Course Outline
FinTech Analytics: Data-driven Credit Modeling

Instructor: Roger M. Stein, Ph. D.
Lectures: See syllabus
Office Hours: TBD

Course Description: This course focuses on the practical challenges that arise in implementing a variety of credit models (e.g., bankruptcy and default models retail and commercial entities). We explore a number of data-driven approaches to modeling the likelihood that credit-risky borrowers will default on their obligations using large data sets. This course tends heavily towards discussions of practical model implementations and the “frictions” that make these implementations difficult in real-world settings. We discuss a number of modeling frameworks for estimating default probabilities (PDs) and loss given default LGD. We do not focus as heavily on the structure of credit markets or the details of pricing a broad variety of credit-risky instruments.

We will take the view that an effective, practical credit modeling framework will be rough around the edges with the odd inconsistency (usually to deal with available data or the lack thereof). This implies that seemingly incompatible models can each have value in specific contexts, resulting in retention of several models despite their theoretical inconsistency. Because the focus is applied, we will also discuss model validation and calibration in detail and highlight data issues in estimation and validation. Since credit models for corporate debt are most well developed, we deal extensively with these models, though we will discuss certain retail asset classes as well. Lectures will to focus on conceptual themes and practical issues, with much of the technical detail underlying these to be found in the instructor’s text.

We will use a number of large data sets over the course of the semester. Students will be required to implement a number of modules in the R language. Though this is not a programming course, some level of comfort with high-level programming languages will be beneficial.

Students are required to do one main project project (see below) which is drawn from industry and which will provide real-world exposure to realistic problems in credit risk. There will also be mini-projects, usually dealing with bite-sized components of the main project. Subject to scheduling constraints, we will also be joined by industry experts who will present on their areas of expertise.

Course Objective: To expose students to the practical challenges associated with building and testing single-borrower credit risk models, such as those used by banks, as well as to the types of solutions that can be used to build them; and to give students credible and realistic experience in solving these in real-world settings with large data sets. When students complete this course, they should achieve solid foundation in some of the challenges (and potential solutions) in developing data-driven default models. They will also leave with a toolkit of a number of useful modules that can be applied in practical commercial settings.

Major project: There is one major assignments required for successful completion of this course.

• The Default Modeling Project. The objective is to develop, individually or in small teams, a default prediction model using a realistic data set. This is not a programming project, though students will find it useful to use the programming language R to estimate the model. Much of the basic command-line code that is required for a basic model is given in the attached project description, along with examples. Successful completion of the project will include presenting and documenting the model in a realistic setting.
Grading:

<table>
<thead>
<tr>
<th>Component</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modeling project:</td>
<td>50%</td>
</tr>
<tr>
<td>Formulation:</td>
<td>15%</td>
</tr>
<tr>
<td>Out of sample performance:</td>
<td>15%</td>
</tr>
<tr>
<td>Documentation:</td>
<td>20%</td>
</tr>
<tr>
<td>Mid-term:</td>
<td>20%</td>
</tr>
<tr>
<td>Final exam:</td>
<td>30%</td>
</tr>
<tr>
<td>Extra credit for best model:</td>
<td>10%/n, n ≤ 3 ≡ the number of team members</td>
</tr>
</tbody>
</table>

Homework: Aside from completing the readings, the only formal homework for this course is the work associated with completing the main modeling project and mini-projects. It is strongly advised that you get started on the main modeling project early as it has a number of components. We will go over some of the mini-projects in class, but they will not be graded.

Attendance: No points are given for attendance. Some of the material on the mid-term and final exams will be drawn from the lectures and discussion. However, students missing more than three class sessions without permission of the professor will lose 10 points on their final course grade.

Primary Text: Bohn, J. R. and R. M. Stein, (2009) *Active Credit Portfolio Management in Practice*, NY, Wiley. (ACPMIP). It is important that you read the assigned material as portions will be covered on the final exam.

Additional reading: 1-2 papers per course session may be assigned (see syllabus). Copies of lecture slides will not be distributed, although key slides will be distributed to reduce note-taking load.
Draft Syllabus
Data-driven Credit Models

Week 1

Introduction to credit risk modeling concepts
- Non-normal return distributions
- Data problems and resolutions
- Key components of credit risk – PD, LGD, (EAD), correlation, size
- Differing modeling paradigms
- Diversification

ACPMIP: Chapter 1, pp. 2-16; 19-23; 32-34; 38; 42-43. Chapter 2, pp. 60-62; 72-74.

Supplemental readings:
- Dhar, V. and R. Stein (1997), Seven Methods for Transforming Corporate Data into Business Intelligence, Prentice Hall, NJ. Chapter 3.

Week 2

R tutorial and data sets
- The R language
- R studio
- Basic operations
- Objects
- Vectorization
- Packages
- Data sets
- Sample modeling session

Mini-project: Function to calculate a simple moving average. Write an R function to calculate the simple moving average of a time series. The function definition should be

Definition: \[ \text{ma} \left( x, k \right) \]

where
- \( x \) is a vector for which to calculate moving averages and
- \( k \) is the number of periods over which to calculate the moving average (including the current period).

Return: The function should return a vector of length \( \text{length}(x) \) with the current period’s \( k \)-period moving average in place of each original data point. The beginning of the moving average vector will be padded with NAs.

Tasks:
1. Implement this in two different ways:
   a. Using a loop
   b. Using vectorization.
2. Use the \text{sys.time()} function to time the two approaches.
Week 3

PD model validation – Part I
- Validating model power
- Validating model calibration

ACPMIP: Chapter 7, pp. 361-397.

Supplemental readings:

Mini-project: Function to calculate the AUC ROC for two different subsets of a single data set.

- **Definition:** `subROC<-function(x, split.val, split.on, score, outcome,...)` where
  - `x` is a dataframe
  - `split.val` is a scalar, factor value, date or string used to divide the data frame
  - `split.on` is a vector of length `nrow(x)` of the same type as `split.val`; `split.on <= split.val` goes to one data subset while the remainder goes to the other
  - `score` is a numerical vector of length `nrow(x)` for calculating the ROC AUC
  - `outcome` is a binary numerical vector of length `nrow(x)` for calculating the ROC AUC
  - ... additional parameters

- **Return:** The function should return a list with three elements:
  - A vector of length 2 with the ROC AUC for each data subset.
  - A vector of length `nrow(x.subset1)` giving the indices of `x` that for the records included in `x.subset1`
  - A vector of length `nrow(x.subset2)` giving the indices of `x` that for the records included in `x.subset2`

- **Tasks:**
  - Implement `subROC`
  - Describe how you would make `subROC` more general so that it could take in an arbitrary one or two variable statistic (function) as an input and return the appropriate data

Week 4

Regression-based models of default and data preprocessing
- Discrete choice models
- Survival models
- Case Study - PDs for firms with public information: the RiskCalc models of private firm default

ACPMIP: Chapter 4, pp. 183-215, 238-252.
**PD model calibration**
- Calibrating to empirical data
- Adjusting for differing baseline default rates
- Mapping between ratings and PDs and back again

**ACPMIP**: Chapter 4, pp. 215-233.

**Supplemental readings**:

**Mini-project**:
- Function to create a calibration curve mapping a variable to a default rate.
- Function to use the calibration curve to map a variable to a default rate (including interpolation).

**Definition**: estimateCalibCurve <- function(x, outcome, k, ...) where
- x is a vector of model scores
- outcome is a binary numerical vector of length length(x)
- k is a scalar, denoting the number of “buckets” to use in the mapping
- split.on is a vector of length nrow(x) of the same type as split.val; split.on <= split.val goes to one data
- ... additional parameters

**Return**: The function should return a list with three elements:
- A list containing
  - map a dataframe of length k with two columns
    - the cutoff (on the same scale as x)
    - the mapped PD corresponding to the cutoff
  - baseline a scalar containing the baseline PD for outcome

**Definition**: applyCalibCurve <- function(x, map, values, baseline=NULL, ...) where
- x is a vector of model scores
- map is a dataframe returned in map by buildCalibCurve
- baseline is a scalar, to be used if baseline adjustment is to be applied after mapping
- split.on is a vector of length nrow(x) of the same type as split.val; split.on <= split.val goes to one data
- ... additional parameters

**Return**: The function should return a vector of length length(x) containing the mapped PD for each element of x.

**Tasks**:
- Implement estimateCalibCurve
- Implement applyCalibCurve
- Test on estimateCalibCurve and applyCalibCurve on Week 5 data set

---

**Week 6**

**Mid-term**

**Week 7**

**Tree-based models**
- CART
- RandomForests

**Readings**:
Week 8

PD model validation – Part II
• Calculating confidence bounds
• Noisy data
• Walk-forward analysis
• What is a more powerful model worth?

ACPMIP: Chapter 7 pp. 396-437

Mini-project: Walk-forward engine
Write an R function to implement the walk-forward approach to estimating a linear regression model.

- Definition: The function definition should be \( \text{wf} \leftarrow \text{function}(x,k,\text{time}.\text{idx},\text{formula}) \) where
  - \( x \) is a dataframe
  - \( k \) is the minimum value of the time to include in the analysis
  - \( \text{time}.\text{idx} \) is the index of the column in which the timestamp for the record is located
  - \( \text{formula} \) is the formula for the regression

- Return: The function should return a list with three elements:
  - A list containing:
    - \( \text{models} \) list of \( \text{lm} \) objects, one for each out-of-sample (walk-forward) period
      (you may wish to turn-off storage of original data frames in \( \text{lm} \) return)
    - \( \text{pred} \) A data frame of length \( nrow(x[x[\text{time}.\text{idx}>k,]) \) and width 2 containing one prediction for each record in each out of sample period (all periods concatenated)

- Tasks:
  - Implement \( \text{wf} \)
  - Test \( \text{wf} \) on Week8 data set

Week 9

Loss Given Default: theory, data acquisition and modeling
• The default and resolution process
• Definitions and measures of LGD
• What makes estimating LGD hard (when everyone used to think it was easy)

ACPMIP: Chapter 5;

Supplemental reading:
• Van de Castle, K., D. Keisman and R. Yang “Suddenly Structure Mattered: Insights into Recoveries from Defaulted Debt.”

Guest lecturer: David Keisman, Moody’s

Week 10

Mortgages
• Mortgage structures
• Mortgage dynamics
• Hazard rate models
• One model or many?

Supplemental reading:

<table>
<thead>
<tr>
<th>Week 11</th>
<th>Presentation of Student Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Week 12</td>
<td>Wrap-up and review</td>
</tr>
<tr>
<td>Week 13</td>
<td>Final Exam</td>
</tr>
</tbody>
</table>
Credit Default Model Project

**Objective:** You are being asked to estimate and test a simple model of corporate default. The model should take as input financial statement data on each of the firms being evaluated, and produce as output a one-year probability of default (and any ancillary measures you wish to include) for each firm. You will be provided a development data set and you may use either R or Matlab to estimate and test the model. *It is strongly recommended that you use R.* The model will be tested on a holdout sample that the instructor maintains. You may work individually or in small groups.

**Deliverables:** In completion of this assignment you will be expected to turn in

1. A PowerPoint or Keynote deck (10-15 slides) describing:
   a. your data
   b. the definitions of the variables you included
   c. the relative importance of the variables in the model
   d. the functional form of the models you considered
   e. any data preprocessing you performed
   f. the details of your final model
   g. your testing results, and
   h. a technical appendix (if needed);

2. The source code you used to estimate and test your model;

3. Source code that takes as input a data file of the same format as the development sample and produces outputs in the form of probabilities of default for each firm in the holdout sample

4. A file containing PDs for a the validation data (holdout sample) you will be given

**Software:** You may use either the R language or Matlab to estimate and test your model. *It is recommended (strongly) that you use R,* as much of the statistical support for estimation and testing of R models is freely available in Open Source form and because the examples that follow are implemented in R. Some useful links are given below:

   R software downloads: [http://cran.us.r-project.org](http://cran.us.r-project.org)

(You will also find a large repository of statistical routines including the caTools package for ROC analysis.)


This is an integrated development environment, currently also free, which makes loading data, installing packages and overall development generally easier than in native R. I recommend you try this out as I have found it streamlines the model estimation process.

**Data:** You will be provided a data set (the development data set) containing abbreviated financial statement data and financial ratios for public companies. A similar validation data set (the holdout data set) will be used to test your models, though you will not have access to this data. The holdout sample may contain future financial statements for the firms in the sample as well as financial statements for additional firms not in the your data set. This development data set is relatively large. You may wish to sample the data set down for initial experiments before using larger portions for estimation of your final models. It is also strongly recommended that you split your sample into both an estimation and a testing sample to allow you to evaluate the robustness of your model before you submit it.

**Testing:** After you finish your model, you will be given a new data set, in the same format as the first one, but with no default flags. You will use your model to produce PDs for each record in the data set and to then submit this for grading.

**Important dates:**

Week 3: Working groups due

Week 9: Final PowerPoint deck and source code due

Week 12: Presentation of selected student models
Getting started on the modeling project:

A simple (naive) PD model

The following is an example session, with comments, to outline how one might estimate a simple logit default
model in R. The goal is only to show mechanics, so I have not done any data cleaning or model diagnostics and
I’ve made some very naive assumptions about the variables.

Let’s start by seeing how large the data set is:

```r
> n<-nrow(pubfirm.in)
> n
[1] 92490
```

There are about 93K firm years, so this is a moderately sized data set. However, we will be modeling corporate
default and to do this, we need to know the number and timing of defaults. We will then need to associate each
default event with the appropriate record in the financial statement data we will be using, for whatever default
horizon we are modeling:

```r
> days.to.default<-pubfirm.in$defaultdate - pubfirm.in$datadate
```

Since each default date is associated with all records for a given firm, for a one-year horizon can (naively\(^1\)) de-
dine those default events that happened within one year of a given financial statement date:

```r
> default.in.cur.year<-days.to.default >0 & days.to.default <= 365.25
```

Finally, we define a one-year default flag:
- \(1\) (true) if the default date occurs within one year of the financial statement reporting date (datadate).
- \(0\) if the default date does not fall in this range:

```r
> pubfirm.in$default.flag<- ifelse(default.in.cur.year,1,0)
> summary(pubfirm.in$default.flag)
```

<table>
<thead>
<tr>
<th></th>
<th>Min.</th>
<th>1st Qu.</th>
<th>Median</th>
<th>Mean</th>
<th>3rd Qu.</th>
<th>Max.</th>
<th>NA's</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.04</td>
<td>0.00</td>
<td>1.00</td>
<td>76307</td>
</tr>
</tbody>
</table>

There are a (large) number of of NAs (firms that never defaulted in our dataset do not have a default date). We
need to clean these up:

```r
> pubfirm.in$default.flag<-ifelse(is.na(pubfirm.in$default.flag),0,pubfirm.in$default.flag)
> summary(pubfirm.in$default.flag)
```

<table>
<thead>
<tr>
<th></th>
<th>Min.</th>
<th>1st Qu.</th>
<th>Median</th>
<th>Mean</th>
<th>3rd Qu.</th>
<th>Max.</th>
<th>0.000000</th>
<th>0.000000</th>
<th>0.000000</th>
<th>0.000000</th>
<th>1.000000</th>
<th>NA's</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>0.04</td>
<td>0.00</td>
<td>0.00</td>
<td>1.00</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1.000000</td>
<td>0.000000</td>
</tr>
<tr>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>0.04</td>
<td>0.00</td>
<td>0.00</td>
<td>1.00</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1.000000</td>
<td>0.000000</td>
</tr>
</tbody>
</table>

(There are 617 default records in the data set.)

Note how the inclusion of the “NA” records provides a richer and less biased data set, and also gives a better
estimate of the default rate in the database.

We will find it useful to create new variables, such as financial ratios for our models:

\(^1\) See Chapter 4 of Bohn and Stein (2009).
> pubfirm.in$tl.ta<-pubfirm.in$lt/(pubfirm.in$at+1) #total liabilities/total assets,a form of leverage
> pubfirm.in$size<-log(pubfirm.in$at+1)
> pubfirm.in$wc.ta<-pubfirm.in$wcap/(pubfirm.in$at+1)#total working cap/total assets,a form of liquidity

(Note that we added 1 in the denominator and log, respectively, to allow for cases in which total assets is zero.)

> summary(pubfirm.in$wc.ta)
  Min.  1st Qu.   Median     Mean  3rd Qu.     Max.     NA's
-930.400    0.018    0.155    0.142    0.319    0.995     6927

> summary(pubfirm.in$size)
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
0.000   4.962   6.285   6.289   7.626 13.080    3249

(Here we would normally examine the new ratios both graphically and statistically to determine whether to perform additional transformations and to make sure there were no counter-intuitive features.)

The liquidity look like it has some extreme values. We may need to do some work on the data to understand why and figure out what to do. For purposes of this quick example, we will apply a naïve fix-up estimating the model. But before we do, we need to segment our data into a training and testing sample, if possible. To create a fair test, we need to only use data from our training sample in any statistical manipulation.

> set.seed(1) # I included this to ensure that random number generates same results as in this example
> training.sample.filter<-runif(n)<0.70  # create a filter to include approximately 70% of the sample

(Instead you may wish to subset the data differently – say by time – to increase the chance of a robust model out-of-sample).

> traindat<-pubfirm.in[training.sample.filter,]
> nrow(traindat)
[1] 64925
> sum(traindat$default.flag, na.rm=T)
[1] 413

We now return to the problem of the outliers. We will use a simple (and costly!) technique to curtail extreme values. We will set any values of the variables outside the (arbitrary) range [0, 20] to the appropriate endpoint of this range:

> testdat$tl.ta.adj<- ifelse(testdat$tl.ta <0,
                                 0,
                                 ifelse(testdat$tl.ta > 20,
                                         20,
                                         testdat$tl.ta))

(Note that I wrote the ifelse calls in an unusual way to highlight how the command works.)

> summary(pubfirm.in$tl.ta)
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
0.000  0.432   0.572   0.614   0.689 930.400    5146

We will set any values of liquidity less than 0 to 0:

> testdat$wc.ta.adj<- ifelse(testdat$wc.ta <0,0, testdat$wc.ta)

> summary(testdat$wc.ta.adj)
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
0.000  0.292   0.538   0.617   0.788 40.000   3758

---

2 To create a fair test, we need to only use data from our training sample in any statistical manipulation.

3 Why is this costly?
If we were calculating parameters for preprocessing (e.g., percentiles for Winsorization or density estimates for univariate transforms) rather than just using constants as above, robust testing would require us to estimate these parameters on the training data only, and to then use the training sample parameters to test both the training and testing models.

We are now “ready” to estimate our first model. Let’s try looking at a logit model that produces a PD using two variables: size and liquidity \((\text{wc.ta.adj})\).

As a first pass, we might expect the coefficient on size to be negative (small firms are more susceptible to economic shocks) and the coefficient on liquidity also to be negative (higher liquidity implies lower default probability).

We estimate the model in the form of a \textit{generalized additive model} with a logit linking function:

\[
> \text{my.first.model} <- \text{glm(default.flag~size+wc.ta.adj, family=binomial(link="logit"),} \\
> \text{data=traindat, na.action=na.exclude)}
\]

(Note the use of \texttt{na.action=na.exclude} which allows the routine to run by returning “NA” for those rows with missing or “NA” observations rather than dropping them.)

This seemed to work without generating errors, so let’s take a look at the model:

\[
> \text{summary(my.first.model)}
\]

The signs on the coefficients look like they are in the expected directions. We might next want to see standard diagnostic plots of the regression:

\[
> \text{plot(my.first.model)}
\]

[The plots are not shown here, but would appear on the screen. They suggest a more work needs to be done on this model and with the data we are using.]
Also recall that in R, models are actually objects that can be manipulated. So, for example, the model we just estimated has a number of components that we could examine:

```r
> names(my.first.model)
```

(For more detail on these components, you can type “? glm” at the command line.)

We can summarize (e.g.) the range of fitted values (PDs) by examining `fitted.values` in our model `my.first.model`:

```r
> summary(my.first.model$fitted.values)
```

Using the `round` function makes these values a bit easier to read.

```r
> round(summary(my.first.model$fitted.values),5)
```

Note that our model produces a wide range of PD estimates, from less than a basis point to more than 2.25%.

In our work, we will often find it useful to compare model output for defaulted and non-defaulted firms.

```r
> n.ndef<-sum(nondefault.records, na.rm=T)
> n.def<-sum(default.records,na.rm=T)
> n.ndef
[1] 59410
> n.def
[1] 387
```

(Note that we have lost some defaults due to missing values in the independent variables. The smaller the number of defaults the harder it will be to determine how robust the model is. How might we address this?)

A number of the tests we will want to perform depend on our being able to separate the estimated PDs of defaulted firms from the estimated PDs of non-defaulted firms:

```r
> nndef.PDs<- my.first.model$fitted.values[nondefault.records]  #model estimates for non-defaulted firms
> def.PDs<- my.first.model$fitted.values[default.records]      #model estimates for defaulted firms
```

Now, using our newly filtered PDs, we might plot histograms of the fitted values by default status (Not shown here is that I played around with the parameters to `hist` in order to get a reasonable set of plots. Note also that I plotted the log PDs rather than the raw for readability.):

```r
> par(mfrow=c(2,1)) # set up to plot two rows by one column
> hist(log(def.PDs),xlim=c(-9.5,-3.5),col="red",nclass=21)
> hist(log(nndef.PDs),xlim=c(-9.5,-3.5),col="red",nclass=21)
```
The histograms are suggestive, but to get a more quantitative sense of the *discriminatory power* of the model, we can use the Wilcoxon test as a quick approximation to the area under the ROC\(^4\):

\[
W \leftarrow \text{wilcox.test}(x = \text{def.PDs}, y = \text{ndef.PDs}, \text{paired} = \text{FALSE})
\]

\[
\text{aprox.AUC} \leftarrow W$\text{statistic}/(n.ndef \times n.\text{def})
\]

\[
\text{aprox.AUC}
\]

\[
W
\]

0.6961734

This is somewhat encouraging (a \(W\) of 0.5 is random while 1 is perfect). To test the model out of sample (but not out of time)\(^5\) we predict default for the data we excluded from \texttt{traindat} (we omit the steps to set up the data set, though it is the same as we did for \texttt{traindat}). To keep things simple, we will create a “clean” test set containing only valid records:

\[
\text{valid.records} \leftarrow !\text{is.na(testdat$size)} \& !\text{is.na(testdat$wc.ta.adj)}
\]

\[
\text{testdat.clean} \leftarrow \text{testdat}[\text{valid.records},]
\]

We then use our model to predict on this clean test sample:

\[
\text{out.pred} \leftarrow \text{predict(\text{my.first.model}, \text{newdat = testdat.clean}, \text{type = "response"}, \text{na.action=na.pass})} # \text{generate a}
\]

…and calculate the performance:

\[
\text{default.records.o} \leftarrow \text{testdat.clean$default.flag} = 1\quad \# \text{actual defaults}
\]

\[
\text{n.ndef.o} \leftarrow \text{sum(\text{nondefault.records.o}, \text{na.rm=T})}
\]

\[
\text{n.\text{def}.o} \leftarrow \text{sum(\text{default.records.o}, \text{na.rm=T})}
\]

\[
\text{ndef.PDs.o} \leftarrow \text{out.pred[nondefault.records.o]}\quad \# \text{model estimates for non-defaulted firms}
\]

\[
\text{def.PDs.o} \leftarrow \text{out.pred[default.records.o]}\quad \# \text{model estimates for defaulted firms}
\]

\[
W.o \leftarrow \text{wilcox.test}(x = \text{def.PDs.o}, y = \text{ndef.PDs.o}, \text{paired} = \text{FALSE})
\]

\[
\text{aprox.AUC.o} \leftarrow W$\text{statistic}/(n.ndef.o \times n.\text{def})
\]

\[
W
\]

0.6857898

\(^4\) See Chapter 7 of Bohn and Stein (2009).

\(^5\) \textit{Ibid}
It looks like our model is fairly stable out of sample (though it would be useful to know how it would perform in a different time period). However, before we get very excited, we should examine how a simple univariate model (e.g., a single ratio) might do on this data.

```
> ndef.liq.o<- testdat.clean$wc.ta.adj[nondefault.records.o]  #liquidity for non-defaulted firms
> def.liq.o<- testdat.clean$wc.ta.adj[default.records.o]     #liquidity for nodefaulted firms
> W.liq<-wilcox.test(x=-def.liq.o,y=-ndef.liq.o,paired=FALSE)
> aprox.AUC.liq<- W.liq$statistic/(n.ndef.o*n.def.o)
> aprox.AUC.liq
0.6778404
```

(Note the negative sign in front of `wc.ta.adj` since higher values imply lower default probability.)

Clearly this result is disappointing: our model is only slightly better than a univariate model. (NB: It is not clear from inspection that this difference is statistically significant. Indeed, testing suggests that it is not\(^6\).)

For larger data sets, or if we wish to plot an ROC curve, the `caTools` package has a function called `colAUC` that generates one or more ROC curves (one AUC for each column in the data frame) and then integrates the area under them numerically. For moderate sized data sets, the results are generally identical, or almost so, to the Wilcoxon test:

```
> library("caTools", lib.loc="/Library/Frameworks/R.framework/Versions/3.1/Resources/library")
   #load the caTools package (your path may be different depending on how R was installed)
> colAUC(data.frame(out.pred), testdat.clean$default.flag, alg=c("ROC"))
out.pred
 0 vs. 1 0.6857898
```

We may also use this routine compare our two models visually:

```
> colAUC(data.frame(model=out.pred, liquidity=testdat.clean$wc.ta.adj), testdat.clean$default.flag,
plotROC=TRUE, alg=c("ROC"))
  model liquidity
  0 vs. 1 0.6857898 0.6778404
```

\(^6\) ibid