IST 597: Midterm

Justin Silverman

Problem 1

```
library(tidyverse)
library(lubridate)

dat <- read_csv("data_train_midterm_problem1.csv")
```

Part A

Are there any missing values?

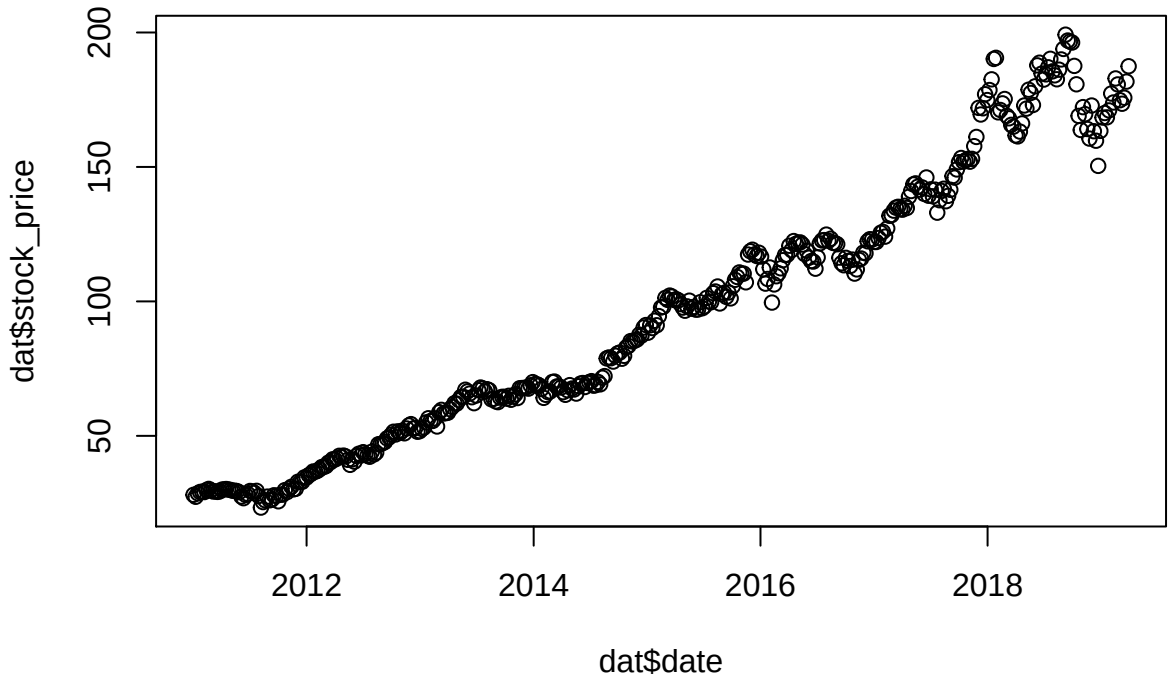
```
# test that the difference between successive observations is always 7 days
foo <- dat$date-lag(dat$date)
foo <- foo[!is.na(foo)] # remove first NA that is just caused by lagging the first point.
all(foo==7)
```

```
## [1] TRUE
```

So we are all good on that front, the data is all equally spaced and this also tells us there is no duplicate observations.

Lets just plot the data.

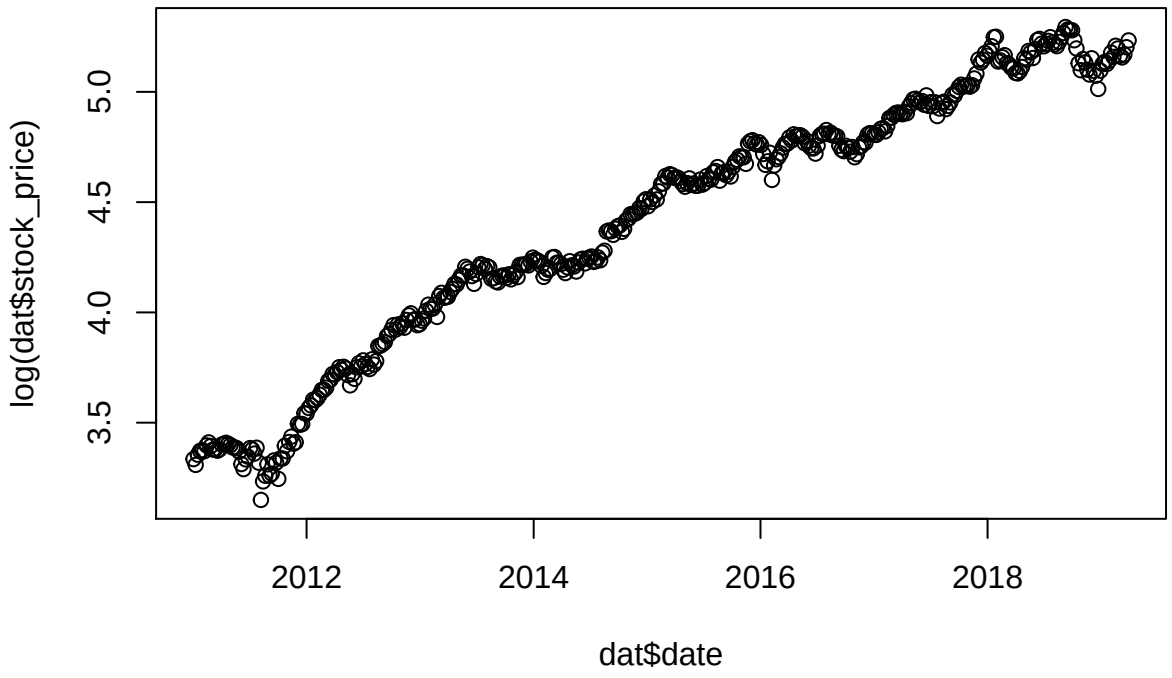
```
plot(dat$date, dat$stock_price)
```



I notice a few things here. One is that the variance seems to increase later in the time-series. This could either be due to heteroscedasticity (variance depends on date) or due to a mean-variance relationship (variance depends on stock_price). Either way not sure what to do about it right now (also not sure if I will do anything about it based on what I have "learned so far in this course").

What about plotting on a log-scale?

```
plot(dat$date, log(dat$stock_price))
```



That does look a little better, the variance does seem to be much more stable across the time-series now. This was probably some weird mean-variance relationship. As a result I am going to perform

all my modeling on the log scale time-series.

Before I go any farther I am going to set aside the last bit of data for a final validation later on. (Note even some of the following data exploration can be considered modeling and I don't want to overfit). Also, to make things easy down the road, I am going to translate dates to days from the first observation (as an integer) – I am not sure how scikit-learn handles dates but my experience tells me not to chance it.

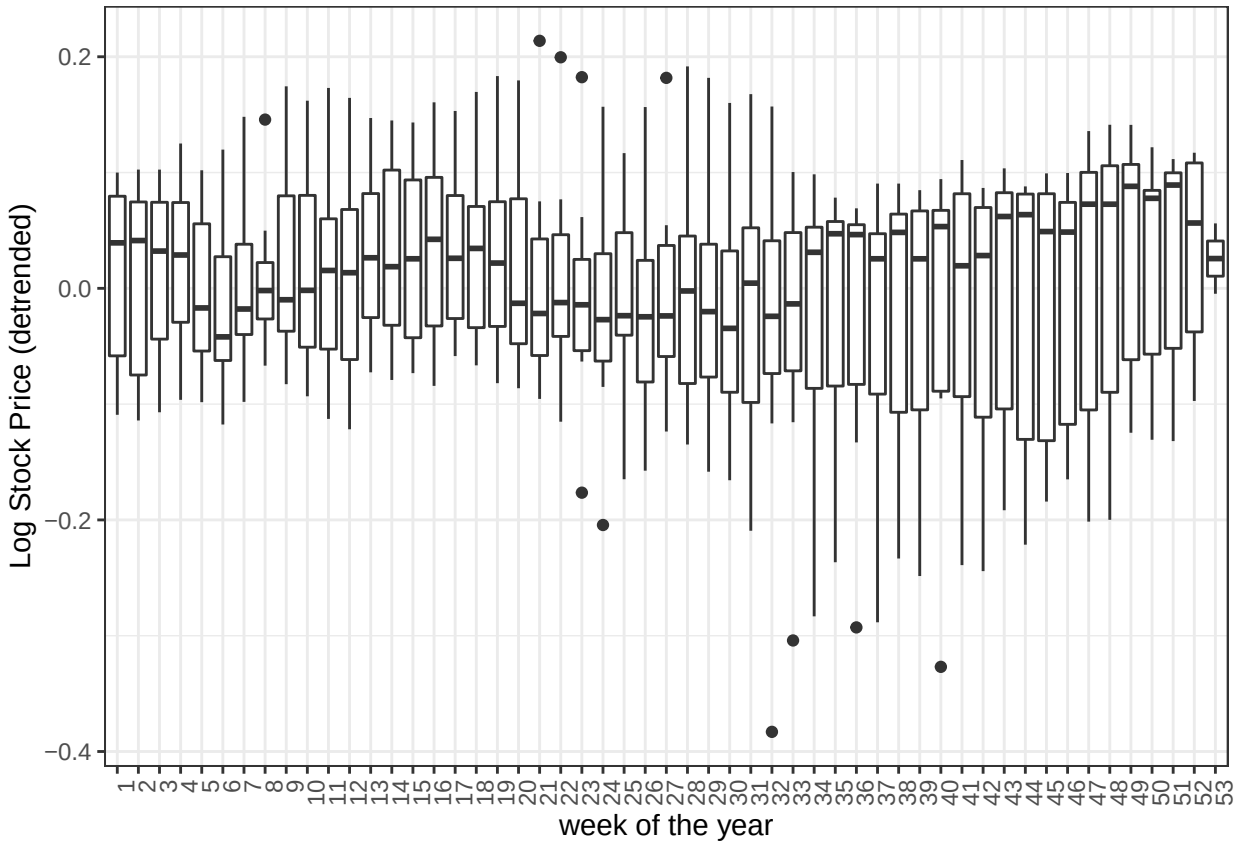
```
dat$idx <- as.numeric(dat$date - min(dat$date)) # translate date to integer
dat$log_stock_price <- log(dat$stock_price) # create new log_stock_price feature

# Create training and holdout set
br <- max(dat$date) - years(1)
print(br) # this is the break between training and validation/testing
```

```
## [1] "2018-03-31"
```

```
dat_train <- dat %>% filter(date <= br)
dat_test<- dat %>% filter(date > br)
```

Next I hypothesize that the week of the year makes a difference (lots of companies have seasonal patterns in their stock prices). So I am going to use some simple data visualization to explore this (after removing the obvious linear-ish trend).

```
detrend <- function(d){
  lm_fit <- lm(log_stock_price ~ idx, data=d)
  d$lsp_detrend <- residuals(lm_fit)
  return(d)
}

dat_train %>%
  mutate(week = factor(week(date))) %>% # calculate week of the year as an integer
  detrend() %>%
  ggplot(aes(x=week, y=lsp_detrend)) +
  geom_boxplot() +
  xlab("week of the year") +
  ylab("Log Stock Price (detrended)")+
  theme_bw() +
  theme(axis.text.x = element_text(angle=90, hjust=1))
```



Well alright then... I guess I was wrong. Seems week of the year not particularly important. Since I don't know the filing quarter for this particular company I am going to avoid trying to model quarterly effects. In essence, I am going to just skip modeling periodic effects at this point.

Part B

This is hard, while there are multiple ways of doing it, I am going to choose to only forecast forwards in time. For example, Starting 2014 till 2017 I am going to create 3 "folds".

fold	testing years	training years
1	2011 to 2014	2015
2	2011 to 2015	2016
3	2011 to 2016	2017

Finally I am going to leave the final year of data (2018) as my final validation set.

Part C

This is really the meat of the problem. It turns out that in this case, SciKit-Learn actually has most of what we need.

Note: I use the package `reticulate` in R to allow me to move flexible between R and Python, if you notice the language changes, you are correct.

```r
head(dat_train)
```

```
## # A tibble: 6 x 4
##   date       stock_price   idx log_stock_price
##   <date>           <dbl> <dbl>           <dbl>
## 1 2011-01-02        28.1     0            3.33
## 2 2011-01-09        27.3     7            3.31
## 3 2011-01-16        28.6    14            3.35
## 4 2011-01-23        29.1    21            3.37
## 5 2011-01-30        29.2    28            3.38
## 6 2011-02-06        29.1    35            3.37
```

```python
import numpy as np

from matplotlib import pyplot as plt
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels \
    import RBF, WhiteKernel, RationalQuadratic, ExpSineSquared, DotProduct

# Extract key data elements
y = r.dat_train.log_stock_price
x = r.dat_train.idx

# reshape x to 2d array
x = np.array(x).reshape(-1, 1)


# Create Kernel
# RBF*DotProduct Kernel for Long term Linear Trend
# RQ Kernel  for RBF Like Kernel that allows variable length scales
# White Noise Kernel for short term errors
# Note these kernels have parameters that will be fit using maximum marginal
# likelihood -- This is a default within scikit learn
kernel = 1**2 * RBF()*DotProduct() + 1**2 *RationalQuadratic()
# Mean function is mean of training set with normalize_y = True
# n_restarts_optimizer=10 (multiple restarts due to potential local optima)
gpr = GaussianProcessRegressor(kernel = kernel, normalize_y=True,
                               n_restarts_optimizer=10,
                               alpha=.04) # this is the white noise level
                                          # e.g., observation errors.

# Create the Time Series Cross Validation Setup I proposed Above.
# I chose the structure of the kernel (above) by minimizing the mean squared
#  error over the folds
# I chose the parameter alpha by hand based ion the results of part D. Probably
#  would have been a little better if I chose it based on cross validation
```

```
#  but I knew that would be computationally intensive and I didn't want to bother
# Again, the other kernel parameters are being fit by maximum marginal likelihood
# Also note that with 7 splits its not exactly what I proposed above
#   (each fold is smaller than a year), but... close enough.
n_splits = 7
tscv = TimeSeriesSplit(n_splits=n_splits)

# Create list to save results to
test_error = []

## Not the main code ##
for train_index, test_index in tscv.split(x):

  # ignore first few splits -- they are too small - want min 2 years
  if len(train_index) < 53*2: continue

  # split for training
  x_train, x_test = x[train_index], x[test_index]
  y_train, y_test = y[train_index], y[test_index]

  # fit gpr model
  _  = gpr.fit(x_train, y_train)
  y_pred_train = gpr.predict(x_train) # get the training error
  y_pred_test = gpr.predict(x_test) # get the training error

  # Remember we are on log-scale (in this case I want error on linear scale)
  test_error.append(mean_squared_error(np.exp(y_pred_test), np.exp(y_test)))



# Calculate MSE
print(np.mean(test_error))
```

```
## 1611.255033236833
```

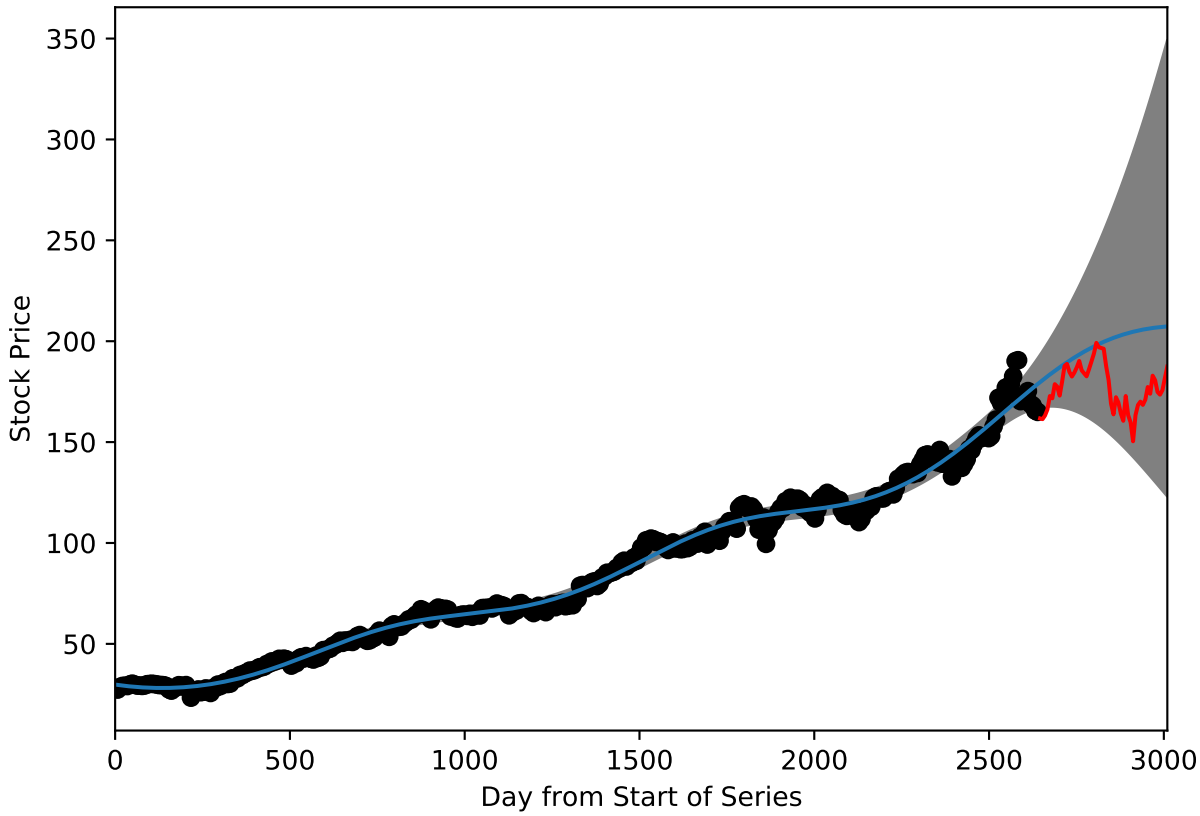
Part D

First going to test the model on the held-out validation set visually. Then I will remake the plot.

```
import textwrap

# Not plot the overall fitted model (all the training data)
_ = gpr.fit(x, y)
print(textwrap.fill("\nLearned kernel:\n%s" % gpr.kernel_, 70))
```

```
##  Learned kernel: 0.00316**2 * RBF(length_scale=2.44e+04) *
## DotProduct(sigma_0=0.0052) + 0.91**2 *
## RationalQuadratic(alpha=7.47e+03, length_scale=492)
```

```
x_ = np.linspace(x.min(), x.max() + 53*7, 1000)[:, np.newaxis]
y_pred, y_std = gpr.predict(x_, return_std=True)

# Illustration
_ = plt.scatter(x, np.exp(y), c='k')
_ = plt.plot(x_, np.exp(y_pred))
_ = plt.plot(r.dat_test.idx, r.dat_test.stock_price, color="red")
_ = plt.fill_between(x_[:, 0], np.exp(y_pred - 1.96* y_std),
                               np.exp(y_pred + 1.96* y_std),
                               alpha=0.5, color='k')
_ = plt.xlim(x_.min(), x_.max())
_ = plt.xlabel("Day from Start of Series")
_ = plt.ylabel("Stock Price")
plt.tight_layout()
plt.show()
```



I could probably continue to iterate on this for a while, but at the moment I think this is probably good enough. Notice how even though there was a bit of a "surprise" in 2018-2019 (the held out validation data) the posterior interval is still doing a fairly good job accounting for that. Again, probably could do a little better ultimately if I chose to tune `alpha` by cross validation but I didn't want to deal with the extra computational overhead (*e.g.,* I didn't wan to have to wait more than 60 seconds for my code to run because I am impatient and I don't think it would make a massive difference ultimately.) Overall I think this is reasonable.

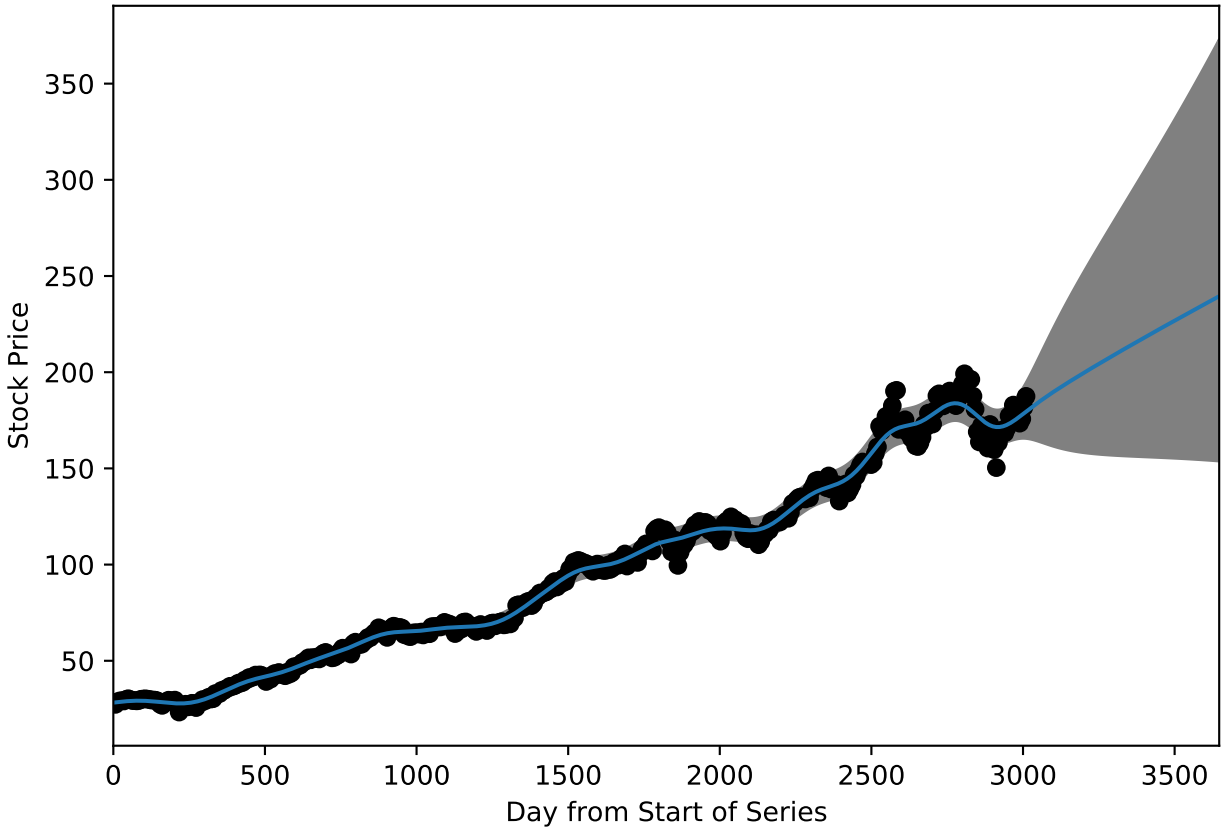
Now remake the plot with all the training data and show the predictions over the test-set.

```
test <- read_csv("data_test_midterm_problem1.csv")

# Create our week as an integer feature
test$idx <- as.numeric(test$date - min(dat$date)) # translate date to integer
```

```
import pandas as pd

# Extract data elements from entire training set
y = r.dat.log_stock_price
x = r.dat.idx
x_ = r.test.idx

# reshape xs to 2d array
x = np.array(x).reshape(-1, 1)
x_ = np.array(x_).reshape(-1, 1)

# Not plot the overall fitted model (all the training data)
_ = gpr.fit(x, y)
print(textwrap.fill("\nLearned kernel:\n%s" % gpr.kernel_, 70))
```

```
##  Learned kernel: 0.00316**2 * RBF(length_scale=2.23e+04) *
## DotProduct(sigma_0=0.187) + 1.84**2 * RationalQuadratic(alpha=0.00287,
## length_scale=1.49e+03)
```

```
x_ = np.linspace(x.min(), x_.max(), 1300)[:, np.newaxis]
y_pred, y_std = gpr.predict(x_, return_std=True)

# Illustration
_ = plt.scatter(x, np.exp(y), c='k')
_ = plt.plot(x_, np.exp(y_pred))
_ = plt.fill_between(x_[:, 0], np.exp(y_pred - 1.96* y_std),
                               np.exp(y_pred + 1.96* y_std),
                               alpha=0.5, color='k')
_ = plt.xlim(x_.min(), x_.max())
_ = plt.xlabel("Day from Start of Series")
_ = plt.ylabel("Stock Price")
plt.tight_layout()
plt.show()
```



While I am not going to show you the true values of the stock prices on the held-out set, I will point out that 2020 was a very volatile year (think COVID-19). Students that do well on this problem will likely have well calibrated 95% credible sets that can account for the high volatility.

Part E

Now I am going to generate 1000 samples from the posterior, this is easy with the method `sample_y` from scikit-learn.

```
# copy test data 1000 times (for sample_y function in scikit learn)
```

```
x_ = r.test.idx
x_ = np.array(x_).reshape(-1, 1) # Reshape to 2d array

y_ = gpr.sample_y(x_, n_samples=1000)

# Remember to transform back to original scale
y_ = np.exp(y_)
```

```
library(scoringRules)

soln <- read_csv("data_soln_midterm_problem1.csv")
```

```
## Parsed with column specification:
## cols(
```

```
##   date = col_date(format = ""),
##   stock_price = col_double()
## )
```

```
scores <- crps_sample(soln$stock_price, py$y_)
mean(scores) # Here is my score on this problem
```

```
## [1] 15.13912
```

Problem 2

This is a tough one. The answer is that there is no signal. This is a tough situation, you are often asked to "find" something and you need to be confident in your skills and your results, know when there is nothing there.

```
dat <- read_csv("data_midterm_problem2.csv")
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
## Parsed with column specification:
## cols(
##   X1 = col_double(),
##   y = col_double(),
##   x1 = col_double(),
##   x2 = col_double(),
##   x3 = col_double(),
##   x4 = col_double(),
##   x5 = col_double(),
##   x6 = col_double(),
##   x7 = col_double(),
##   x8 = col_double(),
##   x9 = col_double(),
##   x10 = col_double(),
##   x11 = col_double(),
##   x12 = col_double(),
##   x13 = col_double()
## )
```

```
# The first column is just row numbers - so remove it.
dat <- dat[,-1]

dim(dat)
```

```
## [1] 1000   14
```

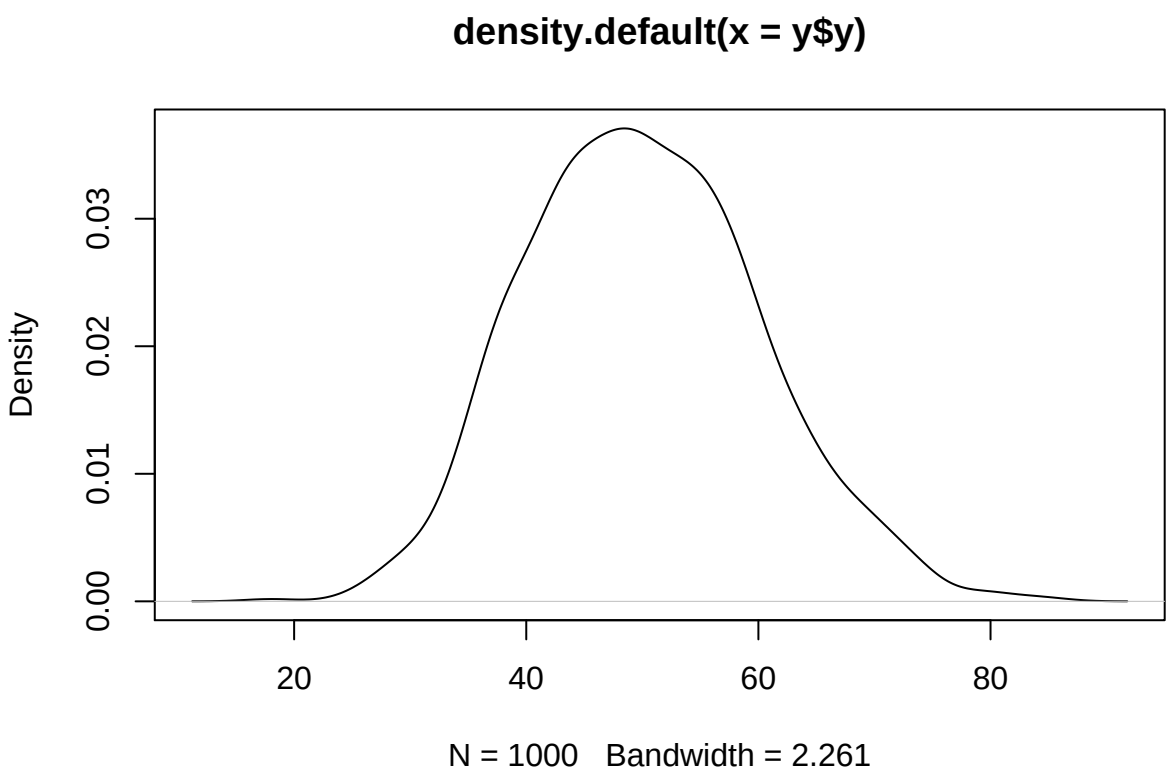
```
X <- dat[,-1]
y <- dat[,1]
```

Bit of Exploratory Data Analysis

```r
summary(y) # response seems to be integer valued
```

```
##        y
##  Min.   :18.00
##  1st Qu.:43.00
##  Median :50.00
##  Mean   :50.05
##  3rd Qu.:57.00
##  Max.   :85.00
```

```r
plot(density (y$y))
```




```r
summary(X)
```

```
##        x1              x2                x3              x4
##  Min.   :2.123   Min.   :-2.7989   Min.   :10.40   Min.   :-12.784
##  1st Qu.:2.916   1st Qu.:-1.5691   1st Qu.:12.12   1st Qu.:-11.248
##  Median :3.182   Median :-1.2363   Median :12.51   Median :-10.804
##  Mean   :3.186   Mean   :-1.2438   Mean   :12.51   Mean   :-10.762
##  3rd Qu.:3.450   3rd Qu.:-0.8893   3rd Qu.:12.90   3rd Qu.:-10.258
##  Max.   :4.448   Max.   : 0.1998   Max.   :14.30   Max.   : -8.493
##        x5               x6               x7              x8
##  Min.   :-11.41   Min.   : 8.485   Min.   :3.765   Min.   :13.07
##  1st Qu.:-10.57   1st Qu.: 9.614   1st Qu.:5.043   1st Qu.:14.29
##  Median :-10.31   Median : 9.880   Median :5.362   Median :14.55
##  Mean   :-10.29   Mean   : 9.887   Mean   :5.332   Mean   :14.56
##  3rd Qu.:-10.02   3rd Qu.:10.154   3rd Qu.:5.622   3rd Qu.:14.84
```

```
##  Max.   : -9.01   Max.   :11.522   Max.   :6.646   Max.   :15.70
##        x9              x10              x11             x12
##  Min.   :1.441   Min.   :0.8216   Min.   :3.376   Min.   :3.374
##  1st Qu.:2.484   1st Qu.:1.8322   1st Qu.:4.119   1st Qu.:4.024
##  Median :2.744   Median :2.1295   Median :4.342   Median :4.212
##  Mean   :2.748   Mean   :2.1333   Mean   :4.336   Mean   :4.213
##  3rd Qu.:3.021   3rd Qu.:2.4337   3rd Qu.:4.558   3rd Qu.:4.410
##  Max.   :3.900   Max.   :3.4292   Max.   :5.431   Max.   :5.054
##       x13
##  Min.   :-2.9334
##  1st Qu.:-1.6857
##  Median :-1.3102
##  Mean   :-1.3162
##  3rd Qu.:-0.9591
##  Max.   : 0.3868
```

I am going to scale the covariates and try a few different models here to try to predict y from the covariates $x_1, \ldots, x_13$. I am going to try Poisson regression with ridge penalty, and ridge regression. I have a feeling the target is count data (hence the Poisson regression), but I am not sure (hence the Ridge regression as well). Just to see if I should be looking for non-linear regression methods as well I will try throwing in a random forrest regression as well.

Now switch over to scikit-learn

```
import numpy as np
from scipy.stats import randint, uniform
from sklearn.linear_model import PoissonRegressor, Ridge
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import RandomizedSearchCV, train_test_split
from sklearn.metrics import r2_score

# Import from R
X = r.X
y = r.y
y = np.array(y).reshape(-1, 1)

pipelinePoisson = make_pipeline( StandardScaler(), PoissonRegressor(fit_intercept=True) )
pipelineRidge = make_pipeline(StandardScaler(), Ridge(fit_intercept=True) )
pipelineRF = make_pipeline(StandardScaler(), RandomForestRegressor(criterion="mse"))

pipelines = {"Poisson": pipelinePoisson,
             "Ridge": pipelineRidge,
             "RF" : pipelineRF}
paramdist = {}
paramdist["Poisson"] = {"poissonregressor__alpha" : uniform(0.01, .9) }
paramdist["Ridge"] = {"ridge__alpha" : uniform(0.01, .9) }
paramdist["RF"] = {"randomforestregressor__max_depth" : randint(2,6) }
```

```
# Create Hold-out set for final validation
X_train, X_test, y_train, y_test = train_test_split(r.X,r.y,test_size=0.2)


# tune parameters by randomized CV
search = {}
for p in pipelines:
  search[p] = RandomizedSearchCV(estimator=pipelines[p], n_iter=10,
                                 param_distributions=paramdist[p])

  # Fit parameters by CV
  _ = search[p].fit(X_train, y_train.values.ravel())
  print("Best Parameter: ", p, " ", search[p].best_params_)

# Now go an evaluate model performance on the held-out validation set
```

```
## Best Parameter:  Poisson   {'poissonregressor__alpha': 0.8226562509132176}
## Best Parameter:  Ridge   {'ridge__alpha': 0.8418123431137364}
## Best Parameter:  RF   {'randomforestregressor__max_depth': 3}
```

```
for p in pipelines:
  y_pred = search[p].predict(X_test)
  print("Model R2: ", p, " " , r2_score(y_test, y_pred))
```

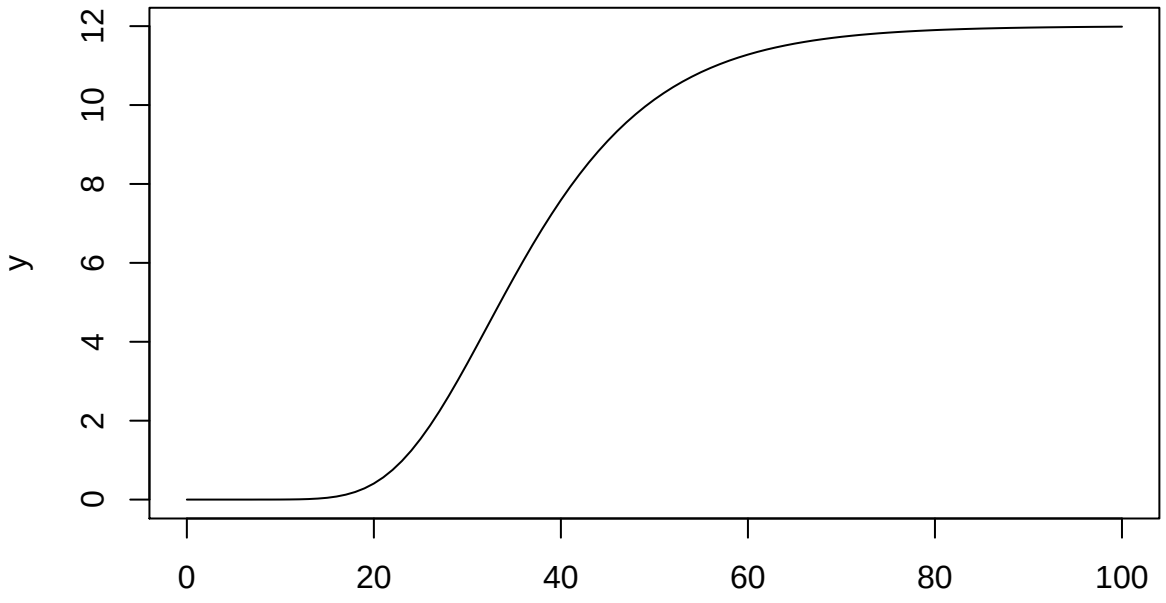
```
## Model R2:  Poisson   -0.027057540687607684
## Model R2:  Ridge   -0.030167375646770855
## Model R2:  RF   -0.022377564485793977
```

Wow that is terrible performance! 3 Very reasonable models, tuned reasonably, all do horrendously. Explaining at most about 2.5% of the variation in the dataset. **I am going to conclude that there is not much signal here.**

Problem 3

First I am going to simulate some Gompertz functions to see what these functions look like

```
x <- 0:100
a <- 12
b <- 25
c <- .1
y <- a*exp(-b*exp(-c*x))
plot(x,y, type="l")
```

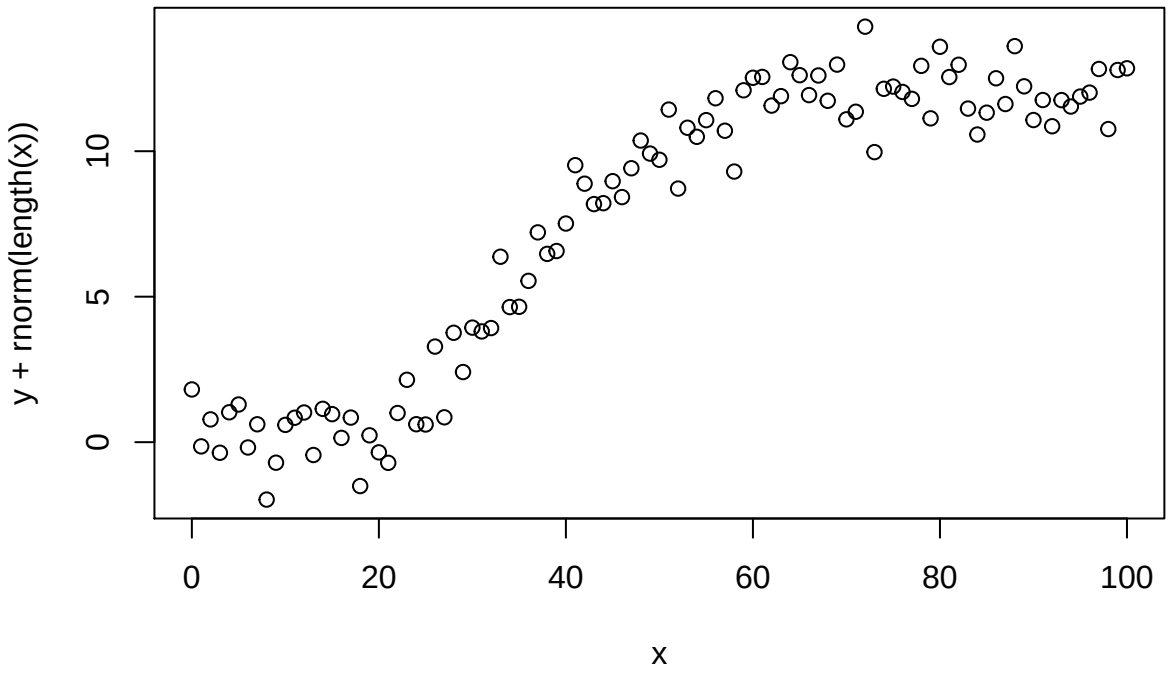


I played around with the parameters for `a`, `b`, and `c` until it looked remotely like the image we are trying to match (fitting by visualization). This gave me a pretty food feel for the Gompertz function. In particular, I notice that its a pretty smooth function. The observed data look pretty randomly distributed about what I expect to be the function mean. Without knowing more it seems reasonable to assume its normally distributed noise about the function. This leads me to the following solutions to the parts.

Part A

$$y_i \sim N(ae^{-be^{cx_i}}, \sigma^2)$$

```
plot(x, y+rnorm(length(x)))
```



Ya that's not that far off from whats in the figure.

Part B

$$(\hat{a}, \hat{b}, \hat{c}, \hat{\sigma^2}) = \underset{a,b,c,\sigma^2}{\operatorname{argmax}} N(y_i | ae^{-be^{cx_i}}, \sigma^2)$$

Part C

From playing around with the function, I am pretty sure that `a`, `b`, and `c` all need to be positive.

$$\begin{aligned} y_i &\sim N(ae^{-be^{cx_i}}, \sigma^2) \\ a &\sim Gamma(\alpha, \beta) \\ b &\sim Gamma(\delta, \epsilon) \\ c &\sim Gamma(\rho, \nu) \\ \sigma^2 &\sim InverseGamma(a, b) \end{aligned}$$

Part D

In this case I chose to specify a ridge penalty on the parameters `a`, `b`, and `c`

$$(\hat{a}, \hat{b}, \hat{c}, \hat{\sigma^2}) = \underset{a,b,c,\sigma^2}{\operatorname{argmax}} N(y_i | ae^{-be^{cx_i}}, \sigma^2) + \lambda(a^2 + b^2 + c^2)$$

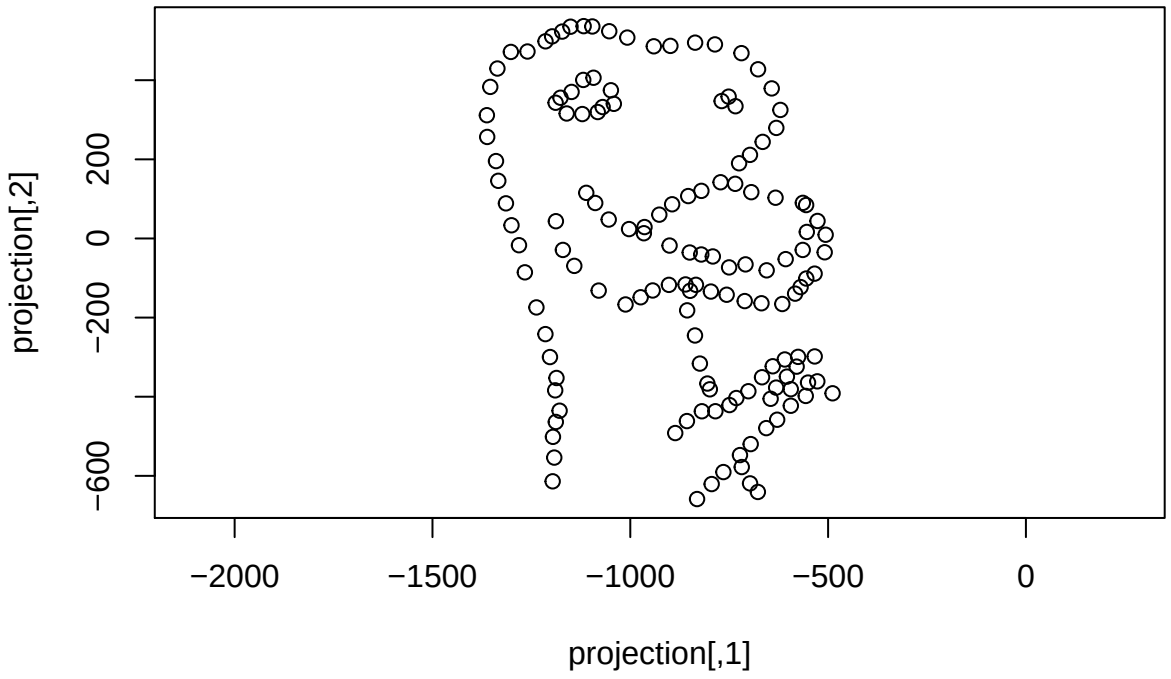
Part E

$$(\hat{a}, \hat{b}, \hat{c}) = \underset{a,b,c}{\operatorname{argmin}} (y_i - ae^{-be^{cx_i}})^2 + \lambda(a^2 + b^2 + c^2)$$

Problem 4

This sounds like a dimension reduction problem. Going to start with PCA

```
dat <- read_csv("data_midterm_problem4.csv")

decomp <- svd(dat)

projection <- decomp$u[,1:2] %*%  diag(decomp$d[1:2])

plot(projection, asp=1)
```



Hey! Its a Tyrannosaurus rex!

Problem 5

The answer is $p(\beta|\cdot) = Uniform(-\infty, \infty)$ or simply $p(\beta|\cdot) = c$ for some scalar constant c. Now to explain this solution:

Remember that $\hat{\beta}_{OLS} = \hat{\beta}_{MLE}$

$$\hat{\beta}_{MLE} = \underset{\beta}{\text{argmax}}\, N(y_i|\beta^T x_i, \sigma^2).$$

In other words, recall that MLE estimation and OLS are the same for linear regression. So lets write this problem a different (and more general way).

We are then trying to find $p(\beta|\cdot)$ such that

$$\underset{\beta}{\text{argmax}}\, N(y_i|\beta^T x_i, \sigma^2) = \underset{\beta}{\text{argmax}}\, N(y_i|\beta^T x_i, \sigma^2)p(\beta|\cdot)$$

This holds so long as $p(\beta|\cdot)$ is a constant, aka a flat prior over the entire range of possible values of β.

More generally this is pretty important. Beyond linear regression, you can always think of MLE as equivalent to MAP estimation with a uniform prior.

Written in the general case

Let $p(X|\theta)$ be a likelihood model for observed data X and parameters θ. Let θ_{MAP} be the *maximum a posteriori* estimate for θ given some prior $p(\theta)$:

$$\hat{\theta}_{MAP} = \underset{\theta}{\operatorname{argmax}}\, p(\theta|X).$$

Let $\hat{\theta}_{MLE}$ denote the maximum likelihood estimate for θ:

$$\hat{\theta}_{MLE} = \underset{\theta}{\operatorname{argmax}}\, p(X|\theta).$$

What probability distributions $p(\theta)$ guarantees that $\hat{\theta}_{MAP} = \hat{\theta}_{MLE}$.

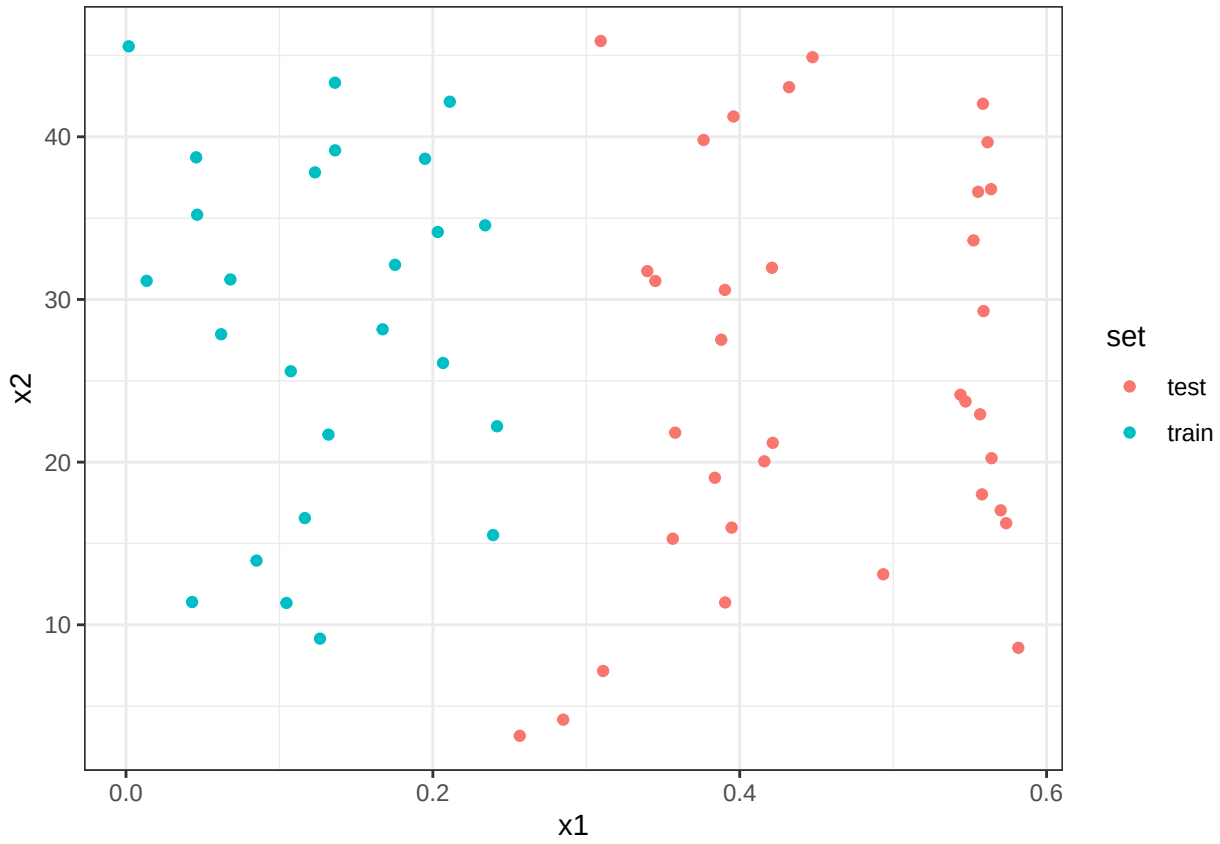
The answer is $p(\theta) = c$.

Why is this important? It points to one of the conceptual issues that can arise with MLE estimation. Given any linear regression problem, before seeing the data, do you truly believe that any value of β is equally likely? Probably, not. This is one of the major motivations behind Bayesian and penalized regression methods.

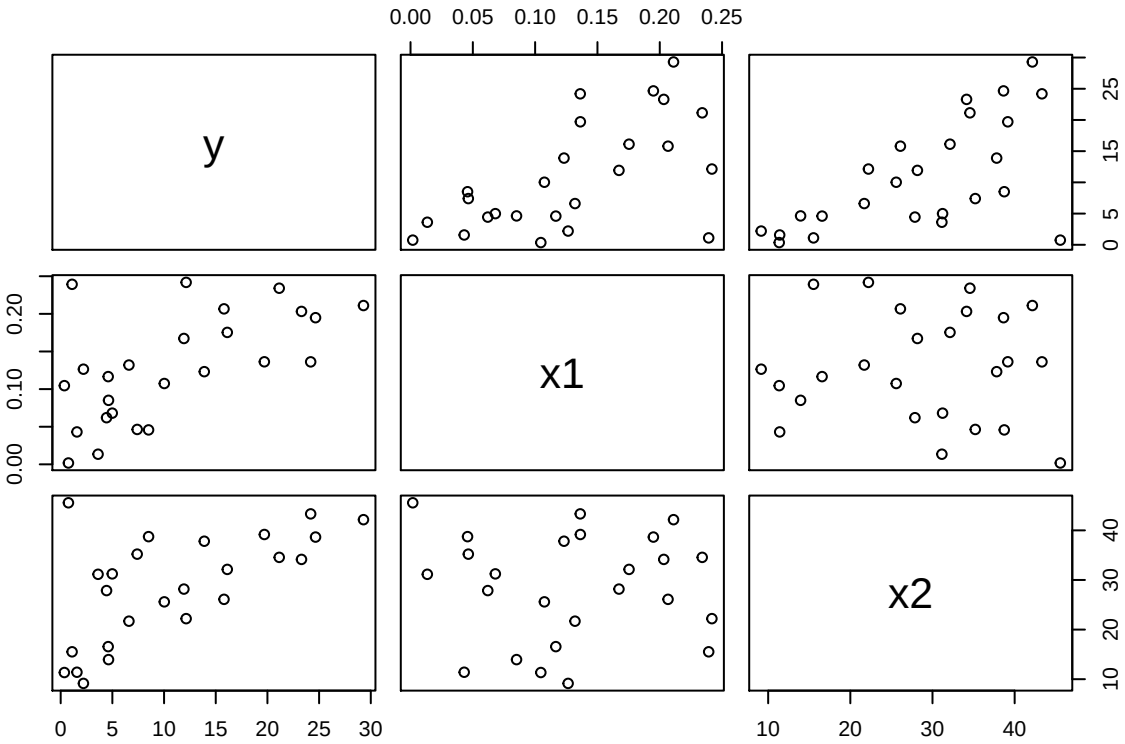
Problem 6

Exploratory Data Analysis

```
dat_train <- read_csv("data_train_midterm_problem6.csv")
dat_test <- read_csv("data_test_midterm_problem6.csv")

dat_train$set <-"train"
dat_test$set <- "test"

dat <- bind_rows(dat_train, dat_test)

dat %>%
  ggplot(aes(x=x1, y=x2, color=set)) +
  geom_point()+
  theme_bw()
```




```
plot(dat_train[,-4])
```



Conclusions from EDA:

If you notice, your friend was nasty and the held-out data is outside of the range of the training data. Also, the basic relationship between the knobs and the range of the ball is questionably non-linear and kinda a mess. Also there is not much data here. You are going to have a hard time using classic machine learning techniques here. Probably better to resort to basic physics and merge that with some of the concepts you have learned in this course so far.

A brief wikipedia search suggests that (ignoring wind drag), the range of a projectile thrown from ground level is given by:

$$y = \frac{v^2}{g}\sin(2\theta)$$

where v is the initial velocity (really speed) of the projectile and θ is the initial angle (in radians).

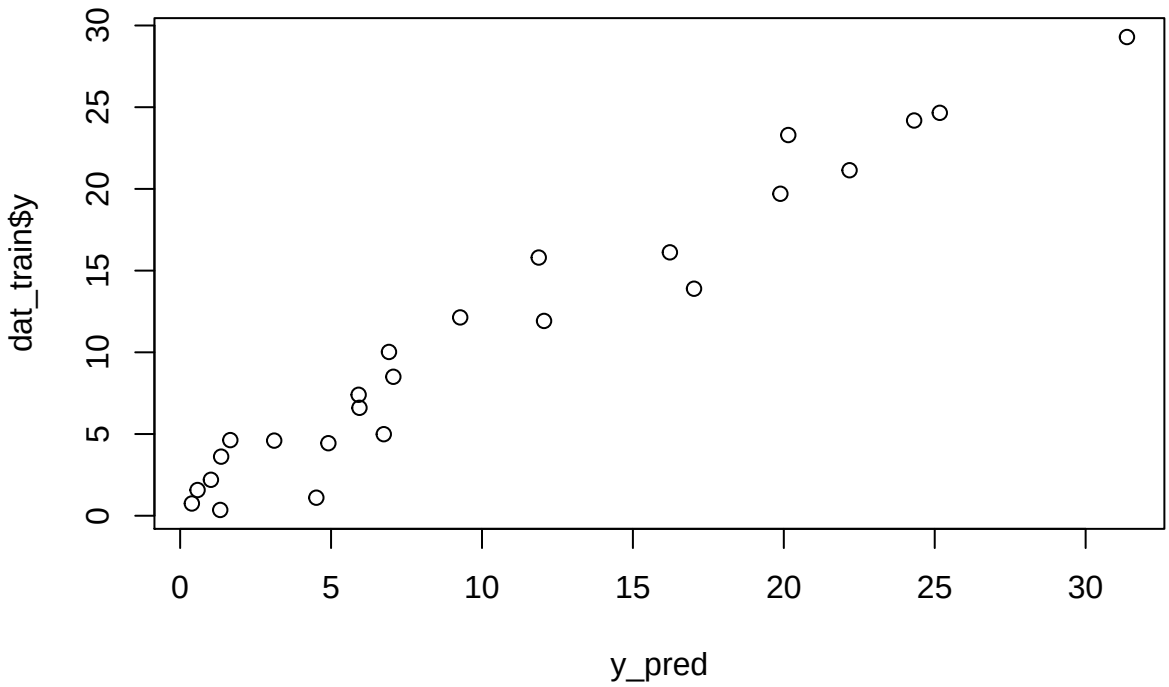
Your friend thought that the first and second knobs were linearly related to angle and velocity, so lets let $v = \alpha x_2$ and $\theta = \beta x_1$. Then lets formulate the problem as non-linear least squares:

$$\hat{\alpha}, \hat{\beta} = \underset{\alpha,\beta}{\operatorname{argmin}} \frac{1}{N}\sum_i \left(y_i - \frac{(\alpha x_2)^2}{g}\sin(2\beta x_1) \right)^2$$

We are going to just brute-force this and solve it using numerical optimization.

```
physics_predict <- function(dat, alpha, beta){
  x1 <- dat$x1
  x2 <- dat$x2
  g <- 9.8
  y_hat <- (alpha*x2)^2/g*sin(2*beta*x1)
  return( y_hat )
}

loss <- function(dat, alpha, beta){
  y_hat <- physics_predict(dat, alpha, beta)
  return(mean((dat$y-y_hat)^2))
}
# create partial function to make optimization easier
loss_train <- function(pars) loss(dat_train, pars[1], pars[2])

# Non-convex (use random restarts)
inits <- expand.grid(seq(.1, 3, by=0.3), seq(0.1, 3, by=0.3))
res <- list()
for (i in 1:nrow(inits)){
  res[[i]] <- optim(unlist(inits[i,]), loss_train)
}
best <- res %>%
  map("value") %>%
  unlist() %>%
  which.min()

res <- res[[best]]
```

```r
# Lets visualize the predictions
y_pred <- physics_predict(dat_train, res$par[1], res$par[2])

# Plot predicted vs observed
plot(y_pred, dat_train$y)
```




```r
# See whats the correlation to the training data?
cor(dat_train$y, y_pred)
```

```
## [1] 0.9765917
```

Seems like a very good model.

Lets now predict with it:

```r
y_pred <- physics_predict(dat_test, res$par[1], res$par[2])

# now calculate the MSE to the real data:
soln <- read_csv("data_soln_midterm_problem6.csv")

mean((y_pred - soln$y)^2)
```

```
## [1] 5.255104
```

We are getting extremely accurate predictions, even far outside of the training set, with a very small sample size and with the presence of noise in the outcome.

Nothing will beat a well formulated (and accurate) mechanistic model.