Noé Flores

Introduction:

The St. Louis Fed Financial Stress Index (STLFSI) measures the degree of financial stress in U.S. markets. The index consists of seven interest rate series, six yield spreads and five other indicators, for a total of 18 financial market variables that are subsequently compiled into one using PCA (Principal Component Analysis) which is a. As it currently stands, the STLFSI is a black box model. It provides an output for us to read, but we don't have a way to validate its effectiveness at predicting or indicating future stress in the market. We will perform an exploratory data analysis of the STLFSI and the S&P500 to gather information and build several models for comparison to determine the effectiveness of the STLFSI as an indicator variable for market performance. Finally, we will select the best predictive model and cover and go over any concerns in our analysis.

Data Preparation:

Our data contains two different variables. The table provides a breakdown and description of those variables.

Figure 1: Variables and Descriptions.

	DATE	STLFS (Head)
1	1/1/1994	0.146
2	2/1/1994	0.334
3	3/1/1994	0.481
	DATE	STLFSI (Tail)
284	8/1/2017	-1.459
285	9/1/2017	-1.542
286	10/1/2017	-1.526
286 obs. of 2 variables:		

	DATE	SP500 (Head)
1	12/1/2007	1468.36
2	1/1/2008	1378.55
3	2/1/2008	1330.63
	DATE	SP500 (Tail)
117	8/1/2017	2471.65
118	9/1/2017	2519.36
119	10/1/2017	2575.26
119 obs. of 2 variables:		

Our datasets have two distinct starting points. We will fix some of this issue by first converting the date from a string object to a proper date field(a similar approach will be taken with the S&P500 data). Second, we will also create three new forward-looking dates of +1,+2,+3 months.

Figure 2: STLFSI Forward-looking Dates.

	DATE	STLFSI	RDATE0	RDATE1	RDATE2	RDATE3
1	1/1/1994	0.146	1994-01-01	1994-02-01	1994-03-01	1994-04-01
2	2/1/1994	0.334	1994-02-01	1994-03-01	1994-04-01	1994-05-01
3	3/1/1994	0.481	1994-03-01	1994-04-01	1994-05-01	1994-06-01
4	4/1/1994	0.617	1994-04-01	1994-05-01	1994-06-01	1994-07-01

We perform a similar date adjustment to the S&P500 data and create three new variables which we can use to merge against the STLFSI data.

Figure 3: S&P500 Forward looking Dates.

	DATE	SP500	RDATE	SP500_0	SP500_1	SP500_2	SP500_3
1	12/1/2007	1468.36	2007-12-01	1468.36	1468.36	1468.36	1468.36
2	1/1/2008	1378.55	2008-01-01	1378.55	1378.55	1378.55	1378.55
3	2/1/2008	1330.63	2008-02-01	1330.63	1330.63	1330.63	1330.63
4	3/1/2008	1322.70	2008-03-01	1322.70	1322.70	1322.70	1322.70

The STLFSI and S&P500 are now merged together, and we now have the correct S&P500 value for RDATE0, RDATE1, RDATE2, and RDATE3. We use these S&P500 values to compute the returns over these windows, which will serve as response variables to compare against STLFSI.

Figure 4: Merger of STLFSI and S&P500

	RDATE3	RDATE2	RDATE1	RDATE0	DATE	STLFSI	SP500_0	SP500_1	SP500_2	SP500_3
1	2008-03-01	2008-02-01	2008-01-01	2007-12-01	12/1/2007	1.000	1468.36	1378.55	1330.63	1322.70
2	2008-04-01	2008-03-01	2008-02-01	2008-01-01	1/1/2008	0.896	1378.55	1330.63	1322.70	1385.59
3	2008-05-01	2008-04-01	2008-03-01	2008-02-01	2/1/2008	0.861	1330.63	1322.70	1385.59	1400.38
4	2008-06-01	2008-05-01	2008-04-01	2008-03-01	3/1/2008	1.150	1322.70	1385.59	1400.38	1280.00

To complete the data build for our analysis, we compute the log-returns for the S&P500 over the course of the new time windows we created. We also generate a minR(Min-return) variable using the pmin() function in R. This allows us to take more than one vector and produce a single vector with the smallest value for each data point. In our case, we wanted the lowest value across R1, R2, and R3 returns. The table below represents an abbreviated version of those results.

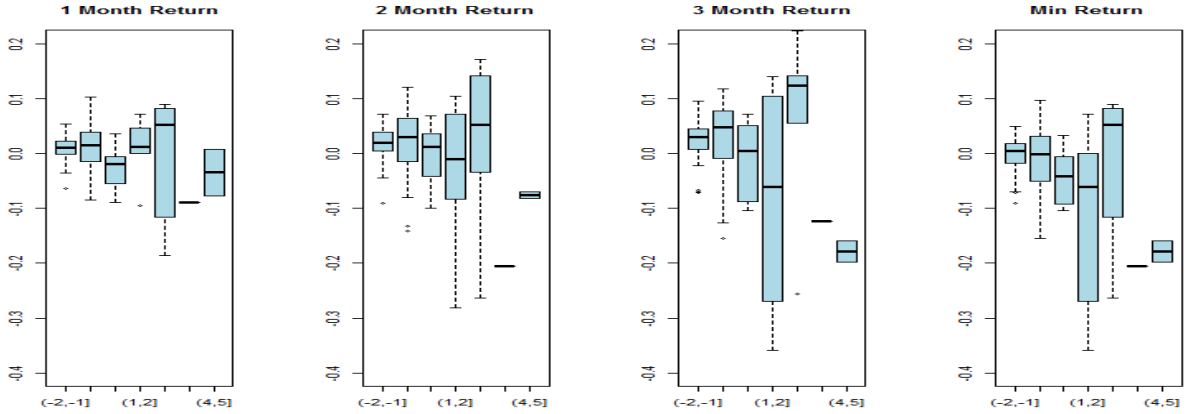
Figure 5: S&P500 returns.

	RDATE3	RDATE2	RDATE1	RDATE0	DATE	STLFSI	R1	R2	R3	minR
1	2008-03-01	2008-02-01	2008-01-01	2007-12-01	12/1/2007	1.000	-0.063113910	-0.09849362	-0.10447103	-0.104471030
2	2008-04-01	2008-03-01	2008-02-01	2008-01-01	1/1/2008	0.896	-0.035379708	-0.04135712	0.00509382	-0.041357120
3	2008-05-01	2008-04-01	2008-03-01	2008-02-01	2/1/2008	0.861	-0.005977412	0.04047353	0.05109111	-0.005977412
4	2008-06-01	2008-05-01	2008-04-01	2008-03-01	3/1/2008	1.150	0.046450940	0.05706853	-0.03281502	-0.032815024
5	2008-07-01	2008-06-01	2008-05-01	2008-04-01	4/1/2008	1.203	0.010617587	-0.07926596	-0.08917426	-0.089174264
6	2008-08-01	2008-07-01	2008-06-01	2008-05-01	5/1/2008	0.649	-0.089883550	-0.09979185	-0.08767505	-0.099791851

Exploratory Data Analysis:

Exploratory analysis is a critical first step for predictive modeling. The box-plots provide a view of the distributions of returns for discrete values of the STLFSI. The y-axes have a normal range across our boxplots is so we can compare the effect of the STLFSI across our return horizons. The boxplots give us an indication of the range of the results, but it is hard to discern the actual relationship between the values of our variables in the short run. We expect a negative correlation between the two variables and that becomes more prominent in the two and three month returns. While not totally clear, we can get a sense of the frequency of the results at each level of STLFSI. This is something we would want to clarify.

Figure 6: Box-plots and Q-Q plot of variables.



We want a better idea of the predictive power of the STLFSI. One way we could do that is to create a table to see where the majority of our return data lies, with respect to specific values of STLFSI. The table was made by first creating an indicator variable for the different return windows. Every time a return is less than -0.05, we will mark that with a (+1). From there, we can calculate the empirical probabilities

Figure 7: Box-plots and Q-Q plot of variables.

R1							
	(-2,-1]	**(-1,0]**	**(0,1]**	**(1,2]**	**(2,3]**	**(3,4]**	**(4,5]**
0	0.98	0.88	0.67	0.83	0.60	0.00	0.50
1	0.02	0.12	0.33	0.17	0.40	1.00	0.50
R2							
	(-2,-1]	(-1,0]	(0,1]	(1,2]	(2,3]	(3,4]	(4,5]
0	0.98	0.84	0.78	0.50	0.80	0.00	0.00
1	0.02	0.16	0.22	0.50	0.20	1.00	1.00
R3							
	(-2,-1]	**(-1,0]**	**(0,1]**	**(1,2]**	**(2,3]**	**(3,4]**	**(4,5]**
0	0.94	0.86	0.67	0.50	0.80	0.00	0.00
1	0.06	0.14	0.33	0.50	0.20	1.00	1.00
Rmin							
	(-2,-1]	**(-1,0]**	**(0,1]**	**(1,2]**	**(2,3]**	**(3,4]**	**(4,5]**
0	0.92	0.74	0.56	0.50	0.60	0.00	0.00
1	0.08	0.26	0.44	0.50	0.40	1.00	1.00

