Data %>% Power %>% R

You ready?
• a powerful open source programming language
• available in the database as Oracle R Enterprise
• that lets you do virtually anything...
Especially (but not only)

- Statistics
- Machine Learning
- Visualization
- Presentation (evidently)
Fine.

But I'm a SQL girl.

Now I gotta learn something totally new?
“All tidy datasets are alike, every untidy dataset is untidy in its own way.”
(Hadley Wickham)
library(readr)
library(dplyr)
library(tidyr)
library(ggplot2)
library(lubridate)
But we’re not just gonna do R syntax.

Let's look at some real data!
Everyone talks about the weather

I certainly do.
This is what I want for winter.
This is what I get.
Let’s try to find out about the future

- Berkeley Earth Surface Temperature Study
  - monthly averages, 1743-2013

- Weather data from weather station Munich airport
  - retrieved from [www.wunderground.com](http://www.wunderground.com)
  - daily averages, 1997-2015
Berkeley Earth global land temperatures

```r
df <- read_csv('data/GlobalLandTemperaturesByCity.csv')
head(df, 3)
```

```r
# A tibble: 3 x 7
  dt AverageTemperature AverageTemperatureUncertainty City
  <date>                  <dbl>                         <dbl> <chr>
1 1743-11-01             6.068                         1.737 Århus
2 1743-12-01              NA                            NA Århus
3 1744-01-01              NA                            NA Århus
# ... with 3 more variables: Country <chr>, Latitude <chr>, Longitude <chr>
```
Before we even start

*Let's get ourselves some nicer variable names.*

```r
df <- rename(df,
  avg_temp = AverageTemperature,
  avg_temp_95p = AverageTemperatureUncertainty,
  city = City,
  country = Country,
  lat = Latitude,
  long = Longitude)
head(df, 3)
```

```
# A tibble: 3 x 7

  dt avg_temp avg_temp_95p city    country     lat   long
  <date>   <dbl>        <dbl> <chr>   <chr>    <chr>  <chr>
1 1743-11-01 6.068       1.737 Århus Denmark 57.05N 10.33E
2 1743-12-01 NA            NA Århus Denmark 57.05N 10.33E
3 1744-01-01 NA            NA Århus Denmark 57.05N 10.33E
```
First thing I'd like to know: Which locations are available?

```r
head(distinct(df, country), 3)
# A tibble: 3 × 1
  country
1 Denmark
2 Turkey
3 Kazakhstan

head(distinct(df, city), 3)
# A tibble: 3 × 1
  city
1 Århus
2 Çorlu
3 Çorum
```
Let's see some Munich data!

```r
head(filter(df, city == 'Munich'), 3)
```

```
# A tibble: 3 × 7
dt avg_temp avg_temp_95p city country    lat   long
<date>    <dbl>        <dbl>  <chr>   <chr>  <chr>  <chr>
1 1743-11-01    1.323        1.783 Munich Germany 47.42N 10.66E
2 1743-12-01       NA           NA Munich Germany 47.42N 10.66E
3 1744-01-01       NA           NA Munich Germany 47.42N 10.66E
```
# AND
head(filter(df, city == 'Munich' & year(dt) > 2000), 3)

# A tibble: 3 × 7
dt avg_temp avg_temp_95p  city country    lat   long
<date>    <dbl>        <dbl> <chr>   <chr>  <chr>  <chr>
1 2001-01-01    -3.162        0.396 Munich Germany 47.42N 10.66E
2 2001-02-01    -1.221        0.755 Munich Germany 47.42N 10.66E
3 2001-03-01     3.165        0.512 Munich Germany 47.42N 10.66E

# OR
head(filter(df, city == 'Munich' | year(dt) > 2000), 3)

# A tibble: 3 × 7
dt avg_temp avg_temp_95p  city country    lat   long
<date>    <dbl>        <dbl> <chr>   <chr>  <chr>  <chr>
1 2001-01-01     1.918        0.381 Århus Denmark 57.05N 10.33E
2 2001-02-01     0.241        0.328 Århus Denmark 57.05N 10.33E
3 2001-03-01     1.310        0.236 Århus Denmark 57.05N 10.33E
Just concentrate on the important variables.

```r
# A tibble: 8 × 3
dt avg_temp avg_temp_95p
<date>    <dbl>        <dbl>
1 1743-11-01    1.323        1.783
2 1743-12-01       NA           NA
3 1744-01-01       NA           NA
4 1744-02-01       NA           NA
5 1744-03-01       NA           NA
6 1744-04-01    5.498        2.267
7 1744-05-01    7.918        1.603
8 1744-06-01   11.070        1.584
```
Coldest months ever.

```r
head(arrange(select(filter(df, city == 'Munich'), dt, avg_temp), avg_temp), 8)
```

```
# A tibble: 8 × 2
t   dt avg_temp
  <date>    <dbl>
1 1956-02-01  -12.008
2 1830-01-01  -11.510
3 1767-01-01  -11.384
4 1929-02-01  -11.168
5 1795-01-01  -11.019
6 1942-01-01  -10.785
7 1940-01-01  -10.643
8 1895-02-01  -10.551
```
All these parens are starting to be a pain...

What if we added still further data processing steps?
Enter: %>%

What you see: \(x \%>\% f(y)\)
What you get: \(f(x, y)\)
Now that we have %>%, we can tackle more complex statements.
What countries have most measurements?

going by (GROUP BY)

df %>% group_by(country) %>% summarise(count=n()) %>% arrange(count %>% desc()) %>% head(8)

# A tibble: 8 × 2
country  count
<chr>     <int>
1 India    1014906
2 China    827802
3 United States  687289
4 Brazil   475580
5 Russia   461234
6 Japan    358669
7 Indonesia 323255
8 Germany  262359
What are the average, min and max monthly temperatures in Germany after 1949?

```r
df %>% filter(country == 'Germany', !is.na(avg_temp), year(dt) > 1949) %>%
group_by(month(dt)) %>%
summarise(count = n(), avg = mean(avg_temp), min = min(avg_temp), max = max(avg_temp))
```

<table>
<thead>
<tr>
<th><code>month(dt)</code></th>
<th>count</th>
<th>avg</th>
<th>min</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5184</td>
<td>0.3329</td>
<td>-10.26</td>
<td>6.070</td>
</tr>
<tr>
<td>2</td>
<td>5184</td>
<td>1.1156</td>
<td>-12.00</td>
<td>7.233</td>
</tr>
<tr>
<td>3</td>
<td>5184</td>
<td>4.5513</td>
<td>-3.85</td>
<td>8.718</td>
</tr>
<tr>
<td>4</td>
<td>5184</td>
<td>8.2728</td>
<td>1.122</td>
<td>13.754</td>
</tr>
<tr>
<td>5</td>
<td>5184</td>
<td>12.9169</td>
<td>5.601</td>
<td>16.602</td>
</tr>
<tr>
<td>6</td>
<td>5184</td>
<td>15.9862</td>
<td>9.824</td>
<td>21.631</td>
</tr>
<tr>
<td>7</td>
<td>5184</td>
<td>17.8332</td>
<td>11.697</td>
<td>23.795</td>
</tr>
<tr>
<td>8</td>
<td>5184</td>
<td>17.4978</td>
<td>11.390</td>
<td>23.111</td>
</tr>
<tr>
<td>9</td>
<td>5103</td>
<td>14.8571</td>
<td>7.233</td>
<td>18.444</td>
</tr>
<tr>
<td>10</td>
<td>5103</td>
<td>9.4111</td>
<td>0.759</td>
<td>13.857</td>
</tr>
<tr>
<td>11</td>
<td>5103</td>
<td>4.6673</td>
<td>-2.601</td>
<td>9.127</td>
</tr>
<tr>
<td>12</td>
<td>5103</td>
<td>1.3649</td>
<td>-8.483</td>
<td>6.217</td>
</tr>
</tbody>
</table>
Let's join the two different data sources on the month column:

```r
monthly_1997_2015 <- daily_1997_2015 %>%
  group_by(month = floor_date(daily_1997_2015$day, "month")) %>%
  summarise(mean_temp = mean(mean_temp))

df_1949 <- df %>%
  select(dt, avg_temp) %>%
  filter(year(dt) > 1949)

df_1949 %>%
  inner_join(monthly_1997_2015, by = c("dt" = "month")) %>%
  head(3)
```

```
# A tibble: 3 × 3
  dt       avg_temp mean_temp
  <date>    <dbl>     <dbl>
1 1997-01-01 -0.742 -3.580645
2 1997-02-01  2.771  3.392857
3 1997-03-01  4.089  6.064516
```
Union of pre-2016 and 2016 Munich daily weather data.

daily_2016 <- read_csv('data/munich_2016.csv')
daily_1997_2015 %>% dplyr::union(daily_2016) %>% arrange(day) %>% head(8)

<table>
<thead>
<tr>
<th>day</th>
<th>max_temp</th>
<th>mean_temp</th>
<th>min_temp</th>
<th>dew</th>
<th>mean_dew</th>
<th>min_dew</th>
<th>max_hum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1997-01-01</td>
<td>-8</td>
<td>-12</td>
<td>-16</td>
<td>-13</td>
<td>-14</td>
<td>-17</td>
<td>92</td>
</tr>
<tr>
<td>1997-01-02</td>
<td>0</td>
<td>-8</td>
<td>-16</td>
<td>-9</td>
<td>-13</td>
<td>-18</td>
<td>92</td>
</tr>
<tr>
<td>1997-01-03</td>
<td>-4</td>
<td>-6</td>
<td>-7</td>
<td>-6</td>
<td>-8</td>
<td>-9</td>
<td>93</td>
</tr>
<tr>
<td>1997-01-04</td>
<td>-3</td>
<td>-4</td>
<td>-5</td>
<td>-5</td>
<td>-6</td>
<td>-6</td>
<td>93</td>
</tr>
<tr>
<td>1997-01-05</td>
<td>-1</td>
<td>-3</td>
<td>-6</td>
<td>-4</td>
<td>-5</td>
<td>-7</td>
<td>100</td>
</tr>
<tr>
<td>1997-01-06</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
<td>-4</td>
<td>-5</td>
<td>-6</td>
<td>93</td>
</tr>
<tr>
<td>1997-01-07</td>
<td>0</td>
<td>-4</td>
<td>-9</td>
<td>-6</td>
<td>-9</td>
<td>-10</td>
<td>93</td>
</tr>
<tr>
<td>1997-01-08</td>
<td>0</td>
<td>-3</td>
<td>-7</td>
<td>-7</td>
<td>-7</td>
<td>-8</td>
<td>100</td>
</tr>
</tbody>
</table>

# ... with 15 more variables: mean_hum, min_hum, max_hpa, mean_hpa, min_hpa, max_visib, mean_visib, min_visib, max_wind, mean_wind, max_gust, prep, cloud, events, winddir
5% hottest days in Munich in 2016.

```r
filter(daily_2016, cume_dist(desc(mean_temp)) < 0.05) %>% select(day, mean_temp) %>% arrange(desc(mean_temp))
```

<table>
<thead>
<tr>
<th>day</th>
<th>mean_temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-07-11</td>
<td>24</td>
</tr>
<tr>
<td>2016-06-24</td>
<td>22</td>
</tr>
<tr>
<td>2016-06-25</td>
<td>22</td>
</tr>
<tr>
<td>2016-07-29</td>
<td>22</td>
</tr>
<tr>
<td>2016-07-30</td>
<td>22</td>
</tr>
</tbody>
</table>
Top 3 coldest days in Munich in 2016.

```r
filter(daily_2016, dense_rank(mean_temp) < 4) %>%
  select(day, mean_temp) %>%
  arrange(mean_temp)
```

# A tibble: 4 × 2
  day mean_temp
  <date>     <int>
1 2016-01-22   -10
2 2016-01-19     -8
3 2016-01-18     -7
4 2016-01-20     -7
Consecutive days where mean temperatures differ by more than 5 degrees.

```
daily_2016 %>% mutate(yesterday_temp = lag(mean_temp)) %>% filter(abs(yesterday_temp - mean_temp) > 5) %>% select(day, mean_temp, yesterday_temp)
```

# A tibble: 6 × 3

<table>
<thead>
<tr>
<th>day</th>
<th>mean_temp</th>
<th>yesterday_temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-02-01</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>2016-02-21</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>2016-06-26</td>
<td>16</td>
<td>22</td>
</tr>
<tr>
<td>2016-07-12</td>
<td>18</td>
<td>24</td>
</tr>
<tr>
<td>2016-08-05</td>
<td>14</td>
<td>21</td>
</tr>
<tr>
<td>2016-08-13</td>
<td>19</td>
<td>13</td>
</tr>
</tbody>
</table>
OK. So we can do all that with R.

How about what we CAN'T do in SQL?
There was this original question about snow.

Let's see what we might predict about the future.
How cold is it in Munich, compared to, say, Bern or Oslo?

```r
df_cities <- df %>% filter(city %in% c("Munich", "Bern", "Oslo"), year(dt) > 1949, !is.na(avg_temp))
p_1950 <- ggplot(df_cities, aes(dt, avg_temp, color = city)) + geom_point() + xlab("") + ylab("avg monthly temp") + theme_solarized()
```
Can we see a trend?

```r
start_time <- as.Date("1992-01-01")
end_time <- as.Date("2013-08-01")
limits <- c(start_time, end_time)
(p_1992 <- p_1950 + (scale_x_date(limits=limits)) + geom_smooth(se = TRUE))
```
Let's concentrate on Munich.

```r
df_munich <- df_cities %>% filter(city == 'Munich')
p_munich_1950 <- ggplot(df_munich, aes(dt, avg_temp)) + geom_point() + xlab("") + ylab("avg monthly temp") + theme_solarized() + geom_smooth()
p_munich_1950
```
Difficult.

There might be a trend, or there might not.

Of course, there is not just one smoothing algorithm only.
LOESS (Local Polynomial Regression Fitting) vs. LM (Linear Model)

```r
loess <- p_munich_1950 + stat_smooth(method = "loess", colour = "red") + labs(title = 'loess')
lm <- p_munich_1950 + stat_smooth(method = "lm", color = "green") + labs(title = 'lm')
ggrid.arrange(loess, lm, ncol=2)
```
So?

We need to dig deeper and really analyze those time series.
Time series: A quick first glance

```r
# Time series 1950
ts_1950 <- ts(df_munich$avg_temp, start = c(1950,1), end=c(2013,8), frequency = 12)
# Time series 1997
par(mfrow=c(1,2))
plot(ts_1950, main = 'Munich, 1950-2013', ylab = 'avg. monthly temp.')
plot(ts_1997, main = 'Munich, 1997-2015', ylab = 'avg. monthly temp., aggregated')
par(mfrow=c(1,1))
```

![Munich, 1950-2013](chart1.png)

![Munich, 1997-2015](chart2.png)
Time series decomposition

Time series may be decomposed into different effects:

- trend
- seasonal (if available)
- remainder
library(stlplus)
fit <- stlplus(ts_1950, s.window="periodic", t.window=37)
plot(fit)
fit <- stlplus(ts_1997, s.window="periodic", t.window=37)
plot(fit)
Exponential Smoothing

- Predicted values at time $t+1$ are weighted averages of values at times $t$, $t-1$, $t-2$ ...:
  \[ \hat{y}_{T+1|T} = \alpha y_T + \alpha (1 - \alpha) y_{T-1} + \alpha (1 - \alpha)^2 y_{T-2} + \cdots \]

- Can also model trends and seasonal effects (Holt-Winters)
State Space Models

- Conceptually equivalent to exponential smoothing
- BUT also generate prediction intervals!
- http://www.exponentialsmoothing.net/
- R: ets() function in forecast package
• When called without parameters, ets() determines a suitable model itself
• Using maximum likelihood estimation
library(forecast)
fit <- ets(ts_1950)
summary(fit)

ETS(A,N,A)

Call:
  ets(y = ts_1950)

Smoothing parameters:
  alpha = 0.0202
  gamma = 1e-04

Initial states:
  l = 5.3364
  s=-8.3652 -4.6693 0.7114 5.4325 8.8662 9.3076
     7.3288 4.1447 -0.677 -4.3463 -8.2749 -9.4586

  sigma:  1.7217

AIC     AICc      BIC
5932.003 5932.644 6001.581

Training set error measures:
  ME    RMSE      MAE      MPE     MAPE      MASE
Training set 0.02606379 1.72165 1.345745 22.85555 103.3956 0.7208272

ACF1
Training set 0.09684748
# Plot the smoothed components - no trend!

```r
plot(fit)
```
# Forecast next 3 years
plot(forecast(fit, h=36))
fit <- ets(ts_1997)
summary(fit)

ETS(A,N,A)
Call:
  ets(y = ts_1997)
  
  Smoothing parameters:
      alpha = 0.2459
      gamma = 1e-04

  Initial states:
      l = 9.0339
      s=-9.1638 -8.2689 -4.6247 0.2518 4.8168 8.9262
  sigma:  1.6258

  AIC     AICc      BIC
  878.6766 882.4562 923.1193

  Training set error measures:
        ME     RMSE      MAE      MPE     MAPE      MASE
  Training set -0.0305885 1.625765 1.320291 4.385226 40.82934 0.7068611
        ACF1
  Training set 0.1407344
# Plot the smoothed components - no trend!
plot(fit)
# Forecast next 3 years
plot(forecast(fit, h=36))
**ARIMA**

- **AR(p):**
  autoregressive:
  \[ y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + e_t \]

- **I(d):**
  number of times differencing has to be applied to obtain a stationary series

- **MA(q):**
  moving average:
  \[ y_t = c + e_t + \theta_1 e_{t-1} + \theta_2 e_{t-2} + \cdots + \theta_q e_{t-q} \]

**Stationarity:** If a time series \( y_t \) is stationary, then for all \( s \), the distribution of \( (y_{t}, \ldots, y_{t+s}) \) does not depend on \( t \).
Stationarity: ACF and PACF

Berkeley Earth data, 1950-2013

```r
tsdisplay(ts_1950)
```
Munich weather data, 1997-2015

Stationarity: ACF and PACF

`tsdisplay(ts_1997)`
Cyclostationary process

Mean and variance are constant for seasonally corresponding measurements.
How do we decide which $p,d,q$ to use for ARIMA?

Just use `auto.arima()`.

(But see later.)
fit <- auto.arima(ts_1950)
summary(fit)

Series: ts_1950
ARIMA(1,0,5)(0,0,2)[12] with non-zero mean

Coefficients:
    ar1     ma1     ma2     ma3     ma4     ma5    sma1    sma2
 0.6895 -0.0794 -0.0408 -0.1266 -0.3003 -0.2461  0.3972  0.3555
  s.e. 0.0409  0.0535  0.0346  0.0389  0.0370  0.0347  0.0413  0.0342

intercept
  5.1667
  s.e. 0.1137

sigma^2 estimated as 7.242:  log likelihood=-1838.47
AIC=3696.94   AICc=3697.24   BIC=3743.33

Training set error measures:
ME  RMSE   MAE  MPE  MAPE  MASE
Training set -0.001132392 2.675279 2.121494 23.30576 116.9652 1.136345
ACF1
Training set 0.03740769
fit <- auto.arima(ts_1997)
summary(fit)

Series: ts_1997
ARIMA(0,0,4)(0,0,2)[12] with zero mean
Coefficients:
                 ma1     ma2     ma3     ma4    sma1    sma2
           0.9293  0.8470  0.6736  0.3146  0.6144  0.3083
            s.e.  0.0763  0.0973  0.0754  0.0652  0.0846  0.0590
sigma^2 estimated as 9.321:  log likelihood=-566.86
AIC=1147.72   AICc=1148.22   BIC=1171.72
Training set error measures:
                ME     RMSE      MAE      MPE    MAPE     MASE
Training set  1.275131 3.059881 2.417012 10.06209   85.7372 1.283957
                ACF1
Training set -0.141014
Let's get forecasting!

Wait.

ARIMA forecasts are based on the assumption that residuals are uncorrelated and normally distributed.
Residuals: Autocorrelation

Berkeley Earth data, 1950-2013

```r
fit <- auto.arima(ts_1950)
res <- fit$residuals
acf <- acf(res, main='Autocorrelation of residuals')
```
fit <- auto.arima(ts_1950, max.order = 10, stepwise=FALSE)
summary(fit)

Series: ts_1950
ARIMA(5,0,1)(0,0,2)[12] with non-zero mean

Coefficients:
      ar1  ar2  ar3  ar4  ar5  ma1  sma1  sma2
      0.8397 -0.0588 -0.0691 -0.2087 -0.2071 -0.4538  0.0869  0.1422
      s.e.  0.0531  0.0516  0.0471  0.0462  0.0436  0.0437  0.0368  0.0371

intercept
      5.1709
      s.e.     0.0701

sigma^2 estimated as 4.182:  log likelihood=-1629.11
AIC=3278.22   AICc=3278.51   BIC=3324.61

Training set error measures:

          ME   RMSE     MAE     MPE    MAPE    MASE
Training set -0.003793625 2.033009 1.607367 -0.2079903 110.0913 0.8609609

ACF1
Training set -0.0228192
Forecast: Berkeley Earth data, 1950-2013

plot(forecast(fit,h=36),include=80)
Allowing for more parameters: Munich weather data, 1997-2015

```r
fit <- auto.arima(ts_1997, max.order = 10, stepwise=FALSE)
summary(fit)
```

Series: ts_1997
ARIMA(4,0,0)(0,0,2)[12] with non-zero mean

Coefficients:
```
              ar1     ar2     ar3     ar4    sma1    sma2  intercept
  Estimate:  0.6874  0.1731  -0.0106  -0.5435  0.0501  -0.0097     8.9000
  Std. Error: 0.0577  0.0751   0.0750   0.0581  0.0772   0.0706     0.2174
```

sigma^2 estimated as 4.597:  log likelihood=-485.93
AIC=987.87   AICc=988.52   BIC=1015.3

Training set error measures:
```
                  ME     RMSE      MAE      MPE     MAPE      MASE
Training set -0.008284777 2.144125 1.683188 27.02366 84.54225 0.8941374
                  ACF1
Training set -0.1716898
```
plot(forecast(fit, h=36), include=80)
What happened?

- ARIMA has problems with too few data points
- ARIMA works better with the longer series
- while ETS handles the shorter series better
- uncertainty intervals are larger with ARIMA than with ETS
Beyond the weather ...

- Forecasting is required (and done!) everywhere
- R gives you all the tools you need
- you need to know what you're doing though!
... is about more than just having data.

It needs (a bit of) science to get to the insights!
At Trivadis, we’d love to help you with this

Thank you!