Pandas

Pandas is a python library for data analysis. It adopts the Matlab and R philosophy for organizing 2-dimensional data in a special structure called data frame. In bioinformatics pandas are useful for doing tasks that are usually done with Microsoft Excel. The advantages of pandas are:

- It is fast. Part of it is implemented in C and has very good performance for tables that have up to millions of rows.
- Provides an interface that simulates databases. This way we can write short expressions that make complex processes.
    - In terms of Computer Science, we say that pandas is for " declarative language ", in contrast to the classic python that offers control flow programming.
- Supported by third-party libraries for visualization, Machine Learning (eg sci-kit) and statistics (eg statmodels).
- Provides its own methods for fast plotting and statistical analysis
- Easy and fast input / output in various formats (excel included)

We usually import pandas as follows:

In [23]:

```
import pandas as pd
```

If it is not installed then you can install it as follows:

```
pip install pandas
```

Caution. The `pip` must be in the same location as the python.

Let's import a CSV file:

In [24]:

```
hs = pd.read_csv('https://ftp.ncbi.nlm.nih.gov/gene/DATA/GENE_INFO/Mammali
```

`sep='\t'` means that the columns are separated by tabs.

The first lines:

In [10]:

```
hs.head()
```

Out[10]:

	#tax_id	GeneID	Symbol	LocusTag	Synonyms	
0	9606	1	A1BG	-	A1B|ABG|GAB|HYST2477	MIM:138670|HGNC
1	9606	2	A2M	-	A2MD|CPAMD5|FWP007|S863-7	MIM:103950|HGNC
2	9606	3	A2MP1	-	A2MP	HGNC:

	#tax_id	GeneID	Symbol	LocusTag	Synonyms	
3	9606	9	NAT1	-	AAC1|MNAT|NAT-1|NATI	MIM:108345|HGNC:HG

The last lines:

In [11]:

```
hs.tail()
```

Out[11]:

	#tax_id	GeneID	Symbol	LocusTag	Synonyms	dbXrefs	chromosome	map_locatic
62011	741158	8923215	trnD	-	-	-	MT	
62012	741158	8923216	trnP	-	-	-	MT	
62013	741158	8923217	trnA	-	-	-	MT	
62014	741158	8923218	COX1	-	-	-	MT	
62015	741158	8923219	16S rRNA	-	-	-	MT	

The same can be done with:

In [13]:

```
hs[:5] # The first 5 lines
```

Out[13]:

	#tax_id	GeneID	Symbol	LocusTag	Synonyms	
0	9606	1	A1BG	-	A1B|ABG|GAB|HYST2477	MIM:138670|HGNC
1	9606	2	A2M	-	A2MD|CPAMD5|FWP007|S863-7	MIM:103950|HGNC
2	9606	3	A2MP1	-	A2MP	HGNC:
3	9606	9	NAT1	-	AAC1|MNAT|NAT-1|NATI	MIM:108345|HGNC:HG
4	9606	10	NAT2	-	AAC2|NAT-2|PNAT	MIM:612182|HGNC:HG

In [15]:

```
hs[-5:]
```

Out[15]:

	#tax_id	GeneID	Symbol	LocusTag	Synonyms	dbXrefs	chromosome	map_locatic
62011	741158	8923215	trnD	-	-	-	MT	
62012	741158	8923216	trnP	-	-	-	MT	
62013	741158	8923217	trnA	-	-	-	MT	
62014	741158	8923218	COX1	-	-	-	MT	

	#tax_id	GeneID	Symbol	LocusTag	Synonyms	dbXrefs	chromosome	map_locatic
62015	741158	8923219	16S				MT	

All columns:

```
In [16]: hs.columns
```

```
Out[16]: Index(['#tax_id', 'GeneID', 'Symbol', 'LocusTag', 'Synonyms', 'dbXrefs',
               'chromosome', 'map_location', 'description', 'type_of_gene',
               'Symbol_from_nomenclature_authority',
               'Full_name_from_nomenclature_authority', 'Nomenclature_status',
               'Other_designations', 'Modification_date', 'Feature_type'],
              dtype='object')
```

Or in list:

```
In [19]: list(hs.columns.values)
```

```
Out[19]: ['#tax_id',
          'GeneID',
          'Symbol',
          'LocusTag',
          'Synonyms',
          'dbXrefs',
          'chromosome',
          'map_location',
          'description',
          'type_of_gene',
          'Symbol_from_nomenclature_authority',
          'Full_name_from_nomenclature_authority',
          'Nomenclature_status',
          'Other_designations',
          'Modification_date',
          'Feature_type']
```

Nice. Let's just take three columns:

```
In [22]: hs[['Symbol', 'chromosome', 'type_of_gene']][:5]
```

Out[22]:

	Symbol	chromosome	type_of_gene
0	A1BG	19	protein-coding
1	A2M	12	protein-coding
2	A2MP1	12	pseudo
3	NAT1	8	protein-coding
4	NAT2	8	protein-coding

Caution! The above is equivalent to:

```
In [23]: hs[:5][['Symbol', 'chromosome', 'type_of_gene']]
```

Out[23]:

	Symbol	chromosome	type_of_gene
0	A1BG	19	protein-coding
1	A2M	12	protein-coding
2	A2MP1	12	pseudo
3	NAT1	8	protein-coding

Symbol **chromosome** **type_of_gene**

How many rows and how many columns does the Data Frame have?

In [25]:
```
hs.shape
```

Out[25]:
```
(62016, 16)
```

Indexes

A dataframe must have an index. By default the index is the serial number of the rows.

You can directly ask for a specific value of an index

In [43]:
```
hs.iloc[10]
```

Out[43]:
```
#tax_id
9606
GeneID
16
Symbol
AARS1
LocusTag
-
Synonyms                                                           AARS|CMT
2N|DEE29|EIEE29
dbXrefs                                       MIM:601065|HGNC:HGNC:20|Ensembl:
ENSG00000090861
chromosome
16
map_location
16q22.1
description                                                        alanyl-tR
NA synthetase 1
type_of_gene
protein-coding
Symbol_from_nomenclature_authority
AARS1
Full_name_from_nomenclature_authority                              alanyl-tR
NA synthetase 1
Nomenclature_status
O
Other_designations                          alanine--tRNA ligase, cytoplasmic|
alaRS|alanin...
Modification_date
20210404
Feature_type
-
New_date                                                                2021
-04-04 00:00:00
Name: 10, dtype: object
```

In [286…
```
hs.iloc[:2]
```

Out[286…

	#tax_id	GeneID	Symbol	LocusTag	Synonyms	
0	9606	1	A1BG	-	A1B|ABG|GAB|HYST2477	MIM:138670|HGNC:HC

#tax id GeneID Svmbol LocusTaa Svnonvms

Rows 100, 200 and 400:

In [287…

```
hs.iloc[[100, 200, 400]]
```

Out[287…

	#tax_id	GeneID	Symbol	LocusTag	Synonyms	
100	9606	119	ADD2	-	ADDB	MIM:102681|HGNC:HGNC:244|Ense
200	9606	240	ALOX5	-	5-LO|5-LOX|5LPG|LOG5	MIM:152390|HGNC:HGNC:435|Ense
400	9606	479	ATP12A	-	ATP1AL1|H-K-ATPase|HK	MIM:182360|HGNC:HGNC:13816|E

Filtering

By filtering we can request for a specific subset of our data that have a ginven property. A similar term is query (or querying).

Pandas and numpy (we will see in the next lecture) have a common mechanism for filtering. The interesting thing is that this mechanism is, influenced by R.

For starters we need to make a list that is the same size as the number of rows in the DataFrame. This list will only have True or False values. Let's see this in practice:

Suppose we only want the odd lines of a dataframe. Initially, we make a list where the odd positions contain the `True` value and the even ones the `False` value:

In [25]:

```
rows = hs.shape[0]
l = [x%2==1 for x in range(rows)]
```

Now `l` can act as a filter if we pass it to a Data Frame:

In [26]:

```
hs_filtered = hs[l]
```

In [28]:

```
hs_filtered.shape # Half lines
```

Out[28]:

```
(31397, 16)
```

In [29]:

```
hs_filtered.head()
```

Out[29]:

	#tax_id	GeneID	Symbol	LocusTag	Synonyms	
1	9606	2	A2M	-	A2MD|CPAMD5|FWP007|S863-7	MIM:103950|HGNC
3	9606	9	NAT1	-	AAC1|MNAT|NAT-1|NATI	MIM:108345|HGNC:HG
5	9606	11	NATP	-	AACP|NATP1	

	#tax_id	GeneID	Symbol	LocusTag	Synonyms	
7	9606	13	AADAC	-	CES5A1|DAC	MIM:600338|HGNC

We notice that it only has odd indexes.

OK, but how does this help me filter? We can simply do a logical operation with one column and the result is a list of logical values that can be used as a filter! For example. All genes on chromosome 8:

In [35]:
```python
filter_chr_8 = hs['chromosome'] == '8'
```

In [37]:
```python
filter_chr_8[:10] # Print first 10
```

Out[37]:
```
0    False
1    False
2    False
3     True
4     True
5     True
6    False
7    False
8    False
9    False
Name: chromosome, dtype: bool
```

We confirm with:

In [38]:
```python
hs['chromosome'][:10]
```

Out[38]:
```
0    19
1    12
2    12
3     8
4     8
5     8
6    14
7     3
8     2
9    17
Name: chromosome, dtype: object
```

we can now apply this filter:

In [41]:
```python
hs[filter_chr_8][:10] # First 10 genes in chromosome 8
```

Out[41]:

	#tax_id	GeneID	Symbol	LocusTag	Synonyms	
3	9606	9	NAT1	-	AAC1|MNAT|NAT-1|NATI	MIM:108345|HGNC:H
4	9606	10	NAT2	-	AAC2|NAT-2|PNAT	MIM:612182|HGNC:H
5	9606	11	NATP	-	AACP|NATP1	
54	9606	66	ACTBP6	-	H8-PSI-BETA-AC3	

	#tax_id	GeneID	Symbol	LocusTag	Synonyms	
95	9606	114	ADCY8	-	AC8|ADCY3|HBAC1	MIM:103070|HGNC:H
125	9606	148	ADRA1A	-	ADRA1C|ADRA1L1|ALPHA1AAR	MIM:104221|HGNC:H
131	9606	155	ADRB3	-	BETA3AR	MIM:109691|HGNC:H
237	9606	284	ANGPT1	-	AGP1|AGPT|ANG1	MIM:601667|HGNC:H
238	9606	285	ANGPT2	-	AGPT2|ANG2	MIM:601922|HGNC:H
239	9606	286	ANK1	-	ANK|SPH1|SPH2	MIM:612641|HGNC:H

If we replace `filter_chr_8` with `hs['chromosome&'] == '8'` (see also cell 35) we will have:

In [42]:

```
hs[hs['chromosome'] == '8'][:10]
```

Out[42]:

	#tax_id	GeneID	Symbol	LocusTag	Synonyms	
3	9606	9	NAT1	-	AAC1|MNAT|NAT-1|NATI	MIM:108345|HGNC:H
4	9606	10	NAT2	-	AAC2|NAT-2|PNAT	MIM:612182|HGNC:HG
5	9606	11	NATP	-	AACP|NATP1	
54	9606	66	ACTBP6	-	H8-PSI-BETA-AC3	
95	9606	114	ADCY8	-	AC8|ADCY3|HBAC1	MIM:103070|HGNC:H
125	9606	148	ADRA1A	-	ADRA1C|ADRA1L1|ALPHA1AAR	MIM:104221|HGNC:H
131	9606	155	ADRB3	-	BETA3AR	MIM:109691|HGNC:H
237	9606	284	ANGPT1	-	AGP1|AGPT|ANG1	MIM:601667|HGNC:H
238	9606	285	ANGPT2	-	AGPT2|ANG2	MIM:601922|HGNC:H
239	9606	286	ANK1	-	ANK|SPH1|SPH2	MIM:612641|HGNC:H

Αυτό:

`hs[hs['chromosome'] == '8']`

Αξίζει να το ξαναδούμε. Καταρχήν μπορεί να "ξενίζει" η διπλή αναφορά στο hs και η διπλή χρήση του `..[..[` . Αλλά αν προσέξουμε μας δίνει ένα πολύ δυνατό και εκφραστικό εργαλείο να ορίζουμε φίλτρα δεδομένων. Επίσης δεν πρέπει να ξεχνάμε ότι ακριβώς τον ίδιο μηχανισμό χρησιμοποιεί η R και η Matlab. Αφού λοιπόν τα φίλτρα είναι πίνακες από λογικές τιμές, μπορούμε να κάνουμε λογικές πράξεις!

Τα γονίδια που ανήκουν στο χρωμόσωμα 8 και είναι pseudo genes:

This:

`hs[hs['chromosome'] == '8']`

is worth examining again. At a first glance, this double reference to `hs` and the dual use of brackets (`..[..[`) might be odd. But if we pay attention it gives us a very powerful and expressive tool to define data filters. We must also not forget that R and Matlab use exactly the same mechanism. So since filters are tables of boolean values, we can do logical operations!

The genes that belong to chromosome 8 and are pseudo genes:

In [45]:
```
hs[ (hs['chromosome'] == '8') & (hs['type_of_gene'] == 'pseudo') ]
```

Out[45]:

	#tax_id	GeneID	Symbol	LocusTag	Synonyms	
5	9606	11	NATP	-	AACP|NATP1	
54	9606	66	ACTBP6	-	H8-PSI-BETA-AC3	
568	9606	693	BTF3P1	-	HUMBTFA	
1289	9606	1587	ADAM3A	-	ADAM3|CYRN1|tMDCI	HGNC:HGNC:2
2019	9606	2503	FTH1P11	-	FTHL11	
...	...	...	...	...	...	
58933	9606	112268397	LOC112268397	-	-	
58934	9606	112268399	LOC112268399	-	-	
59895	9606	112935968	PTMAP15	-	-	
60739	9606	115482726	H2AZP7	-	-	HGNC:HGNC:544
61434	9606	117751737	HIKESHIP3	-	-	HGNC:HGNC:549

690 rows × 16 columns

ΠΡΟΣΟΧΗ! Οι παρενθέσεις είναι υποχρεωτικές:

Για μια στιγμή, γιατί δεν χρησιμοποιήσαμε τον αγαπημένο μας τελεστή and και χρησημοποιήσαμε αυτό το `&` ; Θυμόμαστε ότι το αποτέλεσμα της and είναι **πάντα** (ακόμα και στη pandas) λογικές τιμές (δηλαδή είτε True είτε False). Ναι αλλά εμείς δεν θέλουμε True ή False θέλουμε λίστες από True ή False. Για να κάνουμε αυτή τη διάκριση χρησιμοποιούμε το `&` .

Οι τελεστές που μπορούμε να χρησιμοποιήσουμε είναι:

- `&` --> and
- `|` --> or
- `~` --> not

Για παράδειγμα: Όλα τα γονίδια που ΔΕΝ ανήκουν στο χρωμόσωμα 8 ή 9 και είναι pseudo genes (τυπώνουμε τα πρώτα 5):

CAUTION! Brackets are required:

```
hs[ (hs['chromosome'] == '8') & (hs['type_of_gene'] == 'pseudo') ]
    ^                       ^   ^                              ^
```

Wait, why didn't we use our favorite operator and use this `&` instead? Remember that the result of `and` , `or` and `not` are **always** (even in pandas) boolean values (ie either `True` or `False`). Yet we do not want this operator to return `True` or `False` we want it to return lists with `True` or `False` values. To make this distinction we define another set of operators:

The operators we can use are:

- `&` --> and
- `|` --> or
- `~` --> not

For example: All genes that do NOT belong to chromosome 8 or 9 and are pseudo genes (print the first 5):

In [54]:
```
hs[~((hs['chromosome'] == 8) | (hs['chromosome'] == 9)) & (hs['type_of_gene
```

Out[54]:

	#tax_id	GeneID	Symbol	LocusTag	Synonyms	dbXr
2	9606	3	A2MP1	-	A2MP	HGNC:HGNC:8|Ensembl:ENSG0000025
5	9606	11	NATP	-	AACP|NATP1	HGNC:HGN
51	9606	62	ACTBP2	-	-	HGNC:HGNC:
52	9606	63	ACTBP3	-	-	HGNC:HGNC:

	#tax_id	GeneID	Symbol	LocusTag	Synonyms	dbXr

There are many pandas functions that return filters (lists with `True` / `False` values). For example:

Check if the values of a column contain a given string:

In [19]:
```
hs[hs['description'].str.contains('membrane')][:5]
```

Out[19]:

	#tax_id	GeneID	Symbol	LocusTag	Synonyms	
243	9606	290	ANPEP	-	APN|CD13|GP150|LAP1|P150|PEPN	MIM:15
411	9606	490	ATP2B1	-	PMCA1|PMCA1kb	MIM:10
412	9606	491	ATP2B2	-	PMCA2|PMCA2a|PMCA2i	MIM:10
413	9606	492	ATP2B3	-	CFAP39|CLA2|OPCA|PMCA3|PMCA3a|SCAX1	MIM:30
414	9606	493	ATP2B4	-	ATP2B2|MXRA1|PMCA4|PMCA4b|PMCA4x	MIM:10

Same as before but case insensitive:

In [18]:
```
hs[hs['description'].str.contains('membrane', case=False)][:5]
```

Out[18]:

	#tax_id	GeneID	Symbol	LocusTag	Synonyms	
243	9606	290	ANPEP	-	APN|CD13|GP150|LAP1|P150|PEPN	MIM:15
411	9606	490	ATP2B1	-	PMCA1|PMCA1kb	MIM:10
412	9606	491	ATP2B2	-	PMCA2|PMCA2a|PMCA2i	MIM:10
413	9606	492	ATP2B3	-	CFAP39|CLA2|OPCA|PMCA3|PMCA3a|SCAX1	MIM:30
414	9606	493	ATP2B4	-	ATP2B2|MXRA1|PMCA4|PMCA4b|PMCA4x	MIM:10

Series

Μία στήλη σε pandas ονομάζεται Series. Με τη describe μπορούμε να έχουμε μία καλή εικόνα των τιμών που περιέχει:

Series

A column in pandas is called a Series. With the `describe` function we can get a good overview of its values:

```
In [9]:  hs['type_of_gene'].describe()
```

```
Out[9]: count                62016
        unique                  11
        top         protein-coding
        freq                 19696
        Name: type_of_gene, dtype: object
```

Some of the operations that we can do in a column are:

Find all its unique values:

```
In [13]:  hs['type_of_gene'].unique()
```

```
Out[13]: array(['protein-coding', 'pseudo', 'other', 'unknown', 'ncRNA', 'tRNA',
                'rRNA', 'scRNA', 'snoRNA', 'snRNA', 'biological-region'],
               dtype=object)
```

Find the number of lines that each unique values has:

```
In [16]:  hs['type_of_gene'].value_counts()
```

```
Out[16]: protein-coding       19696
         ncRNA                17513
         pseudo               16556
         biological-region     4754
         unknown               1383
         other                  840
         tRNA                   595
         snoRNA                 541
         snRNA                   71
         rRNA                    63
         scRNA                    4
         Name: type_of_gene, dtype: int64
```

Adding columns

We can create a new Series from another by using the `apply` function. `apply` takes a function and applies it to all lines returning a new Series. You can use this Series as a new column.

For example we notice that the `dbXrefs` column contains many IDs to other databases. We can have one of these codes in a new column:

In [150…

```python
import re
def create_ensembl(row):
    m = re.search(r'Ensembl:(ENSG\d+)', row['dbXrefs'])
    if not m:
        return pd.NA

    return m.group(1)

hs['ENSEMBL'] = hs.apply(create_ensembl, axis=1)
```

In [17]:

```python
hs[:5]
```

Out[17]:

	#tax_id	GeneID	Symbol	LocusTag	Synonyms	
0	9606	1	A1BG	-	A1B|ABG|GAB|HYST2477	MIM:138670|HGNC
1	9606	2	A2M	-	A2MD|CPAMD5|FWP007|S863-7	MIM:103950|HGNC
2	9606	3	A2MP1	-	A2MP	HGNC:
3	9606	9	NAT1	-	AAC1|MNAT|NAT-1|NATI	MIM:108345|HGNC:HG
4	9606	10	NAT2	-	AAC2|NAT-2|PNAT	MIM:612182|HGNC:HG

Notice that the last column contains the ENSEMBL ID.

We can add a new column with the map:

In [284…

```python
hs['is_pseudo'] = hs['type_of_gene'].map(lambda x : x=='pseudo')
```

In [285…

```python
hs[:5]
```

Out[285…

	#tax_id	GeneID	Symbol	LocusTag	Synonyms	
0	9606	1	A1BG	-	A1B|ABG|GAB|HYST2477	MIM:138670|HGNC
1	9606	2	A2M	-	A2MD|CPAMD5|FWP007|S863-7	MIM:103950|HGNC
2	9606	3	A2MP1	-	A2MP	HGNC:
3	9606	9	Mitsos	-	AAC1|MNAT|NAT-1|NATI	MIM:108345|HGNC:HG

#tax_id	GeneID	Symbol	LocusTag	Synonyms

NA = Not Available

What is this `pd.NA` ? It is the pandas constant used when a value is not.. available. Pandas has a large collection of functions to manage this value:

For example check is a value in a series is `pd.NA` or not.

In [151…

```
hs['ENSEMBL'].isna().value_counts()
```

Out[151…

```
False    35145
True     26871
Name: ENSEMBL, dtype: int64
```

Delete the rows where the ENSEMBL column is `pd.NA` :

In [152…

```
hs.dropna(subset=['ENSEMBL'])[:5]
```

Out[152…

	#tax_id	GeneID	Symbol	LocusTag	Synonyms	
0	9606	1	A1BG	-	A1B|ABG|GAB|HYST2477	MIM:138670|HGNC
1	9606	2	A2M	-	A2MD|CPAMD5|FWP007|S863-7	MIM:103950|HGNC
2	9606	3	A2MP1	-	A2MP	HGNC:
3	9606	9	Mitsos	-	AAC1|MNAT|NAT-1|NATI	MIM:108345|HGNC:HG
4	9606	10	NAT2	-	AAC2|NAT-2|PNAT	MIM:612182|HGNC:HG

Replace `pd.NA` with another value:

In [153…

```
hs['ENSEMBL'].fillna('Does not exist')
```

Out[153…

```
0        ENSG00000121410
1        ENSG00000175899
2        ENSG00000256069
3        ENSG00000171428
4        ENSG00000156006
              ...
62011     Does not exist
62012     Does not exist
62013     Does not exist
62014     Does not exist
62015     Does not exist
Name: ENSEMBL, Length: 62016, dtype: object
```

Changing a value

To change a value we need to know the row and the column. To be more precise we need to know the index of the line.

In [155…

```
hs.at[3, 'Symbol'] = 'Mitsos'
```

In [156…

```
hs[:5]
```

Out[156…

	#tax_id	GeneID	Symbol	LocusTag	Synonyms	
0	9606	1	A1BG	-	A1B|ABG|GAB|HYST2477	MIM:138670|HGNC
1	9606	2	A2M	-	A2MD|CPAMD5|FWP007|S863-7	MIM:103950|HGNC
2	9606	3	A2MP1	-	A2MP	HGNC:
3	9606	9	Mitsos	-	AAC1|MNAT|NAT-1|NATI	MIM:108345|HGNC:HG
4	9606	10	NAT2	-	AAC2|NAT-2|PNAT	MIM:612182|HGNC:HG

Μετονομασία στήλης

Rename column

In [157…

```
hs = hs.rename(columns={'ENSEMBL': 'ENSEMBL genes'})
```

Διαγραφή στήλης:

Delete column:

In [159…

```
hs = hs.drop('ENSEMBL genes', axis=1)
```

Ημερομηνίες

Πρατηρούμε ότι η στήλη: `hs['Modification_date']` έχει ημερομηνίες αλλά η pandas τα βλέπει σαν string. Μπορούμε να αλλάξουμε τον τύπο μίας στήλης και να δηλώσουμε ότι περιέχει ημερομηνίες.

Για να το κάνουμε αυτό πρέπει να δηλώσουμε το format της ημερομηνίας: https://docs.python.org/3/library/datetime.html#strftime-and-strptime-behavior

Dates

Note that the column: `hs[&#39;Modification_date&#39;]` has dates but pandas sees them as a string. We can change the type of a column and state that it contains dates.

To do this we must declare the format of the date: https://docs.python.org/3/library/datetime.html#strftime-and-strptime-behavior

In [34]:

```
hs['New_date'] = pd.to_datetime(hs['Modification_date'], format='%Y%m%d')
```

Προσέξτε τη διαφορά

Notice the difference

In [35]:

```
hs[['New_date', 'Modification_date']][:10]
```

Out[35]:

	New_date	Modification_date
0	2021-03-02	20210302
1	2021-04-04	20210404
2	2021-03-02	20210302
3	2021-03-02	20210302
4	2021-03-22	20210322
5	2021-03-02	20210302
6	2021-03-07	20210307
7	2021-03-02	20210302
8	2021-03-02	20210302
9	2021-03-02	20210302

Τώρα μπορούμε να κάνουμε ταξινόμηση, filtering, κτλ με βάση την ημερομηνία. Ποιο είναι το γονίδιο το οποίο ανανεώθηκε πιο παλιά:

Now we can do sorting, filtering, etc by date. What is the oldest renewed gene:

In [45]:

```
hs.iloc[hs['New_date'].idxmin()]
```

Out[45]:

```
#tax_id                                                              9606
GeneID                                                               7909
Symbol                                                               HEMC
LocusTag                                                                -
Synonyms                                                              HCI
dbXrefs                                                        MIM:602089
chromosome                                                              -
map_location                                                            -
description                             hemangioma, capillary, hereditary
type_of_gene                                                      unknown
Symbol_from_nomenclature_authority                                      -
Full_name_from_nomenclature_authority                                   -
Nomenclature_status                                                     -
Other_designations                                                      -
Modification_date                                                20170402
Feature_type                                                            -
New_date                                                2017-04-02 00:00:00
Name: 6234, dtype: object
```

Ταξινόμηση

Χρησιμοποιούμε τη sort_values:

Classification

We use sort_values:

In [50]:
```
hs.sort_values('description')[:5]
```

Out[50]:

	#tax_id	GeneID	Symbol	LocusTag	Synonyms	
57645	9606	110599572	LOC110599572	-	-	
2120	9606	2632	GBE1	-	APBD|GBE|GSD4	M
8277	9606	10554	AGPAT1	-	1-AGPAT1|G15|LPAAT-alpha|LPAATA	I
8278	9606	10555	AGPAT2	-	1-AGPAT2|BSCL|BSCL1|LPAAB|LPAAT-beta	
13332	9606	56894	AGPAT3	-	1-AGPAT 3|LPAAT-GAMMA1|LPAAT3	

Από python --> pandas

From python -> pandas

Αν έχετε μία δομή σε python μπορείτε να τη μετασχηματίσετε ώστε να μπορεί να μπει σαν είσοδο στη DataFrame και να επιστρέψει ένα DataFrame.

Η DataFrame υποστηρίζει δύο διαφορετικές δομές:

Δομή 1η: Λίστα από dictionaries. Κάθε κλειδί στο dictionary είναι το όνομα μίας στήλης:

If you have a structure in python you can transform it so that it can enter the DataFrame and return a DataFrame.

DataFrame supports two different structures:

Structure 1: List of dictionaries. Each key in the dictionary is the name of a column:

In [52]:
```
l = [
    {'col_1': 1, 'col_2': 1, 'col_3': 'test_1'},
    {'col_1': 2, 'col_2': 4, 'col_3': 'test_2'},
    {'col_1': 3, 'col_2': 5, 'col_3': 'test_5'},
    {'col_1': 1, 'col_2': 1, 'col_3': 'test_2'},
    {'col_1': 1, 'col_2': 2, 'col_3': 'test_4'},
]

df = pd.DataFrame(l)
df
```

Out[52]:

	col_1	col_2	col_3
0	1	1	test_1
1	2	4	test_2

	col_1	col_2	col_3
2	3	5	test_5
3	1	1	test_2

Δομή 2η: Dictionaries με λίστες:

Structure 2 : Dictionaries with lists:

```
In [54]: l = {
             'col_1': [1,2,3,1,1],
             'col_2': [1,4,5,1,2],
             'col_3': ['test_1', 'test_2', 'test_5', 'test_2', 'test_4'],
         }
         df = pd.DataFrame(l)
         df
```

Out[54]:

	col_1	col_2	col_3
0	1	1	test_1
1	2	4	test_2
2	3	5	test_5
3	1	1	test_2
4	1	2	test_4

Από pandas --> python

Με τη μέθοδο `to_dict` μπορείτε να μετατρέψετε σε dictionary ή λίστα:

From pandas -> python

With the `to_dict` method you can convert to dictionary or list:

```
In [57]: df.to_dict('records') # Λίστα από dictionaries
```

```
Out[57]: [{'col_1': 1, 'col_2': 1, 'col_3': 'test_1'},
          {'col_1': 2, 'col_2': 4, 'col_3': 'test_2'},
          {'col_1': 3, 'col_2': 5, 'col_3': 'test_5'},
          {'col_1': 1, 'col_2': 1, 'col_3': 'test_2'},
          {'col_1': 1, 'col_2': 2, 'col_3': 'test_4'}]
```

```
In [59]: df.to_dict('index') # Dictionaies από dictionaries. Τα κλειδιά είναι τα in
```

```
Out[59]: {0: {'col_1': 1, 'col_2': 1, 'col_3': 'test_1'},
          1: {'col_1': 2, 'col_2': 4, 'col_3': 'test_2'},
          2: {'col_1': 3, 'col_2': 5, 'col_3': 'test_5'},
          3: {'col_1': 1, 'col_2': 1, 'col_3': 'test_2'},
          4: {'col_1': 1, 'col_2': 2, 'col_3': 'test_4'}}
```

```
In [62]: df.to_dict('dict') # Dictionary από dictionaries. Τα κλειδιά είναι οι στήλ
```

```
Out[62]: {'col_1': {0: 1, 1: 2, 2: 3, 3: 1, 4: 1},
          'col_2': {0: 1, 1: 4, 2: 5, 3: 1, 4: 2},
          'col_3': {0: 'test_1', 1: 'test_2', 2: 'test_5', 3: 'test_2', 4: 'test_4
         '}}
```

```
In [63]: df.to_dict('list') # Dictionary από λίστες. Τα κλειδιά είναι οι στήλες
```

Out[63]:

```
{'col_1': [1, 2, 3, 1, 1],
 'col_2': [1, 4, 5, 1, 2],
 'col_3': ['test_1', 'test_2', 'test_5', 'test_2', 'test_4']}
```

In [64]:

```
pd.DataFrame({0: {'col_1': 1, 'col_2': 1, 'col_3': 'test_1'},
 1: {'col_1': 2, 'col_2': 4, 'col_3': 'test_2'},
 2: {'col_1': 3, 'col_2': 5, 'col_3': 'test_5'},
 3: {'col_1': 1, 'col_2': 1, 'col_3': 'test_2'},
 4: {'col_1': 1, 'col_2': 2, 'col_3': 'test_4'}})
```

Out[64]:

	0	1	2	3	4
col_1	1	2	3	1	1
col_2	1	4	5	1	2
col_3	test_1	test_2	test_5	test_2	test_4

Σώζοντας ένα pandas DataFrame

Saving a DataFrame pandas

Τα pandas είναι μία "εξωστρεφής" βιβλιοθήκη. Αυτό σημαίνει ότι μπορεί να σώζει και να φορτώνει από/σε πολλά φορμάτ. Το πιο κοινό είναι το csv:

Pandas are an "extroverted" library. This means it can save and load from / to many formats. The most common is csv:

In [65]:

```
df.to_csv('test.csv')
```

In [67]:

```
!cat test.csv # Για windows: !type test.csv
```

```
,col_1,col_2,col_3
0,1,1,test_1
1,2,4,test_2
2,3,5,test_5
3,1,1,test_2
4,1,2,test_4
```

In [69]:

```
df.to_csv('test.csv', index=None) # Σώζουμε χωρίς το index
```

In [70]:

```
!cat test.csv # Για windows: !type test.csv
```

```
col_1,col_2,col_3
1,1,test_1
2,4,test_2
3,5,test_5
1,1,test_2
1,2,test_4
```

Μπορούμε να σώσουμε ένα αρχείο σε φορμάτ excel:

We can save a file in excel format:

In [71]:

```
df.to_excel('test.xlsx')
```

Και να διαβάσουμε από excel:

And read from excel:

In [72]:

```
df2 = pd.read_excel('test.xlsx')
```

In [74]:

```
df2
```

Out[74]:

	Unnamed: 0	col_1	col_2	col_3
0	0	1	1	test_1
1	1	2	4	test_2
2	2	3	5	test_5
3	3	1	1	test_2
4	4	1	2	test_4

Διαβάστε εδώ: https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html για τα διαφορετικά φορμάτ που μπορούμε να διαβάσουμε και να γράψουμε.

Read here: https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html for the different formats we can read and write.

Iteration

Iteration

Αν και σπάνια το χρειαζόμαστε (..και προσπαθούμε να αντίσταθούμε στον πειρασμό να το χρησιμοποιήσουμε) μπορούμε να κάνουμε iterate (επανάληψη) σε κάθε γραμμή του DataFrame. Αν και υπάρχουν πολλοί τρόποι για να το κάνουμε αυτό, εδω δείχνουμε το itertuples:

Although we rarely need it (..and try to resist the temptation to use it) we can iterate on each line of the DataFrame. Although there are many ways to do this, here we show itertuples:

In [183…

```
for x in df2.itertuples():
    print ('Index: ', x.Index, 'col_2', x.col_2)
```

```
Index:  0 col_2 1
Index:  1 col_2 4
Index:  2 col_2 5
Index:  3 col_2 1
Index:  4 col_2 2
```

Grouping

To grouping είναι μία από τις πιο βασικές λειτουργίες των βιβλιοθηκών που χειρίζονται 2-διάστατα δεδομένα. Στην ουσία με το grouping χωρίζουμε τις γραμμές σε groups. Από κάθε group παίρνουμε κάποιες στήλες και σε όλες τις τιμές του group της κάθε στήλης εφαρμόζουμε μία συνάρτηση. Με αυτόν τον τρόπο μπορούμε να κάνουμε πολύ χρήσιμες ερωτήσεις όπως: "για κάθε χρωμόσωμα ποιο είναι το μεγαλύτερο γονίδιο;", "για κάθε νομό ποια είναι η μεγαλύτερη πόλη", "ποιος είναι ο μέσος όρος των γονιδίων που έχουν συσχετιστεί με τον καρκίνο για κάθε χρωμόσωμα;" κτλ..

Η pandas έχει μία βασική δομή για το grouping:

```
df.groupby( [ ... ] )[ [...] ].aggregate( ... )
                |          |                   |
                |          |                   > Συνάρτηση που θα
```

```
εφαρμόσουμε σε κάθε group-αρισμένη στήλη
                |         > Στήλες που θα group-αριστούύν
                > Στήλες που περιέχουν τις τιμές με βάση τις οποίες
θα γίνει το  grouping.
                > Κάθε διαφορετική τιμή τους θα είναι και ένα group
```

Ας δούμε μερικά παραδείγματα. Φτιάχνουμε ένα random dataframe με 3 στήλες 20 γραμμές και τιμές από 1-4:

Grouping

Grouping is one of the most basic functions of libraries that handle 2-dimensional data. In essence, with grouping we divide the lines into groups. We take some columns from each group and we apply a function to all the group values of each column. In this way we can ask very useful questions such as: "for each chromosome what is the largest gene?", "For each county what is the largest city", "what is the average of the genes associated with cancer for each chromosome? " etc ..

