Practical 1

In [1]:
```python
from pymongo import MongoClient
client = MongoClient('localhost', 27017)
```

In [2]:
```python
mycoll = client.DBforETL.mycoll
print(mycoll)
```

```
Collection(Database(MongoClient(host=['localhost:27017'], document_class=dict, tz_aware=False, connect=True), 'DBforETL'), 'myc
oll')
```

In [3]:
```python
print(client.list_database_names())
```

```
['Movies_DB', 'admin', 'config', 'local', 'movie1', 'moviedb1']
```

In [4]:
```python
#inserting records
mydict = { "name": "John", "address": "Highway 37" }
x = mycoll.insert_one(mydict)
```

In [5]:
```python
mylist = [
  { "name": "Amy", "address": "Apple st 652"},
  { "name": "Hannah", "address": "Mountain 21"},
  { "name": "Michael", "address": "Valley 345"},
  { "name": "Sandy", "address": "Ocean blvd 2"},
  { "name": "Betty", "address": "Green Grass 1"},
  { "name": "Richard", "address": "Sky st 331"},
  { "name": "Susan", "address": "One way 98"},
  { "name": "Vicky", "address": "Yellow Garden 2"},
  { "name": "Ben", "address": "Park Lane 38"},
  { "name": "William", "address": "Central st 954"},
  { "name": "Chuck", "address": "Main Road 989"},
  { "name": "Viola", "address": "Sideway 1633"}
]
x = mycoll.insert_many(mylist)

#print list of the _id values of the inserted documents:
print(x.inserted_ids)
```

```
[ObjectId('5ca32c50a974580a14673634'), ObjectId('5ca32c50a974580a14673635'), ObjectId('5ca32c50a974580a14673636'), ObjectId('5c
a32c50a974580a14673637'), ObjectId('5ca32c50a974580a14673638'), ObjectId('5ca32c50a974580a14673639'), ObjectId('5ca32c50a974580
a1467363a'), ObjectId('5ca32c50a974580a1467363b'), ObjectId('5ca32c50a974580a1467363c'), ObjectId('5ca32c50a974580a1467363d'),
ObjectId('5ca32c50a974580a1467363e'), ObjectId('5ca32c50a974580a1467363f')]
```

In [6]:
```python
x = mycoll.find_one()
print(x)
```

```
{'_id': ObjectId('5ca32c4ba974580a14673633'), 'name': 'John', 'address': 'Highway 37'}
```

In [7]:
```python
for x in mycoll.find():
  print(x)
```

```
{'_id': ObjectId('5ca32c4ba974580a14673633'), 'name': 'John', 'address': 'Highway 37'}
{'_id': ObjectId('5ca32c50a974580a14673634'), 'name': 'Amy', 'address': 'Apple st 652'}
{'_id': ObjectId('5ca32c50a974580a14673635'), 'name': 'Hannah', 'address': 'Mountain 21'}
{'_id': ObjectId('5ca32c50a974580a14673636'), 'name': 'Michael', 'address': 'Valley 345'}
{'_id': ObjectId('5ca32c50a974580a14673637'), 'name': 'Sandy', 'address': 'Ocean blvd 2'}
{'_id': ObjectId('5ca32c50a974580a14673638'), 'name': 'Betty', 'address': 'Green Grass 1'}
{'_id': ObjectId('5ca32c50a974580a14673639'), 'name': 'Richard', 'address': 'Sky st 331'}
{'_id': ObjectId('5ca32c50a974580a1467363a'), 'name': 'Susan', 'address': 'One way 98'}
{'_id': ObjectId('5ca32c50a974580a1467363b'), 'name': 'Vicky', 'address': 'Yellow Garden 2'}
{'_id': ObjectId('5ca32c50a974580a1467363c'), 'name': 'Ben', 'address': 'Park Lane 38'}
{'_id': ObjectId('5ca32c50a974580a1467363d'), 'name': 'William', 'address': 'Central st 954'}
{'_id': ObjectId('5ca32c50a974580a1467363e'), 'name': 'Chuck', 'address': 'Main Road 989'}
{'_id': ObjectId('5ca32c50a974580a1467363f'), 'name': 'Viola', 'address': 'Sideway 1633'}
```

In [8]:
```python
for x in mycoll.find({},{ "_id": 1, "name":1}):
  print(x)
```

```
{'_id': ObjectId('5ca32c4ba974580a14673633'), 'name': 'John'}
{'_id': ObjectId('5ca32c50a974580a14673634'), 'name': 'Amy'}
{'_id': ObjectId('5ca32c50a974580a14673635'), 'name': 'Hannah'}
{'_id': ObjectId('5ca32c50a974580a14673636'), 'name': 'Michael'}
{'_id': ObjectId('5ca32c50a974580a14673637'), 'name': 'Sandy'}
{'_id': ObjectId('5ca32c50a974580a14673638'), 'name': 'Betty'}
{'_id': ObjectId('5ca32c50a974580a14673639'), 'name': 'Richard'}
{'_id': ObjectId('5ca32c50a974580a1467363a'), 'name': 'Susan'}
{'_id': ObjectId('5ca32c50a974580a1467363b'), 'name': 'Vicky'}
{'_id': ObjectId('5ca32c50a974580a1467363c'), 'name': 'Ben'}
{'_id': ObjectId('5ca32c50a974580a1467363d'), 'name': 'William'}
{'_id': ObjectId('5ca32c50a974580a1467363e'), 'name': 'Chuck'}
{'_id': ObjectId('5ca32c50a974580a1467363f'), 'name': 'Viola'}
```

In [9]:

```python
myquery = { "address": "Park Lane 38" }
myquery = { "address": { "$gt": "S" } }
mydoc = mycoll.find(myquery)

for x in mydoc:
  print(x)
```

```
{'_id': ObjectId('5ca32c50a974580a14673636'), 'name': 'Michael', 'address': 'Valley 345'}
{'_id': ObjectId('5ca32c50a974580a14673639'), 'name': 'Richard', 'address': 'Sky st 331'}
{'_id': ObjectId('5ca32c50a974580a1467363b'), 'name': 'Vicky', 'address': 'Yellow Garden 2'}
{'_id': ObjectId('5ca32c50a974580a1467363f'), 'name': 'Viola', 'address': 'Sideway 1633'}
```

In [10]:

```python
mydoc = mycoll.find().sort("name")
for x in mydoc:
  print(x)
```

```
{'_id': ObjectId('5ca32c50a974580a14673634'), 'name': 'Amy', 'address': 'Apple st 652'}
{'_id': ObjectId('5ca32c50a974580a1467363c'), 'name': 'Ben', 'address': 'Park Lane 38'}
{'_id': ObjectId('5ca32c50a974580a14673638'), 'name': 'Betty', 'address': 'Green Grass 1'}
{'_id': ObjectId('5ca32c50a974580a1467363e'), 'name': 'Chuck', 'address': 'Main Road 989'}
{'_id': ObjectId('5ca32c50a974580a14673635'), 'name': 'Hannah', 'address': 'Mountain 21'}
{'_id': ObjectId('5ca32c4ba974580a14673633'), 'name': 'John', 'address': 'Highway 37'}
{'_id': ObjectId('5ca32c50a974580a14673636'), 'name': 'Michael', 'address': 'Valley 345'}
{'_id': ObjectId('5ca32c50a974580a14673639'), 'name': 'Richard', 'address': 'Sky st 331'}
{'_id': ObjectId('5ca32c50a974580a14673637'), 'name': 'Sandy', 'address': 'Ocean blvd 2'}
{'_id': ObjectId('5ca32c50a974580a1467363a'), 'name': 'Susan', 'address': 'One way 98'}
{'_id': ObjectId('5ca32c50a974580a1467363b'), 'name': 'Vicky', 'address': 'Yellow Garden 2'}
{'_id': ObjectId('5ca32c50a974580a1467363f'), 'name': 'Viola', 'address': 'Sideway 1633'}
{'_id': ObjectId('5ca32c50a974580a1467363d'), 'name': 'William', 'address': 'Central st 954'}
```

In [11]:

```python
mydoc = mycoll.find().sort("name", -1)
for x in mydoc:
  print(x)
```

```
{'_id': ObjectId('5ca32c50a974580a1467363d'), 'name': 'William', 'address': 'Central st 954'}
{'_id': ObjectId('5ca32c50a974580a1467363f'), 'name': 'Viola', 'address': 'Sideway 1633'}
{'_id': ObjectId('5ca32c50a974580a1467363b'), 'name': 'Vicky', 'address': 'Yellow Garden 2'}
{'_id': ObjectId('5ca32c50a974580a1467363a'), 'name': 'Susan', 'address': 'One way 98'}
{'_id': ObjectId('5ca32c50a974580a14673637'), 'name': 'Sandy', 'address': 'Ocean blvd 2'}
{'_id': ObjectId('5ca32c50a974580a14673639'), 'name': 'Richard', 'address': 'Sky st 331'}
{'_id': ObjectId('5ca32c50a974580a14673636'), 'name': 'Michael', 'address': 'Valley 345'}
{'_id': ObjectId('5ca32c4ba974580a14673633'), 'name': 'John', 'address': 'Highway 37'}
{'_id': ObjectId('5ca32c50a974580a14673635'), 'name': 'Hannah', 'address': 'Mountain 21'}
{'_id': ObjectId('5ca32c50a974580a1467363e'), 'name': 'Chuck', 'address': 'Main Road 989'}
{'_id': ObjectId('5ca32c50a974580a14673638'), 'name': 'Betty', 'address': 'Green Grass 1'}
{'_id': ObjectId('5ca32c50a974580a1467363c'), 'name': 'Ben', 'address': 'Park Lane 38'}
{'_id': ObjectId('5ca32c50a974580a14673634'), 'name': 'Amy', 'address': 'Apple st 652'}
```

In [12]:

```python
myquery = { "address": "Mountain 21" }
mycoll.delete_one(myquery)
```

Out[12]:

```
<pymongo.results.DeleteResult at 0x20af7ed13c8>
```

In [13]:

```python
myquery = { "address": {"$regex": "^S"} }
x = mycoll.delete_many(myquery)
print(x.deleted_count, " documents deleted.")
```

```
2  documents deleted.
```

In [14]:

```python
myquery = { "address": "Valley 345" }
newvalues = { "$set": { "address": "Canyon 123" } }
mycoll.update_one(myquery, newvalues)
#print "customers" after the update:
for x in mycoll.find():
  print(x)
```

```
{'_id': ObjectId('5ca32c4ba974580a14673633'), 'name': 'John', 'address': 'Highway 37'}
{'_id': ObjectId('5ca32c50a974580a14673634'), 'name': 'Amy', 'address': 'Apple st 652'}
{'_id': ObjectId('5ca32c50a974580a14673636'), 'name': 'Michael', 'address': 'Canyon 123'}
{'_id': ObjectId('5ca32c50a974580a14673637'), 'name': 'Sandy', 'address': 'Ocean blvd 2'}
{'_id': ObjectId('5ca32c50a974580a14673638'), 'name': 'Betty', 'address': 'Green Grass 1'}
{'_id': ObjectId('5ca32c50a974580a1467363a'), 'name': 'Susan', 'address': 'One way 98'}
{'_id': ObjectId('5ca32c50a974580a1467363b'), 'name': 'Vicky', 'address': 'Yellow Garden 2'}
{'_id': ObjectId('5ca32c50a974580a1467363c'), 'name': 'Ben', 'address': 'Park Lane 38'}
{'_id': ObjectId('5ca32c50a974580a1467363d'), 'name': 'William', 'address': 'Central st 954'}
{'_id': ObjectId('5ca32c50a974580a1467363e'), 'name': 'Chuck', 'address': 'Main Road 989'}
```

In [15]:

```python
myresult = mycoll.find().limit(5)
#print the result:
for x in myresult:
  print(x)
```

```
{'_id': ObjectId('5ca32c4ba974580a14673633'), 'name': 'John', 'address': 'Highway 37'}
{'_id': ObjectId('5ca32c50a974580a14673634'), 'name': 'Amy', 'address': 'Apple st 652'}
{'_id': ObjectId('5ca32c50a974580a14673636'), 'name': 'Michael', 'address': 'Canyon 123'}
{'_id': ObjectId('5ca32c50a974580a14673637'), 'name': 'Sandy', 'address': 'Ocean blvd 2'}
{'_id': ObjectId('5ca32c50a974580a14673638'), 'name': 'Betty', 'address': 'Green Grass 1'}
```

Practical 2

In [1]:
```
## Remember to import the following libraries
import pymongo
from pymongo import MongoClient
import pprint
#from IPython.display import clear_output
```

In [4]:
```
## We start with establishing the connection to our local mongo db cluster using MongoClient()
client=MongoClient('localhost',27017)
```

In [6]:
```
## Checking what databases are present in the current MongoDb cluster
print(client.list_database_names())
```

```
['Movies_DB', 'admin', 'config', 'local', 'movie1', 'moviedb1']
```

In [7]:
```
## Creating a database entry point
MovieDB=client.moviedb1
```

In [8]:
```
## Checking the collection in current database
print(MovieDB.list_collection_names())
```

```
['moviecollection']
```

In [9]:
```
## Creating a collection entry point
Movie_col=MovieDB.moviecollection
```

In [7]:
```
#Listing sample of 100 documents
pprint.pprint(list(Movie_col.find().limit(100)))
```

```
[{'_id': ObjectId('5c4d36add298a5c5dc777e54'),
  'awards': '',
  'cast': 'Carmencita',
  'country': 'USA',
  'director': 'William K.L. Dickson',
  'fullplot': 'Performing on what looks like a small wooden stage, wearing a '
              'dress with a hoop skirt and white high-heeled pumps, Carmencita '
              'does a dance with kicks and twirls, a smile always on her face.',
  'genre': 'Documentary, Short',
  'imdbID': 1,
  'imdbRating': 5.9,
  'imdbVotes': 1032,
  'language': '',
  'lastupdated': '2015-08-26 00:03:45.040000000',
  'metacritic': '',
  'plot': 'Performing on what looks like a small wooden stage, wearing a dress '
          'with a hoop skirt and white high-heeled pumps, Carmencita does a '
          'dance with kicks and twirls, a smile always on her face.',
  'poster': 'http://ia.media-imdb.com/images/M/MV5BMjAzNDEwMzk3OV5BMl5BanBnXkFtZTcwOTk4OTM5Ng@@._V1_SX300.jpg',
  'rating': 'NOT RATED',
```

In [8]:
```
pipeline=[
    {
        '$sortByCount':"$language"
    }
   ,
    {
        '$limit':10
    }
]
pprint.pprint(list(Movie_col.aggregate(pipeline)))
```

```
[{'_id': 'English', 'count': 25325},
 {'_id': 'French', 'count': 1784},
 {'_id': 'Italian', 'count': 1480},
 {'_id': 'Japanese', 'count': 1290},
 {'_id': '', 'count': 1115},
 {'_id': 'Spanish', 'count': 875},
 {'_id': 'Russian', 'count': 777},
 {'_id': 'English, Spanish', 'count': 728},
 {'_id': 'German', 'count': 674},
 {'_id': 'English, French', 'count': 584}]
```

In [13]:

```python
## using filter
filter={'language':'English, Hindi'}
pprint.pprint(list(Movie_col.find(filter)))
```

```
              'War, young Sara Crewe is placed into the care of Amanda '
              'Minchin, the head of an exclusive private school for girls. '
              'Sara lives a wonderful life of a privileged child and is quite '
              'happy in her surroundings. When her father is listed as missing '
              'in action however, her life goes from one of plenty to that of '
              'a poor house maid. Mrs. Minchin agrees to keep her on at the '
              'school, but in the absence of her tuition payments, she has to '
              'work for her keep. She is soon cleaning out the fireplace and '
              'scrubbing floors and is dubbed the little princess by her '
              'former schoolmates. She also refuses to accept that her father '
              'is dead and prowls the hospitals in the hope of locating him. '
              'Luck - and Royal intervention - assist her in her quest.',
 'genre': 'Comedy, Drama, Family',
 'imdbID': 31580,
 'imdbRating': 7.3,
 'imdbVotes': 3928,
 'language': 'English, Hindi',
 'lastupdated': '2015-09-02 00:40:41.367000000',
 'metacritic': '',
 'plot': 'A little girl is left by her father in an exclusive seminary for '
```

In [10]:

```python
pipeline=[
    {
        '$limit':100
    },
    {
        '$project':{
            'title':1,'year':1,'directors':{'$split':["$director",", "]},
            'cast':{'$split':["$cast",", "]},'writers':{'$split':["$writer",", "]},
            'genres':{'$split':["$genre",", "]},'languages':{'$split':["$language",", "]},
            'countries':{'$split':["$country",", "]},
            'plot':1,'fullPlot':"$fullplot",
            'rated':"$rating",
            'released':{'$cond':{'if':{'$ne':["$released",""]},
                    'then':{
                        '$dateFromString':{'dateString':'$released'}},
                    'else':""
                }
            },
            'runtime':1,
            'poster':1,
            'imdb':{
                'id':"$imdbID",
                'rating':'$imdbRating',
                'votes':"$imdbVotes"
            },
            'metacritic':1,
            'awards':1,
            'type':1,
            'lastUpdated':"$lastupdated"
        }
    },
    {
        '$out':'movies_scratch'
    }

]

pprint.pprint(list(Movie_col.aggregate(pipeline)))
```

In [19]:

```python
pprint.pprint(list(Movie_col.find().limit(10)))
```

```
[{'_id': ObjectId('5c4d36add298a5c5dc777e54'),
  'awards': '',
  'cast': ['Carmencita'],
  'countries': ['USA'],
  'directors': ['William K.L. Dickson'],
  'fullPlot': 'Performing on what looks like a small wooden stage, wearing a '
              'dress with a hoop skirt and white high-heeled pumps, Carmencita '
              'does a dance with kicks and twirls, a smile always on her face.',
  'genres': ['Documentary', 'Short'],
  'imdb': {'id': 1, 'rating': 5.9, 'votes': 1032},
  'languages': [''],
  'lastUpdated': '2015-08-26 00:03:45.040000000',
  'metacritic': '',
  'plot': 'Performing on what looks like a small wooden stage, wearing a dress '
          'with a hoop skirt and white high-heeled pumps, Carmencita does a '
          'dance with kicks and twirls, a smile always on her face.',
  'poster': 'http://ia.media-imdb.com/images/M/MV5BMjAzNDEwMzk3OV5BMl5BanBnXkFtZTcwOTk4OTM5Ng@@._V1_SX300.jpg',
  'rated': 'NOT RATED',
  'released': '',
```

```
In [19]:  1  pprint.pprint(list(Movie_col.find().limit(10)))
```

```
  'year': 1894},
 {'_id': ObjectId('5c4d36add298a5c5dc777e55'),
  'awards': '1 win.',
  'cast': ['Charles Kayser', 'John Ott'],
  'countries': ['USA'],
  'directors': ['William K.L. Dickson'],
  'fullPlot': 'A stationary camera looks at a large anvil with a blacksmith '
              'behind it and one on either side. The smith in the middle draws '
              'a heated metal rod from the fire, places it on the anvil, and '
              'all three begin a rhythmic hammering. After several blows, the '
              'metal goes back in the fire. One smith pulls out a bottle of '
              'beer, and they each take a swig. Then, out comes the glowing '
              'metal and the hammering resumes.',
  'genres': ['Short'],
  'imdb': {'id': 5, 'rating': 6.2, 'votes': 1189},
  'languages': [''],
  'lastUpdated': '2015-08-26 00:03:50.133000000',
  'metacritic': '',
  'plot': 'Three men hammer on an anvil and pass a bottle of beer around.',
  'poster': '',
  'rated': 'UNRATED',
```

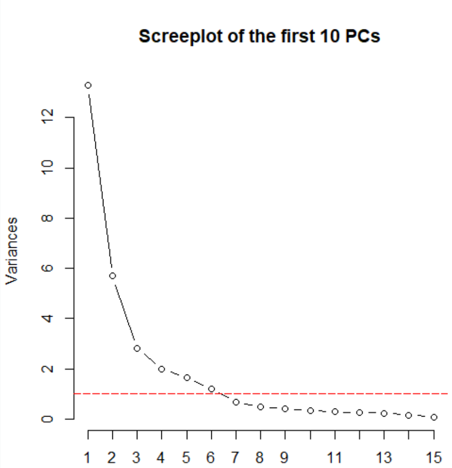
```
_id: ObjectId("5c74cb4723f11a53b1bed401")
title: "Pauvre Pierrot"
year: 1892
runtime: "4 min"
metacritic: ""
poster: ""
plot: "One night, Arlequin come to see his lover Colombine. But then Pierrot ..."
awards: ""
type: "movie"
directors: Array
    0: "�mile Reynaud"
cast: Array
    0: ""
writers: Array
    0: ""
genres: Array
    0: "Animation"
    1: "Comedy"
    2: "Short"
languages: Array
    0: ""

countries: Array
    0: "France"
fullPlot: "One night, Arlequin come to see his lover Colombine. But then Pierrot ..."
rated: ""
released: 1892-10-28 00:00:00.000
imdb: Object
    id: 3
    rating: 6.7
    votes: 566
lastUpdated: "2015-08-12 00:06:02.720000000"
```

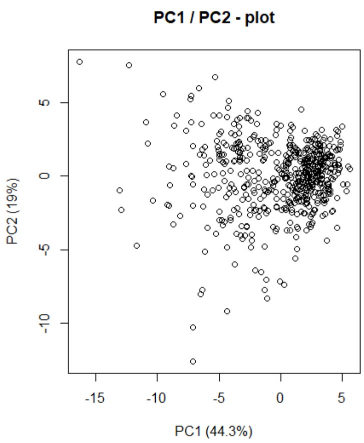
<u>Practical 3</u>

Principal component analysis

```
> wdbc.pr <- prcomp(wdbc[c(3:32)], center = TRUE, scale = TRUE)
> summary(wdbc.pr)
Importance of components:
                          PC1    PC2     PC3     PC4     PC5     PC6     PC7     PC8    PC9    PC10   PC11
   PC12
Standard deviation     3.6444 2.3857 1.67867 1.40735 1.28403 1.09880 0.82172 0.69037 0.6457 0.59219 0.5421
0.51104
Proportion of Variance 0.4427 0.1897 0.09393 0.06602 0.05496 0.04025 0.02251 0.01589 0.0139 0.01169 0.0098
0.00871
Cumulative Proportion  0.4427 0.6324 0.72636 0.79239 0.84734 0.88759 0.91010 0.92598 0.9399 0.95157 0.9614
0.97007
                          PC13    PC14    PC15    PC16    PC17    PC18    PC19    PC20   PC21    PC22    PC
23   PC24
Standard deviation     0.49128 0.39624 0.30681 0.28260 0.24372 0.22939 0.22244 0.17652 0.1731 0.16565 0.156
02 0.1344
Proportion of Variance 0.00805 0.00523 0.00314 0.00266 0.00198 0.00175 0.00165 0.00104 0.0010 0.00091 0.000
81 0.0006
Cumulative Proportion  0.97812 0.98335 0.98649 0.98915 0.99113 0.99288 0.99453 0.99557 0.9966 0.99749 0.998
30 0.9989
```

```
> screeplot(wdbc.pr, type = "l", npcs = 15, main = "Screeplot of the first 10 PCs")
> abline(h = 1, col="red", lty=5)
```



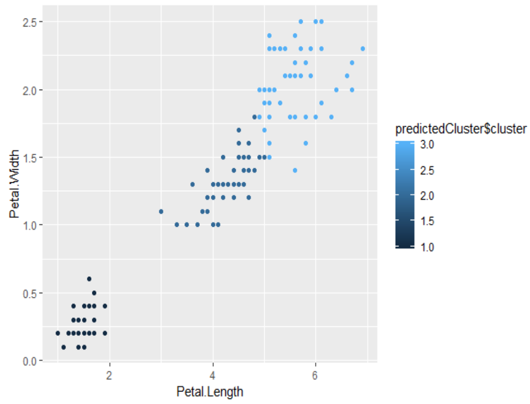



```
> plot(wdbc.pr$x[,1],wdbc.pr$x[,2], xlab="PC1 (44.3%)", ylab = "PC2 (19%)", main = "PC1 / PC2 - plot")
```

Practical 4

K-means clustering

```
> library(datasets)
> library(ggplot2)
> attach(iris)
The following objects are masked from iris (pos = 3):

    Petal.Length, Petal.Width, Sepal.Length, Sepal.Width, Species

The following objects are masked from iris (pos = 4):

    Petal.Length, Petal.Width, Sepal.Length, Sepal.Width, Species

The following objects are masked from iris (pos = 5):

    Petal.Length, Petal.Width, Sepal.Length, Sepal.Width, Species

The following objects are masked from iris (pos = 6):

    Petal.Length, Petal.Width, Sepal.Length, Sepal.Width, Species

> realClusters=ggplot(iris,aes(Petal.Length,Petal.Width,color=Species))+geom_point()
> realClusters
```

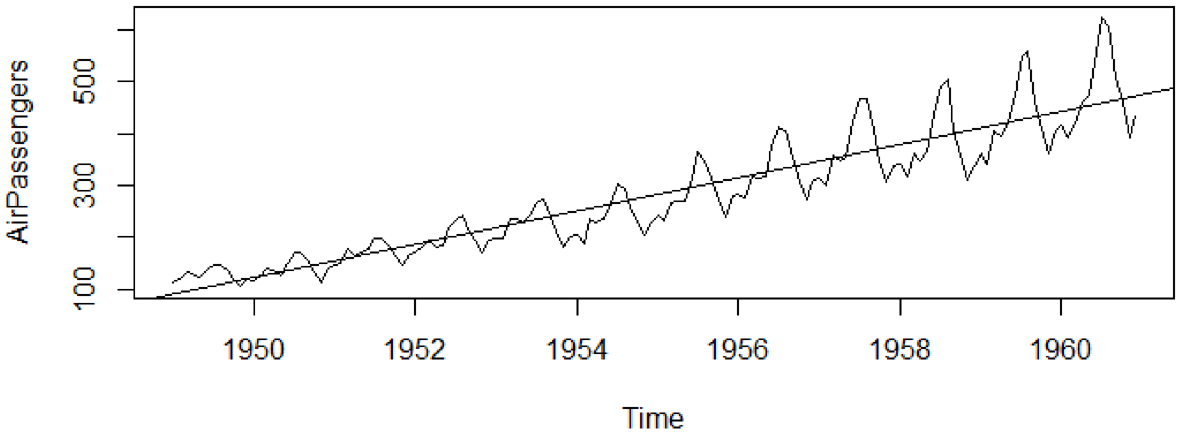



```
> set.seed(20)
> predictedCluster=kmeans(iris[,3:4],centers=3,nstart = 10)
> predictedCluster
K-means clustering with 3 clusters of sizes 50, 52, 48

Cluster means:
  Petal.Length Petal.Width
1     1.462000    0.246000
2     4.269231    1.342308
3     5.595833    2.037500

Clustering vector:
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [35] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [69] 2 2 2 2 2 2 2 3 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3
[103] 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3
[137] 3 3 2 3 3 3 3 3 3 3 3 3 3 3

Within cluster sum of squares by cluster:
[1]  2.02200 13.05769 16.29167
 (between_SS / total_SS =  94.3 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"
[5] "tot.withinss" "betweenss"    "size"         "iter"
[9] "ifault"
> table(predictedCluster$cluster,iris$Species)

    setosa versicolor virginica
  1     50          0         0
  2      0         48         4
  3      0          2        46
> predClusterPlot=ggplot(iris,aes(Petal.Length,Petal.Width,color=predictedCluster$cluster))+geom_point()
> predClusterPlot
```

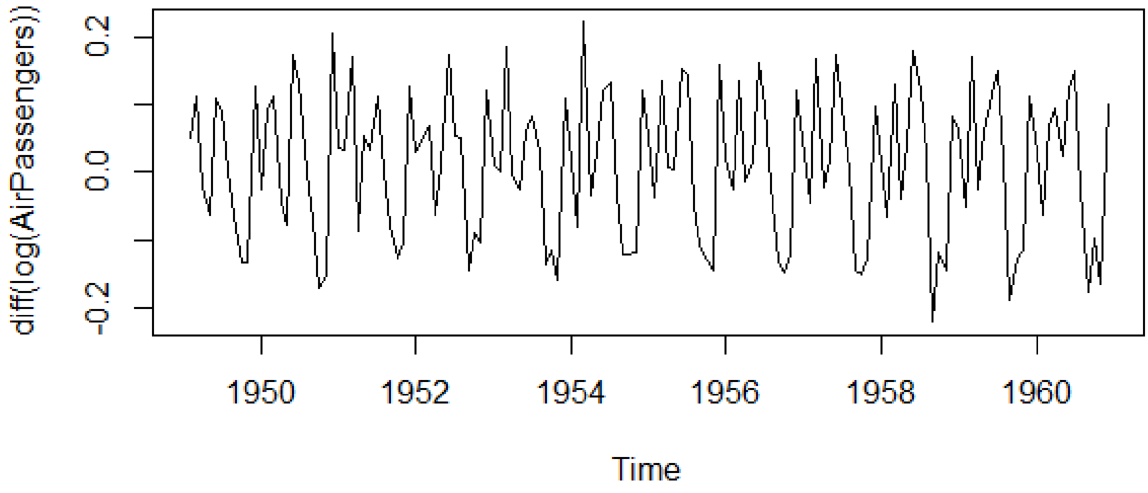
Practical 5

Time series forecasting

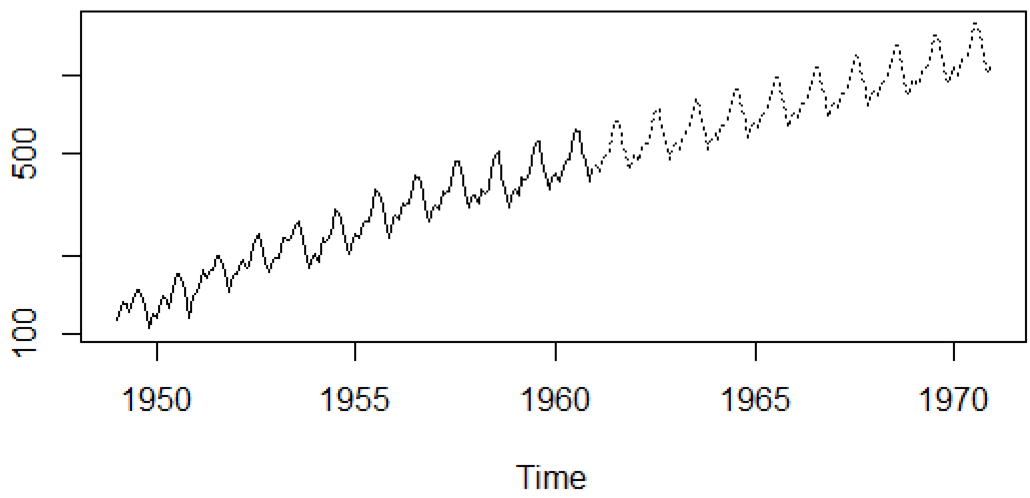
```
> library(TSA)
> library(tseries)
> data(AirPassengers)
> start(AirPassengers)
[1] 1949    1
> end(AirPassengers)
[1] 1960   12
> frequency(AirPassengers)
[1] 12
> summary(AirPassengers)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  104.0   180.0   265.5   280.3   360.5   622.0
> plot(AirPassengers)
> abline(reg=lm(AirPassengers~time(AirPassengers)))
> cycle(AirPassengers)
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1949   1   2   3   4   5   6   7   8   9  10  11  12
1950   1   2   3   4   5   6   7   8   9  10  11  12
1951   1   2   3   4   5   6   7   8   9  10  11  12
1952   1   2   3   4   5   6   7   8   9  10  11  12
1953   1   2   3   4   5   6   7   8   9  10  11  12
1954   1   2   3   4   5   6   7   8   9  10  11  12
1955   1   2   3   4   5   6   7   8   9  10  11  12
1956   1   2   3   4   5   6   7   8   9  10  11  12
1957   1   2   3   4   5   6   7   8   9  10  11  12
1958   1   2   3   4   5   6   7   8   9  10  11  12
1959   1   2   3   4   5   6   7   8   9  10  11  12
1960   1   2   3   4   5   6   7   8   9  10  11  12
```




```
> adf.test(log(AirPassengers), alternative="stationary")

        Augmented Dickey-Fuller Test

data:  log(AirPassengers)
Dickey-Fuller = -6.4215, Lag order = 5, p-value = 0.01
alternative hypothesis: stationary

Warning message:
In adf.test(log(AirPassengers), alternative = "stationary") :
  p-value smaller than printed p-value

> plot(diff(log(AirPassengers)))
> fit = arima(log(AirPassengers), c(0, 1, 1),seasonal = list(order = c(0, 1, 1), period = 12))
```




```
> pred = predict(fit, n.ahead = 10*12)
> ts.plot(AirPassengers,2.718^pred$pred, log = "y", lty = c(1,3))
```



Practical 6

Simple and Multiple linear regression

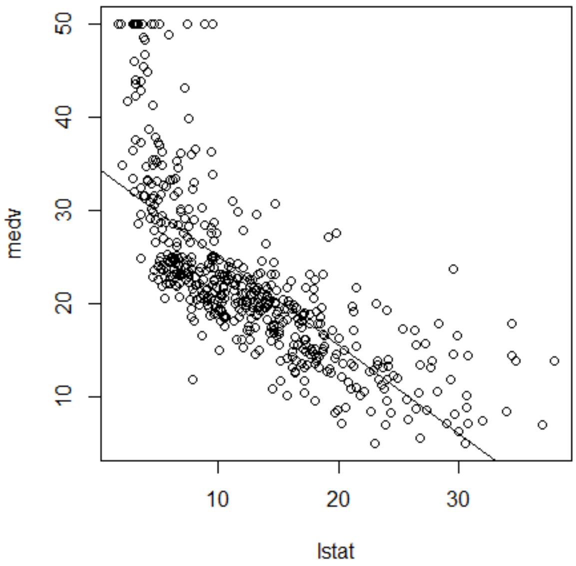
```
> library(MASS)
> attach(Boston)
The following objects are masked from Boston (pos = 3):

    age, black, chas, crim, dis, indus, lstat, medv, nox, ptratio, rad, rm, tax, zn

The following objects are masked from Boston (pos = 4):

    age, black, chas, crim, dis, indus, lstat, medv, nox, ptratio, rad, rm, tax, zn

> str(Boston)
'data.frame':   506 obs. of  14 variables:
 $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
 $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
 $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
 $ chas   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
 $ rm     : num  6.58 6.42 7.18 7 7.15 ...
 $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
 $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
 $ rad    : int  1 2 2 3 3 3 5 5 5 5 ...
 $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
 $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
 $ black  : num  397 397 393 395 397 ...
 $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
 $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
> simple.fit1=lm(medv~lstat)
> simple.fit2=lm(medv~rm)
```

```
> summary(simple.fit1)

Call:
lm(formula = medv ~ lstat)

Residuals:
    Min      1Q  Median      3Q     Max 
-15.168  -3.990  -1.318   2.034  24.500 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 34.55384    0.56263   61.41   <2e-16 ***
lstat       -0.95005    0.03873  -24.53   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 6.216 on 504 degrees of freedom
Multiple R-squared:  0.5441,	Adjusted R-squared:  0.5432 
F-statistic: 601.6 on 1 and 504 DF,  p-value: < 2.2e-16
```

```
> summary(simple.fit2)

Call:
lm(formula = medv ~ rm)

Residuals:
    Min      1Q  Median      3Q     Max 
-23.346  -2.547   0.090   2.986  39.433 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  -34.671      2.650  -13.08   <2e-16 ***
rm             9.102      0.419   21.72   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 6.616 on 504 degrees of freedom
Multiple R-squared:  0.4835,	Adjusted R-squared:  0.4825 
F-statistic: 471.8 on 1 and 504 DF,  p-value: < 2.2e-16
```

```
> #Checking by akaike information criteria
> AIC(simple.fit1)
[1] 3288.975
> AIC(simple.fit2)
[1] 3352.151
> #Checking by bayesian information criteria
> BIC(simple.fit1)
[1] 3301.655
> BIC(simple.fit2)
[1] 3364.831
```

```
> plot(lstat,medv)
> abline(simple.fit1)
```



