Data Science Training

Session 1 Exercises

DS Analytics, April 2018

Oil production and prices data

- Data source here https://www.eia.gov/petroleum/
- In this exercise we're going to practice the following skills
  - Reading in data from Excel
  - Cleaning up messy raw format
  - Handling more complicated date formats
  - Joining data sets together
  - Plotting
  - Outputting data to CSV

1) Download the two files - one is an xls file and one is a csv file and move them to an appropriate folder

2) Create a new R script

- Note use lots of white space and comment all your code sections
- Give objects sensible names

3) At the top load your packages, we're going to use the following:

```
# Data source: https://www.eia.gov/petroleum/gasdiesel/

# Load packages
library(tidyverse)
library(lubridate)
library(readxl)
```

- Take a look at the two files in Excel

PET_CRD_CRPDN_ADC_MBBL_M.xls

- Data is in the raw format I downloaded it in.
- The data we need is in the second sheet.
- We don't want to read in the first row. *(We could skip the first three rows and create our own column names, but won't right now)*
- The series we're going to use is in the second column.
- The date is in an unfriendly format.

prices_data.csv

- I've cleaned the data up a bit already.
- The date is in an unfriendly format.

Reading in Excel files

- Take a look at the documentation, either google 'readxl R package' or look at ?read_excel.
- We need to change arguments in the **read_excel()** function from its defaults.
- We need to tell it to choose the second sheet, to skip the first row and to not read in col_names.

4) Read in the PET_CRD_CRPDN_ADC_MBBL_M.xls data using the read_excel() function, calling the object 'crude_oil_production' setting the arguments as appropriate

5) Read in the prices_data.csv file as we've done before, calling the object 'petrol_prices'

6) Inspect the files using the head() function

7) Alter the 'crude_oil_production' object, keeping only the first two columns using the index method

8) Remove the first two rows

Examples of indexing:

```
# Indexing
# Keeping the first two rows and all columns
df <- df[1:2, ]

# Keeping the first two rows and first two columns
df <- df[1:2, 1:2]

# Removing the first two columns and keeping rows 1 to 10
df <- df[1:10, -c(1:2)]
```

- You should be left with a dataframe with dimensions (1117, 2)

7) Use the colnames() function to change the column names of the 'crude_oil_production' dataframe to "month_year" and "field_production"

```
# Hint - colnames() goes to the left of the assign arrow <-
# On the right of the assign arrow goes a vector of names
colnames(df) <- c('colname1', 'colname2')
```

- Excel converts dates to integer numbers starting at the start of last century.
- That's why the date column 'month_year' is a number.
- You'll see from str(crude_oil_production) that it's actually stored as a character vector (*chr*)
- To convert this to an R-friendly date, you need to tell the **as.Date()** function the 'origin' date from which this integer number is counting
- Before you do that, you'll have to convert the character column *month_year* to a numeric column, using the **as.numeric()** function

8) First convert the 'month_year' column to a numeric column

```
# Hint - converting column types you need the column vector to the left of the assign arrow
# Eg.
df$col_name <- as.character(df$col_name)
```

9) Use as.Date() to convert to a date column, setting the *origin=* argument to '1900-01-01'

- Note the dates you should get will be the 17th of the month

10) Convert the 'field_production' column to numeric

11) Inspect the dataframe using summary()

12) Use ggplot() to plot a time series, with 'month_year' on the x-axis and 'field_production' on the y-axis

```
# Hint - in this case put the aes() bit inside the ggplot() function.
# This is for what follows below. Ie.
ggplot(df, aes(x = x_column, y = y_column)) + geom_line()
```

13) Add a smoothed trend line to the plot using the geom_smooth() layer. You can use ?geom_smooth to check the methods available

14) Add additional layers to make the plot prettier. E.g. using themes, changing the axis labels and adding a title. Google if stuck

---

15) Convert the colnames in the 'petrol_prices' dataframe to lower case versions of themselves

- If you inspect the top of the dataframe, you should notice the date column is in the following format: 'Nov 28, 1994'
- We need to the tell the **as.Date()** function where to find the month, day and year bits

16) Use the help in ?strptime to find out which % symbols you need to specify the date types we've got

17) Use the as.Date() function with the *format* argument

- The *format=* argument takes a string with the % symbols

```
example_date_string <- "05 29-09"

as.Date(example_date_string, format = "%m %d-%y")
```

```
## [1] "2009-05-29"
```

- Use a similar logic to convert the date column

- This data is weekly
- We want to convert to monthly, averaging over the price variable, and then join on to the production table
- To do this we first need to add a *month_year* column
- To do this we're going to glue together the year, month and make the day the 17th, matching our production table

18) Use the piping method and mutate() to add new columns to the 'petrol_prices' dataframe. One which is the month of the 'date' column and one is the year

- Hint: use the **year()** and **month()** columns from the lubridate package

19) Create a 'month_year' column using the paste0() function to join up the year, month and '-16', giving us a string that looks like an R date. Then use the as.Date() function to convert to a date

```
# Hint
year_col = c(2011, 2011, 2012, 2012)
month_col = c(1, 5, 6, 2)
date_string <- paste0(year_col, "-", month_col, "-17")
as.Date(date_string)
```

```
## [1] "2011-01-17" "2011-05-17" "2012-06-17" "2012-02-17"
```

20) Create a new dataframe called 'petrol_prices_monthly' using the group_by() and summarise() functions to get monthly mean averages for petrol prices. The dataframe should have dimensions (282, 2)

Data Joins

- We use joins to merge multiple dataframes together
- There are several join functions but the most commonly used is **inner_join()**
- We'll go through how these work in the next session
- For now we're going to use **inner_join()** in a pipe stack to merge the 'crude_oil_production' onto the 'petrol_prices_monthly' dataframe

21) Try the following. Make sure you've named the date column 'month_year' in both dataframes

```
petrol_prices_monthly <- petrol_prices_monthly %>%
  inner_join(crude_oil_production)
```

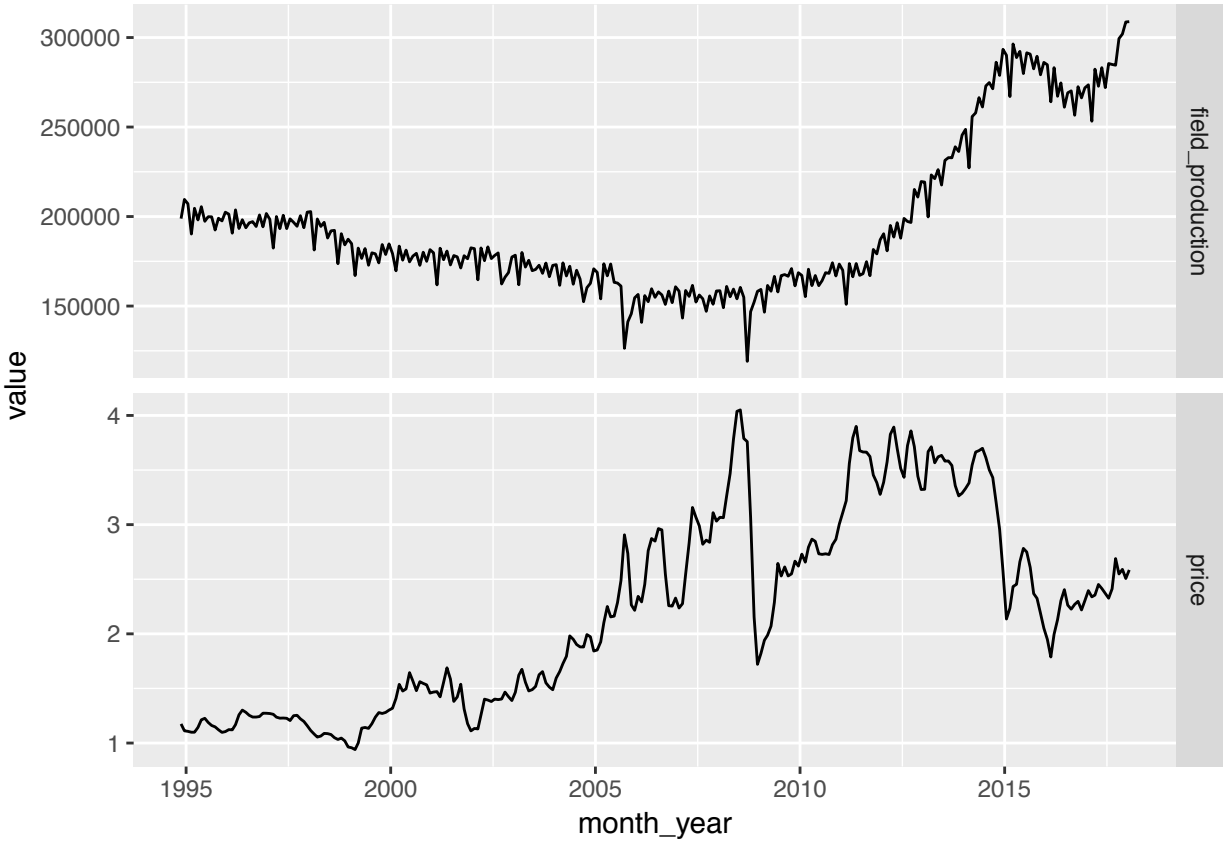
- You should have a dataframe with dimensions (279, 3) (wide format)
- We want to plot both time series on top of one another using the **facet_grid()**
- To do this we need to convert the dataframe into long format

22) Use the gather() function to convert 'petrol_prices_monthly from wide to long. The resulting dataframe has dimensions (558, 3)

- Hint: the **gather()** function needs the following elements: *key=* and *value =* arguments, which create new column names, and then the column names for the columns that are to be shifted about (i.e. all of them except the date column)

```
# Something like this:
petrol_prices_monthly <- petrol_prices_monthly %>%
  gather(key = ..., value = ..., col2, col3)
```

23) Plot the time series using ggplot. You should end up with something like this:



24) Create a CSV file with the 'petrol_prices_monthly' data frame using the write_csv() function