IDS 572 HW3

Kritika Raghuwanshi, Yaze Gao, Shixun Jiang

2023-03-09

Problem 1

Suppose we two different tests, T1 and T2 that detect a particular type of cancer. T1 had been evaluated on a population of 200 instances, out of which 100 were known to be suffering from cancer, while the remaining 100 were healthy. T2 had been evaluated on a different population of 1100 instances, out of which 100 were known to be suffering from cancer, while the remaining 1000 were healthy. The results of these tests are shown in the following confusion matrices, along with the values of the following evaluation measures: TPR, FPR, Precision, the F-measure, and TPR/FPR.

(a) Calculate the TPR, FPR, Precision, F-score and TPR/FPR for both tests.

A.

1. For Test T1:

- TPR = TP / TP + FN = 40 / 40 + 60 = 0.4
- FPR = FP / FP + TN = 10 / 10 + 90 = O.1
- Precision = TP / TP + FP = 40 / 40 + 10 = 0.8
- Recall = TP / TP + FN = 40 / 40 + 60 = 0.4
- F-score = 2 (Precision * Recall) / Precision + Recall = 0.53
- TPR/FPR = 0.4 / 0.1 = 4

2. For Test T2:

- TPR = TP / TP + FN = 40 / 40 + 60 = 0.4
- FPR = FP / FP + TN = 55 / 55 + 945 = O.055
- Precision = TP / TP + FP = 40 / 40 + 55 = 0.421
- Recall = TP / TP + FN = 40 / 40 + 60 = 0.4
- F-score = 2 (Precision * Recall) / Precision + Recall = 0.41
- TPR/FPR = 0.4 / 0.055 = 7.27

(b) According to the F-score, which test is better?

A. Because the core idea of F-score is to improve the precision and recall as much as possible while hoping that the difference between precision and recall is as small as possible, the F- score of Test2 = 0.41, which is better than the F- score of Test1 = 0.53 from the F- score point of view.

(c) According to TPR/FPR, which test is better?

A. TPR/FPR is a measure of the trade-off between true positives and false positives, so Test2 has a higher value and therefore Test2 is better.

(d) For this question, which evaluation measure between F-score and TPR/FPR, should we use to make our selection between T1 and T2? Explain.

A.In this case, we should use two evaluation measures to choose between T1 and T2. the F-score measures the balance between accuracy and recall, while the TPR/FPR measures the trade-off between true positives and false positives. Thus, by considering both measures, we can get a more complete picture of each test's performance and make more informed decisions.

Problem 2

Consider a classifier that classifies documents as being either relevant or non-relevant.

(a) Which evaluation measure do we use for this classifier? Accuracy, precision, and/or recall? Justify your answer.

A. When evaluating the performance of a binary classifier that classifies documents as relevant or non-relevant, we should consider both Accuracy and Precision / Recall as evaluation metrics.

Accuracy measures the percentage of correct predictions made by the classifier, which is the ratio of the number of correctly classified documents to the total number of documents in the data set. However, accuracy alone may not be sufficient to evaluate the classifier's performance, especially when the data set is imbalanced. For example, if the data set contains a large proportion of non-relevant documents, a classifier that simply labels all documents as non-relevant will achieve a high accuracy but not be useful in practical applications.

Precision measures the proportion of true positives (relevant documents correctly classified as relevant) to the total number of documents classified as relevant. It answers the question, "What proportion of the documents that the classifier identified as relevant were actually relevant?" Precision is important when we want to minimize false positives, i.e., when we don't want to classify non-relevant documents as relevant.

Recall, on the other hand, measures the proportion of true positives to the total number of relevant documents in the data set. It answers the question, "What proportion of the relevant documents in the data set did the classifier correctly identify as relevant?" Recall is important when we want to minimize false negatives, i.e., when we don't want to miss any relevant documents.

n summary, while accuracy is an important evaluation metric, **it may not provide a complete picture of the classifier's performance, especially when the data set is imbalanced**. Precision and recall provide additional insights into the classifier's performance in terms of minimizing false positives and false negatives, respectively. **Therefore, it is recommended to consider all three metrics when evaluating a classifier for document classification**.

(b) Suppose that we have a collection of 10 documents- named D1, ... D10- and two different classifiers A and B. Give an example of two result sets of documents, Aq and Bq, assumed to have been returned by two different systems in response to a query q, constructed such that Aq has clearly higher precision than Bq, but Aq and Bq have the same accuracy.

A. This problem can be solved in a few different ways and there's no one correct answer, but here are a couple of scenarios:

1. Example 1:

Suppose the following relevance labels for the 10 documents:

Relevant: D1, D2, D3, D4

Non-relevant: D5, D6, D7, D8, D9, D10

Now, suppose that both classifiers A and B are asked to retrieve the relevant documents for the query q, classifier A returns the following set of documents:

- Aq = {D1, D2, D3}

Classifier B, on the other hand, returns a larger set of documents:

- Bq = {D1, D2, D3, D5, D6}

Both sets have the same accuracy, which is the fraction of relevant and non-relevant documents correctly identified:

- Accuracy = (3+5)/10 = 8/10 = 0.8

However, Aq has higher precision than Bq, which is the fraction of retrieved documents that are relevant:

- For classifier A, precision = (# of relevant documents in Aq) / (# of documents in Aq) = 3/3 = 1.0
- For classifier B, precision = (# of relevant documents in Bq) / (# of documents in Bq) = 3/5 = 0.6

In this example, **both classifiers have the same accuracy**, but **Aq has higher precision than Bq** because it returns fewer irrelevant documents. Classifier A retrieves only relevant documents, while classifier B retrieves two additional non-relevant documents, which reduces its precision.

(c) Suppose a classifier returns 3 relevant documents and 2 irrelevant documents to a search query. There are a total of 8 relevant documents in the collection. What is the precision of the system on this search, and what is its recall?

A.

1. Precision measures the fraction of relevant documents among the documents that the classifier returned as relevant. It is calculated as:

- precision = TP / TP + FP
- In our case, precision = number of relevant documents returned / total number of documents returned

Since the classifier returned 3 relevant documents and 2 irrelevant documents. Therefore, the precision is:

- precision = 3 / (3 + 2) = 0.6

Hence the Precision of the system is 0.6 or 60%.

2. Recall measures the fraction of relevant documents among all the relevant documents in the collection. It is calculated as:

- recall = TP / TP + FN
- In this case, recall = number of relevant documents returned / total number of relevant documents in the collection

Since there are a total of 8 relevant documents in the collection, and the classifier returned 3 relevant documents. Therefore, the recall is:

- recall = 3 / 8 = 0.375

Hence the Recall of the system is 0.375 or 37.5%.

Problem 3

We are looking to develop a machine learning algorithm to predict whether someone will pay a loan back or not.

(a) What is the positive class?

A. The positive class in this scenario would be the people / borrowers who pay back the loan.

(b) What would a recall of 75% mean?

A. A recall of 75% means that out of all the actual positive cases (people who paid back the loan), the algorithm correctly identified 75% of them as positive. In other words, 75% of the borrowers that would pay back the loan are approved by this system. We miss 25% (False Negatives / FN) of people that would have paid us back by rejecting them. In general, the problem with a low recall is that we are rejecting customers who we would have paid us back.

(c) What would a precision of 85% mean?

A. A precision of 85% means that out of all the predicted positive cases (people who the algorithm predicted would pay back the loan), 85% of them are actually positive (i.e., they actually paid back the loan). In other words, of all the loans we approve, 85% pay us back. The remaining 15% (False Positives / FP) of approved loans go into default.

(d) Which measure do you choose to evaluate your model?

A. In this example, we'd prefer precision over recall, as approving a bad loan is more costly than missing out on the profit we could make from a good loan. One measure that would be suitable for this problem is the **receiver operating characteristic (ROC) curve**. This curve plots the true positive rate (TPR) against the false positive rate (FPR) at various threshold values, and is a good way to visualize the trade-off between the two. The **area under the curve (AUC)** is also a popular metric for evaluating machine learning models, and is a good way to compare different models. The AUC ranges from 0 to 1, with a higher AUC indicating a better model.

In the context of loan repayment, a False Positive (FP) would be when the model predicts that a loan will be repaid, but the loan is actually not repaid. A False Negative (FN) would be when the model predicts that a loan will not be repaid, but the loan is actually repaid.

Problem 4

(Random Forest in R) In this assignment, we want to construct machine learning models to predict the price of houses in California based on different predictors such as houses' locations, number of rooms, and the number of people residing in a block using a dataset that was collected in 1990. To do so, download the dataset "housing.csv" from Blackboard. Notice that this dataset contains the price of houses in 1990. Since the housing prices have increased dramatically since then, we cannot generalize our model to predict the the current price of houses in California. The "housing" dataset contains 20640 observations and 10 variables (9 predictors and 1 response).

(a) Loading and Checking the Summary of Data

Q. Load the data set into R and check the summary of data.

A. First, we loaded the data set using the read.csv command and ran the summary function command:

```
library(readr)
housing <- read.csv("C:/housing.csv")
summary(housing)
```

```
##    longitude         latitude     housing_median_age  total_rooms
##  Min.   :-124.3   Min.   :32.54   Min.   : 1.00      Min.   :    2
##  1st Qu.:-121.8   1st Qu.:33.93   1st Qu.:18.00      1st Qu.: 1448
##  Median :-118.5   Median :34.26   Median :29.00      Median : 2127
##  Mean   :-119.6   Mean   :35.63   Mean   :28.64      Mean   : 2636
##  3rd Qu.:-118.0   3rd Qu.:37.71   3rd Qu.:37.00      3rd Qu.: 3148
##  Max.   :-114.3   Max.   :41.95   Max.   :52.00      Max.   :39320
##
##  total_bedrooms     population      households     median_income
##  Min.   :   1.0   Min.   :    3   Min.   :   1.0   Min.   : 0.4999
##  1st Qu.: 296.0   1st Qu.:  787   1st Qu.: 280.0   1st Qu.: 2.5634
##  Median : 435.0   Median : 1166   Median : 409.0   Median : 3.5348
##  Mean   : 537.9   Mean   : 1425   Mean   : 499.5   Mean   : 3.8707
##  3rd Qu.: 647.0   3rd Qu.: 1725   3rd Qu.: 605.0   3rd Qu.: 4.7432
##  Max.   :6445.0   Max.   :35682   Max.   :6082.0   Max.   :15.0001
##  NA's   :207
##  median_house_value ocean_proximity
##  Min.   : 14999     Length:20640
##  1st Qu.:119600     Class :character
##  Median :179700     Mode  :character
##  Mean   :206856
##  3rd Qu.:264725
##  Max.   :500001
##
```

We notice that variable "ocean_proximity" is not numeric and can be converted into a factor, so we do that using the following command:

```
housing$ocean_proximity = as.factor(housing$ocean_proximity)
str(housing)
```

```
## 'data.frame':    20640 obs. of  10 variables:
##  $ longitude         : num  -122 -122 -122 -122 -122 ...
##  $ latitude          : num  37.9 37.9 37.9 37.9 37.9 ...
##  $ housing_median_age: num  41 21 52 52 52 52 52 52 42 52 ...
##  $ total_rooms       : num  880 7099 1467 1274 1627 ...
##  $ total_bedrooms    : num  129 1106 190 235 280 ...
##  $ population        : num  322 2401 496 558 565 ...
##  $ households        : num  126 1138 177 219 259 ...
##  $ median_income     : num  8.33 8.3 7.26 5.64 3.85 ...
##  $ median_house_value: num  452600 358500 352100 341300 342200 ...
##  $ ocean_proximity   : Factor w/ 5 levels "<1H OCEAN","INLAND",..: 4 4 4 4 4 4 4 4 4 4 ...
```

```
summary(housing)
```

```
##    longitude         latitude     housing_median_age  total_rooms
##  Min.   :-124.3   Min.   :32.54   Min.   : 1.00      Min.   :    2
##  1st Qu.:-121.8   1st Qu.:33.93   1st Qu.:18.00      1st Qu.: 1448
##  Median :-118.5   Median :34.26   Median :29.00      Median : 2127
##  Mean   :-119.6   Mean   :35.63   Mean   :28.64      Mean   : 2636
##  3rd Qu.:-118.0   3rd Qu.:37.71   3rd Qu.:37.00      3rd Qu.: 3148
##  Max.   :-114.3   Max.   :41.95   Max.   :52.00      Max.   :39320
##
##  total_bedrooms     population      households     median_income
##  Min.   :   1.0   Min.   :    3   Min.   :   1.0   Min.   : 0.4999
##  1st Qu.: 296.0   1st Qu.:  787   1st Qu.: 280.0   1st Qu.: 2.5634
##  Median : 435.0   Median : 1166   Median : 409.0   Median : 3.5348
##  Mean   : 537.9   Mean   : 1425   Mean   : 499.5   Mean   : 3.8707
##  3rd Qu.: 647.0   3rd Qu.: 1725   3rd Qu.: 605.0   3rd Qu.: 4.7432
##  Max.   :6445.0   Max.   :35682   Max.   :6082.0   Max.   :15.0001
##  NA's   :207
##  median_house_value   ocean_proximity
##  Min.   : 14999     <1H OCEAN :9136
##  1st Qu.:119600     INLAND    :6551
##  Median :179700     ISLAND    :   5
##  Mean   :206856     NEAR BAY  :2290
##  3rd Qu.:264725     NEAR OCEAN:2658
##  Max.   :500001
##
```

(b) Cleaning the Data

Q. Cleaning the data: Are there any outliers or missing values in this dataset? How do you handle them if there are any? Make sure you check both numerical and categorical variables.

A. In the above result, we notice a couple of things, which we need to take care of in order to clean the data set:

1. 207 observations in the variable **"total_bedrooms"** have NA values

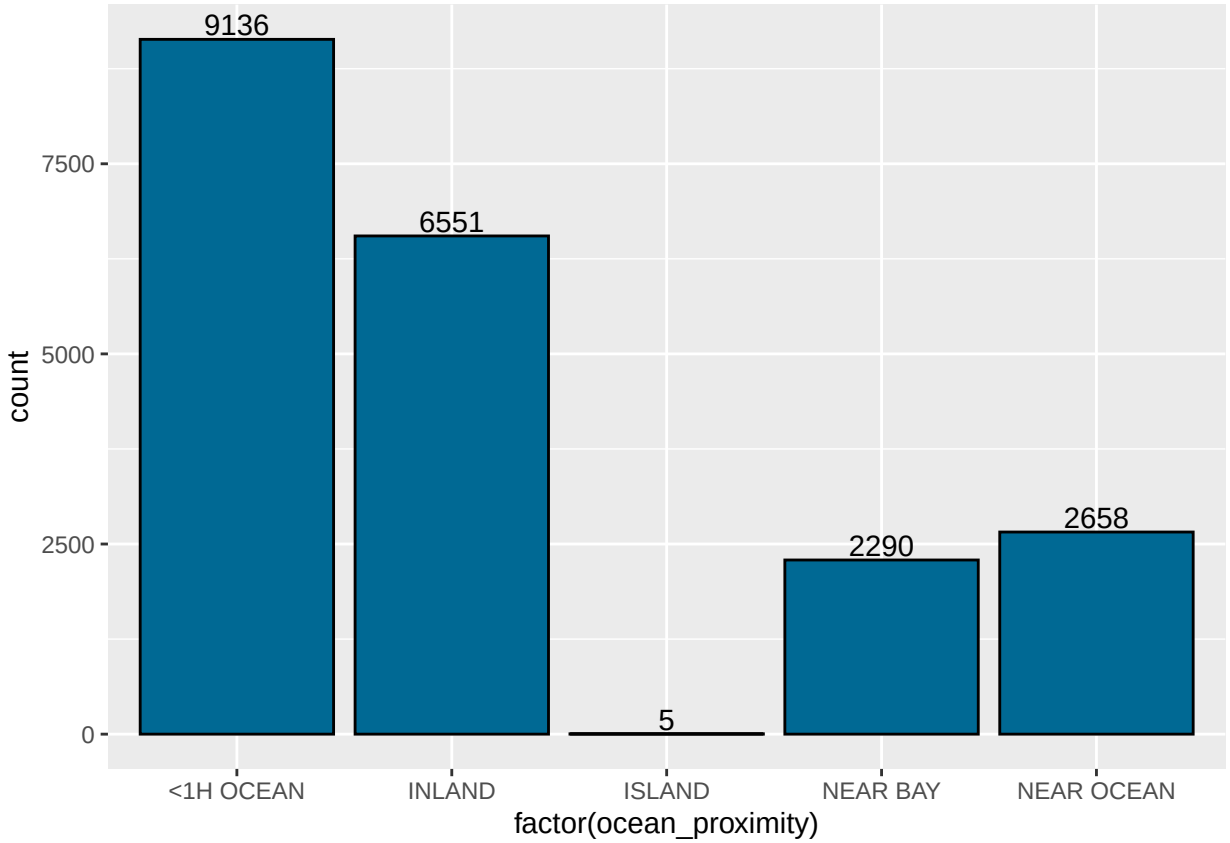
```
sum(is.na(housing))
```

```
## [1] 207
```

2. Number of observations in the variable **"ocean_proximity"**, with value "ISLAND", is only 5, which is very less compared to others:

```
library(ggplot2)
ggplot(housing, aes(x = factor(ocean_proximity))) +
geom_bar(stat = "count", color = "black", fill = "#006994", labels = TRUE) +
geom_text(aes(label = after_stat(count)),
stat = "count", vjust = -0.2, colour = "black")
```

```
## Warning in geom_bar(stat = "count", color = "black", fill = "#006994", labels =
## TRUE): Ignoring unknown parameters: `labels`
```

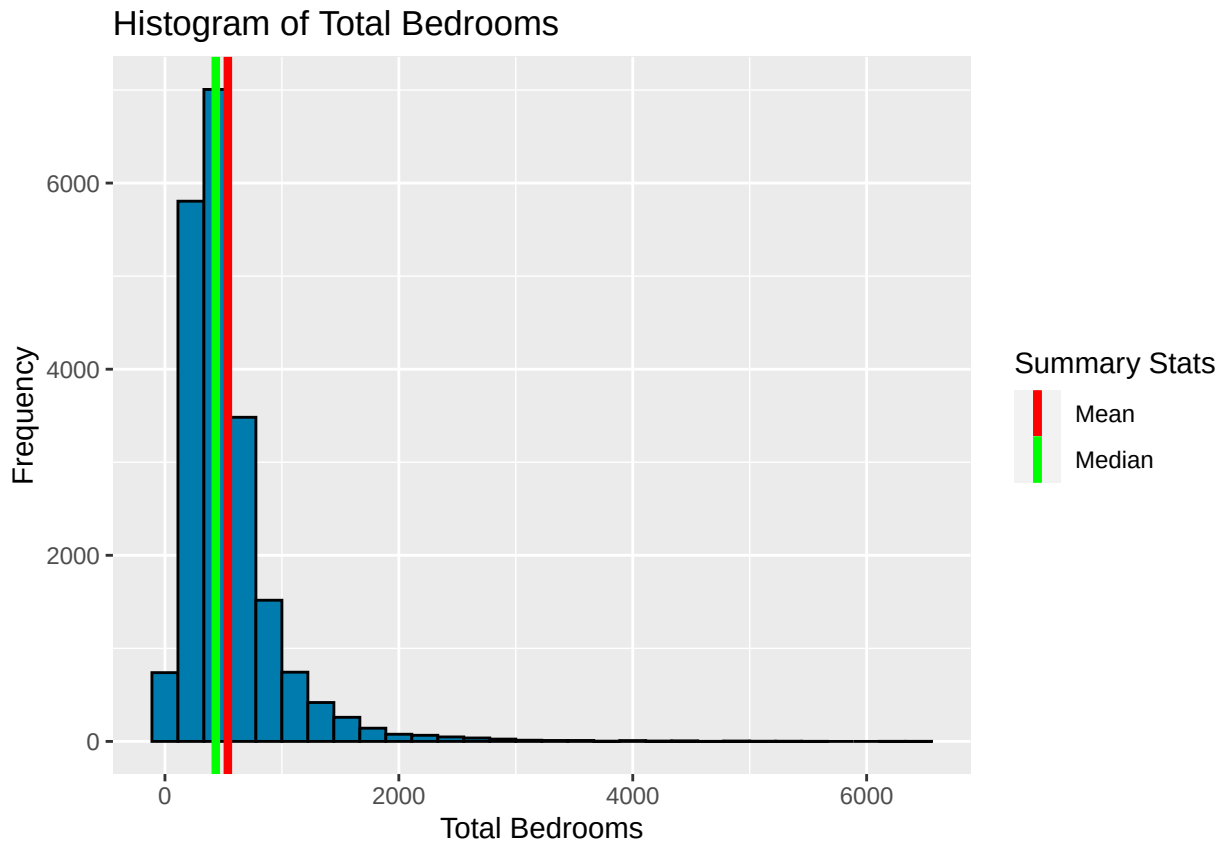


In order to fix these problems, we take the following steps:

1. To solve the **problem of the 207 NA's**, we decided to impute the data points, which we can do by looking at the frequency distribution of this variable, as follows:

```
library(ggplot2)
mean_bedroom <- mean(housing$total_bedrooms, na.rm = TRUE)
median_bedroom <- median(housing$total_bedrooms, na.rm = TRUE)
ggplot(housing, aes(x = total_bedrooms)) +
geom_histogram(bins = 30, color = "black", fill = "#007bae") +
geom_vline(aes(xintercept = mean_bedroom, color = "Mean"), linewidth = 1.5) +
geom_vline(aes(xintercept = median_bedroom, color = "Median"), linewidth = 1.5) +
xlab("Total Bedrooms") +
ylab("Frequency") +
ggtitle("Histogram of Total Bedrooms") +
scale_color_manual(name = "Summary Stats", labels = c("Mean", "Median"),
values = c("red", "green"))
```

```
## Warning: Removed 207 rows containing non-finite values (`stat_bin()`).
```



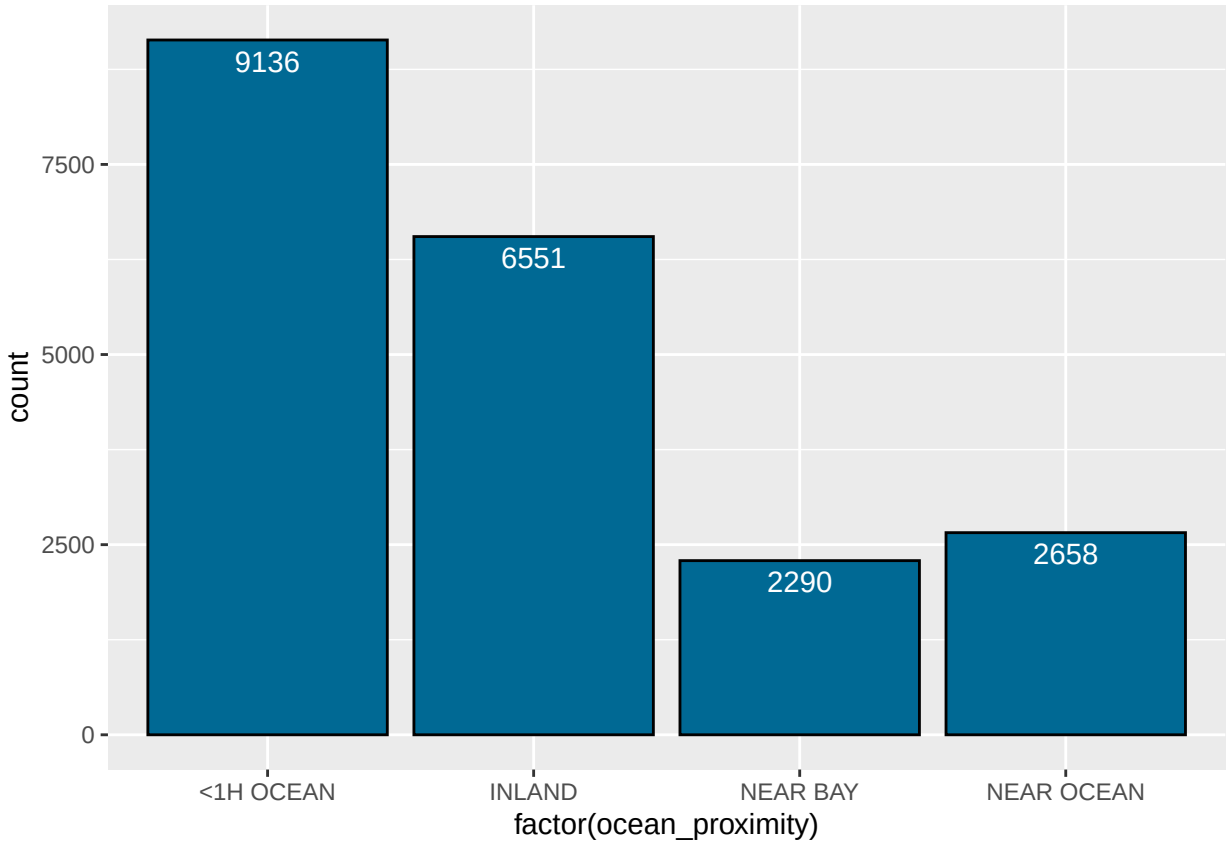
Looking at the above data, we decided to take median of the **total_bedrooms** variables for our data point imputation:

```
housing$total_bedrooms[is.na(housing$total_bedrooms)] = median_bedroom
sum(is.na(housing))
```

```
## [1] 0
```

2. To solve the problem of the observations with a value of "ISLAND" in the **ocean_proximity"**, we simply exclude this level from the variable, as it could lead to more issues with model fitting later. This can be done as follows:

```
housing <- housing[housing$ocean_proximity != "ISLAND", ]
ggplot(housing, aes(x = factor(ocean_proximity))) +
geom_bar(stat = "count", color = "black", fill = "#006994") +
geom_text(aes(label = after_stat(count)),
stat = "count", vjust = 1.5, colour = "white")
```

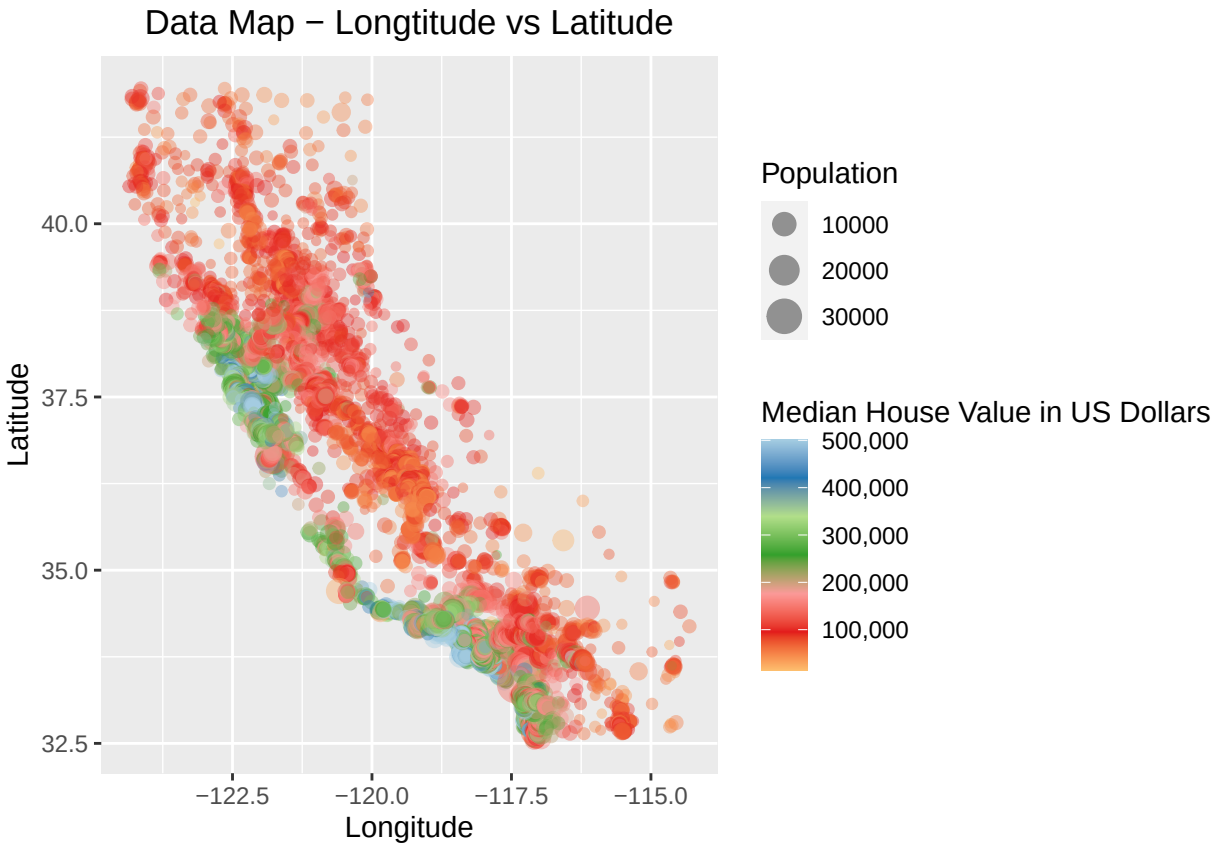


(c) Plot to Visualize all Data Points (ggplot)

Q. Use the ggplot() function to visualize all data points in the data set with longitude on the x-axis, latitude on the y-axis, and median_house_value in shown in color codes. Change the size of the points represented in your plot based on the size of the population. Do not forget to add a title, legend, and label for each axis. What do you observe from this plot?

A. To get a visual representation of all the data points in the data set with **longitude** on the X-axis, **latitude** on the y-axis, and the **median_house_value** shown in color, here's the code:

```
plot = ggplot(housing,
aes(x = longitude, y = latitude, color = median_house_value)) +
geom_point(aes(size = population), alpha = 0.4) +
xlab("Longitude") +
ylab("Latitude") +
ggtitle("Data Map - Longtitude vs Latitude") +
theme(plot.title = element_text(hjust = 0.5)) +
scale_color_distiller(palette = "Paired", labels=scales::comma) +
labs(color = "Median House Value in US Dollars", size = "Population")

plot
```

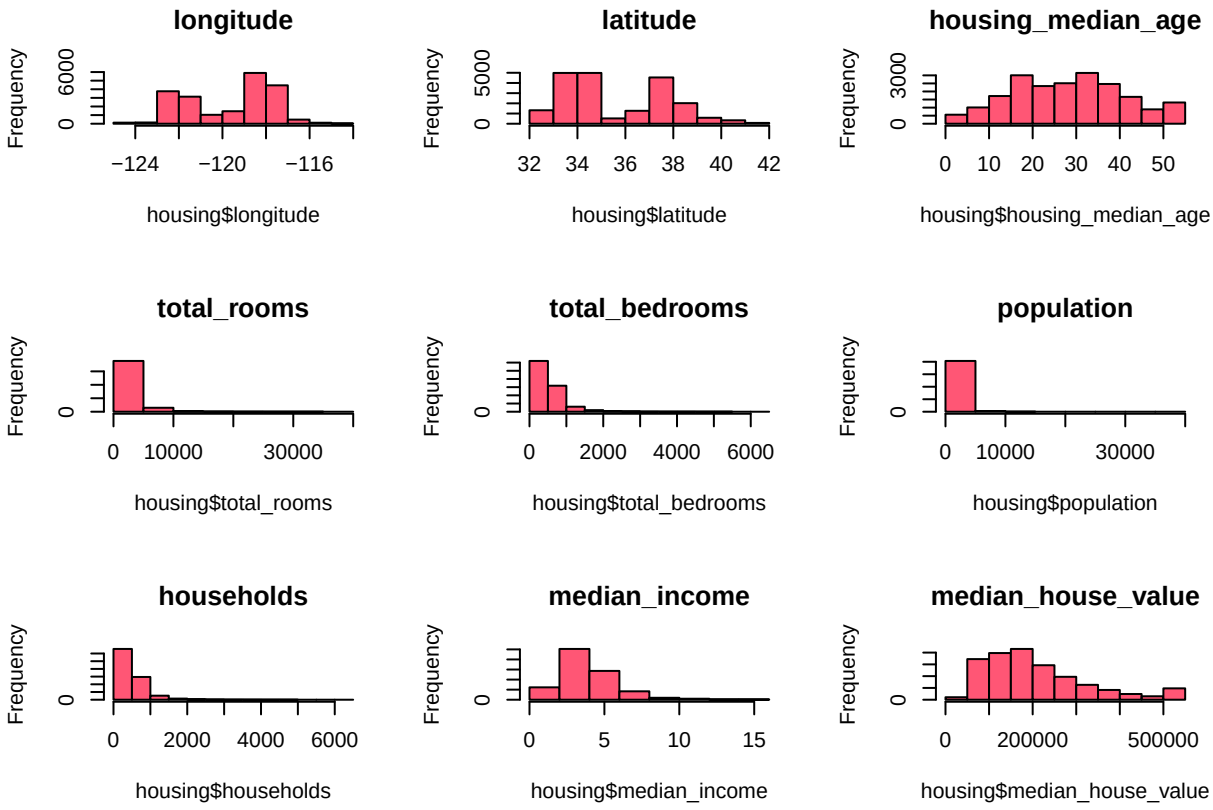


(d) Distribution of Numeric Variables

Q. For a better sense of the distribution of the nine numeric variables, Look at histograms for each of them and describe your observation. Add all these histograms in one plot using par(mfrow = c(3, 3)).

A. In order to get a distribution of the nine numerical variables using histograms, we can use the hist function and to display them side by side, we can use the par(mfrow..) function as follows:

```
options(scipen = 5) # to remove scientific notations
par(mfrow = c(3,3))
hist(housing$longitude, main = "longitude",
breaks = 10, col = "#FF5675", border = "black")

hist(housing$latitude, main = "latitude",
breaks = 10, col = "#FF5675", border = "black")

hist(housing$housing_median_age, main = "housing_median_age",
breaks = 10, col = "#FF5675", border = "black")

hist(housing$total_rooms, main = "total_rooms",
breaks = 10, col = "#FF5675", border = "black")

hist(housing$total_bedrooms, main = "total_bedrooms",
breaks = 10, col = "#FF5675", border = "black")

hist(housing$population, main = "population",
breaks = 10, col = "#FF5675", border = "black")

hist(housing$households, main = "households",
breaks = 10, col = "#FF5675", border = "black")

hist(housing$median_income, main = "median_income",
breaks = 10, col = "#FF5675", border = "black")

hist(housing$median_house_value, main = "median_house_value",
breaks = 10, col = "#FF5675", border = "black")
```

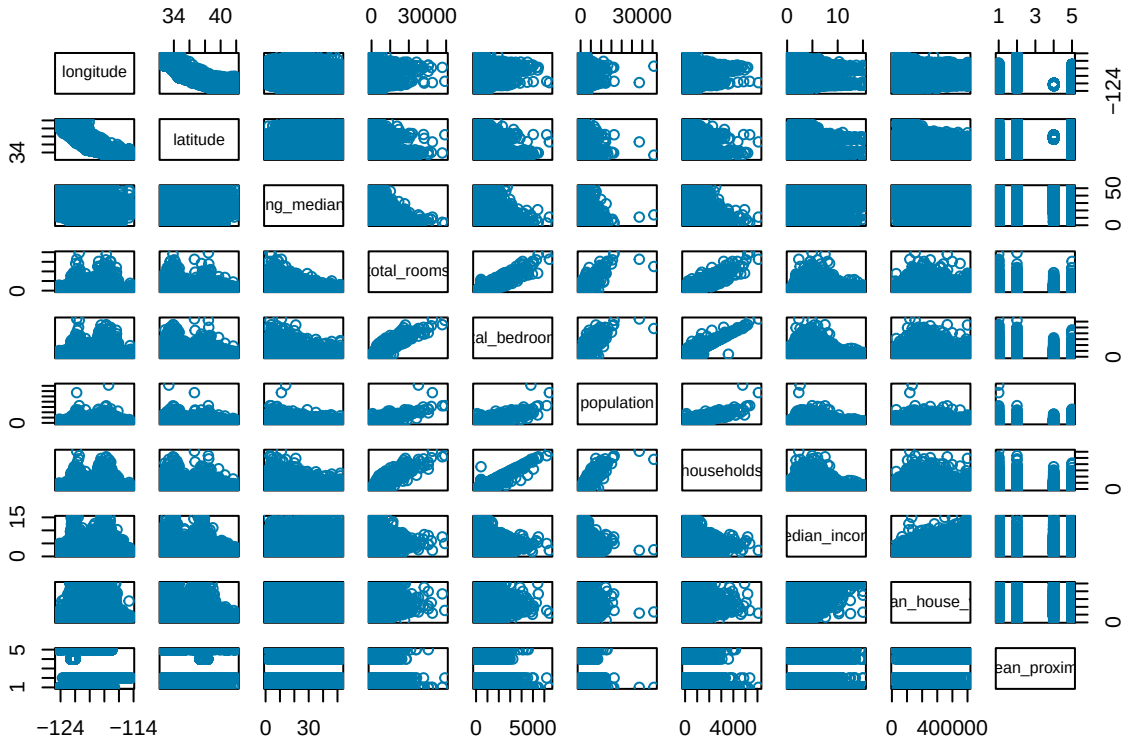

longitude
Frequency
0
6000
−124
−120
−116
housing$longitude
latitude
Frequency
0
5000
32
34
36
38
40
42
housing$latitude
housing_median_age
Frequency
0
3000
0
10
20
30
40
50
housing$housing_median_age
total_rooms
Frequency
0
0
10000
30000
housing$total_rooms
total_bedrooms
Frequency
0
0
2000
4000
6000
housing$total_bedrooms
population
Frequency
0
0
10000
30000
housing$population
households
Frequency
0
0
2000
4000
6000
housing$households
median_income
Frequency
0
0
5
10
15
housing$median_income
median_house_value
Frequency
0
0
200000
500000
housing$median_house_value

(e) Relationship between Variables (Pairs function)

Q. Look at the relationship between all variables using the pair() function. Describe any noteworthy observations.

A. In order to see how each variable is related to each other, we use the pairs() function as follows:

```
par(mfrow = c(1, 1))
pairs(housing, col = "#007bae")
```



Observation

We see the relationship between all variables, but also notice that **"ocean_proximity"** doesn't add much to the chart due to collinearity, so we should get rid of that and only keep the relationship between the numeric variables. Secondly, we can also observe in the graphs for **household**, **total_rooms**, and **total_bedrooms** variables that there might be some relationship between them. However, we can't suggest that yet unless we look at the correlation between the variables.

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
longitude	1.00	-0.92	-0.11	0.04	0.07	0.10	0.06	-0.02	-0.05
latitude	-0.92	1.00	0.01	-0.04	-0.07	-0.11	-0.07	-0.08	-0.14
housing_median_age	-0.11	0.01	1.00	-0.36	-0.32	-0.30	-0.30	-0.12	0.11
total_rooms	0.04	-0.04	-0.36	1.00	0.93	0.86	0.92	0.20	0.13
total_bedrooms	0.07	-0.07	-0.32	0.93	1.00	0.87	0.97	-0.01	0.05
population	0.10	-0.11	-0.30	0.86	0.87	1.00	0.91	0.00	-0.02
households	0.06	-0.07	-0.30	0.92	0.97	0.91	1.00	0.01	0.07
median_income	-0.02	-0.08	-0.12	0.20	-0.01	0.00	0.01	1.00	0.69
median_house_value	-0.05	-0.14	0.11	0.13	0.05	-0.02	0.07	0.69	1.00

(f) Correlation between Variables

Q. Check the correlation between your variables. Are there any highly correlated variables? If yes, how do you handle them?

A. In order to look into the correlation between all variables, we made use of the cor() function. However, before we do that, we remove *"ocean_proximity"* variable from the considerations as mentioned in the previous section. Lastly, we display the result using the kable() function.

```
library(knitr)
library(kableExtra)
correlation <- round(cor(housing[, 1:9]), digits = 2)

kable(correlation,
      format="latex", booktabs=TRUE) %>%
  kable_styling(latex_options = c("striped", "scale_down")) %>%
  column_spec(1,width = "1in") %>%
  column_spec(2,width = "1in") %>%
  column_spec(3,width = "1in") %>%
  column_spec(4,width = "1in") %>%
  column_spec(5,width = "1in") %>%
  column_spec(6,width = "1in") %>%
  column_spec(7,width = "1in") %>%
  column_spec(8,width = "1in") %>%
  column_spec(9,width = "1in")
```

In order to fit the whole table on one page, we had to use the kableExtra package and scale down the whole thing

Correlation Observations

Based on the result below, we can see that there is high collinearity between **housholds** and **total_bedrooms** variables, as well as between **households** and **total_rooms** variables, which confirms our *theory* above. Even though they're highly collinear, it's best to leave them in the data set, as they could be influential when determining the price of a home. This can cause issues with multicollinearity, but we can handle that later if we run into an issue.

(g) Train & Test Data

Q. Partition your data into a train and a test data.

A. In order to divide our data into train and test data, we first set the seed value to 1, which would give us an equal split each time. Secondly, we're choosing 70% and 30% split into train and test data respectively. Also, since we'll be creating two different models in this exercise, we copy the contents of the housing data set into a clone called **housing_rf**. To do this, we use the following code below:

```
set.seed(1)
index_rf <- sample(2, nrow(housing), replace= TRUE, prob = c(0.75, 0.25))
train_rf <- housing[index_rf == 1,  ]
test_rf <- housing[index_rf == 2, ]
```

(h) Random Forest Model

Q. Use your training data to construct a random forest model.

A. Before we create a Random Forest model, we need to consider the important data point for our model, so we consider median_house_value as that variable, which could be our target variable. Since this is not converted into a categorical variable, the randomForest method assumes it's a regression model and provides information accordingly.

```
library(randomForest)
rf_housing <- randomForest(median_house_value ~ ., data = train_rf,
ntree = 500, mtry = 4, proximity = T, importance = T)
```

In the above model, we're considering **ntree** to be 500, as it denotes the number of trees considered in the Random Forest, so we're essentially creating a random forest model with 500 trees. **mtry** indicates the number of variables randomly sampled as candidates at each split. For our model, we're taking **mtry** to be 4. To get the proximity matrix and important variables suggested by this model, we've set proximity and importance to be true.

(i) Model's Performance on Training Examples

Q. How does your model perform on the training examples? Is a linear model a good model to predict the California housing prices? Justify your answer.

A. Model's performance on the training set can be viwed using the following codes:

```
print(rf_housing)
```

Call: randomForest(formula = median_house_value ~ ., data = train_rf, ntree = 500, mtry = 4, proximity = T, importance = T) Type of random forest: regression Number of trees: 500 No. of variables tried at each split: 4

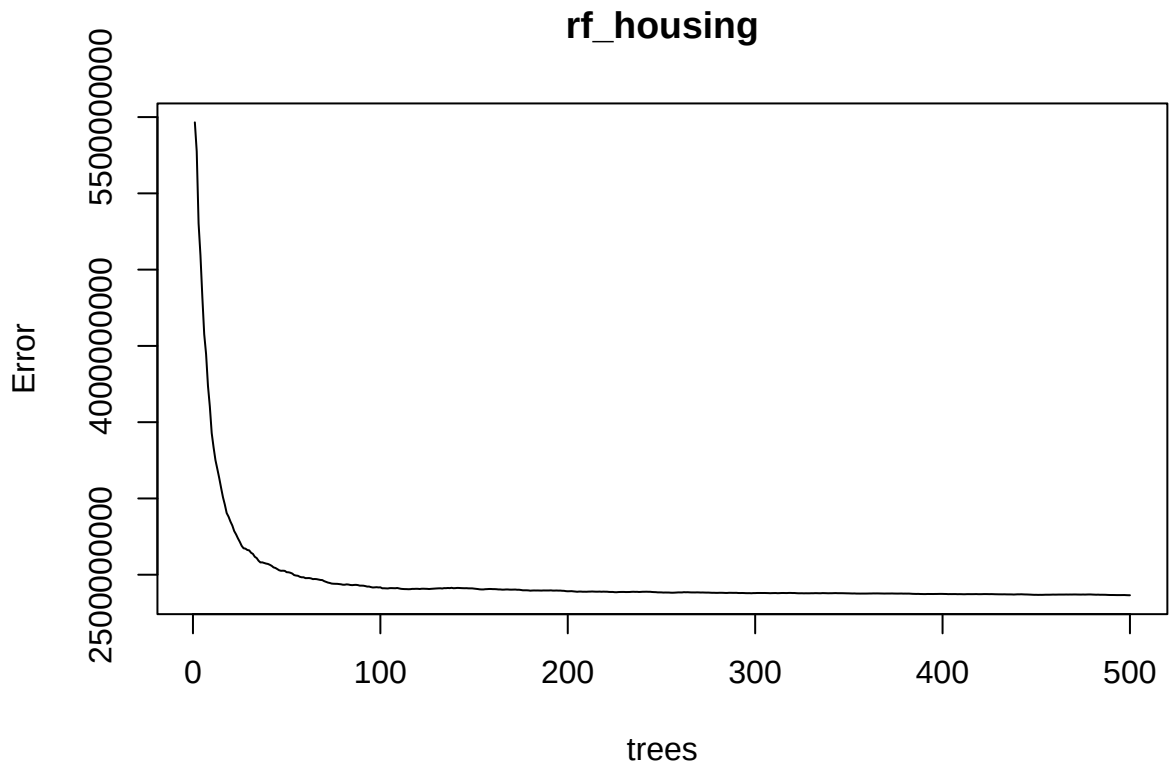
```
      Mean of squared residuals: 2365516853
                % Var explained: 82.17
```

```
names(rf_housing)
```

[1] "call" "type" "predicted" "mse"
[5] "rsq" "oob.times" "importance" "importanceSD"
[9] "localImportance" "proximity" "ntree" "mtry"
[13] "forest" "coefs" "y" "test"
[17] "inbag" "terms"

We plot the error rates with various number of trees:

```
plot(rf_housing)
```



Finally, we obtain the predicted classes and the Mean Squared Error (MSE) on the training data set using the following codes:

```
head(rf_housing$predicted, n=50)
```

```
##        1        2        3        5        8        9       10       11
## 430981.8 421656.0 406778.6 245017.1 254792.4 173903.1 261033.5 219122.1
##       12       13       14       16       17       19       22       23
## 275801.9 238244.2 157729.4 135842.1 199125.4 136102.0 132582.1 176608.3
##       24       25       26       27       28       30       31       32
## 136580.5 180886.7 126088.6 141841.4 126767.9 119845.8 133263.6 147086.9
##       33       34       36       38       39       40       42       44
## 124484.5 125334.9 123762.3 120485.5 241024.9 197583.6 132285.2 212538.7
##       45       47       48       49       50       51       53       54
## 220460.0 137762.6 127604.8 128319.6 137663.7 165551.8 198744.2 146262.4
##       55       56       57       58       59       60       62       63
## 104806.8 102050.9 111604.0 129200.0 107016.2 200246.9 320114.2 106129.9
##       64       65
## 111894.3 140864.2
```

```
head(rf_housing$mse, n = 50)
```

```
##  [1] 5465497944 5275433460 4802613607 4594271352 4319894007 4076552711
##  [7] 3946759578 3737747661 3602630077 3428452698 3335376235 3250708021
## [13] 3193366526 3134539732 3069756159 3008055666 2959775198 2904128671
## [19] 2880152934 2848490483 2822627914 2786600753 2764427409 2738382026
## [25] 2715453957 2689002809 2673281920 2671045108 2662370447 2659745632
## [31] 2644077772 2636377550 2617722409 2609122603 2591615451 2580409785
## [37] 2580967376 2577006800 2573207364 2570600484 2565049049 2558345684
## [43] 2548090567 2544214041 2536928921 2529315403 2525844085 2526435010
## [49] 2525190677 2516684509
```

(j) Variables Considered Important for Random Forest Model

Q. What variables are considered important for your predictions using your random forest model?

A. We obtain the importance of the variables using the importance() and varImpPlot() methods as follows:

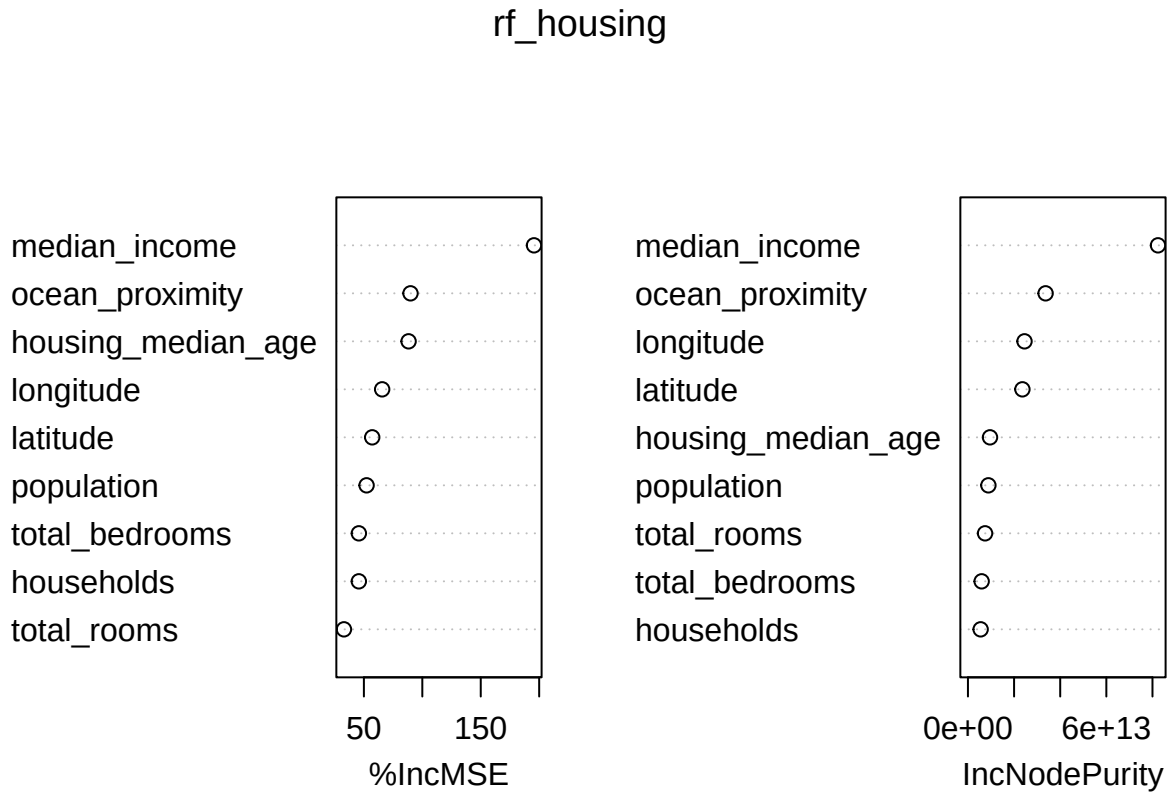
```
importance(rf_housing, type = 1)
```

```
##                     %IncMSE
## longitude           65.62403
## latitude            57.13190
## housing_median_age  88.25347
## total_rooms         32.93655
## total_bedrooms      45.69175
## population          52.30074
## households          45.67421
## median_income      195.58159
## ocean_proximity     89.89760
```

```
importance(rf_housing, type = 2)
```

```
##                     IncNodePurity
## longitude          24515735717933
## latitude           23605988168326
## housing_median_age  9562706390230
## total_rooms         7424332039763
## total_bedrooms      5936218406015
## population          8859552460364
## households          5497380222873
## median_income      82394813988373
## ocean_proximity    33661271977317
```

```
varImpPlot(rf_housing)
```



%IncMSE shows how much our model's accuracy will decrease if we leave out a variable. IncNodePurity is a measure of variable's importance based on the Gini impurity index used for calculating the splits in the trees. The higher the value of mean decrease accuracy or mean decrease gini score, the higher the importance of the variable to our model. Per the plots, we can see that **median_income** is the most important variable in our random forest model, as it's got the highest value in both the %IncMSE chart (type = 1) and the IncNodePurity chart (type = 2). After that, it's **ocean_proximity**, which also makes sense as we can determine the median house value based on a person's median income (as in the ability to buy) and the proximity to the ocean, which directly impacts the cost. We also see that **housing_median_age** comes in at 3rd position with 91% %IncMSE, but on the second factor, IncNodePurity, longitude comes in at 3rd position.

(k) Performance of Random Forest Model on Test Instance

Q. How is the performance of your model on the test instances?

A. To test the performance of our Random Forest Model, we perform the following calculations:

```
pred_rf <- predict(rf_housing, newdata = test_rf)
head(pred_rf, n=50)
```

```
##        4        6        7       15       18       20       21       29
## 350032.0 208981.8 253238.8 190231.0 142920.0 178672.4 132443.7 138192.4
##       35       37       41       43       46       52       61       68
## 199002.2 123991.5 129835.3 137300.7 197876.6 130554.4 206987.7 171282.9
##       70       72       76       77       79       80       85       94
## 146686.2 116608.8 161347.5 169554.4 132746.2 130806.7 120264.0 176834.8
##       95       96       99      104      109      111      121      125
## 159815.2 238442.0 220850.7 215569.9 270722.4 315793.4 365583.4 367994.3
##      135      139      150      162      164      165      172      173
## 415049.0 355926.0 269441.5 244996.9 264574.1 245599.4 128475.7 161192.0
##      176      180      183      185      187      189      194      198
## 127667.3 202613.8 132795.6 140366.4 199931.6 125873.9 119869.6 153084.5
##      200      211
## 147148.1 180917.2
```

Finally, we also check a couple of parameters and how they perform against our Random Forest Model, one is called Mean Absolute Error (MAE) and the other is Mean Squared Error (MSE). MAE is the average distance between the real data and the predicted data. MSE measures the average squared difference between the estimated values and the actual value. The main draw for using MSE is that it squares the error, which results in large errors being clearly highlighted.

```
true_value_rf <- test_rf$median_house_value
```

Mean Absolute Error (MAE) of this model is:

```
library(Metrics)
library(caret)
MAE(pred_rf, test_rf$median_house_value)
```

```
## [1] 31641.15
```

Mean Squared Error (MSE) of this model is:

```
mse_rf <- mean((pred_rf - true_value_rf)^2)
print(mse_rf)
```

```
## [1] 2260361799
```

The value of MSE is high, which isn't ideal for our model.

(l) Decision Tree Model

Q. Construct a decision tree model to predict the median_house_value using your training data. Play with the pruning parameters to check how they affect the performance of your decision tree model. Justify your final choice of these parameters.

A. To construct the decision tree model, we first divide our dataset into train and test data like we did above with the Random Forest model:

```
set.seed(1)
index_dt <- sample(2, nrow(housing), replace= TRUE, prob = c(0.75, 0.25))
train_dt <- housing[index_dt == 1,  ]
test_dt <- housing[index_dt == 2, ]
```

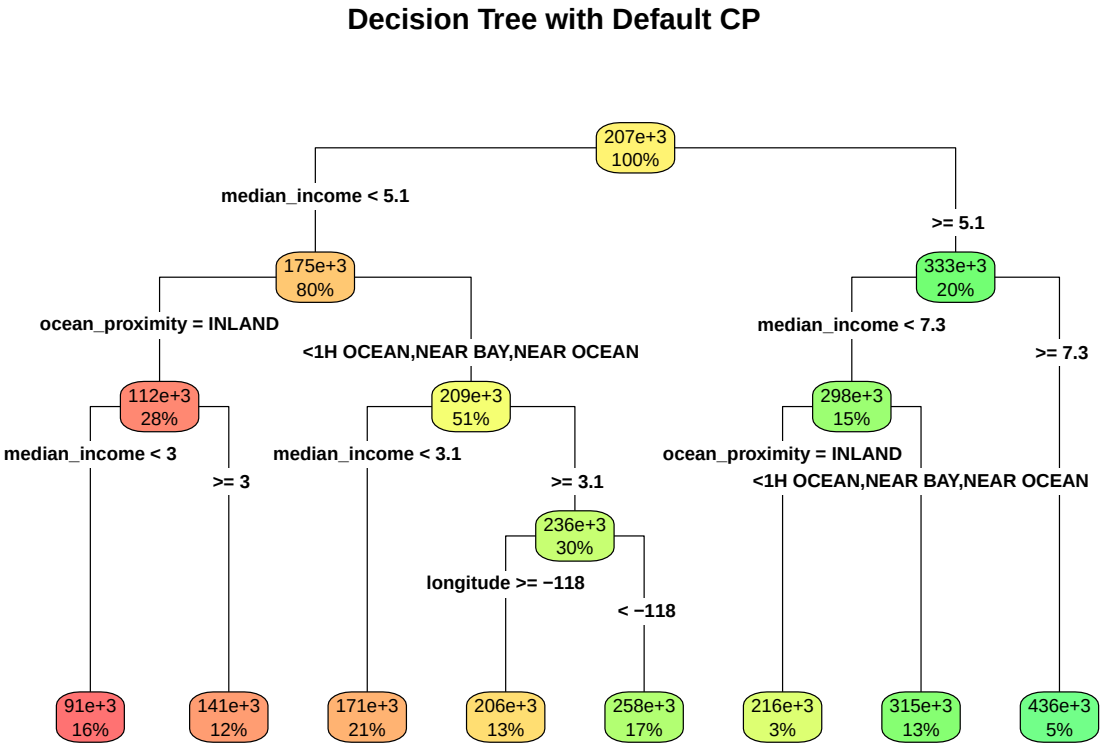
After this division, we build a decision tree model using the "rpart" function from "rpart" package. The first and second arguments to the rpart function are respectively the formula and training data. We're using default **cp** of 0.01, which means any split that does not reduce the tree's overall complexity by a factor of 0.01, is not attempted. The code for that is as follows:

```
library(rpart)
dt_model <- rpart(median_house_value ~ ., train_dt)
print(dt_model)
```

```
## n= 15437
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 15437 204836000000000 206814.7
##    2) median_income< 5.08625 12301 103480100000000 174633.8
##      4) ocean_proximity=INLAND 4399  12495580000000 112416.8
##        8) median_income< 3.0366 2508   3929457000000  91043.7 *
##        9) median_income>=3.0366 1891   5900950000000 140763.6 *
##      5) ocean_proximity=<1H OCEAN,NEAR BAY,NEAR OCEAN 7902  64476540000000 209269.7
##       10) median_income< 3.07625 3267  20091420000000 171397.8 *
##       11) median_income>=3.07625 4635  36396530000000 235963.9
##         22) longitude>=-118.275 1940   9620032000000 205555.6 *
##         23) longitude< -118.275 2695  23691350000000 257853.3 *
##    3) median_income>=5.08625 3136  38647910000000 333044.6
##      6) median_income< 7.26075 2350  22245550000000 298442.5
##       12) ocean_proximity=INLAND 403   2532496000000 216487.1 *
##       13) ocean_proximity=<1H OCEAN,NEAR BAY,NEAR OCEAN 1947  16445960000000 315406.0 *
##      7) median_income>=7.26075 786   5176305000000 436498.8 *
```

To plot an rpart decision tree we can use the "rpart.plot()" function from "rpart.plot" package:

```
library(rpart.plot)
rpart.plot(dt_model, box.palette = "RdYlGn", type = 4,
main = "Decision Tree with Default CP")
```




```
rpart.rules(dt_model)
```

median_house_value
91044 when median_income < 3.0 & ocean_proximity is INLAND
140764 when median_income is 3.0 to 5.1 & ocean_proximity is INLAND
171398 when median_income < 3.1 & ocean_proximity is <1H OCEAN or NEAR BAY or NEAR OCEAN
205556 when median_income is 3.1 to 5.1 & ocean_proximity is <1H OCEAN or NEAR BAY or NEAR OCEAN & longitude >= -118 216487 when median_income is 5.1 to 7.3 & ocean_proximity is INLAND
257853 when median_income is 3.1 to 5.1 & ocean_proximity is <1H OCEAN or NEAR BAY or NEAR OCEAN & longitude < -118 315406 when median_income is 5.1 to 7.3 & ocean_proximity is <1H OCEAN or NEAR BAY or NEAR OCEAN
436499 when median_income >= 7.3

Summary of the default model is:

```
summary(dt_model)
```

```
## Call:
## rpart(formula = median_house_value ~ ., data = train_dt)
##   n= 15437
##
##           CP nsplit rel error    xerror        xstd
## 1 0.30613759      0 1.0000000 1.0001272 0.012306829
## 2 0.12941054      1 0.6938624 0.7004942 0.009653862
## 3 0.05480511      2 0.5644519 0.5696249 0.008253639
## 4 0.03899996      3 0.5096468 0.5160899 0.008148390
## 5 0.01594980      4 0.4706468 0.4797456 0.007821308
## 6 0.01506155      5 0.4546970 0.4664401 0.007722742
## 7 0.01301127      6 0.4396354 0.4528101 0.007586472
## 8 0.01000000      7 0.4266242 0.4376297 0.007478092
##
## Variable importance
##      median_income    ocean_proximity           latitude          longitude
##                 63                 22                  7                  4
## housing_median_age        total_rooms         population         households
##                  2                  1                  1                  1
##
## Node number 1: 15437 observations,    complexity param=0.3061376
##   mean=206814.7, MSE=1.326916e+10
##   left son=2 (12301 obs) right son=3 (3136 obs)
##   Primary splits:
##       median_income   < 5.08625  to the left,  improve=0.30613760, (0 missing)
##       ocean_proximity splits as  RL-RR,        improve=0.23553770, (0 missing)
##       latitude        < 37.925   to the right, improve=0.06640784, (0 missing)
##       longitude       < -121.865 to the right, improve=0.03836320, (0 missing)
##       total_rooms     < 1944.5   to the left,  improve=0.02805858, (0 missing)
##   Surrogate splits:
##       total_rooms < 14091.5  to the left,  agree=0.798, adj=0.004, (0 split)
##
## Node number 2: 12301 observations,    complexity param=0.1294105
##   mean=174633.8, MSE=8.412329e+09
##   left son=4 (4399 obs) right son=5 (7902 obs)
##   Primary splits:
##       ocean_proximity    splits as  RL-RR,        improve=0.25616460, (0 missing)
##       median_income      < 3.07055  to the left,  improve=0.15825560, (0 missing)
##       latitude           < 37.915   to the right, improve=0.06165936, (0 missing)
##       longitude          < -121.865 to the right, improve=0.03782196, (0 missing)
##       housing_median_age < 51.5     to the left,  improve=0.03606145, (0 missing)
##   Surrogate splits:
##       latitude           < 37.955   to the right, agree=0.732, adj=0.250, (0 split)
##       housing_median_age < 15.5     to the left,  agree=0.668, adj=0.073, (0 split)
##       longitude          < -116.905 to the right, agree=0.664, adj=0.060, (0 split)
##       population         < 338.5    to the left,  agree=0.651, adj=0.024, (0 split)
##       households         < 129.5    to the left,  agree=0.651, adj=0.024, (0 split)
##
## Node number 3: 3136 observations,    complexity param=0.05480511
##   mean=333044.6, MSE=1.232395e+10
```

```
##   left son=6 (2350 obs) right son=7 (786 obs)
##   Primary splits:
##       median_income      < 7.26075  to the left,  improve=0.29047000, (0 missing)
##       ocean_proximity    splits as  RL-RR,        improve=0.13743780, (0 missing)
##       housing_median_age < 27.5     to the left,  improve=0.09792600, (0 missing)
##       latitude           < 37.965   to the right, improve=0.05485402, (0 missing)
##       longitude          < -117.735 to the right, improve=0.04145480, (0 missing)
##   Surrogate splits:
##       population     < 64       to the right, agree=0.752, adj=0.011, (0 split)
##       total_rooms    < 208      to the right, agree=0.752, adj=0.009, (0 split)
##       total_bedrooms < 25.5     to the right, agree=0.751, adj=0.008, (0 split)
##       households     < 18.5     to the right, agree=0.751, adj=0.008, (0 split)
##
## Node number 4: 4399 observations,    complexity param=0.01301127
##   mean=112416.8, MSE=2.840551e+09
##   left son=8 (2508 obs) right son=9 (1891 obs)
##   Primary splits:
##       median_income      < 3.0366   to the left,  improve=0.21328940, (0 missing)
##       latitude           < 34.825   to the right, improve=0.03347188, (0 missing)
##       total_rooms        < 2336.5   to the left,  improve=0.02652757, (0 missing)
##       longitude          < -118.325 to the left,  improve=0.02470675, (0 missing)
##       housing_median_age < 14.5     to the right, improve=0.02234194, (0 missing)
##   Surrogate splits:
##       housing_median_age < 14.5     to the right, agree=0.628, adj=0.135, (0 split)
##       total_rooms        < 2830.5   to the left,  agree=0.621, adj=0.119, (0 split)
##       population         < 2149.5   to the left,  agree=0.592, adj=0.051, (0 split)
##       households         < 881.5    to the left,  agree=0.587, adj=0.040, (0 split)
##       total_bedrooms     < 890.5    to the left,  agree=0.586, adj=0.038, (0 split)
##
## Node number 5: 7902 observations,    complexity param=0.03899996
##   mean=209269.7, MSE=8.159522e+09
##   left son=10 (3267 obs) right son=11 (4635 obs)
##   Primary splits:
##       median_income      < 3.07625  to the left,  improve=0.12389920, (0 missing)
##       longitude          < -118.305 to the right, improve=0.07329892, (0 missing)
##       latitude           < 33.995   to the left,  improve=0.03821639, (0 missing)
##       housing_median_age < 50.5     to the left,  improve=0.03652235, (0 missing)
##       total_rooms        < 2032     to the left,  improve=0.03197246, (0 missing)
##   Surrogate splits:
##       total_rooms < 1111.5   to the left,  agree=0.617, adj=0.073, (0 split)
##       latitude    < 32.775   to the left,  agree=0.602, adj=0.037, (0 split)
##       longitude   < -123.045 to the left,  agree=0.596, adj=0.024, (0 split)
##       households  < 132.5    to the left,  agree=0.592, adj=0.013, (0 split)
##       population  < 319.5    to the left,  agree=0.591, adj=0.010, (0 split)
##
## Node number 6: 2350 observations,    complexity param=0.0159498
##   mean=298442.5, MSE=9.466191e+09
##   left son=12 (403 obs) right son=13 (1947 obs)
##   Primary splits:
##       ocean_proximity    splits as  RL-RR,        improve=0.14686500, (0 missing)
##       housing_median_age < 27.5     to the left,  improve=0.11571190, (0 missing)
##       median_income      < 6.27875  to the left,  improve=0.09079596, (0 missing)
##       latitude           < 37.955   to the right, improve=0.04758064, (0 missing)
##       longitude          < -118.025 to the right, improve=0.04194649, (0 missing)
```

```
##   Surrogate splits:
##       latitude           < 38.335   to the right, agree=0.861, adj=0.191, (0 split)
##       longitude          < -116.605 to the right, agree=0.832, adj=0.022, (0 split)
##       housing_median_age < 2.5      to the left,  agree=0.831, adj=0.017, (0 split)
##
## Node number 7: 786 observations
##   mean=436498.8, MSE=6.58563e+09
##
## Node number 8: 2508 observations
##   mean=91043.7, MSE=1.566769e+09
##
## Node number 9: 1891 observations
##   mean=140763.6, MSE=3.120545e+09
##
## Node number 10: 3267 observations
##   mean=171397.8, MSE=6.149807e+09
##
## Node number 11: 4635 observations,    complexity param=0.01506155
##   mean=235963.9, MSE=7.852542e+09
##   left son=22 (1940 obs) right son=23 (2695 obs)
##   Primary splits:
##       longitude          < -118.275 to the right, improve=0.08476486, (0 missing)
##       housing_median_age < 47.5     to the left,  improve=0.07032831, (0 missing)
##       latitude           < 33.985   to the left,  improve=0.04678097, (0 missing)
##       ocean_proximity    splits as  L--RR,        improve=0.02732555, (0 missing)
##       total_bedrooms     < 407.5    to the left,  improve=0.02711282, (0 missing)
##   Surrogate splits:
##       latitude           < 34.145   to the left,  agree=0.887, adj=0.729, (0 split)
##       ocean_proximity    splits as  L--RR,        agree=0.603, adj=0.051, (0 split)
##       housing_median_age < 7.5      to the left,  agree=0.591, adj=0.024, (0 split)
##       population         < 2982     to the right, agree=0.584, adj=0.006, (0 split)
##       total_rooms        < 11473.5  to the right, agree=0.583, adj=0.004, (0 split)
##
## Node number 12: 403 observations
##   mean=216487.1, MSE=6.284108e+09
##
## Node number 13: 1947 observations
##   mean=315406, MSE=8.44682e+09
##
## Node number 22: 1940 observations
##   mean=205555.6, MSE=4.958779e+09
##
## Node number 23: 2695 observations
##   mean=257853.3, MSE=8.790854e+09
```

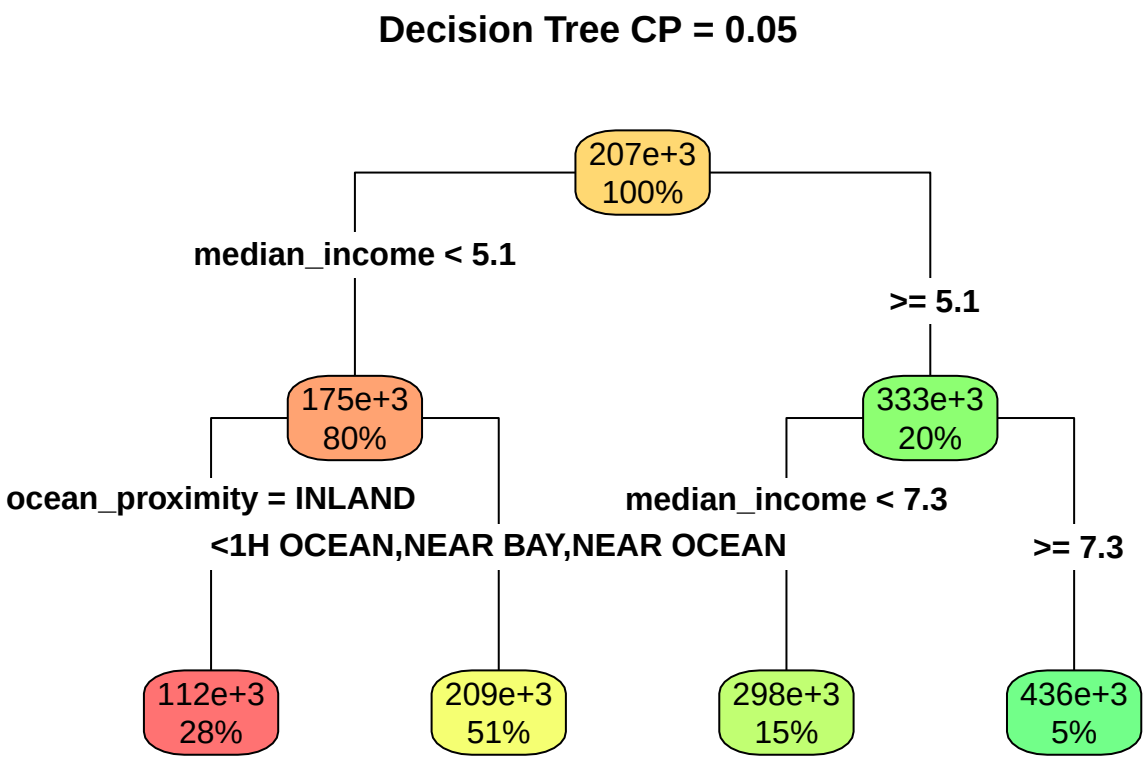
Model with split = "information", cp = 0.05

In the above section, we used a default cp of 0.01, but if we try to run this model with different pruning parameters like cp=0.05, split = information, etc., we're seeing different results as follows:

```
dt_model1 <- rpart(median_house_value ~ ., train_dt,
parms = list(split = "information"),
control = rpart.control(minbucket = 3, minsplit = 5, cp = 0.05))
print(dt_model1)
```

```
## n= 15437
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
## 1) root 15437 204836000000000 206814.7
##   2) median_income< 5.08625 12301 103480100000000 174633.8
##     4) ocean_proximity=INLAND 4399  12495580000000 112416.8 *
##     5) ocean_proximity=<1H OCEAN,NEAR BAY,NEAR OCEAN 7902  64476540000000 209269.7 *
##   3) median_income>=5.08625 3136  38647910000000 333044.6
##     6) median_income< 7.26075 2350  22245550000000 298442.5 *
##     7) median_income>=7.26075 786  5176305000000 436498.8 *
```

```
rpart.plot(dt_model1, box.palette = "RdYlGn", type = 4,
main = "Decision Tree CP = 0.05")
```



Rules for this model are as follows:

```
rpart.rules(dt_model1)
```

median_house_value
112417 when median_income < 5.1 & ocean_proximity is INLAND 209270 when median_income < 5.1 & ocean_proximity is <1H OCEAN or NEAR BAY or NEAR OCEAN 298442 when median_income is 5.1 to 7.3
436499 when median_income >= 7.3

Summary of the model with cp = 0.05 is:

```
summary(dt_model1)
```

```
## Call:
## rpart(formula = median_house_value ~ ., data = train_dt, parms = list(split = "information"),
##     control = rpart.control(minbucket = 3, minsplit = 5, cp = 0.05))
##   n= 15437
##
##           CP nsplit rel error    xerror        xstd
## 1 0.30613759      0 1.0000000 1.0000652 0.012307486
## 2 0.12941054      1 0.6938624 0.6976518 0.009579710
## 3 0.05480511      2 0.5644519 0.5688975 0.008203310
## 4 0.05000000      3 0.5096468 0.5158051 0.008124524
##
## Variable importance
##      median_income    ocean_proximity           latitude housing_median_age
##                 66                 24                  6                  2
##          longitude         population         households
##                  1                  1                  1
##
## Node number 1: 15437 observations,    complexity param=0.3061376
##   mean=206814.7, MSE=1.326916e+10
##   left son=2 (12301 obs) right son=3 (3136 obs)
##   Primary splits:
##       median_income   < 5.08625  to the left,  improve=0.30613760, (0 missing)
##       ocean_proximity splits as  RL-RR,        improve=0.23553770, (0 missing)
##       latitude        < 37.925   to the right, improve=0.06640784, (0 missing)
##       longitude       < -121.865 to the right, improve=0.03836320, (0 missing)
##       total_rooms     < 1944.5   to the left,  improve=0.02805858, (0 missing)
##   Surrogate splits:
##       total_rooms < 14091.5  to the left,  agree=0.798, adj=0.004, (0 split)
##
## Node number 2: 12301 observations,    complexity param=0.1294105
##   mean=174633.8, MSE=8.412329e+09
##   left son=4 (4399 obs) right son=5 (7902 obs)
##   Primary splits:
##       ocean_proximity    splits as  RL-RR,        improve=0.25616460, (0 missing)
##       median_income      < 3.07055  to the left,  improve=0.15825560, (0 missing)
##       latitude           < 37.915   to the right, improve=0.06165936, (0 missing)
##       longitude          < -121.865 to the right, improve=0.03782196, (0 missing)
##       housing_median_age < 51.5     to the left,  improve=0.03606145, (0 missing)
##   Surrogate splits:
##       latitude           < 37.955   to the right, agree=0.732, adj=0.250, (0 split)
##       housing_median_age < 15.5     to the left,  agree=0.668, adj=0.073, (0 split)
##       longitude          < -116.905 to the right, agree=0.664, adj=0.060, (0 split)
##       population         < 338.5    to the left,  agree=0.651, adj=0.024, (0 split)
##       households         < 129.5    to the left,  agree=0.651, adj=0.024, (0 split)
##
## Node number 3: 3136 observations,    complexity param=0.05480511
##   mean=333044.6, MSE=1.232395e+10
##   left son=6 (2350 obs) right son=7 (786 obs)
##   Primary splits:
##       median_income      < 7.26075  to the left,  improve=0.29047000, (0 missing)
```

```
##       ocean_proximity    splits as  RL-RR,        improve=0.13743780, (0 missing)
##       housing_median_age < 27.5     to the left,  improve=0.09792600, (0 missing)
##       latitude           < 37.965   to the right, improve=0.05485402, (0 missing)
##       longitude          < -117.735 to the right, improve=0.04145480, (0 missing)
##   Surrogate splits:
##       population     < 64       to the right, agree=0.752, adj=0.011, (0 split)
##       total_rooms    < 208      to the right, agree=0.752, adj=0.009, (0 split)
##       total_bedrooms < 25.5     to the right, agree=0.751, adj=0.008, (0 split)
##       households     < 18.5     to the right, agree=0.751, adj=0.008, (0 split)
##
## Node number 4: 4399 observations
##   mean=112416.8, MSE=2.840551e+09
##
## Node number 5: 7902 observations
##   mean=209269.7, MSE=8.159522e+09
##
## Node number 6: 2350 observations
##   mean=298442.5, MSE=9.466191e+09
##
## Node number 7: 786 observations
##   mean=436498.8, MSE=6.58563e+09
```

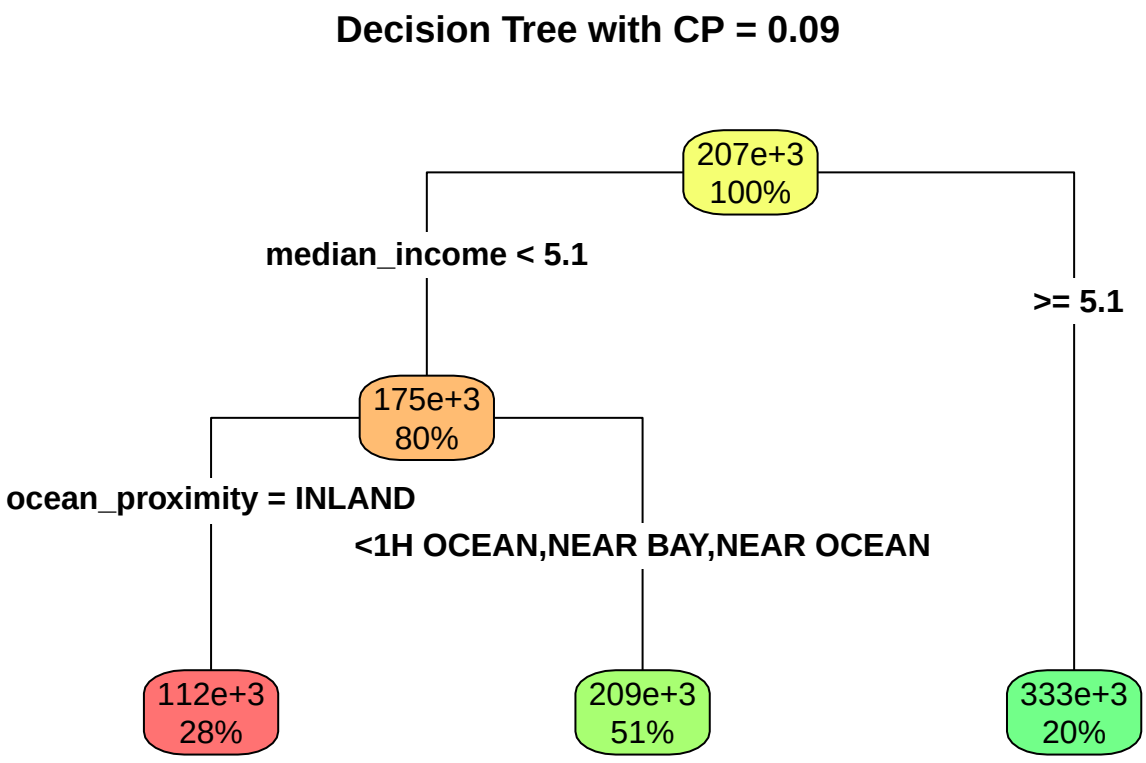
Model with split = "gini", cp = 0.09

When we try to run this model with different pruning parameters like cp=0.09, split = Gini, etc., we're seeing different results as follows:

```
dt_model2 <- rpart(median_house_value ~ ., train_dt,
parms = list(split = "gini"),
control = rpart.control(minbucket = 3, minsplit = 5, cp = 0.09))
print(dt_model2)
```

```
## n= 15437
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
## 1) root 15437 204836000000000 206814.7
##   2) median_income< 5.08625 12301 103480100000000 174633.8
##     4) ocean_proximity=INLAND 4399  12495580000000 112416.8 *
##     5) ocean_proximity=<1H OCEAN,NEAR BAY,NEAR OCEAN 7902  64476540000000 209269.7 *
##   3) median_income>=5.08625 3136  38647910000000 333044.6 *
```

```
rpart.plot(dt_model2, box.palette = "RdYlGn", type = 4,
main = "Decision Tree with CP = 0.09")
```



Rules for this model are as follows:

```
rpart.rules(dt_model2)
```

median_house_value
112417 when median_income < 5.1 & ocean_proximity is INLAND 209270 when median_income < 5.1 & ocean_proximity is <1H OCEAN or NEAR BAY or NEAR OCEAN 333045 when median_income >= 5.1

Summary of the model with cp = 0.09 is:

```
summary(dt_model2)
```

```
## Call:
## rpart(formula = median_house_value ~ ., data = train_dt, parms = list(split = "gini"),
##     control = rpart.control(minbucket = 3, minsplit = 5, cp = 0.09))
##   n= 15437
##
##          CP nsplit rel error    xerror        xstd
## 1 0.3061376      0 1.0000000 1.0000555 0.012307702
## 2 0.1294105      1 0.6938624 0.6969726 0.009568835
## 3 0.0900000      2 0.5644519 0.5684884 0.008192538
##
## Variable importance
##      median_income    ocean_proximity           latitude housing_median_age
##                 62                 26                  7                  2
##          longitude         population         households
##                  2                  1                  1
##
## Node number 1: 15437 observations,    complexity param=0.3061376
##   mean=206814.7, MSE=1.326916e+10
##   left son=2 (12301 obs) right son=3 (3136 obs)
##   Primary splits:
##       median_income   < 5.08625  to the left,  improve=0.30613760, (0 missing)
##       ocean_proximity splits as  RL-RR,        improve=0.23553770, (0 missing)
##       latitude        < 37.925   to the right, improve=0.06640784, (0 missing)
##       longitude       < -121.865 to the right, improve=0.03836320, (0 missing)
##       total_rooms     < 1944.5   to the left,  improve=0.02805858, (0 missing)
##   Surrogate splits:
##       total_rooms < 14091.5  to the left,  agree=0.798, adj=0.004, (0 split)
##
## Node number 2: 12301 observations,    complexity param=0.1294105
##   mean=174633.8, MSE=8.412329e+09
##   left son=4 (4399 obs) right son=5 (7902 obs)
##   Primary splits:
##       ocean_proximity    splits as  RL-RR,        improve=0.25616460, (0 missing)
##       median_income      < 3.07055  to the left,  improve=0.15825560, (0 missing)
##       latitude           < 37.915   to the right, improve=0.06165936, (0 missing)
##       longitude          < -121.865 to the right, improve=0.03782196, (0 missing)
##       housing_median_age < 51.5     to the left,  improve=0.03606145, (0 missing)
##   Surrogate splits:
##       latitude           < 37.955   to the right, agree=0.732, adj=0.250, (0 split)
##       housing_median_age < 15.5     to the left,  agree=0.668, adj=0.073, (0 split)
##       longitude          < -116.905 to the right, agree=0.664, adj=0.060, (0 split)
##       population         < 338.5    to the left,  agree=0.651, adj=0.024, (0 split)
##       households         < 129.5    to the left,  agree=0.651, adj=0.024, (0 split)
##
## Node number 3: 3136 observations
##   mean=333044.6, MSE=1.232395e+10
##
## Node number 4: 4399 observations
##   mean=112416.8, MSE=2.840551e+09
##
```

```
## Node number 5: 7902 observations
##   mean=209269.7, MSE=8.159522e+09
```

Model with split = "information", cp = 0 (fully grown tree)

When we try to run this model with different pruning parameters like cp=0 (fully grown tree), split = information, etc., we're seeing different results as follows. Since a full tree of this magnitude renders a big tree, we used **maxdepth** parameter, to give it a more balanced look. By default the max depth value is 30, so we changed it to **3**.

```
dt_model3 <- rpart(median_house_value ~ ., train_dt,
parms = list(split = "information"),
control = rpart.control(minbucket = 3, minsplit = 5, maxdepth = 3, cp = 0))
print(dt_model3)
```

n= 15437

node), split, n, deviance, yval * denotes terminal node

1) root 15437 204836000000000 206814.7

2) median_income< 5.08625 12301 103480100000000 174633.8
4) ocean_proximity=INLAND 4399 12495580000000 112416.8

8) median_income< 3.0366 2508 3929457000000 91043.7 *
9) median_income>=3.0366 1891 5900950000000 140763.6 *

5) ocean_proximity=<1H OCEAN,NEAR BAY,NEAR OCEAN 7902 64476540000000 209269.7

10) median_income< 3.07625 3267 20091420000000 171397.8 *
11) median_income>=3.07625 4635 36396530000000 235963.9 *

3) median_income>=5.08625 3136 38647910000000 333044.6
6) median_income< 7.26075 2350 22245550000000 298442.5

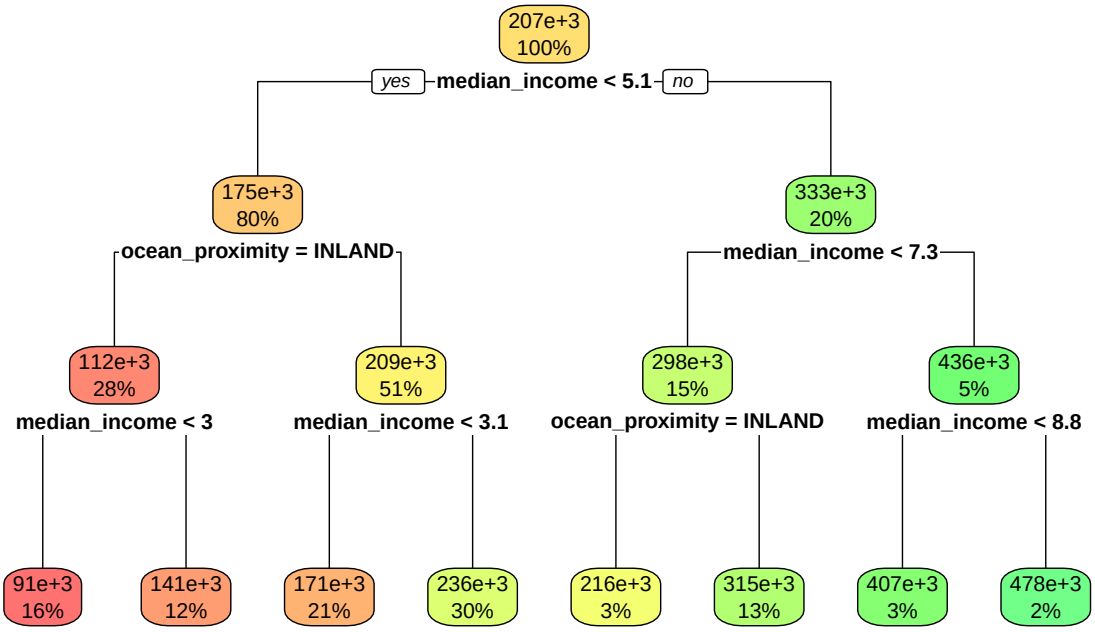
12) ocean_proximity=INLAND 403 2532496000000 216487.1 *
13) ocean_proximity=<1H OCEAN,NEAR BAY,NEAR OCEAN 1947 16445960000000 315406.0 *

7) median_income>=7.26075 786 5176305000000 436498.8

14) median_income< 8.7801 460 3068633000000 407426.8 *
15) median_income>=8.7801 326 1170296000000 477520.7 *

```
rpart.plot(dt_model3, box.palette = "RdYlGn",
main = "Fully Grown Decision Tree with MaxDepth 3")
```

Fully Grown Decision Tree with MaxDepth 3



Rules for this model are as follows:

```
rpart.rules(dt_model3)
```

median_house_value
91044 when median_income < 3.0 & ocean_proximity is INLAND 140764 when median_income is 3.0 to 5.1 & ocean_proximity is INLAND 171398 when median_income < 3.1 & ocean_proximity is <1H OCEAN or NEAR BAY or NEAR OCEAN 216487 when median_income is 5.1 to 7.3 & ocean_proximity is INLAND 235964 when median_income is 3.1 to 5.1 & ocean_proximity is <1H OCEAN or NEAR BAY or NEAR OCEAN 315406 when median_income is 5.1 to 7.3 & ocean_proximity is <1H OCEAN or NEAR BAY or NEAR OCEAN 407427 when median_income is 7.3 to 8.8
477521 when median_income >= 8.8

Summary of the model is as follows:

```
summary(dt_model3)
```

```
## Call:
## rpart(formula = median_house_value ~ ., data = train_dt, parms = list(split = "information"),
##     control = rpart.control(minbucket = 3, minsplit = 5, maxdepth = 3,
##         cp = 0))
##   n= 15437
##
##            CP nsplit rel error    xerror        xstd
## 1 0.306137589      0 1.0000000 1.0002063 0.012309433
## 2 0.129410540      1 0.6938624 0.6982176 0.009579240
## 3 0.054805114      2 0.5644519 0.5693470 0.008200786
## 4 0.038999958      3 0.5096468 0.5147197 0.008109174
## 5 0.015949803      4 0.4706468 0.4782737 0.007775751
## 6 0.013011270      5 0.4546970 0.4628688 0.007654834
## 7 0.004576226      6 0.4416857 0.4488106 0.007590885
## 8 0.000000000      7 0.4371095 0.4442364 0.007599173
##
## Variable importance
##      median_income    ocean_proximity           latitude housing_median_age
##                 65                 23                  6                  2
##          longitude        total_rooms         population         households
##                  1                  1                  1                  1
##
## Node number 1: 15437 observations,    complexity param=0.3061376
##   mean=206814.7, MSE=1.326916e+10
##   left son=2 (12301 obs) right son=3 (3136 obs)
##   Primary splits:
##       median_income   < 5.08625  to the left,  improve=0.30613760, (0 missing)
##       ocean_proximity splits as  RL-RR,        improve=0.23553770, (0 missing)
##       latitude        < 37.925   to the right, improve=0.06640784, (0 missing)
##       longitude       < -121.865 to the right, improve=0.03836320, (0 missing)
##       total_rooms     < 1944.5   to the left,  improve=0.02805858, (0 missing)
##   Surrogate splits:
##       total_rooms < 14091.5  to the left,  agree=0.798, adj=0.004, (0 split)
##
## Node number 2: 12301 observations,    complexity param=0.1294105
##   mean=174633.8, MSE=8.412329e+09
##   left son=4 (4399 obs) right son=5 (7902 obs)
##   Primary splits:
##       ocean_proximity    splits as  RL-RR,        improve=0.25616460, (0 missing)
##       median_income      < 3.07055  to the left,  improve=0.15825560, (0 missing)
##       latitude           < 37.915   to the right, improve=0.06165936, (0 missing)
##       longitude          < -121.865 to the right, improve=0.03782196, (0 missing)
##       housing_median_age < 51.5     to the left,  improve=0.03606145, (0 missing)
##   Surrogate splits:
##       latitude           < 37.955   to the right, agree=0.732, adj=0.250, (0 split)
##       housing_median_age < 15.5     to the left,  agree=0.668, adj=0.073, (0 split)
##       longitude          < -116.905 to the right, agree=0.664, adj=0.060, (0 split)
##       population         < 338.5    to the left,  agree=0.651, adj=0.024, (0 split)
##       households         < 129.5    to the left,  agree=0.651, adj=0.024, (0 split)
##
```

```
## Node number 3: 3136 observations,    complexity param=0.05480511
##   mean=333044.6, MSE=1.232395e+10
##   left son=6 (2350 obs) right son=7 (786 obs)
##   Primary splits:
##       median_income      < 7.26075  to the left,  improve=0.29047000, (0 missing)
##       ocean_proximity    splits as  RL-RR,        improve=0.13743780, (0 missing)
##       housing_median_age < 27.5     to the left,  improve=0.09792600, (0 missing)
##       latitude           < 37.965   to the right, improve=0.05485402, (0 missing)
##       longitude          < -117.735 to the right, improve=0.04145480, (0 missing)
##   Surrogate splits:
##       population     < 64       to the right, agree=0.752, adj=0.011, (0 split)
##       total_rooms    < 208      to the right, agree=0.752, adj=0.009, (0 split)
##       total_bedrooms < 25.5     to the right, agree=0.751, adj=0.008, (0 split)
##       households     < 18.5     to the right, agree=0.751, adj=0.008, (0 split)
##
## Node number 4: 4399 observations,    complexity param=0.01301127
##   mean=112416.8, MSE=2.840551e+09
##   left son=8 (2508 obs) right son=9 (1891 obs)
##   Primary splits:
##       median_income      < 3.0366   to the left,  improve=0.21328940, (0 missing)
##       latitude           < 34.825   to the right, improve=0.03347188, (0 missing)
##       total_rooms        < 2336.5   to the left,  improve=0.02652757, (0 missing)
##       longitude          < -118.325 to the left,  improve=0.02470675, (0 missing)
##       housing_median_age < 14.5     to the right, improve=0.02234194, (0 missing)
##   Surrogate splits:
##       housing_median_age < 14.5     to the right, agree=0.628, adj=0.135, (0 split)
##       total_rooms        < 2830.5   to the left,  agree=0.621, adj=0.119, (0 split)
##       population         < 2149.5   to the left,  agree=0.592, adj=0.051, (0 split)
##       households         < 881.5    to the left,  agree=0.587, adj=0.040, (0 split)
##       total_bedrooms     < 890.5    to the left,  agree=0.586, adj=0.038, (0 split)
##
## Node number 5: 7902 observations,    complexity param=0.03899996
##   mean=209269.7, MSE=8.159522e+09
##   left son=10 (3267 obs) right son=11 (4635 obs)
##   Primary splits:
##       median_income      < 3.07625  to the left,  improve=0.12389920, (0 missing)
##       longitude          < -118.305 to the right, improve=0.07329892, (0 missing)
##       latitude           < 33.995   to the left,  improve=0.03821639, (0 missing)
##       housing_median_age < 50.5     to the left,  improve=0.03652235, (0 missing)
##       total_rooms        < 2032     to the left,  improve=0.03197246, (0 missing)
##   Surrogate splits:
##       total_rooms < 1111.5   to the left,  agree=0.617, adj=0.073, (0 split)
##       latitude    < 32.775   to the left,  agree=0.602, adj=0.037, (0 split)
##       longitude   < -123.045 to the left,  agree=0.596, adj=0.024, (0 split)
##       households  < 132.5    to the left,  agree=0.592, adj=0.013, (0 split)
##       population  < 319.5    to the left,  agree=0.591, adj=0.010, (0 split)
##
## Node number 6: 2350 observations,    complexity param=0.0159498
##   mean=298442.5, MSE=9.466191e+09
##   left son=12 (403 obs) right son=13 (1947 obs)
##   Primary splits:
##       ocean_proximity    splits as  RL-RR,        improve=0.14686500, (0 missing)
##       housing_median_age < 27.5     to the left,  improve=0.11571190, (0 missing)
##       median_income      < 6.27875  to the left,  improve=0.09079596, (0 missing)
```

```
##       latitude           < 37.955   to the right, improve=0.04758064, (0 missing)
##       longitude          < -118.025 to the right, improve=0.04194649, (0 missing)
##   Surrogate splits:
##       latitude           < 38.335   to the right, agree=0.861, adj=0.191, (0 split)
##       longitude          < -116.605 to the right, agree=0.832, adj=0.022, (0 split)
##       housing_median_age < 2.5      to the left,  agree=0.831, adj=0.017, (0 split)
##
## Node number 7: 786 observations,    complexity param=0.004576226
##   mean=436498.8, MSE=6.58563e+09
##   left son=14 (460 obs) right son=15 (326 obs)
##   Primary splits:
##       median_income      < 8.7801   to the left,  improve=0.18108970, (0 missing)
##       ocean_proximity    splits as  RL-RR,        improve=0.15074610, (0 missing)
##       housing_median_age < 18.5     to the left,  improve=0.13714080, (0 missing)
##       longitude          < -117.775 to the right, improve=0.07536763, (0 missing)
##       total_rooms        < 373.5    to the left,  improve=0.07009753, (0 missing)
##   Surrogate splits:
##       total_bedrooms     < 189.5    to the right, agree=0.615, adj=0.071, (0 split)
##       households         < 174      to the right, agree=0.612, adj=0.064, (0 split)
##       population         < 422      to the right, agree=0.609, adj=0.058, (0 split)
##       housing_median_age < 38.5     to the left,  agree=0.607, adj=0.052, (0 split)
##       total_rooms        < 791      to the right, agree=0.604, adj=0.046, (0 split)
##
## Node number 8: 2508 observations
##   mean=91043.7, MSE=1.566769e+09
##
## Node number 9: 1891 observations
##   mean=140763.6, MSE=3.120545e+09
##
## Node number 10: 3267 observations
##   mean=171397.8, MSE=6.149807e+09
##
## Node number 11: 4635 observations
##   mean=235963.9, MSE=7.852542e+09
##
## Node number 12: 403 observations
##   mean=216487.1, MSE=6.284108e+09
##
## Node number 13: 1947 observations
##   mean=315406, MSE=8.44682e+09
##
## Node number 14: 460 observations
##   mean=407426.8, MSE=6.670941e+09
##
## Node number 15: 326 observations
##   mean=477520.7, MSE=3.589867e+09
```

Pruning Parameters Observation

Based on playing with the pruning parameters, we notice the following:

1. Default cp (0.01) - the most in-depth tree outside of the full tree:

- The model predicts the median_house_value to be **$436,499** when **median_income is >= 7.3**.
- The model predicts the median_house_value to be **$140,764** when **median_income is 3.0 to 5.1 and ocean_proximity is INLAND**.

2. cp value of 0.05, minbucket=3, minsplit=5 & split = "information" - Here we see the same tree as above, but it doesn't branch further after yes = ocean_proximity(value=INLAND) and no = median_income(value<7.3), so it doesn't give us in-depth view of the tree.

- The model predicts the median_house_value to be **$436,499** when **median_income is >= 7.3**.
- The model predicts the median_house_value to be $112,417 when **median_income is < 5.1 & ocean_proximity is INLAND**.

3. cp value of 0.09, minbucket=3, minsplit=5 & split = "gini" - Here we see that the root node doesn't branch further into median_income(value<6.8), instead only branches into ocean_proximity(value=INLAND). This is not as good as the first two options.

- The model predicts the median_house_value to be **$333,045** when **median_income is >= 5.1**.
- The model predicts the median_house_value to be $112,417 when **median_income is < 5.1 & ocean_proximity is INLAND**.

4. cp value of 0 - Here we see that a full blown tree is formed, but it had 1000+ nodes to cover, so the plot has overplotting in R and also the rules don't return a value, as it can't render a result with more than 1000 values. We had to reduce the MaxDepth to 3 to avoid over plotting.

(m) Variables Considered Important for Decision Tree Model

Q. What are the important variables suggested by your decision tree model? Are they the same as what you got from the random forest model?

A. Based on the summaries of the three models above (dt_model, dt_model1, and dt_model2), we notice that the two most important variables for the Decision Tree are **median_income** and **ocean_proximity**. **This is the same thing we noticed in the Random Forest model as well, where the same variable were considered most important to the model**. These two have the highest number in the variable importance section of "Summary" function.

1. Default model - **median_income** has a variable importance of **63** and **ocean_proximity** has a variable importance of **22**.
2. Model with cp = 0.05 - **median_income** has a variable importance of **66** and **ocean_proximity** has a variable importance of **24**.
3. Model with cp = 0.09 - **median_income** has a variable importance of **62** and **ocean_proximity** has a variable importance of **26**.
4. Model with cp = 0 - **median_income** has a variable importance of **58** and **ocean_proximity** has a variable importance of **20**.

(n) Performance of Decision Tree Model on Test Instance

Q. How is the performance of your decision tree model on the test instances?

A. In order to get the performance of the decision tree we've built on the test instance, we can calculate the Mean Absolute Error (MAE) and Mean Squared Error(MSE) values like we did in our Random Forest Model. The code for that is as follows:

```
pred_dt <- predict(dt_model, newdata = test_dt)
head(pred_dt, n=50)
```

```
##        4        6        7       15       18       20       21       29
## 315406.0 257853.3 257853.3 171397.8 171397.8 171397.8 171397.8 171397.8
##       35       37       41       43       46       52       61       68
## 171397.8 171397.8 171397.8 171397.8 171397.8 171397.8 257853.3 171397.8
##       70       72       76       77       79       80       85       94
## 171397.8 171397.8 171397.8 171397.8 171397.8 171397.8 171397.8 171397.8
##       95       96       99      104      109      111      121      125
## 171397.8 171397.8 171397.8 171397.8 257853.3 315406.0 315406.0 315406.0
##      135      139      150      162      164      165      172      173
## 436498.8 315406.0 257853.3 171397.8 257853.3 171397.8 171397.8 171397.8
##      176      180      183      185      187      189      194      198
## 171397.8 171397.8 171397.8 171397.8 171397.8 171397.8 171397.8 171397.8
##      200      211
## 171397.8 257853.3
```

Mean Absolute Error (MAE) of this model is:

```
library(Metrics)
library(caret)
MAE(pred_dt, test_dt$median_house_value)
```

```
## [1] 54312.93
```

Mean Squared Error (MSE) of this model is:

```
true_value_dt <- test_dt$median_house_value
mse_dt <- mean((pred_dt - true_value_dt)^2)
print(mse_dt)
```

```
## [1] 5585664420
```

(o) Which Model is Best?

Q. Which model (random forest vs decision tree) performs better? Why?

A. Both models don't quite perform well on this data set, but out of the two, we can go with the lower MSE value, which is **Random Forest**. The lower MSE sets the Random Forest model apart from the Decision Tree model. Also, Random Forest is giving us accurate predictions compared to the other model.

Problem 5

(Building models in R) Phishing attacks are the most common type of cyber-attacks used to trick users into clicking on phishing links, stealing user information, and ultimately using user data to fake logging in with related accounts to steal funds. Phishing attacks have been affecting individuals as well as organizations across the globe. In this question, we want to build a machine learning model to detect website phishing. To do so, download the "Phising websites" data set from UCI Machine Learning Repository. Before constructing any models, explore this data set and clean it if required. Construct a decision tree, naive Bayes, and a random forest model to detect phishing websites from the legitimate. Use cross-validation to check the performance of your models. Which model will be chosen as your final model? What evaluation measure(s) do you use to select the best model? Justify your answer.

```
library(caret)
library(randomForest)
library(e1071)
library(rpart)
```

```
# Load the foreign package
library(foreign)

# Read in the ARFF file
data <- read.arff("C:/Training Dataset.arff")

# Write the data to a CSV file
write.csv(data, file = "data.csv", row.names = FALSE)
```

```
# Display the dataset structure
str(data)
```

```
## 'data.frame':    11055 obs. of  31 variables:
##  $ having_IP_Address          : Factor w/ 2 levels "-1","1": 1 2 2 2 2 1 2 2 2 2 ...
##  $ URL_Length                 : Factor w/ 3 levels "-1","0","1": 3 3 2 2 2 2 2 2 2 3 ...
##  $ Shortining_Service         : Factor w/ 2 levels "-1","1": 2 2 2 2 1 1 1 2 1 1 ...
##  $ having_At_Symbol           : Factor w/ 2 levels "-1","1": 2 2 2 2 2 2 2 2 2 2 ...
##  $ double_slash_redirecting   : Factor w/ 2 levels "-1","1": 1 2 2 2 2 1 2 2 2 2 ...
##  $ Prefix_Suffix              : Factor w/ 2 levels "-1","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ having_Sub_Domain          : Factor w/ 3 levels "-1","0","1": 1 2 1 1 3 3 1 1 3 1 ...
##  $ SSLfinal_State             : Factor w/ 3 levels "-1","0","1": 1 3 1 1 3 3 1 1 3 3 ...
##  $ Domain_registeration_length: Factor w/ 2 levels "-1","1": 1 1 1 2 1 1 2 2 1 1 ...
##  $ Favicon                    : Factor w/ 2 levels "-1","1": 2 2 2 2 2 2 2 2 2 2 ...
##  $ port                       : Factor w/ 2 levels "-1","1": 2 2 2 2 2 2 2 2 2 2 ...
##  $ HTTPS_token                : Factor w/ 2 levels "-1","1": 1 1 1 1 2 1 2 1 1 2 ...
##  $ Request_URL                : Factor w/ 2 levels "-1","1": 2 2 2 1 2 2 1 1 2 2 ...
##  $ URL_of_Anchor              : Factor w/ 3 levels "-1","0","1": 1 2 2 2 2 2 1 2 2 2 ...
##  $ Links_in_tags              : Factor w/ 3 levels "-1","0","1": 3 1 1 2 2 2 2 1 3 3 ...
##  $ SFH                        : Factor w/ 3 levels "-1","0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Submitting_to_email        : Factor w/ 2 levels "-1","1": 1 2 1 2 2 1 1 2 2 2 ...
##  $ Abnormal_URL               : Factor w/ 2 levels "-1","1": 1 2 1 2 2 1 1 2 2 2 ...
##  $ Redirect                   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ on_mouseover               : Factor w/ 2 levels "-1","1": 2 2 2 2 1 2 2 2 2 2 ...
##  $ RightClick                 : Factor w/ 2 levels "-1","1": 2 2 2 2 2 2 2 2 2 2 ...
##  $ popUpWidnow                : Factor w/ 2 levels "-1","1": 2 2 2 2 1 2 2 2 2 2 ...
```

```
##  $ Iframe                    : Factor w/ 2 levels "-1","1": 2 2 2 2 2 2 2 2 2 2 ...
##  $ age_of_domain             : Factor w/ 2 levels "-1","1": 1 1 2 1 1 2 2 1 2 2 ...
##  $ DNSRecord                 : Factor w/ 2 levels "-1","1": 1 1 1 1 1 2 1 1 1 1 ...
##  $ web_traffic               : Factor w/ 3 levels "-1","0","1": 1 2 3 3 2 3 1 2 3 2 ...
##  $ Page_Rank                 : Factor w/ 2 levels "-1","1": 1 1 1 1 1 1 1 1 2 1 ...
##  $ Google_Index              : Factor w/ 2 levels "-1","1": 2 2 2 2 2 2 2 2 2 2 ...
##  $ Links_pointing_to_page    : Factor w/ 3 levels "-1","0","1": 3 3 2 1 3 1 2 2 2 2 ...
##  $ Statistical_report        : Factor w/ 2 levels "-1","1": 1 2 1 2 2 1 1 2 2 2 ...
##  $ Result                    : Factor w/ 2 levels "-1","1": 1 1 1 1 2 2 1 1 2 1 ...
```

```
# Display the dataset summary statistics
summary(data)
```

```
##  having_IP_Address URL_Length Shortining_Service having_At_Symbol
##  -1:3793           -1:8960    -1:1444            -1:1655
##  1 :7262           0 : 135    1 :9611            1 :9400
##                    1 :1960
##  double_slash_redirecting Prefix_Suffix having_Sub_Domain SSLfinal_State
##  -1:1429                  -1:9590       -1:3363           -1:3557
##  1 :9626                  1 :1465       0 :3622           0 :1167
##                                         1 :4070           1 :6331
##  Domain_registeration_length Favicon   port      HTTPS_token Request_URL
##  -1:7389                     -1:2053   -1:1502   -1:1796     -1:4495
##  1 :3666                     1 :9002   1 :9553   1 :9259     1 :6560
##
##  URL_of_Anchor Links_in_tags SFH       Submitting_to_email Abnormal_URL
##  -1:3282       -1:3956       -1:8440   -1:2014             -1:1629
##  0 :5337       0 :4449       0 : 761   1 :9041             1 :9426
##  1 :2436       1 :2650       1 :1854
##  Redirect on_mouseover RightClick popUpWidnow Iframe     age_of_domain
##  0:9776   -1:1315      -1:  476   -1:2137     -1: 1012   -1:5189
##  1:1279   1 :9740      1 :10579   1 :8918     1 :10043   1 :5866
##
##  DNSRecord web_traffic Page_Rank Google_Index Links_pointing_to_page
##  -1:3443   -1:2655     -1:8201   -1:1539      -1: 548
##  1 :7612   0 :2569     1 :2854   1 :9516      0 :6156
##            1 :5831                            1 :4351
##  Statistical_report Result
##  -1:1550            -1:4898
##  1 :9505            1 :6157
##
```

The data set has 31 columns and 11,055 rows, and all columns are of the integer data type. The target variable is the "Result" column, which has two values: "1" for a phishing website and "0" for a legitimate website.

Next, we will check for missing values and duplicate rows in the data set:

```
# Check for missing values
sum(is.na(data))
```

```
## [1] 0
```

```
# Check for duplicate rows
sum(duplicated(data))
```

```
## [1] 5206
```

There are no missing values or duplicate rows in the data set. Thus, the data set is clean and ready for modeling.

