH)
require(lubridate)          #Lubridate is a useful dates package
idx.remove=which(Price==0)  #Clean up HH prices/returns for non-trading days
retHH=retHH[-idx.remove,]
y=ymd(as.character(Date))
par(mar=c(4, 5, 4, 2))      #Note how to set margins of the plotting device
plot(y,Price,"l",xlab="Date",ylab="Returns",main=
       "Henry Hub Historical Returns")
hist(Price,breaks=30,xlim=range(Price),main="Histogram of HH Returns")
set.seed(123)
pSim=rnorm(length(Price),mean=mean(Price),sd=sd(Price))
hist(pSim,breaks=30,add=T,col='red')
```

Here we illustrated the following:

1. function `sedwd()` to set working directory so that subsequent I/O will require less typing of full file path.

2. function `read.xlsx(..., sheet index)` to read data from an Excel file

3. subsetting method `data.frame[which(var==criteria),]` to select rows matching a certain criteria using the combination of `which` and `[]`.

4. function `par(mar=c(5.1,4.1,4.1,2.1))` sets the margins of the plotting device by number of "lines". The numbers here are the default.

5. function `set.seed(123)` sets the seed for the random number generator so that future replication of results using the same code is possible

6. function `hist(data, breaks=n, add=T, col='red')` specifies the histogram parameters including whether to plot over previous by using `(add=T)`.

2.2.2 Fitting GARCH (1,1)

```
require(fGarch)
modGarch <- garchFit(formula=~garch(1,1),data=Price,cond.dist="norm")
summary(modGarch)
str(modGarch)       #Show structure of the object
set.seed(123)
parList=modGarch@fit$par
modGarchSpec=garchSpec(model=parList,cond.dist="norm")
simGarch <- garchSim(modGarchSpec,n=length(y))
plot(y,Price,"l",xlab="Date",ylab="Returns",main=
       "Henry Hub Historical Returns")
lines(y,simGarch,col="red")
predict(modGarch,n.ahead=10,plot=TRUE)
```

Frequently, the first thing one does on encountering a new function such as `garchFit` in the **fGarch** package is to type `?garchFit` in the console. Although not always, the help page will routinely turn up some self-explanatory examples. It's quite obvious for example by observing what is passed into the garchFit function that we want to fit a GARCH(1,1) model, that the dataset to apply the model to is called "Price", and that the conditional distribution of the error terms is specified as normal.

The `garchFit` function, as those in many well-written packages, will return an object of a custom class (in this case called fGARCH) that contains many slots, which can be accessed by using the @ symbol (contrast that with accessing named variables in a data frame by using `$`, with accessing elements in a matrix using `[i,j]`, and with accessing single elements in a list by using `[[...]]`). What type of object is being returned and what it contains is usually found out by executing the `str()` function on the variable where the object is contained. It's also common for such class objects to have methods such as `summary()` specified.

2.2.3 Dependence Structure

```
require(xlsx)
histRet2= read.xlsx("C:\\Users\\Yangster\\Dropbox\\Private\\HistOilGasSpot.xlsx",1)
attach(histRet2)
require(lubridate)
y=ymd(as.character(Date))
range_axis=range(HH)
par(pty="s")
plot(HH,WTI,xlim=range_axis,ylim=range_axis,main="HH vs. WTI")
par(pty="m")
mu_2=c(mean(HH),mean(WTI))
sigma_2=var(cbind(HH,WTI))
require(mvtnorm)
simNorm_2=rmvnorm(length(HH),mean=mu_2,sigma=sigma_2)
par(mfrow=c(1,2))
plot(HH,WTI,xlim=range_axis,ylim=range_axis,main="HH vs. WTI")
plot(simNorm_2,xlim=range_axis,ylim=range_axis,main="Simulated Normal",
     xlab="HH_sim",ylab="WTI_sim")
qqnorm(HH,main="Q-Q for HH")
qqnorm(WTI,main="Q-Q for WTI")
# Plot the returns series side by side
par(mfrow=c(2,1))
par(mar=c(2,2.5,2,1))
plot(y,HH,"l",main=
       "Henry Hub Historical Returns",cex.lab=0.8,cex.main=0.8,cex.axis=0.8)
plot(y,WTI,"l",main=
       "WTI Historical Returns",cex.lab=0.8,cex.main=0.8,cex.axis=0.8)
par(mfrow=c(1,1))
par(mar=c(4,5,4,2))
```

As is often the case with presenting time series data and their analysis, plotting provides visual cues for the type of data one is dealing with and the appropriate techniques. It will become time-saving to memorize the basic options accepted by the function `plot`. Besides `main, xlim, xlabel, type` mentioned above,

```
pch: type of points used
cex: a vector reflecting how things should be scaled relative to default
lwd: line width
```

For more on these parameters, type `?plot.default` in the console. A longer list of manipulatable graphic paramters, type `?par`. The latter command was used in combination with `mfrow=c()` and `mar=c()` to specify the rows and columns of returned plots and the plot margins.

2.2.4 `gtools`, `tseries`, and `forecast`

```
require(gtools)
run_cor=running(HH,WTI,width=20,fun=cor,allow.fewer=FALSE)
plot(Date[20:length(Date)],run_cor,type='s',xlab="")
# Is the running correlation a stationary series?
require(tseries)
adf.test(run_cor)
# Automated ARIMA
require(forecast)
cor.fit=auto.arima(run_cor)
plot(forecast(cor.fit,h=20))
set.seed(123)
plot(simulate(cor.fit,nsim=length(run_cor)),lwd='2',col='red',ylab='',xlab='')
cor.mat=matrix(ncol=1000,nrow=length(run_cor))
for (i in 1:1000)
  cor.mat[,i]=simulate(cor.fit,nsim=length(run_cor))
hist(cor.mat[length(run_cor),],breaks=30,main="",xlab='')
# Summary report
summary(cor.fit)
```

This is a good example of when R can save us from having to write our own functions. Function `running` in the **gtools** package is ideally suited here for getting moving averages of the desired number of data points. `adf.test` in the **tseries** package returns the statistics of Augmented Dickey-Fuller test. The `auto.arima` and `forecast` functions in the **forecast** package, which is more extensively written about in the slide deck referred to above, provides excellent coverage of ARIMA modeling and forecasting that otherwise would have required a lot of code writing or repetitive execution of existing functions (to find the best-fit model, etc.). The forecast object even come with a decent default plot method!

3 Using Markdown and slidify

This section on using **R Markdown** and **slidify** returns us back to ways of presenting data and results of data analysis. While LaTeXis powerful for producing professional documents, sometimes we only need a html format without having to worry about page breaks or be fastidious about layout and formatting. Or, we want to have a distilled version of our research to present to an audience but would like to retain the dynamic nature not easily afforded by Microsoft Powerpoint. In situations like these, **R Markdown** and **slidify** are capable tools to consider. We run over some basics of each below. They are by no means meant to be comprehensive (as is apparent by the length of the sections) but are probably the first things that one is likely to look up as a novice.

3.1 R Markdown

Since **Markdown** is a simple markup language, we provide links to some help pages that can be used for self-study.

```
http://www.rstudio.com/ide/docs/authoring/using_markdown
http://yihui.name/knitr/
https://daringfireball.net/projects/markdown/syntax
```

It is advised that one read through the second site and in particular, the Options tab, which goes to some length to explain some of the more complicated features of the language. These include how to configure graphic output of R code, how to use the cache functionality provided and its pitfalls (using it in chunks that include loading new packages should be avoided).

3.1.1 Lists

```
Ordered lists:

1. item name
2. item name
3. item name

Unordered lists:

* item name
* item name
```

3.1.2 Headers

```
This is an H1
=============

This is an H2
-------------

# This is an H1

## This is an H2

###### This is an H6
```

3.2 slidify

As the name suggests, **slidify** is an R package that creates impressive presentation slide decks based on **R Markdown**. It is contributed by Mr. Ramnath (whose first initial is V, hence his GitHub username ramnathv), who is also behind the package **rCharts**. The short demo video at `slidify.org` is a good place to start.

Things to remember:

1. Use `---` to separte one slide from the next
2. Use `> 1. item name` to create an animated bullet point
3. In fact, `>` can be used with an unordered list as well to produce animation. Although it does not seem to work with sub-bullets.
4. Use 'as is' for the `results` parameter in any R code chunk whose output is a figure

5. Since `gvisMotionChart` in the **googleVis** package produces output that uses flash and there are security issues preventing these motion charts to be reflected correctly when running from a local directory, it is necessary to use a hosting server to view such charts. One easy trick is to upload the code files to GitHub and navigate to the raw view of the index.html file and remove the dot in ...raw.github.com... to say ...rawgithub.com...

Since **slidify** is just a wrapper around packages such as **knitr** and borrows HTML5 templates from different individuals and organizations, it does not come with a lot of documentation. For example, the demo slide deck uses the style called "io2012", which comes from Google I/O 2012. The template has its own little intro here.

Additionally, unless one is familiar with web development tools used to create these frameworks, deep customization is not straightforward. I, for example, hunted for a while for the correct item in a CSS file to configure in order to change the font size of some elements on the slides. The lack of documentation and difficulty in customization should, however, not deter one from trying out the different frameworks included in the package as they are often sufficient for general purpose presentations.

I'll mention in passing that the package **rCharts** is similarly constrained by the fact that it borrows chart libraries written in javascript, as is the **googleVis** package. There is a rudimentary how-to for the former here and a very extensive API documentation for the latter here. One has to make the decision as to how much learning is needed, and whether it's worth it, in order to fine-tune graphic elements in these charting tools, no matter how awesome the final results may look.

4 Example Code

4.1 R, slidify and Markdown examples

```
---
title      : Brief Introduction to R
subtitle   : Why you should learn this "language"?
author     : David Yang
job        : R Enthusiast
framework  : io2012      # {io2012, html5slides, shower, dzslides, ...}
highlighter : highlight.js # {highlight.js, prettify, highlight}
hitheme    : tomorrow    #
widgets    : []          # {mathjax, quiz, bootstrap}
mode       : selfcontained # {standalone, draft}
---

## Agenda

1. Introduce some basic concepts of the statistical computing language
2. Introduce two packages that are powerful and popular
3. Point to further uses of R with a fun example

---

## This is easier than you think:

> 1. R is a high level language that skips over most programming niceties
> 2. R has a big support community and more packages than you have time to explore
> 3. Anything you want to achieve, someone probably has done it in R, and done it beautifully

---

## Let's start with `ggplot2`
```

~~~
> * Contributed by Hadley Wickham of Rice University
> * Some data to start us off
> * Historical price-load relationship during August based on hourly PJM data
```{r echo=F, fig.width=5, fig.height=5,results='asis',message=F,cache=T}
require(xlsx)
require(ggplot2)
priceload<-read.csv("C:\\Users\\Yangster\\Dropbox\\Private\\3AugHrlyLdPrice_PJM.csv")
p<-ggplot(priceload,aes(x=E_W_LD,y=Price))+ geom_point()+xlab("Load")+
  ylab("Price")
p+facet_grid(Date~.)
p+facet_grid(Date~.)+geom_smooth()
~~~

## What happened

```{r echo=T, eval=F}
require(xlsx)
require(ggplot2)
priceload<-read.csv("C:\\Users\\Yangster\\Dropbox\\Private\\3AugHrlyLdPrice_PJM.csv")
p<-ggplot(priceload,aes(x=E_W_LD,y=Price))+ geom_point()+xlab("Load")+
  ylab("Price")
p+facet_grid(Date~.)
p+facet_grid(Date~.)+geom_smooth()
```

With three simple lines of code, we
* Imported data from an Excel file
* Created elegant charts using `ggplot2`'s graphics language
  * using aesthetics (aes) to create mapping of data to graphic elements
  * using geometry (geom) to specify the way we want to present the data including
      sophisticated "faceting" and non-parametric regression

## Data visualization done right

Let's look at another example using historical forward data for Henry Hub natural gas for
    August 2013 delivery. Trade dates of each month are colored differently, a local regression
     line is produced as well as it's error bands.

```{r echo=F,fig.align='center',fig.width=7,fig.height=5,results='asis',message=F,cache=TRUE}
require(xlsx)
require(lubridate)
require(ggplot2)
futNGDel= read.xlsx("C:\\Users\\Yangster\\Dropbox\\Private\\FUTURE_NG_Del.xlsx",2,header=TRUE)
s2<-ggplot(data=subset(futNGDel,subset=year(DelMo)==2013 &
                      month(DelMo)==8 & TradeDate>as.Date('4/1/2013','%m/%d/%Y')),
         aes(x=TradeDate,y=Price,colour=as.factor(month(TradeDate))))
s2+geom_point()+geom_smooth()+theme(legend.position="none")
```

## Some programming asides

```{r echo=T, eval=F}
require(xlsx)
require(lubridate)
require(ggplot2)
```

~~~
futNGDel= read.xlsx("C:\\Users\\Yangster\\Dropbox\\Private\\FUTURE_NG_Del.xlsx",2,header=TRUE)
s2<-ggplot(data=subset(futNGDel,subset=year(DelMo)==2013 &
                     month(DelMo)==8 & TradeDate>as.Date('4/1/2013','%m/%d/%Y')),
         aes(x=TradeDate,y=Price,colour=as.factor(month(TradeDate))))
s2+geom_point()+geom_smooth()+theme(legend.position="none")
```

> * We do need to mention some R-related programming knowledge such as the following
   * How to subset data
   * How to use `month()` and `year()` functions in the lubridate package to extract
       information from dates
   * How to convert dates stored as `factors` to dates stored as `dates` and back

---

## A lot more cool stuff awaits

Once we are more familiar with R, we can do a lot by typing relatively little. What's more
    important is probably the dynamic and reproducible nature of it.

Any guesses what the code below will do?

```{r echo=T, eval=F}
require(xlsx)
require(lubridate)
require(ggplot2)
# Importing and processing of data
price.Seg= read.csv("C:\\Users\\Yangster\\Dropbox\\Private\\3AugHrlyLdPrice_PJM_2.csv",header=
    TRUE)
price.Seg$Date_Long<-as.Date(price.Seg$Date_Long,"%m/%d/%Y")
price.Seg<-price.Seg[,c('Date_Long','Date','Price','SegType')]
aggreg.Seg<-aggregate(price.Seg,list(price.Seg$SegType,price.Seg$Date_Long),mean)
aggreg.Seg<-aggreg.Seg[,c(1,2,4,5)]
names(aggreg.Seg)<-c('peaktype','date','year','price')
```

---

## (continued from above)

```{r echo=T, eval=F}
aggreg.Seg<-aggreg.Seg[order(aggreg.Seg$peaktype,aggreg.Seg$date),]
aggreg.Seg$peaktype<-factor(aggreg.Seg$peaktype,levels=c('OnPeak','OffPeak',"Wknd"))
# Setting up plotting
p_seg<-ggplot(aggreg.Seg,aes(x=date,y=price,colour=factor(year)))+
  geom_point()+xlab("Date")+ylab("Price")
p_seg+facet_grid(peaktype~year,scale='free_x')+
  theme(axis.text.x = element_text(angle = 90, vjust=0.5))
p_seg_hist<-ggplot(aggreg.Seg,aes(x=price,fill=factor(year),colour=factor(year),..density..))+
  geom_density(alpha=0.5,position="identity")
p_seg_hist+facet_grid(peaktype~.,scale='free_x')
```

---

## Side-by-side comparison of August power prices by segment and year

```{r echo=F,fig.width=6.5,fig.height=6.5,results='asis',message=F,warning=FALSE,cache=TRUE}
require(xlsx)
require(lubridate)
~~~

~~~
require(ggplot2)
# Importing and processing of data
price.Seg= read.csv("C:\\Users\\Yangster\\Dropbox\\Private\\3AugHrlyLdPrice_PJM_2.csv",header=
    TRUE)
price.Seg$Date_Long<-as.Date(price.Seg$Date_Long,"%m/%d/%Y")
price.Seg<-price.Seg[,c('Date_Long','Date','Price','SegType')]
aggreg.Seg<-aggregate(price.Seg,list(price.Seg$SegType,price.Seg$Date_Long),mean)
aggreg.Seg<-aggreg.Seg[,c(1,2,4,5)]
names(aggreg.Seg)<-c('peaktype','date','year','price')
aggreg.Seg<-aggreg.Seg[order(aggreg.Seg$peaktype,aggreg.Seg$date),]
aggreg.Seg$peaktype<-factor(aggreg.Seg$peaktype,levels=c('OnPeak','OffPeak',"Wknd"))
# Setting up plotting
p_seg<-ggplot(aggreg.Seg,aes(x=date,y=price,colour=factor(year)))+
  geom_point()+xlab("Date")+ylab("Price")
p_seg+facet_grid(peaktype~year,scale='free_x')+
  theme(axis.text.x = element_text(angle = 90, vjust=0.5))
p_seg_hist<-ggplot(aggreg.Seg,aes(x=price,fill=factor(year),colour=factor(year),..density..))+
  geom_density(alpha=0.5,position="identity")
p_seg_hist+facet_grid(peaktype~.,scale='free_x')
```

---

## Now let's look at `forecast`

> * Contributed by Rob Hyndman of Monash University, Australia
> * An excellent source of information on time series modeling using R is here: https://www.
    otexts.org/fpp/resources
> * The package `forecast` enables complex methods to be invoked with just the right function
    call
> * We'll see an example using a double-seasonality Holt-Winters procedure to forecast hourly
    power prices using the same hourly PJM data
   * The double seasonality refers to both the daily cycle as well as the weekly (weekday/
       weekend) cycle
> * The example uses a training set of 600 hourly data; the forecast (blue) is compared to the
    actual out-of-sample data of hour number 601 to 744 (plotted in black)

---

## Automated time-series forecast

```{r echo=F, message=F,warning=FALSE,cache=TRUE}
require(forecast)
test12<-price.Seg[price.Seg$Date==2012,]$Price
test12.train<-test12[1:600]
test12.rest<-test12[601:length(test12)]
test12.train<-ts(test12.train,start=1,deltat=1)
fcast.test12<-dshw(test12.train, period1=24,period2=24*7)
```

```{r echo=F, message=FALSE,warning=FALSE,results='asis'}
library(googleVis)
op <- options(gvis.plot.tag = "chart")
a1<-test12
a2=c(test12.train,fcast.test12$mean[1:144])
temp12<-seq(as.POSIXct("2012-08-01 01:00:00"), as.POSIXct("2012-09-01 00:00:00"), by="hour")
temp12.dat<-cbind.data.frame(temp12,a1,a2)
names(temp12.dat)<-c('Date','Actual','Forecast')
m2 <- gvisLineChart(temp12.dat,xvar = 'Date', yvar = c('Forecast','Actual'),
                  options=list(
~~~

```
                   title="DSHW Forecast for August 2012",
                   series="[{color:'blue'},
                          {color: 'black'}]",
                   hAxis="{textPosition:'out'}",
                   width=900, height=550))
plot(m2)
```

## Automated time-series forecast

```{r echo=F, message=F,warning=FALSE,cache=TRUE}
test13<-price.Seg[price.Seg$Date==2013,]$Price
test13.train<-test13[1:600]
test13.rest<-test13[601:length(test13)]
test13.train<-ts(test13.train,start=1,deltat=1)
fcast.test13<-dshw(test13.train, period1=24,period2=24*7)
```

```{r echo=F, results='asis'}
a3<-test13
a4=c(test13.train,fcast.test13$mean[1:144])
temp13<-seq(as.POSIXct("2013-08-01 01:00:00"), as.POSIXct("2013-09-01 00:00:00"), by="hour")
temp13.dat<-cbind.data.frame(temp13,a3,a4)
names(temp13.dat)<-c('Date','Actual','Forecast')
m3 <- gvisLineChart(temp13.dat,xvar = 'Date', yvar = c('Forecast','Actual'),
                  options=list(
                   title="DSHW Forecast for August 2013",
                   series="[{color:'blue'},
                          {color: 'black'}]",
                   hAxis="{textPosition:'out'}",
                   width=900, height=550))
plot(m3)
```

## Let's have some fun, finally

Here is a neat way of visualizing 10 paths of random walk. This is realized in R using the
    googleVis package.
```{r, message=FALSE,echo=FALSE}
require(reshape2)
s_0=10
mu=0.05/250
sigma=0.2/sqrt(250)
set.seed(123)
x=matrix(s_0,ncol=10,nrow=50)
for (t in 2: 50)
  x[t,]=x[t-1,]+rnorm(10)*sigma+mu
colnames(x)<-paste('Path',seq(1,10,by=1),sep="_")
new_x<-melt(x)
# Note that cbind alone will convert Date objects, use the following syntax instead
# Note also that cbind recycles the variable of shorter length
idx.Time<-seq.Date(as.Date('2011-01-01'),le=50,by='1 day')
x_chart<-cbind.data.frame(idx.Time,new_x)
colnames(x_chart)<-c("time","idx","path","price")
```
```
```

~~~
```{r, results='asis',echo=F}
BB.motion<-gvisMotionChart(x_chart,idvar="path",timevar="time",
                  xvar="time",yvar="price",options = list(width = 600, height = 450))
plot(BB.motion)
## Set options back to original options
options(op)
```

---
~~~

4.2 LaTeXexamples

```
\documentclass[10pt]{article}

\usepackage{mdframed,xcolor,hyperref,listings}

\lstset{
basicstyle=\small\ttfamily,
columns=flexible,
breaklines=true
}

\addtolength{\oddsidemargin}{-.875in}
\addtolength{\evensidemargin}{-.875in}
\addtolength{\textwidth}{1.75in}
\addtolength{\topmargin}{-.9in}
\addtolength{\textheight}{1.8in}

\begin{document}

\tableofcontents

\title{Appendix to Basics in Exploratory Data Analysis Using R}
\date{}
\maketitle

% Body of the document is not shown for brevity

\end{document}
```