Computational Statistics with Python

Topic 3: Further Python

Expected lecture time: 2-3 hours

Giancarlo Manzi

Department of Economics, Management and Quantitative Methods

University of Milan, Milan, Italy

The Pandas series object

Series is a one-dimensional labeled array from the library Pandas capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.).

In [2]:

```
import pandas as pd
import matplotlib.pyplot as plt
# Using Numpy's pseudo random number generator
import numpy as np
data = np.random.randn(20)
index = range(1990, 2010)
```

In [3]:

```
print (data)
print (list(index))
```

```
[ 1.07412852  0.03522929 -0.00307068 -0.7969454   0.08838703 -0.09
938906
 -0.56601135  0.98303324 -1.46928736  1.53688249  0.18417489  0.61
477394
 -0.12605095  1.60410251  0.74209285  0.74957552 -0.312181   -0.46
701963
  1.1470691  -1.22639548]
[1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000,
2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009]
```

In [4]:

```
y = pd.Series(data, index=index)
```

In [5]:
```
print (y)
```

```
1990    1.074129
1991    0.035229
1992   -0.003071
1993   -0.796945
1994    0.088387
1995   -0.099389
1996   -0.566011
1997    0.983033
1998   -1.469287
1999    1.536882
2000    0.184175
2001    0.614774
2002   -0.126051
2003    1.604103
2004    0.742093
2005    0.749576
2006   -0.312181
2007   -0.467020
2008    1.147069
2009   -1.226395
dtype: float64
```

In [6]:
```
salaries = {
    'juan': 1500, 'maria': 2560.34, 'cesc': None, 'juan carlos'
}
```

In [7]:
```
s = pd.Series(salaries)
```

In [7]:
```
print (s)
```

```
juan           1500.00
maria          2560.34
cesc               NaN
juan carlos    2451.00
dtype: float64
```

Access series as arrays

In [9]:
```
print (s[:2])
print (s[s > s.median()], '\n')
print (np.log(s), '\n')
print (s + s, '\n')
print (s * 3, '\n')
print (y[4:8] + y[4:10])
```

```
juan     1500.00
maria    2560.34
dtype: float64
maria    2560.34
dtype: float64

juan           7.313220
maria          7.847895
cesc                NaN
juan carlos    7.804251
dtype: float64

juan           3000.00
maria          5120.68
cesc               NaN
juan carlos    4902.00
dtype: float64

juan           4500.00
maria          7681.02
cesc               NaN
juan carlos    7353.00
dtype: float64

1994    0.176774
1995   -0.198778
1996   -1.132023
1997    1.966066
1998         NaN
1999         NaN
dtype: float64
```

Difference between Python list and Pandas series

In [10]:
```
my_list = ['a', 'b', 'c', 'd']
print(my_list)
```

```
['a', 'b', 'c', 'd']
```

In [11]:
```
my_series = pd.Series(my_list, index = [2,1,3,0])
print(my_series)
```

```
2    a
1    b
3    c
0    d
dtype: object
```

In [14]:
```
print(my_series[3])
```

```
c
```

In [15]:
```
print(my_list[3])
```

```
d
```

Data Frames

From http://pandas.pydata.org/pandas-docs/stable/dsintro.html#dataframe (http://pandas.pydata.org/pandas-docs/stable/dsintro.html#dataframe)

DataFrame is a 2-dimensional labeled data structure with columns of potentially different types. You can think of it like a spreadsheet or SQL table, or a dict of Series objects. It is generally the most commonly used pandas object. Like Series, DataFrame accepts many different kinds of input:

- Dict of 1D ndarrays, lists, dicts, or Series
- 2-D numpy.ndarray
- Structured or record ndarray
- A series
- Another data frame

Along with the data, you can optionally pass index (row labels) and columns (column labels) arguments. If you pass an index and / or columns, you are guaranteeing the index and / or columns of the resulting DataFrame. Thus, a dict of Series plus a specific index will discard all data not matching up to the passed index.

If axis labels are not passed, they will be constructed from the input data based on common sense rules.

In [16]:
```
y = 2020
s = 2000
k = {'Smith': y, 'McDonald': s}
k
```

Out[16]:
```
{'Smith': 2020, 'McDonald': 2000}
```

In [18]:
```
df = pd.DataFrame(k.items())
df
```

Out[18]:

	0	1
0	Smith	2020
1	McDonald	2000

In [19]:
```
print (df)
```

```
          0     1
0     Smith  2020
1  McDonald  2000
```

In [20]:
```
pd.DataFrame(k.items(), columns=['Name', 'Salary'])
```

Out[20]:

	Name	Salary
0	Smith	2020
1	McDonald	2000

In [21]:
```
s = pd.Series(k, name='DateValue')
```

In [22]:
```
s.index.name = 'Name'
s
```

Out[22]:
```
Name
Smith       2020
McDonald    2000
Name: DateValue, dtype: int64
```

Loading and manipulating data

Retrieve the complete local dataset from [Kaggle website (https://www.kaggle.com/daveianhickey/2000-16-traffic-flow-england-scotland-wales)](https://www.kaggle.com/daveianhickey/2000-16-traffic-flow-england-scotland-wales).

In [27]:

```
accidents = 'accidents_2012_to_2014.csv'
A = pd.read_csv(accidents, low_memory=False, index_col=0)
A
```

Out[27]:

	Location_Easting_OSGR	Location_Northing_OSGR	Longitude	Latitude	Pc
Accident_Index					
201201BS70001	527200	178760	-0.169101	51.493429	
201201BS70002	524930	181430	-0.200838	51.517931	
201201BS70003	525860	178080	-0.188636	51.487618	
201201BS70004	524980	181030	-0.200259	51.514325	
201201BS70005	526170	179200	-0.183773	51.497614	
...	...	...	...	...	
2.01E+12	310037	597647	-3.417278	55.264773	
2.01E+12	321509	574063	-3.230255	55.054855	
2.01E+12	321337	566365	-3.230826	54.985668	
2.01E+12	323869	566853	-3.191397	54.990446	
2.01E+12	314072	579971	-3.348426	55.106700	

464697 rows × 32 columns

In [28]:
```
A.head()
```

Out[28]:

	Location_Easting_OSGR	Location_Northing_OSGR	Longitude	Latitude	Po
Accident_Index					
201201BS70001	527200	178760	-0.169101	51.493429	
201201BS70002	524930	181430	-0.200838	51.517931	
201201BS70003	525860	178080	-0.188636	51.487618	
201201BS70004	524980	181030	-0.200259	51.514325	
201201BS70005	526170	179200	-0.183773	51.497614	

5 rows × 32 columns

In [29]:
```
A[['Date', 'Time']].head()
```

Out[29]:

	Date	Time
Accident_Index		
201201BS70001	19/01/2012	20:35
201201BS70002	04/01/2012	17:00
201201BS70003	10/01/2012	10:07
201201BS70004	18/01/2012	12:20
201201BS70005	17/01/2012	20:24

In [23]:
```
A.dtypes
```

Out[23]:
```
Location_Easting_OSGR                              int64
Location_Northing_OSGR                             int64
Longitude                                        float64
Latitude                                         float64
Police_Force                                       int64
Accident_Severity                                  int64
Number_of_Vehicles                                 int64
Number_of_Casualties                               int64
Date                                              object
Day_of_Week                                        int64
Time                                              object
Local_Authority_(District)                         int64
Local_Authority_(Highway)                         object
1st_Road_Class                                     int64
1st_Road_Number                                    int64
Road_Type                                         object
Speed_limit                                        int64
Junction_Detail                                  float64
Junction_Control                                  object
2nd_Road_Class                                     int64
2nd_Road_Number                                    int64
Pedestrian_Crossing-Human_Control                 object
Pedestrian_Crossing-Physical_Facilities           object
Light_Conditions                                  object
Weather_Conditions                                object
Road_Surface_Conditions                           object
Special_Conditions_at_Site                        object
Carriageway_Hazards                               object
Urban_or_Rural_Area                                int64
Did_Police_Officer_Attend_Scene_of_Accident       object
LSOA_of_Accident_Location                         object
Year                                               int64
dtype: object
```

In [30]:
```
from datetime import datetime

def todate(d, t):
    try:
        dt = datetime.strptime(" ".join([d, t]), '%d/%m/%Y %H:%
    except TypeError:
        dt = np.nan
    return dt
```

In [31]:
```
A['Datetime'] = [todate(x.Date, x.Time) for i, x in A.iterrows(
```

In [32]:
```
A[['Datetime', 'Police_Force']].head()
```

Out[32]:

	Datetime	Police_Force
Accident_Index		
201201BS70001	2012-01-19 20:35:00	1
201201BS70002	2012-01-04 17:00:00	1
201201BS70003	2012-01-10 10:07:00	1
201201BS70004	2012-01-18 12:20:00	1
201201BS70005	2012-01-17 20:24:00	1

In [33]:
```
A.shape
```

Out[33]:
```
(464697, 33)
```

In [34]:
```
A.dtypes
```

Out[34]:
```
Location_Easting_OSGR                int64
Location_Northing_OSGR               int64
Longitude                          float64
Latitude                           float64
Police_Force                         int64
Accident_Severity                    int64
Number_of_Vehicles                   int64
Number_of_Casualties                 int64
Date                                object
Day_of_Week                          int64
Time                                object
Local_Authority_(District)           int64
Local_Authority_(Highway)           object
1st_Road_Class                       int64
1st_Road_Number                      int64
Road_Type                           object
Speed_limit                          int64
Junction_Detail                    float64
Junction_Control                    object
2nd_Road_Class                       int64
```

Access dataframe by index and col

In [35]:
```
my_df = A.iloc[2:6] # gets rows (or columns) at particular posi
my_df
```

Out[35]:

	Location_Easting_OSGR	Location_Northing_OSGR	Longitude	Latitude	P
Accident_Index					
201201BS70003	525860	178080	-0.188636	51.487618	
201201BS70004	524980	181030	-0.200259	51.514325	
201201BS70005	526170	179200	-0.183773	51.497614	
201201BS70006	526090	177600	-0.185496	51.483253	

4 rows × 33 columns

In [36]:
```
#SUBSETTING a data frame
selection = A[A['Road_Surface_Conditions'] == 'Dry'].sort_value
    'Number_of_Casualties', ascending=False)
selection
#selection[['Weather_Conditions', 'Police_Force',
#           'Accident_Severity', 'Number_of_Vehicles', 'Number_
```

Out[36]:

	Location_Easting_OSGR	Location_Northing_OSGR	Longitude	Latitude	P
Accident_Index					
20144100J0489	523000	199780	-0.222211	51.683269	
201411NH11644	418196	552132	-1.718034	54.863663	
2.01E+12	591380	169440	0.749417	51.391660	
2.01E+12	375840	203065	-2.351207	51.725734	
201422E404170	355950	235980	-2.643371	52.020443	
...	...	...	...	...	
201297QC00409	304120	637780	-3.524192	55.624152	
201297QC00510	281810	652360	-3.884585	55.750175	
201297QC00605	294640	612550	-3.665077	55.395579	
201297QC00606	302140	641540	-3.556962	55.657530	
2.01E+12	311812	580747	-3.384080	55.113274	

319370 rows × 33 columns

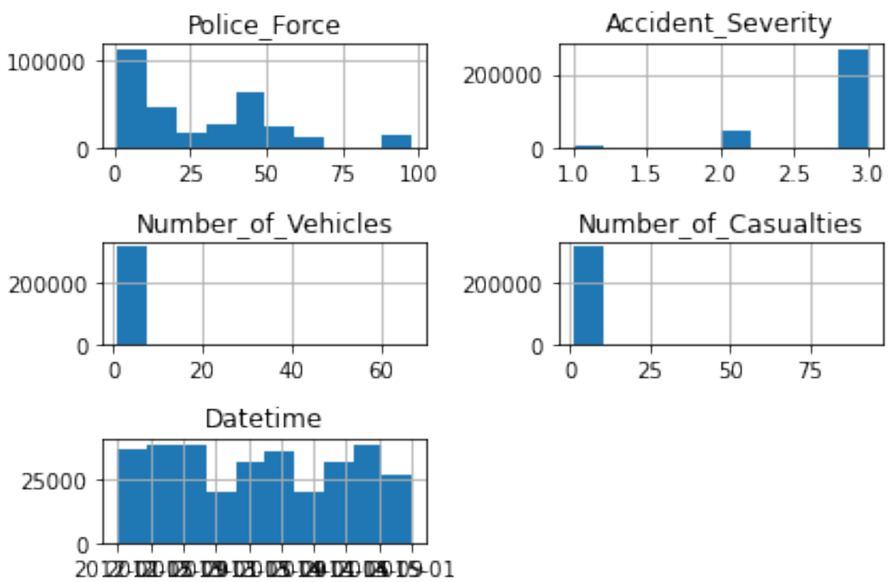
In [37]:
```
selection[['Weather_Conditions', 'Police_Force', 'Accident_Seve
          'Number_of_Vehicles', 'Number_of_Casualties']].group
```

Out[37]:

Weather_Conditions	Police_Force	Accident_Severity	Number_of_Vehicles	Number_of_Casua
Fine with high winds	32.652875	2.811360	1.796283	1.35
Fine without high winds	27.051892	2.830949	1.846165	1.32
Fog or mist	39.051163	2.797674	1.997674	1.52
Other	29.449333	2.868000	1.788000	1.26
Raining with high winds	32.687500	2.833333	1.895833	1.45
Raining without high winds	38.734211	2.873684	1.792105	1.29
Snowing with high winds	45.666667	2.666667	1.777778	1.77
Snowing without high winds	31.560976	2.902439	1.780488	1.19
Unknown	27.058422	2.872004	1.766977	1.21

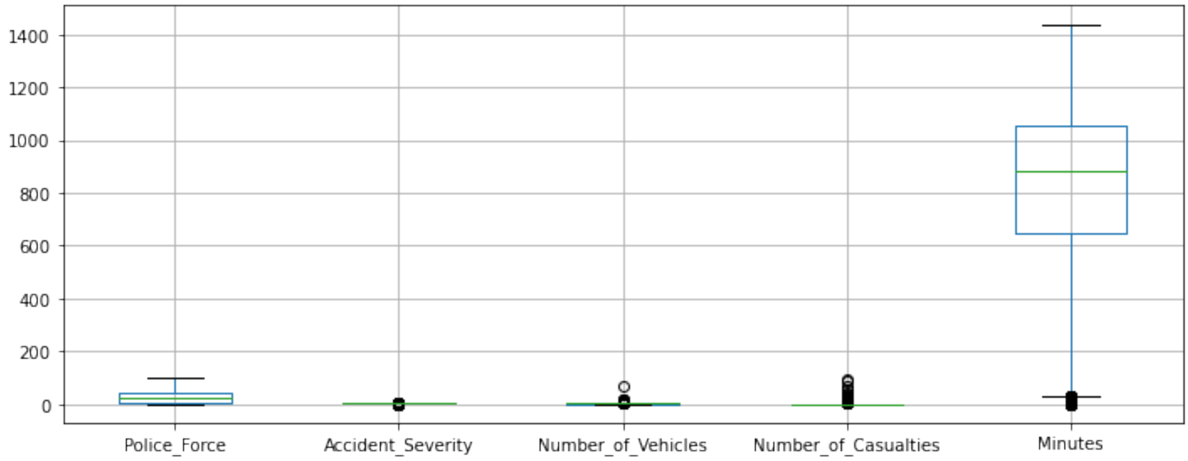
In [38]:
```
sel = selection[['Weather_Conditions', 'Police_Force', 'Acciden
          'Number_of_Vehicles', 'Number_of_Casualties', 'Datet
```

In [34]:
```
sel.hist()
plt.tight_layout()
plt.show()
```

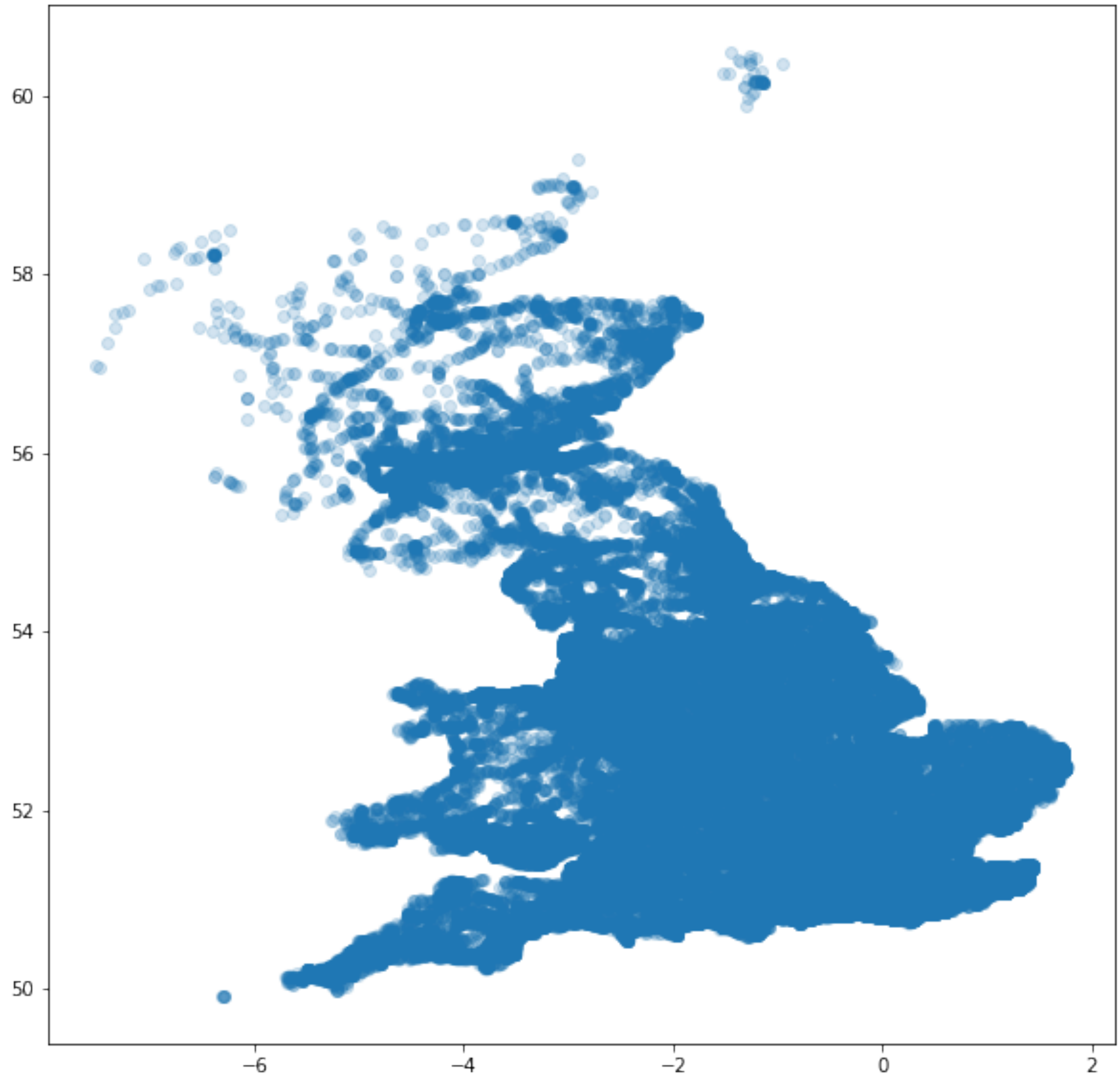


In [40]:
```
minutes = []
for i, row in sel.iterrows():
    h, m = row['Datetime'].hour, row['Datetime'].minute
    minutes.append(h*60 + m)
sel = sel.copy()
sel['Minutes'] = minutes
```

In [42]:
```
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(10, 4), sha
sel.boxplot(ax=axes)
plt.tight_layout()
plt.show()
```



In [38]:
```
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(10, 10), sh
axes.scatter(selection.Longitude.values, selection.Latitude.val
plt.show()
```



In [43]:
```
pip install geopandas
```

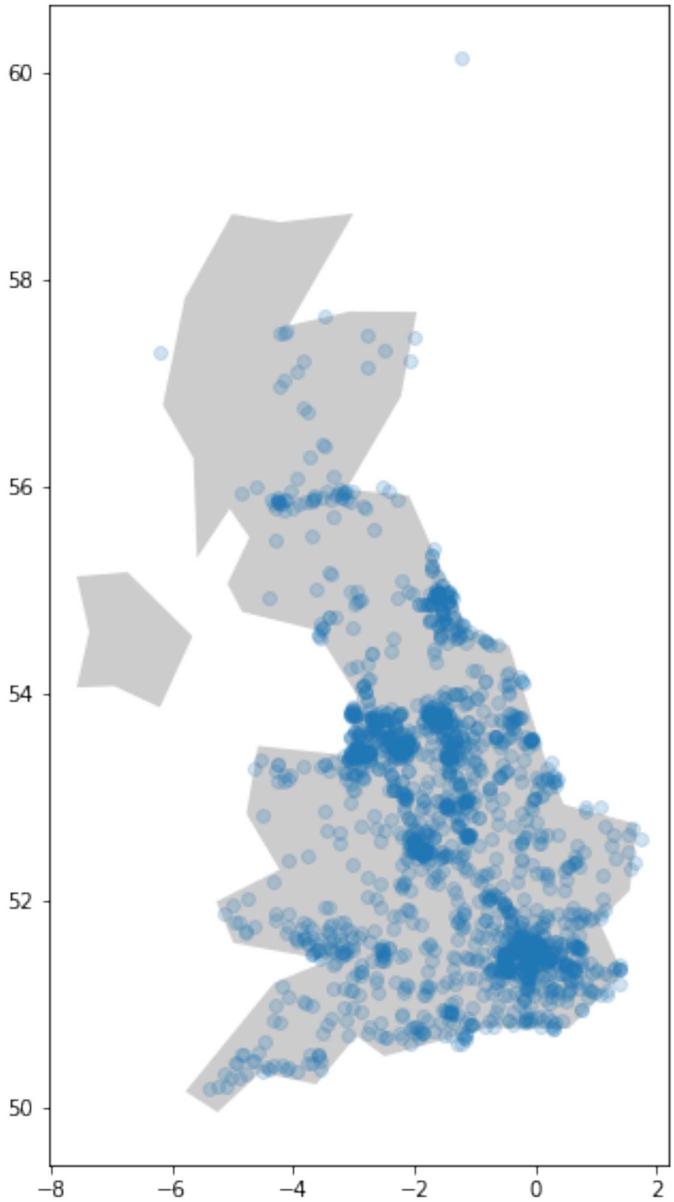
```
Requirement already satisfied: geopandas in /opt/anaconda3/lib/python3.8/site-packages (0.9.0)
Requirement already satisfied: fiona>=1.8 in /opt/anaconda3/lib/python3.8/site-packages (from geopandas) (1.8.19)
Requirement already satisfied: shapely>=1.6 in /opt/anaconda3/lib/python3.8/site-packages (from geopandas) (1.7.1)
Requirement already satisfied: pandas>=0.24.0 in /opt/anaconda3/lib/python3.8/site-packages (from geopandas) (1.3.4)
Requirement already satisfied: pyproj>=2.2.0 in /opt/anaconda3/lib/python3.8/site-packages (from geopandas) (3.0.1)
Requirement already satisfied: attrs>=17 in /opt/anaconda3/lib/python3.8/site-packages (from fiona>=1.8->geopandas) (20.3.0)
Requirement already satisfied: cligj>=0.5 in /opt/anaconda3/lib/python3.8/site-packages (from fiona>=1.8->geopandas) (0.7.1)
Requirement already satisfied: six>=1.7 in /opt/anaconda3/lib/python3.8/site-packages (from fiona>=1.8->geopandas) (1.15.0)
Requirement already satisfied: certifi in /opt/anaconda3/lib/python3.8/site-packages (from fiona>=1.8->geopandas) (2020.12.5)
Requirement already satisfied: click-plugins>=1.0 in /opt/anaconda3/lib/python3.8/site-packages (from fiona>=1.8->geopandas) (1.1.1)
Requirement already satisfied: click<8,>=4.0 in /opt/anaconda3/lib/python3.8/site-packages (from fiona>=1.8->geopandas) (7.1.2)
Requirement already satisfied: munch in /opt/anaconda3/lib/python3.8/site-packages (from fiona>=1.8->geopandas) (2.5.0)
Requirement already satisfied: python-dateutil>=2.7.3 in /opt/anaconda3/lib/python3.8/site-packages (from pandas>=0.24.0->geopandas) (2.8.1)
Requirement already satisfied: pytz>=2017.3 in /opt/anaconda3/lib/python3.8/site-packages (from pandas>=0.24.0->geopandas) (2021.1)
Requirement already satisfied: numpy>=1.17.3 in /opt/anaconda3/lib/python3.8/site-packages (from pandas>=0.24.0->geopandas) (1.22.1)
WARNING: You are using pip version 21.2.2; however, version 22.1.2 is available.
You should consider upgrading via the '/opt/anaconda3/bin/python -m pip install --upgrade pip' command.
Note: you may need to restart the kernel to use updated packages.
```

In [44]:
```
import geopandas as gpd
```

In [45]:
```
world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowre
```

In [46]:
```
UK = world[world['iso_a3']=='GBR']
```

In [47]:
```
limit = 2000
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(10, 10), sh
UK.plot(ax=axes, color='#CCCCCC')
axes.scatter(selection.Longitude.values[:limit], selection.Lati
plt.show()
```



Standard tools for machine learning

In [51]:
```
# Standard import for ML
import numpy as np
import os
import tarfile
import requests
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline

# Matplotlib defaul setting
# When using the 'inline' backend, your matplotlib graphs will
%matplotlib inline

mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)
#np1. + any method or function you want to use
```

Get the data

Get the housing data (https://www.kaggle.com/harrywang/housing (https://www.kaggle.com/harrywang/housing)) from the Web through requests and load into a DataFrame from file