The table above appears to suggest that as the values of STLFSI increase, we also see an increase in the number of times our returns are less than (-0.05) which gets flagged as a (+1) up to a certain point. We can also observe that as STLFSI increases well past (1,2], which is where most of our data is located, that the relationship between the two variables isn't as clear the further out you go. This could once again be due to the fact that the data is limited at those levels of the STLFSI.

Modeling:
We will look to build three distinct models and compare their performance to assess the predictive power of the STLFSI with respect to the future returns of the S&P500

Model 1:

Model 1 Logistic Regression
model.1 <- glm(drop5.minR ~ STLFSI, family=binomial, data=returns.df)

Model 1 is a logistic regression model regressing the return data against the STLFSI index data, which is continuous. The summary of the model is below.

Model 1

```
Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.5716  -0.6459  -0.5468  -0.4968   1.9844

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -0.9925     0.2432  -4.082 4.47e-05 ***
STLFSI        0.6645     0.1860   3.572 0.000354 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 125.88  on 115  degrees of freedom
Residual deviance: 110.39  on 114  degrees of freedom
AIC: 114.39
```

Since out model saves the fitted values (estimated responses) we can use that information to aggregate those results against the discreteSTL and compare it to the actual results from our data.

Discrete vs Fitted Values(estimated Response)

	discreteSTL	x
1	(-2,-1]	0.1350247
2	(-1,0]	0.1947642
3	(0,1]	0.3595218
4	(1,2]	0.4513278
5	(2,3]	0.6800380
6	(3,4]	0.7692787
7	(4,5]	0.8803854

Discret vs Actual results Rmin Table

	discreteSTL	x
1	(-2,-1]	0.0800000
2	(-1,0]	0.2558140
3	(0,1]	0.4444444
4	(1,2]	0.5000000
5	(2,3]	0.4000000
6	(3,4]	1.0000000
7	(4,5]	1.0000000

As we can see from the results above, the model actually does a reasonably decent job fitting the values close to the actual result. We also take note of the AIC (Akaike's Information Criterion) value. AIC is an estimator of the relative quality of statistical models for a given set of data. We can use this as a comparison to our other models. We typically want to see lower values of AIC.

Model 2:

Model 2 Logistic Regression
model.3 <- glm(drop5.minR ~ STLFSI, family=binomial, data=returns.df)

Model 2 is a logistic regression model regressing the return data against the discrete STLFSI index data, which is NOT continuous. The summary of the model is below.

Model 2

```
Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.1774  -0.7687  -0.4084  -0.4084   2.2475

Coefficients:
                  Estimate Std. Error z value Pr(>|z|)
(Intercept)        -2.4423     0.5213  -4.685  2.8e-06 ***
discreteSTL(-1,0]   1.3745     0.6276   2.190   0.0285 *
discreteSTL(0,1]    2.2192     0.8496   2.612   0.0090 **
discreteSTL(1,2]    2.4423     0.9687   2.521   0.0117 *
discreteSTL(2,3]    2.0369     1.0512   1.938   0.0527 .
discreteSTL(3,4]   19.0084  2399.5448   0.008   0.9937
discreteSTL(4,5]   19.0084  1696.7344   0.011   0.9911
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 125.88  on 115  degrees of freedom
Residual deviance: 104.19  on 109  degrees of freedom
AIC: 118.19
```

We can use the coefficients to build expected values or instances of our discrete STLFSI predictor variable.

pi Values		Rmin Table Results	1	0
	0.08	(-2,-1]	0.08	0.92
discreteSTL(-1,0]	0.26	(-1,0]	0.26	0.74
discreteSTL(0,1]	0.44	(0,1]	0.44	0.56
discreteSTL(1,2]	0.5	(1,2]	0.5	0.5
discreteSTL(2,3]	0.4	(2,3]	0.4	0.6
discreteSTL(3,4]	1.0	(3,4]	1.0	0.0
discreteSTL(4,5]	1.0	(4,5]	1.0	0.0

The Pi calculation actually produces the exact same results from the Rmin table we calculated earlier, which displayed where the majority of the return data fell with respect to discrete values of STLFSI. Based on these results, it would appear that the discrete predictor variable is able to produce better fitting coefficients.

Model 3:

Model 3 Linear Regression
model.3 <- lm(drop5.minR ~ discreteSTL, data=returns.df)

Model 2 is a logistic regression model regressing the return data against the discrete STLFSI index data, which is NOT continuous. The summary of the model is below.

Model 3

```
Deviance Residuals:
    Min       1Q   Median       3Q      Max
-0.5000  -0.2558  -0.0800  -0.0800   0.9200

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)        0.08000    0.05550   1.441  0.15234
discreteSTL(-1,0]  0.17581    0.08162   2.154  0.03344 *
discreteSTL(0,1]   0.36444    0.14211   2.565  0.01169 *
discreteSTL(1,2]   0.42000    0.16956   2.477  0.01479 *
discreteSTL(2,3]   0.32000    0.18408   1.738  0.08496 .
discreteSTL(3,4]   0.92000    0.39636   2.321  0.02214 *
discreteSTL(4,5]   0.92000    0.28300   3.251  0.00153 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 0.1540208)

Residual standard error: 0.3925 on 109 degrees of freedom
Multiple R-squared:  0.1896, Adjusted R-squared:  0.145
AIC: 120.98
```

Once again, since out model saves the fitted values, we can use that information to aggregate those results against the discreteSTL and compare it to the actual results from our data.

Discrete vs. Fitted Values Model 3	x
(-2,-1]	0.08
(-1,0]	0.26
(0,1]	0.44
(1,2]	0.5
(2,3]	0.4
(3,4]	1.0
(4,5]	1.0

Discret vs. Actual results Rmin Table	1	0
(-2,-1]	0.08	0.92
(-1,0]	0.26	0.74
(0,1]	0.44	0.56
(1,2]	0.5	0.5
(2,3]	0.4	0.6
(3,4]	1.0	0.0
(4,5]	1.0	0.0

The linear regression model was able to produce the same results as the Rmin table we calculated earlier. Typically, we do not use linear regression when we have discrete variables. Straight away we can see that the R-Squared value is very low. R-squared effectively explains how much of variability in the dependent variable is explained by the independent variable. In our case, it's very low. Since we were able to tie this model to some of the empirical values in our data, it is valid in that sense

Model Selection:

Before completing the full judgment on our models, we used the Akaike information criterion (AIC) to have a baseline comparison

Figure 6: Model AIC

Model	AIC
Model 1	**114.39**
Model 2	118.19
Model 3	120.98

What we are ultimately trying to do is model the predictive ability of the STLFSI for the log-returns of the S&P500. If base the decision on the AIC criterion, then we would choose Model.1. If we wanted better interpretability, would choose Model.3 since linear regression models produce the most interpretable results. However, we know this model suffered from a low R-Squared, and it had the Highest AIC value. Model.2 performed well, but it uses discrete variables for building the model. With that said, I believe Model.1 is the best model for the task.

Discussion:

limitations:

There were some limitations with the models we created. The data wasn't uniform, and it took some manipulation to get everything in order. The manner in which the data is downloaded is also an issue. The data on the FRED website is downloaded by the user which could lead to discrepancies or issues with compatibility. We also paid little attention to the assumption on normality, although I believe assumptions on normality aren't as critical when working with logistic regression.

Future work and learning:

In future models of this nature, I would look to experiment with more sophisticated modeling techniques. In this particular case, we made use of time-tested regression models. Incorporating Neural Networks or Decision Tree's could have helped us produce more statistically significant results. The classification Tree's have the added benefit of interpretability, even for those who aren't adept at evaluating models. Even without the use of more advanced techniques, we learned that we should build a few models and have set criteria for model selection.

Appendix: R-Code

```
STLFSI.df <- read.csv(file.path("C:/Users/Noe/Downloads/STLFSI.csv"),sep=",")
SP500.df <- read.csv(file.path("C:/Users/Noe/Downloads/SP500.csv"),sep=",")
#1) Working with dates in R is not particularly easy. In order to make working with dates easier
#let's begin by installing the lubridate package
# install the lubridate package;
install.packages('lubridate', dependencies=TRUE)
#(2) Let's also make our file reads easier by defining some file parameters
my.path <- 'C:/Users/Noe/Downloads/';
file.1 <- paste(my.path,'STLFSI.csv',sep='');
file.2 <- paste(my.path,'SP500.csv',sep='');
#(3) Read in the STLFSI data using read.csv(). In addition we need to create an R date object for
#the date. Notice that when we read in the file, the field DATE is a string, not a date.
#We will also create three new dates +1, +2, +3 months in the future using the lubridate package
fsi <- read.csv(file.1);
head(fsi)
tail(fsi)
str(fsi)
sp <- read.csv(file.2);
head(sp)
tail(sp)
str(sp)
# Create an R date from the field DATE;
fsi$RDATE0 <- as.Date(fsi$DATE,'%m/%d/%Y');
str(fsi)
head(fsi)
# Increment dates using the lubridate package;
library(lubridate)
```

```
# create R date plus 1 month
fsi$RDATE1 <- fsi$RDATE0 + months(1);
# create R date plus 2 months
fsi$RDATE2 <- fsi$RDATE0 + months(2);
# create R date plus 3 months
fsi$RDATE3 <- fsi$RDATE0 + months(3);
# Check the dates;
head(fsi)
#(4) Now let's read in the S&P 500 data. Just like we did with the STLFSI data, we will use the
#DATE field to create the R date RDATE. In addition we will create the variables SP500_0, SP500_1,
#SP500_2, SP500_3. Can you guess why we need these fields?
sp500 <- read.csv(file.2);
head(sp500)
str(sp500)
# create an R date from the date string;
sp500$RDATE <- as.Date(sp500$DATE,'%m/%d/%Y');
# Add replicate columns for merging;
sp500$SP500_0 <- sp500$SP500;
sp500$SP500_1 <- sp500$SP500;
sp500$SP500_2 <- sp500$SP500;
sp500$SP500_3 <- sp500$SP500;
# Check your dataframe and its components
str(sp500)
head(sp500)
#(5) Use the merge() function to perform repeated merging of the data sets. Note that the merge()
#function will produce 'inner joins'.
merge.0 <- merge(x=fsi,y=sp500[,c('RDATE','SP500_0')],by.x='RDATE0',by.y='RDATE');
head(merge.0)
```

```
merge.1 <- merge(x=merge.0,y=sp500[,c('RDATE','SP500_1')],by.x='RDATE1',by.y='RDATE');
head(merge.1)
merge.2 <- merge(x=merge.1,y=sp500[,c('RDATE','SP500_2')],by.x='RDATE2',by.y='RDATE');
head(merge.2)
merge.3 <- merge(x=merge.2,y=sp500[,c('RDATE','SP500_3')],by.x='RDATE3',by.y='RDATE');
head(merge.3)
# Compare the merge results to the original SP500 file;
head(sp500)
#Compare the results from the merges to the original SP500 file. Do the S&P 500 values seem to be
#in the correct place? For each RDATE0 we should have the S&P 500 value for RDATE0, RDATE1, RDATE2,
#and RDATE3. We can use these future S&P 500 values to compute the returns over these windows.
#These will be our response variables. They are defined to have forward looking
#performance windows of +1, +2, and +3 months.
#(6) Finish the data build by computing the log-returns over the performance windows.
#We need to have some returns to validate the STLFSI.
returns.df <- merge.3;
returns.df$R1 <- log(returns.df$SP500_1)-log(returns.df$SP500_0);
returns.df$R2 <- log(returns.df$SP500_2)-log(returns.df$SP500_0);
returns.df$R3 <- log(returns.df$SP500_3)-log(returns.df$SP500_0);
returns.df$minR <- pmin(returns.df$R1,returns.df$R2,returns.df$R3);
##Check
head(returns.df)
#We now have four different returns computed over the 1-3 month performance windows.
#R Note: Do we know how the pmin() and pmax() functions operate? How do they differ from
#the functions min() and max()? Why did we use pmin() instead of min()?
#Part 2: Exploratory Data Analysis
#Now that we have a basic data set built, we will begin our exploratory data analysis.
#What types of EDA might be useful on this type of data? What might we be trying to show?
```

```
 #(7) Let's begin by making a panel of boxplots.
# Discretize index values;
returns.df$discreteSTL <- cut(returns.df$STLFSI,breaks=c(-2,-1,0,1,2,3,4,5));
# Use par(mfrow) to make a panel of plots in R;
par(mfrow=c(1,4))
boxplot(R1 ~ discreteSTL,data=returns.df, ylim=c(-0.4,0.2), col = "lightblue", main='1 Month Return');
boxplot(R2 ~ discreteSTL,data=returns.df, ylim=c(-0.4,0.2), col = "lightblue",main='2 Month Return');
boxplot(R3 ~ discreteSTL,data=returns.df, ylim=c(-0.4,0.2), col = "lightblue",main='3 Month Return');
boxplot(minR ~ discreteSTL,data=returns.df, ylim=c(-0.4,0.2), col = "lightblue",main='Min Return');
#Is this panel of boxplots particularly useful? What does it show? What does it not show?
#What might we want to know that we cannot discern from these boxplots?
#(8) Make a Table: For simple data sets tables still help you understand the data regardless of the
#number of observations. Intuitively,
#we might think that higher STLFSI values should predict declines in the S&P 500 Index.
#We might make a graph, but our data set is so simple that a table would be far more precise
#than a graph. From this table we can see where the majority of our data lies,
#and we can compute empirical probabilities.
# Create indicator variables for positive returns;
returns.df$drop5.R1 <- ifelse(-0.05 > returns.df$R1,1,0);
returns.df$drop5.R2 <- ifelse(-0.05 > returns.df$R2,1,0);
returns.df$drop5.R3 <- ifelse(-0.05 > returns.df$R3,1,0);
returns.df$drop5.minR <- ifelse(-0.05 > returns.df$minR,1,0);
# Note: Cannot have returns.df$R1 < -0.05;
# R will interpret as the assignment operator!
# returns.df$drop5.minR <- ifelse(returns.df$minR<-0.05,1,0);
head(returns.df)
# Cross-tab of drop5.R1 and discreteSTL;
table(returns.df$drop5.R1,returns.df$discreteSTL)
```

```
table(returns.df$discreteSTL)
# drop5.R1 In percent format;
table(returns.df$drop5.R1,returns.df$discreteSTL)/rbind(table(returns.df$discreteSTL),
                                    table(returns.df$discreteSTL))
# drop5.R2 In percent format;
table(returns.df$drop5.R2,returns.df$discreteSTL)/rbind(table(returns.df$discreteSTL),
                                    table(returns.df$discreteSTL))
# drop5.R3 In percent format;
table(returns.df$drop5.R3,returns.df$discreteSTL)/rbind(table(returns.df$discreteSTL),
                                    table(returns.df$discreteSTL))
# drop5.minR In percent format;
table(returns.df$drop5.minR,returns.df$discreteSTL)/rbind(table(returns.df$discreteSTL),
                                     table(returns.df$discreteSTL))
# Cross-tab of drop5.R1 and discreteSTL;
table(returns.df$drop5.minR,returns.df$discreteSTL)
#########################################################################Part 3: Model Building
#Let's now consider three different models for modeling STLFSI. Here are some code snippets that
#define the three models. We have also included some hints about the models.
model.1 <- glm(drop5.minR ~ STLFSI, family=binomial, data=returns.df)
summary(model.1)
# What information is saved in model.1?
names(model.1)
returns.df$model1.scores <- model.1$fitted.values;
aggregate(x=returns.df$model1.scores, by=list(discreteSTL=returns.df$discreteSTL), FUN=mean)
aggregate(x=returns.df$drop5.minR, by=list(discreteSTL=returns.df$discreteSTL), FUN=mean)
#############################################################################################
Model #2
model.2 <- glm(drop5.minR ~ discreteSTL, family=binomial, data=returns.df)
```

```
summary(model.2)

# What information is saved in model.1?

names(model.2)

model.2$coef

XB <- model.2$coef[1] + c(0,model.2$coef[2:7]);

pi <- exp(XB)/(1+exp(XB));

##############################################################################################
Model #3

model.3 <- lm(drop5.minR ~ STLFSI, data=returns.df)

summary(model.3)

model.3$coef

XB1 <- model.3$coef[1] + c(0,model.3$coef[2:7]);

pi2 <- exp(XB1)/(1+exp(XB1));

returns.df$model.3.scores <- model.3$fitted.values;

aggregate(x=returns.df$model.3.scores, by=list(discreteSTL=returns.df$discreteSTL), FUN=mean)

aggregate(x=returns.df$drop5.minR, by=list(discreteSTL=returns.df$discreteSTL), FUN=mean)

plot(model.1$residuals)

plot(model.1$fitted.values)
```