Split into training data and testing data: We will now split the data set into training and testing sets, with a **70:30** split.

```
# Split the dataset into training and testing sets
set.seed(123)
trainIndex <- createDataPartition(data$Result, p = 0.7, list = FALSE, times = 1)
trainset <- data[trainIndex, ]
testset <- data[-trainIndex, ]
```

```
# Build a decision tree model
tree_model <- rpart(Result ~ ., data = trainset, method = "class")
# Build a naive Bayes model
nb_model <- naiveBayes(Result ~ ., data = trainset)
# Build a random forest model
rf_model <- randomForest(Result ~ ., data = trainset, ntree = 500)
```

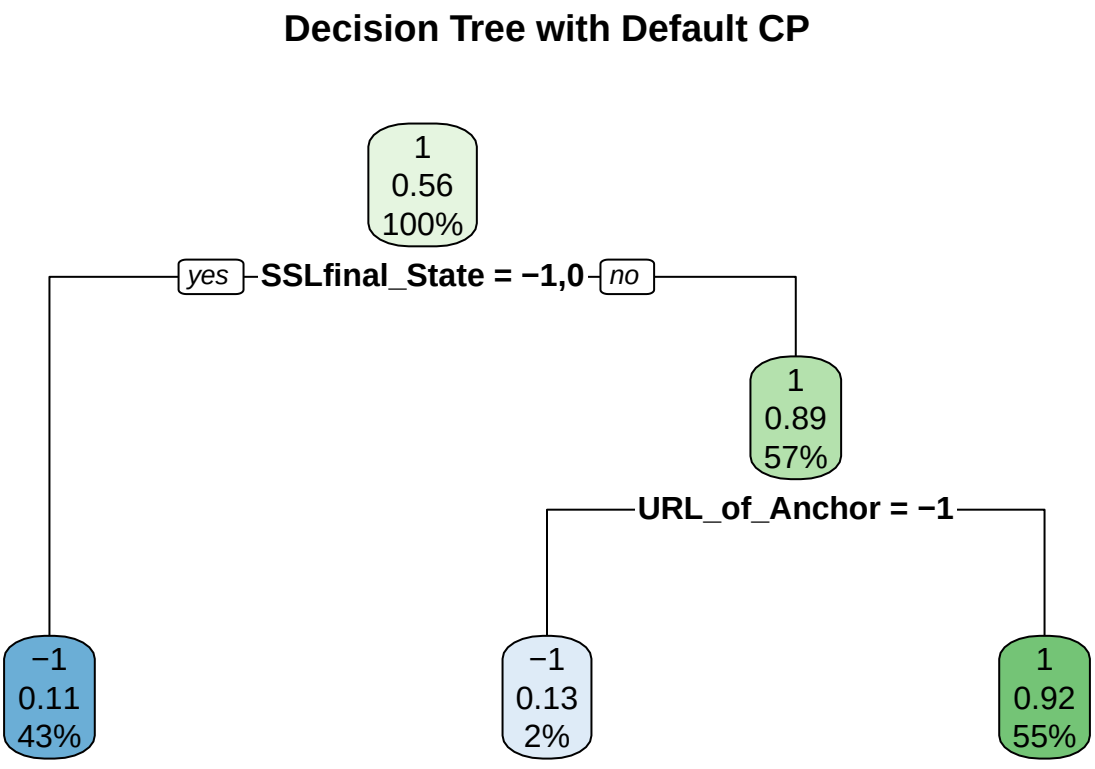
Decision tree

First create the classification tree using the default 'cp' and compare the 'Result' variable with all predictors.

```
library(rpart)
fishing_tree_model1_train <- rpart(Result ~., trainset)
print(fishing_tree_model1_train)
```

```
## n= 7739
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 7739 3429 1 (0.4430805 0.5569195)
##   2) SSLfinal_State=-1,0 3310  365 -1 (0.8897281 0.1102719) *
##   3) SSLfinal_State=1 4429  484 1 (0.1092797 0.8907203)
##     6) URL_of_Anchor=-1 189   25 -1 (0.8677249 0.1322751) *
##     7) URL_of_Anchor=0,1 4240  320 1 (0.0754717 0.9245283) *
```

```
library(rpart.plot)
rpart.plot(fishing_tree_model1_train, main = "Decision Tree with Default CP")
```



When SSLfinal State=-1,0 this website is phishing website, this part is **45%**. When SSLfinal State=1, according to URL of anchorz again classified, **2%** is phishing website, **55%** is legitimate website.

```
tree_pred_prob1 <- predict(fishing_tree_model1_train, trainset, type = "prob")
tree_pred_class1 <- predict(fishing_tree_model1_train, trainset, type = "class")
```

```
testerror_train <- mean(tree_pred_class1 != trainset$Result)
print(testerror_train)
```

```
## [1] 0.09174312
```

Error rate of our decision tree model on train data is **0.097174312**.

```
tree_pred_test1 <- predict(fishing_tree_model1_train, testset, type = "class")
testerror_test1 <- mean(tree_pred_test1 != testset$Result)
print(testerror_test1)
```

```
## [1] 0.09831122
```

Error rate of our decision tree model on test data is **0.109831122**.

```
t1 <- table(tree_pred_test1, testset$Result)
acc1 <- sum(diag(t1))/nrow(testset)*100
print(acc1)
```

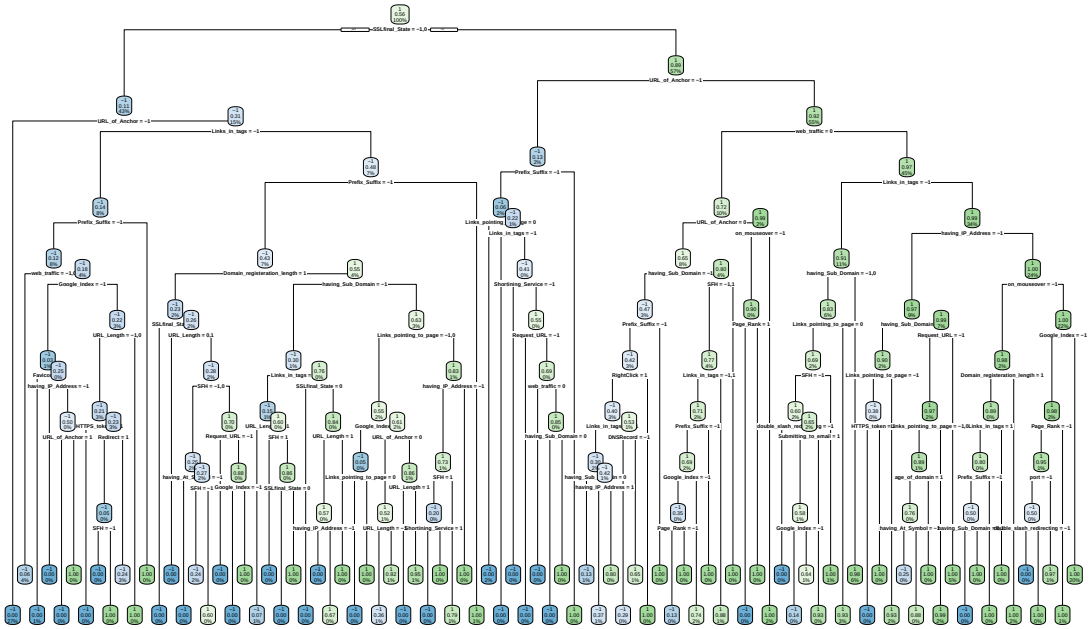
```
## [1] 90.16888
```

We calculated the accuracy of model is **90.16888**.

Fully Grown Decision Tree

```
tree_model_full <- rpart(Result ~ ., trainset,
parms = list(split = "information"),
control = rpart.control(minbucket = 0, minsplit = 0, maxdepth = 10, cp = 0))
rpart.plot(tree_model_full, main = "Fully Grown Decision Tree")
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```




```
pred_test_model_full <-
predict(tree_model_full, testset, type = "class")
error_pred_model_full <-
mean(pred_test_model_full != testset$Result)
```

Select best CP

Use CP table to see best CP

```
printcp(tree_model_full)
```

```
##
## Classification tree:
## rpart(formula = Result ~ ., data = trainset, parms = list(split = "information"),
##     control = rpart.control(minbucket = 0, minsplit = 0, maxdepth = 10,
##         cp = 0))
##
## Variables actually used in tree construction:
##  [1] age_of_domain               DNSRecord
##  [3] Domain_registeration_length double_slash_redirecting
##  [5] Favicon                     Google_Index
##  [7] having_At_Symbol            having_IP_Address
##  [9] having_Sub_Domain           HTTPS_token
## [11] Links_in_tags               Links_pointing_to_page
## [13] on_mouseover                Page_Rank
## [15] port                        Prefix_Suffix
## [17] Redirect                    Request_URL
## [19] RightClick                  SFH
## [21] Shortining_Service          SSLfinal_State
## [23] Submitting_to_email         URL_Length
## [25] URL_of_Anchor               web_traffic
##
## Root node error: 3429/7739 = 0.44308
##
## n= 7739
##
##             CP nsplit rel error  xerror      xstd
## 1  0.752405949      0   1.00000 1.00000 0.0127442
## 2  0.040536600      1   0.24759 0.24759 0.0080178
## 3  0.004957714      2   0.20706 0.20706 0.0074057
## 4  0.004082823      7   0.17294 0.17994 0.0069492
## 5  0.003791193      8   0.16885 0.17906 0.0069337
## 6  0.003207932      9   0.16506 0.16915 0.0067551
## 7  0.002770487     10   0.16185 0.16798 0.0067336
## 8  0.002478857     17   0.14202 0.16010 0.0065863
## 9  0.001166521     21   0.13094 0.14465 0.0062833
## 10 0.000874891     22   0.12978 0.13940 0.0061759
## 11 0.000729076     24   0.12803 0.13969 0.0061820
## 12 0.000641586     26   0.12657 0.13969 0.0061820
## 13 0.000583260     33   0.12044 0.13677 0.0061213
## 14 0.000437445     34   0.11986 0.13590 0.0061030
## 15 0.000388840     47   0.11257 0.13415 0.0060660
## 16 0.000364538     54   0.10965 0.13502 0.0060845
## 17 0.000291630     58   0.10819 0.13502 0.0060845
## 18 0.000233304     64   0.10645 0.13007 0.0059788
## 19 0.000194420     69   0.10528 0.13007 0.0059788
## 20 0.000145815     72   0.10470 0.13094 0.0059976
## 21 0.000097210     80   0.10353 0.13123 0.0060039
```

```
## 22 0.000072908     83   0.10324 0.13094 0.0059976
## 23 0.000000000     87   0.10295 0.13211 0.0060226
```

```
mincp_model_fishing <- which.min(tree_model_full$cptable[, 'xerror'])
```

```
optCP_model_fishing <- tree_model_full$cptable[mincp_model_fishing, "CP"]
print(optCP_model_fishing)
```

```
## [1] 0.0002333042
```

Pruning decision tree

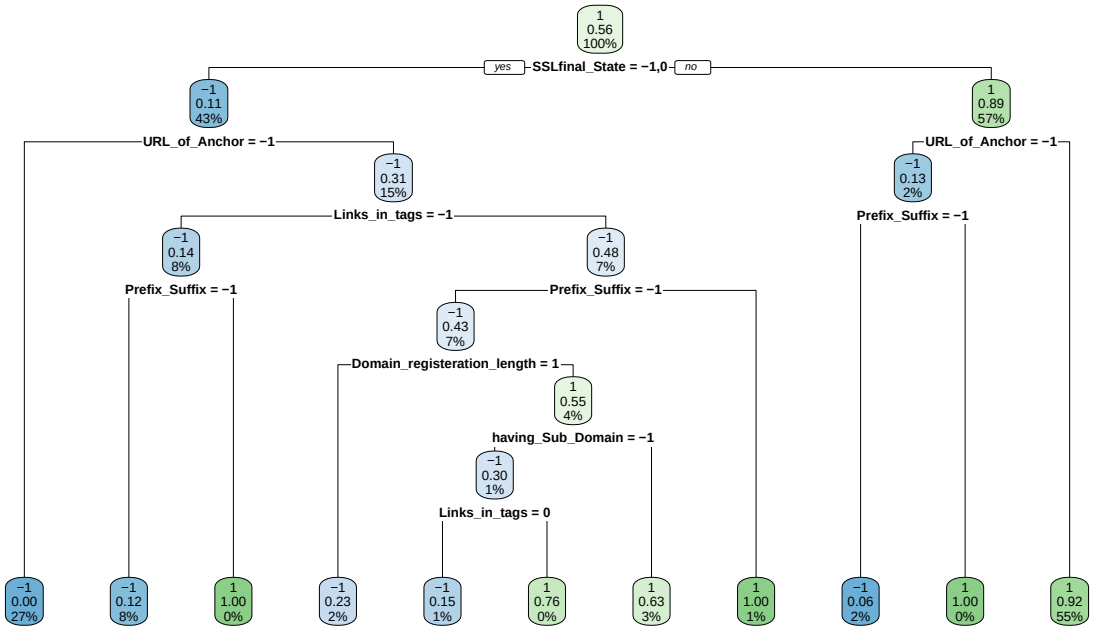
```
mincp_model_fishing <- which.min(tree_model_full$cptable[, 'xerror'])
tree_model_full$cptable
```

```
##               CP nsplit rel error    xerror        xstd
## 1  0.75240594926      0 1.0000000 1.0000000 0.012744197
## 2  0.04053659959      1 0.2475941 0.2475941 0.008017769
## 3  0.00495771362      2 0.2070575 0.2070575 0.007405699
## 4  0.00408282298      7 0.1729367 0.1799358 0.006949182
## 5  0.00379119277      8 0.1688539 0.1790610 0.006933727
## 6  0.00320793234      9 0.1650627 0.1691455 0.006755076
## 7  0.00277048702     10 0.1618548 0.1679790 0.006733623
## 8  0.00247885681     17 0.1420239 0.1601050 0.006586288
## 9  0.00116652085     21 0.1309420 0.1446486 0.006283334
## 10 0.00087489064     22 0.1297754 0.1393992 0.006175928
## 11 0.00072907553     24 0.1280257 0.1396909 0.006181960
## 12 0.00064158647     26 0.1265675 0.1396909 0.006181960
## 13 0.00058326043     33 0.1204433 0.1367746 0.006121301
## 14 0.00043744532     34 0.1198600 0.1358997 0.006102950
## 15 0.00038884028     47 0.1125693 0.1341499 0.006066034
## 16 0.00036453777     54 0.1096530 0.1350248 0.006084529
## 17 0.00029163021     58 0.1081948 0.1350248 0.006084529
## 18 0.00023330417     64 0.1064450 0.1300671 0.005978753
## 19 0.00019442014     69 0.1052785 0.1300671 0.005978753
## 20 0.00014581511     72 0.1046952 0.1309420 0.005997593
## 21 0.00009721007     80 0.1035287 0.1312336 0.006003857
## 22 0.00007290755     83 0.1032371 0.1309420 0.005997593
## 23 0.00000000000     87 0.1029455 0.1321085 0.006022597
```

```
optcp <- 3.207932e-03
```

```
pruning_tree <- prune(tree_model_full, cp=optcp)
rpart.plot(pruning_tree, main = "Pruning Decision Tree")
```

Pruning Decision Tree



Naive bayes

To build a NB model, we will use the naiveBayes() function from the e1071 package

```
library(e1071)
nb <- naiveBayes(Result ~ ., data = trainset)

print(nb)
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##        -1         1
## 0.4430805 0.5569195
##
## Conditional probabilities:
##     having_IP_Address
## Y           -1         1
##   -1 0.3939924 0.6060076
##   1  0.2953596 0.7046404
##
##     URL_Length
## Y             -1           0           1
##   -1 0.834937300 0.017497813 0.147564888
##   1  0.793735499 0.008584687 0.197679814
##
##     Shortining_Service
## Y           -1         1
##   -1 0.1020706 0.8979294
##   1  0.1440835 0.8559165
##
##     having_At_Symbol
## Y           -1         1
##   -1 0.1749781 0.8250219
##   1  0.1382831 0.8617169
##
##     double_slash_redirecting
## Y           -1         1
##   -1 0.1108195 0.8891805
##   1  0.1348028 0.8651972
##
##     Prefix_Suffix
## Y           -1         1
##   -1 1.0000000 0.0000000
##   1  0.7670534 0.2329466
##
##     having_Sub_Domain
## Y           -1         0         1
```

```
##   -1 0.3724118 0.4648586 0.1627297
##   1  0.2501160 0.2155452 0.5343387
##
##     SSLfinal_State
## Y             -1           0           1
##   -1 0.627588218 0.231262759 0.141149023
##   1  0.081206497 0.003480278 0.915313225
##
##     Domain_registeration_length
## Y           -1         1
##   -1 0.5485564 0.4514436
##   1  0.7610209 0.2389791
##
##     Favicon
## Y           -1         1
##   -1 0.1843103 0.8156897
##   1  0.1909513 0.8090487
##
##     port
## Y           -1         1
##   -1 0.1513561 0.8486439
##   1  0.1287703 0.8712297
##
##     HTTPS_token
## Y           -1         1
##   -1 0.1385244 0.8614756
##   1  0.1679814 0.8320186
##
##     Request_URL
## Y           -1         1
##   -1 0.5479732 0.4520268
##   1  0.2925754 0.7074246
##
##     URL_of_Anchor
## Y             -1           0           1
##   -1 0.666958297 0.302420531 0.030621172
##   1  0.005800464 0.625754060 0.368445476
##
##     Links_in_tags
## Y           -1         0         1
##   -1 0.4887722 0.3575386 0.1536891
##   1  0.2526682 0.4368910 0.3104408
##
##     SFH
## Y            -1          0          1
##   -1 0.86585010 0.05511811 0.07903179
##   1  0.68074246 0.08143852 0.23781903
##
##     Submitting_to_email
## Y           -1         1
##   -1 0.1959755 0.8040245
##   1  0.1807425 0.8192575
##
##     Abnormal_URL
```

```
## Y           -1         1
##   -1 0.1192768 0.8807232
##   1  0.1605568 0.8394432
##
##     Redirect
## Y           0         1
##   -1 0.8783902 0.1216098
##   1  0.8935035 0.1064965
##
##     on_mouseover
## Y           -1         1
##   -1 0.1335666 0.8664334
##   1  0.1104408 0.8895592
##
##     RightClick
## Y            -1          1
##   -1 0.05016040 0.94983960
##   1  0.04338747 0.95661253
##
##     popUpWidnow
## Y           -1         1
##   -1 0.1921843 0.8078157
##   1  0.1974478 0.8025522
##
##     Iframe
## Y            -1          1
##   -1 0.09419656 0.90580344
##   1  0.09489559 0.90510441
##
##     age_of_domain
## Y           -1         1
##   -1 0.5287256 0.4712744
##   1  0.4118329 0.5881671
##
##     DNSRecord
## Y           -1         1
##   -1 0.3519977 0.6480023
##   1  0.2770302 0.7229698
##
##     web_traffic
## Y           -1         0         1
##   -1 0.3391659 0.3458734 0.3149606
##   1  0.1617169 0.1429234 0.6953596
##
##     Page_Rank
## Y           -1         1
##   -1 0.7961505 0.2038495
##   1  0.6988399 0.3011601
##
##     Google_Index
## Y            -1          1
##   -1 0.19014290 0.80985710
##   1  0.09721578 0.90278422
##
```

```
##     Links_pointing_to_page
## Y            -1          0          1
##   -1 0.03820356 0.60367454 0.35812190
##   1  0.05684455 0.53085847 0.41229698
##
##     Statistical_report
## Y           -1         1
##   -1 0.1726451 0.8273549
##   1  0.1174014 0.8825986
```

```
nb.pred <- predict(nb, newdata = testset, type = 'class')
head(nb.pred)
```

```
## [1] -1 -1 -1 1  -1 1
## Levels: -1 1
```

Then, create the confusion matrix:

```
nb.cm <- table(true = testset$Result, predicted = nb.pred)
nb.cm
```

```
##     predicted
## true   -1    1
##   -1 1317  152
##   1    79 1768
```

The diagonal cells of the table indicate the number of correct predictions, while the off-diagonal cells indicate the number of incorrect predictions. In the table, **1317** instances with a true category of -phishing sites were correctly predicted as phishing sites, and **1768** instances with a true category of legitimate sites were correctly predicted as legitimate sites. However, **152** instances where the true category was phishing were incorrectly predicted as legitimate, and **79** instances where the true category was legitimate were incorrectly predicted as phishing.

Random forest

```
library(randomForest)
rf <- randomForest(Result ~ ., data = data,
                   mtry = sqrt(ncol(data)-1), ntree = 300,
                   proximity = T, importance = T)
```

```
print(rf)
```

Call: randomForest(formula = Result ~ ., data = data, mtry = sqrt(ncol(data) - 1), ntree = 300, proximity = T, importance = T) Type of random forest: classification Number of trees: 300 No. of variables tried at each split: 5

```
    OOB estimate of  error rate: 3.15%
```

Confusion matrix: -1 1 class.error -1 4688 210 0.04287464 1 138 6019 0.02241351

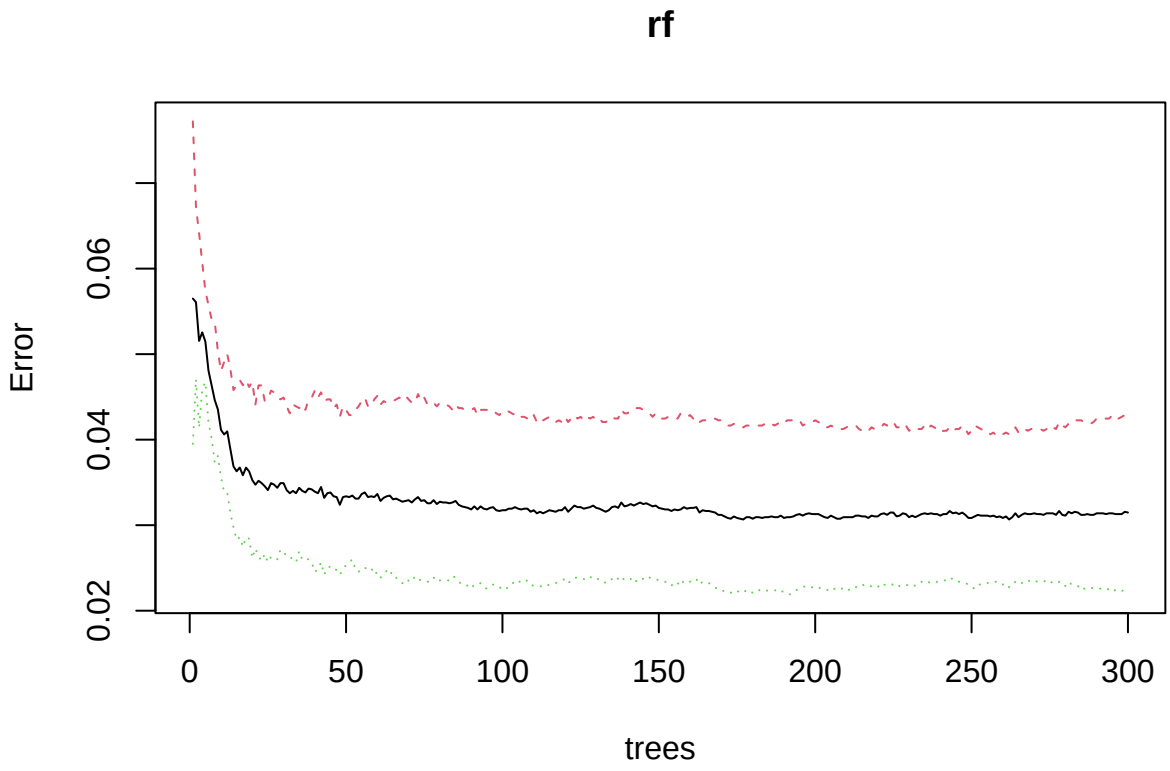
The OOB error rate of our random forest model is **0.0312**.

```
attributes(rf)
```

```
## $names
##  [1] "call"            "type"            "predicted"       "err.rate"
##  [5] "confusion"       "votes"           "oob.times"       "classes"
##  [9] "importance"      "importanceSD"    "localImportance" "proximity"
## [13] "ntree"           "mtry"            "forest"          "y"
## [17] "test"            "inbag"           "terms"
##
## $class
## [1] "randomForest.formula" "randomForest"
```

We plot the error rates with various number of trees.

```
plot(rf)
```



The black line is the error rate on OOB, the red curve is the error rate for positive class, the green curve is for negative class.

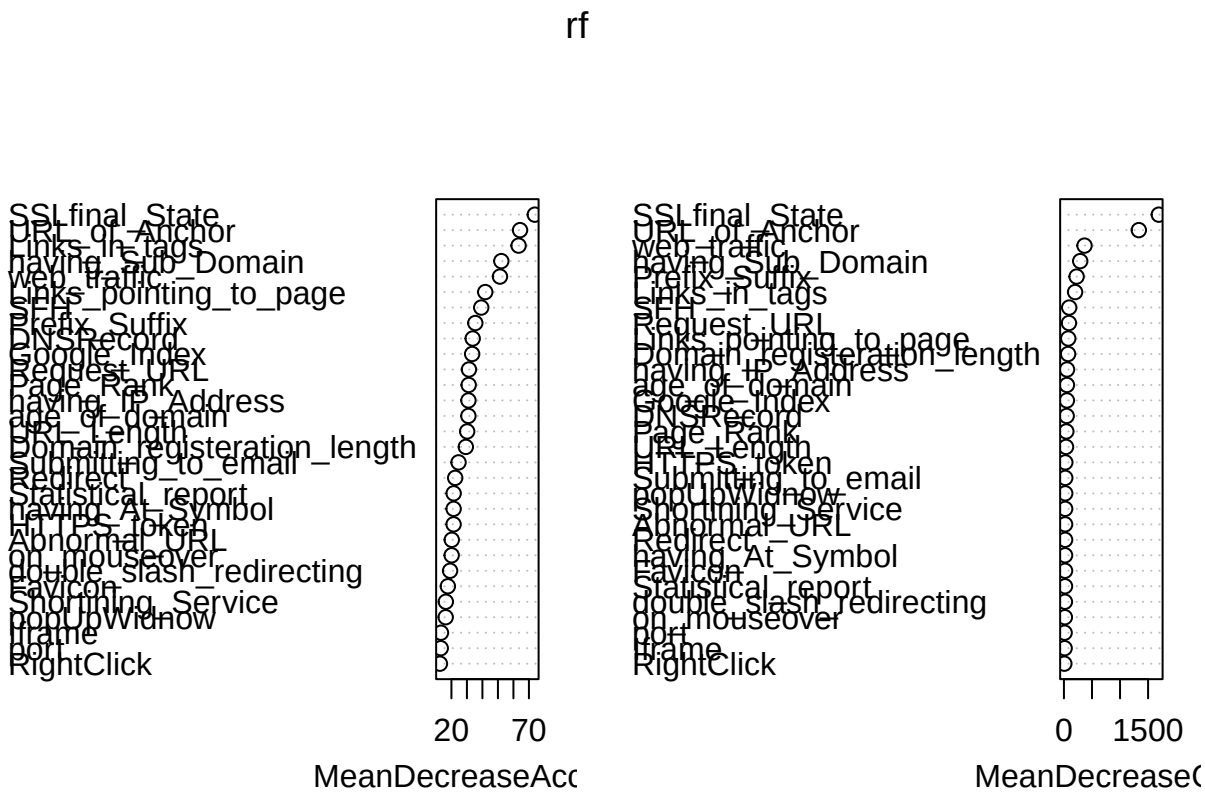
```
randomForest::importance(rf, type=1)
```

```
##                                MeanDecreaseAccuracy
## having_IP_Address                          30.97634
## URL_Length                                 30.16263
## Shortining_Service                         16.39697
## having_At_Symbol                           21.42609
## double_slash_redirecting                   19.14854
## Prefix_Suffix                              35.40801
## having_Sub_Domain                          52.18723
## SSLfinal_State                             73.79138
## Domain_registeration_length                29.29625
## Favicon                                    17.70838
## port                                       13.11635
## HTTPS_token                                21.40054
## Request_URL                                31.31438
## URL_of_Anchor                              64.27424
## Links_in_tags                              63.33862
## SFH                                        39.23903
## Submitting_to_email                        24.59844
## Abnormal_URL                               20.23265
## Redirect                                   22.44171
## on_mouseover                               20.16883
```

```
## RightClick                    12.59253
## popUpWidnow                   16.30141
## Iframe                        13.26266
## age_of_domain                 30.94396
## DNSRecord                     33.72657
## web_traffic                   51.30874
## Page_Rank                     31.13846
## Google_Index                  33.37694
## Links_pointing_to_page        41.82480
## Statistical_report            21.43757
```

Get the importance of variables by the function “importance()”. Include type = 1 in the importance function to get the important variables based on MeanDecreaseAccuracy.

```
varImpPlot(rf)
```



We obtain the predicted classes and predicted probabilities using the following codes.

```
head(rf$predicted)
```

```
##  1  2  3  4  5  6
## -1 -1 -1 -1  1  1
## Levels: -1 1
```

```
head(rf$votes)
```

```
##          -1           1
## 1 1.0000000 0.000000000
## 2 0.8256881 0.174311927
## 3 0.9910714 0.008928571
## 4 0.8691589 0.130841121
## 5 0.0000000 1.000000000
## 6 0.0400000 0.960000000
```

```
pr.err <- c()
for(mt in seq(1,ncol(trainset))){
    library(randomForest)
    rf1 <- randomForest(Result~., data = trainset, ntree = 100,
                        mtry = ifelse(mt == ncol(trainset),
                                      mt-1, mt))
    predicted <- predict(rf1, newdata = testset, type = "class")
    pr.err <- c(pr.err,mean(testset$Result != predicted))
  }
  bestmtry <- which.min(pr.err)
```

```
bestmtry
```

```
## [1] 10
```

The best mtry value based on our analysis is **12**.

Confusion matrix

Use the table() function to construct the confusion matrix. Add the predicted categories and the actual labels to this function as input parameters.

```
table(rf$predicted, data$Result, dnn = c("Predicted", "Actual"))
```

```
##           Actual
## Predicted   -1    1
##        -1 4688  138
##        1   210 6019
```

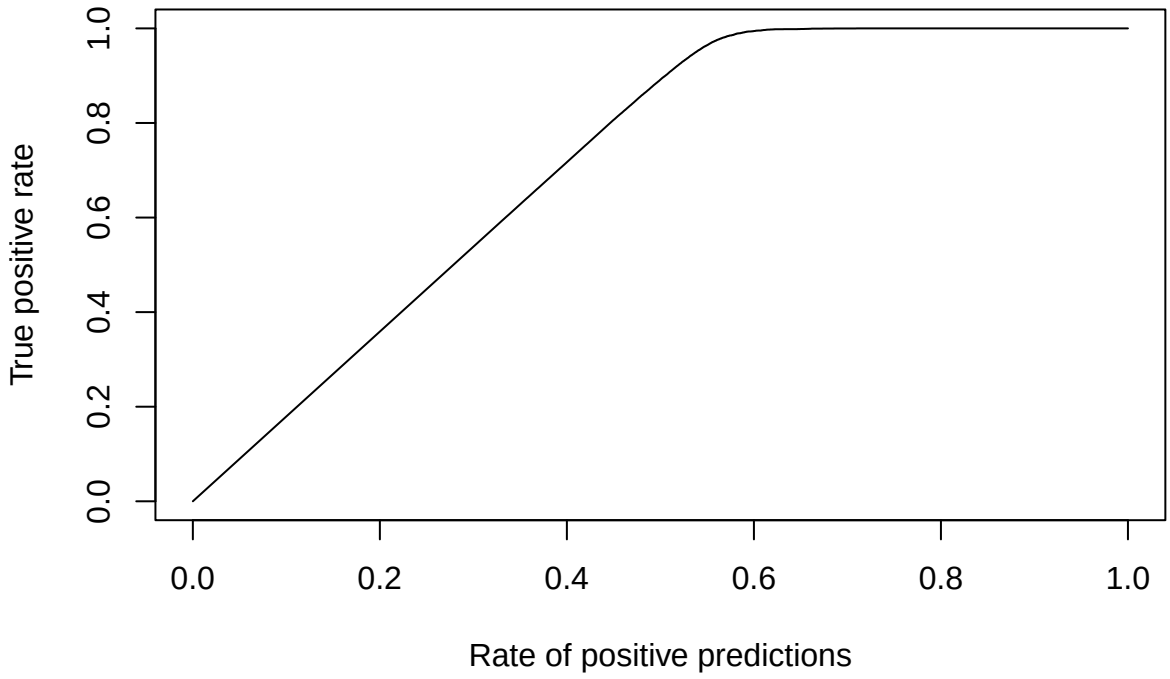
Evaluation Charts

To draw the evaluation charts we use "ROCR" package. We use the "ROCR" package to plot evaluation Two graphs are plotted based on prediction and performance. The predict() function takes two inputs: (1) the predicted probability of the positive class and (2) the true label. The output of the prediction function will be given to the performance() function to plot the graphs.

```
library(ROCR)
score <- rf$vote[, 2]
pred <- prediction(score, data$Result)
```

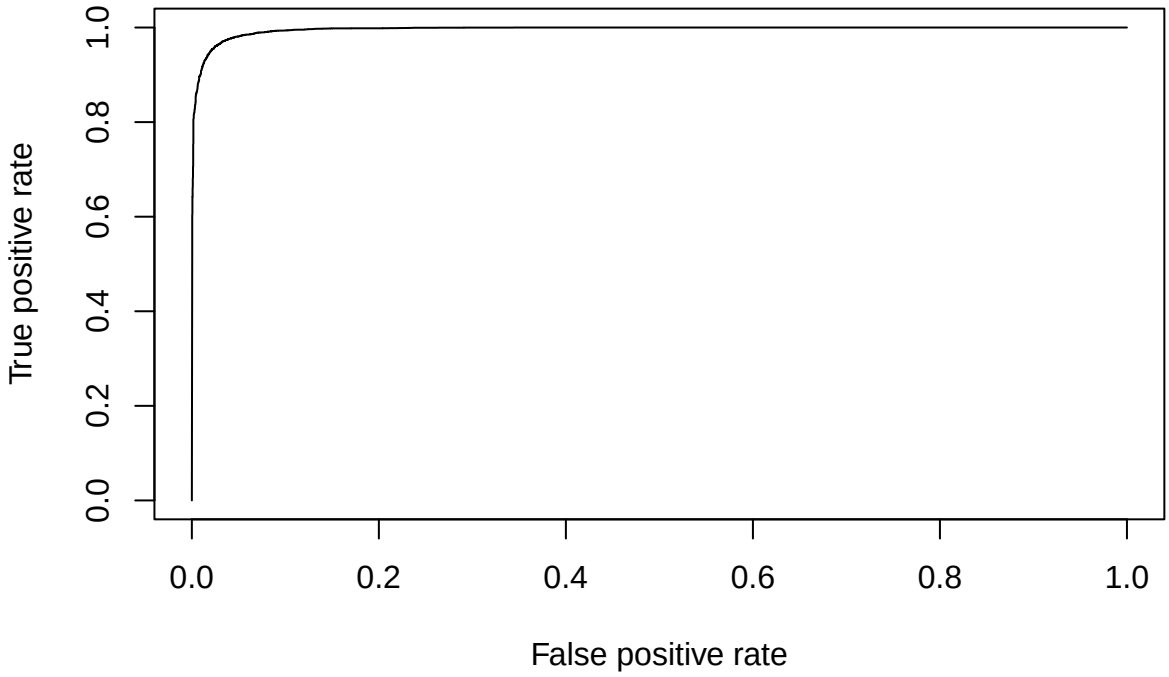
Gain Chart

```
perf <- performance(pred, "tpr", "rpp")
plot(perf)
```



ROC Curve

```
 perf <- performance(pred, "tpr", "fpr")
plot(perf)
```



Obtaining the Area Under the ROC Curve

```
auc <- unlist(slot(performance(pred, "auc"), "y.values"))
```

```
auc
```

```
## [1] 0.9957049
```

The area under the curve of our ROC curve is **0.9955913**.

Determining the best cut-off point

The performance() function for ROC curve returns tpr, fpr and alpha-values (cut-off points). Write a function that receives these information and returns the best cut-off point as the point closest to the corner [0, 1]. We can use the following code. The input argument to this function is perf (the output of the performance() function).

```
opt.cut <- function(perf){
  cut.ind <- mapply(FUN = function(x,y,p){d=(x-0)^2+(y-1)^2
  ind <- which(d == min(d))
  c(recall = y [[ind]], specificity = 1-x [[ind]], cutoff = p [[ind]])},
  perf@x.values , perf@y.values, perf@alpha.values)
}
```

```
opt.cut
```

```
## function(perf){
##   cut.ind <- mapply(FUN = function(x,y,p){d=(x-0)^2+(y-1)^2
##   ind <- which(d == min(d))
##   c(recall = y [[ind]], specificity = 1-x [[ind]], cutoff = p [[ind]])},
##   perf@x.values , perf@y.values, perf@alpha.values)
## }
```

The mapply function applies the function FUN to all **perf@x.values, perf@y.values,perf@alpha.values**. In the function FUN(), we first compute the distance of all the points on the ROC curve from the corner point [0,1]. These distance values are stored in the vector "d". We then find the index of the point that is the closest point to the corner. This index is stored in the variable named "ind". The output of this function is then the tpr, fpr and the probability threshold corresponding to this index.

```
ctrl <- trainControl(method = "cv", number = 10)
```

```
library(naivebayes)
```

```
results_dt<- train(Result ~ ., data = trainset,
method = "rpart", trControl = ctrl)


results_nb<- train(Result ~ ., data = trainset,
method = "naive_bayes", trControl = ctrl)

results_rf<- train(Result ~ ., data = trainset,
method = "rf", ntree = 500, trControl = ctrl)
```

```
results<- resamples(list(decision_tree = results_dt,
naive_bayes = results_nb, random_forest = results_rf))
summary(results)
```

```
##
## Call:
## summary.resamples(object = results)
##
```

```
## Models: decision_tree, naive_bayes, random_forest
## Number of resamples: 10
##
## Accuracy
##                    Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## decision_tree 0.8863049 0.8943798 0.9005168 0.9012798 0.9085612 0.9160207    0
## naive_bayes   0.8382924 0.8611111 0.8643411 0.8662582 0.8695090 0.8953488    0
## random_forest 0.9547804 0.9615633 0.9683463 0.9665338 0.9725360 0.9754522    0
##
## Kappa
##                    Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## decision_tree 0.7711306 0.7849072 0.7968605 0.7997914 0.8145489 0.8309987    0
## naive_bayes   0.6819380 0.7260082 0.7322180 0.7357011 0.7423111 0.7917378    0
## random_forest 0.9080205 0.9220101 0.9358497 0.9320909 0.9442966 0.9501576    0
```

The output is the summary of the results of resampling three models (decision_tree, naive_bayes, random_forest). The summary includes the accuracy and kappa scores for each model.

For each model, the summary provides the minimum, first quartile, median, mean, third quartile, and maximum accuracy and kappa scores across 10 re-samples. The NAs column indicates if any missing values were present in the data.

The accuracy score represents the proportion of correct predictions made by the model, while the kappa score measures the agreement between the predicted and actual classes, taking into account the possibility of agreement by chance.

Overall, the **random_forest model** appears to have the highest accuracy and kappa scores, followed by the **decision_tree model** and then the **naive_bayes model**.

References

1. How to solve Tex Memory Size Issue?
2. RPart in Decision Trees in R
3. Latex Engines
4. Kable and KableExtra
5. Sizing Tables in PDF
6. RPart Plots in R