Prediction Competition 2:Regression Trees, Cross-Validation, and Variable Importance

Anonymized name: Sukuna
$R^2 = 0.40$
$MSE = 1.06$

Q2

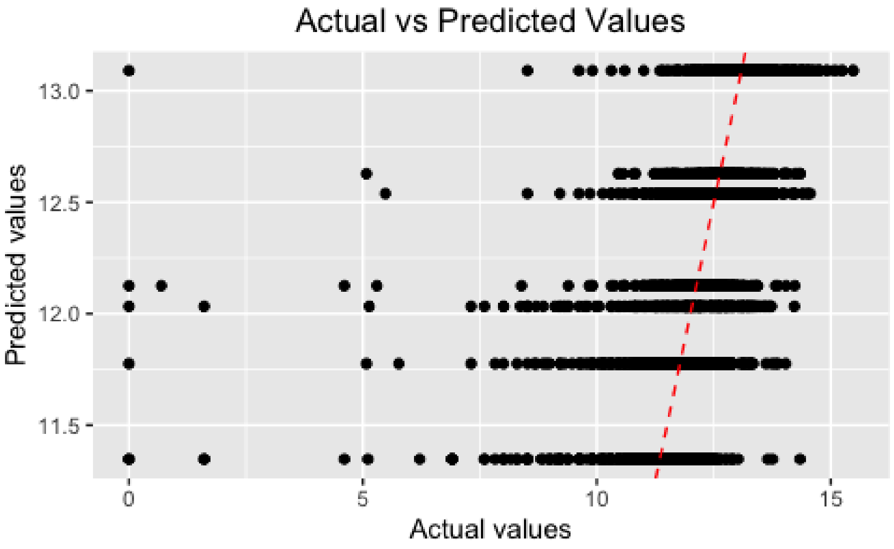


Figure 1: Actual vs. Predicted values

Q3

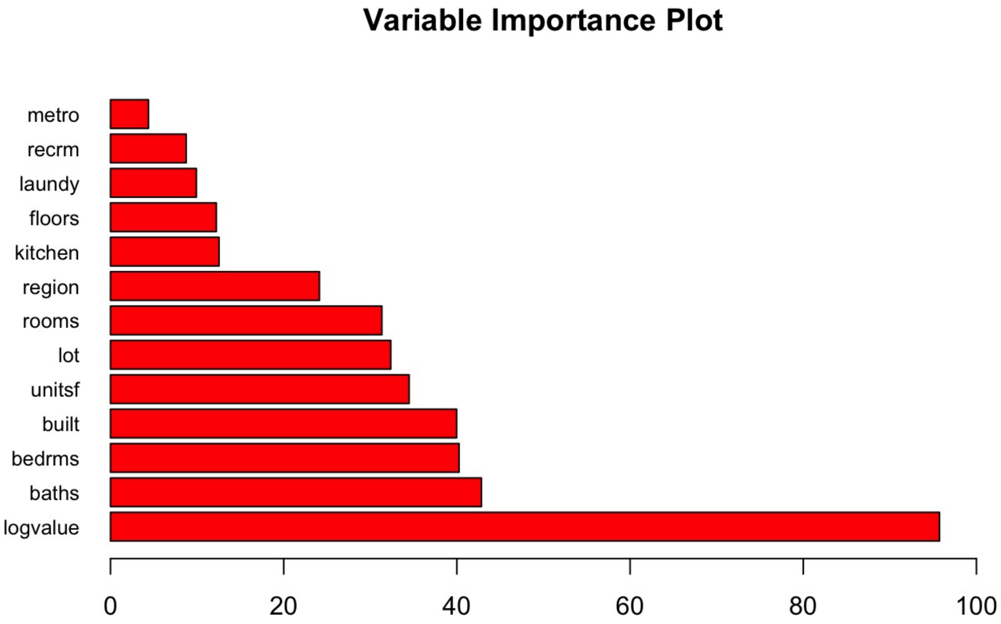


Figure 2: Variable Importance Bargraph

Q4

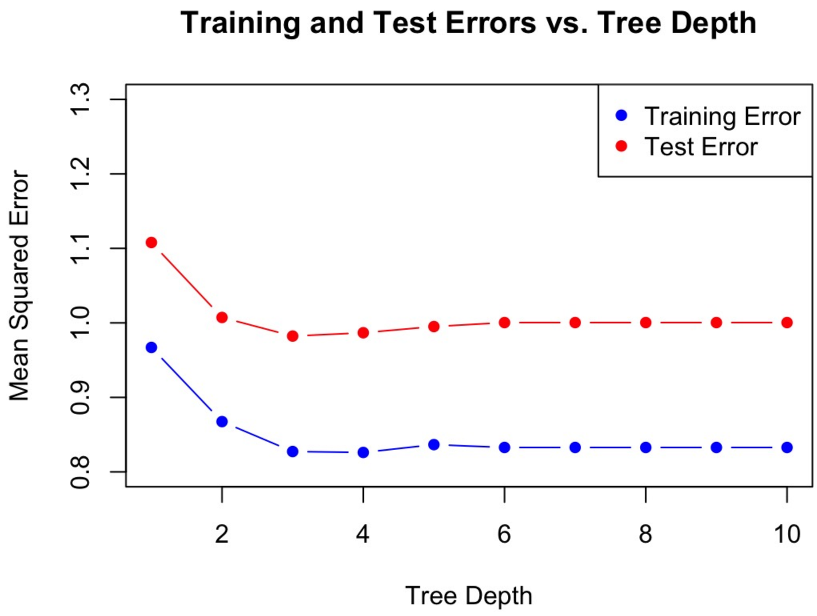
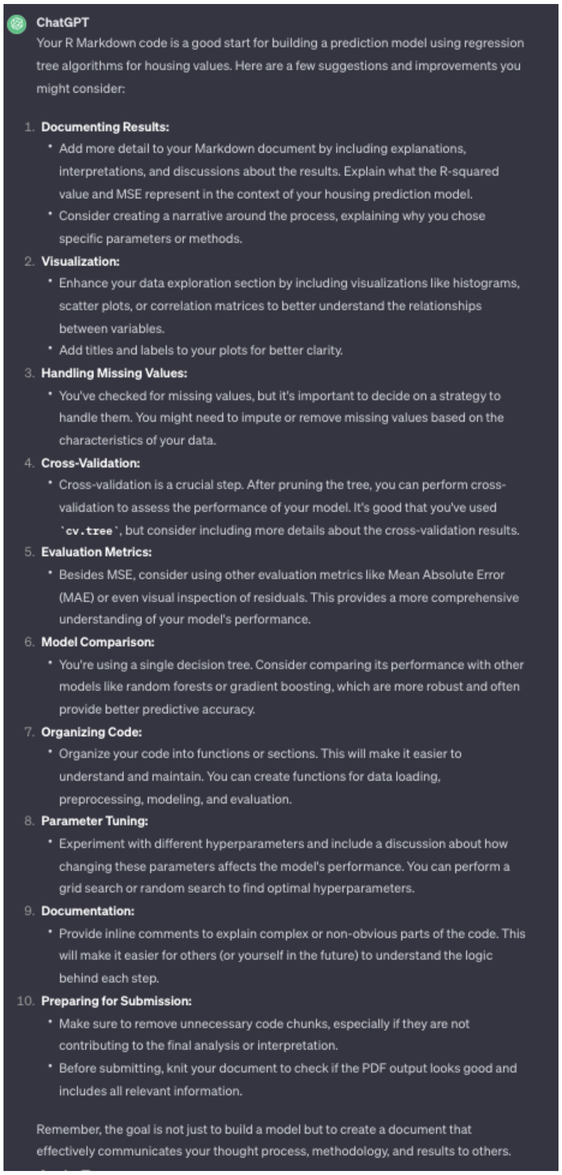


Figure 3: Test vs train

Q5



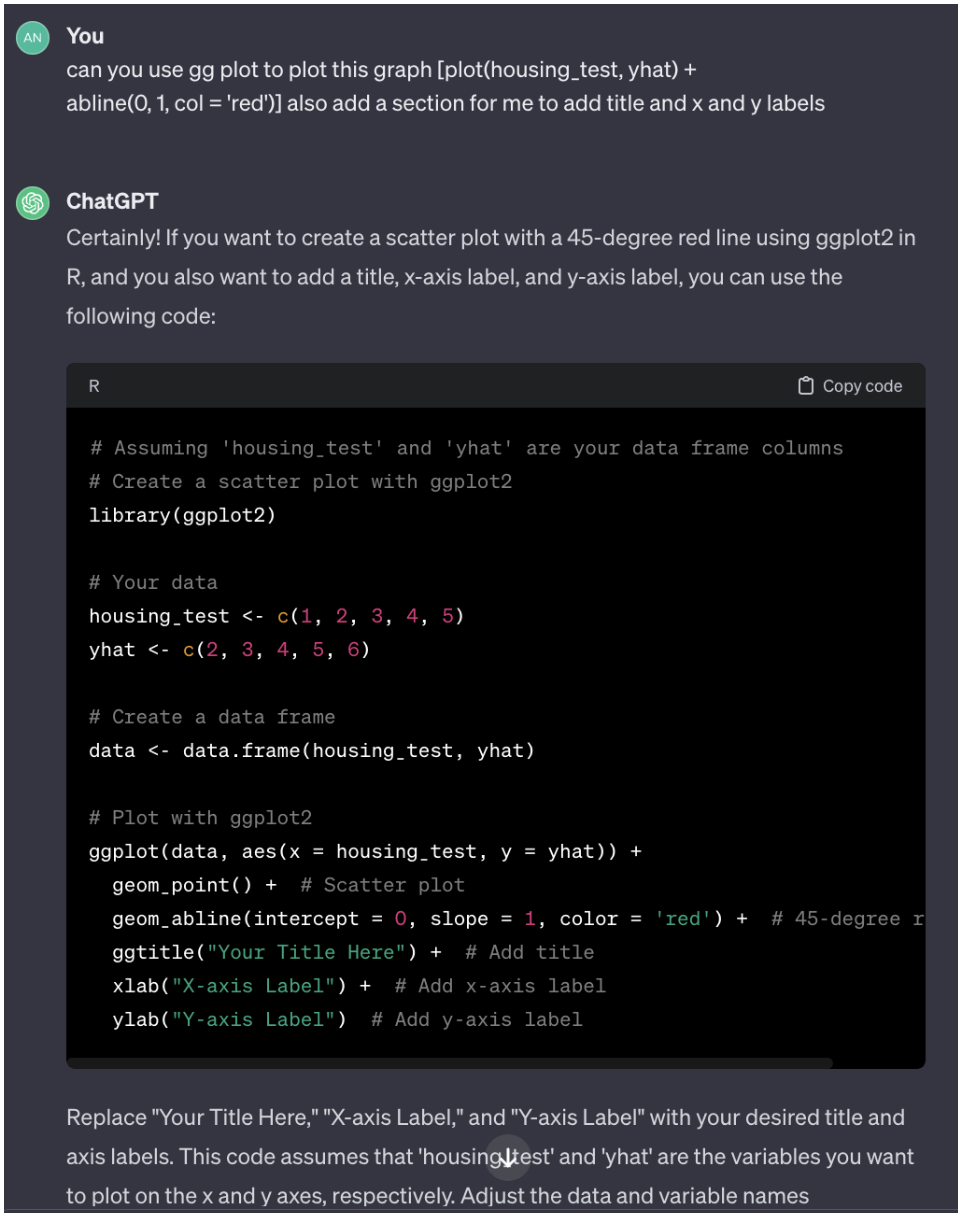
ChatGPT

Your R Markdown code is a good start for building a prediction model using regression tree algorithms for housing values. Here are a few suggestions and improvements you might consider:

1. **Documenting Results:**
    - Add more detail to your Markdown document by including explanations, interpretations, and discussions about the results. Explain what the R-squared value and MSE represent in the context of your housing prediction model.
    - Consider creating a narrative around the process, explaining why you chose specific parameters or methods.
2. **Visualization:**
    - Enhance your data exploration section by including visualizations like histograms, scatter plots, or correlation matrices to better understand the relationships between variables.
    - Add titles and labels to your plots for better clarity.
3. **Handling Missing Values:**
    - You've checked for missing values, but it's important to decide on a strategy to handle them. You might need to impute or remove missing values based on the characteristics of your data.
4. **Cross-Validation:**
    - Cross-validation is a crucial step. After pruning the tree, you can perform cross-validation to assess the performance of your model. It's good that you've used `` `cv.tree` ``, but consider including more details about the cross-validation results.
5. **Evaluation Metrics:**
    - Besides MSE, consider using other evaluation metrics like Mean Absolute Error (MAE) or even visual inspection of residuals. This provides a more comprehensive understanding of your model's performance.
6. **Model Comparison:**
    - You're using a single decision tree. Consider comparing its performance with other models like random forests or gradient boosting, which are more robust and often provide better predictive accuracy.
7. **Organizing Code:**
    - Organize your code into functions or sections. This will make it easier to understand and maintain. You can create functions for data loading, preprocessing, modeling, and evaluation.
8. **Parameter Tuning:**
    - Experiment with different hyperparameters and include a discussion about how changing these parameters affects the model's performance. You can perform a grid search or random search to find optimal hyperparameters.
9. **Documentation:**
    - Provide inline comments to explain complex or non-obvious parts of the code. This will make it easier for others (or yourself in the future) to understand the logic behind each step.
10. **Preparing for Submission:**
    - Make sure to remove unnecessary code chunks, especially if they are not contributing to the final analysis or interpretation.
    - Before submitting, knit your document to check if the PDF output looks good and includes all relevant information.

Remember, the goal is not just to build a model but to create a document that effectively communicates your thought process, methodology, and results to others.

Figure 4: Screen capture of my conversation with ChatGPT



You

can you use gg plot to plot this graph [plot(housing_test, yhat) + abline(0, 1, col = 'red')] also add a section for me to add title and x and y labels

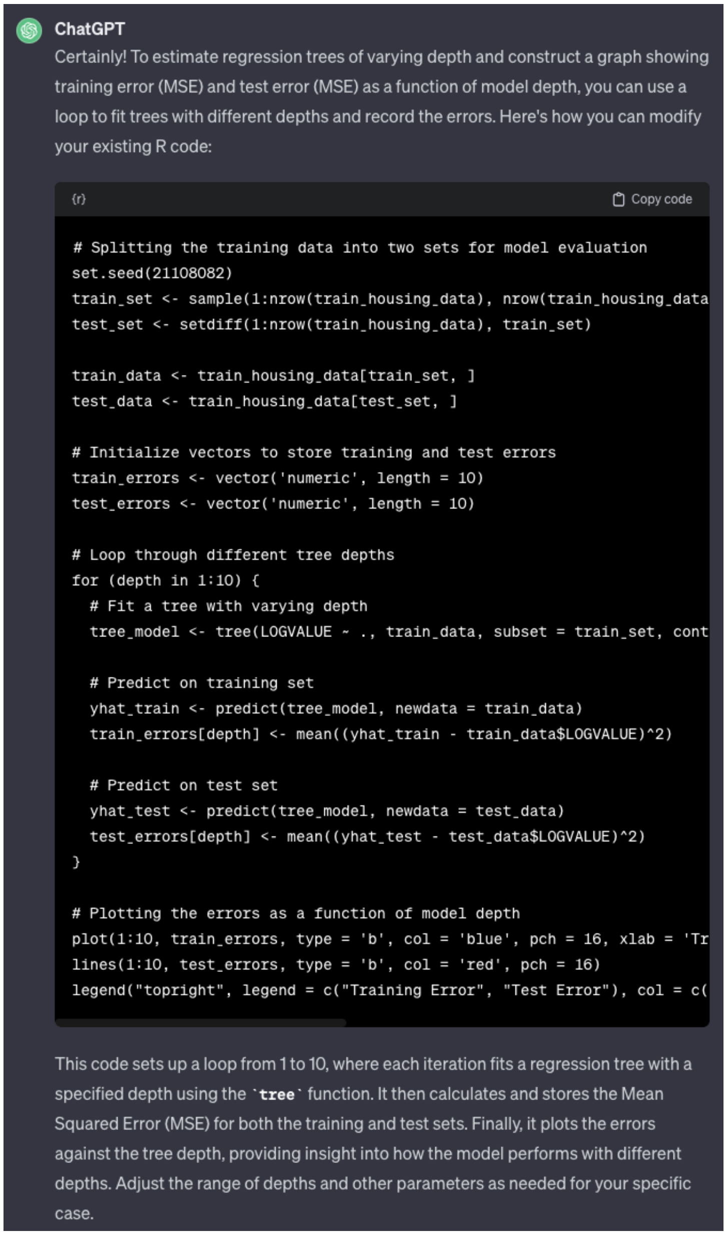
ChatGPT

Certainly! If you want to create a scatter plot with a 45-degree red line using ggplot2 in R, and you also want to add a title, x-axis label, and y-axis label, you can use the following code:

```r
# Assuming 'housing_test' and 'yhat' are your data frame columns
# Create a scatter plot with ggplot2
library(ggplot2)

# Your data
housing_test <- c(1, 2, 3, 4, 5)
yhat <- c(2, 3, 4, 5, 6)

# Create a data frame
data <- data.frame(housing_test, yhat)

# Plot with ggplot2
ggplot(data, aes(x = housing_test, y = yhat)) +
  geom_point() +  # Scatter plot
  geom_abline(intercept = 0, slope = 1, color = 'red') +  # 45-degree r
  ggtitle("Your Title Here") +  # Add title
  xlab("X-axis Label") +  # Add x-axis label
  ylab("Y-axis Label")  # Add y-axis label
```

Replace "Your Title Here," "X-axis Label," and "Y-axis Label" with your desired title and axis labels. This code assumes that 'housing_test' and 'yhat' are the variables you want to plot on the x and y axes, respectively. Adjust the data and variable names

Figure 5: Screen capture of my conversation with ChatGPT



ChatGPT

Certainly! To estimate regression trees of varying depth and construct a graph showing training error (MSE) and test error (MSE) as a function of model depth, you can use a loop to fit trees with different depths and record the errors. Here's how you can modify your existing R code:

```
# Splitting the training data into two sets for model evaluation
set.seed(21108082)
train_set <- sample(1:nrow(train_housing_data), nrow(train_housing_data
test_set <- setdiff(1:nrow(train_housing_data), train_set)

train_data <- train_housing_data[train_set, ]
test_data <- train_housing_data[test_set, ]

# Initialize vectors to store training and test errors
train_errors <- vector('numeric', length = 10)
test_errors <- vector('numeric', length = 10)

# Loop through different tree depths
for (depth in 1:10) {
  # Fit a tree with varying depth
  tree_model <- tree(LOGVALUE ~ ., train_data, subset = train_set, cont

  # Predict on training set
  yhat_train <- predict(tree_model, newdata = train_data)
  train_errors[depth] <- mean((yhat_train - train_data$LOGVALUE)^2)

  # Predict on test set
  yhat_test <- predict(tree_model, newdata = test_data)
  test_errors[depth] <- mean((yhat_test - test_data$LOGVALUE)^2)
}

# Plotting the errors as a function of model depth
plot(1:10, train_errors, type = 'b', col = 'blue', pch = 16, xlab = 'Tr
lines(1:10, test_errors, type = 'b', col = 'red', pch = 16)
legend("topright", legend = c("Training Error", "Test Error"), col = c(
```

This code sets up a loop from 1 to 10, where each iteration fits a regression tree with a specified depth using the `tree` function. It then calculates and stores the Mean Squared Error (MSE) for both the training and test sets. Finally, it plots the errors against the tree depth, providing insight into how the model performs with different depths. Adjust the range of depths and other parameters as needed for your specific case.

Figure 6: Screen capture of my conversation with ChatGPT

Code

~~~
---
title: "Prediction Competition 2"
output: pdf_document
---

Anonymized name: Sukuna
$R^2 = 0.40$
$MSE = 1.06$



```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
~~~

Importing libraries
```{r, include=FALSE}
library(readr)
library(ggplot2)
library(tree)
library(randomForest)
library(rpart)
```

Importing data
```{r}
setwd('/Users/andrew/Downloads/UW courses/ECON 626/Prediction Competition 2')
train_housing_data = read.csv('ECON626_PC2_train.csv')
test_housing_data = read.csv('ECON626_PC2_test.csv')
```

Viewing our data
```{r}
names(train_housing_data) <- lapply(names(train_data), tolower)
head_train = head(train_housing_data)
print(head_train)
```

```{r}
head_test = head(test_housing_data)
print(head_test)
```
```
```

```
‘‘‘{r}
#Count of na
sum(is.na(train_housing_data))
sum(is.na(test_housing_data))

#Searching for na values
which(is.na(train_housing_data))
which(is.na(test_housing_data))
‘‘‘

‘‘‘{r}
set.seed(21108082)
#set seed for reproducability

help(tree)

train <- sample(1:nrow(train_housing_data), nrow(train_housing_data) *.8)
tree_housing <- tree(logvalue ˜ ., train_housing_data , subset = train)
model = summary(tree_housing)
model
‘‘‘

