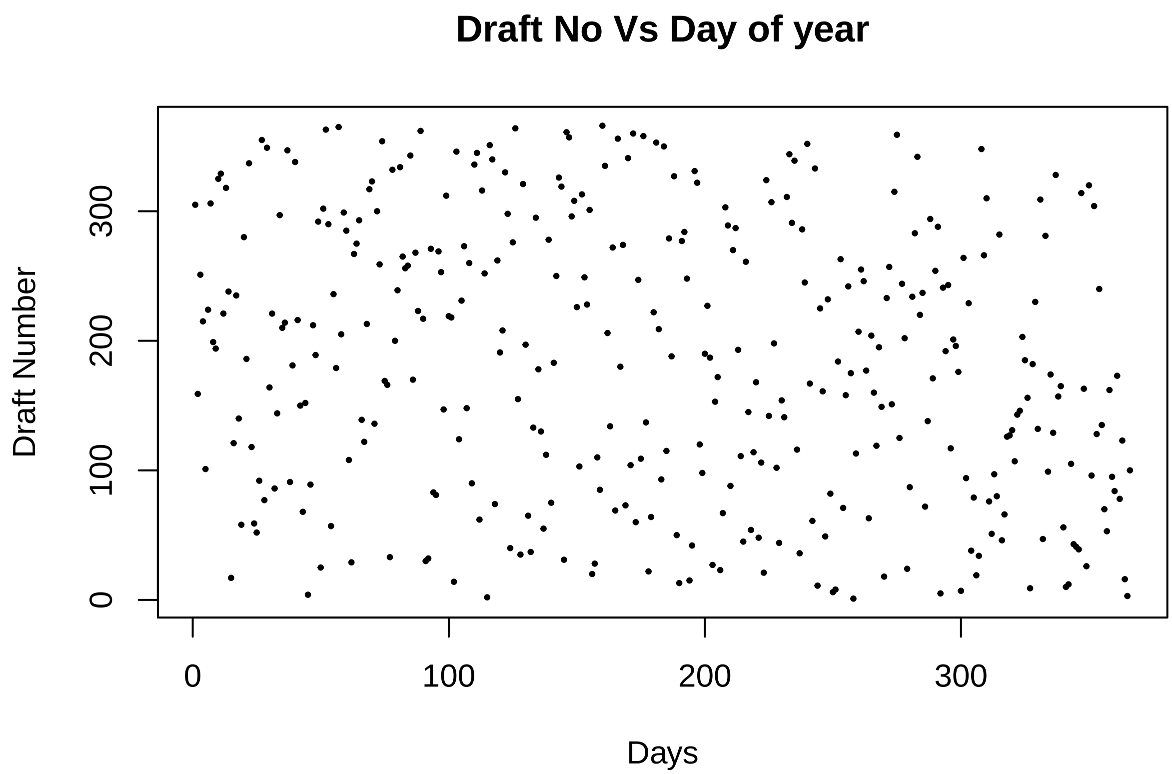
Computational statistics Lab 05 Report

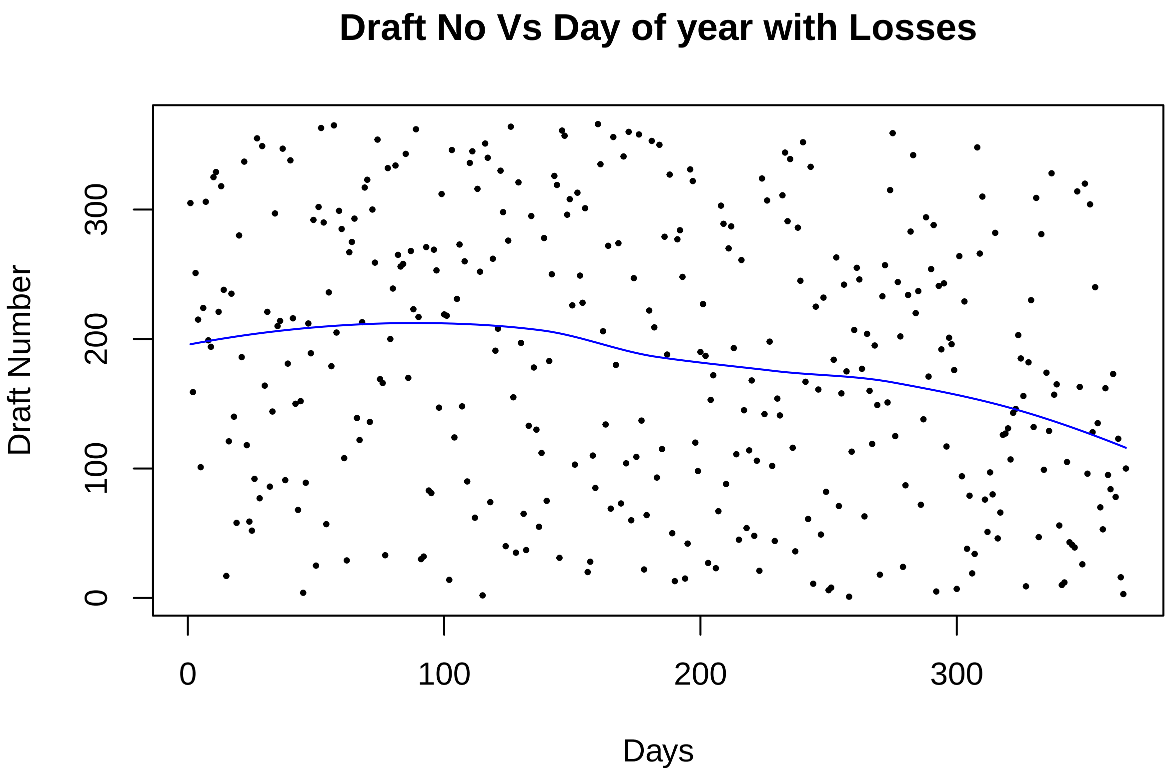
Karthikeyan Devarajan - Karde799

Question 1: Hypothesis testing

Part 01

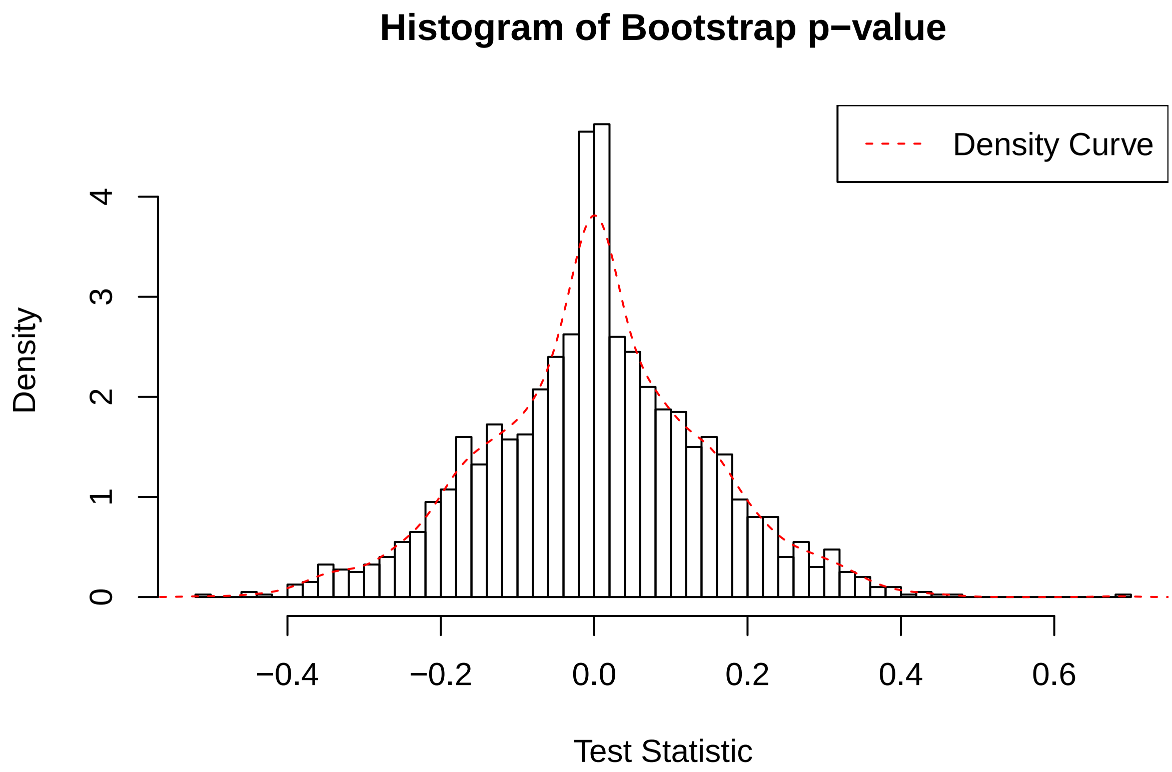


Part 02



The Losses fit shows that 'Draft number' is getting lower value when number of 'Day of the year' increases.

Part 03



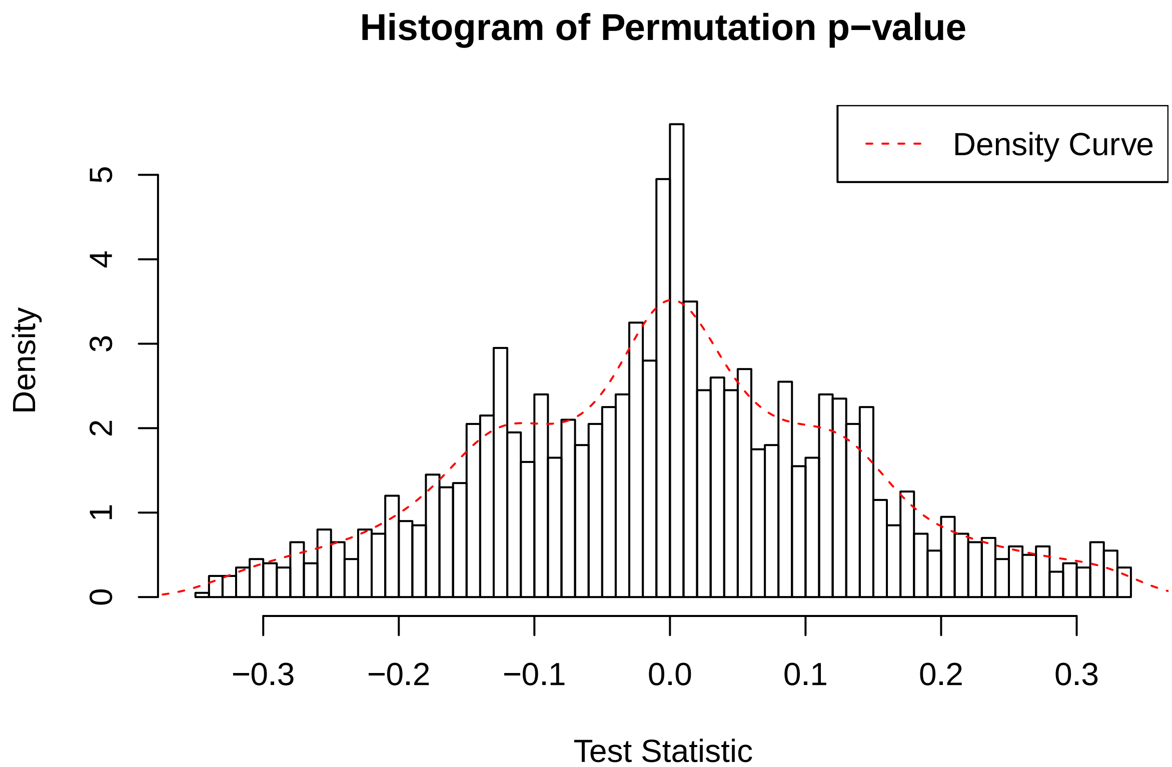

```
## p_value is:-  0.02
```

```
## T Value is:-  -0.3479163
```

```
##       2.5%      97.5%
## -0.2972837  0.3004428
```

Two - sided test was used for the given senario. Let's take $\alpha = 0.05$ for the test. Generated p-value is lower than the $\alpha = 0.05$. Hence we could reject the H_0 Hypothesis and Lottery is not random.

Part 04



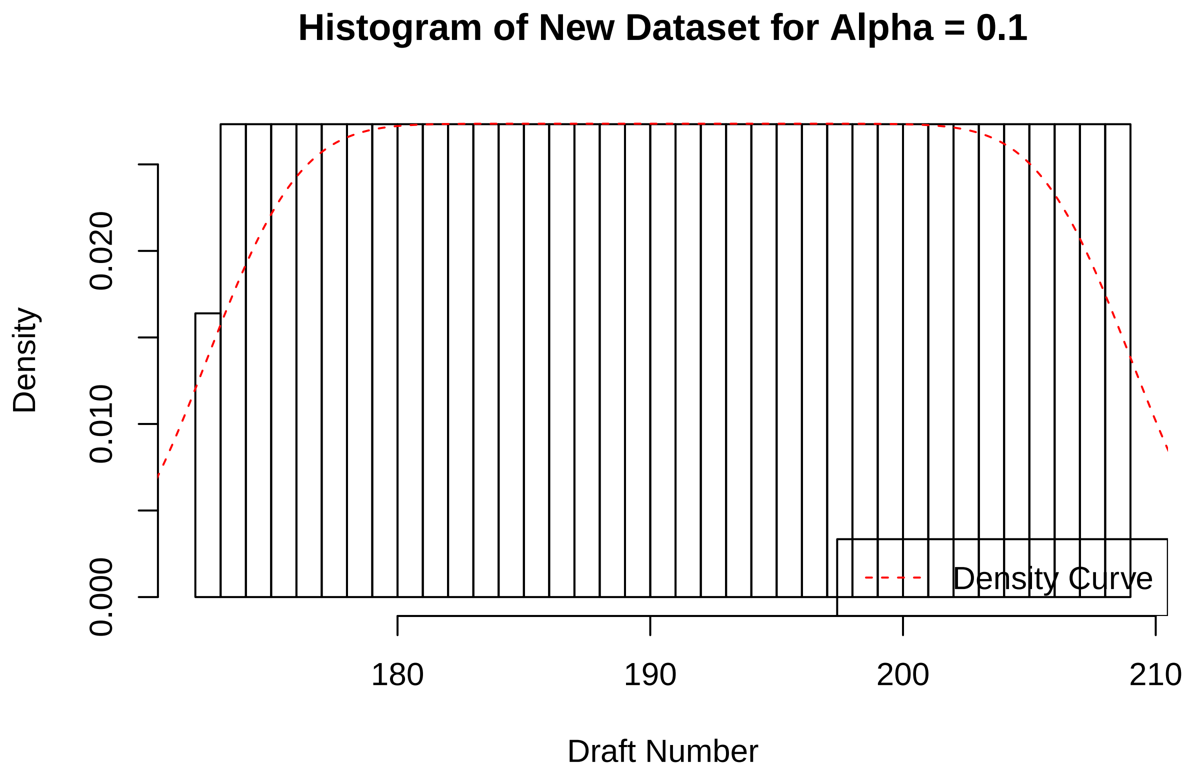

```
## p_value is:-  0
```

```
## T Value is:-  -0.3479163
```

```
##         5%        95%
## -0.2302565  0.2352898
```

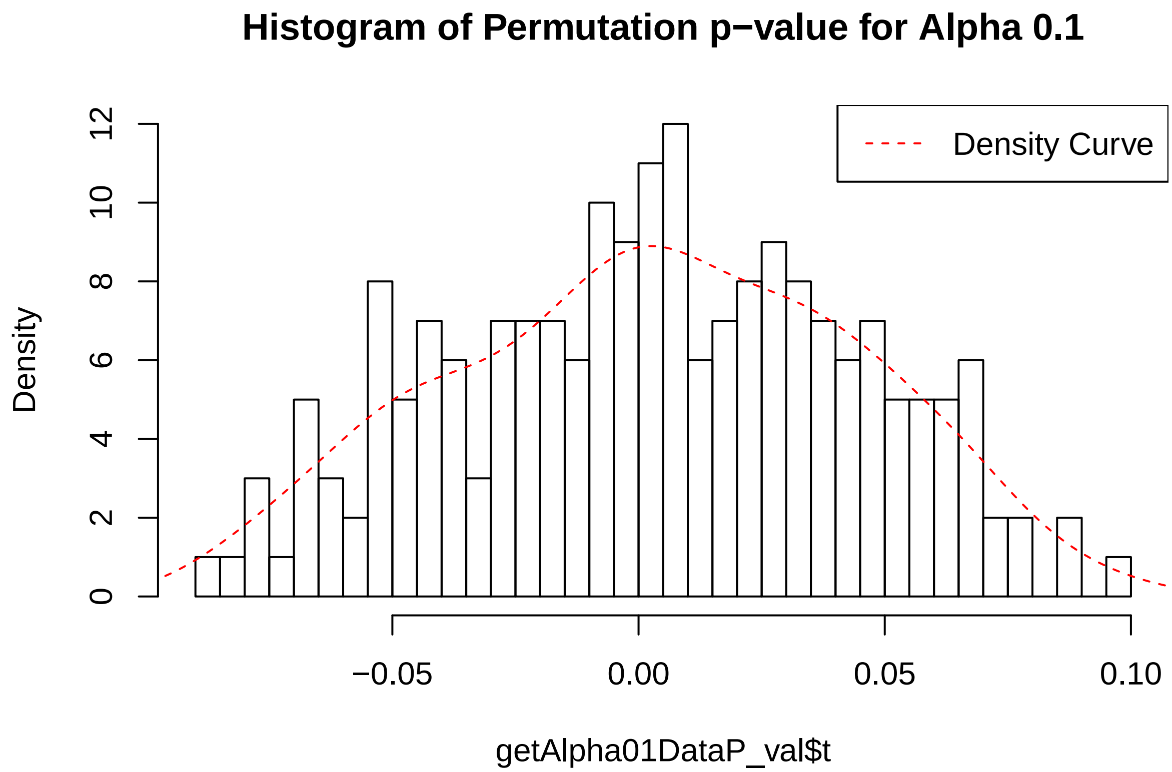
Two - sided test was used for the above senario. Generated p-value is 0 and it's below than the $\alpha = 0.05$. Hence we could reject the H_0 and Lottery is not random.

Part 05 A



Generated histogram is similar to the uniform distribution.

Part 05 B




```
## p_value is:-  0

## T Value is:-  0.1

##          5%         95%
## -0.06584932  0.06715068
```

Two - sided test was used for the above senario. Generated p-value is 0 and it's below than the $\alpha = 0.05$. Hence we could reject the H_0 and Lottery is not random.

Part 05 C

Here are the generated p-values for the α values.

```
##   Alpha p.values
## 1   0.2    0.000
## 2   0.3    0.000
## 3   0.4    0.000
## 4   0.5    0.000
## 5   0.6    0.000
## 6   0.7    0.000
## 7   0.8    0.000
```

```
## 8    0.9    0.000
## 9    1.0    0.000
## 10   1.1    0.000
## 11   1.2    0.000
## 12   1.3    0.000
## 13   1.4    0.000
## 14   1.5    0.000
## 15   1.6    0.000
## 16   1.7    0.000
## 17   1.8    0.000
## 18   1.9    0.000
## 19   2.0    0.000
## 20   2.1    0.000
## 21   2.2    0.000
## 22   2.3    0.000
## 23   2.4    0.000
## 24   2.5    0.000
## 25   2.6    0.000
## 26   2.7    0.000
## 27   2.8    0.000
## 28   2.9    0.000
## 29   3.0    0.000
## 30   3.1    0.000
## 31   3.2    0.000
## 32   3.3    0.000
## 33   3.4    0.000
## 34   3.5    0.000
## 35   3.6    0.000
## 36   3.7    0.000
## 37   3.8    0.000
## 38   3.9    0.000
## 39   4.0    0.000
## 40   4.1    0.000
## 41   4.2    0.000
## 42   4.3    0.000
## 43   4.4    0.000
## 44   4.5    0.000
## 45   4.6    0.000
## 46   4.7    0.000
## 47   4.8    0.005
## 48   4.9    0.000
## 49   5.0    0.000
## 50   5.1    0.000
## 51   5.2    0.000
## 52   5.3    0.000
## 53   5.4    0.000
## 54   5.5    0.000
## 55   5.6    0.000
## 56   5.7    0.000
## 57   5.8    0.000
## 58   5.9    0.000
## 59   6.0    0.000
## 60   6.1    0.000
## 61   6.2    0.000
```

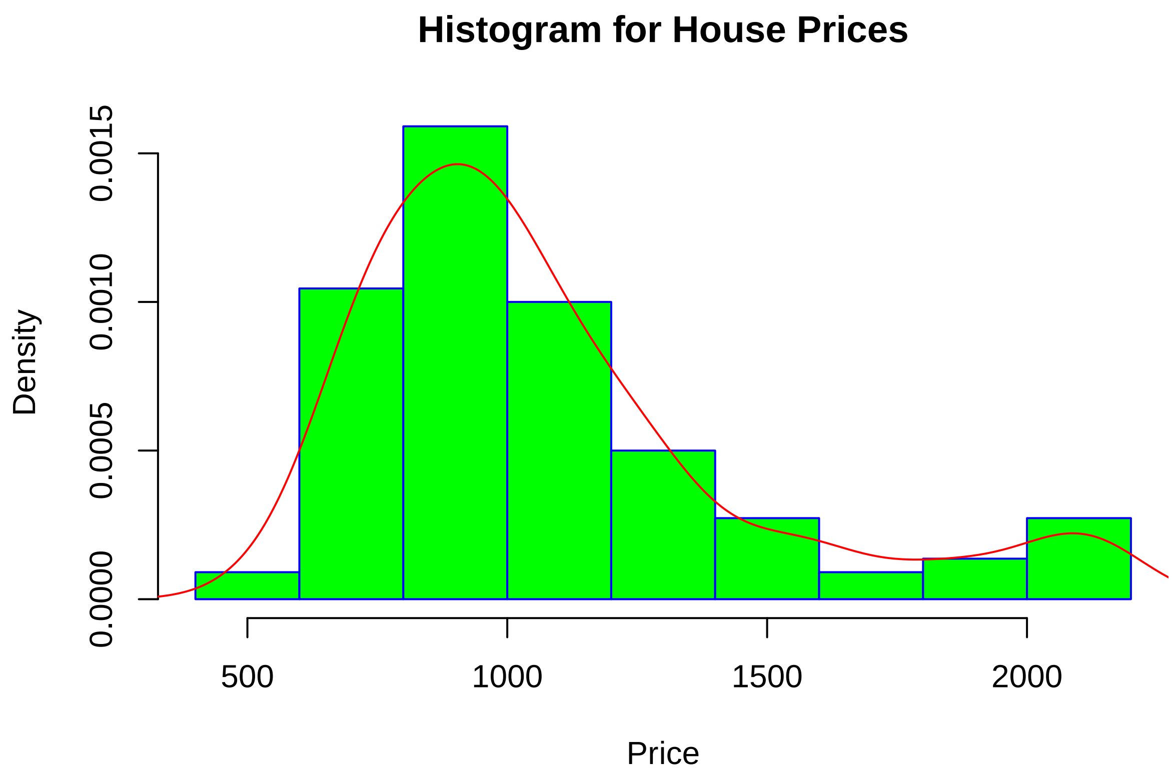
```
## 62   6.3    0.000
## 63   6.4    0.000
## 64   6.5    0.000
## 65   6.6    0.000
## 66   6.7    0.000
## 67   6.8    0.000
## 68   6.9    0.000
## 69   7.0    0.000
## 70   7.1    0.000
## 71   7.2    0.000
## 72   7.3    0.000
## 73   7.4    0.000
## 74   7.5    0.000
## 75   7.6    0.000
## 76   7.7    0.000
## 77   7.8    0.000
## 78   7.9    0.000
## 79   8.0    0.000
## 80   8.1    0.000
## 81   8.2    0.000
## 82   8.3    0.000
## 83   8.4    0.000
## 84   8.5    0.000
## 85   8.6    0.000
## 86   8.7    0.000
## 87   8.8    0.000
## 88   8.9    0.000
## 89   9.0    0.000
## 90   9.1    0.000
## 91   9.2    0.000
## 92   9.3    0.000
## 93   9.4    0.000
## 94   9.5    0.000
## 95   9.6    0.000
## 96   9.7    0.000
## 97   9.8    0.000
## 98   9.9    0.000
## 99  10.0    0.000
```

All p-values are equal to 0 and all values are rejecting H_0.

Lower the significance level α, the lower the power of the test. If significance level reduced, the acceptance region gets bigger. Hence it's less likely to reject H_0. and less likely to reject the H_0 when it is false, so it's more likely to make a Type II error. Finally, the power of the test is reduced when the significance level reduces.

Question 2: Bootstrap, jackknife and confidence intervals

Part 01




```
## Mean House Price :-  1080.473
```

Part 02

Bootstrap Variance Estimator :-

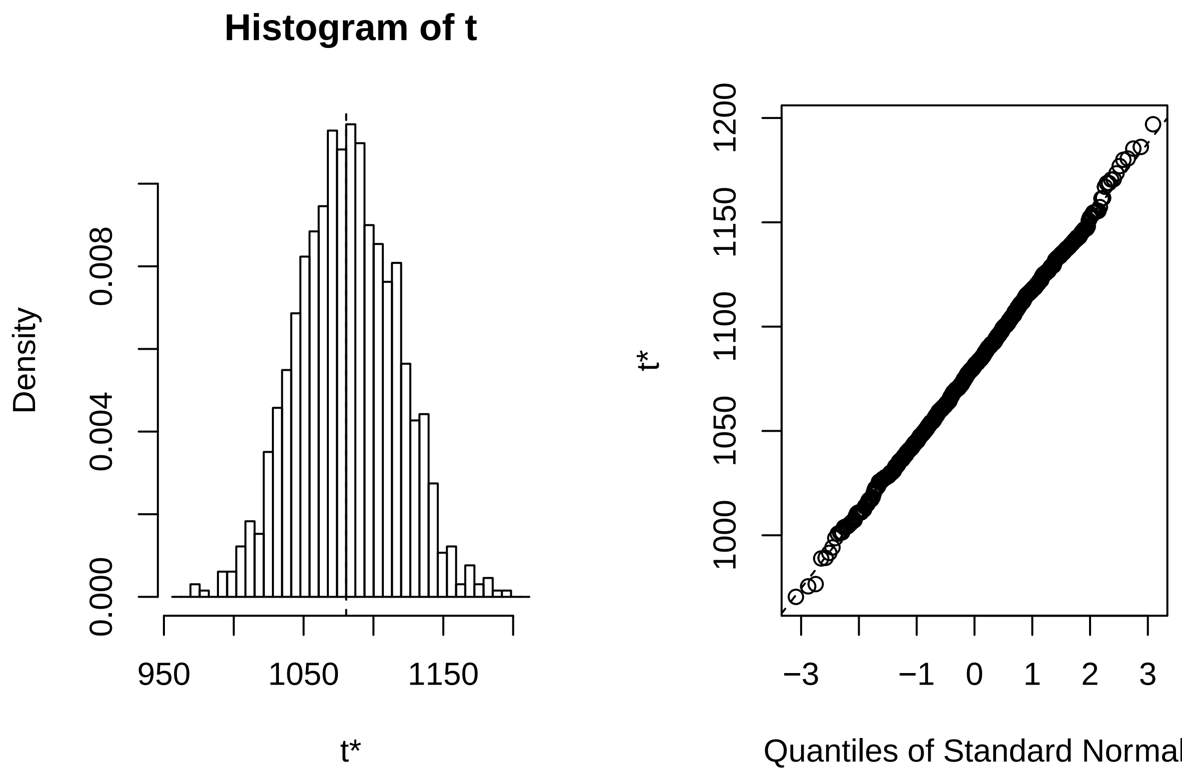
$$Var[\hat{T}(.)] = \frac{1}{B-1}\sum_{i=1}^{B}(T(D_i^*)) - \overline{T(D^*)})^2$$

Bias Correction :-

$$T_1 = 2T(D) - \frac{1}{B}\sum_{i=1}^{B} T_i^*$$

```
## Estimated Variance of the Mean :-  1265.359

## Bootstrap Bias Correction :-  1079.682
```




```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = houseBootData)
##
## Intervals :
## Level      Normal              Basic
## 95%   (1010, 1149 )   (1013, 1150 )
##
## Level     Percentile            BCa
## 95%   (1011, 1148 )   (1012, 1153 )
## Calculations and Intervals on Original Scale
```

Part 03

JackKnife Variance of Estimator :-

$$Var[\hat{T}(.)] = \frac{1}{n(n-1)} \sum_{i=1}^{n} ((T_i^*) - \overline{T_i^*})^2$$

Where ;

$$T_i^* = nT(D) - (n-1)T(D_i^*)$$

$$\overline{T_i^*} = J(T) = \frac{1}{n}\sum_{i=1}^{n} T_i^*$$

```
## Jackknief Estimated Variance of the mean :- 1320.911
```

```
## Bootstrap Variance Estimate :-  1265.359
```

Jackknief method gives a much larger Estimated Variance of the mean rather than the Bootstrap Estimated Variance of the mean.

Part 04

For large n, the jackknife estimate is approximately normally distributed about the true parameter mean. A 95% confidence interval for mean can be estimated as

$$\hat{\theta} + t_{0.975,n-1}\sqrt{Var(\hat{\theta})}$$
$$\hat{\theta} - t_{0.975,n-1}\sqrt{Var(\hat{\theta})}$$

```
##                Lower    Upper     Mean Length
## Percentile 1010.991 1148.014 1079.888   1000
## BCa        1012.467 1152.808 1079.888   1000
## Normal     1009.962 1149.402 1079.888   1000
## Jackknife  1010.489 1150.456 1080.473    110
```

References

https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470906514.app2

https://stattrek.com/hypothesis-test/power-of-test.aspx

https://stats.stackexchange.com/questions/20701/computing-p-value-using-bootstrap-with-r

APPENDIX

```
knitr::opts_chunk$set(echo = TRUE)
RNGversion(min(as.character(getRversion()),"3.6.2"))
set.seed(12345, kind = "Mersenne-Twister", normal.kind = "Inversion")
# Data Loading
library(readxl)
lotteryData <- read.csv(file.choose(),sep = ";")

# Y = Draft_No
# X = Day_of_year
# Part 01

plotDaftVsYear = function() {
  plot(x = lotteryData$Day_of_year,
```

```
       y = lotteryData$Draft_No,
       pch = 19,
       cex = 0.3,
       type = "p",
       col = "black",
       xlab = "Days",
       ylab = "Draft Number",
       main = "Draft No Vs Day of year"
  )
}

plotDaftVsYear()
# Part 02

yFit = loess(formula = Draft_No ~ Day_of_year,
             data = lotteryData)

yPred = predict(yFit)

plotDaftVsYearWithLosses = function() {
  plot(x = lotteryData$Day_of_year,
       y = lotteryData$Draft_No,
       pch = 19,
       cex = 0.3,
       type = "p",
       col = "black",
       xlab = "Days",
       ylab = "Draft Number",
       main = "Draft No Vs Day of year with Losses"
  )
  lines(x = lotteryData$Day_of_year,
       y = yPred,
       col = "blue"
  )
  # legend("bottomright",
  #        legend = c("Draft No Vs Day of year ","Losses Line"),
  #        col = c("black","blue"),
  #        pch = c(19,NA),
  #        lty = c(NA,1),
  #        cex = 1)

}

plotDaftVsYearWithLosses()

# Part 03
getTestStat = function(testdata) {

  bootstrap_fit = loess(formula = Draft_No ~ Day_of_year,
                        data = testdata)
  yBootPred = predict(bootstrap_fit)

  X_b = testdata$Day_of_year[which(yBootPred == max(yBootPred))][1]
```

```r
  X_a = testdata$Day_of_year[which(yBootPred == min(yBootPred))][1]

  # Test Statistics
  if (X_a == X_b) {
    return(0)
  } else {
    T_value = (yBootPred[X_b] - yBootPred[X_a]) / (X_b - X_a)
    return(T_value)
  }

}

getBootPValue = function(B,casedata,oneSide = T) {
  bootStat = numeric(B)
  n = dim(casedata)[1]

  for (b in 1:B) {
    # create new sample with Replacement(Bootstrap method)
    generated_bs = sample(casedata$Day_of_year, n, replace = T)
    newTestDB = casedata # Copy original DB
    newTestDB$Draft_No = newTestDB$Draft_No[generated_bs] # Append new sample
    newTestDB$Day_of_year = newTestDB$Day_of_year[generated_bs] # Append new sample
    bootStat[b] = getTestStat(newTestDB)
  }

  bootStat0 = getTestStat(casedata)
  test_p_val = 0

  if (oneSide == T) {
    test_p_val = mean(bootStat > bootStat0)
  } else {
    test_p_val = mean(abs(bootStat) > abs(bootStat0))
  }

  returnData = list("t0" = bootStat0,
                    "t" = bootStat,
                    "p_Value" = test_p_val)
  return(returnData)
}

getBootstrpPdata = getBootPValue(B = 2000,
                                 casedata = lotteryData,
                                 oneSide = F)
hist(getBootstrpPdata$t,
     breaks = 50,
     probability = T,
     main = 'Histogram of Bootstrap p-value',
     xlab = 'Test Statistic')
lines(density(getBootstrpPdata$t),
      col = 'red',
      lty = 2)
legend("topright",
       legend = c("Density Curve"),
```

```
       col = c("red"),
       lty = c(2),
       cex = 1)
cat('p_value is:- ', getBootstrpPdata$p_Value, '\n')
cat('T Value is:- ', getBootstrpPdata$t0, '\n')
print(quantile(getBootstrpPdata$t,c(0.025,0.975)))

# Part 04

getPermutationTestPValue = function(B,casedata,oneSide = T) {
  bootStat = numeric(B)
  n = dim(casedata)[1]

  for (b in 1:B) {
    # create new sample without replacement(Permutation method)
    generated_bs = sample(casedata$Day_of_year, n, replace = F) # create new sample
    newTestDB = casedata # Copy original DB
    newTestDB$Draft_No = newTestDB$Draft_No[generated_bs] # Append new sample
    newTestDB$Day_of_year = newTestDB$Day_of_year[generated_bs] # Append new sample
    bootStat[b] = getTestStat(newTestDB)
  }

  bootStat0 = getTestStat(casedata)
  test_p_val = 0

  if (oneSide == TRUE) {
    test_p_val = mean(bootStat > bootStat0)
  } else {
    test_p_val = mean(abs(bootStat) > abs(bootStat0))
  }

  returnData = list("t0" = bootStat0,
                    "t" = bootStat,
                    "p_Value" = test_p_val)
  return(returnData)
}


permu_text_data = getPermutationTestPValue(B = 2000,
                                           casedata = lotteryData,
                                           oneSide = F)

hist(permu_text_data$t,
     breaks = 50,
     probability = T,
     main = 'Histogram of Permutation p-value',
     xlab = 'Test Statistic')
lines(density(permu_text_data$t),
      col = 'red',
      lty = 2)
legend("topright",
       legend = c("Density Curve"),
       col = c("red"),
```

```
       lty = c(2),
       cex = 1)

cat('p_value is:- ', permu_text_data$p_Value, '\n')
cat('T Value is:- ', permu_text_data$t0, '\n')
print(quantile(permu_text_data$t,c(0.05,0.95)))

generateNewDataset = function(alpha,userdataset) {

  x_data = userdataset$Day_of_year
  betaValue = rnorm(1,
                    mean = 183,
                    sd = 10)
  newY_values = c()
  for (index in 1:length(x_data)) {
    generatedValue = (alpha * x_data[index]) + betaValue
    newY_values = c(newY_values, max(c(0, min(c(generatedValue, 366)))))
  }

  userdataset$Draft_No = newY_values
  return(userdataset)
}

alpha01Dataset = generateNewDataset(0.1,lotteryData)
hist(alpha01Dataset$Draft_No,
     breaks = 50,
     probability = T,
     main = 'Histogram of New Dataset for Alpha = 0.1',
     xlab = 'Draft Number')
lines(density(alpha01Dataset$Draft_No),
      col = 'red',
      lty = 2)
legend("bottomright",
       legend = c("Density Curve"),
       col = c("red"),
       lty = c(2),
       cex = 1)
# Part b

getAlpha01DataP_val = getPermutationTestPValue(B = 200,
                                               casedata = alpha01Dataset,
                                               oneSide = F)
hist(getAlpha01DataP_val$t,
     breaks = 50,
     probability = T,
     main = 'Histogram of Permutation p-value for Alpha 0.1')
lines(density(getAlpha01DataP_val$t),
      col = 'red',
      lty = 2)
legend("topright",
       legend = c("Density Curve"),
       col = c("red"),
       lty = c(2),
```

```
      cex = 1)

cat('p_value is:- ', getAlpha01DataP_val$p_Value, '\n')
cat('T Value is:- ', getAlpha01DataP_val$t0, '\n')
print(quantile(getAlpha01DataP_val$t,c(0.05,0.95)))

alphaSeq = seq(from = 0.2,
               to = 10,
               by = 0.1)

alphaPValues = c()

for (alpha in alphaSeq) {

  newdataset = generateNewDataset(alpha,lotteryData)
  getNewdataP_val = getPermutationTestPValue(B = 200,
                                             casedata = newdataset,
                                             oneSide = F)
  alphaPValues = c(alphaPValues, getNewdataP_val$p_Value)
}

printAlphaRejetionTable = function() {
  rejectionTable = data.frame('Alpha' = alphaSeq,
                              'p-values' = alphaPValues)
  print(rejectionTable)
}

printAlphaRejetionTable()
library(readxl)
library(boot)

homePricesData <- read.csv(file.choose(),sep = ";")
hist(homePricesData$Price,
     main = "Histogram for House Prices",
     xlab = "Price",
     border = "blue",
     col = "green",
     prob = TRUE)
lines(density(homePricesData$Price),
      col = "red")

meanHousePrice = mean(homePricesData$Price)

cat('Mean House Price :- ', meanHousePrice, '\n')
# Part 02

getBootdata = function(userdata,B) {

  calcBootStat = function(data, indices) {
    selectedData = data[indices,] # allows boot to select sample
    c(mean(selectedData$Price))
  }
```

```r
  boot_data = boot(data = userdata,
                   statistic = calcBootStat,
                   R = B)

  return(boot_data)
}

getBootVarEstimate = function(bootdata) {
  B = 1000 # Bootstrap
  bootstrap_var_stat = (1 / (B - 1)) * sum((bootdata$t - mean(bootdata$t)) ** 2)
  return(bootstrap_var_stat)
}

getBootBiasCorrection = function(bootdata) {
  bias_correction = (2 * mean(homePricesData$Price)) - mean(bootdata$t)
  return(bias_correction)
}

houseBootData = getBootdata(homePricesData, 1000)
cat('Estimated Variance of the Mean :- ', getBootVarEstimate(houseBootData), '\n')
cat('Bootstrap Bias Correction :- ', getBootBiasCorrection(houseBootData), '\n')

plot(houseBootData) # Plot bootstrap data
boot.ci(houseBootData) # Bootstrap confidence Interval

# Part 03

# Jackknief Mean

getJKMean = function(userdata) {
  jk_mean = c()

  for (index in 1:nrow(userdata)) {
    selectedset = userdata[-index,]
    # get mean estimator (In this case mean price)
    jk_mean[index] = mean(selectedset$Price)
  }
  return(jk_mean)
}


getJKVarianceEstimate = function(userdata) {
  jk_mean_val = getJKMean(userdata)

  n = nrow(userdata)
  #Calculate Ti
  t_i = (n * mean(userdata$Price)) - ((n - 1) * jk_mean_val)

  #Calculate Ti - J(T)
  j_t = t_i - mean(t_i)

  return( (1 / (n * (n - 1))) * sum(j_t ** 2))
}
```

```r
#getJKVarianceEstimate(homePricesData)
#getJKMean(homePricesData)
cat('Jackknief Estimated Variance of the mean :-', getJKVarianceEstimate(homePricesData), '\n')
cat('Bootstrap Variance Estimate :- ', getBootVarEstimate(houseBootData), '\n')

# Part 04

getJKConfidenceInterval = function(confLevel,userdata) {
  jk_mean = mean(getJKMean(userdata))
  hbootdata = getBootdata(userdata, 1000)
  var_est = getBootVarEstimate(hbootdata)
  n = nrow(userdata)
  ci = confLevel + ((1 - confLevel) / 2)

  lower_limit = jk_mean - (qt(ci, n - 1) * (sqrt(var_est)))
  upper_limit = jk_mean + (qt(ci, n - 1) * (sqrt(var_est)))

  return(list('Lower' = lower_limit,
              'Upper' = upper_limit,
              'Mean' = mean(getJKMean(homePricesData))))
}

jkdata = getJKConfidenceInterval(0.95,homePricesData)
genBootData = getBootdata(homePricesData, 1000)
genBootCIData = boot.ci(houseBootData)

com_lower_vect = c(genBootCIData$percent[4],
                   genBootCIData$bca[4],
                   genBootCIData$normal[2],
                   jkdata$Lower)

com_upper_vect = c(genBootCIData$percent[5],
                   genBootCIData$bca[5],
                   genBootCIData$normal[3],
                   jkdata$Upper)

com_mean_vect = c(mean(genBootData$t),
                  mean(genBootData$t),
                  mean(genBootData$t),
                  jkdata$Mean)

com_len_vect = c(1000,
                 1000,
                 1000,
                 nrow(homePricesData))


comp_dataset = data.frame('Lower' = com_lower_vect,
                          'Upper' = com_upper_vect,
                          'Mean' = com_mean_vect,
                          'Length' = com_len_vect)

rownames(comp_dataset) = c('Percentile',
```

```
                                'BCa',
                                'Normal',
                                'Jackknife')

print(comp_dataset)
```