In [ ]:
```

```

In [53]:
```
url_data = "https://raw.githubusercontent.com/ageron/handson-ml
data_path = os.path.join("datasets", "housing")
if not os.path.isdir(data_path):
    os.makedirs(data_path)
with open(os.path.join(data_path,'housing.tgz'),'wb') as f:
    f.write(requests.get(url_data).content)
housing_tgz = tarfile.open(os.path.join(data_path,'housing.tgz'
housing_tgz.extractall(path=data_path)
housing_tgz.close()
```

In [54]:
```
housing = pd.read_csv('datasets/housing/housing.csv')
```

Data exploration

In [8]:
```python
housing.head()
```

Out[8]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	house
0	-122.23	37.88	41.0	880.0	129.0	322.0	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1
2	-122.24	37.85	52.0	1467.0	190.0	496.0	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	

Get information about all the columns

In [55]:
```python
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object 
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

Count the unique values in column (e.g. *ocean_proximity*)

In [56]:
```python
housing['ocean_proximity'].value_counts()
```

Out[56]:
```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

Summary statistics of the columns

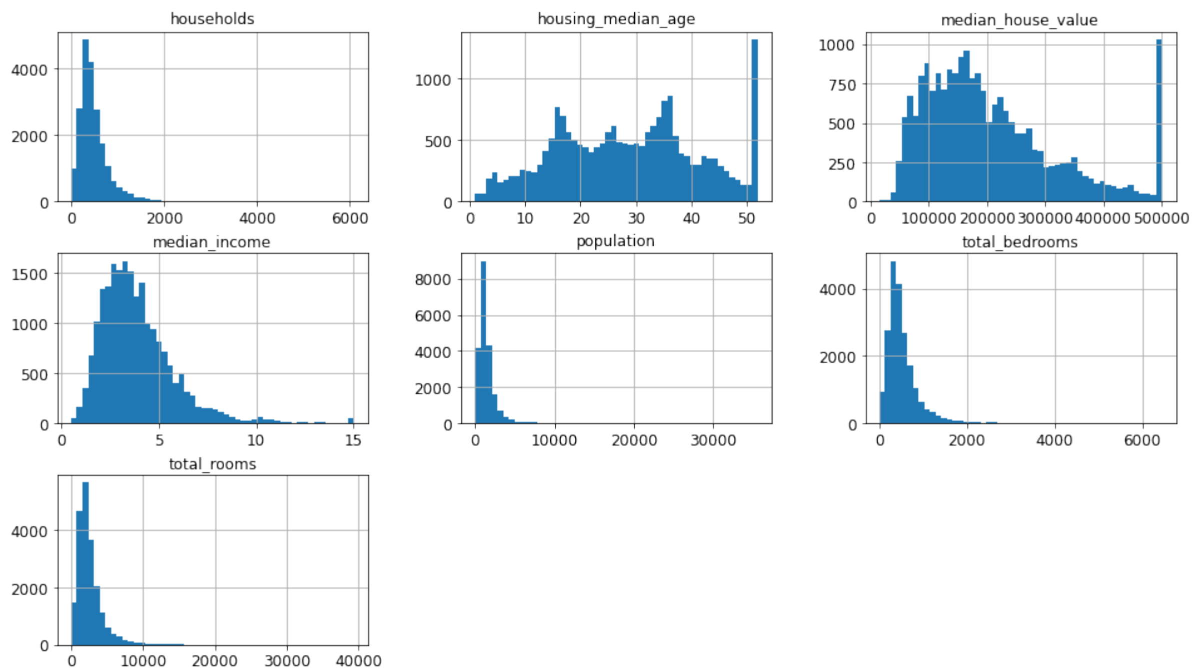
In [57]:
```
housing.describe()
```

Out[57]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20(
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	14
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1'
min	-124.350000	32.540000	1.000000	2.000000	1.000000	
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1'
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1'
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35(

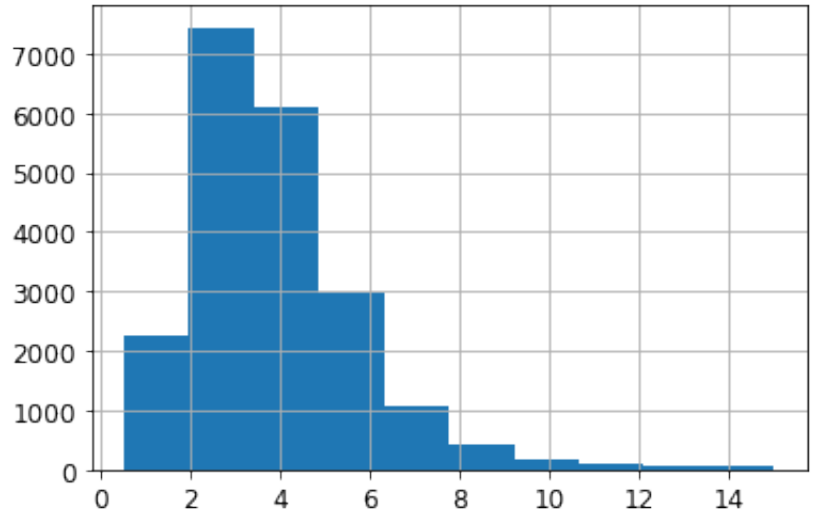
Simple visualization of the distribution of a subset of features: 'households','housing_median_age', 'median_house_value','median_income','population','total_bedrooms','total_rooms'

In [58]:
```
housing.hist(column=['households','housing_median_age', 'median_
             bins=50,
             figsize=(16,9))
plt.show()
```



In [59]:
```
housing['median_income'].hist()
```

Out[59]: <AxesSubplot:>



In [60]:
```
housing['income_cat'] = pd.cut(housing['median_income'],
                               bins=[0,1.5,3,4.5,6, np.inf],
                               labels = ['Very Low', 'Low', 'Med
housing
```

Out[60]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	h
0	-122.23	37.88	41.0	880.0	129.0	322.0	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	
...	...	...	...	...	...	...	
20635	-121.09	39.48	25.0	1665.0	374.0	845.0	
20636	-121.21	39.49	18.0	697.0	150.0	356.0	
20637	-121.22	39.43	17.0	2254.0	485.0	1007.0	
20638	-121.32	39.43	18.0	1860.0	409.0	741.0	
20639	-121.24	39.37	16.0	2785.0	616.0	1387.0	

20640 rows × 11 columns

Partitioning the dataset into separate training and test sets

1) Random partition

In [61]:
```
from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(housing, test_size = 0.2
```

In [62]:
```
train_set.head()
```

Out[62]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	h
14196	-117.03	32.71	33.0	3126.0	627.0	2300.0	
8267	-118.16	33.77	49.0	3382.0	787.0	1314.0	
17445	-120.48	34.66	4.0	1897.0	331.0	915.0	
14265	-117.11	32.69	36.0	1421.0	367.0	1418.0	
2271	-119.80	36.78	43.0	2382.0	431.0	874.0	

We assign 20% of the sample to the test and the remaining 80% to the training, BUT no guarantee both training and test sets have the same label/outcome distribution, especially when the dataset is small. Let's see...

In [63]:
```
def income_cat_proportions(data):
    return data['income_cat'].value_counts() / len(data)
```

In [64]:
```
compare_props = pd.DataFrame({
    'Overall': income_cat_proportions(housing),
    'Random' : income_cat_proportions(test_set)
}).sort_index()
```

In [65]:
```
compare_props['Rand %error'] = 100 * compare_props['Random'] /
```

In [67]:
```
compare_props
```

Out[67]:

	Overall	Random	Rand %error
Very Low	0.039826	0.040213	0.973236
Low	0.318847	0.324370	1.732260
Medium	0.350581	0.358527	2.266446
High	0.176308	0.167393	-5.056334
Very High	0.114438	0.109496	-4.318374

2) Stratified Sampling

In [68]:
```
# StratifiedShuffleSplit creates splits by preserving the same 
# for each target class as in the complete set.
from sklearn.model_selection import StratifiedShuffleSplit

splitObject = StratifiedShuffleSplit(n_splits=1, test_size=0.2,

train_index, test_index = next(splitObject.split(housing, housi

stratified_train_set = housing.loc[train_index]
stratified_test_set = housing.loc[test_index]

stratified_train_set.shape, stratified_test_set.shape, housing.
```

Out[68]: ((16512, 11), (4128, 11), (20640, 11))

In [69]:
```
compare_props['Stratified'] = stratified_test_set['income_cat']
```

In [70]:
```
compare_props['Stratified %error'] = 100 * compare_props['Strat
```

In [71]:
```
compare_props
```

Out[71]:

	Overall	Random	Rand %error	Stratified	Stratified %error
Very Low	0.039826	0.040213	0.973236	0.039729	-0.243309
Low	0.318847	0.324370	1.732260	0.318798	-0.015195
Medium	0.350581	0.358527	2.266446	0.350533	-0.013820
High	0.176308	0.167393	-5.056334	0.176357	0.027480
Very High	0.114438	0.109496	-4.318374	0.114583	0.127011

Prepare the data for Machine Learning algorithms

In [72]:
```
stratified_train_set
```

Out[72]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	h
16126	-122.47	37.79	52.0	437.0	105.0	194.0	
17709	-121.82	37.33	23.0	3279.0	647.0	2582.0	
2501	-120.38	36.76	25.0	991.0	272.0	941.0	
2123	-119.71	36.76	28.0	2675.0	527.0	1392.0	
2144	-119.76	36.77	36.0	2507.0	466.0	1227.0	
...	...	...	...	...	...	...	
3382	-118.27	34.25	35.0	779.0	143.0	371.0	
841	-122.08	37.59	16.0	1816.0	365.0	1367.0	
11749	-121.15	38.80	20.0	2104.0	370.0	745.0	
3940	-118.59	34.21	34.0	1943.0	320.0	895.0	
18827	-122.26	41.66	17.0	1885.0	350.0	953.0	

16512 rows × 11 columns

In [73]:
```
housing = stratified_train_set.drop('median_house_value',axis=1
housing_label = stratified_train_set['median_house_value'].copy
housing.columns
```

Out[73]:
```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms
',
       'total_bedrooms', 'population', 'households', 'median_incom
e',
       'ocean_proximity', 'income_cat'],
      dtype='object')
```

In [74]:
```
housing_label
```

Out[74]:
```
16126    500001.0
17709    175800.0
2501      58000.0
2123      72000.0
2144      72300.0
           ...
3382     230100.0
841      156300.0
11749    217500.0
3940     227700.0
18827     61400.0
Name: median_house_value, Length: 16512, dtype: float64
```

Identifying missing values

In [76]:
```
sample_incomplete_rows = housing[housing.isnull().any(axis=1)].
sample_incomplete_rows
```

Out[76]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	h
8383	-118.36	33.96	26.0	3543.0	NaN	2742.0	
10915	-117.87	33.73	45.0	2264.0	NaN	1970.0	
11311	-117.96	33.78	33.0	1520.0	NaN	658.0	
696	-122.10	37.69	41.0	746.0	NaN	387.0	
15137	-116.91	32.83	16.0	5203.0	NaN	2515.0	

Eliminating rows with missing values

In [78]:
```
sample_incomplete_rows.dropna(subset=['total_bedrooms'], axis=0
```

Out[78]:

longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	househ

Eliminating variables with missing values

In [79]:
```
sample_incomplete_rows.dropna(axis=1)
```

Out[79]:

	longitude	latitude	housing_median_age	total_rooms	population	households	medi
8383	-118.36	33.96	26.0	3543.0	2742.0	951.0	
10915	-117.87	33.73	45.0	2264.0	1970.0	499.0	
11311	-117.96	33.78	33.0	1520.0	658.0	242.0	
696	-122.10	37.69	41.0	746.0	387.0	161.0	
15137	-116.91	32.83	16.0	5203.0	2515.0	862.0	

Imputing missing values

1) Pandas

In [80]:
```
median = housing['total_bedrooms'].median()
sample_incomplete_rows['total_bedrooms'].fillna(median, inplace
sample_incomplete_rows
```

Out[80]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	h
8383	-118.36	33.96	26.0	3543.0	437.0	2742.0	
10915	-117.87	33.73	45.0	2264.0	437.0	1970.0	
11311	-117.96	33.78	33.0	1520.0	437.0	658.0	
696	-122.10	37.69	41.0	746.0	437.0	387.0	
15137	-116.91	32.83	16.0	5203.0	437.0	2515.0	

2) Scikit-Learn

The **SimpleImputer** class.

The SimpleImputer class provides basic strategies for imputing missing values. Missing values can be imputed with a provided constant value, or using the statistics (mean, median or most frequent) of each column in which the missing values are located.

In [147]:
```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values = np.nan, strategy = 'me
```

Remove the text attribute because median can only be calculated on numerical attributes:

In [148]:
```
housing_num = housing.select_dtypes(include=[np.number])
```

In [149]:
```
imputer.fit(housing_num)
```

Out[149]:
```
SimpleImputer(strategy='median')
```

In [150]:
```
imputer.statistics_
```

Out[150]:
```
array([-118.5    ,   34.26   ,   29.     , 2137.    ,  437.    , 1170
.    ,
        411.    ,    3.5375])
```

Transform the training set:

In [151]:
```
X = imputer.transform(housing_num)
X
```

Out[151]:
```
array([[-1.2247e+02,  3.7790e+01,  5.2000e+01, ...,  1.9400e+02,
         8.7000e+01,  2.8125e+00],
       [-1.2182e+02,  3.7330e+01,  2.3000e+01, ...,  2.5820e+03,
         6.3000e+02,  4.3782e+00],
       [-1.2038e+02,  3.6760e+01,  2.5000e+01, ...,  9.4100e+02,
         2.6200e+02,  1.8125e+00],
       ...,
       [-1.2115e+02,  3.8800e+01,  2.0000e+01, ...,  7.4500e+02,
         3.1400e+02,  4.1685e+00],
       [-1.1859e+02,  3.4210e+01,  3.4000e+01, ...,  8.9500e+02,
         3.0500e+02,  5.0462e+00],
       [-1.2226e+02,  4.1660e+01,  1.7000e+01, ...,  9.5300e+02,
         3.2800e+02,  2.1607e+00]])
```

Scikit-Learn API is organized around a bunch of design principles:

- **Consistency**: all object share a consistent and simple interface
    1. **Estimator**: object that can estimate some parameters. Estimation performed by the method *fit* which takes only a dataset as parameter, any other parameter is an hyperparameter
    2. **Transformers**: some estimators transform a dataset. The transformation is performed by the method *transform* with the dataset to transform as a parameter. It returns the transformed dataset. There is a convenient *fit_transform* method, which is optimized and runs much faster
    3. **Predictors**: some estimator are able to make predictions. A predictor has a method *predict* that takes a dataset of new samples and returns the corresponding predictions
- **Inspection**: all hyperparameter are accessible via instance variable as well as the learned parameters (underscore suffix)
- **Nonproliferation of classes**: datasets are Numpy arrays or Scipy sparse matrices. No homemade classes
- **Composition**: existing building block are reusable
- **Sensible defaults**: reasonable default values.

In [152]:
```
imputer.strategy
```
Out[152]:
```
'median'
```

In [153]:
```
housing_trasformed = pd.DataFrame(X, columns= housing_num.colum
```

Encoding nominal features

In [154]:
```python
housing_cat = housing[['ocean_proximity']]
housing_cat
```

Out[154]:

	ocean_proximity
16126	NEAR BAY
17709	<1H OCEAN
2501	INLAND
2123	INLAND
2144	INLAND
...	...
3382	<1H OCEAN
841	NEAR BAY
11749	INLAND
3940	<1H OCEAN
18827	INLAND

16512 rows × 1 columns

To convert categorical features to such integer codes, we can use the **OrdinalEncoder**. This estimator transforms each categorical feature to one new feature of integers (0 to n_categories - 1)

In [155]:
```python
from sklearn.preprocessing import OrdinalEncoder
```

In [156]:
```python
ordinal_encoder = OrdinalEncoder()
housing_cat_encoded = ordinal_encoder.fit_transform(housing_cat
housing_cat_encoded[:10]
```

Out[156]:
```
array([[3.],
       [0.],
       [1.],
       [1.],
       [1.],
       [0.],
       [4.],
       [1.],
       [0.],
       [1.]])
```

Such integer representation can, however, not be used directly with all scikit-learn estimators, as these expect continuous input, and would interpret the categories as being ordered, which is often not desired. A common workaround to this issue is to use a technique called **one-hot encoding**

In [157]:
```
from sklearn.preprocessing import OneHotEncoder

cat_encoder = OneHotEncoder()
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)

```

By default, the `OneHotEncoder` class returns a sparse array, but we can convert it to a dense array if needed by calling the `toarray()` method:

In [158]:
```
housing_cat_1hot.toarray()[0:6]
```

Out[158]:
```
array([[0., 0., 0., 1., 0.],
       [1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [1., 0., 0., 0., 0.]])
```

Alternatively, you can set `sparse=False` when creating the `OneHotEncoder` :

In [56]:
```
cat_encoder = OneHotEncoder(sparse = False)
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
type(housing_cat_1hot)
```

Out[56]:
```
numpy.ndarray
```

In [57]:
```
cat_encoder.categories_
```

Out[57]:
```
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN']
,
       dtype=object)]
```

Attributes creation

Let's create a new transformer to add extra attributes. All you need is to convert an existing Python function into a transformer to assist in data cleaning or processing. You can implement a transformer from an arbitrary function with the class **FunctionTransformer**

In [92]:
```
housing.columns
```

Out[92]:
```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms
',
       'total_bedrooms', 'population', 'households', 'median_incom
e',
       'ocean_proximity', 'income_cat'],
      dtype='object')
```

In [100]:
```
rooms_ix, bed_rooms_ix, population_ix, household_ix = [
    list(housing.columns).index(col) for col in ['total_rooms',
]

def add_extra_features(X):
    roomsXhouse = X[:, rooms_ix] / X[:, household_ix]
    popXhouse = X[:,population_ix] / X[:,household_ix]
    return np.c_[X,roomsXhouse, popXhouse]

from sklearn.preprocessing import FunctionTransformer
attr_adder = FunctionTransformer(add_extra_features, validate =

housinhg_extra = attr_adder.fit_transform(housing.values)
```

In [101]:
```
housinhg_extra_df = pd.DataFrame(housinhg_extra,
                                 columns = list(housing.columns)
housinhg_extra_df.head()
```

Out[101]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	house
0	-122.47	37.79	52.0	437.0	105.0	194.0	
1	-121.82	37.33	23.0	3279.0	647.0	2582.0	
2	-120.38	36.76	25.0	991.0	272.0	941.0	
3	-119.71	36.76	28.0	2675.0	527.0	1392.0	
4	-119.76	36.77	36.0	2507.0	466.0	1227.0	

Attribute or feature scaling

ML algorithms don't perform well when the numerical attributes have very different scales. Two classes to report all the attributes to the same scale:

- **Mix-max scaling**: SkLearn provides the transformer **MinMaxScaler**
- **Standardization**: SkLearn provides the transformer **StandardScaler**

Transformation Pipeline

Since there are many transformation steps that need to be executed in the right order, need a way to automatically create this sequence of transformation. SkLearn provides the **Pipeline** class. This class takes an arbitrary number of SkLearn transformers, as a list of name/estimator pairs. When you call the method *fit()*, it runs the method *fit_transform()* of each element in list, sequentially

Now let's build a pipeline for preprocessing the numerical attributes:

In [103]:

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy = 'median')),
    ('attribs_adder', FunctionTransformer(add_extra_features, v
    ('std_scaler', StandardScaler())
])

housing_num_tr = num_pipeline.fit_transform(housinhg_extra_df)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent c
all last)
<ipython-input-103-b29347a14477> in <module>
      8 ])
      9
---> 10 housing_num_tr = num_pipeline.fit_transform(housinhg_extra
_df)

/opt/anaconda3/lib/python3.8/site-packages/sklearn/pipeline.py in
fit_transform(self, X, y, **fit_params)
    376         """
    377         fit_params_steps = self._check_fit_params(**fit_pa
rams)
--> 378         Xt = self._fit(X, y, **fit_params_steps)
    379
    380         last_step = self._final_estimator

/opt/anaconda3/lib/python3.8/site-packages/sklearn/pipeline.py in
_fit(self, X, y, **fit_params_steps)
    301                 cloned_transformer = clone(transformer)
    302             # Fit or load from cache the current transform
er
--> 303             X, fitted_transformer = fit_transform_one_cach
ed(
    304                 cloned_transformer, X, y, None,
    305                 message_clsname='Pipeline',
```

```
/opt/anaconda3/lib/python3.8/site-packages/joblib/memory.py in __c
all__(self, *args, **kwargs)
    350
    351     def __call__(self, *args, **kwargs):
--> 352         return self.func(*args, **kwargs)
    353
    354     def call_and_shelve(self, *args, **kwargs):

/opt/anaconda3/lib/python3.8/site-packages/sklearn/pipeline.py in
_fit_transform_one(transformer, X, y, weight, message_clsname, mes
sage, **fit_params)
    752     with _print_elapsed_time(message_clsname, message):
    753         if hasattr(transformer, 'fit_transform'):
--> 754             res = transformer.fit_transform(X, y, **fit_pa
rams)
    755         else:
    756             res = transformer.fit(X, y, **fit_params).tran
sform(X)

/opt/anaconda3/lib/python3.8/site-packages/sklearn/base.py in fit_
transform(self, X, y, **fit_params)
    697         if y is None:
    698             # fit method of arity 1 (unsupervised transfor
mation)
--> 699             return self.fit(X, **fit_params).transform(X)
    700         else:
    701             # fit method of arity 2 (supervised transforma
tion)

/opt/anaconda3/lib/python3.8/site-packages/sklearn/impute/_base.py
in fit(self, X, y)
    286         self : SimpleImputer
    287         """
--> 288         X = self._validate_input(X, in_fit=True)
    289
    290         # default fill_value is 0 for numerical input and
"missing_value"

/opt/anaconda3/lib/python3.8/site-packages/sklearn/impute/_base.py
in _validate_input(self, X, in_fit)
    258                 new_ve = ValueError("Cannot use {} strateg
y with non-numeric "
    259
"data:\n{}".format(self.strategy, ve))
--> 260                 raise new_ve from None
    261             else:
    262                 raise ve

ValueError: Cannot use median strategy with non-numeric data:
could not convert string to float: 'NEAR BAY'
```

In [62]:
```
housing_num_tr.shape
```
Out[62]: (16512, 10)

If you have a Pandas DataFrame it is now preferable to use the **ColumnTransformer** class that was introduced in SkLearn 0.20.

In [104]:
```
from sklearn.compose import ColumnTransformer
```

In [106]:
```
num_attribs = list(housing_num)
cat_attribs = ['ocean_proximity']

full_pipeline = ColumnTransformer([
    ('num', num_pipeline, num_attribs),
    ('cat', OneHotEncoder(), cat_attribs)
])

housing_final = full_pipeline.fit_transform(housing)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent c
all last)
<ipython-input-106-1a782cbf41f5> in <module>
----> 1 num_attribs = list(housing_num)
      2 cat_attribs = ['ocean_proximity']
      3 
      4 full_pipeline = ColumnTransformer([
      5     ('num', num_pipeline, num_attribs),

NameError: name 'housing_num' is not defined
```

In [65]:
```
housing_final.shape, housing.values.shape
```
Out[65]: ((16512, 15), (16512, 10))

In [68]:
```
housing_final[0:6]
```

Out[68]:
```
array([[-1.44853942,  1.00749903,  1.85042332, -1.00225667, -1.026
08971,
        -1.09987832, -1.0732447 , -0.55467031, -0.17413103, -0.074
71997,
         0.        ,  0.        ,  0.        ,  1.        ,  0.
],
       [-1.12372271,  0.79233009, -0.44832649,  0.28300714,  0.252
2925 ,
         1.0191734 ,  0.32798234,  0.26444129, -0.09511204,  0.086
67068,
         1.        ,  0.        ,  0.        ,  0.        ,  0.
],
       [-0.40412878,  0.52570771, -0.28979202, -0.75171615, -0.632
19704,
        -0.43700912, -0.62165219, -1.0778303 , -0.71341051,  0.042
89592,
         0.        ,  1.        ,  0.        ,  0.        ,  0.
],
       [-0.06931772,  0.52570771, -0.05199032,  0.00985466, -0.030
74415,
        -0.03680296,  0.04670472, -0.81713968, -0.12571786, -0.036
55167,
         0.        ,  1.        ,  0.        ,  0.        ,  0.
],
       [-0.09430362,  0.5303853 ,  0.58214756, -0.06612152, -0.174
62112,
        -0.18321985, -0.07458012, -0.56905721, -0.05847994, -0.043
73597,
         0.        ,  1.        ,  0.        ,  0.        ,  0.
],
       [ 0.72023674, -0.84950247,  1.21628544, -0.46770993, -0.613
32793,
        -0.54881839, -0.56488056,  0.87481206,  0.12047978, -0.019
45555,
         1.        ,  0.        ,  0.        ,  0.        ,  0.
]])
```

Extra material

Model persistence using joblib

In [105]:
```
my_model = full_pipeline
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-105-c829ee97a232> in <module>
----> 1 my_model = full_pipeline

NameError: name 'full_pipeline' is not defined
```

In [71]:
```
#from sklearn.externals import joblib
import joblib
joblib.dump(my_model, "full_pipeline.pkl") # DIFF
my_model_loaded = joblib.load("full_pipeline.pkl") # DIFF
```

In [72]:
```
my_model_loaded
```

Out[72]:
```
ColumnTransformer(transformers=[('num',
                                 Pipeline(steps=[('imputer',
                                                  SimpleImputer(strategy='median')),
                                                 ('attribs_adder',
                                                  FunctionTransformer(func=<function add_extra_features at 0x7fe58e0aaa60>)),
                                                 ('std_scaler',
                                                  StandardScaler())]),
                                 ['longitude', 'latitude', 'housing_median_age',
                                  'total_rooms', 'total_bedrooms', 'population',
                                  'households', 'median_income']),
                                ('cat', OneHotEncoder(), ['ocean_proximity'])])
```

Some further examples of using pickle

In [107]:
```
import pandas as pd
import pickle
```

In [109]:
```
print ('convert: csv -> pkl')
datimaggio18 = pd.read_csv('01_2018.csv', delimiter=';', header
datimaggio18
```

```
convert: csv -> pkl
```

Out[109]:

	Bicicletta	Tipo_bici	Cliente	Data_riferimento_prelievo	Data_prelievo	Ora_prelievo
0	7486	Bike	141116	01/01/18	01/01/18 07:18	7
1	8279	Bike	265468	01/01/18	01/01/18 07:35	7
2	1284	Bike	232605	01/01/18	01/01/18 07:49	7
3	7411	Bike	21489	01/01/18	01/01/18 07:56	7
4	1730	Bike	220370	01/01/18	01/01/18 07:58	7
...	...	...	...	...	...	...
250156	7780	Bike	308325	31/01/18	01/02/18 00:37	0
250157	3562	Bike	163545	31/01/18	01/02/18 00:42	0
250158	11108	eBike	81098	31/01/18	01/02/18 00:52	0
250159	7828	Bike	17302	31/01/18	01/02/18 00:58	0
250160	6992	Bike	234044	31/01/18	01/02/18 00:59	0

250161 rows × 52 columns

In [110]:
```
pickle.dump(datimaggio18, open("datimaggio18.pkl", "wb"))
```

In [111]:
```
bikemi = pd.read_pickle('datimaggio18.pkl'.format(5,2018))
```

In [78]:
```
print('Number of rents')
len(bikemi)
```

```
Number of rents
```

Out[78]: 250161

In [79]:
```
bikemi['Cliente'].nunique()
```

Out[79]: 24944

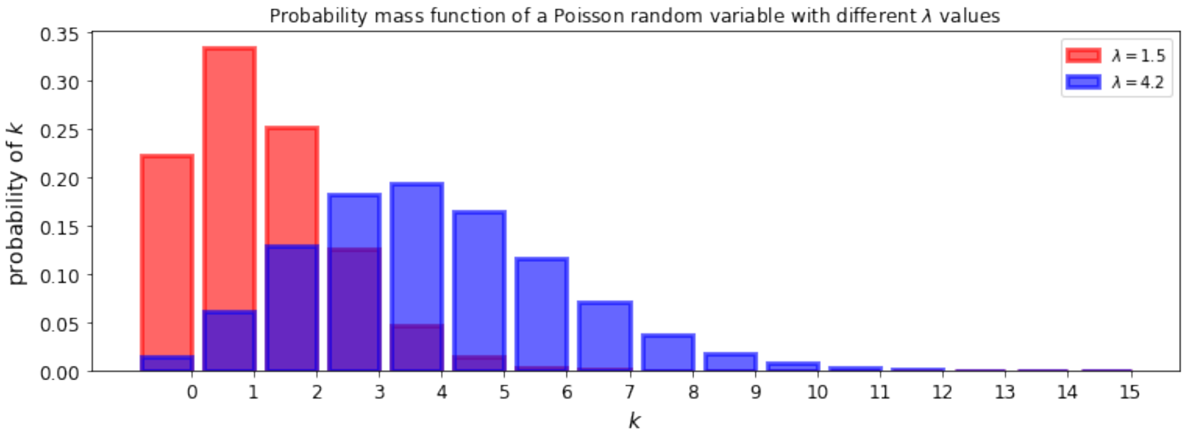
Let's see a bit of statistical visualization tools

In [113]:

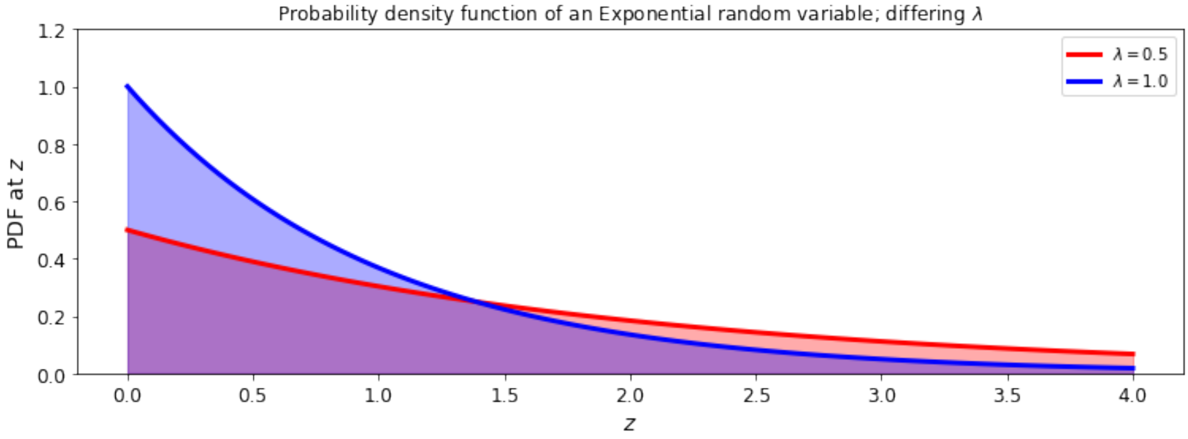
```
from IPython.core.pylabtools import figsize
import numpy as np
from matplotlib import pyplot as plt
figsize(12.5, 4)

import scipy.stats as stats
a = np.arange(16)
poi = stats.poisson
lambda_ = [1.5, 4.25]
colours = ["red", "blue"]

plt.bar(a, poi.pmf(a, lambda_[0]), color=colours[0],
        label="$\lambda = %.1f$" % lambda_[0], alpha=0.60,
        edgecolor=colours[0], lw="3")

plt.bar(a, poi.pmf(a, lambda_[1]), color=colours[1],
        label="$\lambda = %.1f$" % lambda_[1], alpha=0.60,
        edgecolor=colours[1], lw="3")

plt.xticks(a + 0.4, a)
plt.legend()
plt.ylabel("probability of $k$")
plt.xlabel("$k$")
plt.title("Probability mass function of a Poisson random variab
$\lambda$ values")
```

Out[113]:

```
Text(0.5, 1.0, 'Probability mass function of a Poisson random vari
able with different $\\lambda$ values')
```



In [114]:
```
a = np.linspace(0, 4, 100)
expo = stats.expon
lambda_ = [0.5, 1]

for l, c in zip(lambda_, colours):
    plt.plot(a, expo.pdf(a, scale=1. / l), lw=3,
                color=c, label="$\lambda = %.1f$" % l)
    plt.fill_between(a, expo.pdf(a, scale=1. / l), color=c, alp

plt.legend()
plt.ylabel("PDF at $z$")
plt.xlabel("$z$")
plt.ylim(0, 1.2)
plt.title("Probability density function of an Exponential randor
 differing $\lambda$");
```



In [82]:
```
figsize(12.5, 3.5)
count_data = np.loadtxt("/Users/giancarlomanzi/Documents/Box Sy
n_count_data = len(count_data)
plt.bar(np.arange(n_count_data), count_data, color="#348ABD")
plt.xlabel("Time (days)")
plt.ylabel("count of text-msgs received")
plt.title("Did the user's texting habits change over time?")
plt.xlim(0, n_count_data);
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent c
all last)
<ipython-input-82-a439bff65e7e> in <module>
      1 figsize(12.5, 3.5)
----> 2 count_data = np.loadtxt("/Users/giancarlomanzi/Documents/B
ox Sync BackUp PC Lavoro 24062015/documenti/Ricerca/ALTERNATIVE ER
ASMUS PROJECT/NUOVO PROGETTO DI VISITING/Lectures/Topic 2 - Introd
uction to Python and the Anaconda-Jupyter environment - 3 hours/tx
tdata.csv")
      3 n_count_data = len(count_data)
      4 plt.bar(np.arange(n_count_data), count_data, color="#348AB
D")
      5 plt.xlabel("Time (days)")
```

```
/opt/anaconda3/lib/python3.8/site-packages/numpy/lib/npyio.py in l
oadtxt(fname, dtype, comments, delimiter, converters, skiprows, us
ecols, unpack, ndmin, encoding, max_rows, like)
   1040             fname = os_fspath(fname)
   1041         if _is_string_like(fname):
-> 1042             fh = np.lib._datasource.open(fname, 'rt',
encoding=encoding)

   1043             fencoding = getattr(fh, 'encoding', 'latin1')
   1044             line_iter = iter(fh)

/opt/anaconda3/lib/python3.8/site-packages/numpy/lib/_datasource.p
y in open(path, mode, destpath, encoding, newline)
    191
    192     ds = DataSource(destpath)
--> 193     return ds.open(path, mode, encoding=encoding, newline=
newline)
    194
    195

/opt/anaconda3/lib/python3.8/site-packages/numpy/lib/_datasource.p
y in open(self, path, mode, encoding, newline)
    530                                       encoding=encoding, n
ewline=newline)
    531         else:
--> 532             raise FileNotFoundError(f"{path} not found.")
    533
    534

FileNotFoundError: /Users/giancarlomanzi/Documents/Box Sync BackUp
PC Lavoro 24062015/documenti/Ricerca/ALTERNATIVE ERASMUS PROJECT/N
UOVO PROGETTO DI VISITING/Lectures/Topic 2 - Introduction to Pytho
n and the Anaconda-Jupyter environment - 3 hours/txtdata.csv not f
ound.
```

Pipelines in text mining/natural language processing

- We spend a lot of time in pre-processing and cleaning data
- Therefore we need to create multipurpose software objects to be used in different situations.
- For this we can use the Pipeline tool in scikit-learn.
- It is composed by *transformers* (tools for transforming data, for example to normalize a variable) and *estimators* (for example a fitting or predicting tool).
- All transformers and estimators in scikit-learn are implemented as Python *classes*, each with their own attributes and methods.
- We use *inherited* classes from scikit-learn to implement our own class.

Pipeline

Step	Task	Data
Pre-processing	Cleaning data and extracting features	Fit on the training set and apply to the whole dataset
Training	Tuning model parameters	Training set
Validation	Selecting the best model	Validation set
Inference	Evaluating the final model	Holdout set

In [115]:

```
# Example of inheritance
from sklearn.preprocessing import OneHotEncoder
#Some data:
X = [[0, 0, 3], [1, 1, 0], [0, 2, 1], [1, 0, 2]]
#Initializing an object of class OneHotEncoder
# Here we are "inheriting" classes from OneHotEncoder into the
one_hot_enc = OneHotEncoder( sparse = True )

#Calling methods on our OneHotEncoder object
one_hot_enc.fit( X ) #returns nothing
transformed_data = one_hot_enc.transform(X).toarray() #returns
#fit_transformed_data = one_hot_enc.transform( X ) #returns som
```

In [116]:

```
print(pd.DataFrame(X))
#Comments: The first column takes on 2 values, the second 3 and
```

```
   0  1  2
0  0  0  3
1  1  1  0
2  0  2  1
3  1  0  2
```

In [117]:

```
print(transformed_data)
#Comments: the first two columns express the binary coding of t
# the next three columns express the binary coding of the secon
# The next four columns express the binary coding of the third
```

```
[[1. 0. 1. 0. 0. 0. 0. 0. 1.]
 [0. 1. 0. 1. 0. 1. 0. 0. 0.]
 [1. 0. 0. 0. 1. 0. 1. 0. 0.]
 [0. 1. 1. 0. 0. 0. 0. 1. 0.]]
```

Pipelines in text mining/natural language processing (2)

- Our own trasformer will be formed by inheriting from some other scikit-learn class.
- See a tutorial here [https://www.programiz.com/python-programming/class (https://www.programiz.com/python-programming/class)](https://www.programiz.com/python-programming/class) about classes and objects in python and a tutorial here [https://www.programiz.com/python-programming/inheritance (https://www.programiz.com/python-programming/inheritance)](https://www.programiz.com/python-programming/inheritance) about inheritance.
- The base classes inherited from scikit-learn are TransformerMixin ([https://scikit-learn.org/stable/modules/generated/sklearn.base.TransformerMixin.html (https://scikit-learn.org/stable/modules/generated/sklearn.base.TransformerMixin.html)](https://scikit-learn.org/stable/modules/generated/sklearn.base.TransformerMixin.html)) and BaseEstimator ([https://scikit-learn.org/stable/modules/generated/sklearn.base.BaseEstimator.html (https://scikit-learn.org/stable/modules/generated/sklearn.base.BaseEstimator.html)](https://scikit-learn.org/stable/modules/generated/sklearn.base.BaseEstimator.html)).

In [118]:
```
import numpy as np
import pandas as pd
```

In [121]:
```
# Load Data
pd.set_option('display.max_colwidth', None)
BikeSent = pd.read_csv("BikeMiSentiment2019_UTF-8.csv", sep=';'
BikeSent
```

Out[121]:

	v1	v2
0	positive	When I had problems with the return of the bike, the assistance was very kind, but could not solve the problem quickly nor prevent me from being charged for the rental because I had exceeded half an hour (not because of the route I took carried out, but due to the impossibility of hanging up the bike in 2 different stations). It would be desirable to be able to solve problems more efficiently or at least not to charge the user in the event of a reported malfunction. For the rest, when the service works, it's really practical and useful.\t
1	positive	more electric bikes. often even if present they are not available when there are few, why?\t
2	negative	pay more attention to stations that very often are without bicycles or full and do not allow their repositioning\t
3	positive	essential to insert bikes with child seats\t
4	positive	extension completed at train and metro stations not yet served\t
...	...	...
995	negative	Main problem I think is the maintenance of traditional bikes, often you are forced to change bikes several times before finding a functioning one\t
996	positive	I feel good but without a credit card you can't even buy a day card, it doesn't seem right because students like me often only have a prepaid card\t
997	positive	I don't have any suggestions at the moment. the comment, thank you for the excellent service provided.\t
998	positive	I would like it if the number of red ebikes increased considerably\t
999	positive	need more maintenance, stations in the center with too many bikes\t

1000 rows × 2 columns

In [122]:
```
# Rename columns
BikeSent.columns = ["target", "text"]
BikeSent.head()
```

Out[122]:

	target	text
0	positive	When I had problems with the return of the bike, the assistance was very kind, but could not solve the problem quickly nor prevent me from being charged for the rental because I had exceeded half an hour (not because of the route I took carried out, but due to the impossibility of hanging up the bike in 2 different stations). It would be desirable to be able to solve problems more efficiently or at least not to charge the user in the event of a reported malfunction. For the rest, when the service works, it's really practical and useful.\t
1	positive	more electric bikes. often even if present they are not available when there are few, why?\t
2	negative	pay more attention to stations that very often are without bicycles or full and do not allow their repositioning\t
3	positive	essential to insert bikes with child seats\t
4	positive	extension completed at train and metro stations not yet served\t

In [123]:
```
# Encode categories
BikeSent['target'] = np.where(BikeSent['target']=='positive',1,
BikeSent.head()
```

Out[123]:

	target	text
0	1	When I had problems with the return of the bike, the assistance was very kind, but could not solve the problem quickly nor prevent me from being charged for the rental because I had exceeded half an hour (not because of the route I took carried out, but due to the impossibility of hanging up the bike in 2 different stations). It would be desirable to be able to solve problems more efficiently or at least not to charge the user in the event of a reported malfunction. For the rest, when the service works, it's really practical and useful.\t
1	1	more electric bikes. often even if present they are not available when there are few, why?\t
2	0	pay more attention to stations that very often are without bicycles or full and do not allow their repositioning\t
3	1	essential to insert bikes with child seats\t
4	1	extension completed at train and metro stations not yet served\t

In [124]:
```
# split the sample in train (used also for cross-validation) +
from sklearn.model_selection import train_test_split
X = BikeSent[['text']]
y = BikeSent['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_
```

In [125]:
```
X_train
```

Out[125]:

	text
716	Luckily I have tried the new bicycle models a few times and they are definitely uncomfortable. I think there is a design error because the saddle is too far forward and you have difficulty pedaling. I hope you realize this before increasing the number of bikes you buy\t
351	Improve the bike pickup and storage system. For older people they are too heavy to lift\t
936	I kindly ask you to make the stall n. 151 Balilla - Tibaldi. Sometimes it is uninhabitable and there are few bicycles available or they are generally few or poorly functioning (eg deflated wheels, poorly functioning brakes, gearshift changes).\t
256	bikes should be maintained much, much better, often with badly maintained bicycles and without brakes or even for the electrics that the battery does not work\t
635	Increase maintenance\t
...	...
106	A really useful service, I hope in the possibility of using 24 hours a day, especially for us young people it can be very useful at night when the vehicles are almost zero and you are forced to use taxis.\t
270	only problem to report too often the stalls do not record the correct establishment of the bike and you risk icorrerere nela penalty\t
860	Some discounts for the renewal of the subscription. The offers seem to me always and only for the new subscribers. In addition, a few more conventions for Bikemi subscribers who give discounts elsewhere.\t
435	the service is smart but is very limited by the location of the stations. They are all a center. There isn't one in Stazione Lambrate or the eastern suburbs.\t
102	good\t

900 rows × 1 columns

Custom Transformers

Cleaning Text

- We create here our own transformer (which will be a class) inheriting the TransformerMixin and the BaseEstimator classes from scikit-learn

In [126]:

```
from sklearn.base import BaseEstimator
from sklearn.base import TransformerMixin
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import SnowballStemmer

# Custom Transformer (Inheriting from classes)
class CleanText( BaseEstimator, TransformerMixin ):

    # Class Constructor
    # The class constructor is formed by a function with double
    # these are called 'special functions' as they have special
    # In particular the '__init__' gets called whenever
    # a new object of that class is instantiated,
    # and are used to initialize all the necessary variables.
    # In this example we initialize the language variable 'lang
    # and pick the SnowballStemmer as the default stemmer.
    def __init__( self, lang = "english"):
        self.lang = lang
        self.stemmer = SnowballStemmer(self.lang)

    # The 'fit' method here is used to instantiate the class on
    # and return the object itself
    def fit( self, X, y = None ):
        return self

    # Custom function: this applies the stemmer just created in
    # part to the 'self' variable
    def clean( self, x ):
        words   = [self.stemmer.stem(word) for word in word_tok
        return " ".join(words)

    # Method that describes what we need this transformer to do
    # in the 'text' column in the data frame.
    # This will be used later on in the usage of the custom tra
    # within the pipeline.
    def transform( self, X, y = None ):
        return X["text"].apply(self.clean)
```

Feature extraction

In [127]:
```
# Custom Transformer: same parts as the previous custom transfo
# This one will be used for feature extraction

class CustomFeatures( BaseEstimator, TransformerMixin ):

    # Class Constructor
    def __init__( self ):
        return

    # Return self nothing else to do here
    def fit( self, X, y = None ):
        return self

    # Method that describes what we need this transformer to do
    # returning length, digits and punctuations in the 'text' c
    def transform( self, X, y = None ):
        f           = pd.DataFrame()
        f['len']    = X['text'].str.len()
        f['digits'] = X['text'].str.findall(r'\d').str.len()
        f['punct']  = X['text'].str.findall(r'[^a-zA-Z\d\s:]').
        return f[['len','digits','punct']]
```

Pipeline usage

Pipeline for data pre processing

In [129]:
```
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
# FeatureUnion combines two or more pipelines or transformers
# and is very fast!
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.preprocessing import StandardScaler
# Our first pipeline called 'pipe' will be formed by three 'ste
# 1)"extract" which in turns is formed through FeatureUnion whi
# put together two parts:
# "terms" (formed by a pipeline with the CleanText() transforme
# and the TfidVectorize text vectorizing transformer from sciki
# (formed by the CustomFeatures transformer we created above);
# 2) "select", formed by the scikit-learn transformer method "S
# selection with a chi squared score function;
# 3) "scale", same as 2) using the StandardScaler method from s
# The whole pipeline will be used as pre-processing task in cla
pipe = Pipeline([("extract", FeatureUnion([("terms", Pipeline([

                                    ("custom", CustomFea
                  ("select", SelectKBest(score_func = chi2)),
                  ("scale", StandardScaler(with_mean = False))])
```

Classifier implemented through pipelines: Logistic Model

In [131]:
```
# Logistic Model
from sklearn.linear_model import LogisticRegression
pipe_logistic = Pipeline([('pre_process', pipe),
                          ('classify', LogisticRegression(max_i
```

In [132]:
```
# Fit on training
pipe_logistic.fit(X_train, y_train)
```

Out[132]:
```
Pipeline(steps=[('pre_process',
                 Pipeline(steps=[('extract',
                                  FeatureUnion(transformer_list=[(
'terms',

Pipeline(steps=[('clean',

CleanText()),

('tfidf',

TfidfVectorizer())])),
                                                                  (
'custom',

CustomFeatures())])),
                                 ('select',
                                  SelectKBest(score_func=<function
chi2 at 0x7f83b3de78b0>)),
                                 ('scale', StandardScaler(with_mea
n=False))])),
                ('classify', LogisticRegression(max_iter=10000, to
l=0.1))])
```

In [133]:
```
# Evaluate on test
# The F1 score can be interpreted as a weighted average of the 
# where an F1 score reaches its best value at 1 and worst score
#The relative contribution of precision
# and recall to the F1 score are equal. The formula for the F1 
# F1 = 2 * (precision * recall) / (precision + recall)
from sklearn.metrics import f1_score
y_pred = pipe_logistic.predict(X_test)
f1_score(y_test, y_pred)
```

Out[133]:
```
0.7972972972972973
```

In [98]:
```
# we can classify new messages!
msg = pd.DataFrame(columns = ["text"],
                   data    = ["The bikes are heavy and unwieldy

pipe_logistic.predict(msg)
```

Out[98]: array([1])

In [99]:
```
# we can classify new messages!
#msg = pd.DataFrame(columns = ["text"],
                   #data    = ["REMINDER FROM O2: To get 2.50 p

msg = pd.DataFrame(columns = ["text"],
                   data    = ["Satisfied"])


pipe_logistic.predict(msg)
```

Out[99]: array([1])

Using bi-grams

In [100]:
```
# extract features
pipe_extract = FeatureUnion([("terms", Pipeline([('clean', Clea
                                                  ('tfidf', Tfid
                             ("custom", CustomFeatures())])

# select and scale features
pipe_select_scale = Pipeline([("select", SelectKBest(score_func
                              ("scale", StandardScaler(with_mea
```

In [101]:
```
# extract features
# you can also use bi-grams:
X_extract = pipe_extract.set_params(terms__tfidf__ngram_range =
```

In [102]:
```
print(X_extract)
```

```
  (0, 697)	0.07090144070985319
  (0, 745)	0.17483279959508186
  (0, 785)	0.043180618930796104
  (0, 814)	0.17483279959508186
  (0, 1163)	0.1648630344692586
  (0, 1895)	0.14402927962636455
  (0, 1900)	0.17483279959508186
  (0, 1940)	0.1648630344692586
  (0, 1942)	0.17483279959508186
  (0, 2001)	0.1523026158602164
  (0, 2004)	0.17483279959508186
  (0, 2318)	0.1648630344692586
  (0, 2320)	0.17483279959508186
```

```
  (0, 2605)	0.1307761823598249
  (0, 2609)	0.17483279959508186
  (0, 2799)	0.157789373540365
  (0, 2800)	0.17483279959508186
  (0, 3315)	0.13525918980549953
  (0, 3323)	0.17483279959508186
  (0, 3509)	0.09334204887662222
  (0, 3529)	0.12370252143093134
  (0, 4020)	0.17483279959508186
  (0, 4021)	0.17483279959508186
  (0, 4343)	0.1523026158602164
  (0, 4346)	0.17483279959508186
  :	:
  (898, 1293)	0.14945135091427067
  (898, 1311)	0.25345986443799984
  (898, 2151)	0.23900642479096382
  (898, 2153)	0.25345986443799984
  (898, 3706)	0.20880316378705438
  (898, 3708)	0.25345986443799984
  (898, 3882)	0.20880316378705438
  (898, 3884)	0.25345986443799984
  (898, 3923)	0.1771738498926844
  (898, 3935)	0.23900642479096382
  (898, 4862)	0.1334273238341794
  (898, 4894)	0.25345986443799984
  (898, 6233)	0.09131789001952145
  (898, 6325)	0.25345986443799984
  (898, 6480)	0.23900642479096382
  (898, 6481)	0.25345986443799984
  (898, 6700)	0.0829510509697508
  (898, 6721)	0.20404322788705004
  (898, 6857)	0.25345986443799984
  (898, 6858)	0.25345986443799984
  (898, 6955)	0.18673654039843596
  (898, 8149)	158.0
  (898, 8151)	4.0
  (899, 3095)	1.0
  (899, 8149)	5.0
```

In [103]:

```
# extract all features
X_select_scale = pipe_select_scale.set_params(select__k = 500).
print(X_select_scale)
```

```
  (0, 222)	1.7463975570695727
  (0, 224)	4.868401476553422
  (0, 332)	3.3310922794931095
  (0, 391)	3.8799422337719514
  (0, 457)	4.664456373753558
  (0, 498)	2.431404727966195
  (0, 499)	0.6518039044082895
  (1, 215)	1.955545214342361
  (1, 216)	4.391664155129969
  (1, 498)	0.7954037771785323
  (1, 499)	0.32590195220414475
```

```
  (2, 190)      3.9374599587618193
  (2, 415)      2.9160268437218404
  (2, 498)      2.205437745813203

  (2, 499)      2.607215617633158
  (3, 30)       3.467617207975343
  (3, 35)       19.81764758098014
  (3, 44)       4.029061322449084
  (3, 254)      17.490606015887288
  (3, 283)      21.01627223472825
  (3, 294)      1.5146934286923344
  (3, 295)      9.407246305477612
  (3, 487)      4.614820388219279
  (3, 498)      1.4371500064930298
  (3, 499)      0.6518039044082895
  :     :
  (893, 494)    2.240618278439294
  (893, 498)    1.6992717057905007
  (893, 499)    0.6518039044082895
  (894, 347)    2.7889851989069125
  (894, 498)    0.831558494323011
  (895, 211)    2.8248704708645893
  (895, 397)    1.0910163308028165
  (895, 498)    1.8529292536545352
  (895, 499)    0.9777058566124343
  (896, 72)     24.41599739893815
  (896, 294)    1.6522845847666836
  (896, 309)    16.430593476368912
  (896, 352)    2.675593080916396
  (896, 380)    4.5285814862791804
  (896, 386)    13.6708637361617
  (896, 498)    1.202144345053918
  (897, 13)     2.5695129210768552
  (897, 191)    6.145400961179911
  (897, 498)    1.8438905743684155
  (897, 499)    1.303607808816579
  (898, 397)    1.4938347656108477
  (898, 498)    1.4281113272069101
  (898, 499)    1.303607808816579
  (899, 193)    11.497860975131363
  (899, 498)    0.04519339643059842
```

Using cross-validation with parameters (grid)

In [104]:

```
# Select best hyperparameters by cross validation
from sklearn.model_selection import GridSearchCV

# Model
logistic = LogisticRegression(max_iter=10000, tol=0.1, solver='

# Parameters: (np.logspace returns numbers spaced evenly on a l
param_logistic = {
    'C': np.logspace(-4, 4, 4)
}

# For an explanation of the 'C' parameter in scikit-learn logis
# https://stackoverflow.com/questions/22851316/what-is-the-inve
# C= 1/\lambda where \lambda can be assimilated to the regulati
# you probably have seen in the lasso regression
# Grid Search
cv_logistic = GridSearchCV(logistic, param_logistic, cv=10, sco
cv_logistic.fit(X_select_scale, y_train)
```

Out[104]:

```
GridSearchCV(cv=10, estimator=LogisticRegression(max_iter=10000, t
ol=0.1),
             param_grid={'C': array([1.00000000e-04, 4.64158883e-0
2, 2.15443469e+01, 1.00000000e+04])},
             scoring='f1')
```

In [105]:

```
# See https://scikit-learn.org/stable/modules/generated/sklearn
print(cv_logistic.best_estimator_)
```

```
LogisticRegression(C=0.046415888336127774, max_iter=10000, tol=0.1
)
```

In [106]:

```
print(cv_logistic.best_score_)
```

```
0.9020406477845386
```

Similar with pipeline

In [107]:
```
# Pipe Logistic
pipe_logistic = Pipeline([('select_scale', pipe_select_scale),
                          ('classify', LogisticRegression(max_i

# Parameters of pipelines can be set using '__' separated param
param_logistic = {
    'classify__C': np.logspace(-4, 4, 3),
    'select_scale__select__k': [600, 1000, 5000]
}

cv_logistic = GridSearchCV(pipe_logistic, param_logistic, cv=10
cv_logistic.fit(X_extract, y_train)
```

Out[107]:
```
GridSearchCV(cv=10,
             estimator=Pipeline(steps=[('select_scale',
                                        Pipeline(steps=[('select',
                                                         SelectKBe
st(k=500,

score_func=<function chi2 at 0x7fe565f9b1f0>)),
                                                        ('scale',
                                                         StandardS
caler(with_mean=False))])),
                                       ('classify',
                                        LogisticRegression(max_ite
r=10000,
                                                           tol=0.1
))]),
             param_grid={'classify__C': array([1.e-04, 1.e+00, 1.e
+04]),
                         'select_scale__select__k': [600, 1000, 50
00]},
             scoring='f1')
```

In [108]:
```
print(cv_logistic.best_estimator_)
```

```
Pipeline(steps=[('select_scale',
                 Pipeline(steps=[('select',
                                  SelectKBest(k=5000,
                                              score_func=<function
chi2 at 0x7fe565f9b1f0>)),
                                 ('scale', StandardScaler(with_mea
n=False))])),
                ('classify',
                 LogisticRegression(C=10000.0, max_iter=10000, tol
=0.1))])
```

In [109]:
```
print(cv_logistic.best_score_)
```

```
0.8457155973785053
```

Other Models

Naive Bayes

In [110]:
```python
from sklearn.naive_bayes import MultinomialNB

# Pipe NB
pipe_nb = Pipeline([('select_scale', pipe_select_scale),
                    ('classify', MultinomialNB())])

# Parameters of pipelines can be set using '__' separated param
param_nb = {
    'classify__alpha': [0.5, 1, 10],
    'select_scale__select__k': [600, 1000, 5000]
}

cv_nb = GridSearchCV(pipe_nb, param_nb, cv=10, scoring='f1')
cv_nb.fit(X_extract, y_train)
print(cv_nb.best_score_)


```

0.8301298848126655

In [111]:
```python
# full pipeline
model = Pipeline([("extract", FeatureUnion([("terms", Pipeline(

                                                          ("custom", CustomFea
                  ("select", SelectKBest(score_func = chi2, k =
                  ("scale", StandardScaler(with_mean = False)),
                  ("classify", MultinomialNB())])

# fitting
model.fit(X_train, y_train)

# final evaluation
y_pred = model.predict(X_test)
f1_score(y_test, y_pred)
```

Out[111]: 0.782608695652174

In [112]:

```
# we are now able to classify new messages!
#msg = pd.DataFrame(columns = ["text"],
                   #data     = ["REMINDER FROM O2: To get 2.50 p

msg = pd.DataFrame(columns = ["text"],
                   data    = ["The bikes are heavy and unwieldy

model.predict(msg)
```

Out[112]: array([0])

In [113]:

```
# we are now able to classify new messages!
#msg = pd.DataFrame(columns = ["text"],
                   #data     = ["REMINDER FROM O2: To get 2.50 p

msg = pd.DataFrame(columns = ["text"],
                   data    = ["Satisfied"])


model.predict(msg)
```

Out[113]: array([1])

Support Vector Machine

In [114]:

```
from sklearn.svm import LinearSVC

# Pipe SVC
pipe_svc = Pipeline([('select_scale', pipe_select_scale),
                     ('classify', LinearSVC(max_iter=10000, tol

# Parameters of pipelines can be set using '__' separated param
param_svc = {
    'classify__C': [0.01, 0.1, 1],
    'select_scale__select__k': [600, 1000, 5000]
}

cv_svc = GridSearchCV(pipe_svc, param_svc, cv=10, scoring='f1')
cv_svc.fit(X_extract, y_train)
print(cv_svc.best_score_)
```

0.8463498495228174

In [115]:
```
# full pipeline
model = Pipeline([("extract", FeatureUnion([("terms", Pipeline(

                                               ("custom", CustomFea
                  ("select", SelectKBest(score_func = chi2, k =
                  ("scale", StandardScaler(with_mean = False)),
                  ("classify", LinearSVC(C = 1, max_iter=10000,

# fitting
model.fit(X_train, y_train)

# final evaluation
y_pred = model.predict(X_test)
f1_score(y_test, y_pred)
```

Out[115]: 0.7972972972972973

In [116]:
```
# we are now able to classify new messages!
#msg = pd.DataFrame(columns = ["text"],
                   #data    = ["REMINDER FROM O2: To get 2.50 p

msg = pd.DataFrame(columns = ["text"],
                   data    = ["The bikes are heavy and unwieldy

model.predict(msg)
```

Out[116]: array([0])

In [117]:
```
# we are now able to classify new messages!
#msg = pd.DataFrame(columns = ["text"],
                   #data    = ["REMINDER FROM O2: To get 2.50 p

msg = pd.DataFrame(columns = ["text"],
                   data    = ["Satisfied"])


model.predict(msg)
```

Out[117]: array([1])

Random Forest

In [118]:
```
from sklearn.ensemble import RandomForestClassifier

# Pipe RF
pipe_rf = Pipeline([('select_scale', pipe_select_scale),
                    ('classify', RandomForestClassifier())])

# Parameters of pipelines can be set using '__' separated param
param_rf = {
    'classify__n_estimators': [100, 200],
    'select_scale__select__k': [600, 1000]
}

cv_rf = GridSearchCV(pipe_rf, param_rf, cv=10, scoring='f1')
cv_rf.fit(X_extract, y_train)
print(cv_rf.best_score_)
```

0.850498859773985

In [119]:
```
# full pipeline
model = Pipeline([("extract", FeatureUnion([("terms", Pipeline(

                                          ("custom", CustomFea
                 ("select", SelectKBest(score_func = chi2, k =
                 ("scale", StandardScaler(with_mean = False)),
                 ("classify", RandomForestClassifier())])

# fitting
model.fit(X_train, y_train)

# final evaluation
y_pred = model.predict(X_test)
f1_score(y_test, y_pred)
```

Out[119]: 0.8079470198675497

In [120]:
```
# we are now able to classify new messages!
#msg = pd.DataFrame(columns = ["text"],
                   #data    = ["REMINDER FROM O2: To get 2.50 p

msg = pd.DataFrame(columns = ["text"],
                   data    = ["The bikes are heavy and unwieldy

model.predict(msg)
```

Out[120]: array([1])

In [121]:
```
# we are now able to classify new messages!
#msg = pd.DataFrame(columns = ["text"],
                    #data    = ["REMINDER FROM O2: To get 2.50 p

msg = pd.DataFrame(columns = ["text"],
                   data    = ["Satisfied"])


model.predict(msg)
```

Out[121]: array([1])

Long Example 1: Text clustering

In [122]:
```
import re
import string
import pandas as pd
```

In [123]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering
```

Full text for clustering

This corpus contain some strings about Google and some strings about TF-IDF from Wikipedia. Just for example

In [124]:
```
all_text = """
Google and Facebook are strangling the free press to death. Dem
Your 60-second guide to security stuff Google touted today at N
A Guide to Using Android Without Selling Your Soul to Google
Review: Lenovo’s Google Smart Display is pretty and intelligent
Google Maps user spots mysterious object submerged off the coas
Android is better than IOS
In information retrieval, tf-idf or TFIDF, short for term frequ
is a numerical statistic that is intended to reflect how import
a word is to a document in a collection or corpus.
It is often used as a weighting factor in searches of informati
text mining, and user modeling. The tf-idf value increases prop
to the number of times a word appears in the document
and is offset by the frequency of the word in the corpus
""".split("\n")[1:-1]
```

In [125]:
```
all_text
```

Out[125]:
```
['Google and Facebook are strangling the free press to death. Demo
cracy is the loser',
 "Your 60-second guide to security stuff Google touted today at Ne
xt '18",
 'A Guide to Using Android Without Selling Your Soul to Google',
 'Review: Lenovo’s Google Smart Display is pretty and intelligent'
,
 'Google Maps user spots mysterious object submerged off the coast
of Greece - and no-one knows what it is',
 'Android is better than IOS',
 'In information retrieval, tf–idf or TFIDF, short for term freque
ncy–inverse document frequency',
 'is a numerical statistic that is intended to reflect how importa
nt ',
 'a word is to a document in a collection or corpus.',
 'It is often used as a weighting factor in searches of informatio
n retrieval',
 'text mining, and user modeling. The tf-idf value increases propo
rtionally',
 'to the number of times a word appears in the document',
 'and is offset by the frequency of the word in the corpus']
```

Preprocessing and tokenizing

Firstly, we must bring every chars to lowercase and remove all punctuation, because it's not important for our task, but is very harmful for clustering algorithm. After that, we'll split strings to array of words.

In [126]:
```
def preprocessing(line):
    line = line.lower()
    line = re.sub(r"[{}]".format(string.punctuation), " ", line
    return line
```

Now, let's calculate tf-idf for this corpus

In [127]:
```
tfidf_vectorizer = TfidfVectorizer(preprocessor=preprocessing)
tfidf = tfidf_vectorizer.fit_transform(all_text)
```

K-means

In [128]:
```
kmeans = KMeans(n_clusters=2)
```

In [129]:
```
list(zip(kmeans.fit_predict(tfidf), all_text))
```

Out[129]:
```
[(1,
  'Google and Facebook are strangling the free press to death. Dem
ocracy is the loser'),
 (1, "Your 60-second guide to security stuff Google touted today a
t Next '18"),
 (1, 'A Guide to Using Android Without Selling Your Soul to Google
'),
 (1, 'Review: Lenovo’s Google Smart Display is pretty and intellig
ent'),
 (1,
  'Google Maps user spots mysterious object submerged off the coas
t of Greece - and no-one knows what it is'),
 (1, 'Android is better than IOS'),
 (0,
  'In information retrieval, tf–idf or TFIDF, short for term frequ
ency–inverse document frequency'),
 (1, 'is a numerical statistic that is intended to reflect how imp
ortant '),
 (0, 'a word is to a document in a collection or corpus.'),
 (0,
  'It is often used as a weighting factor in searches of informati
on retrieval'),
 (0,
  'text mining, and user modeling. The tf-idf value increases prop
ortionally'),
 (0, 'to the number of times a word appears in the document'),
 (0, 'and is offset by the frequency of the word in the corpus')]
```

Agglomerative Clustering

In [130]:
```
hac = AgglomerativeClustering(n_clusters=2, affinity='cosine',
```

In [131]:
```
list(zip(hac.fit_predict(tfidf.toarray()), all_text))
```

Out[131]:
```
[(0,
  'Google and Facebook are strangling the free press to death. Dem
ocracy is the loser'),
 (1, "Your 60-second guide to security stuff Google touted today a
t Next '18"),
 (1, 'A Guide to Using Android Without Selling Your Soul to Google
'),
 (0, 'Review: Lenovo’s Google Smart Display is pretty and intellig
ent'),
 (0,
  'Google Maps user spots mysterious object submerged off the coas
t of Greece - and no-one knows what it is'),
 (1, 'Android is better than IOS'),
 (0,
  'In information retrieval, tf–idf or TFIDF, short for term frequ
ency–inverse document frequency'),
 (1, 'is a numerical statistic that is intended to reflect how imp
ortant '),
 (0, 'a word is to a document in a collection or corpus.'),
 (0,
  'It is often used as a weighting factor in searches of informati
on retrieval'),
 (0,
  'text mining, and user modeling. The tf–idf value increases prop
ortionally'),
 (0, 'to the number of times a word appears in the document'),
 (0, 'and is offset by the frequency of the word in the corpus')]
```

Example 2: Topic model (1): BikeMi survey

In [134]:
```
import nltk
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/giancarlomanzi/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[134]:
```
True
```

Cleaning and pre-processing

In [135]:
```
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
import string
stop=set(stopwords.words('english'))
exclude=set(string.punctuation)
lemma=WordNetLemmatizer()
def clean(doc):
    stop_free=" ".join([i for i in doc.lower().split() if i not
    punc_free=''.join(ch for ch in stop_free if ch not in exclu
    normalized=" ".join(lemma.lemmatize(word) for word in punc_
    return normalized
```

In [136]:
```
import pandas as pd
df = pd.read_csv('Polarity2014Reduced.csv', sep = ";", header =
df.columns=['review','sentiment']
df2=df[df['sentiment']==-1]
df2.shape
```

Out[136]: (354, 2)

In [137]:
```
doc_complete=df2.iloc[0:2065,0].values.tolist()
doc_clean=[clean(doc).split() for doc in doc_complete]
```

Getting the document-term matrix

In [138]:
```
from sklearn.feature_extraction.text import CountVectorizer
import numpy as np
SOME_FIXED_SEED = 42
np.random.seed(SOME_FIXED_SEED)
```

In [139]:
```
cv=CountVectorizer(min_df=2,max_df=50,ngram_range=(1,2), token_
```

In [140]:
```
cv_features=cv.fit_transform(doc_clean)
print(cv_features.shape)
vocabulary=np.array(cv.get_feature_names())
```

```
(354, 1392)
```

In [141]:
```
vocabulary
```

Out[141]:
```
array(['1', '1 volta', '10', ..., '√® stato', '√® troppo', '√® un'
],
      dtype='<U24')
```

In [142]:
```
vocabulary
```

Out[142]:
```
array(['1', '1 volta', '10', ..., '√® stato', '√® troppo', '√® un'
],
      dtype='<U24')
```

LDA ANALYSIS

In [143]:
```
# Using sklearn.decomposition LDA with 11 topics
from sklearn.decomposition import LatentDirichletAllocation
TOTAL_TOPICS=11
```

In [144]:
```
lda_model=LatentDirichletAllocation(n_components=TOTAL_TOPICS,m
```

In [145]:
```
# Using the transformer 'fit_transform'
document_topics=lda_model.fit_transform(cv_features)

```

In [146]:
```
document_topics.shape
```

Out[146]:
```
(354, 11)
```

In [145]:

```
# Extraqcting the most important 10 terms for each topic
topic_terms=lda_model.components_
top_terms=10 # number of 'top terms'
topic_key_terms_idxs=np.argsort(-np.absolute(topic_terms), axis
topic_keyterms=vocabulary[topic_key_terms_idxs]
topics=[', '.join(topic) for topic in topic_keyterms]
pd.set_option('display.max_colwidth',-1)
topics_df=pd.DataFrame(topics,columns=['Term per Topic'], index
topics_df
```

```
<ipython-input-145-515db2202b96>:7: FutureWarning: Passing a negat
ive integer is deprecated in version 1.0 and will not be supported
in future version. Instead, use None to not limit the column width
.
  pd.set_option('display.max_colwidth',-1)
```

Out[145]:

	Term per Topic
Topic1	della, completamente, servizio, la bici, possibilit√†, possibilit√† di, tramite, segnalare, app, troppo
Topic2	servizio, bicicletta, lasciare, la bicicletta, cambio, stalli, se, le stazioni, lasciare la, la bici
Topic3	piene, con, sempre, al, vuote, molto, tutte, alcune, le colonnine, colonnine
Topic4	mi, da, mi √®, servizio, stazione, la bici, se, √® capitato, capitato, bikemi
Topic5	con, tempo, migliorare, al, ecc, cambio, sistema, delle bici, migliorare il, anche
Topic6	essere, con, le bici, frequenza, essere pi√π, manutenzione, bike, bici con, bici sono, troppo
Topic7	pi√π spesso, al, gomme, spesso le, della, cambio, le bici, del, gonfiare, controllare
Topic8	possibilit√† di, possibilit√†, segnalare, della, di segnalare, che non, dei, controllare pi√π, delle biciclette, al
Topic9	troppo, piste, piste ciclabili, ciclabili, cambio, con, problemi, ruote, manutenzione, sgonfie
Topic10	anche, le stazioni, stazione, molto, piene, servizio, tempo, del, da, cambio
Topic11	stalli, gli, centro, gli stalli, ci, nelle, di punta, punta, aumentare, le bici

In [146]:

```
dt_df=pd.DataFrame(document_topics,columns=['T'+str(i) for i in
dt_df
```

Out[146]:

	T1	T2	T3	T4	T5	T6	T7	T8	
0	0.005348	0.005348	0.005348	0.005348	0.005348	0.005348	0.005348	0.005348	0.946
1	0.009091	0.909084	0.009091	0.009091	0.009091	0.009091	0.009092	0.009092	0.009
2	0.002841	0.971589	0.002841	0.002841	0.002841	0.002841	0.002841	0.002841	0.002
3	0.015152	0.015153	0.015154	0.848456	0.015152	0.015160	0.015157	0.015152	0.015
4	0.001567	0.001567	0.984325	0.001567	0.001567	0.001567	0.001567	0.001567	0.001
...	...	...	...	...	...	...	...	...	
349	0.003637	0.003637	0.003636	0.003637	0.003636	0.003637	0.003637	0.003637	0.433
350	0.010101	0.010102	0.010102	0.898987	0.010101	0.010101	0.010102	0.010101	0.010
351	0.018183	0.018182	0.018182	0.018182	0.018183	0.018182	0.018183	0.818175	0.018
352	0.002392	0.002393	0.002393	0.002392	0.002392	0.002392	0.002392	0.002392	0.002
353	0.011365	0.011364	0.011364	0.011364	0.886355	0.011365	0.011365	0.011365	0.011

354 rows × 11 columns

In [147]:

```
# Column 'Contribution%' gives the max probability among the 35
# features (terms) for each topic
dt_df=pd.DataFrame(document_topics,columns=['T'+str(i) for i in
pd.options.display.float_format='{:,.5f}'.format
pd.set_option('display.max_colwidth',200)
max_contrib_topics=dt_df.max(axis=0)
dominant_topics=max_contrib_topics.index
contrib_perc=max_contrib_topics.values
document_numbers=[dt_df[dt_df[t]==max_contrib_topics.loc[t]].in
results_df=pd.DataFrame({'Dominant Topic':dominant_topics,'Cont
results_df
```

Out[147]:

	Dominant Topic	Contribution%	Answer Num	Topic
Topic1	T1	0.97159	193	della, completamente, servizio, la bici, possibilit√†, possibilit√† di, tramite, segnalare, app, troppo
Topic2	T2	0.99209	179	servizio, bicicletta, lasciare, la bicicletta, cambio, stalli, se, le stazioni, lasciare la, la bici
Topic3	T3	0.98510	52	piene, con, sempre, al, vuote, molto, tutte, alcune, le colonnine, colonnine
Topic4	T4	0.98978	328	mi, da, mi √®, servizio, stazione, la bici, se, √® capitato, capitato, bikemi
Topic5	T5	0.99072	28	con, tempo, migliorare, al, ecc, cambio, sistema, delle bici, migliorare il, anche
Topic6	T6	0.96503	126	essere, con, le bici, frequenza, essere pi√π, manutenzione, bike, bici con, bici sono, troppo
Topic7	T7	0.98557	342	pi√π spesso, al, gomme, spesso le, della, cambio, le bici, del, gonfiare, controllare
Topic8	T8	0.96503	174	possibilit√† di, possibilit√†, segnalare, della, di segnalare, che non, dei, controllare pi√π, delle biciclette, al
Topic9	T9	0.98943	294	troppo, piste, piste ciclabili, ciclabili, cambio, con, problemi, ruote, manutenzione, sgonfie
Topic10	T10	0.98864	198	anche, le stazioni, stazione, molto, piene, servizio, tempo, del, da, cambio
Topic11	T11	0.99126	114	stalli, gli, centro, gli stalli, ci, nelle, di punta, punta, aumentare, le bici

In [148]:

```
# This gives, for each topic, the % of features having prob >0.
numT1=np.count_nonzero(dt_df['T1']>0.9)
FrT1=numT1/2133
numT2=np.count_nonzero(dt_df['T2']>0.9)
FrT2=numT2/2133
numT3=np.count_nonzero(dt_df['T3']>0.9)
FrT3=numT3/2133
numT4=np.count_nonzero(dt_df['T4']>0.9)
FrT4=numT4/2133
numT5=np.count_nonzero(dt_df['T5']>0.9)
FrT5=numT5/2133
```

```
numT6=np.count_nonzero(dt_df['T6']>0.9)
FrT6=numT6/2133
numT7=np.count_nonzero(dt_df['T7']>0.9)
FrT7=numT7/2133
numT8=np.count_nonzero(dt_df['T8']>0.9)
FrT8=numT8/2133
numT9=np.count_nonzero(dt_df['T9']>0.9)
FrT9=numT9/2133
numT10=np.count_nonzero(dt_df['T10']>0.9)
FrT10=numT10/2133
numT11=np.count_nonzero(dt_df['T11']>0.9)
FrT11=numT11/2133
d=(FrT1,FrT2,FrT3,FrT4,FrT5,FrT6,FrT7,FrT8,FrT9,FrT10,FrT11)
df_Fr=pd.DataFrame(data=d)
results_df.insert(4,'Freq 0.9-1',df_Fr.values)
results_df
```

Out[148]:

	Dominant Topic	Contribution%	Answer Num	Topic	Freq 0.9-1
Topic1	T1	0.97159	193	della, completamente, servizio, la bici, possibilit√†, possibilit√† di, tramite, segnalare, app, troppo	0.00516
Topic2	T2	0.99209	179	servizio, bicicletta, lasciare, la bicicletta, cambio, stalli, se, le stazioni, lasciare la, la bici	0.01547
Topic3	T3	0.98510	52	piene, con, sempre, al, vuote, molto, tutte, alcune, le colonnine, colonnine	0.01172
Topic4	T4	0.98978	328	mi, da, mi √®, servizio, stazione, la bici, se, √® capitato, capitato, bikemi	0.00891
Topic5	T5	0.99072	28	con, tempo, migliorare, al, ecc, cambio, sistema, delle bici, migliorare il, anche	0.00609
Topic6	T6	0.96503	126	essere, con, le bici, frequenza, essere pi√π, manutenzione, bike, bici con, bici sono, troppo	0.00469
Topic7	T7	0.98557	342	pi√π spesso, al, gomme, spesso le, della, cambio, le bici, del, gonfiare, controllare	0.00797
Topic8	T8	0.96503	174	possibilit√† di, possibilit√†, segnalare, della, di segnalare, che non, dei, controllare pi√π, delle biciclette, al	0.00797
Topic9	T9	0.98943	294	troppo, piste, piste ciclabili, ciclabili, cambio, con, problemi, ruote, manutenzione, sgonfie	0.02391
Topic10	T10	0.98864	198	anche, le stazioni, stazione, molto, piene, servizio, tempo, del, da, cambio	0.00656
Topic11	T11	0.99126	114	stalli, gli, centro, gli stalli, ci, nelle, di punta, punta, aumentare, le bici	0.01828

In [159]:

```
#This is to let you have larger fonts...
from IPython.core.display import HTML
HTML("""
<style>

div.cell { /* Tunes the space between cells */
margin-top:1em;
margin-bottom:1em;
}

div.text_cell_render h1 { /* Main titles bigger, centered */
font-size: 2.2em;
line-height:1.4em;
text-align:center;
}

div.text_cell_render h2 { /*  Parts names nearer from text */
margin-bottom: -0.4em;
}


div.text_cell_render { /* Customize text cells */
font-family: 'Times New Roman';
font-size:1.5em;
line-height:1.4em;
padding-left:3em;
padding-right:3em;
}
</style>
""")
```

Out[159]:

In [ ]: