DIKW ACADEMY

P05 : DATA WRANGLING

DATA INFORMATIE KENNIS WIJSHEID

DIKW ACADEMY

CONTENTS

In this course we learn to read data from different sources and transform the data using Pandas dataframe techniques and more including:

```
Transforming data using apply and map functions
Transforming data using dfply verbs
```

DATA INFORMATIE KENNIS WIJSHEID

DIKW ACADEMY

CHEAT SHEETS

NumPy Basics Cheat Sheet

https://www.datacamp.com/community/blog/python-numpy-cheat-sheet

Pandas Basics Cheat Sheet

https://www.datacamp.com/community/blog/python-pandas-cheat-sheet

Pandas Data Wrangling Cheat Sheet

https://www.datacamp.com/community/blog/pandas-cheat-sheet-python

DATA WRANGLING

from wikipedia:

Data munging or data wrangling is loosely the process of manually converting or mapping data from one “raw” form into another format that allows for more convenient consumption of the data with the help of semi-automated tools.

In almost every dataset, it is necessary to perform some data munging before you can use the data to perform eg. a classification task. Therefore, as a data scientist, you’ll spend a lot of time munging and wrangling the data!

DATA MANIPULATION IN PYTHON

Let us first look at munging techniques available in basic Python:

```
Renaming columns
Adding or removing columns
Reordering columns
Merging data frames
Finding and removing duplicates
```

DIKW ACADEMY

LOAD IRIS DATASET AS DATAFRAME

The iris dataset is included in the sklearn.datasets. However, the dataset was changed in scikit-learn version 0.20 and fixed two wrong data points according to Fisher's paper. The new version is the same as in R, but not as in the UCI Machine Learning Repository.

In [77]:
```
# let's import a few libraries
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
```

DIKW ACADEMY

In [79]:
```
# https://stackoverflow.com/questions/38105539/how-to-convert-a-scikit-learn-dataset-to-a-pandas-dataset
# if you'd like to check dataset type use: type(load_iris())
# if you'd like to view list of attributes use: dir(load_iris())

# save load_iris() sklearn dataset to iris_bunch
iris_bunch = load_iris()

# list of attributes
dir(iris_bunch)
```

Out[79]:
```
['DESCR',
 'data',
 'data_module',
 'feature_names',
 'filename',
 'frame',
 'target',
 'target_names']
```

DATA INFORMATIE KENNIS WIJSHEID

DIKW ACADEMY

In [80]:
```
# so what's in iris_bunch?

# sample of the data
iris_bunch.data[1:10,]
```

Out[80]:
```
array([[4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1]])
```

In [81]:
```
# list with feature names
iris_bunch.feature_names
```

Out[81]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

DIKW ACADEMY

In [82]:
```
# target variable (3 different species)
iris_bunch.target
```

Out[82]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 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, 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, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

DIKW ACADEMY

In [83]:
```
# np.c_ is the numpy concatenate function
# which is used to concat iris['data'] and iris['target'] arrays
# for pandas column argument: concat iris['feature_names'] list
# and string list (in this case one string); you can make this anything you'd like..
# the original dataset would probably call this ['Species']
df_iris = pd.DataFrame(data= np.c_[iris_bunch['data'], iris_bunch['target']],
                       columns= iris_bunch['feature_names'] + ['target'])
```

In [84]:
```
df_iris.head()
```

Out[84]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0.0
1	4.9	3.0	1.4	0.2	0.0
2	4.7	3.2	1.3	0.2	0.0
3	4.6	3.1	1.5	0.2	0.0
4	5.0	3.6	1.4	0.2	0.0

DIKW ACADEMY

RENAMING COLUMNS IN A DATA FRAME

You can do any of the following with Python's built-in functions. Note that these modify iris dataframe directly; that is, you don't have to save the result back into df_iris when using the "inplace" argument with a value set to True.

RENAME THE COLUMN NAMED "SEPAL LENGTH (CM)" TO "SEPAL LENGTE (CM)"

In [86]:
```
df_iris.rename(columns={'sepal length (cm)':'sepal lengte (cm)'}, inplace=True)
df_iris.head()
```

Out[86]:

	sepal lengte (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0.0
1	4.9	3.0	1.4	0.2	0.0
2	4.7	3.2	1.3	0.2	0.0
3	4.6	3.1	1.5	0.2	0.0
4	5.0	3.6	1.4	0.2	0.0

DIKW ACADEMY

RENAME BY INDEX IN NAMES LIST: CHANGE THIRD ITEM, "PETAL LENGTH (CM)", TO "PETAL LENGTE (CM)"

In [87]:
```
df_iris.rename(columns={ df_iris.columns[2]: "petal lengte (cm)" }, inplace = True)
df_iris.head()
```

Out[87]:

	sepal lengte (cm)	sepal width (cm)	petal lengte (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0.0
1	4.9	3.0	1.4	0.2	0.0
2	4.7	3.2	1.3	0.2	0.0
3	4.6	3.1	1.5	0.2	0.0
4	5.0	3.6	1.4	0.2	0.0

DIKW ACADEMY

SHOW ALL COLUMN NAMES AS A LIST, INCLUDING THE MODIFIED COLUMN NAMES

```
In [88]: df_iris.columns.to_list()

Out[88]: ['sepal lengte (cm)',
          'sepal width (cm)',
          'petal lengte (cm)',
          'petal width (cm)',
          'target']
```

DIKW ACADEMY

ADDING AND REMOVING COLUMNS IN A DATA FRAME

There are many different ways of adding columns to a dataframe and removing columns from a data frame.

