Lecture 1

INTRODUCTION

What is data science?

- “statistics on a mac”
- Just a buzzword for business?
- Compare with computer science: what is it, and is it important to business? To science and research?
- A hybrid discipline between statistics and computer science
- Includes also the art of communicating the findings to the decision makers and/or clients etc.

Why are we talking about it now?

- ”The internet” is creating and collecting vast amounts of data
- Internet of Things (IoT)
- Open data

What's in it for research?

- Research imagination is the only limit
- However, data analysis in research is more about explanation and confirmation while data science is more about solving problems and making predictions
- Randomized controlled trials are not getting obsolete
- On the other hand, maybe more chances for natural experiments?

Big Data

- Technically, data is Big when you need some special techniques to handle and analyze it, because of memory and other resource limitations
  - Hadoop, Spark
- The word is also used just to refer to the general idea or power of data intensive analysis, in the same way ‘Big Pharma’, ‘Big Money’ or ‘Big Government’ are used, including the sinister associations

Data Wrangling

- a.k.a data munging, managing data, cleaning data…
- A mundane, maybe even boring, part of data analysis that can’t be avoided and which takes a lot of time and effort in any data analysis project
- Also isn’t easily diverted to more technical people: it requires understanding both the technical and analytical aspects of the data at hand

Why use both R and Python in this course

- Both are used and are good for data science
- The have different backgrounds and different communities and neither is likely to take over
- Both have their upsides and downsides
- They are similar enough so that knowing one is a good starting point for learning the other
- We hope to show concepts, not cookbook recipes

Python

- Python is a modern, high-level, object oriented, general-purpose programming language
- Interpreted, portable (Unix/Linux, Windows, Mac OS X)
- Open source
- Emphasizes the readability of the code
- Huge standard library shipped with the Python installation by default
  - Humongous collection of external libraries can be installed with tools like pip from Python Packaging Index (PyPI)

Python libraries for data processing

- Numpy *numpy.org*
  - Adds support for static, multidimensional arrays and matrices, similar to MATLAB
- Pandas *pandas.pydata.org*
  - Adds support for DataFrame objects, similar to R
- SciPy
  - Collection of high-level scientific tools: integrals, optimization, machine-learning, ...
- Matplotlib *matplotlib.org*
  - Plotting library

Installing Python stack

- Windows
  - Continuum provides a free Python distribution called Anaconda with necessary libraries preinstalled
  - http://continuum.io/downloads
- Linux
  - sudo apt-get install python-numpy python-scipy python-matplotlib python-pandas
- Mac OS X
  - Install Anaconda or use Homebrew/Macports

R

- R is a ”language and environment” for data manipulation, calculation and graphical display
- Interpreted, portable (Unix/Linux, Windows, Mac OS X)
- Open source
- Vehicle for newly developing methods of interactive data analysis
- Large collection of external packages
- RStudio is a popular GUI for R

Reminder: reading data

```
#for local files with a header and whitespace
    separated columns:
read.table("filename.txt",header=TRUE)

#for csv files at a given url:
read.csv("http://www.domain.com/filename.csv"
    )

#for csv files with , as the decimal
    separator and ; as a field separator
read.csv2("filename.csv")
```

```
import pandas as pd
# for local files with a header and tab
# separated columns, use kwarg sep with
# reqexp like "\W+" to match whitespaces
pd.read_table("filename.txt")

# for csv files at a given url
pd.read_csv(
    "http://example.org/filename.csv"
)

# for csv files with , as the decimal
# separator and ; as field separator
pd.read_csv(
    "filename.csv",
    sep=";",
    decimal=","
)
```

Lecture 2

DATA ACQUISITION

CSV files

- CSV is great for clean, tabular data
- Easy to read, supported data types: numeral and text
- Problems:
  - Coding issues, multiple conventions to encode the data
    - Quotation: ”””””, line-breaks?
    - Localization: decimal numbers use decimal comma or point?
    - Separator: ”,” ”;” ” ”
  - Does not support nested structures

JSON

- Human readable & machine parseable data interchange format
- Based on JavaScript, used widely on the web
- Value types: null, number, string, true, false, object, list
- Value types object and list can contain multiple values

```
{
    "title": "Wuthering Heights",
    "author": "Emily Bronte"
}
```

- Object in JSON is
  - A collection of key-value pairs
  - Each pair is separated from another pair with comma
  - Key has a type string, value can be any JSON value type
- e.g. representing the metadata of a single book in JSON

```
[
    {
        "title": "Wuthering Heights",
        "author": "Emily Bronte"
    },
    {
        "title": "Anna Karenina",
        "author": "Leo Tolstoy"
    }
]
```

- A list in JSON is
  - An ordered list of values
- e.g. representing the metadata of multiple books in JSON

```
library(jsonlite)

#straight to data frame (if possible)
books<-fromJSON("file.json")
is.data.frame(books)
books

#now a "raw" list
books<-fromJSON("file.json",simplifyVector =
    FALSE)

for(i in 1:length(books))
  cat(books[[i]]$title,"\n")
```

```
import json

with open("file.json", "w") as fd:
    json.dump(books, fd)

bks = json.load(open("file.json"))
for book in bks:
    print "Book record:", book

print "Titles", [b['title'] for b in bks]
```

Data dumps

- Data provided as a file or as a collection of files
  - csv, txt, spreadsheet, HDF5
- Data updated seldom
- Size varies from kilobytes to terabytes
  - Processing large datasets starts to require even more sophisticated methods and tools

Web APIs

- Multiple web services offer open APIs to query parts of the data used by the service
  - e.g. Twitter, Google, OpenAhjo (dev.hel.fi)
- Queries are HTTP requests
  - Might require authentication

- In Python:
  - For HTTP requests use either of the following libraries
    - urllib2 (in standard library)
    - requests (not in standard library, simpler interface)
- In R:
  - See help page for ‘connections’
  - Package RCurl

```
library(RCurl)
API_BASE_URL<-"http://example.org/api/"
response<-getURL(apiurl)
response<-fromJSON(response)
min_round<-response[["/api/round/<round>"]]$
    param_min
max_round<-response[["/api/round/<round>"]]$
    param_max
```

```
import requests
API_BASE_URL = "http://example.org/api/"
response = requests.get(API_BASE_URL)
print "Status code:", response.status_code
print "Content:", response.text
response = response.json()
min_round =
  response['/api/round/<round>']['param_min']
max_round =
  response['/api/round/<round>']['param_max']
# Rest in the Exercise 2…
```

Scraping

- Extract information from structured documents in Web
- Web documents tend to be HTML files. HTML files might be valid XML files, or they might not.
- Python has a library BeautifulSoup which can usually read even quite broken XML files
- Scraping should be the last resort
  - Takes a lot of time and causes unnecessary traffic
  - Service authors might not be happy, ask permission
  - If possible, try caching the downloaded data
    - If processing fails, fix & continue from where we left

Lecture 3

DATA WRANGLING, PT. 1

- In R: the function `summary` used on a data frame will give a summary of all the column variables
  - A quick way to spot blatant problems
- In Python: `describe` method shows only numeric fields by default, use `kwarg include='all'` to show all columns

```
fakedf<-data.frame(stringsAsFactors = FALSE,
    letterf=as.factor(sample(c(letters,NA),30
    0,replace=TRUE)),
    letterc=sample(c(letters,NA),300,replace=
    TRUE),
    number=sample(c(1:10,NA),300,replace=TRUE
    ), bool=sample(c(T,F),300,replace=TRUE))

summary(fakedf)

data(iris)
write.csv(iris,"iris.csv")
summary(read.csv("iris.csv"))
summary(read.csv2("iris.csv"))
```

```
letters =
    np.concatenate((com.load_data('letters'),
    [None]))
lets = pd.Series(letters)
letc = pd.Series(letters, dtype='category')
nums = pd.Series(
  np.concatenate((range(1, 11), [np.nan])))
booleans = pd.Series([True, False])
fakedf = pd.DataFrame({
  'letterC': np.random.choice(lettersC, 300),
  'letterS': np.random.choice(lettersS, 300),
  'number': np.random.choice(numbers, 300),
  'bool': np.random.choice(booleans, 300)})
fakedf.describe()
iris = com.load_data('iris')
iris.to_csv('iris.csv')
pd.read_csv('iris.csv').describe()
pd.read_csv('iris.csv', sep=";",
    decimal=",").describe()
```

- In the following example some values in the data file that should be numbers are letters
  - In R: the column ends up being a factor instead of numeric
  - In Python: not that much of a problem

```r
faketrees<-read.csv("faketrees.csv")
summary(faketrees)
is.factor(faketrees$height)
class(faketrees$height)
summary(faketrees$height)
levels(faketrees$height)
```

```python
faketrees = pd.read_csv('faketrees.csv')
faketrees.describe(include='all')
faketrees.dtypes
faketrees['height'].describe()
```

- Ways to deal with the problem:
  - In R/Python:
    - declare more missing value strings
    - Read in as character, fix manually

```
faketrees<-
    read.csv("faketrees.csv",na.strings=c("-
    9999","E"))


faketrees<-
    read.csv("faketrees.csv",as.is=TRUE)
faketrees$heightn<-
    as.numeric(faketrees$height)
#compare:
as.numeric(c("1","345.9","foo","8"))
faketrees$heightn<-
    ifelse(faketrees$heightn>0,faketrees$heig
    htn,NA)
```

```
faketrees = pd.read_csv(
  "faketrees.csv",
  na_values=["-9999", "E"])

# integer columns with NaN values are
# promoted to floats, booleans to objects

# another approach to deal with NaN values
# with Pandas is to use masked arrays.
```

- The other way around:
  - In R: simply using the function `factor`
    - Extra levels become missing values
  - In Python: Pandas has categorical data type

```
faketrees$speciesf<-
    factor(faketrees$species,labels=c("pine",
    "spruce"))
table(faketrees$species)
levels(as.factor(faketrees$species))
faketrees$speciesf<-
    factor(faketrees$species,levels=1:2,label
    s=c("pine","spruce"))
summary(faketrees)

is.na(fakedf$number)
sum(complete.cases(fakedf))
```

```
faketrees['speciesf'] =
  pd.Categorical(faketrees['species'],
  categories=[1, 2]).
    rename_categories(['pine', 'spruce'])
faketrees.describe(include='all')

np.sum(pd.isnull(fakedf['number']))
```

- Outliers will show up nicely in a histogram or a boxplot
- Barplots are nice for frequencies of categorical data
- More quick and dirty ways to explore data visually:
  - Scatterplots
  - Boxplots by category

```
hist(faketrees$heightn)

boxplot(faketrees$heightn)

barplot(table(faketrees$species))

barplot(table(faketrees$speciesf))

plot(faketrees$speciesf)

plot(heightn~diam,data=faketrees)

boxplot(heightn~speciesf,data=faketrees)
```

```
faketrees['height'].hist()

faketrees['height'].plot(kind='box')

pd.value_counts(
  faketrees['species'],
  normalize=True
).plot(kind='bar')

pd.value_counts(np.asarray(
  faketrees['speciesf']),normalize=True
).plot(kind='bar')

faketrees.plot(kind='scatter', x='diam',
    y='height')

faketrees.boxplot(column=['height'],
    by=['speciesf'])
```

Lecture 4

DATA WRANGLING, PT. 2

- Adding or replacing columns in data frames
  - In R: direct assignment, row order must match
  - In Python: direct assignment
- Dropping columns
  - In R: assigning NULL, or selecting all others using names or indices (positive or negative)
  - In Python: DataFrame.drop or del statement

```r
data(iris)
names(iris)
iris$Sepal.Area<-
    with(iris,Sepal.Length*Sepal.Width)
names(iris)

iris$Sepal.Area<-NULL
names(iris)

#all of the following do the same in this
    case
iris<-iris[1:5] #pick just the first 5
iris<-iris[-6] #pick everything but the 6th
iris<-iris[c("Sepal.Length","Sepal.Width",
    "Petal.Length","Petal.Width","Species")]
iris<-iris[-match(names(iris),"Sepal.Area")]
    #pick everything that isn't called...
```

```python
iris = com.load_data('iris')
iris.columns
iris['Sepal.Area'] =
  iris['Sepal.Length'] * iris['Sepal.Width']
iris.columns

del iris['Sepal.Area']
iris.columns

iris.iloc[:, :5] #pick just the first 5
iris.iloc[:, :-1] #... but the 6th
iris.head().loc[:,
 ["Sepal.Length", "Sepal.Width",
  "Petal.Length","Petal.Width","Species"]]

# inverse matching not that easy, see
    examples
```

- Binning, or breaking a continuous variable in to a discrete one
  - In R: using the function `cut`
    - Breaks can be counted using e.g. `seq` or `quantile`
  - In Python: `cut, qcut`

```r
hist(iris$Sepal.Length)
range(iris$Sepal.Length)
iris$SLgroup<-cut(iris$Sepal.Length,
    breaks=c(0,4,5,7,8),
    labels=c("xs","small","medium","large"))

intr<-with(iris, seq(min(Sepal.Length),
    max(Sepal.Length),length.out=11))
iris$SLgroup<-cut(iris$Sepal.Length,
    breaks=intr,include.lowest=TRUE)

dec<-quantile(iris$Sepal.Length,
    probs=(0:10)/10)
iris$SLgroup<-cut(iris$Sepal.Length,
    breaks=dec,labels=FALSE,
    include.lowest=TRUE)
```

```python
iris['Sepal.Length'].hist() ; plt.show()
min(iris['Sepal.Length'])
iris['SepalLengthGroup'] = pd.cut(
  iris['Sepal.Length'],
  [0, 4, 5, 7, 8],
  labels=["xs", "small", "medium", "large"]
)

iris['SepalLengthGroup'] = pd.cut(
  iris['Sepal.Length'],
  10,
  include_lowest=True
)

dec = pd.Series(range(0, 11)) / 10
iris['PedalSLLengthGroup'] =
  pd.qcut(iris['Sepal.Length'], q=dec)
```

- Regrouping, or combining existing classes in to one
  - Requires naming the “old class – new class” pairs one way or another
  - In R: various more or less cumbersome ways
  - In Python: also various ways to do this, probably a good use case for lambda functions

```r
zoo<-data.frame(animalID=1:50,
    spec=factor(sample(c("cat", "dog",
    "flatworm", "kiwi", "sparrow",
    "fruitfly", "swordfish", "pike",
    "crocodile"),50,replace=TRUE)),
    awesomeness=rnorm(50))
head(zoo)
zoo$animgroup<-with(zoo,
    ifelse(spec%in%c("cat","dog"),"mammal",
    levels(spec)[spec]))
head(zoo)
zoo$animgroup<-factor(zoo$animgroup)
(alvls<-levels(zoo$spec))
agroup<-c("mammal","reptile","mammal",
    "insect","insect","bird","fish","bird",
    "fish")
names(agroup)<-alvls
agroup
zoo$animgroup<-agroup[zoo$spec]
```

```python
animals = pd.Series(("cat", "dog",
    "flatworm", "kiwi", "sparrow",
    "fruitfly", "swordfish", "pike",
    "crocodile"), dtype='category')
zoo = pd.DataFrame({
    'animalID': range(50),
    'spec': np.random.choice(animals, 50),
    'awesomeness': np.random.randn(50)
})
zoo.head()
mammals = {'cat': 'mammal', 'dog': 'mammal'}
zoo['class'] = zoo['spec'].apply(lambda x:
    mammals.get(x, x))
zoo.head()
zoo.describe()
```

- Working with strings
  - In R: using e.g. functions `strsplit`, `substr`, `paste`
  - In Python: `Series` has special `.str attribute`
- Dates and times are common cases of strings and have special functions for them
  - In R: date-time class, function `strptime`
  - In Python: `to_datetime`, `DatetimeIndex`, automatic date time parsing

```
weather<-read.csv("weather-kumpula.csv")
class(weather$ts)
weather$ts<-as.character(weather$ts)
ts.split<-strsplit(weather$ts,split="-")
class(ts.split)
ts.split[[1]]
ts.split<-simplify2array(ts.split)

weather$year<-ts.split[1,]
weather$month<-ts.split[2,]
weaher$day<-ts.split[3,]

weather$year<-
    substr(weather$ts,start=1,stop=4)
weather$month<-
    substr(weather$ts,start=6,stop=7)
weather$day<-
    substr(weather$ts,start=9,stop=10)
```

```
weather = pd.read_csv("weather-kumpula.csv")

weather['ts'].dtype

weather['tssplit'] =
    weather['ts'].str.split("-")

weather['year'] = weather['tssplit'].str[0]
weather['month'] = weather['tssplit'].str[1]
weather['day'] = weather['tssplit'].str[2]

weather['year'] = weather['ts'].str[:4]
weather['month'] = weather['ts'].str[5:7]
weather['day'] = weather['ts'].str[8:10]
```

```
dates<-weather[c("year","month","day")]

for(n in names(dates)) dates[[n]]<-
    as.numeric(dates[[n]])
with(dates,head(paste(year, month, day,
    sep="-")))

weather$datetime<-
    strptime(weather$ts,format="%F")

weather$date<-as.Date(weather$datetime)

head(as.character(weather$datetime))
```

```
weather['date'] = pd.to_datetime(
  weather['year'] + '/' + weather['month'] +
    '/' + weather['day']
)
weather = pd.read_csv(
  'datasets/weather-kumpula.csv',
  parse_dates=True,
  index_col=0
)

weather[weather > "2014-12-01"]

weather[pd.datetime(2014, 6,
    25):pd.datetime(2014, 6, 30)]
```

Lecture 5

DATA WRANGLING, PT. 3

- Subsetting, or selecting only some rows
  - In R: either using the function `subset` or [,]
  - In Python: indexing or with method query
- Combining rows or columns
  - In R: using cbind (row order must match) or rbind
  - In Python: concat([df1, df2], axis=N)
- Combining whole data frames
  - In R/Python: using the function `merge`

```
nk<-read.csv("NordklimData.csv")
nks<-read.csv("NordklimStationCatalogue.csv")

nksub<-subset(nk,CountryCode=="FIN")
nksub<-subset(nk,
    CountryCode=="FIN"&ClimateElement%in%c(10
    1,111:113,121:123))
nksub<-subset(nk,
    FirstYear==1900,select=May:August)
```

```
nk =
 pd.read_csv("NordklimData.csv", index_col=0)
nks =
 pd.read_csv("NordklimStationCatalogue.csv")
nksub = nk[nk['CountryCode'] == 'FIN']
nksub = nk.query('CountryCode == "FIN"')
nksub = nk[nk.isin({'CountryCode': ['FIN']}
    ).any(1)]

climate_elements =
  (101, 111, 112, 113, 121, 122, 123)
nksub = nk[(nk['CountryCode'] == 'FIN') &
    nk['ClimateElement'].isin(climate_element
    s)]
nksub = nk.query('CountryCode == "FIN" and
    ClimateElement in
    (101,111,112,113,121,122,123)')
nksub = nk[(nk['FirstYear'] == 1900)]
    .loc[:, 'May':'August']
```

```
nksnames<-subset(nks,
    select=c(Nordklim.number,Station.name))


nknamed<-merge(nk,nksnames,
    by.x="NordklimNumber",
    by.y="Nordklim.number")


nknamed<-merge(nk,nksnames,
    by.x="NordklimNumber",
    by.y="Nordklim.number",all=TRUE)
```

```
nksnames = nks.loc[:, ('Nordklim.number',
    'Station.name')]


nknamed = pd.merge(nk, nksnames,
    left_on='NordklimNumber',
    right_on='Nordklim.number')


nknamed = pd.merge(nk, nksnames,
    left_on='NordklimNumber',
    right_on='Nordklim.number', how='left')
```

- Long (or stacked) and wide format: in long format each row is an observation and each column is a variable. This is usually what you want.
- To switch between the two:
  - In R: the function `reshape`
  - In Python: DataFrame methods `melt` and `pivot`

```
nk.rs<-reshape(nk,
    varying=match("January",names(nk)):match(
    "December",names(nk)), v.names="value",
    timevar="month",times=1:12,
    direction="long")


nk.rs<-subset(nk.rs,select=-c(id,X))
nk.rs<-subset(nk.rs,CountryCode=="FIN") #To
    make the next call a bit faster


nk.rs2<-reshape(nk.rs, v.names="value",
    timevar="ClimateElement",
    idvar=c("NordklimNumber","FirstYear","mon
    th"), direction="wide")
```

```
id_vars = ['NordklimNumber', 'CountryCode',
    'ClimateElement', 'FirstYear']
nk_rs = pd.melt(nk, id_vars=id_vars,
    var_name="Month", value_name="value")


months = {
    "January": 1,
    "February": 2,
    …
}
nk_rs.replace(to_replace={"Month": months},
    inplace=True)

nk_rs2 = nk_rs.pivot_table(
    values="value",
    index=["NordklimNumber", "Month"],
    columns=["ClimateElement"],
    aggfunc=np.mean)
```

- Aggregation: calculate a groupwise output variable, such as group mean, sum etc.
  - In R: function `aggregate` (see also package `plyr`)
    - Other groupwise operations may have to be looped
  - In Python:
    - groupby to produce GroupBy object, GroupBy.aggregate
    - pivot_table can take aggfunc parameter for aggregation
    - Both take aggregate function like np.sum, np.mean

```r
tmp<-aggregate(tempmean~month+NordklimNumber,
    data=nk.rs2,FUN=mean)
head(tmp)
plot(tempmean~month,data=tmp)


for(NN in unique(tmp$NordklimNumber))
    lines(tempmean~month,data=tmp,
    subset=NordklimNumber==NN)
```

```python
for NN in nk["NordklimNumber"].unique():
    nk_rs.ix[
        (NN, ),
    :]["tempmean"].plot(kind='line')
```

Lecture 6

VISUALIZATION AND ANALYSIS

- Quick overview of typical graph types:
  - Simple frequencies by group: barplot
  - Cross tabulations by two groupings: stacked or grouped barplots
  - Continuous variable by group: boxplots or density function estimates
  - Continuous variable by time: line graphs
  - Continuous by continuous: scatter plot

```
tmp<-data.frame(group=
    factor(sample(LETTERS[1:5],100,
    replace=TRUE,prob = c(1,1,2,3,4))))
barplot(xtabs(~group,tmp))
tmp$anothergroup<-
    factor(sample(letters[23:26],100,
    replace=TRUE))
barplot(xtabs(~anothergroup+group,tmp),
    beside=TRUE,legend=TRUE)
barplot(prop.table(xtabs(~anothergroup+group,
    tmp),2),legend=TRUE)
```

```
import pandas.rpy.common as com
LETTERS = com.load_data('LETTERS')
letters = com.load_data('letters')
xk = np.arange(5)
pk = (0.1, 0.2, 0.25, 0.4, 0.05)
idxs = scipy.stats.rv_discrete(values=(xk,
    pk)).rvs(size=100)
tmp = pd.DataFrame()
tmp['group'] = LETTERS[idxs]
pd.value_counts(tmp['group'],
    normalize=True).plot(kind='bar')

tmp['anothergroup'] =
    letters[np.random.randint(23, 26, 100)]
pd.crosstab(tmp['group'],
    tmp['anothergroup']).plot(kind='bar')
```

```r
tmp$xvar<-with(tmp,
    rnorm(100,sd=(1:4)[anothergroup]))
boxplot(xvar~group,tmp)
boxplot(xvar~group+anothergroup,tmp)

plot(density(subset(tmp,
    anothergroup=="w")$xvar))
lines(density(subset(tmp,
    anothergroup=="x")$xvar),col="red")
lines(density(subset(tmp,
    anothergroup=="y")$xvar),col="blue")
```

```python
tmp['xvar'] = tmp['idxs'] *
    np.random.randn(100)
tmp[['group',
    'xvar']].groupby('group').boxplot()
```

```
tmp2<-data.frame(y=runif(15),t=1:15)
plot(y~t,tmp2,type="l") #fine
tmp2<-data.frame(y=runif(15),t=sample(15))
plot(y~t,tmp2,type="l") #NOT fine!

tmp$yvar<-1+2*tmp$xvar+rnorm(100,sd=2)
plot(yvar~xvar,tmp)
plot(yvar~xvar,tmp,col=group,
    pch=(1:4)[anothergroup])
plot(yvar~xvar,tmp,type="n")
points(yvar~xvar,tmp,subset=group=="A",
    col="black",pch=16)
points(yvar~xvar,tmp,subset=group=="B",
    col="blue",pch=17,cex=3)
```

- Package ggplot2 is an alternative to the simple R plots
- The syntax differs from the R plotting syntax (based on the idea of “grammar of graphics”
- In Python…?

- Package ggplot2 is an alternative to the simple R plots
- The syntax differs from the R plotting syntax (based on the idea of “grammar of graphics”)
- In Python there is an open source library ggplot
  - Pretty new, might contain some bugs

```r
mt<-read.csv("monthtemps.csv")
mt$NNf<-factor(mt$NordklimNumber)

ggplot(mt)+
    geom_point(aes(x=month,y=tempmean))

ggplot(mt)+
    geom_point(aes(x=month,y=tempmean,
    col=NNf))

ggplot(mt)+geom_line(aes(x=month,y=tempmean,
    col=NNf))

ggplot(mt)+
    geom_line(aes(x=month,y=tempmean))+
    facet_wrap(~NNf)
```

```python
mt = pd.read_csv("monthtemps.csv")

mt['NNf'] = pd.Series(mt['NordklimNumber'],
    dtype="category")

ggplot(mt, aes('month', 'tempmean')) +
    geom_point(colour='steelblue')

ggplot(mt, aes('month', 'tempmean')) +
    geom_point(colour='steelblue') +
    facet_wrap("NNf")
```

- Basic idea of fitting a linear regression model:
  1. You name the response and the explaining variable(s)
  2. You get the intercept and other coefficients
- The result is somewhat different depending on whether the explaining variables are continuous or discrete

```
fit<-lm(yvar~xvar,tmp)
summary(fit)
plot(yvar~xvar,tmp)
abline(fit)

fit<-lm(yvar~group,tmp)
summary(fit)
anova(fit)

summary(lm(yvar~group+xvar,tmp))

summary(lm(yvar~group*xvar,tmp))
#equal to:
summary(lm(yvar~group+xvar+group:xvar,tmp))

summary(lm(yvar~xvar+group:xvar,tmp))
```

```
x = np.linspace(0,10,50)
y = 2.5 * x + 1.2

noise = np.random.randn(y.size)
noisy = y + noise

p = np.polyfit(x,noisy,1)

plt.plot(x, noisy, 'b.')
plt.plot(x, p[0] * x + p[1],'r-')
```

Lecture 7

RESEARCH RELATED TOPICS

Things to think about in data science

- Types of missing data: missing completely at random (MCAR), missing at random (MAR), missing not at random (MNAR)
- One more category of missing: when you don’t even know it should be there
  - Selection bias / survivorship bias
- Researcher degrees of freedom
  - Keep notes!

Statistical significance

- Which of these claims are correct?
  - The p-value is the probability that the null hypothesis is true
  - … the probability that the alternative hypothesis is false
  - … the probability that the finding is “merely a fluke”
  - A very small p-value implies a large and important effect
  - A p-value of >0.05 means the results are not worth publishing

Statistical significance, cont.

- This one is correct:
  - P-value is the probability of observing data as extreme (or more) as was observed, if null hypothesis is true
- Together with an estimate of effect size and/or the power of the test the p-value *does* tell whether the observation is *consistent* with the null hypothesis

"The p-value does not tell you if the result was due to chance. It tells you whether the results are *consistent* with being due to chance. That is not the same thing at all."

http://andrewgelman.com/2013/03/12/misunderstanding-the-p-value/

Significance, pop quiz

- Suppose there is a disease or a condition and a medical test for it. The false positive rate when using this test is 0.05
- A person is tested for the said condition. The result is positive.
- What is the probability that the result is a false positive, i.e. that the person is actually healthy?

Significance, alternative pop quiz

- Suppose there is a disease or a condition and a medical test for it. The false positive rate when using this test is 0.05
- Random people from the population are tested until 100 positive results are found
- How many of the 100 tests are false positives, i.e., how many of the 100 people are actually healthy?

Significance, pop quiz answer

- The question can not be answered with the given knowledge: you need to also know the prevalence of the disease (and the false negative rate, if different than false positive rate)
- The prevalence of the disease matters a lot:
  - Prevalence 1%: ~16 true positives
  - Prevalence 0.1%: 1-2 true positives
- Moral of the story: Rare events become common if you try enough many times!