Pandas have a basic structure for grouping:

Let's look at some examples. We make a random dataframe with 3 columns 20 rows and values from 1-4:

In [78]:

```
import random

d = {
    'col_1': [random.randint(1,4) for x in range(20)],
    'col_2': [random.randint(1,4) for x in range(20)],
    'col_3': [random.randint(1,4) for x in range(20)],
}
df = pd.DataFrame(d)
df
```

Out[78]:

	col_1	col_2	col_3
0	4	4	3
1	1	4	3
2	4	4	1
3	2	2	1
4	1	1	3
5	1	4	2
6	1	1	2
7	2	3	3
8	4	3	2
9	4	4	2
10	4	4	1
11	2	3	3
12	2	4	2
13	1	2	4
14	4	3	2

```
    col_1  col_2  col_3
15      2      4      3
16      3      4      3
17      1      2      4
18      1      3      3
```

Για κάθε διαφορετική τιμή της στήλης col_1, ποια είναι η μικρότερη τιμή της col_2;

For each different value of the col_1 column, what is the smallest value of col_2?

```
In [83]: df.groupby(['col_1'])[['col_2']].aggregate('min')
```

```
Out[83]:
         col_2
col_1
    1        1
    2        2
    3        4
    4        3
```

Για κάθε διαφορετική τιμή της στήλης `col_1` , ποια είναι η μικρότερη και μεγαλύτερη τιμή της `col_2` ;

For each different value of the `col_1` column, what is the smallest and largest value of `col_2` ?

```
In [85]: df.groupby(['col_1'])[['col_2']].aggregate(['min', 'max'])
```

```
Out[85]:
            col_2
          min  max
col_1
    1       1    4
    2       2    4
    3       4    4
    4       3    4
```

Για κάθε διαφορετική τιμή της στήλης `col_1` , ποια είναι η μικρότερη τιμή της στήλης `col_2` και μεγαλύτερη τιμή της στήλης `col_3` ;

For each different value in column `col_1` , what is the smallest value in column `col_2` and the largest value in column `col_3` ?

```
In [88]: df.groupby(['col_1'])[['col_2', 'col_3']].aggregate({'col_2': 'min', 'col_
```

```
Out[88]:
         col_2  col_3
col_1
    1        1      4
    2        2      3
```

	col_2	col_3
col_1		

Για κάθε διαφορετική τιμή της στήλης `col_1` , ποια είναι η μικρότερη και μεγαλύτερη τιμή της στήλης `col_2` και μικρότερη και μεγαλύτερη τιμή της στήλης `col_3` ;

For each different value of the `col_1` column, what is the smallest and largest value of the `col_2` column and the smallest and largest value of the `col_3` column?

In [89]:

```
df.groupby(['col_1'])[['col_2', 'col_3']].aggregate( ['min', 'max'] )
```

Out[89]:

	col_2		col_3	
	min	max	min	max
col_1				
1	1	4	1	4
2	2	4	1	3
3	4	4	3	3
4	3	4	1	3

Ένα group μπορεί να έχει παραπάνω από μία στήλες. Σε αυτή τη περίπτωση κάθε group περιέχει όλες τις διαφορετικές τιμές που προκύπτουν από τους συνδυασμούς των διαφορετικών τιμών των 2 (ή παραπάνω) στηλών.

Για όλες τις διαφορετικές τιμές της στήλης `col_1` και `col_2` ποια είναι η μικρότερη τιμή της στήλης `col_3` ;

A group can have more than one column. In this case each group contains all the different values resulting from the combinations of the different values of the 2 (or more) columns.

For all the different values in column `col_1` and `col_2` what is the smallest value in column `col_3` ?

In [91]:

```
df.groupby(['col_1', 'col_2'])[['col_3']].aggregate('min')
```

Out[91]:

		col_3
col_1	**col_2**	
1	**1**	2
	2	1
	3	3
	4	2
2	**2**	1
	3	3
	4	2
3	**4**	3
4	**3**	2

col 3

Σαν aggregate functions μπορείτε να βάλετε [πηγή]:

- `mean()` : Compute mean of groups
- `sum()` : Compute sum of group values
- `size()` : Compute group sizes
- `count()` : Compute count of group
- `std()` : Standard deviation of groups
- `var()` : Compute variance of groups
- `sem()` : Standard error of the mean of groups
- `describe()` : Generates descriptive statistics
- `first()` : Compute first of group values
- `last()` : Compute last of group values
- `nth()` : Take nth value, or a subset if n is a list
- `min()` : Compute min of group values
- `max()` : Compute max of group values

As aggregate functions you can put [source]:

- `mean()` : Compute mean of groups
- `sum()` : Compute sum of group values
- `size()` : Compute group sizes
- `count()` : Compute count of group
- `std()` : Standard deviation of groups
- `var()` : Compute variance of groups
- `sem()` : Standard error of the mean of groups
- `describe()` : Generates descriptive statistics
- `first()` : Compute first of group values
- `last()` : Compute last of group values
- `nth()` : Take nth value, or a subset if n is a list
- `min()` : Compute min of group values
- `max()` : Compute max of group values

Ας δούμε μερικά παραδείγματα από τα "δικά μας" δεδομένα.

Πόσα γονίδια έχει κάθε χρωμόσωμα;

Let's look at some examples from "our" data.

How many genes does each chromosome have?

In [96]:

```
hs.groupby('chromosome')[['GeneID']].aggregate('count')
```

Out[96]:

	GeneID
chromosome	
-	146
1	5826
10	2463
10|19|3	1

	GeneID
chromosome	
11	3302
12	2826
13	1557
14	2290
15	2098
16	2255
17	2812
18	1122
19	2825
2	4471
20	1521
21	885
22	1409
3	3470
4	2743
5	2943
6	3478
7	3200
8	2470
9	2581
MT	110
Un	66
X	2492

Για κάθε ένα από τα χρωμοσώματα X και Y, πόσα διαφορετικά γονίδια υπάρχουν;

For each of the X and Y chromosomes, how many different genes are there?

In [102…

```
hs[hs['chromosome'].isin(['X', 'Y'])].groupby(['chromosome', 'type_of_gene
```

Out[102…

		GeneID
chromosome	**type_of_gene**	
X	**biological-region**	157
	ncRNA	442
	other	10
	protein-coding	830
	pseudo	906
	snoRNA	16
	tRNA	5

		GeneID
chromosome	**type_of_gene**	
	unknown	126
Y	**biological-region**	11
	ncRNA	107
	other	29
	protein-coding	46

Ας αλλάξουμε τη σειρά των στηλών στο grouping:

Let's change the order of the columns in the grouping:

In [104…

```
hs2 = hs[hs['chromosome'].isin(['X', 'Y'])].groupby(['type_of_gene', 'chro
hs2
```

Out[104…

		GeneID
type_of_gene	**chromosome**	
biological-region	**X**	157
	Y	11
ncRNA	**X**	442
	Y	107
other	**X**	10
	Y	29
protein-coding	**X**	830
	Y	46
pseudo	**X**	906
	Y	389
snoRNA	**X**	16
tRNA	**X**	5
unknown	**X**	126
	Y	6

Κάτι αρκετά εξεζητημένο είναι ότι μπορούμε να εφαρμόσουμε μία συνάρτηση σε κάθε group με την apply. Για παράδειγμα Για κάθε type_of_gene ποιο είναι το ποσοσότ που ανήκει στο χρωμόσωμα X και ποιο το αντίστοιχο για το χρωμόσωμα Y;

Something quite sophisticated is that we can apply a function to any group with apply. For example For each type_of_gene what is the percentage belonging to the X chromosome and what corresponds to the Y chromosome?

In [108…

```
hs2.groupby(level=0).apply(lambda x: 100 * x / float(x.sum()))
```

Out[108…

		GeneID
type_of_gene	**chromosome**	
biological-region	**X**	93.452381

		GeneID
type_of_gene	**chromosome**	
	Y	6.547619
ncRNA	**X**	80.510018
	Y	19.489982
other	**X**	25.641026
	Y	74.358974
protein-coding	**X**	94.748858
	Y	5.251142
pseudo	**X**	69.961390
	Y	30.038610
snoRNA	**X**	100.000000
tRNA	**X**	100.000000

Το level σημαίνει σε ποιο "group" (από τα δύο που έχουμε εφαρμόσουμε) να γίνει το apply. Η αλήθεια είναι οτι αφού το X έχει πολύ περισσότερα γονίδια από το Y ο παραπάνω πίνακας δεν μας λέει και πάρα πολλά. Ας κάνουμε apply με τη function στο δεύτερο level:

The level means in which "group" (of the two we have applied) to apply. The truth is that since X has many more genes than Y the above table does not tell us too much. Let's apply with the function in the second level:

In [111…

```
hs3 = hs2.groupby(level=1).apply(lambda x: 100 * x / float(x.sum()))
hs3
```

Out[111…

		GeneID
type_of_gene	**chromosome**	
biological-region	**X**	6.300161
	Y	1.870748
ncRNA	**X**	17.736758
	Y	18.197279
other	**X**	0.401284
	Y	4.931973
protein-coding	**X**	33.306581
	Y	7.823129
pseudo	**X**	36.356340
	Y	66.156463
snoRNA	**X**	0.642055
tRNA	**X**	0.200642
unknown	**X**	5.056180
	Y	1.020408

Ενδιαφέρον: Στο χρωμόσωμα X το 33.3% των γονίδιων που περιέχει είναι protein-coding. Το αντίστοιχο ποσοστό για το Y είναι 7.8%. Ενώ για το Y το 66.2% των γονιδίων είναι pseudo και για το X είναι 36.3%. Μάλλον δεν κάνει και πολλά πράγματα το Y..

Interesting: On the X chromosome 33.3% of the genes it contains are protein-coding. The corresponding percentage for Y is 7.8%. While for Y 66.2% of the genes are pseudo and for X it is 36.3%. Y .. probably does not do many things ..

Μπορούμε να του πούμε: Το πρώτο group κάντο στήλες:

We can tell him: The first group make columns:

In [114…

```
hs3.unstack(0)
```

Out[114…

```
type_of_gene  biological-   ncRNA      other     protein-    pseudo     snoRNA    tRNA
                   region                          coding
  chromosome
           X     6.300161   17.736758  0.401284  33.306581  36.356340  0.642055  0.200642
           Y     1.870748   18.197279  4.931973   7.823129  66.156463       NaN       NaN
```

Ή το 2ο group κάντο στήλες:

Or the 2nd group make columns:

In [115…

```
hs3.unstack(1)
```

Out[115…

```
                                 GeneID
        chromosome          X          Y
      type_of_gene
 biological-region   6.300161   1.870748
             ncRNA  17.736758  18.197279
             other   0.401284   4.931973
    protein-coding  33.306581   7.823129
            pseudo  36.356340  66.156463
            snoRNA   0.642055        NaN
              tRNA   0.200642        NaN
           unknown   5.056180   1.020408
```

Το unstacking είναι σημαντικό γιατί μας επιτρέπει να κάνουμε τα groups μπάρες (δες και συνέχεια)

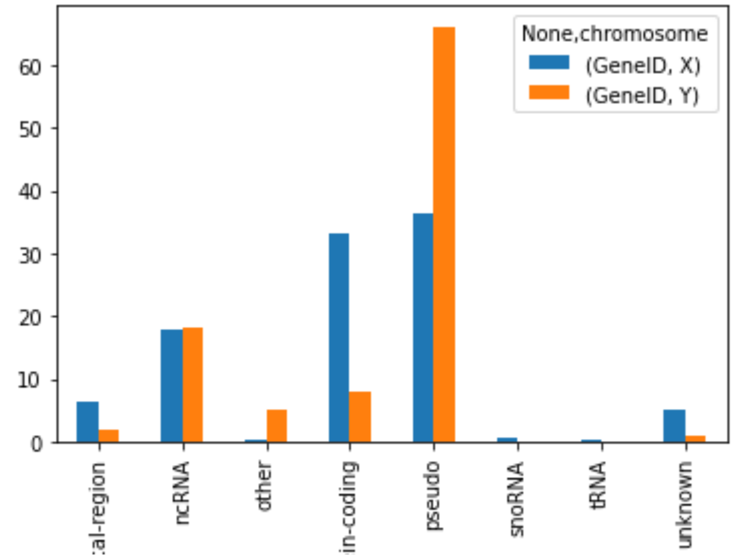
Unstacking is important because it allows us to make groups bars (see also below)

In [117…

```
hs3.unstack(1).plot(kind='bar')
```

Out[117…

```
<AxesSubplot:xlabel='type_of_gene'>
```



Εδώ βλέπουμε πως κάναμε μία "επεξεργασία" των δεδομένων χωρίς να κάνουμε ούτε μία for, if, κτλ.. Αυτός είναι ο "δηλωτικός τρόπος προγραμματισμού". Συνιθίζεται όταν γράφουμε πολλές εντολές που κάνουν διαδοχικές επεξεργασίες να τις γράφουμε με αυτό το στυλ (method chaining):

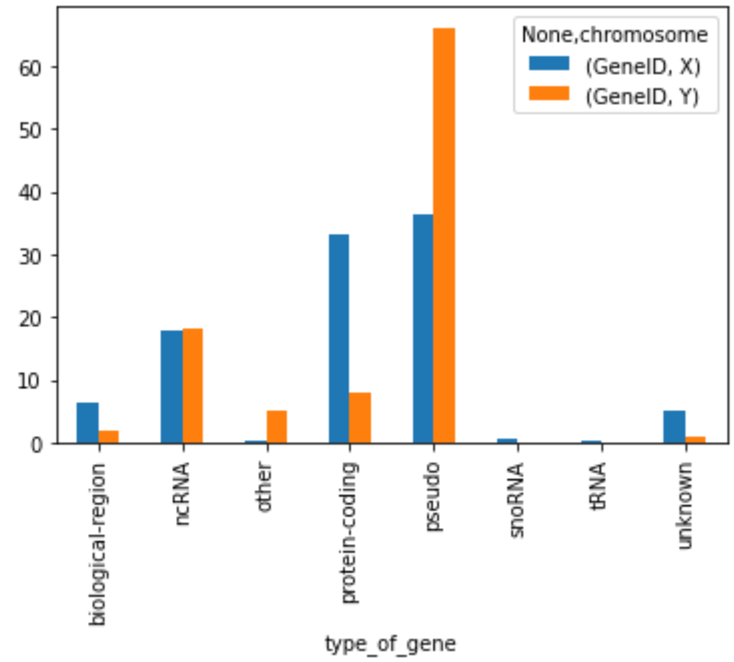
Here we see that we did a "processing" of the data without doing any for, if, etc .. This is the " declarative way of programming ". It is common when writing many commands that do sequential processing to write them in this style (method chaining):

In [125…

```
(hs[hs['chromosome'].isin(['X', 'Y'])]
 .groupby(['type_of_gene', 'chromosome'])[['GeneID']]
 .aggregate('count')
 .groupby(level=1)
 .apply(lambda x: 100 * x / float(x.sum()))
 .unstack(1)
 .plot(kind='bar')
)
```

Out[125…

```
<AxesSubplot:xlabel='type_of_gene'>
```



Plotting

Η pandas υποστηρίζει ένα μεγάλο πλήθος από plots. Σε μελλοντικό μάθημα θα ασχοληθούμε περισσότερο με το πως κάνουμε plots χωρίς της pandas.

Barplots: πλήθος απο γονίδια ανά χρωμόσωμα:

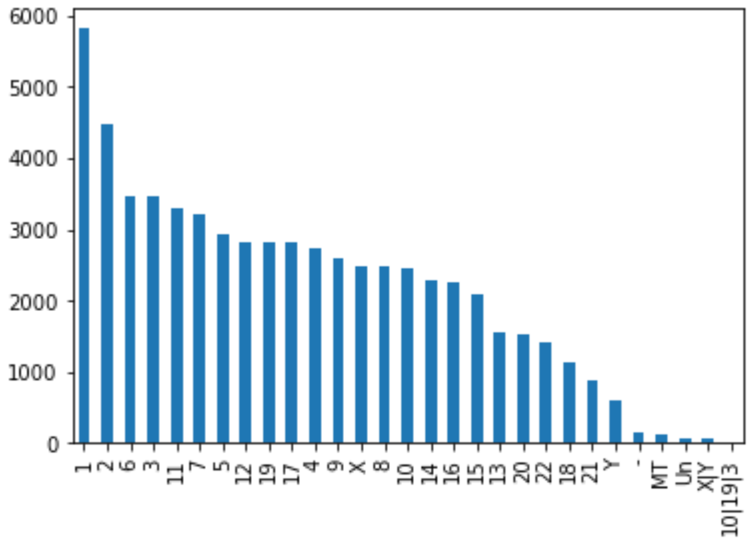
Plotting

Pandas support a large number of plots. In a future lesson we will deal more with how to make plots without pandas.

Barplots: number of genes per chromosome:

In [5]:
```
hs['chromosome'].value_counts().plot(kind='bar')
```

Out[5]:
```
<AxesSubplot:>
```

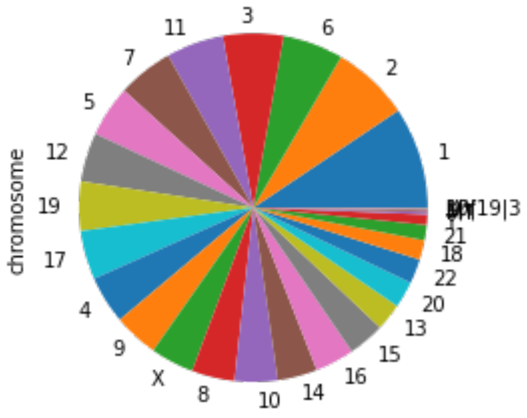


Το ίδιο σε piechart. **ΠΡΟΣΟΧΗ!!** Αποφεύγουμε να χρησιμοποιούμε piechart σε επιστημονικές δημοσιεύσεις! Google: `why are pie charts bad`

The same in piechart. **CAUTION!!** Avoid using piechart in scientific publications! Google: `why are pie charts bad`

In [6]:
```
hs['chromosome'].value_counts().plot(kind='pie')
```

Out[6]:
```
<AxesSubplot:ylabel='chromosome'>
```



Ένα παράδειγμα με GWAS

An example with GWAS

Ας χρησιμοποιήσουμε έναν κατάλογο από GWA studies. Ο κατάλογος βρίσκεται σε αυτό το link: https://www.ebi.ac.uk/gwas/api/search/downloads/full για να το φορτώσετε τρέξτε (κάνει πολύ ώρα!):

Let's use a list from GWA studies . The directory is at this link: https://www.ebi.ac.uk/gwas/api/search/downloads/full to load it run (it takes a long time!):

```
In [160… gwas = pd.read_csv('https://www.ebi.ac.uk/gwas/api/search/downloads/full',
```

```
/Users/admin/anaconda3/lib/python3.8/site-packages/IPython/core/interactive
shell.py:3146: DtypeWarning: Columns (9,11,12,23,27) have mixed types.Speci
fy dtype option on import or set low_memory=False.
  has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

Οι στήλες:

The columns:

```
In [161… gwas.columns
```

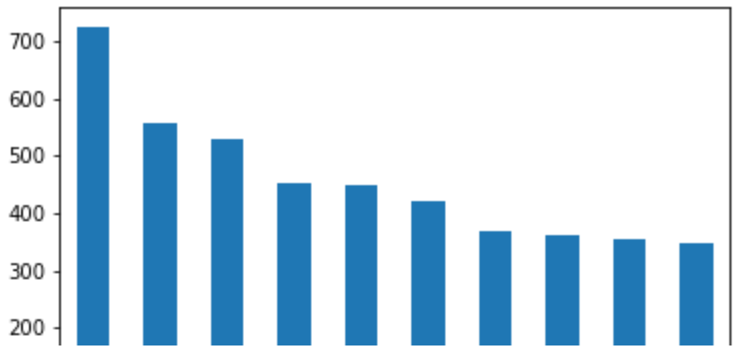
```
Out[161… Index(['DATE ADDED TO CATALOG', 'PUBMEDID', 'FIRST AUTHOR', 'DATE', 'JOURNA
L',
       'LINK', 'STUDY', 'DISEASE/TRAIT', 'INITIAL SAMPLE SIZE',
       'REPLICATION SAMPLE SIZE', 'REGION', 'CHR_ID', 'CHR_POS',
       'REPORTED GENE(S)', 'MAPPED_GENE', 'UPSTREAM_GENE_ID',
       'DOWNSTREAM_GENE_ID', 'SNP_GENE_IDS', 'UPSTREAM_GENE_DISTANCE',
       'DOWNSTREAM_GENE_DISTANCE', 'STRONGEST SNP-RISK ALLELE', 'SNPS',
       'MERGED', 'SNP_ID_CURRENT', 'CONTEXT', 'INTERGENIC',
       'RISK ALLELE FREQUENCY', 'P-VALUE', 'PVALUE_MLOG', 'P-VALUE (TEXT)',
       'OR or BETA', '95% CI (TEXT)', 'PLATFORM [SNPS PASSING QC]', 'CNV'],
      dtype='object')
```

Ποια είναι τα 10 γονίδια στα οποία έχουν γίνει τα περισσότερα GWAS;

What are the 10 genes in which most GWAS have been made?

```
In [162… gwas["MAPPED_GENE"].value_counts()[:10].plot(kind="bar")
```

```
Out[162… <AxesSubplot:>
```



Μετατροπή του DATE από string σε datetime

Convert DATE from string to datetime

```
In [163… gwas['DATE'] = pd.to_datetime(gwas["DATE"]) # Μετατροπή του DATE από strin
```

Ας δούμε μερικά χαρακτηριστικά της στήλης P-VALUE ;

Let's look at some features of the P-VALUE column?

```
In [194… gwas['PVALUE_MLOG'].describe()
```

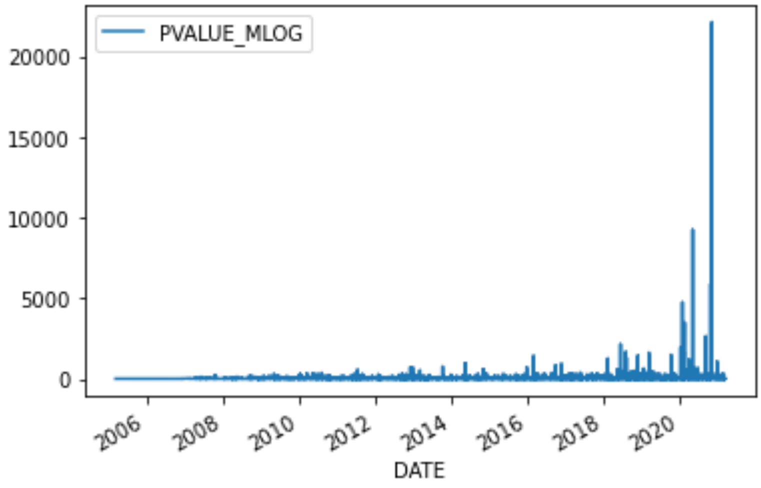
```
Out[194… count    251401.000000
         mean         17.671208
         std          66.731631
         min           5.000000
         25%           7.000000
         50%           9.301030
         75%          14.698970
         max       22135.221849
         Name: PVALUE_MLOG, dtype: float64
```

Ας τη κάνουμε plot με βάση τον χρόνο:

Let's plot it based on time:

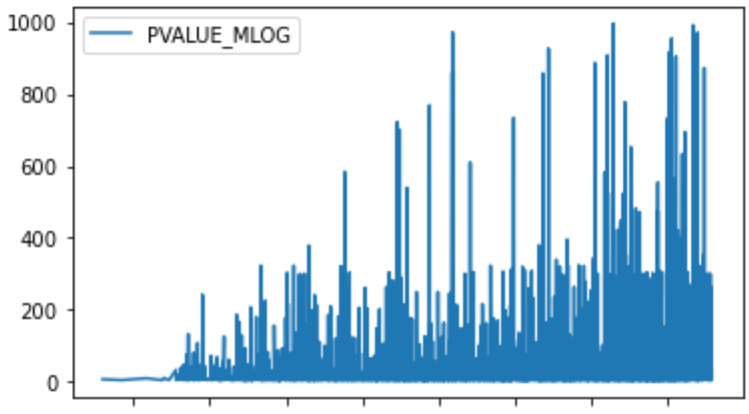
```
In [198… gwas.plot(x='DATE', y='PVALUE_MLOG')
```

```
Out[198… <AxesSubplot:xlabel='DATE'>
```



```
In [208… gwas[gwas['PVALUE_MLOG']<1000].plot(x='DATE', y='PVALUE_MLOG')
```

```
Out[208… <AxesSubplot:xlabel='DATE'>
```



Πόσα gwas δημοσιεύονται κάθε χρόνο;

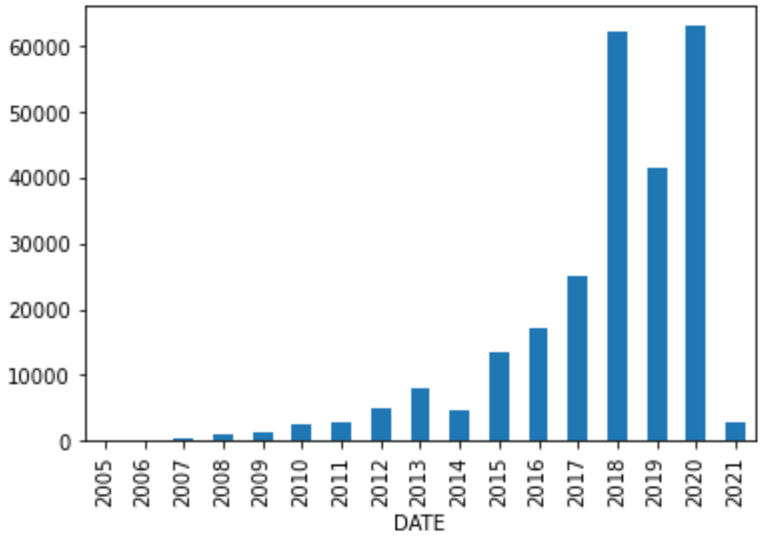
How many gwas are published each year?

In [217…

```
gwas.groupby(gwas['DATE'].dt.year)['DATE'].count().plot(kind='bar')
```

Out[217…

```
<AxesSubplot:xlabel='DATE'>
```



Ένας άλλος τρόπος να τα πλοτάρουμε:

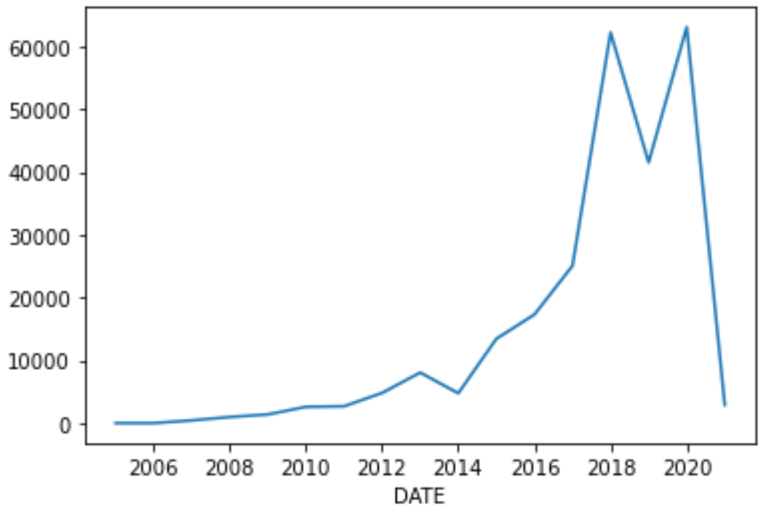
Another way to plot them:

In [218…

```
gwas.groupby(gwas['DATE'].dt.year)['DATE'].count().plot()
```

Out[218…

```
<AxesSubplot:xlabel='DATE'>
```



Ας πάρουμε όλα τα GWAS που έχουν γίνει σε ασθένειες ή φαινότυπους που έχουν μέσα τη λέξη "Breast", και τα SNPs που έχουν βρεθεί έχουν συσχετιστεί με p-value<10^{-10}, και ας τα κατατάξουμε σε χρωμοσώματα:

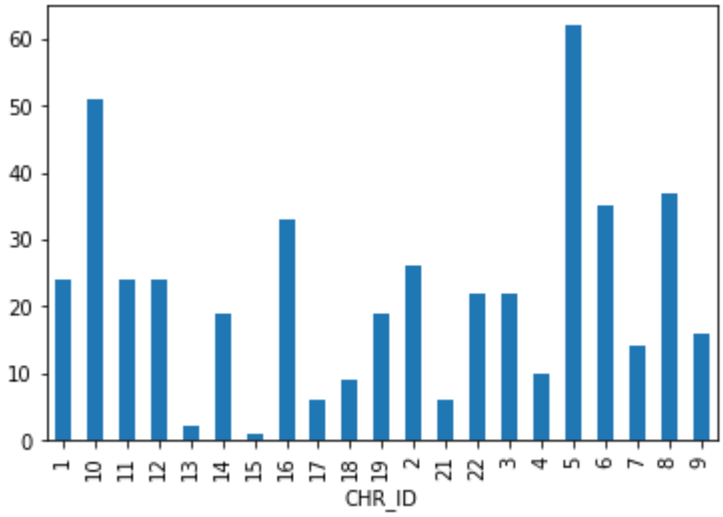
Let's take all the GWAS that have been done on diseases or phenotypes that have the word "Breast" in them, and the SNPs that have been found have been associated with p-value <10 <sup> -10 </sup>, and let's rank them chromosomes:

In [220…

```
(
    gwas[ (gwas["DISEASE/TRAIT"].str.contains('Breast')) & (gwas["PVALUE_MI
    .groupby("CHR_ID")['CHR_ID']
    .aggregate('count').plot(kind='bar')
)
```

Out[220…

```
<AxesSubplot:xlabel='CHR_ID'>
```



Λογικό ότι το χρωμόσωμα 5 που έχει το BRCA2 gene είναι #1

It makes sense that chromosome 5 on the BRCA2 gene is # 1

Ποιος είναι ο ερευνητής που έχει τις περισσότερες δημοσιεύσεις στο Nature Genetics;

Who is the researcher with the most publications in Nature Genetics?

In [221…

```
(
    gwas[gwas['JOURNAL'] == 'Nat Genet']
    .groupby('FIRST AUTHOR')
    .aggregate('count')['PUBMEDID']
    .idxmax()
)
```

Out[221…

```
'Lee JJ'
```

Ποιο region περιέχει τις περισσότερες μελέτες σχετικά με καρκίνο;

Which region contains the most cancer studies?

In [225…

```
(
    gwas[gwas['DISEASE/TRAIT'].str.contains('cancer', case=False, na=False
    .groupby('REGION')
    .aggregate('count')['PUBMEDID']
    .idxmax()
)
```

Out[225… '8q24.21'

Ποιος είναι ο μέσος όρος και το median του allele_frequency για όλα τα variants που ανακαλύπτοντε κάθε χρόνο;

What is the average and median of allele_frequency for all the variants you discover each year?

In [277…
```
gwas['RISK ALLELE FREQUENCY']=pd.to_numeric(gwas['RISK ALLELE FREQUENCY'],

gwas_2 = (
 gwas
 .groupby(gwas_2['DATE'].dt.year)['RISK ALLELE FREQUENCY']
 .aggregate(['mean', 'median'])
)
gwas_2
```

Out[277…

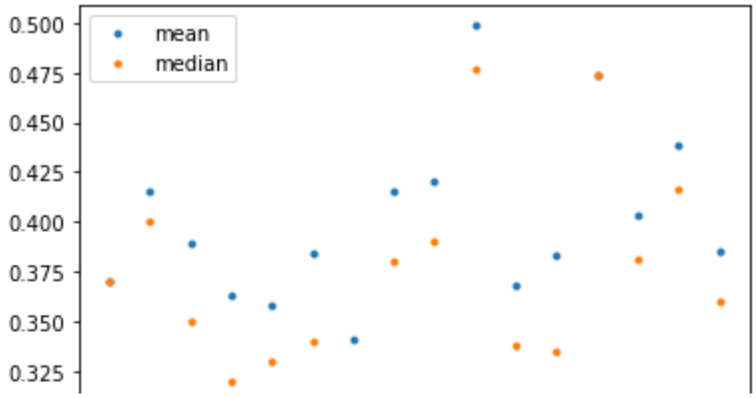
	mean	median
DATE		
2005.0	NaN	NaN
2006.0	0.370000	0.370000
2007.0	0.415373	0.400000
2008.0	0.389350	0.350000
2009.0	0.362769	0.320000
2010.0	0.358266	0.330000
2011.0	0.384316	0.340000
2012.0	0.340888	0.300000
2013.0	0.414986	0.380000
2014.0	0.420903	0.390000
2015.0	0.498522	0.477000
2016.0	0.367788	0.338245
2017.0	0.382940	0.335076
2018.0	0.473471	0.474000
2019.0	0.403819	0.381000
2020.0	0.438670	0.416900
2021.0	0.385103	0.359920

Ας κάνουμε ένα scatter plot με τον x να είναι το YEAR και το y να είναι τα mean και median

Let's make a scatter plot with x being the YEAR and y being the mean and median

In [279…
```
gwas_2.plot(style='.')
```

Out[279…
```
<AxesSubplot:xlabel='DATE'>
```



Και ένα scatter plot με το χ να είναι το mean και το y το median:

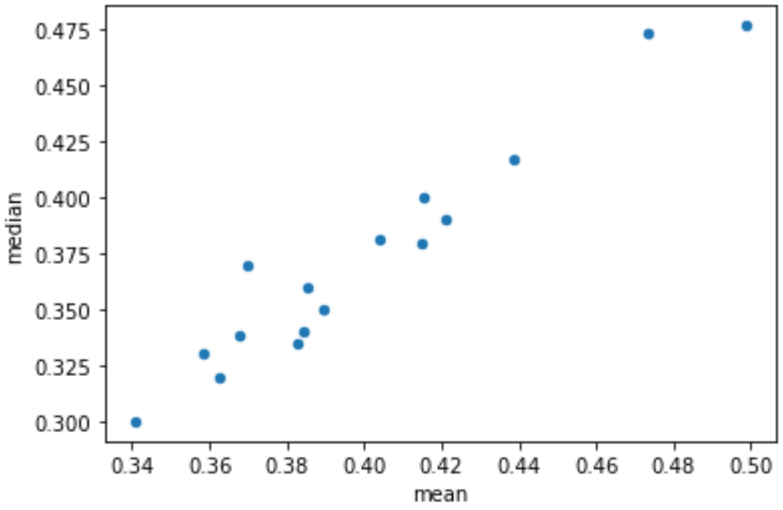
And a scatter plot with x being the mean and y being the median:

In [280…

```
gwas_2.plot.scatter(x='mean', y='median')
```

Out[280…

```
<AxesSubplot:xlabel='mean', ylabel='median'>
```



Περισσότερα

- Cheatsheet
- Introduction to Pandas . plotting with pandas
- 100 pandas puzzles

More

- Cheatsheet
- Introduction to Pandas . plotting with pandas
- 100 pandas puzzles

In [ ]: