University California Davis
Least Squares
MAT-167
December 19,2020

Prithika Rajendran
William Shih

Contents

1. Introduction

Bike sharing rental systems has become increasingly popular throughout the years. Especially with the increasing amounts of bike rental programs that are being offered for people to look for different means of transportation. As these bike rental programs increased so did the the interest in learning about their important role in traffic, environmental and health issues. The goal of this data set it to do a prediction of bike rental count hourly based on the environmental and seasonal settings.

2. Background

The dataset for this project is the "Bike Sharing Dataset Data Set" found in the UCI Machine Learning Repository. The dataset contains hourly count of rental bikes for all of 2011 and 2012 (January 1, 2011 to December 31, 2012) in the Capital Bikeshare System of Washington D.C. area (Washington-Arlington-Alexandria, DC-VA-MD-WV metropolitan area). The UCI Machine Learning Repository cites Hadi Fanaee-T from the "Laboratory of Artificial Intelligence and Decision Support (LIAAD), University of Porto" for the compilation of the data.

The dataset is outdated since data is actually available up to November 2020 on Capital Bikeshare's website (as of December 18, 2020), but this limited dataset will still work for the purposes of demonstrating linear algebra on a real world dataset.

There are two files included in the dataset: a `hour.csv` and a `day.csv`. We will use the `hour.csv` for the regression, since the `day.csv` is simply just a sumamry of the `hour.csv` file. We also made a function that easily converts the `hour.csv` to the `day.csv` called `convert_hour_to_day()`.

There are 14 different variables that are in this dataset that are potentially of interest. Two variables are not useful and immediately thrown out: `instant` (this is simply the row number of the dataset) and `dteday` (date of the year).

2.1 Dataset

Denote x_n as plausible independent variables and denote y_n as plausible dependent variables.

x_1: `season` (1: spring, 2: summer, 3: fall, 4: winter)

x_2: `yr` (0: 2011, 1: 2012)

x_3: `mnth` (1 to 12)

x_4: `hour` (0 to 23)

x_5: `holiday` (whether a holiday (0 or 1) from this list of holidays)

x_6: `weekday` (0 to 6)

x_7: `workingday` (1 if weekday and not holiday, 0 otherwise)

x_8: `weathersit`: Weather conditions (1: Clear, Few clouds, Partly cloudy, Partly cloudy, 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist, 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds, 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog)

x_9: `temp` (0-1, normalized temperature in Celsius. Divided by 41)

x_{10}: `atemp` (0-1, normalized "feels like" temperature in Celsius. Divided by 50)

x_{11}: `hum` (percent humidity)

x_{12}: `windspeed` (0-1, Normalized wind speed. Divided by 67)

y_1: `casual` (count of casual users)

y_2: `registered` (count of registered users)

y_3: `cnt` (count of sum of casual and registered users)

The following least squares regression exercise will try to predict the `casual`, `registered`, or `cnt` as a function of the independent variables. Also, we will try some principal components analysis (PCA) and k-nearest neighbors (kNN) with these variables.

2.2 Data Analysis

Preliminary data analysis shows that the `hour` is by far the most important independent variable for explaining the variation in the dependent variables. Thus, it is important to know how exactly the `hour` variable interacts with `registered`, `casual`, and `cnt`.

We also found that it makes sense to treat `hour` as a categorical variable (treat `hour` as 23 independent dummy variables (0 or 1 for each variable), one for each hour minus the constant term), but in this case, we will try to fit `hour` in terms of a polynomial curve to demonstrate polynomial fitting with linear algebra.

A plot of `hour` on the x-axis and `registered` or `casual` on the y-axis would us some insight of what degree polynomial for `hour` we should be looking for.

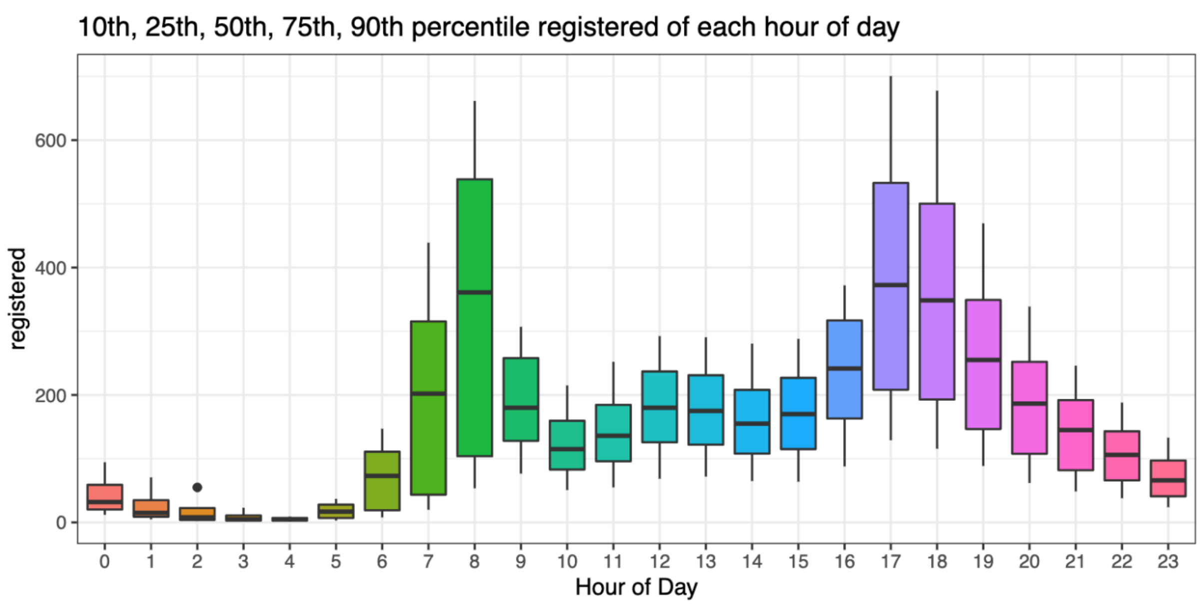

Figure 1: Boxplot of registered users for Jan 1, 2011 to December 31, 2011 for each hour of day. The low whisker represents 10th percentile, the box represents the interquartile range, the top whisker represents the 90th percentile. For example, with 731 days in the dataset, the low whisker represents the 73rd lowest value.

Figure 1 for the number of registered users show a trimodal distribution with three peaks throughout the day. A possible interpretation of the three peaks is that there is an early peak for the morning commute, a central peak for lunchtime, and a late peak for the evening commute. This suggests that we need a high-degree polynomial to accurately the number of registered users throughout the day. A six-degree polynomial is the minimum degree that can represent a trimodal distribution.

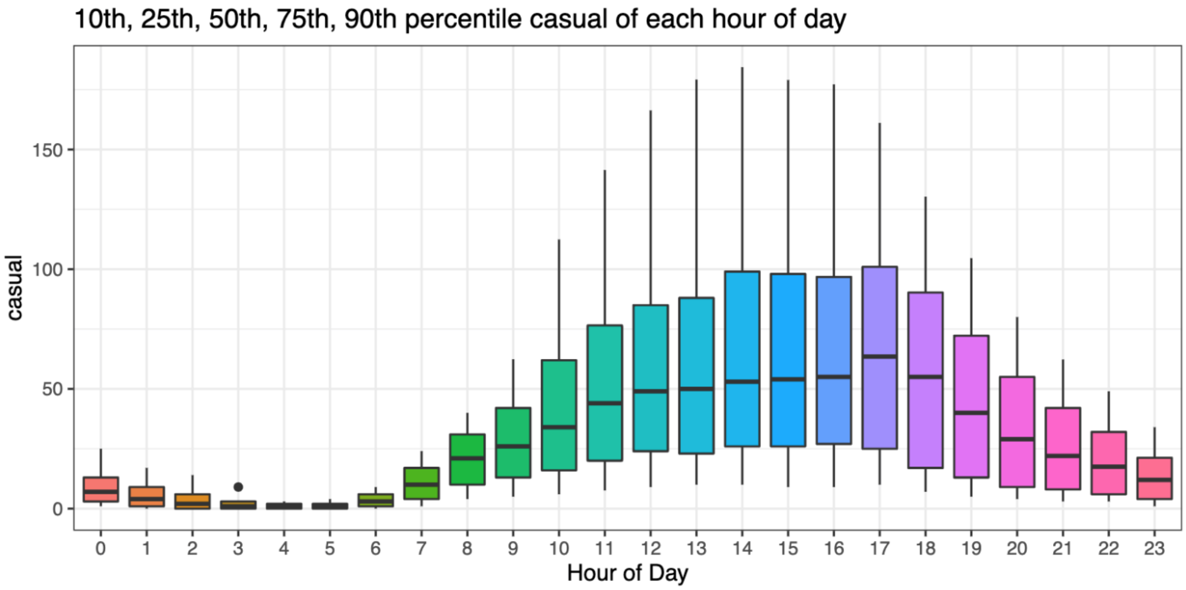

Figure 2: Boxplot of casual users for Jan 1, 2011 to December 31, 2011 for each hour of day. The low whisker represents 10th percentile, the box represents the interquartile range, the top whisker represents the 90th percentile. For example, with 731 days in the dataset, the low whisker represents the 73rd lowest value.

Figure 2 for the number of casual users show a single peak distribution. A possible reason for this is that casual users are tourists that don't commute. Tourists also don't wake up early in the morning and most things to do for tourists occur in the afternoon and evening. Thus, the peak occurs at around 12 PM-6 PM. A two-degree polynomial is potentially sufficient to represent the number of casual users throughout the day.

Figure 1 and **Figure 2** show significant variation so that it is clear that the hour of the day is not the only variable influencing how many users there are for this bike sharing system. We can plot at the number of users for each day for further information.

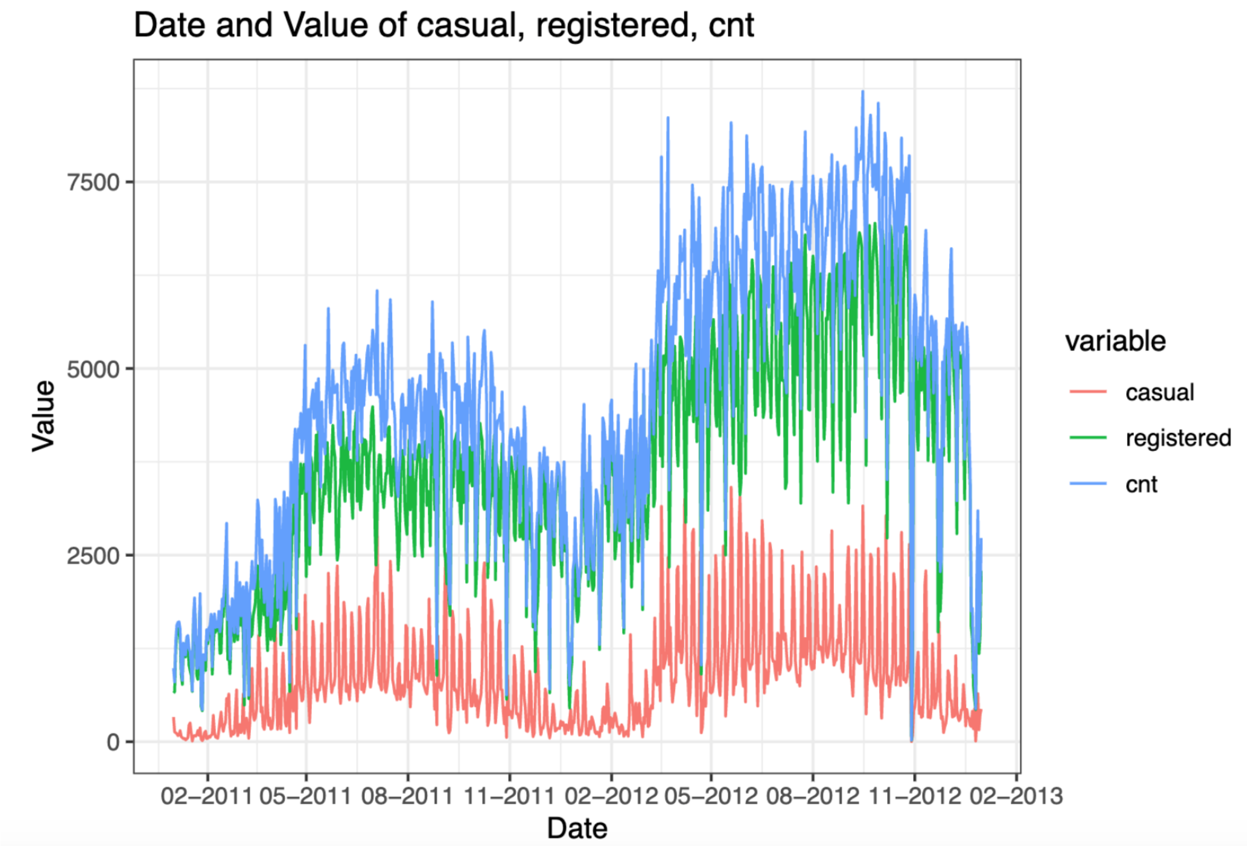

Figure 3: Number of casual, registered, and sum of casual and registered for all 731 days in the dataset

Figure 3 Shows both variation through the year and an increase in number of users from 2011 to 2012. This makes sense if this Capital Bike-Sharing was still a developing system in 2011 and not yet a mature system where the market is already saturated. Thus, we want to include a variable in our least squares regression model that includes controls for seasonal variation and the year. This would be `season` and `yr` from the list of x_n. It turns out that the `weekday` (day of the week) and `mnth` (month of the year) do not explain a large additional amount of variation, so they won't be included in the model.

But **Figure 3** still shows quite a bit of variation day to day within each season. There are quite a few weather related variables in the list of independent variables in the dataset, which would account for some of the remaining variation.

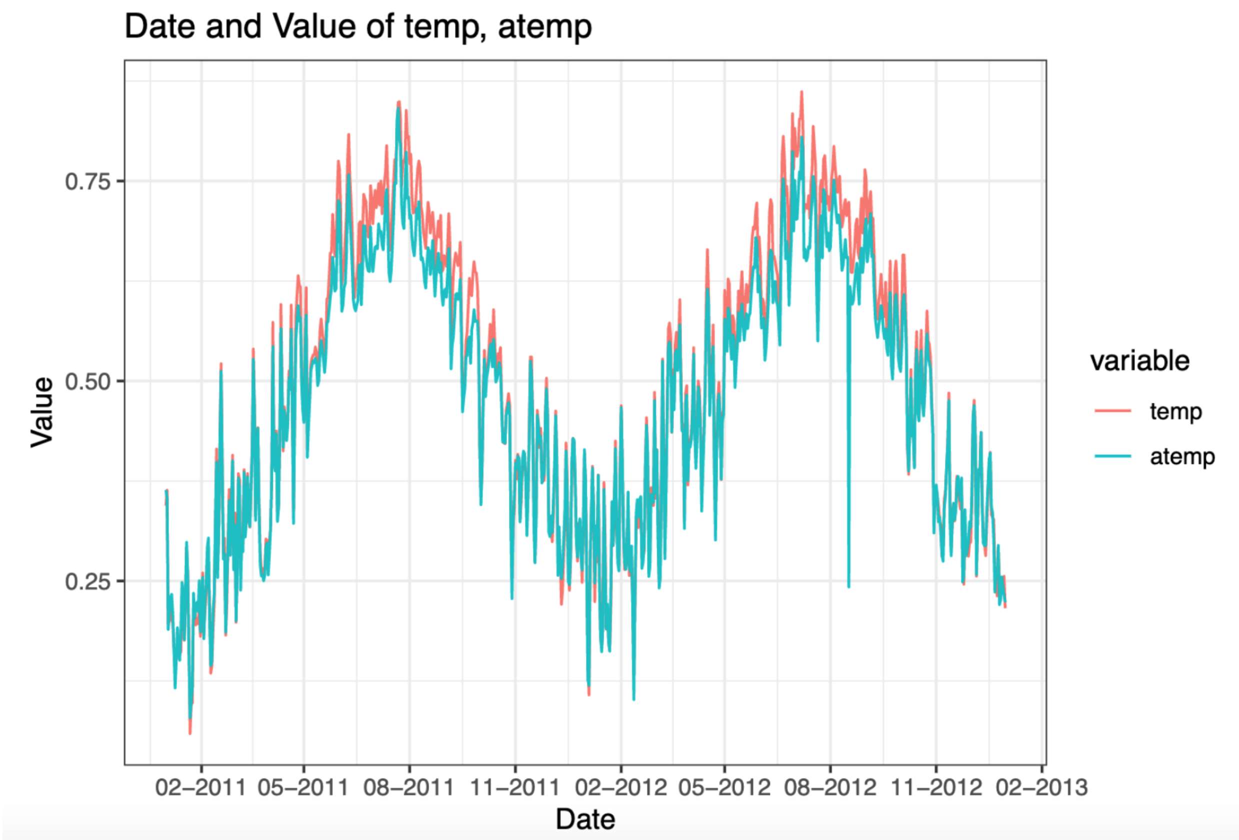

Figure 4: Normalized temperature and “feels like” temperature for all 731 days in the dataset

Figure 4 shows the average `temp` (actual temperature) and `atemp` (“feels like” temperature) for each day of the year. A comparison of **Figure 3** and **Figure 4** shows that temperature shows a strong negative relationship with the number of bike-sharing users, which makes sense. People do not want to bike when it is cold outside. These variables have a very high correlation with each other such that we found it to be sufficient to just add `temp`. In fact, `temp` is good enough to explain most of the weather-related variation and other variables such as `wind` (wind speed) and `hum` (humidity) are not necessary.

Thus, we choose just `season`, `yr`, `hr`, and `temp` as the independent variables and omit the rest of the variables for the least squares regression model. These are x_1, x_2, x_4, and x_9.

3. Design Matrix

The design matrix A is the matrix that will be used to solve the equation $Ax = b$, where x are the coefficients, often referred to as the β's and b is the dependent variable. It is the matrix of the explanatory/independent variables.

The design matrix A will be of $\mathbb{R}^{17379\times(n+4)}$, (where n is the maximum degree of the polynomial for x_4 (`hour`)) since there are 17,379 rows (one for each hour in the dataset) and there are four variables other than the `hour` (x_4) variable. The other four variables are the constant term, x_1 (`season`), x_2 (`yr`), and x_9 (`temp`).

The following will be the form of our design matrix $A \in \mathbb{R}^{17379\times(n+4)}$ (Note that m is left the matrix for simplification, but the of $m = 17379$):

$$A = \begin{bmatrix} 1 & x_{11} & x_{12} & x_{14} & x_{14}^2 & \dots & x_{14}^n & x_{19} \\ 1 & x_{21} & x_{22} & x_{24} & x_{24}^2 & \dots & x_{24}^n & x_{29} \\ 1 & x_{31} & x_{32} & x_{34} & x_{34}^2 & \dots & x_{34}^n & x_{39} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & x_{m1} & x_{m2} & x_{m4} & x_{m4}^2 & \dots & x_{m4}^n & x_{m9} \end{bmatrix}$$

The first value of the index represents the row number and the second value of the index represents the variable number in the matrix. For example x_{32} in the above means the 3rd row of variable x_2.

Now the question remaining is the appropriate value of n. We know that the answer depends on which dependent variable we are using. If b is `registered` or `cnt`, the value of n should be higher than if b is `casual`. One way to find a good for n is calculate the R-squared for each value of n. The R-squared is the proportion of variation in the dependent variable that can be explained by the independent variables. A value of n where $n + 1$ does not increase the R-squared significantly further would be a good value of n.

Table 1: R-squared values for linear regression of each of value of n, maximum power of polynomial for `hour`, including x_1, x_2, and x_9

	R^2_{cnt}	R^2_{casual}	$R^2_{registered}$
x_4	0.343	0.285	0.291
$x_4 + x_4^2$	0.464	0.371	0.394
$x_4 + x_4^2 + x_4^3$	0.515	0.430	0.431
$x_4 + x_4^2 + \cdots + x_4^4$	0.516	0.440	0.436
$x_4 + x_4^2 + \cdots + x_4^5$	0.530	0.446	0.464
$x_4 + x_4^2 + \cdots + x_4^6$	0.552	0.446	0.497
$x_4 + x_4^2 + \cdots + x_4^7$	0.589	0.446	0.551
$x_4 + x_4^2 + \cdots + x_4^8$	0.599	0.446	0.565
$x_4 + x_4^2 + \cdots + x_4^9$	0.606	0.447	0.575
$x_4 + x_4^2 + \cdots + x_4^{10}$	0.606	0.447	0.575
$x_4 + x_4^2 + \cdots + x_4^{11}$	0.613	0.447	0.585
$x_4 + x_4^2 + \cdots + x_4^{12}$	0.614	0.447	0.585
$x_4 + x_4^2 + \cdots + x_4^{13}$	0.638	0.448	0.619
$x_4 + x_4^2 + \cdots + x_4^{14}$	0.638	0.448	0.619
$x_4 + x_4^2 + \cdots + x_4^{15}$	0.638	0.448	0.619

Table 1 shows the values of R-squared regressed upon the dependent variables `cnt`, `casual`, and `registered` for each value of n for values of 1 to 15. For `cnt` and `registered`, a good value of n appears to be 7. $n = 7$ gives a huge increase in R-squared over $n = 6$ for `cnt` and `registered`. Only small increases of R-squared are seen for values of n above 7 for `cnt` and `registered`. Although these increases are still statistically significant, it is not a good idea to have very high order polynomials as such a regression is likely to overfit the data. For example, very high order polynomials may give poor predictions for data in 2013 or beyond.

For `casual`, a good value of n appears to be 3. Only small increases of R-squared are seen for values of n above 3 for `casual`.

Therefore, the design matrix if b is `cnt` or `registered` is $A \in \mathbb{R}^{17379 \times 11}$, where $m = 17379$:

$$A = \begin{bmatrix} 1 & x_{11} & x_{12} & x_{14} & x_{14}^2 & x_{14}^3 & x_{14}^4 & x_{14}^5 & x_{14}^6 & x_{14}^7 & x_{19} \\ 1 & x_{21} & x_{22} & x_{24} & x_{24}^2 & x_{24}^3 & x_{24}^4 & x_{24}^5 & x_{24}^6 & x_{24}^7 & x_{29} \\ 1 & x_{31} & x_{32} & x_{34} & x_{34}^2 & x_{34}^3 & x_{34}^4 & x_{34}^5 & x_{34}^6 & x_{34}^7 & x_{39} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{m1} & x_{m2} & x_{m4} & x_{m4}^2 & x_{m4}^3 & x_{m4}^4 & x_{m4}^5 & x_{m4}^6 & x_{m4}^7 & x_{m9} \end{bmatrix}$$

The design matrix if b is `casual` is $A \in \mathbb{R}^{17379 \times 7}$, where $m = 17379$:

$$A = \begin{bmatrix} 1 & x_{11} & x_{12} & x_{14} & x_{14}^2 & x_{14}^3 & x_{19} \\ 1 & x_{21} & x_{22} & x_{24} & x_{24}^2 & x_{24}^3 & x_{29} \\ 1 & x_{31} & x_{32} & x_{34} & x_{34}^2 & x_{34}^3 & x_{39} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{m1} & x_{m2} & x_{m4} & x_{m4}^2 & x_{m4}^3 & x_{m9} \end{bmatrix}$$

Table 2: Comparison of speed (in milliseconds) of custom design_matrix() functions with built-in R model.matrix.lm() function

expr	min	lq	mean	median	uq	max	neval
design_matrix()	19.032	20.041	30.781	21.834	29.155	212.279	100
design_matrix_Cpp()	7.474	7.972	12.173	8.578	9.948	193.108	100
model.matrix.lm()	19.072	20.411	28.093	22.914	30.381	93.841	100

We made a custom function called `design_matrix()` in R and a Rcpp (C++) version called `design_matrix_Cpp()` to form the design matrix. This Rcpp tutorial was useful in converting the R functions to C++. The R implementation was slightly faster than the base built-in R version called `model.matrix.lm()` and the C++ version was three times faster than built-in R version. **Table 2** shows a speed comparison using R package `microbenchmark` showing minimum, 25th percentile, 50th percentile, mean, 75th percentile, and max time of 100 replications of the function to form the design matrix with $n = 7$ and $A \in \mathbb{R}^{17379 \times 11}$.

4. Normal Equation

The simplest way to solve $Ax = b$ would be to do use the normal equations.

The solution to $Ax = b$ using the normal equations is $x = (A^T A)^{-1} A^T b$

However, computers cannot represent real numbers exactly. The condition number $\kappa(A)$ of the input matrix A represents how much error there could be in the output. The condition number of $A^T A$ is equal to condition number of A^2. For our example, as n increases (a more complex and higher order polynomial), the condition number $\kappa(A)$ increases significantly. If the condition number is too high, then a computer will treat a non-singular matrix as singular. Then, there will be very large errors in the computation.

Table 3: Condition number of each value of n (maximum power of polynomial for `hour`) and relative error of normal equations versus SVD

	$\kappa(A)^2$	Relative error/error message R	Relative error C++
x_4	6.49×10^3	5.87×10^{-12}	6.17×10^{-12}
$x_4 + x_4^2$	2.09×10^6	-1.1×10^{-14}	-1.7×10^{-14}
$x_4 + x_4^2 + x_4^3$	9.08×10^8	1.18×10^{-12}	1.41×10^{-12}
$x_4 + x_4^2 + \cdots + x_4^4$	4.17×10^{11}	2.87×10^{-12}	2.71×10^{-12}
$x_4 + x_4^2 + \cdots + x_4^5$	2.03×10^{14}	1.52×10^{-8}	1.50×10^{-8}
$x_4 + x_4^2 + \cdots + x_4^6$	1.29×10^{17}	Error, Recripocal $\kappa(A)$: 5.89×10^{-18}	1.24×10^{-9}
$x_4 + x_4^2 + \cdots + x_4^7$	1.12×10^{20}	Error, Recripocal $\kappa(A)$: 6.14×10^{-21}	1.18×10^{-3}
$x_4 + x_4^2 + \cdots + x_4^8$	1.11×10^{23}	Error, Recripocal $\kappa(A)$: 6.35×10^{-24}	5.99×10^{-3}
$x_4 + x_4^2 + \cdots + x_4^9$	1.22×10^{26}	Error, Recripocal $\kappa(A)$: 6.78×10^{-27}	1.06×10^{-1}
$x_4 + x_4^2 + \cdots + x_4^{10}$	1.46×10^{29}	Error, Recripocal $\kappa(A)$: 1.29×10^{-29}	1.02×10^1
$x_4 + x_4^2 + \cdots + x_4^{11}$	1.94×10^{32}	Error, Recripocal $\kappa(A)$: 1.07×10^{-32}	1.06×10^1
$x_4 + x_4^2 + \cdots + x_4^{12}$	2.89×10^{35}	Error, Recripocal $\kappa(A)$: 7.16×10^{-35}	1.03×10^0
$x_4 + x_4^2 + \cdots + x_4^{13}$	4.86×10^{38}	Error, Recripocal $\kappa(A)$: 1.13×10^{-37}	2.12×10^0
$x_4 + x_4^2 + \cdots + x_4^{14}$	3.43×10^{45}	Error, Recripocal $\kappa(A)$: 2.67×10^{-40}	1.00×10^0
$x_4 + x_4^2 + \cdots + x_4^{15}$	2.04×10^{45}	Error, Recripocal $\kappa(A)$: 2.51×10^{-43}	1.09×10^0

Table 3 shows the square of the condition number for matrix A, the relative error/error message when solving $Ax = b$ for each value of n using R, and the relative error using Rcpp (C++). The relative error is calculated as the maximum error of any coefficient of the normal equations when compared to SVD (SVD is considered computationally precise).

However, R gives an error trying to find the inverse if the matrix is close to singular. The error message is the following: 'Error in solve.default(t(A) %*% A): system is computationally singular: reciprocal condition number ='. The reciprocal condition number given in the error message by R is close to the conditional number of A^2. On the other hand, RcppArmadillo does not give an error when finding the inverse of a system with a very high condition number. When $n > 9$ and $\kappa(A^TA) > 10^{29}$, the relative error is 100% or higher. R appears to give an error if the relative error is greater than 0.1% and $\kappa(A^TA) > 10^{17}$. With such large errors possible, it is not recommended to ever use the normal equations when solving $Ax = b$ on a computer.

Table 4: Comparison of speed (in milliseconds) of solving $Ax = b$ where $A \in \mathbb{R}^{17379 \times 7}$ using normal equations implemented in R vs Rcpp (C++)

expr	min	lq	mean	median	uq	max	neval
normal_equations(A, b)	5.954	6.286	7.70	6.737	7.517	22.960	100
normal_equations_Cpp(A, b)	2.216	2.342	2.55	2.469	2.744	4.525	100

Table 4 shows that the Rcpp (C++) implementation is on average 4 times faster than the R implementation of solving the normal equations.

5. QR Decomposition

Another way, a more computationally accurate way, to solve $Ax = b$ to use QR decomposition.

First, find the reduced QR decomposition such that $A = \hat{Q}\hat{R}$, $\hat{Q}$ is an orthogonal matrix, and $\hat{R}$ is an upper triangular matrix. $\hat{Q}$ meets the following condition: $\hat{Q}^T\hat{Q} = \hat{Q}\hat{Q}^T = I$. Also, $\hat{Q} \in \mathbb{R}^{17379 \times 7}$ and $\hat{R} \in \mathbb{R}^{7 \times 7}$ in this case as $A \in \mathbb{R}^{17379 \times 7}$

The solution to $Ax = b$ using QR decomposition is $x = \hat{R}^{-1}\hat{Q}^T b$.

Table 5: Comparison of speed (in milliseconds) of solving $Ax = b$ where $A \in \mathbb{R}^{17379\times7}$ using QR decomposition implemented in R vs Rcpp (C++)

expr	min	lq	mean	median	uq	max	neval
qr.solve(A, b)	5.250	5.490	6.873	5.78	6.409	16.309	100
qr_solve_Cpp(A, b)	5.313	5.499	6.077	5.71	6.110	15.186	100

Table 5 shows that the Rcpp (C++) implementation is slightly faster than R implementation of using QR decomposition to solve $Ax = b$.

6. Singular Value Decomposition

A longer way to solve $Ax = b$, but an even more accurate way than QR decomposition is to use Singular Value Decomposition.

First, find the reduced SVD such that $A = \hat{U}\hat{\Sigma}V^T$.

Also, $\hat{U} \in \mathbb{R}^{17379\times7}$, $V \in \mathbb{R}^{7\times7}$, and $\hat{\Sigma} \in \mathbb{R}^{7\times7}$ in this case as $A \in \mathbb{R}^{17379\times7}$.

The solution to $Ax = b$ using SVD is $x = V(\hat{U}^T b/\hat{\Sigma})$

Table 6: Comparison of speed (in milliseconds) of solving $Ax = b$ where $A \in \mathbb{R}^{17379\times7}$ using SVD implemented in R vs Rcpp (C++)

expr	min	lq	mean	median	uq	max	neval
svd_solve(A, b)	9.382	9.775	11.454	10.131	10.905	26.732	100
svd_solve_Cpp(A, b)	8.117	8.324	8.778	8.619	9.017	11.376	100

Table 6 shows that the Rcpp (C++) implementation is slightly faster than R implementation of using SVD to solve $Ax = b$.Now, it is time to compare the speed of solving $Ax = b$. If theory is correct, the normal equations should be fastest, followed by QR decomposition, and followed by SVD.

Table 7: Comparison of speed (in milliseconds) of solving $Ax = b$ where $A \in \mathbb{R}^{17379\times11}$ using QR decomposition or SVD implemented in R vs Rcpp (C++)

expr	min	lq	mean	median	uq	max	neval
normal_equations_Cpp(A, b)	3.838	3.980	4.201	4.058	4.429	5.403	100
qr.solve(A, b)	9.968	10.575	14.597	11.218	13.927	168.261	100
qr_solve_Cpp(A, b)	11.427	11.920	12.717	12.292	13.018	21.020	100
svd_solve(A, b)	19.954	20.799	23.806	22.417	27.291	34.312	100
svd_solve_Cpp(A,b)	17.413	17.957	18.833	18.477	19.313	26.481	100

Note that **Table 7** uses $n = 7$, the design matrix for `registered` and `cnt` instead of $n = 3$, the design matrix for `casual` in **Table 4, 5, 6**.

Normal equations using Rcpp (C++) is about 3 times faster than QR decomposition using C++. QR decomposition is about 70% faster than SVD. QR decomposition implemented in R is about twice as fast as using SVD implemented in R. Thus, we see that the results from **Table 7** are in line with what we should expect from theory.

7. Final Regression Model

Recall that y_1 = `casual`, y_3 = `cnt`, x_1 = `season`, x_2 = `yr`, x_4 = `hour`, x_9 = `temp` for below.

$y_1 = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_4 + \beta_4 x_4^2 + \beta_5 x_4^3 + \beta_6 x_9$

$y_3 = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_4 + \beta_4 x_4^2 + \beta_5 x_4^3 + \beta_6 x_4^4 + \beta_7 x_4^5 + \beta_8 x_4^6 + \beta_9 x_4^7 + \beta_{10} x_9$

The solution to $Ax = b$ for $b = y_1$ is:

$$x = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \\ \beta_5 \\ \beta_6 \end{bmatrix} = \begin{bmatrix} -42.145 \\ 0.6875 \\ 12.887 \\ -4.859 \\ 1.275 \\ -0.047 \\ 89.528 \end{bmatrix}$$

The solution to $Ax = b$ for $b = y_3$ is:

$$x = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \\ \beta_5 \\ \beta_6 \\ \beta_7 \\ \beta_8 \\ \beta_9 \\ \beta_{10} \end{bmatrix} = \begin{bmatrix} -145.05 \\ 17.18 \\ 88.57 \\ 29.45 \\ -63.199 \\ 24.27 \\ -3.62 \\ 0.258 \\ -0.0088 \\ 0.000116 \\ 243.131 \end{bmatrix}$$

Inputting the solution results in:

$y_1 = -42.145 + 0.6875x_1 + 12.887x_2 - 4.859x_4 + 1.275x_4^2 - 0.047x_4^3 + 89.528x_9$

$y_3 = -145.05 + 17.18x_1 + 88.57x_2 + 29.45x_4 - 63.199x_4^2 + 24.27x_4^3 - 3.62x_4^4 + 0.258x_4^5 - 0.0088x_4^6 + 0.000116x_4^7 + 243.131x_9$

Note that $\frac{1}{n}\sum_{i=1}^{n} x_1 = \bar{x_1} = 2.50164$, $\frac{1}{n}\sum_{i=1}^{n} x_2 = \bar{x_2} = 0.5025606$, $\frac{1}{n}\sum_{i=1}^{n} x_8 = \bar{x_8} = 0.4970$.

Input the means of x_1, x_2, and x_8 into the formulas of y_1 and y_3.

This results in eliminating all independent variables that aren't `hour` (x_4), resulting in the following two functions:

$y_1 = -4.859x_4 + 1.275x_4^2 - 0.047x_4^3 + 10.54523$

$y_3 = 29.45x_4 - 63.199x_4^2 + 24.27x_4^3 - 3.62x_4^4 + 0.258x_4^5 - 0.0088x_4^6 + 0.000116x_4^7 + 63.26103.$

This now allows us to graph `casual` (y_1) and `registered` (y_3) as a function of x_4 (`hour`) and see how well the least squares regression fits the actual data.

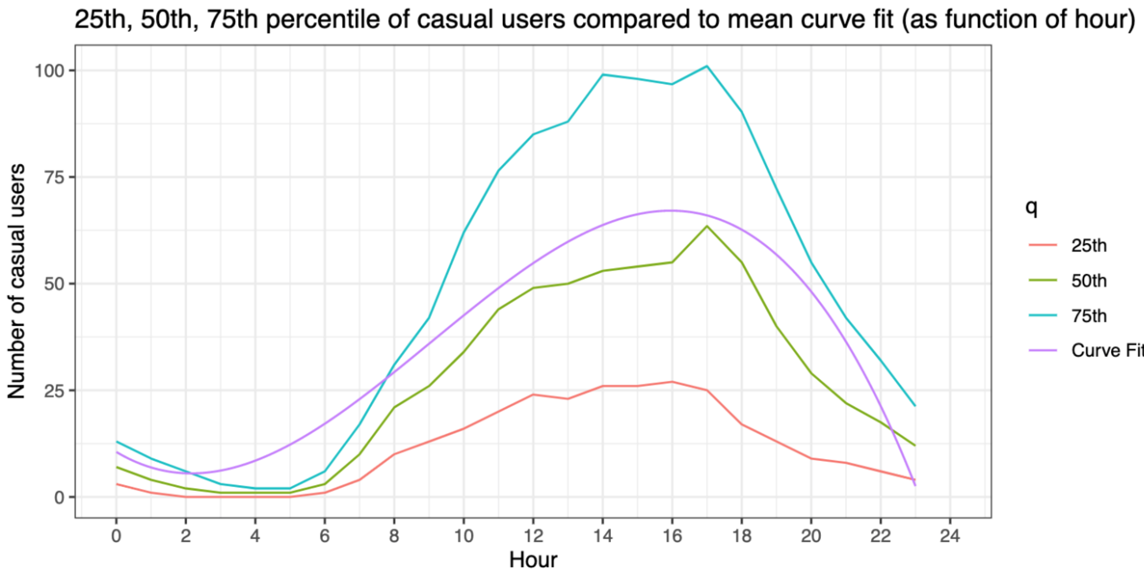

Figure 5: The 25th, 50th, and 75th percentile of casual users (731 days) are graphed for each hour of the day. The mean values of season, yr, and temp are entered into the regression solution to give a constant value. Thus, the regression is now only in terms of hour. Then, the mean curve fit of the regression can be compared to the 25th, 50th, and 75th percentiles of casual users.

Figure 5 shows that the cubic polynomial function is not a great fit for hours 4 to 6 as the mean curve fit is above the 75th percentile. The curve does not dip down far enough from hours 4 to 6. Note that interquartile range (75th percentile minus 25th percentile) for `casual` is very large from **Figure 5** which might make predicting data quite difficult. Also **Figure 5** suggests that `casual` is skewed right since the mean curve fit is almost always above the 50th percentile.

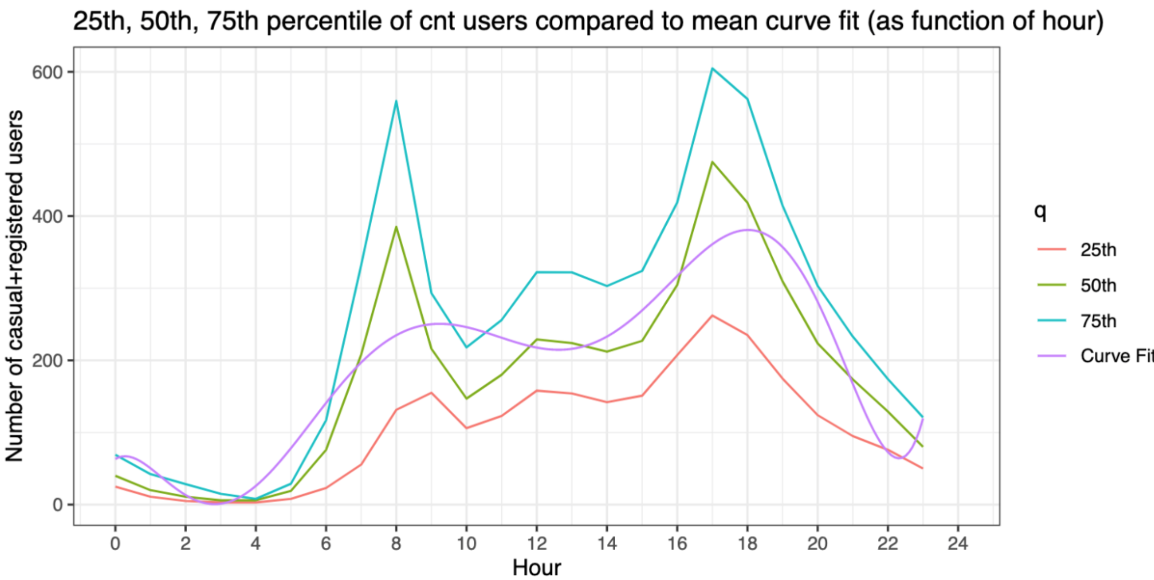

Figure 6: Same description as for Figure 5, except the variable is now cnt (sum of registered and casual users) rather than casual.

Figure 6 shows the 7-degree (septic) polynomial function is quite a good fit for `cnt`. However, near hour 22, the mean curve fit starts to slope upwards again, which is not correct. Also, similar to **Figure 5**, the mean curve fit overestimates the number of users from hours 4 to 6 again.

8. Principal Component Analysis

Using Principal Component Analysis (PCA) to solve Ax = b. To do principal components we have X data which are uncorrelated projections Y of the data points $X1, ..., Xn$ with each Xi are a set real numbers as a row vector. These X are projected onto some directions unit vectors such that var(Y) is maximal. For the computing of finding variance, standard deviation, covariance for PCA we can uses these formulas below:

To compute the mean for our x's we use the mean formula: $\bar{X} = \frac{1}{n}\sum_{i=i}^{n} x_i$

The find the variance we use: $var(x) = \frac{1}{n-1}\sum_{i=i}^{n}(x_i - \bar{x})^2$

Also our covariance is cov(x, y) $= \frac{1}{n-1}\sum_{i=i}^{n}(x_i - \bar{x})(y_i - \bar{y})$

Table 8: This table gives the standard deviation, proportion of variance explained by each of the principal component, and the cumulative proportion of variance explained.

Importance of components:	PC1	PC2	PC3	PC4	PC5	PC6
Standard deviation	1.7015	1.1025	0.9219	0.7758	0.65553	0.08902
Proportion of Variance	0.4825	0.2026	0.1416	0.1003	0.07162	0.00132
Cumulative Proportion	0.4825	0.6851	0.8267	0.9271	0.99868	1.00000

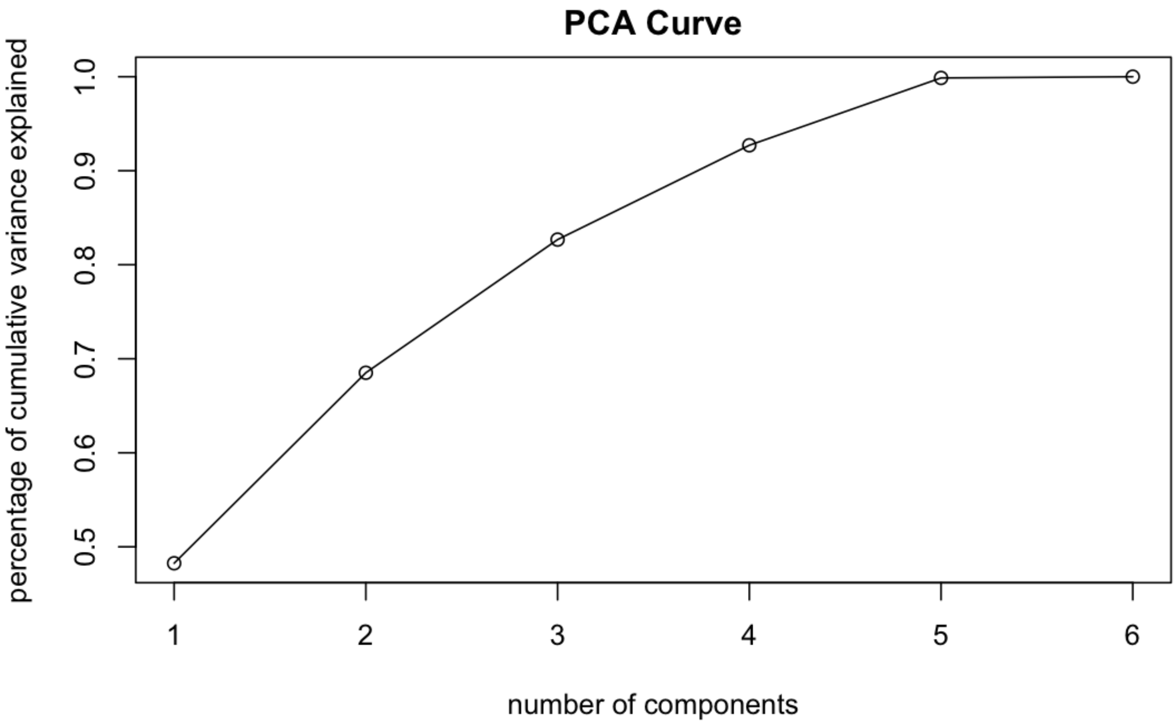

Figure 7: This figure showcases how percentage of the variance in the data explained as we add principal components. For example, out PCA1 48 percent of the data set. The PCA2 explains 20 percent and PCA3 explains 14 percent of the data. The total of the 3 components togther 82 percent of the data.

We made a custom function called `pca_func()` in R which is helpful for capture the majority of variance in your predictors.was useful for fitting the PCA, and using the built in function `prcomp()`. With this function we can prcomp it will performs the principal components analysis on the dataset matrix by taking the SVD.

9. K-Nearest Neighbors

K-nearest neighbour (KNN) is a machine learning algorithm. This algorithms can be used for both classification and regression problems. In addition, (KNN) is supervised learning which means that we know the type of data our data is and what outcomes we are looking for.We are doing a prediction on how "cnt"(count) being our outcome variable affecting predictor variables which our our other variables in our data set. Furthermore, for implementation of the KNN algorithm we will reference back to the PCA problem and use the 3 components based coordinates and other categorical variables which explain the 82 percent variance to do KNN.

When using the KNN formula we are able to find the distance between two points in featured space we can easy use the Euclidean distance formula to do so:

$d(p,q)= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$

k-Nearest Neighbors 512 samples 10 predictor No pre-processing Resampling: Cross-Validated (10 fold, repeated 3 times) Summary of sample sizes: 460, 461, 460, 461, 461, 461, ... Resampling results across tuning parameters:			
k	RMSE	Rsquared	MAE
1	925.1560	0.7839712	690.8140
2	727.9760	0.8608545	560.1111
3	693.0851	0.8758349	538.5740
4	670.6142	0.8858940	526.6748
5	661.4997	0.8910960	523.4297
6	656.4150	0.8942347	519.3416
7	656.6373	0.8953553	522.3512
8	658.6736	0.8959119	522.6202
9	655.1282	0.8980874	521.0988
10	657.1985	0.8981599	524.6721
11	660.3379	0.8978063	527.6320
12	665.1246	0.8967227	534.0605
13	668.0968	0.8965219	536.8265
14	669.2021	0.8964224	539.2323
15	674.8979	0.8959398	543.9940
16	675.1185	0.8968547	546.7142
17	679.8648	0.8962774	552.8564
18	684.0043	0.8959587	556.5736
19	694.5197	0.8933500	565.4170
20	699.3458	0.8931800	570.3835

Table 9: This table here shows a list of RMSE, MAE(Mean absolute error), and R-squared values. We will be focusing on the The RSME is Root Mean Square Error(RMSE) which is the standard deviation also known as (prediction errors). We use the RSME since it is used to select the optimal model using the smallest value. The final value used for the model was k = 9 which is the smallest RSME value.

With KNN we did a Cross-Validation for training and testing data. The training set was set to 70 percent then use the model to test our predictions on the test set which is 30 percent. In addition, out full data will include our 3 PCA used to generate 82 percent and "season", "yr", "mnth", "holiday" ,"weekday", "workingday", "weathersit".

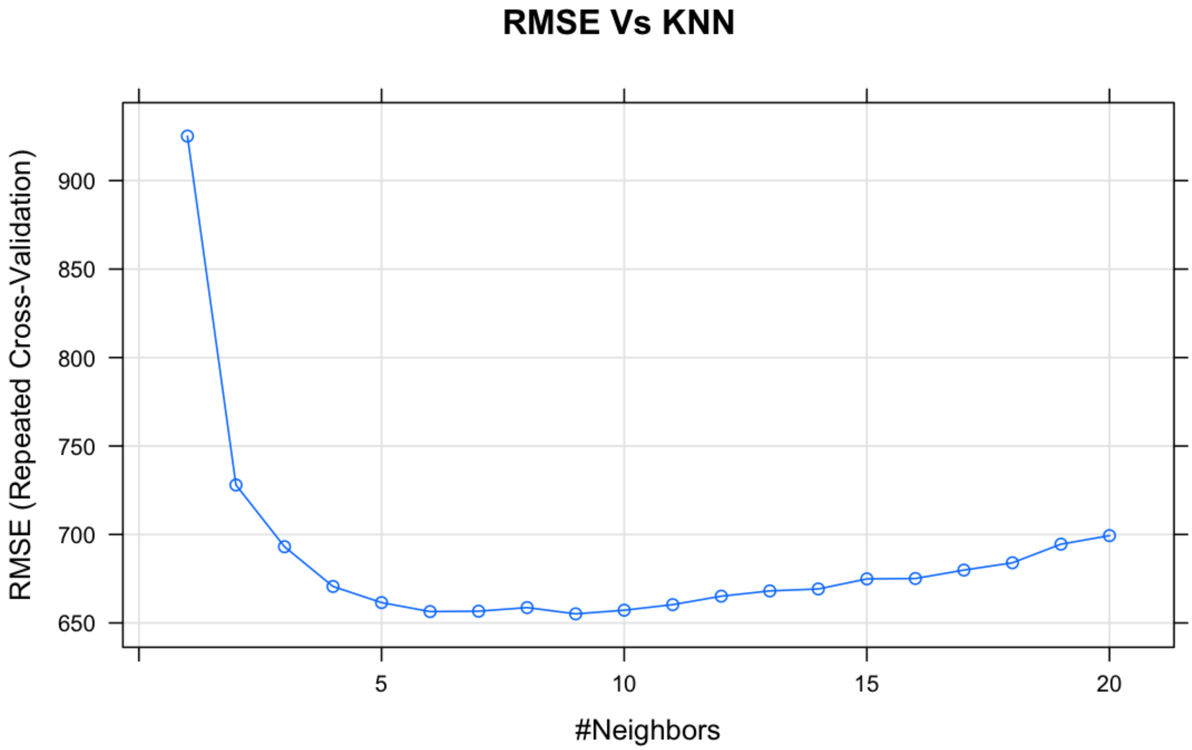

Figure 8: This figure show the average RMSE of the estimated prediction error on a validation set. The value of K for which the RMSE is lowest is the value of K that we will take. So that is K = 9. We can see as our neighbors increase so does the the error reduce and our neighbors RMSE performance gets better after 10 neighbors with increase in RSME value till 20 neighbors.

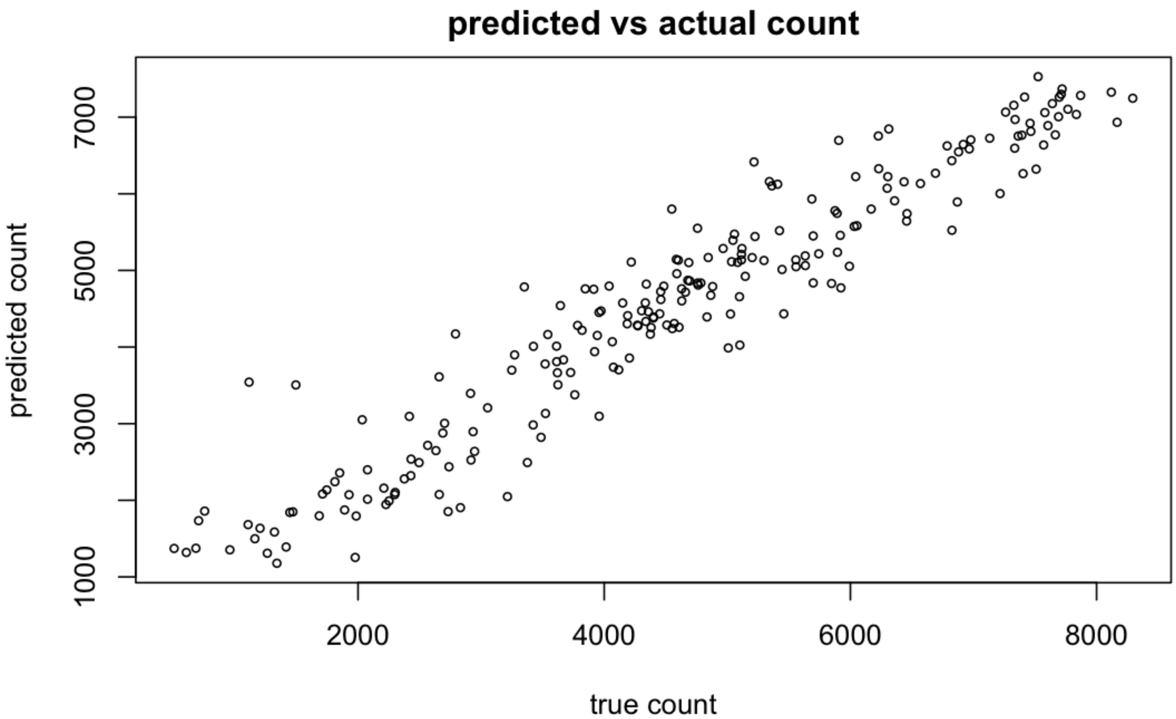

Figure 9: This figure show there's a strong correlation between the model's predictions and its actual results with our K = 9. What K= 9 means it should look into the 9 nearest neighbors and classify the new data point its similar too. In addition, with the $R^2 = 0.912$ being very good since R^2 value close to 1 indicated a positive linear association between the predicted vs actual data and out R-squares is 91 percent.

10. Conclusion

In conclusion linear algebra can be widely used in a variety of different ways and play an important impact on machine learning algorithms, reduce dimension of data, and even more. From this bike sharing set we are able to showcase how linear algebra is used through the different way and formulas used to implement least squares. With the implementations of least squares we are ultimately able to exhibit the best molding curve to fit our bike data set. By doing so it will allow use to see the impact our data make against each other and how bike counts effect environmental/seasonal issues and as well as how further exploration can be done.

11. Reference

Hadi Fanaee-T: UCI Machine Learning Repository,
`https://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset`

Guido Gerig: Chapter 11 Least Squares, Pseudo-Inverses, PCA SVD,
`http://www.sci.utah.edu/ gerig/CS6640-F2012/Materials/pseudoinverse-cis61009sl10.pdf`

Shubham Panchal: k Nearest Neighbor Classifier (kNN)-Machine Learning Algorithms,
`https://medium.com/@equipintelligence/k-nearest-neighbor-classifier-knn-machine-learning-algorithms-ed62feb86582`