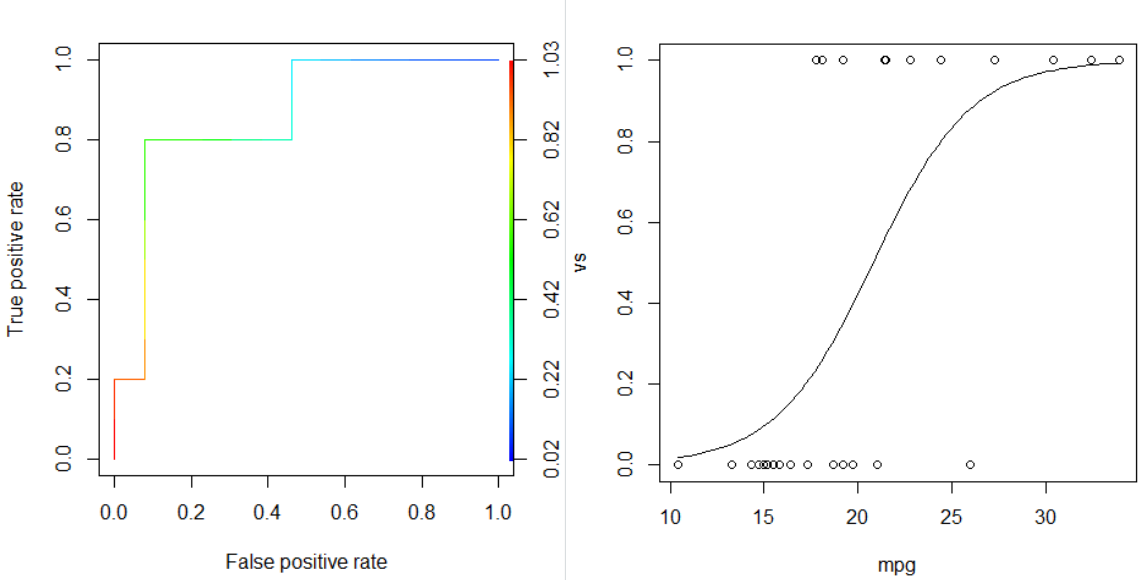

```
> #multiple linear regression
> multiple.fit=lm(medv~lstat+rm)
> summary(multiple.fit)

Call:
lm(formula = medv ~ lstat + rm)

Residuals:
    Min      1Q  Median      3Q     Max 
-18.076  -3.516  -1.010   1.909  28.131 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) -1.35827    3.17283  -0.428    0.669    
lstat       -0.64236    0.04373 -14.689   <2e-16 ***
rm           5.09479    0.44447  11.463   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 5.54 on 503 degrees of freedom
Multiple R-squared:  0.6386,	Adjusted R-squared:  0.6371 
F-statistic: 444.3 on 2 and 503 DF,  p-value: < 2.2e-16
```

Practical 7

Logistic regression

```
> library(caTools)
> attach(mtcars)
The following objects are masked from mtcars (pos = 3):

    am, carb, cyl, disp, drat, gear, hp, mpg, qsec, vs, wt

The following object is masked from package:ggplot2:

    mpg

> #partioning the dataset
> subset_mtcars<- subset(mtcars, select=c(mpg,vs))
> intrain = createDataPartition(y = subset_mtcars$vs, p= 0.7, list = FALSE)
> training = subset_mtcars[intrain,]
> testing = subset_mtcars[-intrain,]
> #logistic regression model
> logisticModel <- glm (vs ~ mpg, data = training, family = binomial)

> summary(logisticModel)

Call:
glm(formula = vs ~ mpg, family = binomial, data = training)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.0623  -0.7672  -0.4256   0.8737   1.6932

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -8.0217     3.5260  -2.275   0.0229 *
mpg           0.3854     0.1727   2.231   0.0257 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 31.492  on 22  degrees of freedom
Residual deviance: 22.697  on 21  degrees of freedom
AIC: 26.697

Number of Fisher Scoring iterations: 5

> predictTrain <- predict(logisticModel, newdata = training, type = 'response')
> predictTest <- predict(logisticModel, newdata = testing, type = 'response')
> #confusion matrix
> table(training$vs,predictTrain>0.5)

    FALSE TRUE
  0    10    3
  1     2    8
> table(testing$vs,predictTest>0.5)

    FALSE TRUE
  0     5    0
  1     1    3
```

```
> library(ROCR)
Loading required package: gplots

Attaching package: 'gplots'

The following object is masked from 'package:stats':

    lowess

> ROCRpred <- prediction(predictTrain, training$vs)
> ROCRperf <- performance(ROCRpred, 'tpr','fpr')
> plot(ROCRperf, colorize = TRUE)
> #plot glm
> plot(mpg,vs)
> curve(predict(logisticModel, data.frame(mpg=x), type="response"), add=TRUE)
```



Practical 8

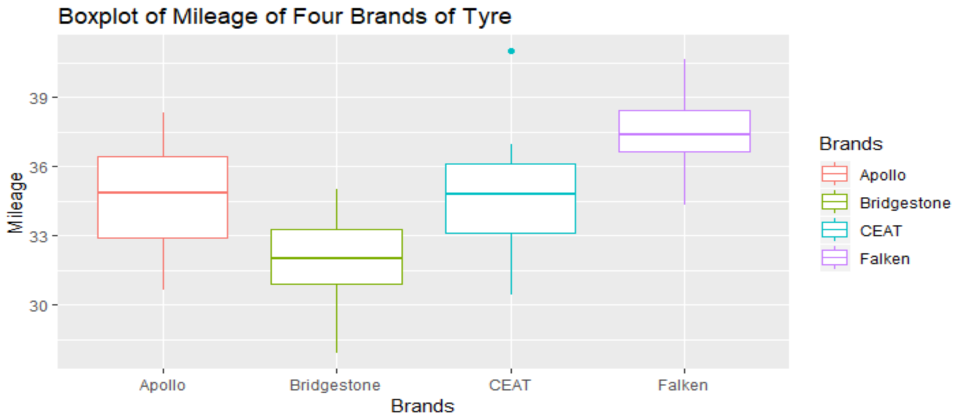
Hypothesis testing

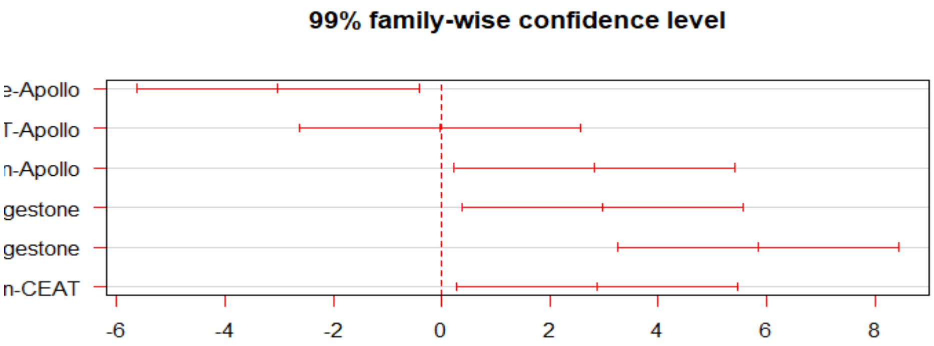
```
> x=c(6.2,6.6,7.1,7.4,7.6,7.9,8,8.3,8.4,8.5,8.6,8.8,9.1,9.2,9.4,9.7,9.9,10.2,10.4,40.8,11
.3,11.9)
> t.test(x,alternative = "two.sided",conf.level = 0.95)

        One Sample t-test

data:  x
t = 6.8881, df = 21, p-value = 8.305e-07
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
  7.149048 13.332770
sample estimates:
mean of x
 10.24091

> t.test(x-10,alternative = "two.sided",conf.level = 0.95)

        One Sample t-test

data:  x - 10
t = 0.16204, df = 21, p-value = 0.8728
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 -2.850952  3.332770
sample estimates:
mean of x
0.2409091
```

Practical 9

Analysis of variance

```
> library(ggplot2)
> attach(tyre)
> ggplot(tyre,aes(Brands,Mileage))+geom_boxplot(aes(col=Brands))+labs(title="Boxplot of Mileage of Four Brands of Tyre")
> boxplot.stats(Mileage[Brands=="CEAT"])
$stats
[1] 30.42748 33.11079 34.78336 36.12533 36.97277

$n
[1] 15

$conf
[1] 33.55356 36.01316

$out
[1] 41.05

> model1<- aov(Mileage~Brands)
> summary(model1)
            Df Sum Sq Mean Sq F value   Pr(>F)    
Brands       3  256.3   85.43   17.94 2.78e-08 ***
Residuals   56  266.6    4.76                     

> TukeyHSD(model1, conf.level = 0.99)
  Tukey multiple comparisons of means
    99% family-wise confidence level

Fit: aov(formula = Mileage ~ Brands)

$Brands
                          diff        lwr        upr     p adj
Bridgestone-Apollo -3.01900000 -5.6155816 -0.4224184 0.0020527
CEAT-Apollo        -0.03792661 -2.6345082  2.5586550 0.9999608
Falken-Apollo       2.82553333  0.2289517  5.4221149 0.0043198
CEAT-Bridgestone    2.98107339  0.3844918  5.5776550 0.0023806
Falken-Bridgestone  5.84453333  3.2479517  8.4411149 0.0000000
Falken-CEAT         2.86345994  0.2668783  5.4600415 0.0037424

> plot(TukeyHSD(model1, conf.level = 0.99),las=1, col = "red")
```

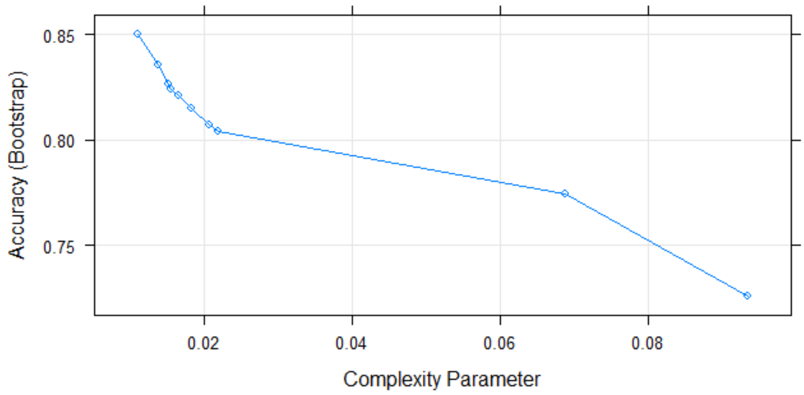




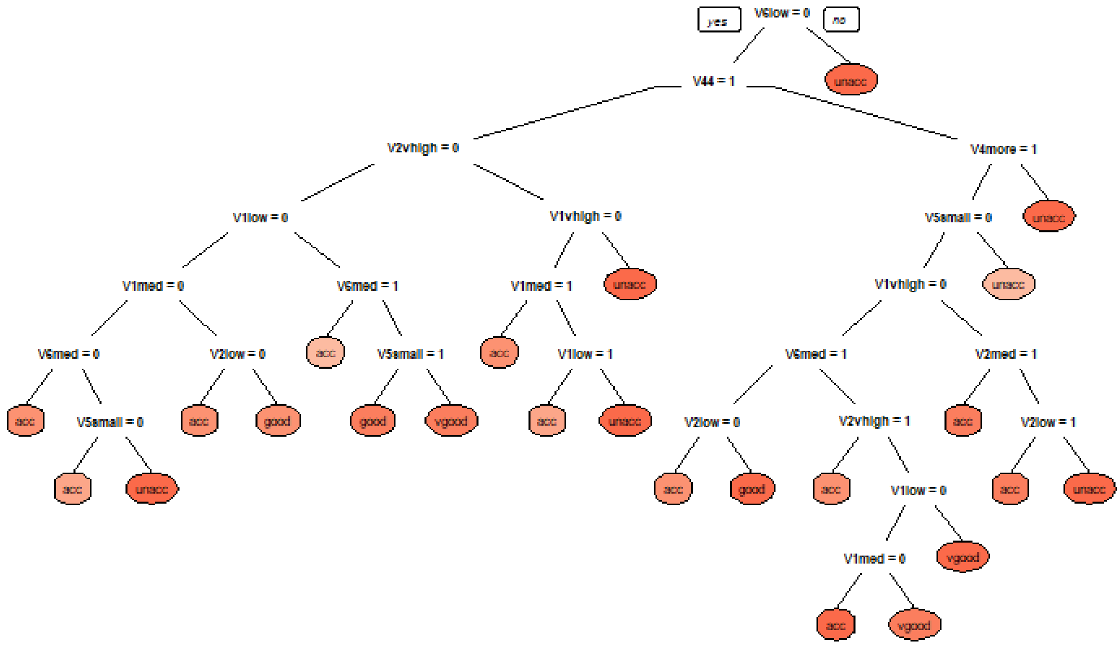
Practical 10

Decision Tree

```
> library(rpart.plot)
> library(e1071)
> attach(car.data)
The following objects are masked from car.data (pos = 3):

    V1, V2, V3, V4, V5, V6, V7

The following objects are masked from car.data (pos = 23):

    V1, V2, V3, V4, V5, V6, V7

> str(car.data)
'data.frame':   1728 obs. of  7 variables:
 $ V1: Factor w/ 4 levels "high","low","med",..: 4 4 4 4 4 4 4 4 4 4 ...
 $ V2: Factor w/ 4 levels "high","low","med",..: 4 4 4 4 4 4 4 4 4 4 ...
 $ V3: Factor w/ 4 levels "2","3","4","5more": 1 1 1 1 1 1 1 1 1 1 ...
 $ V4: Factor w/ 3 levels "2","4","more": 1 1 1 1 1 1 1 1 1 2 ...
 $ V5: Factor w/ 3 levels "big","med","small": 3 3 3 2 2 2 1 1 1 3 ...
 $ V6: Factor w/ 3 levels "high","low","med": 2 3 1 2 3 1 2 3 1 2 ...
 $ V7: Factor w/ 4 levels "acc","good","unacc",..: 3 3 3 3 3 3 3 3 3 3 ...
> set.seed(3033)
> intrain = createDataPartition(y = car.data$V7, p= 0.7, list = FALSE)
> training = car.data[intrain,]
> testing = car.data[-intrain,]
> dim(training)
[1] 1211    7
> dim(testing)
[1] 517   7
> summary(car.data)
    V1          V2          V3         V4         V5          V6         V7
 high :432   high :432   2    :432   2   :576   big  :576   high:576   acc  : 384
 low  :432   low  :432   3    :432   4   :576   med  :576   low :576   good :  69
 med  :432   med  :432   4    :432   more:576   small:576   med :576   unacc:1210
 vhigh:432   vhigh:432   5more:432                                     vgood:  65

> dtree_fit = train(V7 ~., data = training, method = "rpart",parms = list(split = "information"),tuneLength = 10)
> dtree_fit
CART

1211 samples
   6 predictor
   4 classes: 'acc', 'good', 'unacc', 'vgood'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 1211, 1211, 1211, 1211, 1211, 1211, ...
Resampling results across tuning parameters:

  cp          Accuracy   Kappa
  0.01098901  0.8503457  0.6775026
  0.01373626  0.8353616  0.6428245
  0.01510989  0.8264173  0.6213361
  0.01556777  0.8239694  0.6149706
  0.01648352  0.8209174  0.6084876
  0.01831502  0.8151715  0.5968461
  0.02060440  0.8072278  0.5810412
  0.02197802  0.8040991  0.5760616
  0.06868132  0.7744392  0.4976892
  0.09340659  0.7256029  0.2285977

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was cp = 0.01098901.
> plot(dtree_fit)

> prp(dtree_fit$finalModel, box.palette = "Reds")
```



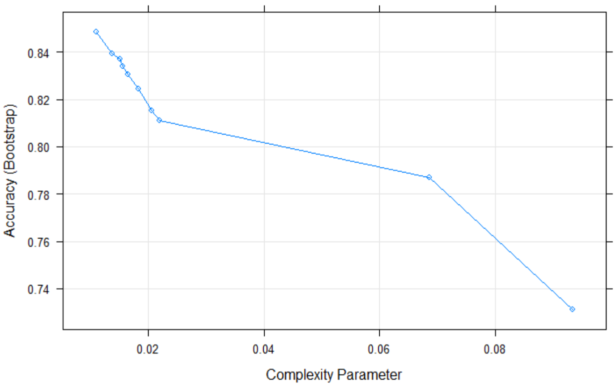
Accuracy (Bootstrap)
0.85
0.80
0.75
0.02
0.04
0.06
0.08
Complexity Parameter

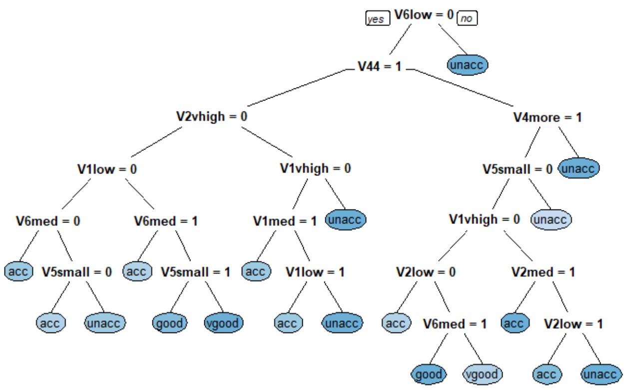


yes
V6low = 0
no
V44 = 1
unacc
V2vhigh = 0
V4more = 1
V1low = 0
V1vhigh = 0
V5small = 0
unacc
V1med = 0
V6med = 1
V1med = 1
unacc
V1vhigh = 0
unacc
V6med = 0
V2low = 0
acc
V5small = 1
acc
V1low = 1
V6med = 1
V2med = 1
acc
V5small = 0
acc
good
good
vgood
acc
unacc
V2low = 0
V2vhigh = 1
acc
V2low = 1
acc
unacc
acc
good
acc
V1low = 0
acc
unacc
V1med = 0
vgood
acc
vgood

```
> test_pred = predict(dtree_fit, newdata = testing)
> confusionMatrix(test_pred, testing$V7 )
Confusion Matrix and Statistics

          Reference
Prediction acc good unacc vgood
     acc    84    8    27     2
     good    7    6     0     1
     unacc  17    5   336     0
     vgood   7    1     0    16

Overall Statistics

               Accuracy : 0.8549
                 95% CI : (0.8216, 0.8842)
    No Information Rate : 0.7021
    P-Value [Acc > NIR] : 3.563e-16

                  Kappa : 0.6839
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: acc Class: good Class: unacc Class: vgood
Sensitivity              0.7304     0.30000       0.9256      0.84211
Specificity              0.9080     0.98390       0.8571      0.98394
Pos Pred Value           0.6942     0.42857       0.9385      0.66667
Neg Pred Value           0.9217     0.97217       0.8302      0.99391
Prevalence               0.2224     0.03868       0.7021      0.03675
Detection Rate           0.1625     0.01161       0.6499      0.03095
Detection Prevalence     0.2340     0.02708       0.6925      0.04642
Balanced Accuracy        0.8192     0.64195       0.8914      0.91302
```

```
> dtree_fit_gini = train(V7 ~., data = training, method = "rpart",parms = list(split = "gini"),tuneLength = 10)
> prp(dtree_fit_gini$finalModel, box.palette = "Blues")
> dtree_fit_gini = train(V7 ~., data = training, method = "rpart",parms = list(split = "gini"),tuneLength = 10)
> plot(dtree_fit_gini)
> prp(dtree_fit_gini$finalModel, box.palette = "Blues")
```





```
> test_pred_gini<-predict(dtree_fit_gini, newdata = testing)
> confusionMatrix(test_pred_gini, testing$V7 )
Confusion Matrix and Statistics

          Reference
Prediction acc good unacc vgood
     acc    87   10    25     8
     good    4    4     0     0
     unacc  22    5   338     0
     vgood   2    1     0    11

Overall Statistics

               Accuracy : 0.8511
                 95% CI : (0.8174, 0.8806)
    No Information Rate : 0.7021
    P-Value [Acc > NIR] : 2.18e-15

                  Kappa : 0.6666
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: acc Class: good Class: unacc Class: vgood
Sensitivity              0.7565    0.200000       0.9311      0.57895
Specificity              0.8930    0.991952       0.8247      0.99398
Pos Pred Value           0.6692    0.500000       0.9260      0.78571
Neg Pred Value           0.9276    0.968566       0.8355      0.98410
Prevalence               0.2224    0.038685       0.7021      0.03675
Detection Rate           0.1683    0.007737       0.6538      0.02128
Detection Prevalence     0.2515    0.015474       0.7060      0.02708
Balanced Accuracy        0.8248    0.595976       0.8779      0.78646
```