In [89]:
```
# A method used often to create a dataframe quickly is to define a dictionary containing your column names and data:
data = {'id': [1,2,3],
        'weight': [20,27,24]}

# ... and convert the dictionary into dataFrame
df = pd.DataFrame(data)
df
```

Out[89]:

	id	weight
0	1	20
1	2	27
2	3	24

DIKW ACADEMY

DIFFERENT WAYS TO ADD A COLUMN TO A DATAFRAME

In [90]:
```
# By declaring a new list as a column
# Declare a list that is to be converted into a column
size = ["small", "large", "medium"]

# Using 'size' as the column name
# and equating it to the list
df['size1'] = size
df
```

Out[90]:

	id	weight	size1
0	1	20	small
1	2	27	large
2	3	24	medium

In [91]:
```
# Using DataFrame.insert() to add a column
# The first argument is the position to insert the new column at
df.insert(len(df), "size2", ["small", "large", "medium"], True)
df
```

Out[91]:

	id	weight	size1	size2
0	1	20	small	small
1	2	27	large	large
2	3	24	medium	medium

DIKW ACADEMY

In [92]:
```
# Using Dataframe.assign() method
# This will create a new dataframe with the new column added to the old dataframe.

# Using 'size3' as the column name and equating it to the list
df = df.assign(size3 = ["small", "large", "medium"])
df
```

Out[92]:

	id	weight	size1	size2	size3
0	1	20	small	small	small
1	2	27	large	large	large
2	3	24	medium	medium	medium

DIKW ACADEMY

In [93]:
```
# Using a Python dictionary to add a column

# Define a dictionary with key values of an existing column and their respective
# value pairs as the values for our new column.

size4 = {1: 'small', 2: 'large', 3: 'medium'}

df['size4'] = df['id'].map(size4)
df
```

Out[93]:

	id	weight	size1	size2	size3	size4
0	1	20	small	small	small	small
1	2	27	large	large	large	large
2	3	24	medium	medium	medium	medium

DIKW ACADEMY

DIFFERENT WAYS TO REMOVE A COLUMN FROM A DATAFRAME

In [94]:
```
# using del
del df['size1']
df
```

Out[94]:

	id	weight	size2	size3	size4
0	1	20	small	small	small
1	2	27	large	large	large
2	3	24	medium	medium	medium

In [95]:
```
# using the dataframe drop method
df.drop('size2', axis=1, inplace=True)
df
```

Out[95]:

	id	weight	size3	size4
0	1	20	small	small
1	2	27	large	large
2	3	24	medium	medium

In [96]:
```
# by listing the columns you would like to keep
# and dropping any columns not listed
df=df[['id','size4']]
df
```

Out[96]:

	id	size4
0	1	small
1	2	large
2	3	medium

In [97]:
```
# The pop method is used to remove the specified column from the DataFrame and return the removed column as a pandas Series.
removed_col=df.pop('size4')
removed_col
```

Out[97]:
```
0     small
1     large
2    medium
Name: size4, dtype: object
```

DIKW ACADEMY

In [98]:
```
# Lets see what is left of the dataframe
df
```

Out[98]:

	id
0	1
1	2
2	3

DATA INFORMATIE KENNIS WIJSHEID

REORDERING COLUMNS IN A DATA FRAME

In [99]:
```
# Create a sample dataframe as we did earlier
# Define a dictionary containing data
data = {
        'id': [1,2,3],
        'weight': [20,27,24],
        'size': ["small", "large", "medium"]
}
# Convert the dictionary into DataFrame
df = pd.DataFrame(data)
df
```

Out[99]:

	id	weight	size
0	1	20	small
1	2	27	large
2	3	24	medium

DIKW ACADEMY

REORDER BY COLUMN NAMES

In [100]:
```
# Using double brackets to reorder columns
df = df[['weight', 'id', 'size']]
df
```

Out[100]:

	weight	id	size
0	20	1	small
1	27	2	large
2	24	3	medium

DIKW ACADEMY

In [101]:
```
# Using pandas.DataFrame.reindex() to reorder columns in a DataFrame
column_names = ['size', 'weight', 'id']
df = df.reindex(columns=column_names)
df
```

Out[101]:

	size	weight	id
0	small	20	1
1	large	27	2
2	medium	24	3

DIKW ACADEMY

In [102]:
```
# Reorder columns to the original sequence
df = df[['id', 'weight', 'size']]
df
```

Out[102]:

	id	weight	size
0	1	20	small
1	2	27	large
2	3	24	medium

In [103]:
```
# get values in a row using the row index to select the row
df.iloc[0]
```

Out[103]:
```
id            1
weight       20
size      small
Name: 0, dtype: object
```

DIKW ACADEMY

MERGING DATA FRAMES

You want to merge two data frames on a given column from each. Just like a join in SQL.

In [104]:
```
# Make a data frame, mapping story numbers to titles
data = {
        'storyid': [1,2,3],
        'title': ['lions', 'tigers', 'bears']
}
stories_df = pd.DataFrame.from_dict(data)
stories_df
```

Out[104]:

	storyid	title
0	1	lions
1	2	tigers
2	3	bears

DIKW ACADEMY

In [105]:
```
# Make another data frame with the data and story numbers, including ratings
data = {
        'subject': [1,1,1,2,2,2],
        'storyid': [1,2,3,2,3,1],
        'rating': [6.7,4.5,3.7,3.3,4.1,5.2]
}
rating_df = pd.DataFrame.from_dict(data)
rating_df
```

Out[105]:

	subject	storyid	rating
0	1	1	6.7
1	1	2	4.5
2	1	3	3.7
3	2	2	3.3
4	2	3	4.1
5	2	1	5.2

DIKW ACADEMY

In [106]:
```
# Merge the two data frames
# how='left' is the default (left join), so it is not required to provide it, but for clarity we do
# Here we provide it in order to make the code more readable
stories_df.merge(rating_df, on=['storyid'], how='left')
```

Out[106]:

	storyid	title	subject	rating
0	1	lions	1	6.7
1	1	lions	2	5.2
2	2	tigers	1	4.5
3	2	tigers	2	3.3
4	3	bears	1	3.7
5	3	bears	2	4.1

MERGING DATA FRAMES ON MORE THAN ONE COLUMN

In [107]:
```
# Make up more data: create a dataframe with animals and a dataframe with observations
data = {
        'size': ['small', 'big', 'small', 'big'],
        'type': ['cat', 'cat', 'dog', 'dog'],
        'name': ['lynx', 'tiger', 'chihuahua', 'great dane']
}
animals_df = pd.DataFrame.from_dict(data)
print(animals_df)
```

```
    size type        name
0  small  cat        lynx
1    big  cat       tiger
2  small  dog   chihuahua
3    big  dog  great dane
```

DIKW ACADEMY

In [108]:
```
data = {'number':[1,2,3,4], 'size': ['big', 'small', 'small', 'big'], 'type': ['cat', 'dog', 'dog', 'dog']}
observations_df = pd.DataFrame.from_dict(data)
print(f"\n{observations_df}" )

# Merge the dataframes
observations_df.merge(animals_df, on=['size', 'type'], how='left')
```

```
   number   size type
0       1    big  cat
1       2  small  dog
2       3  small  dog
3       4    big  dog
```

Out[108]:

	number	size	type	name
0	1	big	cat	tiger
1	2	small	dog	chihuahua
2	3	small	dog	chihuahua
3	4	big	dog	great dane

DATA INFORMATIE KENNIS WIJSHEID

DIKW ACADEMY

FINDING AND REMOVING DUPLICATE RECORDS IN A SERIE

You want to find and/or remove duplicate entries from a serie or data frame.

In [110]:
```
# import the required packages we will be using
from numpy.random import seed
from numpy.random import normal

# Generate a series
seed(3)
x = pd.Series(normal(loc=10, scale=5, size=20).round()).astype(int)
print(*x)
```

```
19 12 10 1 9 8 10 7 10 8 3 14 14 19 10 8 7 2 15 4
```

In [111]:

```
# For each element: is this one a duplicate (first instance of a particular value not counted)
x_duplicates = x.duplicated()
print(*x_duplicates)

# The values of the duplicated entries
# note that '10' appears in the original vector four times, and so it has three entries here
x_duplicates_values = x[x.duplicated()]
print(*x_duplicates_values)
```

```
False False False False False False True False True True False False True True True True True False False False
10 10 8 14 19 10 8 7
```

DIKW ACADEMY

In [112]:
```
# create a list with duplicates
x_list = [1, 2, 3, 3, 4, 4, 5]

# a set is another python data structure: it can only contains unique items
# and therefore we can use it to remove duplicates and convert the set back to a list again
list(set(x_list))
```

Out[112]: [1, 2, 3, 4, 5]

DATA INFORMATIE KENNIS WIJSHEID

DIKW ACADEMY

FINDING AND REMOVING DUPLICATE RECORDS IN DATA FRAMES

In [113]:
```
# Create a sample data frame:
data = {
        'label': ['A', 'B', 'C', 'B', 'B', 'A', 'A', 'A'],
        'type': [4,3,6,3,1,2,4,4]
}
```

DIKW ACADEMY

In [114]:
```
df = pd.DataFrame.from_dict(data)
print(df)

# Is each row a repeat?
x_duplicates = df.duplicated()
x_duplicates
```

```
  label  type
0     A     4
1     B     3
2     C     6
3     B     3
4     B     1
5     A     2
6     A     4
7     A     4
```

Out[114]:
```
0    False
1    False
2    False
3     True
4    False
5    False
6     True
7     True
dtype: bool
```

DATA INFORMATIE KENNIS WIJSHEID

DIKW ACADEMY

In [115]:
```
df_duplicates_values = df[df.duplicated()]
df_duplicates_values
```

Out[115]:

	label	type
3	B	3
6	A	4
7	A	4

In [116]:
```
# Show unique repeat entries
df_duplicates_values_unique = df_duplicates_values.drop_duplicates()
df_duplicates_values_unique
```

Out[116]:

	label	type
3	B	3
6	A	4

In [117]:
```
# Original data with repeats removed. These do the same:
df_unique1 = df.drop_duplicates()
df_unique1
```

Out[117]:

	label	type
0	A	4
1	B	3
2	C	6
4	B	1
5	A	2

In [118]:
```
df_unique2 = df[~df.duplicated()]
df_unique2
```

Out[118]:

	label	type
0	A	4
1	B	3
2	C	6
4	B	1
5	A	2

INTRODUCING APPLY

At any Python Q&A site, you'll frequently see an exchange like this one:

Q: How can I use a loop to [...insert task here...] ? A: Don't. Use one of the apply functions or the Python list comprehension

So, what are these wondrous apply functions and list comprehensions and how do they work? I think the best way to figure out anything in Python is to learn by experimentation, using embarrassingly trivial data and functions.

Let's examine some of those.

DIKW ACADEMY

APPLY_ALONG_AXIS

Description : Returns a series or array or list of values obtained by applying a function to margins of an array or matrix.

OK. We know about series/arrays and functions, but what are these “margins”?

Simple: either the rows (0), the columns (1) or both. By both, we mean apply the function to each individual value.

In [119]:

```
# An example:

# create a matrix of 10 rows x 2 columns
# Python doesn't have a built-in type for matrices.
# However, we can treat a list of a list as a matrix
a = np.array([range(1,11)])
b = np.array([range(11,21)])
m = np.vstack((a,b))
print(m)
```

```
[[ 1  2  3  4  5  6  7  8  9 10]
 [11 12 13 14 15 16 17 18 19 20]]
```

DIKW ACADEMY

In [120]:
```
# mean of the rows
mean_rows = np.apply_along_axis(np.mean, 0, m)
print(*mean_rows)
```

```
6.0 7.0 8.0 9.0 10.0 11.0 12.0 13.0 14.0 15.0
```

In [121]:
```
# mean of the columns
mean_cols = np.apply_along_axis(np.mean, 1, m)
print(*mean_cols)
```

```
5.5 15.5
```

DIKW ACADEMY

In [122]:
```
# Divide all values by 2
# notice that here the axis argument is simply ignored but a value has to be provided
div_2 = np.apply_along_axis(lambda x: x/2, 1, m)
print(div_2.T)
```

```
[[ 0.5  5.5]
 [ 1.   6. ]
 [ 1.5  6.5]
 [ 2.   7. ]
 [ 2.5  7.5]
 [ 3.   8. ]
 [ 3.5  8.5]
 [ 4.   9. ]
 [ 4.5  9.5]
 [ 5.  10. ]]
```

GROUPBY

To illustrate, we can load up the classic iris dataset, which contains a bunch of flower measurements:

In [123]:
```
# As we saw before, the iris data is available in the seaborn package:
import seaborn as sns
# The following script loads the iris data set into a data frame
df_iris_seaborn = sns.load_dataset('iris')
df_iris_seaborn
```

Out[123]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

DIKW ACADEMY

In [124]:
```
df_iris_seaborn.groupby('species').agg('mean')
```

Out[124]:

species	sepal_length	sepal_width	petal_length	petal_width
setosa	5.006	3.428	1.462	0.246
versicolor	5.936	2.770	4.260	1.326
virginica	6.588	2.974	5.552	2.026

Essentially, groupby provides a way to split your data by factors and do calculations on each subset. It returns an object of class “groupby” and there are many more complex ways to use it.

DIKW ACADEMY

MAP

Description : map returns a map object of the same length as X, each element of which is the result of applying FUN to the corresponding element of X

Maps are lazily evaluated, meaning the values are only computed on-demand.

A simple example:

In [125]:
```
import statistics
l = [list(range(1,11)), list(range(11,21))]
mean_map=map(statistics.mean, l)
print(list(mean_map))
```

```
[5.5, 15.5]
```

DIKW ACADEMY

In [126]:
```
# the sum of the values in each element
sum_map=map(sum, l)
print(list(sum_map))
```

```
[55, 155]
```

In [127]:
```
# Notice: alternatively, you can also use list comprehension:
sum_list=[sum(i) for i in l]
print(sum_list)
```

```
[55, 155]
```

DIKW ACADEMY

In [128]:
```
# calculate multiple values
mult_funcs = [min, statistics.median, max]

mean_sum_list = [[x(i) for x in mult_funcs] for i in l]
print(mean_sum_list)
```

```
[[1, 5.5, 10], [11, 15.5, 20]]
```

DATA INFORMATIE KENNIS WIJSHEID

DIKW ACADEMY

REPLICATE WITH NUMPY

NumPy is a package for working with numerical data and used for data science and scientific computing.

An example: let's simulate 5 normal distributions, each with 5 observations:

```
In [129]: np.random.seed(1)
          np.random.normal(loc=0.48, scale=0.05, size=(5,5))

Out[129]: array([[0.56121727, 0.44941218, 0.45359141, 0.42635157, 0.52327038],
                 [0.36492307, 0.56724059, 0.44193965, 0.49595195, 0.46753148],
                 [0.5531054 , 0.37699296, 0.46387914, 0.46079728, 0.53668847],
                 [0.42500544, 0.47137859, 0.43610708, 0.48211069, 0.50914076],
                 [0.42496904, 0.53723619, 0.52507954, 0.50512472, 0.5250428 ]])
```

REPLICATE A FUNCTION ON ELEMENTS WITH MAP

With map we can apply a function to multiple list arguments

In [130]:
```
one_to_four = range(1, 5)
four_to_one = range(4, 0, -1)

rep = lambda value, times: [value]*times
list(map(rep, one_to_four, four_to_one))
```

Out[130]:
```
[[1, 1, 1, 1], [2, 2, 2], [3, 3], [4]]
```

DIKW ACADEMY

DFPLY

The dfply package makes it possible to do R's dplyr-style data manipulation with pipes in Python on pandas DataFrames.

This is an alternative to pandas-ply and dplython, which both engineer dplyr syntax and functionality in Python. There are probably more packages that attempt to enable dplyr-style dataframe manipulation in Python, but those are the two I am aware of.

dfply uses a decorator-based architecture for the piping functionality and to "categorize" the types of data manipulation functions. The goal of this architecture is to make dfply concise and easily extensible, simply by chaining together different decorators that each have a distinct effect on the wrapped function.

dfply is intended to mimic the functionality of dplyr. The syntax is the same for the most part, but will vary in some cases as Python is a considerably different programming language than R.

from kieferk/dfply

DIKW ACADEMY

DFPLY

When working with data you must:

- Figure out what you want to do.
- Precisely describe what you want in the form of a computer program.
- Execute the code.

The dfply package makes each of these steps as fast and easy as possible by:

- Elucidating the most common data manipulation operations, so that your options are helpfully constrained when thinking about how to tackle a problem.
- Providing simple functions that correspond to the most common data manipulation verbs, so that you can easily translate your thoughts into code.
- Using efficient data storage backends, so that you spend as little time waiting for the computer as possible.

DATA: NYCFLIGHTS13

To explore the basic data manipulation verbs of dfply, we'll start with the built-in nycflights13 data frame. This dataset contains all 336776 flights that departed from New York City in 2013. The data comes from the US Bureau of Transporation Statistics

In [132]:
```
from dfply import *
import pandas as pd

flight_data = pd.read_csv('./datasets/flights.csv')
flight_data.shape
```

Out[132]: (336776, 19)

DIKW ACADEMY

In [133]:
```
flight_data.head()
```

Out[133]:

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	flight	tailnum	origin	dest	air_time	distance	hour	minute	time_hour
0	2013	1	1	517.0	515	2.0	830.0	819	11.0	UA	1545	N14228	EWR	IAH	227.0	1400	5	15	2013-01-01T10:00:00Z
1	2013	1	1	533.0	529	4.0	850.0	830	20.0	UA	1714	N24211	LGA	IAH	227.0	1416	5	29	2013-01-01T10:00:00Z
2	2013	1	1	542.0	540	2.0	923.0	850	33.0	AA	1141	N619AA	JFK	MIA	160.0	1089	5	40	2013-01-01T10:00:00Z
3	2013	1	1	544.0	545	-1.0	1004.0	1022	-18.0	B6	725	N804JB	JFK	BQN	183.0	1576	5	45	2013-01-01T10:00:00Z
4	2013	1	1	554.0	600	-6.0	812.0	837	-25.0	DL	461	N668DN	LGA	ATL	116.0	762	6	0	2013-01-01T11:00:00Z

DIKW ACADEMY

SINGLE TABLE VERBS

Dfly aims to provide a function for each basic verb of data manipulating:

- filter_by() (and row_slice())
- arrange()
- select() (and rename())
- distinct()
- mutate() (and transmute())
- summarize()
- sample()

DIKW ACADEMY

In [134]:
```
# dfply works directly on pandas DataFrames, chaining operations on the data with the >> operator

flight_data >> head(5)
```

Out[134]:

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	flight	tailnum	origin	dest	air_time	distance	hour	minute	time_hour
0	2013	1	1	517.0	515	2.0	830.0	819	11.0	UA	1545	N14228	EWR	IAH	227.0	1400	5	15	2013-01-01T10:00:00Z
1	2013	1	1	533.0	529	4.0	850.0	830	20.0	UA	1714	N24211	LGA	IAH	227.0	1416	5	29	2013-01-01T10:00:00Z
2	2013	1	1	542.0	540	2.0	923.0	850	33.0	AA	1141	N619AA	JFK	MIA	160.0	1089	5	40	2013-01-01T10:00:00Z
3	2013	1	1	544.0	545	-1.0	1004.0	1022	-18.0	B6	725	N804JB	JFK	BQN	183.0	1576	5	45	2013-01-01T10:00:00Z
4	2013	1	1	554.0	600	-6.0	812.0	837	-25.0	DL	461	N668DN	LGA	ATL	116.0	762	6	0	2013-01-01T11:00:00Z

DIKW ACADEMY

In [135]:
```
# The DataFrame as it is passed through the piping operations is represented by the symbol X
flight_data >> select(X.dep_time, X.origin) >> head(3)
```

Out[135]:

	dep_time	origin
0	517.0	EWR
1	533.0	LGA
2	542.0	JFK

DIKW ACADEMY

ARRANGE ROWS WITH ARRANGE()

arrange() works similarly to filter() except that instead of filtering or selecting rows, it reorders them. If you provide more than one column name, each additional column will be used to break ties in the values of preceding columns:

In [136]:
```
flight_data >> arrange(X.year, X.month, X.day) >> head(5)
```

Out[136]:

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	flight	tailnum	origin	dest	air_time	distance	hour	minute	time_hour
0	2013	1	1	517.0	515	2.0	830.0	819	11.0	UA	1545	N14228	EWR	IAH	227.0	1400	5	15	2013-01-01T10:00:00Z
1	2013	1	1	533.0	529	4.0	850.0	830	20.0	UA	1714	N24211	LGA	IAH	227.0	1416	5	29	2013-01-01T10:00:00Z
2	2013	1	1	542.0	540	2.0	923.0	850	33.0	AA	1141	N619AA	JFK	MIA	160.0	1089	5	40	2013-01-01T10:00:00Z
3	2013	1	1	544.0	545	-1.0	1004.0	1022	-18.0	B6	725	N804JB	JFK	BQN	183.0	1576	5	45	2013-01-01T10:00:00Z
4	2013	1	1	554.0	600	-6.0	812.0	837	-25.0	DL	461	N668DN	LGA	ATL	116.0	762	6	0	2013-01-01T11:00:00Z

Use desc() to order a column in descending order:

In [137]:
```
flight_data >> arrange(X.arr_delay, ascending=False) >> head(5)
```

Out[137]:

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	flight	tailnum	origin	dest	air_time	distance	hour	minute	time_hour
7072	2013	1	9	641.0	900	1301.0	1242.0	1530	1272.0	HA	51	N384HA	JFK	HNL	640.0	4983	9	0	2013-01-09T14:00:00Z
235778	2013	6	15	1432.0	1935	1137.0	1607.0	2120	1127.0	MQ	3535	N504MQ	JFK	CMH	74.0	483	19	35	2013-06-15T23:00:00Z
8239	2013	1	10	1121.0	1635	1126.0	1239.0	1810	1109.0	MQ	3695	N517MQ	EWR	ORD	111.0	719	16	35	2013-01-10T21:00:00Z
327043	2013	9	20	1139.0	1845	1014.0	1457.0	2210	1007.0	AA	177	N338AA	JFK	SFO	354.0	2586	18	45	2013-09-20T22:00:00Z
270376	2013	7	22	845.0	1600	1005.0	1044.0	1815	989.0	MQ	3075	N665MQ	JFK	CVG	96.0	589	16	0	2013-07-22T20:00:00Z

FILTER ROWS WITH FILTER_BY()

filter_by() allows you to select a subset of the rows of a data frame. The first and subsequent arguments are filtering expressions evaluated in the context of that data frame.

For example, we can select all flights on February 3rd with:

```
In [138]: flight_data >> filter_by(X.month==2, X.day==3) >> head(5)
```

Out[138]:

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	flight	tailnum	origin	dest	air_time	distance	hour	minute	time_hour
112904	2013	2	3	31.0	2359	32.0	458.0	437	21.0	B6	727	N506JB	JFK	BQN	190.0	1576	23	59	2013-02-04T04:00:00Z
112905	2013	2	3	536.0	540	-4.0	927.0	850	37.0	AA	1141	N5EBAA	JFK	MIA	164.0	1089	5	40	2013-02-03T10:00:00Z
112906	2013	2	3	545.0	540	5.0	1049.0	1017	32.0	B6	725	N571JB	JFK	BQN	203.0	1576	5	40	2013-02-03T10:00:00Z
112907	2013	2	3	556.0	600	-4.0	833.0	815	18.0	FL	345	N923AT	LGA	ATL	113.0	762	6	0	2013-02-03T11:00:00Z
112908	2013	2	3	556.0	600	-4.0	935.0	912	23.0	B6	135	N508JB	JFK	RSW	163.0	1074	6	0	2013-02-03T11:00:00Z

MORE FILTER() AND SLICE()

This is equivalent as it can be done using pandas:

In [139]:
```
flight_data[(flight_data["month"] == 1) | (flight_data["day"] == 1)].head(5)
```

Out[139]:

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	flight	tailnum	origin	dest	air_time	distance	hour	minute	time_hour
0	2013	1	1	517.0	515	2.0	830.0	819	11.0	UA	1545	N14228	EWR	IAH	227.0	1400	5	15	2013-01-01T10:00:00Z
1	2013	1	1	533.0	529	4.0	850.0	830	20.0	UA	1714	N24211	LGA	IAH	227.0	1416	5	29	2013-01-01T10:00:00Z
2	2013	1	1	542.0	540	2.0	923.0	850	33.0	AA	1141	N619AA	JFK	MIA	160.0	1089	5	40	2013-01-01T10:00:00Z
3	2013	1	1	544.0	545	-1.0	1004.0	1022	-18.0	B6	725	N804JB	JFK	BQN	183.0	1576	5	45	2013-01-01T10:00:00Z
4	2013	1	1	554.0	600	-6.0	812.0	837	-25.0	DL	461	N668DN	LGA	ATL	116.0	762	6	0	2013-01-01T11:00:00Z

To select rows by position, use row_slice(). You can pass single integer indices or a list of indices to select rows as with. This is going to be the same as using pandas' .iloc.

In [140]:
```
flight_data >> row_slice([0,7,10])
```

Out[140]:

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	flight	tailnum	origin	dest	air_time	distance	hour	minute	time_hour
0	2013	1	1	517.0	515	2.0	830.0	819	11.0	UA	1545	N14228	EWR	IAH	227.0	1400	5	15	2013-01-01T10:00:00Z
7	2013	1	1	557.0	600	-3.0	709.0	723	-14.0	EV	5708	N829AS	LGA	IAD	53.0	229	6	0	2013-01-01T11:00:00Z
10	2013	1	1	558.0	600	-2.0	849.0	851	-2.0	B6	49	N793JB	JFK	PBI	149.0	1028	6	0	2013-01-01T11:00:00Z

SELECT COLUMNS WITH SELECT()

Often you work with large datasets with many columns where only a few are actually of interest to you. select() allows you to rapidly zoom in on a useful subset using operations that usually only work on numeric variable positions:

In [141]:
```
# Select columns by name
flight_data >> select(X.year, X.month, X.day) >> head(10)
```

Out[141]:

	year	month	day
0	2013	1	1
1	2013	1	1
2	2013	1	1
3	2013	1	1
4	2013	1	1
5	2013	1	1
6	2013	1	1
7	2013	1	1
8	2013	1	1
9	2013	1	1

DIKW ACADEMY

In [142]:
```
# The select function accept string labels, integer positions, and/or symbolically represented column names (X.column)
flight_data >> select(X.year, ['month', 'day'],6) >> head(10)
```

Out[142]:

	year	month	day	arr_time
0	2013	1	1	830.0
1	2013	1	1	850.0
2	2013	1	1	923.0
3	2013	1	1	1004.0
4	2013	1	1	812.0
5	2013	1	1	740.0
6	2013	1	1	913.0
7	2013	1	1	709.0
8	2013	1	1	838.0
9	2013	1	1	753.0

EXTRACT DISTINCT (UNIQUE) ROWS

A common use of select() is to find out which values a set of variables takes. This is particularly useful in conjunction with the distinct() verb which only returns the unique values in a table.

In [143]:
```
flight_data >> select(X.tailnum) >> distinct(X.tailnum)
```

Out[143]:

	tailnum
0	N14228
1	N24211
2	N619AA
3	N804JB
4	N668DN
...	...
327436	N766SK
329041	N772SK
330033	N776SK
331007	N785SK
334259	N557AS

4044 rows × 1 columns

DIKW ACADEMY

GROUPED OPERATIONS

These verbs are useful, but they become really powerful when you combine them with the idea of “group by”, repeating the operation individually on groups of observations within the dataset.

The verbs are affected by grouping as follows:

- grouped select() is the same as ungrouped select(), except that grouping variables are always retained.
- grouped arrange() orders first by grouping variables
- mutate() and filter() are most useful in conjunction with window functions (like rank(), or min(x) == x), and are described in detail in vignette("window-function").
- sample() samples the specified number/fraction of rows in each group.
- row_slice() extracts rows within each group.
- summarize() and summarize_each() are easy to understand and very useful, and is described in more detail below.

DIKW ACADEMY

EXAMPLE GROUP BY

In the following example, we split the complete dataset into individual planes and then summarise each plane by counting the number of flights (count = n()) and computing the average distance (distance_mean=X.distance.mean()) and delay (arr_delay_mean=X.arr_delay.mean()).

DIKW ACADEMY

In [144]:
```
delay_df = (flight_data >>
 group_by(X.tailnum) >>
 summarize(tailnum_count=n(X.tailnum), distance_mean=X.distance.mean(), arr_delay_mean=X.arr_delay.mean()) >>
 filter_by(X.tailnum_count >20, X.distance_mean<2000)
)
delay_df >> head(5)
```

Out[144]:

	tailnum	tailnum_count	distance_mean	arr_delay_mean
1	N0EGMQ	371	676.188679	9.982955
2	N10156	153	757.947712	12.717241
3	N102UW	48	535.875000	2.937500
4	N103US	46	535.195652	-6.934783
5	N104UW	47	535.255319	1.804348
...	...	...	...	...
4038	N997DL	63	867.761905	4.903226
4039	N998AT	26	593.538462	29.960000
4040	N998DL	77	857.818182	16.394737
4041	N999DN	61	895.459016	14.311475
4042	N9EAMQ	248	674.665323	9.235294

2961 rows × 4 columns

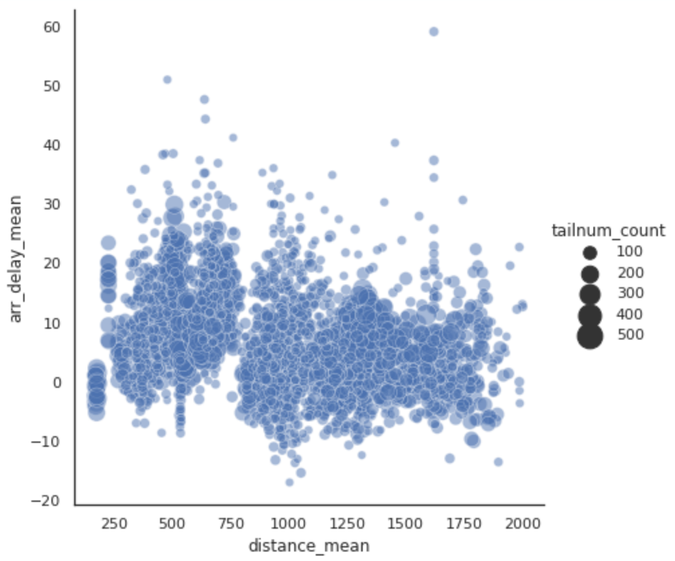
PLOT DELAYS

In [145]:
```
# import the seaborn package for making a visualisation
import seaborn as sns
sns.set_theme(style="white")
```

DIKW ACADEMY

In [146]:
```
# Plot miles per gallon against horsepower with other semantics
sns.relplot(x="distance_mean", y="arr_delay_mean", size="tailnum_count",
            sizes=(40, 400), alpha=.5, palette="muted",
            height=6, data=delay_df)
```

Out[146]: <seaborn.axisgrid.FacetGrid at 0x7fb4308028e0>



DATA INFORMATIE KENNIS WIJSHEID

ADD NEW COLUMNS WITH MUTATE()

As well as selecting from the set of existing columns, it's often useful to add new columns that are functions of existing columns. This is the job of mutate():

In [147]:
```
(flight_data >>
 mutate(
  gain = X.arr_delay - X.dep_delay,
  speed = X.distance / X.air_time * 60) >>
 head())
```

Out[147]:

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	...	tailnum	origin	dest	air_time	distance	hour	minute	time_hour	gain	speed
0	2013	1	1	517.0	515	2.0	830.0	819	11.0	UA	...	N14228	EWR	IAH	227.0	1400	5	15	2013-01-01T10:00:00Z	9.0	370.044053
1	2013	1	1	533.0	529	4.0	850.0	830	20.0	UA	...	N24211	LGA	IAH	227.0	1416	5	29	2013-01-01T10:00:00Z	16.0	374.273128
2	2013	1	1	542.0	540	2.0	923.0	850	33.0	AA	...	N619AA	JFK	MIA	160.0	1089	5	40	2013-01-01T10:00:00Z	31.0	408.375000
3	2013	1	1	544.0	545	-1.0	1004.0	1022	-18.0	B6	...	N804JB	JFK	BQN	183.0	1576	5	45	2013-01-01T10:00:00Z	-17.0	516.721311
4	2013	1	1	554.0	600	-6.0	812.0	837	-25.0	DL	...	N668DN	LGA	ATL	116.0	762	6	0	2013-01-01T11:00:00Z	-19.0	394.137931

5 rows × 21 columns

DIKW ACADEMY

COMBINE DATASETS

- Mutating joins
- Filtering joins
- Set operations
- Binding

RANDOMLY SAMPLE ROWS WITH SAMPLE()

You can use sample() to take a random sample of rows, either a fixed number for sample(n=3, replace=True) or a fixed fraction for sample(frac=0.0002, replace=False).

In [148]:
```
flight_data >> sample(n=3, replace=True)
```

Out[148]:

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	flight	tailnum	origin	dest	air_time	distance	hour	minute	time_hour
328986	2013	9	22	1707.0	1715	-8.0	1937.0	2015	-38.0	AA	2488	N4YHAA	EWR	DFW	181.0	1372	17	15	2013-09-22T21:00:00Z
75368	2013	11	21	2055.0	2015	40.0	2201.0	2126	35.0	B6	418	N337JB	JFK	BOS	44.0	187	20	15	2013-11-22T01:00:00Z
1046	2013	1	2	845.0	842	3.0	1102.0	1059	3.0	EV	4255	N34110	EWR	CHS	108.0	628	8	42	2013-01-02T13:00:00Z

DIKW ACADEMY

```
In [149]: flight_data >> sample(frac=0.00002, replace=True)
```

Out[149]:

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	flight	tailnum	origin	dest	air_time	distance	hour	minute	time_hour
112393	2013	2	2	858.0	900	-2.0	1440.0	1540	-60.0	HA	51	N388HA	JFK	HNL	629.0	4983	9	0	2013-02-02T14:00:00Z
150503	2013	3	16	907.0	910	-3.0	1023.0	1032	-9.0	B6	20	N284JB	JFK	ROC	53.0	264	9	10	2013-03-16T13:00:00Z
214142	2013	5	23	839.0	755	44.0	1023.0	920	63.0	MQ	3737	N3AEMQ	EWR	ORD	111.0	719	7	55	2013-05-23T11:00:00Z
238871	2013	6	18	2050.0	2000	50.0	2239.0	2127	72.0	9E	4287	N836AY	JFK	BWI	39.0	184	20	0	2013-06-19T00:00:00Z
101732	2013	12	20	2158.0	2112	46.0	2343.0	2312	31.0	US	1859	N182UW	EWR	CLT	81.0	529	21	12	2013-12-21T02:00:00Z
82075	2013	11	29	1624.0	1621	3.0	1831.0	1844	-13.0	UA	485	N831UA	EWR	DEN	229.0	1605	16	21	2013-11-29T21:00:00Z
20793	2013	1	24	1927.0	1920	7.0	2343.0	2315	28.0	DL	1345	N3771K	JFK	PHX	308.0	2153	19	20	2013-01-25T00:00:00Z

DIKW ACADEMY

COMMONALITIES

You may have noticed that all these functions are very similar:

- A data frame is piped to the function as input
- The arguments describe what to do with it, and you can refer to columns in the data frame directly.
- The result is a new data frame

Together these properties make it easy to chain together multiple simple steps to achieve a complex result.