Final Project: Walmart Data Exploration

Stephen Kappel (spk2131), Mayank Misra (mm3557), Mandeep Singh (ms4826)

Due: December 5, 2015

Introduction

For our project, we chose to use sales data provided by Walmart as part of the *Walmart Recruiting - Store Sales Forecasting* competition on Kaggle in 2014. (See https://www.kaggle.com/c/walmart-recruiting-store-sales-forecasting.) The dataset contains weekly sales for 45 Walmart stores from 2/5/2010 to 11/1/2012. The sales are broken out by store and by department. Other attributes provide further context to the sales numbers:

- *Economic indicators:* CPI, unemployment rate, fuel prices
- *Store-specific attributes:* store size, store type, markdowns (indicator variables for five types of markdowns)
- *Other:* temperature, holiday (indicator variable)

In our analysis, sales is the outcome variable of primary interest. We aim to understand what factors impact sales. In particular, we try to answer:

1. How do economic factors – over which Walmart has no control – affect Walmart's sales?
2. How well can we predict sales? In predicting sales, which predictors are most significant?

Data Preparation

Kaggle provides the data in the form of three CSVs. We combined the three files together to aggregate weekly sales by store in a single `data.frame` with all associated features. This was implemented using the merge function. For better visualization and analysis, we categorized certain fields in to classes.

```
require(plyr)

get.stores <- function(){
  stores <- read.csv('../data/stores.csv')
  stores$Store <- as.factor(stores$Store)
  return(stores)
}

get.features <- function(){
  features <- read.csv('../data/features.csv')
  features$Store <- as.factor(features$Store)
  features$Date <- as.Date(features$Date)
  return(features)
}

get.train <- function(){
  train <- read.csv('../data/train.csv')
  train$Store <- as.factor(train$Store)
  train$Dept <- as.factor(train$Dept)
  train$Date <- as.Date(train$Date)
```

```
  return(train)
}

get.store.level <- function(){
  # construct a data.frame with detail down to the store level (no departments)
  train <- get.train()
  train <- ddply(train, c('Store', 'Date'), summarize, Sales=sum(Weekly_Sales))
  stores <- get.stores()
  features <- get.features()
  merged <- merge(train, stores, by='Store')
  merged <- merge(merged, features, by=c('Store', 'Date'))
  merged$Sales_Millions <- merged$Sales / 10e6
  merged <- rename(merged, c('Type'='Store_Type', 'Size'='Store_Size'))
  return(merged)
}

get.dept.level <- function(){
  # construct a data.frame with detail down to the department level
  train <- get.train()
  stores <- get.stores()
  features <- get.features()
  merged <- merge(train, stores, by='Store')
  merged <- merge(merged, features, by=c('Store', 'Date'))
  merged <- rename(merged, c('Type'='Store_Type', 'Size'='Store_Size',
                             'Weekly_Sales'='Sales'))
  merged$Sales_Millions <- merged$Sales / 10e6
  return(merged)
}

store.level <- get.store.level()
dept.level <- get.dept.level()
```

As we do our analysis, we often find it helpful to normalize a column within the context of a store. This allows for more meaningful comparisons. Because this is a common operation, we define a function to add a nomalized column to a given data.frame for a specified column/attribute. To start, we add a normalized sales column to the `store.level` data.frame.

```
add.normalized.col <- function(df, col.name){
  # df: data.frame (usually store.level) to add normalized column to
  # col.name: the name of the column in df which should be normalized
  # the normalized column is named as 'Norm_[col.name]'
  df$temp <- df[, col.name]
  store.ply <- ddply(df, c('Store'), summarize, Mean_X=mean(temp),
                     SD_X=sd(temp))
  df <- merge(df, store.ply, by=c('Store'))
  df$Norm_X <- (df[,col.name] - df$Mean_X) / df$SD_X
  df$Mean_X <- NULL
  df$SD_X <- NULL
  df$temp <- NULL
  df <- rename(df, c('Norm_X'=paste('Norm_', col.name, sep='')))
  return(df)
}
```

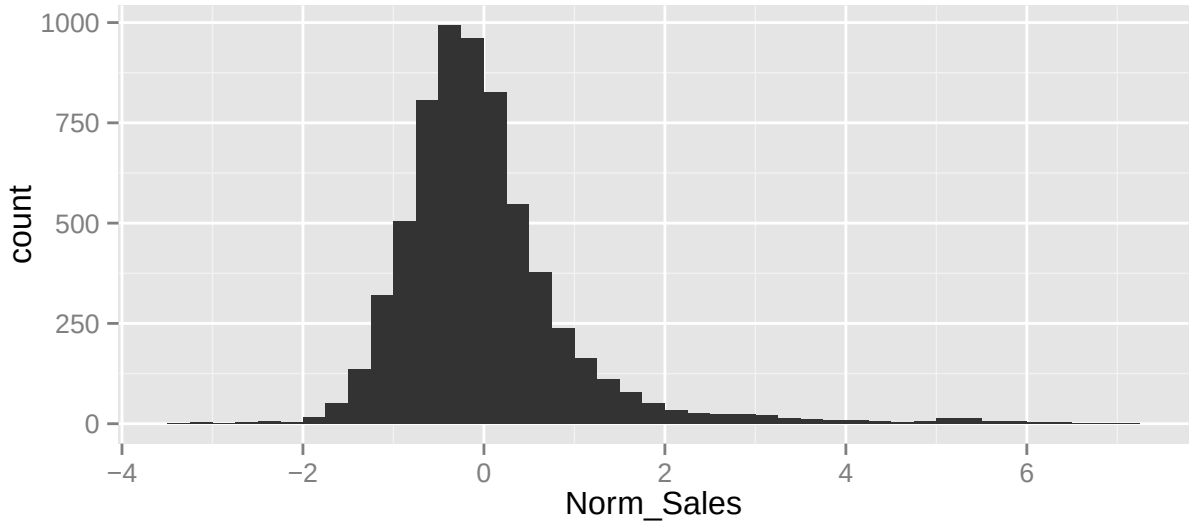
```
store.level <- add.normalized.col(store.level, 'Sales')
```

We also commonly want to get sales aggregates after grouping and/or filtering by some attribute, so we define a function for this operation.

```
get.aggr.sales <- function(df, aggr.dims, filter=NULL){
  # df: data.frame (usually store.level)
  # aggr.dims: a list of column names to aggregate on
  # filter: NULL or c(column name to filter, value to filter to)
  if(!is.null(filter)){
    sub.df <- subset(df, df[,filter[1]] == filter[2])
  } else{
    sub.df <- df
  }
  grouped <- ddply(sub.df, aggr.dims, summarize, Sales=sum(Sales),
                   Sales_Millions=sum(Sales_Millions))
  return(grouped)
}
```

Let's take a look at the normalized distribution of store-level weekly sales. If the data turns out to be normally distributed, we will apply some statistical tests for normally distributed data.

```
require(ggplot2)
ggplot(data=store.level, aes(x=Norm_Sales)) + geom_histogram(binwidth=0.25)
```



The distribution appears roughly symmetric and normal, although there is a light long tail on the positive side.

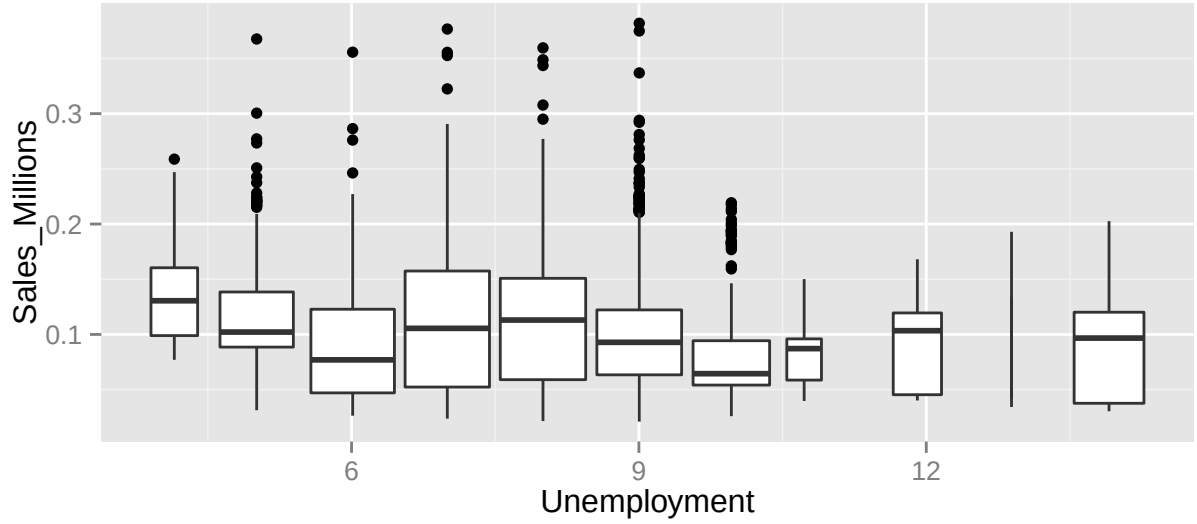
Impact of economic factors

In this section we explore how economic factors (unemployment, CPI, and fuel price) relate to Walmart's sales. It's not immediately obvious how we should expect Walmart's sales to vary with economic conditions.

While poor economic conditions hurt most retailers, Walmart is known for low prices. Could bad economic times drive more people to shop at Walmart?

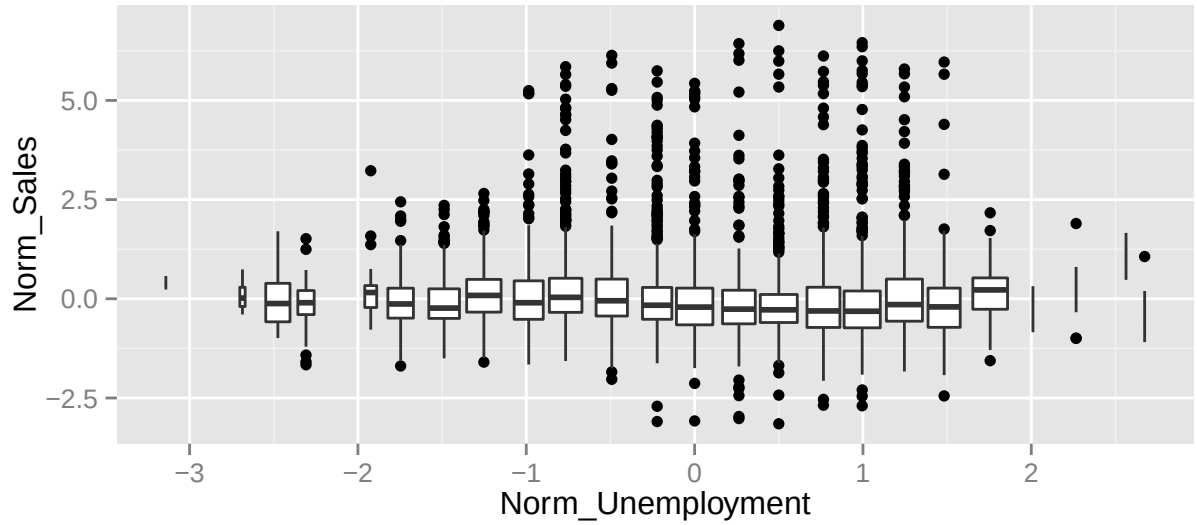
We start by creating a boxplot of sales by unemployment rate.

```
ggplot(data=store.level, aes(x=Unemployment, y=Sales_Millions,
  group=round_any(Unemployment, 1.0))) + geom_boxplot()
```



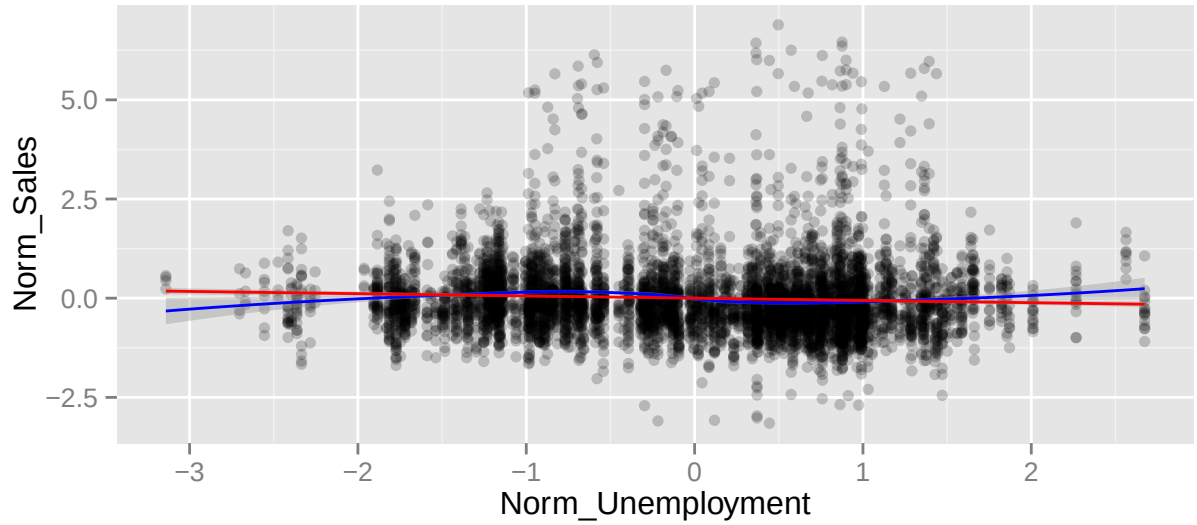
This plot don't show much of a relationship, but we haven't done any normalization, so this probably isn't a meaningful representation of the relationship. We normalize sales and unemployment rate by store and create an updated boxplot.

```
store.level <- add.normalized.col(store.level, 'Unemployment')
ggplot(data=store.level, aes(x=Norm_Unemployment, y=Norm_Sales,
  group=round_any(Norm_Unemployment, 0.25))) + geom_boxplot()
```



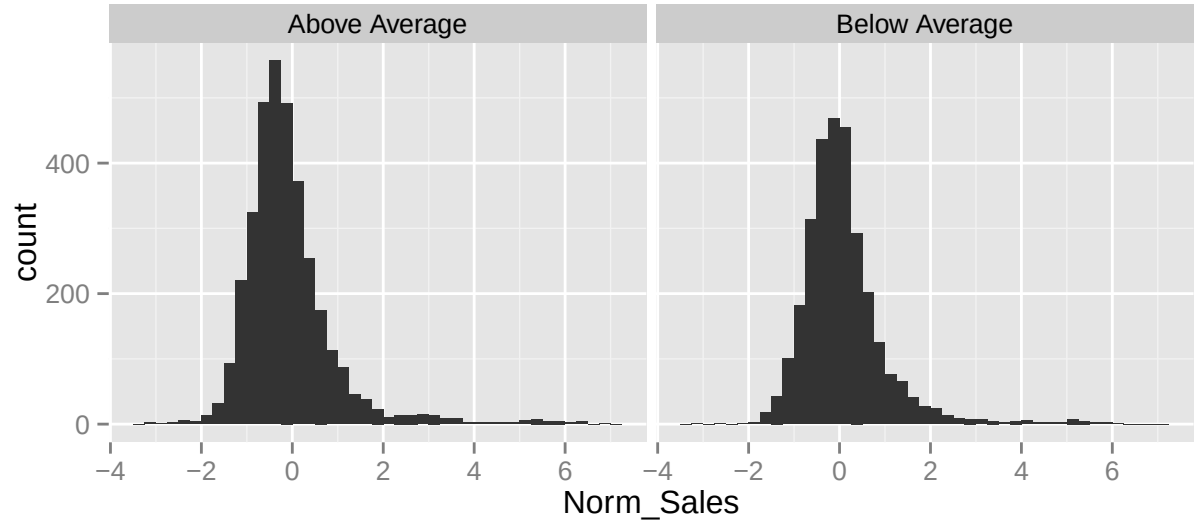
The same data in a scatter plot with LOESS (blue) and GAM (red) curves overlaid:

```
ggplot(data=store.level, aes(x=Norm_Unemployment, y=Norm_Sales)) +
  geom_point(alpha = 1/5) + geom_smooth(method='loess', color='blue') +
  geom_smooth(method='gam', color='red')
```



Within the -1 to 1 normalized unemployment range, a pattern seems to have emerged. There appears to be below-average sales when there is above-average unemployment. Let's create a pair of histograms showing number of weeks by normalized sales – one histogram for weeks with below-average unemployment and one histogram for weeks with above-average unemployment. We want to see if these distributions are noticably different. We also want to see the shape of these distributions to see if the data is roughly normally distributed so that a t-test could be reasonably applied.

```
store.level$Unemployment_Category <- ifelse(
  store.level$Norm_Unemployment < 0, 'Below Average', 'Above Average')
ggplot(data=store.level, aes(x=Norm_Sales)) + geom_histogram(binwidth=0.25) +
  facet_wrap(~Unemployment_Category)
```



The relationship between unemployment and sales isn't too clear from this plot, but the populations do appear to be roughly normal. We now run an F-test to check if the populations have equal variances (as assumed by a two-sample t-test).

```
above <- subset(store.level, Unemployment_Category == 'Above Average')
below <- subset(store.level, Unemployment_Category == 'Below Average')
var.test(above$Norm_Sales, below$Norm_Sales)
```

```
##
##  F test to compare two variances
##
## data:  above$Norm_Sales and below$Norm_Sales
## F = 1.2265, num df = 3477, denom df = 2956, p-value = 9.068e-09
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
##  1.144162 1.314409
## sample estimates:
## ratio of variances
##           1.226456
```

We reject the null hypothesis that the populations have equal variance, so we proceed with Welsh t-test for populations with unequal variances. The null hypothesis of this t-test is that the mean normalized sales during weeks with above-average unemployment rates is equal to the mean normalized sales during weeks with below-average umemployment rates.

```
t.test(above$Norm_Sales, below$Norm_Sales, var.equal=FALSE)
```

```
##
##  Welch Two Sample t-test
##
## data:  above$Norm_Sales and below$Norm_Sales
## t = -6.831, df = 6409.719, p-value = 9.209e-12
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.2166014 -0.1200037
## sample estimates:
##   mean of x   mean of y
## -0.07733810  0.09096447
```

We can confidently reject the null hypothesis. Now, we drop our assumption of normality, and perform a non-parametric test to see determine if sales for weeks with above-average unemployment are drawn from a different distribution than the sales for weeks with below-average unemployment. We apply the Mann-Whitney-Wilcoxon procedure below.

```
wilcox.test(Norm_Sales ~ Unemployment_Category, data=store.level)
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  Norm_Sales by Unemployment_Category
## W = 4330099, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0
```

From all the results above, we can conclude that Walmart's sales are not independent from the unemployment rate; the two variables are negatively correlated.

We repeat the same procedures for CPI and fuel price. We hide the code and output to save space, but we summarize the p-values from the tests in the table below:

Economic Factor	F-test 95% CI	t-test 95% CI	Mann-Whitney-Wilcoxon p-value
Unemployment	(1.14, 1.31)	(-0.22, -0.12)	2.2e-16
CPI	(0.83, 0.95)	(0.11, 0.21)	2.2e-16
Fuel price	(0.34, 0.39)	(-0.15, -0.04)	3.9e-4

All three variables tell a similar story. When external economic factors are unfavorable (high unemployment, low CPI, and high fuel prices), Walmart sales are not as strong as they are during times when external economic factors are favorable.

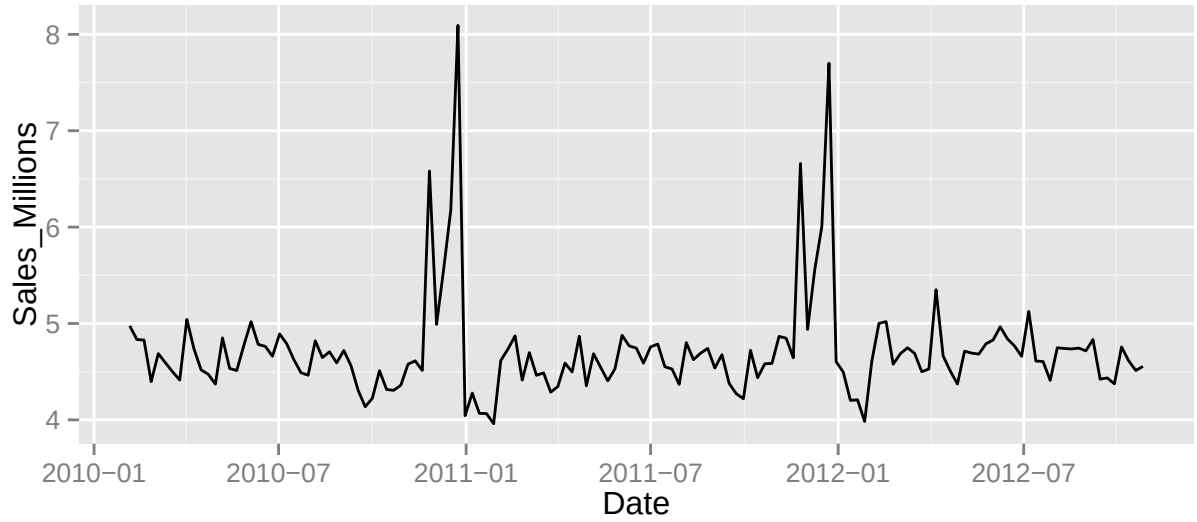
Sales predictions

With the goal of creating a model to predict store-level normalized sales for a given week and store, we build a set of features and then compare the performance of a few different models.

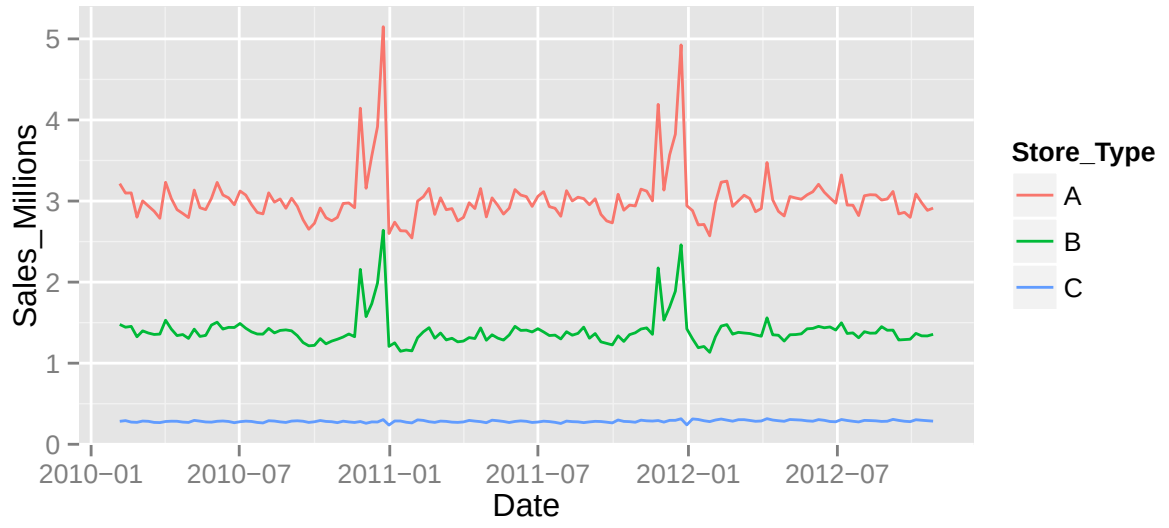
Features

In order to make good predictions about sales, we need to consider much more than just the economic factors we've considered so far. For example, the next couple plots show that dates will also be important in our prediction; we observe seasonality and trending patterns in the time series.

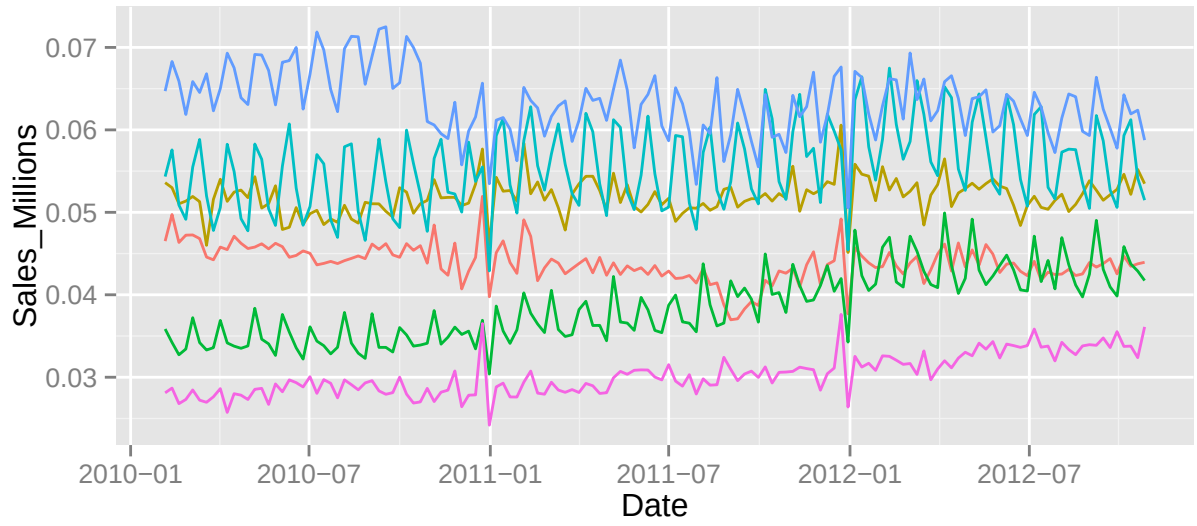
```
# plot sales by week
sales.by.week <- get.aggr.sales(store.level, c('Date'))
ggplot(data=sales.by.week, aes(x=Date, y=Sales_Millions)) + geom_line()
```



```r
# plot sales by week by store type
sales.by.type <- get.aggr.sales(store.level, c('Store_Type','Date'))
ggplot(data=sales.by.type, aes(x=Date, y=Sales_Millions, group=Store_Type,
                               color=Store_Type)) + geom_line()
```




```r
# by store for type C
sales.by.store <- get.aggr.sales(store.level, c('Store', 'Date'), c('Store_Type', 'C'))
ggplot(data=sales.by.store, aes(x=Date, y=Sales_Millions, group=Store, color=Store)) +
  geom_line() + guides(color=FALSE)
```

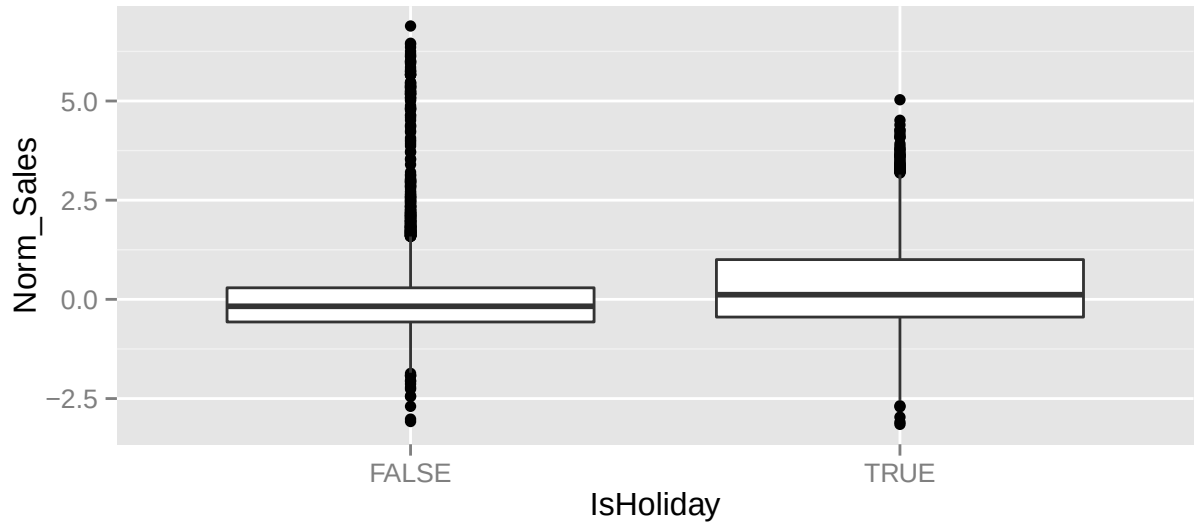


To capture some of the seasonality and trending nature of the data, we add a month-of-year factor feature and we generate "lag" features that represent what the normalized sales were 1, 2, 3, 4, 51, and 52 weeks ago. In an attempt to capture trending, we also create a feature that represents the difference in normalized sales between last week and 53 weeks ago. By adding features representing past values of the same time series, we are creating an auto-regressive model. Because we are using auto-regressive features in addition to other predictors, this can be classified as an ARX model.

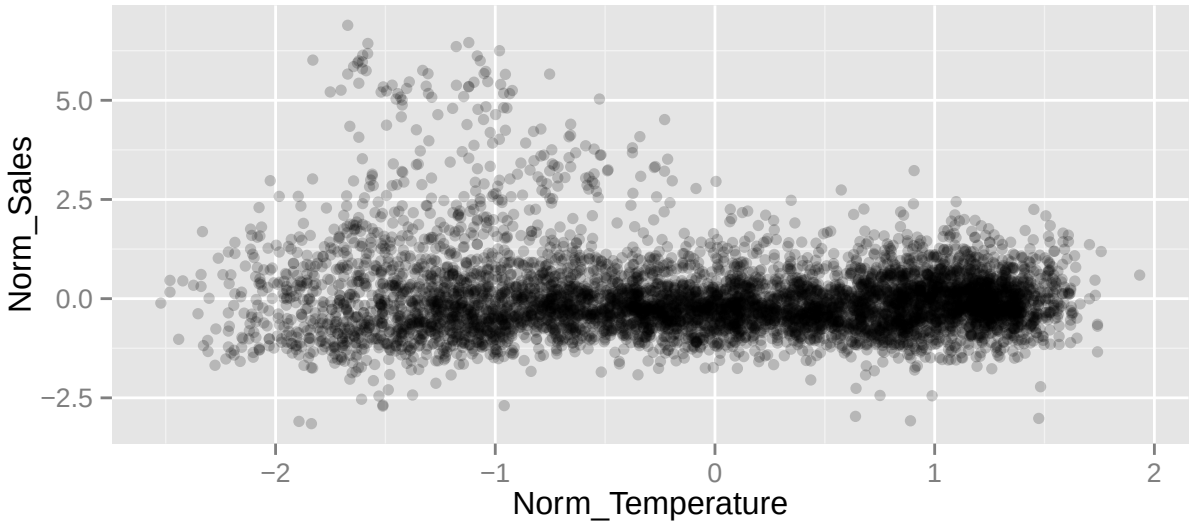
```
require(lubridate)
store.level$Month <- as.factor(month(store.level$Date))

add.lag.column <- function(df, weeks){
  # df: data.frame (usually store.level) to which lag column will be added
  # weeks: an integer representing the number of weeks ago that a lag column
  #        will be created for
  # The added lag column is named 'Lag_[weeks]_Sales'
  shifted <- df[, c('Store', 'Date', 'Norm_Sales')]
  shifted$Date <- shifted$Date + (weeks * 7)
  shifted <- rename(shifted, c('Norm_Sales'=paste('Lag_', weeks, '_Sales', sep='')))
  df <- merge(df, shifted, c('Store', 'Date'), all.x = TRUE, all.y = FALSE)
  return(df)
}

store.level <- add.lag.column(store.level, 1)
store.level <- add.lag.column(store.level, 2)
store.level <- add.lag.column(store.level, 3)
store.level <- add.lag.column(store.level, 4)
store.level <- add.lag.column(store.level, 51)
store.level <- add.lag.column(store.level, 52)
store.level <- add.lag.column(store.level, 53)
store.level$Lag_1_53_Diff <- store.level$Lag_1_Sales - store.level$Lag_53_Sales
```

Some plots of other predictor variables we'll include in our models:

```
ggplot(data=store.level, aes(x=IsHoliday, y=Norm_Sales)) + geom_boxplot()
```




```
store.level <- add.normalized.col(store.level, 'Temperature')
ggplot(data=store.level, aes(x=Norm_Temperature, y=Norm_Sales)) + geom_point(alpha=1/5)
```



Setup

We limit the testing/training sets to data after February 11, 2011, because we cannot calculate all of our lag features for weeks before then. We divide the remain data points randomly into an 80% training set and a 20% test set.

```
test.train <- subset(store.level, Date >= '2011-02-11')
set.seed(123)
row.count=nrow(test.train)
train.ind <- sample(1:row.count, size=row.count*0.8)
train.set <- test.train[train.ind, ]
test.set <- test.train[-train.ind, ]
paste(nrow(train.set), 'training records;', nrow(test.set), 'testing records')
```

```
## [1] "3240 training records; 810 testing records"
```

To allow us to make good comparisons between models, we use a consistent set of predictors for all the models we fit in the following section. We include:

- Normalized external economic factors (unemployment rate, CPI, and fuel price)
- A holiday indicator variable with a Store_Type interaction (because we saw that store type C was less impacted by the holiday season than were the other store types)
- Normalized temperature
- Month-of-year and lag features

```
sales.formula <- Norm_Sales ~ Norm_Unemployment + Norm_CPI + Norm_Fuel_Price +
  IsHoliday*Store_Type + Norm_Temperature + Lag_1_Sales + Lag_2_Sales + Lag_3_Sales +
  Lag_4_Sales + Lag_51_Sales + Lag_52_Sales + Lag_1_53_Diff + Month
```

We define a convenience function for evaluating the model on the training and test sets:

```r
get.mses <- function(model){
  test.mse <- mean((predict(model, test.set) - test.set$Norm_Sales)^2)
  train.mse <- mean((predict(model, train.set) - train.set$Norm_Sales)^2)
  paste('Train MSE:', train.mse,'; Test MSE:', test.mse)
}
```

Model comparison

To start, we try a linear regression model. With no variable selection or shrinkage, it wouldn't be surprising to see the model overfit the data.

```r
linear.model <- lm(sales.formula, data=train.set)
summary(linear.model)
```

```
##
## Call:
## lm(formula = sales.formula, data = train.set)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -3.09847 -0.23758 -0.01113  0.22817  2.30696
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                -0.496979   0.055113  -9.018  < 2e-16 ***
## Norm_Unemployment          -0.027831   0.011355  -2.451 0.014296 *
## Norm_CPI                    0.039837   0.016262   2.450 0.014352 *
## Norm_Fuel_Price             0.002536   0.024391   0.104 0.917206
## IsHolidayTRUE               0.308418   0.047848   6.446 1.32e-10 ***
## Store_TypeB                -0.006876   0.017187  -0.400 0.689153
## Store_TypeC                 0.081380   0.024458   3.327 0.000887 ***
## Norm_Temperature            0.031247   0.024125   1.295 0.195337
## Lag_1_Sales                 0.066701   0.012006   5.556 2.99e-08 ***
## Lag_2_Sales                 0.042111   0.010439   4.034 5.61e-05 ***
## Lag_3_Sales                 0.051329   0.011213   4.578 4.88e-06 ***
## Lag_4_Sales                 0.109297   0.011346   9.633  < 2e-16 ***
## Lag_51_Sales               -0.097507   0.010285  -9.481  < 2e-16 ***
## Lag_52_Sales                0.718462   0.010520  68.296  < 2e-16 ***
## Lag_1_53_Diff               0.395572   0.014301  27.660  < 2e-16 ***
## Month2                      0.633779   0.057863  10.953  < 2e-16 ***
## Month3                      0.430245   0.054500   7.894 3.97e-15 ***
## Month4                      0.458819   0.063369   7.240 5.57e-13 ***
## Month5                      0.466137   0.068735   6.782 1.41e-11 ***
## Month6                      0.451954   0.074134   6.096 1.21e-09 ***
## Month7                      0.376220   0.077686   4.843 1.34e-06 ***
## Month8                      0.455013   0.078680   5.783 8.04e-09 ***
## Month9                      0.277200   0.071454   3.879 0.000107 ***
## Month10                     0.522266   0.062635   8.338  < 2e-16 ***
## Month11                     0.805508   0.062933  12.799  < 2e-16 ***
## Month12                     0.741737   0.065486  11.327  < 2e-16 ***
## IsHolidayTRUE:Store_TypeB   0.066863   0.066245   1.009 0.312891
## IsHolidayTRUE:Store_TypeC -0.443131   0.093618  -4.733 2.30e-06 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4347 on 3212 degrees of freedom
## Multiple R-squared:  0.774,  Adjusted R-squared:  0.7721
## F-statistic: 407.4 on 27 and 3212 DF,  p-value: < 2.2e-16
```

In the summary, we see that:

- Norm_CPI appears very significant while the other economic indicators do not. However, we suspect there is some collinearity between these three predictors, so we do not want to read too much into their relative significances. It seems that collinearity might also be leading to the positive Norm_Fuel_Price coefficient; based on our earlier analysis, we expected this to be negative.
- The IsHoliday indicator variable has a very significant positive coefficient (i.e. holidays lead to more sales), and the store type C interaction has a significant negative coefficient (which aligns with our earlier observation that store type C did not exhibit the same seasonal behavior as the other store types in our time series plots).
- Of all the lag variables, the sales from 52 weeks ago seem most important for prediction. The Lag_1_53_Diff variable, which we are using as a proxy for long-term trend, also stands out as being helpful for prediction.

How does the MSE on the training set compare to the MSE on the test set?

```
get.mses(linear.model)
```

```
## [1] "Train MSE: 0.187299169969781 ; Test MSE: 0.191814323096556"
```

The test error is only slightly higher than the training error. This seems to indicate that we do not have a notable overfitting problem, even though we have not used any subset selection or shrinkage. Below, we apply subset selection using forward and backward stepwise regression to see if/how the results change.

```
forward.model <- step(object=lm(Norm_Sales ~ 1, data=train.set), scope=sales.formula,
                      direction='forward', trace=FALSE)
summary(forward.model)
```

```
##
## Call:
## lm(formula = Norm_Sales ~ Lag_52_Sales + Lag_1_53_Diff + Lag_1_Sales +
##     Lag_51_Sales + Month + Lag_4_Sales + IsHoliday + Lag_3_Sales +
##     Norm_CPI + Lag_2_Sales + Norm_Unemployment + Store_Type +
##     IsHoliday:Store_Type, data = train.set)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -3.09531 -0.23690 -0.01281  0.22820  2.29987
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)               -0.536710   0.045997 -11.668  < 2e-16 ***
## Lag_52_Sales               0.718302   0.010517  68.300  < 2e-16 ***
## Lag_1_53_Diff              0.395776   0.014300  27.677  < 2e-16 ***
```

```
## Lag_1_Sales                 0.067881   0.011952   5.680 1.47e-08 ***
## Lag_51_Sales               -0.096243   0.010217  -9.420  < 2e-16 ***
## Month2                      0.636472   0.057796  11.012  < 2e-16 ***
## Month3                      0.452783   0.049354   9.174  < 2e-16 ***
## Month4                      0.496281   0.051747   9.591  < 2e-16 ***
## Month5                      0.518203   0.051578  10.047  < 2e-16 ***
## Month6                      0.521679   0.051424  10.145  < 2e-16 ***
## Month7                      0.454875   0.049856   9.124  < 2e-16 ***
## Month8                      0.532956   0.051746  10.299  < 2e-16 ***
## Month9                      0.342563   0.050216   6.822 1.07e-11 ***
## Month10                     0.564391   0.052930  10.663  < 2e-16 ***
## Month11                     0.823778   0.061255  13.448  < 2e-16 ***
## Month12                     0.736368   0.065257  11.284  < 2e-16 ***
## Lag_4_Sales                 0.109126   0.011336   9.627  < 2e-16 ***
## IsHolidayTRUE               0.304426   0.047718   6.380 2.03e-10 ***
## Lag_3_Sales                 0.051318   0.011196   4.584 4.74e-06 ***
## Norm_CPI                    0.042416   0.015734   2.696 0.007060 **
## Lag_2_Sales                 0.042300   0.010437   4.053 5.18e-05 ***
## Norm_Unemployment          -0.026730   0.011320  -2.361 0.018273 *
## Store_TypeB                -0.006989   0.017186  -0.407 0.684290
## Store_TypeC                 0.081148   0.024455   3.318 0.000916 ***
## IsHolidayTRUE:Store_TypeB   0.068295   0.066233   1.031 0.302555
## IsHolidayTRUE:Store_TypeC -0.442539   0.093613  -4.727 2.37e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4346 on 3214 degrees of freedom
## Multiple R-squared:  0.7739, Adjusted R-squared:  0.7721
## F-statistic:   440 on 25 and 3214 DF,  p-value: < 2.2e-16
```

```
get.mses(forward.model)
```

```
## [1] "Train MSE: 0.187400944031929 ; Test MSE: 0.191747344190034"
```

```
backward.model <- step(object=lm(sales.formula, data=train.set),
                       direction='backward', trace=FALSE)
summary(backward.model)
```

```
##
## Call:
## lm(formula = Norm_Sales ~ Norm_Unemployment + Norm_CPI + IsHoliday +
##     Store_Type + Lag_1_Sales + Lag_2_Sales + Lag_3_Sales + Lag_4_Sales +
##     Lag_51_Sales + Lag_52_Sales + Lag_1_53_Diff + Month + IsHoliday:Store_Type,
##     data = train.set)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -3.09531 -0.23690 -0.01281  0.22820  2.29987
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)               -0.536710   0.045997 -11.668  < 2e-16 ***
## Norm_Unemployment         -0.026730   0.011320  -2.361 0.018273 *
```

```
## Norm_CPI                   0.042416   0.015734   2.696 0.007060 **
## IsHolidayTRUE              0.304426   0.047718   6.380 2.03e-10 ***
## Store_TypeB               -0.006989   0.017186  -0.407 0.684290
## Store_TypeC                0.081148   0.024455   3.318 0.000916 ***
## Lag_1_Sales                0.067881   0.011952   5.680 1.47e-08 ***
## Lag_2_Sales                0.042300   0.010437   4.053 5.18e-05 ***
## Lag_3_Sales                0.051318   0.011196   4.584 4.74e-06 ***
## Lag_4_Sales                0.109126   0.011336   9.627  < 2e-16 ***
## Lag_51_Sales              -0.096243   0.010217  -9.420  < 2e-16 ***
## Lag_52_Sales               0.718302   0.010517  68.300  < 2e-16 ***
## Lag_1_53_Diff              0.395776   0.014300  27.677  < 2e-16 ***
## Month2                     0.636472   0.057796  11.012  < 2e-16 ***
## Month3                     0.452783   0.049354   9.174  < 2e-16 ***
## Month4                     0.496281   0.051747   9.591  < 2e-16 ***
## Month5                     0.518203   0.051578  10.047  < 2e-16 ***
## Month6                     0.521679   0.051424  10.145  < 2e-16 ***
## Month7                     0.454875   0.049856   9.124  < 2e-16 ***
## Month8                     0.532956   0.051746  10.299  < 2e-16 ***
## Month9                     0.342563   0.050216   6.822 1.07e-11 ***
## Month10                    0.564391   0.052930  10.663  < 2e-16 ***
## Month11                    0.823778   0.061255  13.448  < 2e-16 ***
## Month12                    0.736368   0.065257  11.284  < 2e-16 ***
## IsHolidayTRUE:Store_TypeB  0.068295   0.066233   1.031 0.302555
## IsHolidayTRUE:Store_TypeC -0.442539   0.093613  -4.727 2.37e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4346 on 3214 degrees of freedom
## Multiple R-squared:  0.7739, Adjusted R-squared:  0.7721
## F-statistic:   440 on 25 and 3214 DF,  p-value: < 2.2e-16
```

```
get.mses(backward.model)
```

```
## [1] "Train MSE: 0.187400944031929 ; Test MSE: 0.191747344190034"
```

Using AIC to determine the optimal stopping point, the forward stepwise regression and backward stepwise regression yield exactly the same result. They include all predictors in the model except for Norm_Temperature and Norm_Fuel_Price. The test MSE increased, but we successfully decreased the training MSE. We interpret this to mean that the increase in (squared) bias we incurred by removing predictors was less than the decrease in variance that we gained.

Instead of subset selection, we now try shrinkage. We use lasso regression. 10-fold cross validation is used to find the best value of λ. Then, we use this optimal λ to made predictions on our test set.

```
require(glmnet)
train.matrix <- model.matrix(object=sales.formula, data=train.set)
train.matrix <- train.matrix[,2:ncol(train.matrix)]  # remove the (intercept) column
best.lambda <- cv.glmnet(x=train.matrix, y=train.set$Norm_Sales, nfolds=10, alpha=1)$lambda.min
best.lambda
```

```
## [1] 0.0002311143
```

```
lasso.betas <- glmnet(x=train.matrix, y=train.set$Norm_Sales, alpha=1, lambda=best.lambda)$beta
lasso.betas
```

```
## 27 x 1 sparse Matrix of class "dgCMatrix"
##                                       s0
## Norm_Unemployment         -0.027796833
## Norm_CPI                   0.038503047
## Norm_Fuel_Price            0.007306559
## IsHolidayTRUE              0.308918005
## Store_TypeB               -0.006463830
## Store_TypeC                0.081357549
## Norm_Temperature           0.043659704
## Lag_1_Sales                0.068284458
## Lag_2_Sales                0.041065822
## Lag_3_Sales                0.048989531
## Lag_4_Sales                0.105655563
## Lag_51_Sales              -0.097157762
## Lag_52_Sales               0.720499120
## Lag_1_53_Diff              0.396123164
## Month2                     0.590355556
## Month3                     0.382464664
## Month4                     0.401087566
## Month5                     0.403128565
## Month6                     0.383499298
## Month7                     0.306782147
## Month8                     0.383652619
## Month9                     0.212252962
## Month10                    0.465015217
## Month11                    0.757172681
## Month12                    0.704622814
## IsHolidayTRUE:Store_TypeB  0.065739039
## IsHolidayTRUE:Store_TypeC -0.437125498
```

```
test.matrix <- model.matrix(object=sales.formula, data=test.set)
test.matrix <- test.matrix[,2:ncol(test.matrix)]  # remove the (intercept) column
lasso.test.preds <-  test.matrix %*% lasso.betas
lasso.train.preds <-  train.matrix %*% lasso.betas
lasso.test.mse <- mean((lasso.test.preds - test.set$Norm_Sales)^2)
lasso.train.mse <- mean((lasso.train.preds - train.set$Norm_Sales)^2)
paste('Train MSE:', lasso.train.mse,'; Test MSE:', lasso.test.mse)
```

```
## [1] "Train MSE: 0.385075125287504 ; Test MSE: 0.369776403599856"
```

In this case, the optimal λ was so small that no coefficients were shrunk all the way to zero. The coefficients are very similar to what we observed with ordinary linear regression. λ is small because n is much larger than p. Therefore, we don't have too much overfitting and so the penalty is small. Unlike subset selection - which marginally improved our test MSE - LASSO regression increased the squared bias more than it decreased the variance, thus leading to a much higher MSE than in our original linear regression model.

Next, we try KNN regression. We start by using leave-one-out cross validation on the training set to find the best value of k.

```
require(FNN)
for(k in 1:15){
  knn.cv.model <- knn.reg(train.matrix, k=k, y=train.set$Norm_Sales)
  print(paste('With k=', k, ', MSE is ', mean(knn.cv.model$residuals^2), '.', sep=''))
}
```

```
## [1] "With k=1, MSE is 0.243960083919923."
## [1] "With k=2, MSE is 0.186189133093814."
## [1] "With k=3, MSE is 0.174349435575092."
## [1] "With k=4, MSE is 0.166281758630541."
## [1] "With k=5, MSE is 0.165913187933766."
## [1] "With k=6, MSE is 0.166500046965384."
## [1] "With k=7, MSE is 0.167032300340984."
## [1] "With k=8, MSE is 0.167737842922471."
## [1] "With k=9, MSE is 0.170158867931673."
## [1] "With k=10, MSE is 0.171287212938242."
## [1] "With k=11, MSE is 0.172889459821327."
## [1] "With k=12, MSE is 0.174811004843356."
## [1] "With k=13, MSE is 0.176383770775372."
## [1] "With k=14, MSE is 0.178833227243039."
## [1] "With k=15, MSE is 0.180822622582898."
```

Because k=5 minimized the MSE in cross validation, we use this to make predictions on our test set.

```
knn.test.preds <- knn.reg(train.matrix, k=5, y=train.set$Norm_Sales, test=test.matrix)$pred
knn.train.preds <- knn.reg(train.matrix, k=5, y=train.set$Norm_Sales, test=train.matrix)$pred
knn.test.mse <- mean((knn.test.preds - test.set$Norm_Sales)^2)
knn.train.mse <- mean((knn.train.preds - train.set$Norm_Sales)^2)
paste('Train MSE:', knn.train.mse,'; Test MSE:', knn.test.mse)
```

```
## [1] "Train MSE: 0.106420325523546 ; Test MSE: 0.16080050096667"
```

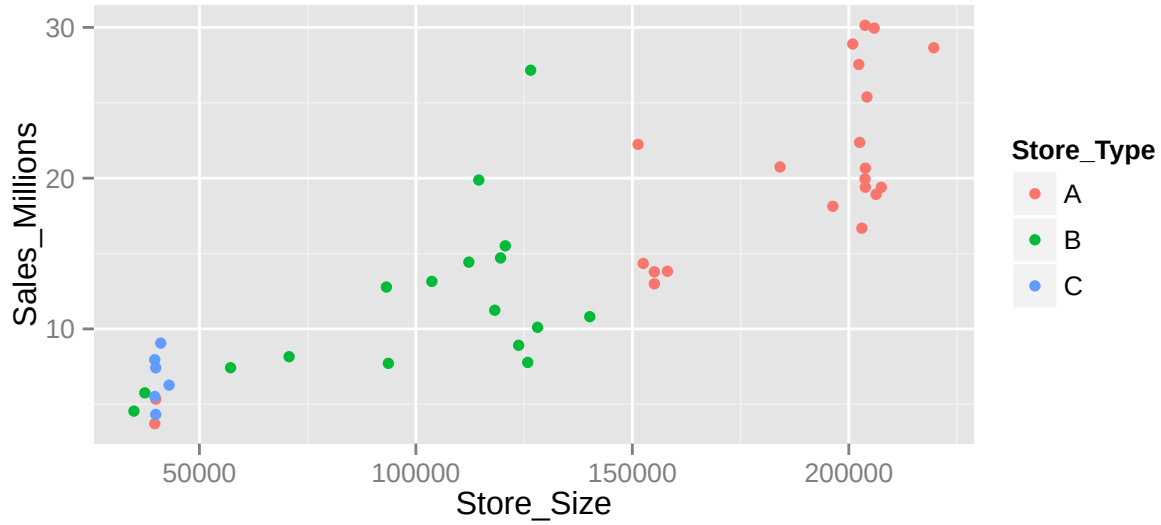
For predicting Walmart's weekly store sales, KNN regression outperforms the linear models. KNN is a nonparametric method which uses only local information to make its prediction. In contrast, linear models fit a model globally and are less flexible. Because the more flexible model gives better results on the test set, we believe there is notable bias in the linear model, and more predictors would need to be collected or engineered for a linear model to yield less error.

While the KNN model does give lower error, it doesn't provide nearly the same interpretability as the linear model. In practice, if we were working for Walmart, our interest would most likely be to understand the factors impacting sales so that Walmart could make decisions and take actions that would influence future sales. From that perspective, even if the linear model yields higher error on the test set than KNN, it is still more useful.

Appendix

The following observations did not directly fit into the analysis presented in the body of the report, but we thought it was interesting and wanted to include it.

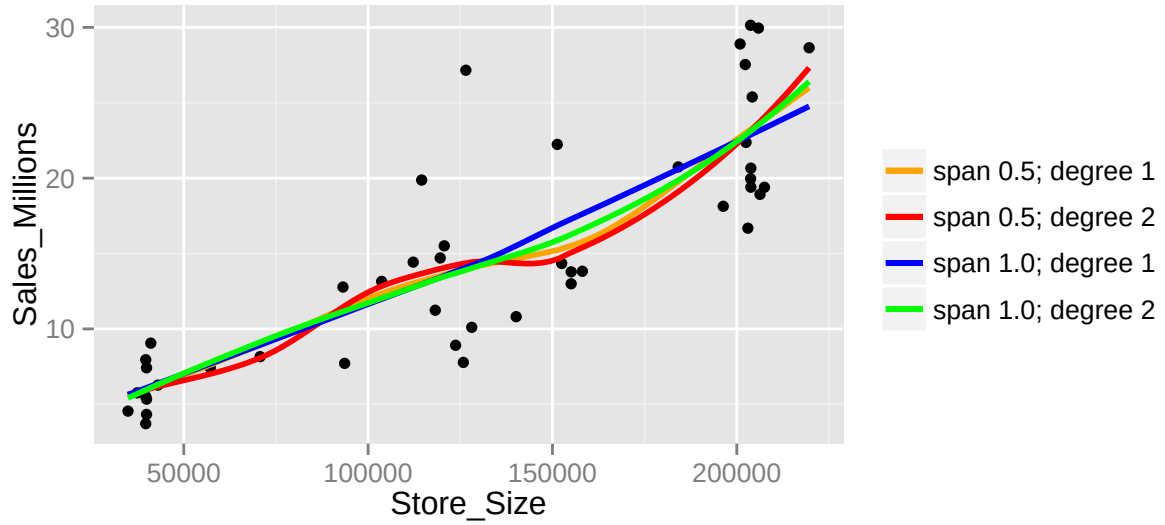
```
sales.by.size <- get.aggr.sales(store.level, c('Store', 'Store_Type', 'Store_Size'))
ggplot(data=sales.by.size, aes(x=Store_Size, y=Sales_Millions, color=Store_Type)) +
  geom_point()
```



There is what appears to be a roughly linear relationship between a store's sales and its size. We also notice in the plot above that store type is closely related to both store sales and store size.

To get a sense of how local regression differs based on parameter values, we applied loess smoothing to this scatter plot.

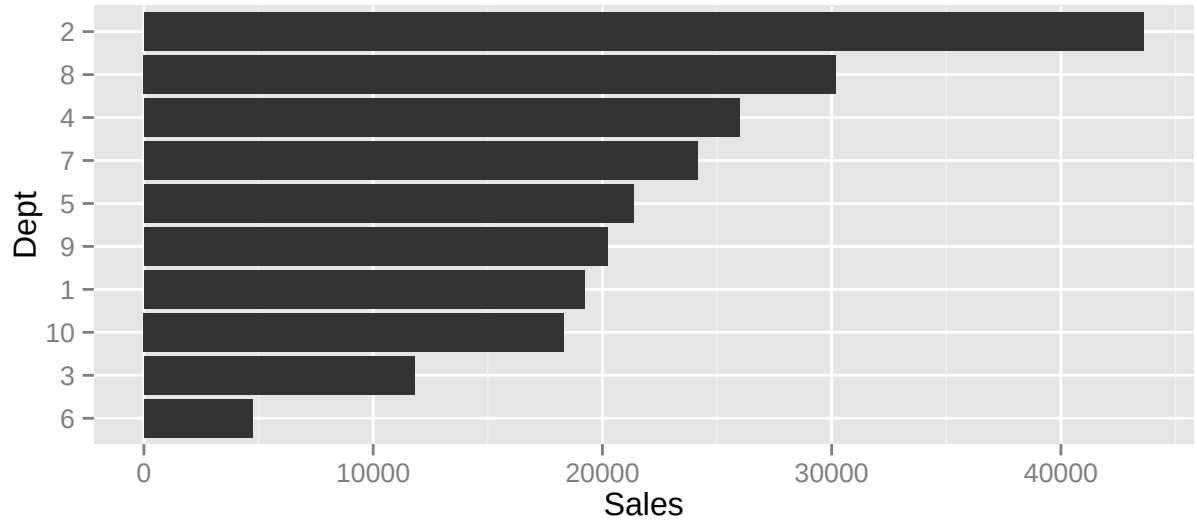
```
ggplot(data=sales.by.size, aes(x=Store_Size, y=Sales_Millions)) +
  geom_point() +
  geom_smooth(method='loess', size=1, span=0.5, degree=1,
              aes(color='span 0.5; degree 1'), se=F) +
  geom_smooth(method='loess', size=1, span=0.5, degree=2,
              aes(color='span 0.5; degree 2'), se=F) +
  geom_smooth(method='loess', size=1, span=1, degree=1,
              aes(color='span 1.0; degree 1'), se=F) +
  geom_smooth(method='loess', size=1, span=1, degree=2,
              aes(color='span 1.0; degree 2'), se=F) +
  scale_color_manual('', values=c('span 0.5; degree 1'='orange',
                                  'span 0.5; degree 2'='red',
                                  'span 1.0; degree 1'='blue',
                                  'span 1.0; degree 2'='green'))
```



As we expect, the local regressions with larger spans (i.e. less local), are less wiggly. And, the local regressions of degree two are more flexible than the regressions of degree 1. The regression with span of 1 and degree 1 is nearly a linear regression.

Below we plot the mean weekly sales for the top 10 departments

```
sales.by.dept <- aggregate(Sales ~ Dept, dept.level, mean)
sales.by.dept$Dept <- reorder(sales.by.dept$Dept, sales.by.dept$Sales)
sales.by.dept <- sales.by.dept[1:10,]
ggplot(data=sales.by.dept, aes(x=Dept, y=Sales)) + geom_bar(stat='identity') + coord_flip()
```



Then we build a decision tree based on a combination of Store, Department and Week number as shown below

```
add.normalized.dept.col <- function(df, col.name){
  # df: data.frame (usually store.level) to add normalized column to
  # col.name: the name of the column in df which should be normalized
```

```
  # the normalized column is named as 'Norm_[col.name]'
  df$temp <- df[, col.name]
  store.ply <- ddply(df, c('Store', 'Dept'), summarize, Mean_X=mean(temp),
                     SD_X=sd(temp))
  df <- merge(df, store.ply, by=c('Store', 'Dept'))
  df$Norm_X <- (df[,col.name] - df$Mean_X) / df$SD_X
  df$Mean_X <- NULL
  df$SD_X <- NULL
  df$temp <- NULL
  df <- rename(df, c('Norm_X'=paste('Norm_', col.name, sep='')))
  return(df)
}

dept.level <- add.normalized.dept.col(dept.level, 'Sales')

dept.level$WeekNum <- as.factor(as.integer(format(dept.level$Date, "%W")) + 1)
# dept.level$WeekNum <- as.factor(store.level$WeekNum)
```

```
row.count=nrow(dept.level)
train.ind <- sample(1:row.count, size=row.count*0.8)
dept.train.set <- dept.level[train.ind, ]
dept.test.set <- dept.level[-train.ind, ]
paste(nrow(dept.train.set), 'training records;', nrow(dept.test.set), 'testing records')
```

```
## [1] "337256 training records; 84314 testing records"
```

```
require(rpart)
dt <- rpart(Norm_Sales ~ Store + Dept + WeekNum, data=dept.train.set)
pred.dt <- predict(dt,  newdata=dept.test.set)
```

We next cluster the weekly store sales by department using an unsupervised kmeans approach.

```
# sales.dp=aggregate(Sales ~ Dept + Date, data=dept.level, mean)
# dp.mat = with(sales.dp, tapply(Weekly_Sales, list(Dept, Date), mean))
# # Replace NA with row means
# k <- which(is.na(dp.mat), arr.ind=TRUE)
# dp.mat[k] <- rowMeans(dp.mat, na.rm=TRUE)[k[,1]]
# km1 <- kmeans(dp.mat,7)
# depts <- data.frame(Dept.C=km1$cluster)
# depts$Dept <- row.names(depts)
# depts$Dept.C <- as.factor(depts$Dept.C)
# rm(km1, dp.mat, k, sales.dp)
# depts <- depts[c("Dept","Dept.C")]
# store_dp_clusters <- merge(x=store.level, y=depts, all.x=TRUE)
```