‘‘‘{r}
plot(tree_housing)
text(tree_housing , pretty = 0)
‘‘‘

‘‘‘{r}
tree_housing
‘‘‘

‘‘‘{r}
cv_housing <- cv.tree(tree_housing)
plot(cv_housing$size , cv_housing$dev, type = "b")
‘‘‘


‘‘‘{r}
yhat <- predict(tree_housing , newdata = train_housing_data[-train , ])

housing_test <- train_housing_data[-train, "logvalue"]

plot_data <- data.frame(yhat, housing_test)

ggplot(plot_data, aes(x = housing_test, y = yhat)) +
  geom_point() +  # Scatter plot
```

~~~
  geom_abline(intercept = 0, slope = 1, color = 'red', linetype = "dashed", show_guide=TRUE
  ggtitle("Actual vs Predicted Values") +  # Add title
  xlab("Actual values") +  # Add x-axis label
  ylab("Predicted values") +  # Add y-axis label
  theme(plot.title = element_text(hjust = 0.5))

```

```{r}
r2 = 1-(sum(model$residuals^2))/sum((train_housing_data$logvalue-mean(train_housing_data$log

MSE = mean((yhat - housing_test)^2)
print(MSE)
```


```{r}

bag_housing <- randomForest(logvalue ~ ., data=train_housing_data , subset = train, mtry = 1

class(bag_housing)

var_importance <- importance(bag_housing)

var_importance

overall_importance <- sort(var_importance[, 1], decreasing = T)


barplot(overall_importance,
        names.arg = colnames(train_housing_data)[-ncol(train_housing_data)],
        las = 1,  # Set las to 1 for horizontal labels
        main = "Variable Importance Plot",
        col = "red",  # Set color to red
        cex.names = 0.8,
        horiz = TRUE,  # Set horiz to TRUE for horizontal bars
        xlim = c(0, 100)
)

```



```{r}
# Splitting the training data into two sets for model evaluation
set.seed(21108082)
~~~

~~~
train_set <- sample(1:nrow(train_housing_data), nrow(train_housing_data) * 0.2)
test_set <- setdiff(1:nrow(train_housing_data), train_set)

train_data <- train_housing_data[train_set, ]
test_data <- train_housing_data[test_set, ]

# Initialize vectors to store training and test errors
train_errors <- vector(’numeric’, length = 10)
test_errors <- vector(’numeric’, length = 10)

# Loop through different tree depths
for (depth in 1:10) {
  # Fit a tree with varying depth
  tree_model <- rpart(logvalue ˜ ., train_data, subset = train_set, control = list(maxdepth

  # Predict on training set
  yhat_train <- predict(tree_model, newdata = train_data)
  train_errors[depth] <- mean((yhat_train - train_data$logvalue)ˆ2)

  # Predict on test set
  yhat_test <- predict(tree_model, newdata = test_data)
  test_errors[depth] <- mean((yhat_test - test_data$logvalue)ˆ2)
}

# Plotting the errors as a function of model depth
plot(1:10, train_errors, type = ’b’, col = ’blue’, pch = 16, xlab = ’Tree Depth’, ylab = ’Me
lines(1:10, test_errors, type = ’b’, col = ’red’, pch = 16)
legend("topright", legend = c("Training Error", "Test Error"), col = c("blue", "red"), pch =


‘‘‘


‘‘‘{r}
set.seed(21108082)

housing_test <- train_housing_data[-train, ]
dim(housing_test)
house_values <- train_housing_data$LOGVALUE[-tograin]
length(house_values)
#make sure both = 4000

pred_tree <- predict(tree_housing , housing_test)
table(pred_tree , house_values)
~~~

~~~~
```


```{r}

test_yhat <- predict(tree_housing , newdata = test_housing_data)

```


```{r, include=FALSE}
# Writing the predictions to a csv

predictions <- test_yhat

r2 <- r2

# Creating a dataframe with a single column of mixed data types
df <- data.frame(Column = c(21108082, "Sukuna", r2, predictions), stringsAsFactors = FALSE)

write.table(df, "/Users/andrew/Downloads/UW courses/ECON 626/Prediction Competition 2/pc2.cs


#In general:

#x <- c(1, 2, 3, 4, 5)

# Creating a dataframe with a single column of mixed data types
#df <- data.frame(Column = c(st number, "Anonymized name", x), stringsAsFactors = FALSE)

#write.table(df, "filelocation/filename.csv", sep=",",row.names=FALSE, col.names=FALSE)

```
~~~~