Practical 10: Decision tree

```
mydata<-data.frame(iris)
```

This line creates a data frame called **`mydata`** that contains the iris dataset.

```
attach(mydata)
```

This line attaches the **`mydata`** data frame to the search path, allowing us to refer to variables in the data frame without specifying the data frame name each time

```
install.packages("rpart")
library(rpart)
```

These two lines install and load the **`rpart`** package, which provides functions for building decision trees.

```
model<-
rpart(Species~Sepal.Length+Sepal.Width+Petal.Length+Petal.Width,data=mydata,method="class")
```

This line builds a decision tree model using the **`rpart()`** function. The model predicts the species of flower (**`Species`**) based on four predictor variables (**`Sepal.Length`**, **`Sepal.Width`**, **`Petal.Length`**, and **`Petal.Width`**) using a classification method. The resulting model is stored in the **`model`** object.

```
plot(model)
text(model,use.n=TRUE,all=TRUE,cex=0.8)
```

These two lines create a plot of the decision tree model using the **`plot()`** function and add text labels to the plot using the **`text()`** function.

```
install.packages("tree")
library(tree)
```

These two lines install and load the `tree` package, which also provides functions for building decision trees.

```
model1<-
tree(Species~Sepal.Length+Sepal.Width+Petal.Length+Petal.Width,data=mydata,method="class",split
="gini")
```

This line builds a decision tree model using the `tree()` function. The model predicts the species of flower (`Species`) based on four predictor variables (`Sepal.Length`, `Sepal.Width`, `Petal.Length`, and `Petal.Width`) using a classification method and the Gini impurity measure. The resulting model is stored in the `model1` object.

```
plot(model1)
text(model1,all=TRUE,cex=0.6)
```

These two lines create a plot of the decision tree model using the `plot()` function and add text labels to the plot using the `text()` function.

```
install.packages("party")
library(party)
```

These two lines install and load the `party` package, which provides more advanced functions for building decision trees.

```
model2<-ctree(Species~Sepal.Length+Sepal.Width+Petal.Length+Petal.Width,data=mydata)
plot(model2)
```

These two lines build a conditional inference tree model using the `ctree()` function. The model predicts the species of flower (`Species`) based on four predictor variables

(`Sepal.Length`, `Sepal.Width`, `Petal.Length`, and `Petal.Width`) using a conditional inference method. The resulting model is stored in the `model2` object and a plot of the model is created using the `plot()` function.

```
model1<-
tree(Species~Sepal.Length+Sepal.Width+Petal.Length+Petal.Width,data=mydata,method="class",con
trol=tree.control(nobs=150,mincut=10))

plot(model1)

text(model1,all=TRUE,cex=0.6)
```

These two lines build a decision tree model using the `tree()` function with some additional control parameters. The `nobs` parameter specifies the number of observations to be used in the model and the `

---

Practical 9: anova

The code performs a basic analysis of variance (ANOVA) using a one-way design, which is used to test if there are significant differences among the means of three or more groups. Here's a step-by-step breakdown of the code:

1. Three sets of data (y1, y2, and y3) are defined, each containing 7 observations.
2. The number of observations in each group is stored in the 'n' variable using the 'rep' function.
3. The 'group' variable is defined as a factor with values 1, 2, and 3, representing the three groups of data.

4. The 'stem' function is used to display the stems and leaves of the combined data.
5. The 'tapply' function is used to compute summary statistics (sum, mean, variance, and sample size) for each group of data.
6. A data frame is created containing the combined data and group information.
7. A linear regression model is fitted using the 'lm' function, with 'group' as the independent variable and 'y' as the dependent variable.
8. An ANOVA table is generated using the 'anova' function.
9. The degrees of freedom for the treatment (trt) and error (err) are extracted from the ANOVA table and stored in the 'df' variable.
10. The F-statistic values corresponding to two different levels of significance (alpha=0.05 and alpha=0.01) are calculated using the 'qf' function.
11. The sum of squares for the residual (error) term is extracted from the ANOVA table.
12. The two-tailed critical values of the F-distribution are calculated using the 'qchisq' function and are used to compute the confidence intervals for the mean differences among groups.

# Practical 8: Hypothesis(T-test)

---

The first part of the code performs a one-sample t-test on a set of 23 observations in the 'x' variable. Here's a step-by-step breakdown:

1. A set of 23 observations is defined and stored in the 'x' variable.
2. The 't.test' function is used to perform a one-sample t-test on 'x' with a null hypothesis that the population mean is 9. The 'alternative' argument is set to "two.sided" to perform a two-tailed test. The 'conf.level' argument is set to 0.95 to calculate a 95% confidence interval.
3. The output of the 't.test' function is printed to the console.

The second part of the code performs a two-sample t-test on two sets of data in the 'x' and 'y' variables. Here's a step-by-step breakdown:

1. Two sets of data are defined and stored in the 'x' and 'y' variables, respectively.
2. The 't.test' function is used to perform a two-sample t-test on 'x' and 'y' with a null hypothesis that the population means are equal. The 'alternative' argument is set to "two.sided" to perform a two-tailed test. The 'mu' argument is set to 0 to test the difference between means. The 'var.equal' argument is set to FALSE to perform a Welch's t-test that does not assume equal variances in the two populations. The 'conf.level' argument is set to 0.95 to calculate a 95% confidence interval.
3. The output of the 't.test' function is stored in the 'test2' variable.
4. The output of the 't.test' function is printed to the console.

---

---

# Practical 7:

---

This code performs a logistic regression analysis on the iris dataset and uses the resulting model to predict the species of flowers in a test set.

First, the `iris` dataset is loaded from the `datasets` package and stored in `ir_data`. The `head()` function is used to display the first few rows of the dataset and `str()` is used to display the structure of the dataset.

Next, `levels(ir_data$species)` is used to display the levels of the `Species` variable, which is a factor variable with three levels: "setosa", "versicolor", and "virginica".

The code then checks if there are any missing values in the dataset using `sum(is.na(ir_data))` and subsets the dataset to contain only the first 100 rows and 4 columns using `ir_data<-ir_data[1:100,]`.

The code then samples 80 rows from the dataset and assigns them to `ir_test`, while the remaining 20 rows are assigned to `ir_ctrl`. The `ggpairs()` function from the `GGally` package is used to create a matrix of scatterplots to visualize the relationships between the variables in the `ir_test` dataset.

Next, a logistic regression model is fit to the `ir_test` dataset using `glm()` with the formula `y~x`, where `y` is the dependent variable `Species` and `x` is the independent variable `Sepal.Length`. The `family` argument is set to "binomial" to specify a logistic regression model. The `summary()` function is used to display a summary of the model's coefficients.

The code then creates a new dataset `newdata` containing only the `Sepal.Length` variable from the `ir_ctrl` dataset. The `predict()` function is used to predict the probability of each flower being in the "versicolor" or "virginica" species based on the logistic regression model fitted to the `ir_test` dataset.

The predicted values are then plotted against `Sepal.Length` using the `qplot()` function from the `ggplot2` package. The color of the points corresponds to the actual `Species` variable in the `ir_ctrl` dataset. The resulting plot shows the logistic regression model's predictions for each value of `Sepal.Length` in the `ir_ctrl` dataset.

---

---

## Practical 5:

---

This code is working with the `AirPassengers` dataset, which is a built-in time series dataset in R that contains the monthly total number of passengers for an airline from 1949 to 1960.

Here is what each line of code does:

1. `data(AirPassengers)` - loads the `AirPassengers` dataset into the current R session.

2. `class(AirPassengers)` - checks the class of the `AirPassengers` dataset, which is "ts" (time series).
3. `start(AirPassengers)` - returns the start date of the time series, which is January 1949.
4. `end(AirPassengers)` - returns the end date of the time series, which is December 1960.
5. `frequency(AirPassengers)` - returns the frequency of the time series, which is 12 (monthly data).
6. `summary(AirPassengers)` - provides summary statistics of the time series, including the minimum, maximum, mean, and quartiles.
7. `plot(AirPassengers)` - creates a time plot of the `AirPassengers` dataset.
8. `abline(reg=lm(AirPassengers~time(AirPassengers)))` - adds a linear trend line to the time plot using a linear regression model with `time(AirPassengers)` as the independent variable and `AirPassengers` as the dependent variable.
9. `cycle(AirPassengers)` - returns the cycle of the time series, which is 12 (the number of months in a year).
10. `plot(aggregate(AirPassengers,FUN=mean))` - creates a plot of the monthly mean of the `AirPassengers` dataset.
11. `boxplot(AirPassengers~cycle(AirPassengers))` - creates a boxplot of the `AirPassengers` dataset by cycle (i.e., by month).
12. `acf(log(AirPassengers))` - creates an autocorrelation plot of the log-transformed `AirPassengers` dataset.
13. `(fit<-arima(log(AirPassengers),c(0,1,1),seasonal=list(order=c(0,1,1),period=12)))` - fits an ARIMA(0,1,1)(0,1,1)[12] model to the log-transformed `AirPassengers` dataset and stores the output in the `fit` object.
14. `pred<-predict(fit,n.ahead=10*12)` - generates forecasts for the next 10 years (i.e., 120 months) using the `fit` object and stores the output in the `pred` object.
15. `ts.plot(AirPassengers,2.718^pred$pred,log="y",lty=c(1,3))` - creates a plot of the `AirPassengers` dataset and the forecasted values (which are back-transformed from the log scale to the original scale using the exponential function) with a logarithmic y-axis scale and two different line types (solid for the actual data and dashed for the forecasted values).

---

---

## Practical 4: