Econometric analysis with Python

Programa de Doctorat en Economia, Universitat de Barcelona

David Moriña

Time series analysis

Reading time series

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import statsmodels.api as sm
from scipy import stats
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.api import qqplot

sun = sm.datasets.sunspots.load_pandas().data
sun.head()
```

	YEAR	SUNACTIVITY
0	1700.0	5.0
1	1701.0	11.0
2	1702.0	16.0
3	1703.0	23.0
4	1704.0	36.0

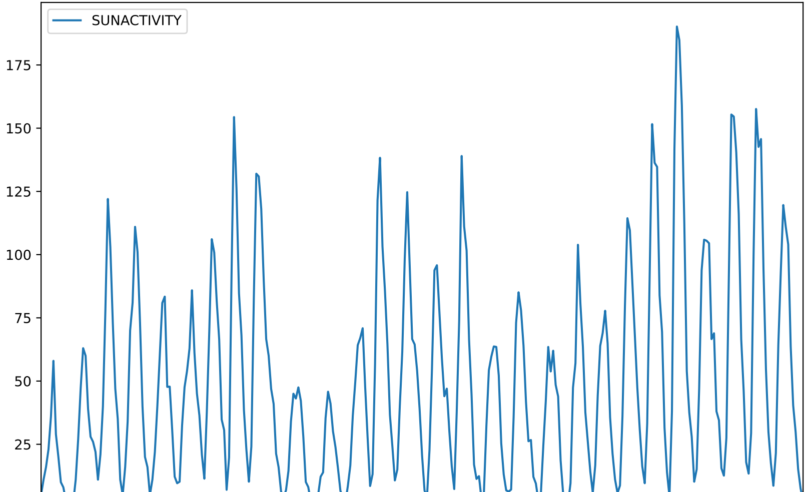
Reading time series

```
sun.index =
 ↪  pd.Index(sm.tsa.datetools.dates_from_range("1700",
 ↪  "2008"))
sun.index.freq = sun.index.inferred_freq
del sun["YEAR"]
```

Reading time series

```
sun.plot()
```

```
<Axes: >
```



Testing stationarity

```python
from statsmodels.tsa.stattools import adfuller

def adf_test(timeseries):
    print("Results of Dickey-Fuller Test:")
    dftest = adfuller(timeseries, autolag="AIC")
    dfoutput = pd.Series(
        dftest[0:4],
        index=[
            "Test Statistic",
            "p-value",
            "#Lags Used",
            "Number of Observations Used",
        ],
    )
    for key, value in dftest[4].items():
        dfoutput["Critical Value (%s)" % key] = value
    print(dfoutput)
```

Testing stationarity

```
adf_test(sun)
```

```
Results of Dickey-Fuller Test:
Test Statistic                  -2.837781
p-value                          0.053076
#Lags Used                       8.000000
Number of Observations Used    300.000000
Critical Value (1%)             -3.452337
Critical Value (5%)             -2.871223
Critical Value (10%)            -2.571929
dtype: float64
```

Testing stationarity

```
from statsmodels.tsa.stattools import kpss


def kpss_test(timeseries):
    print("Results of KPSS Test:")
    kpsstest = kpss(timeseries, regression="c",
 ↪  nlags="auto")
    kpss_output = pd.Series(
        kpsstest[0:3], index=["Test Statistic",
 ↪  "p-value", "Lags Used"]
    )
    for key, value in kpsstest[3].items():
        kpss_output["Critical Value (%s)" % key] =
 ↪  value
    print(kpss_output)
```

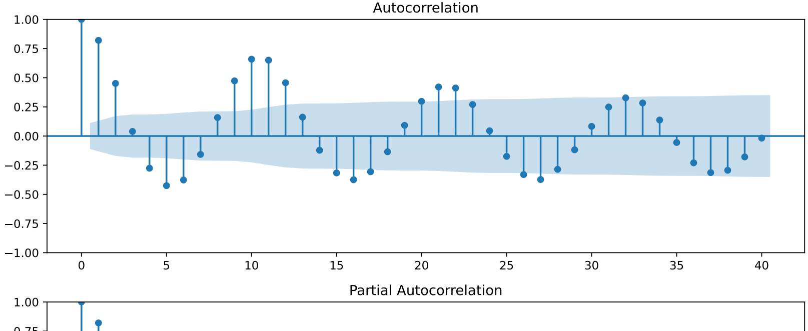
Testing stationarity

```
kpss_test(sun)
```

```
Results of KPSS Test:
Test Statistic            0.669866
p-value                   0.016285
Lags Used                 7.000000
Critical Value (10%)      0.347000
Critical Value (5%)       0.463000
Critical Value (2.5%)     0.574000
Critical Value (1%)       0.739000
dtype: float64
```

Classical models

```
fig = plt.figure()
ax1 = fig.add_subplot(2, 1, 1)
fig = sm.graphics.tsa.plot_acf(sun.values.squeeze(),
 ↪  lags=40, ax=ax1)
ax2 = fig.add_subplot(2, 1, 2)
fig = sm.graphics.tsa.plot_pacf(sun, lags=40, ax=ax2)
fig.tight_layout()
```



Classical models

```
arma_mod30 = ARIMA(sun, order=(2, 0, 0)).fit()
print(arma_mod30.summary())
print(arma_mod30.aic)
```

```
                                   SARIMAX Results
===================================================================================
Dep. Variable:              SUNACTIVITY   No. Observations:
Model:                   ARIMA(2, 0, 0)   Log Likelihood
Date:                  Tue, 19 Mar 2024   AIC
Time:                          15:58:44   BIC
Sample:                      12-31-1700   HQIC
                           - 12-31-2008
Covariance Type:                    opg
===================================================================================
                 coef    std err          z      P>|z|
-----------------------------------------------------------------------------------
const         49.7462      3.938     12.631      0.000
ar.L1          1.3906      0.037     37.694      0.000
```

Classical models

```
arma_mod40 = ARIMA(sun, order=(3, 0, 0)).fit()
print(arma_mod40.summary())
print(arma_mod40.aic)
```

```
                                 SARIMAX Results
==============================================================
Dep. Variable:            SUNACTIVITY   No. Observations:
Model:                 ARIMA(3, 0, 0)   Log Likelihood
Date:                Tue, 19 Mar 2024   AIC
Time:                        15:58:44   BIC
Sample:                    12-31-1700   HQIC
                         - 12-31-2008
Covariance Type:                  opg
==============================================================
                 coef    std err          z      P>|z|
--------------------------------------------------------------
const         49.7519      3.518     14.141      0.000
ar.L1          1.3008      0.050     25.763      0.000
```

Classical models

```
sarma = sm.tsa.statespace.SARIMAX(sun, order=(2, 1,
 ↪  2), seasonal_order=(1, 0, 1, 11))
results = sarma.fit()
print(results.summary())
print(results.aic)
```

```
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =              7     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  4.95602D+00    |proj g|=  1.39577D+0

At iterate    5    f=  4.20132D+00    |proj g|=  7.37926D-0
```

Diagnosis

```
resid = results.resid
stats.normaltest(resid)
```

```
NormaltestResult(statistic=32.8786803250079, pvalue=7.25245
```

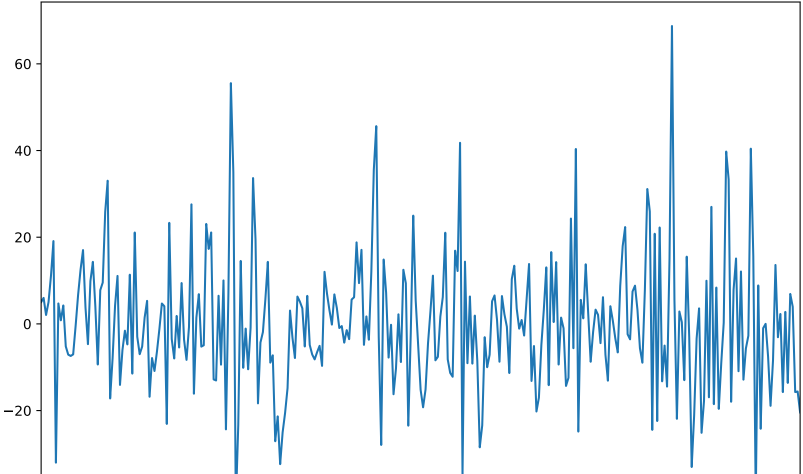
Diagnosis

```
sm.stats.durbin_watson(resid)
```

```
1.8624493581693025
```

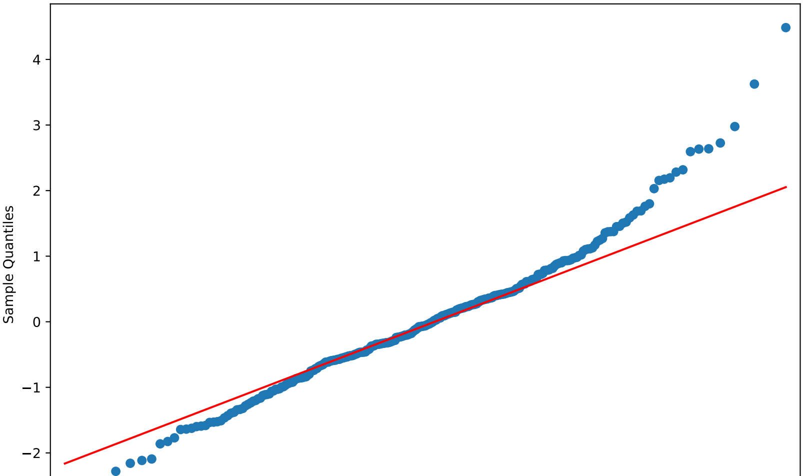
Diagnosis

```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax = results.resid.plot(ax=ax)
```



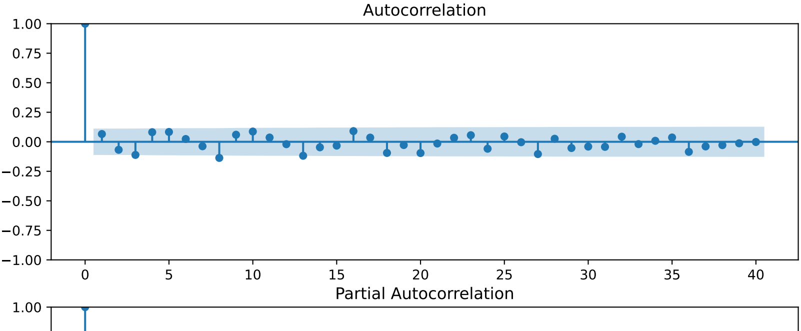
Diagnosis

```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
fig = qqplot(resid, line="q", ax=ax, fit=True)
```



Diagnosis

```
fig = plt.figure()
ax1 = fig.add_subplot(2, 1,1)
fig = sm.graphics.tsa.plot_acf(resid, lags=40,
 ↪  ax=ax1)
ax2 = fig.add_subplot(2, 1, 2)
fig = sm.graphics.tsa.plot_pacf(resid, lags=40,
 ↪  ax=ax2)
```



Prediction

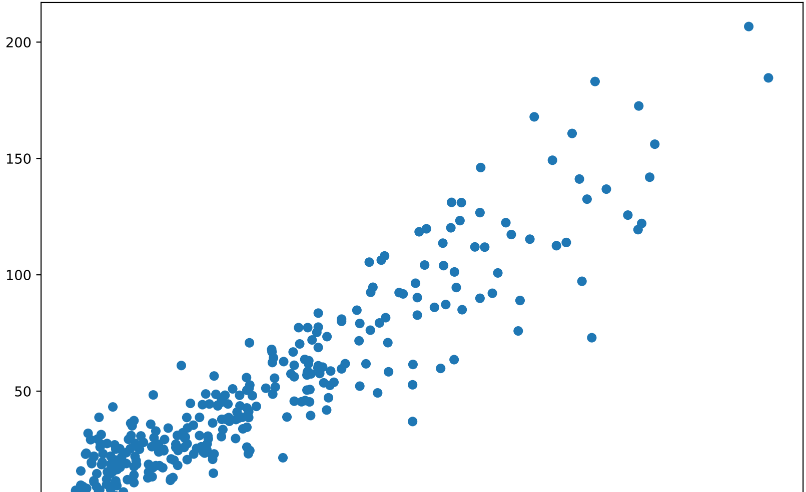
```
predict_sunspots = results.predict("1700", "2008")
def mean_forecast_err(y, yhat):
    return y.sub(yhat).mean()
mean_forecast_err(sun.SUNACTIVITY, predict_sunspots)
```

```
0.033881177132218006
```

Prediction

```
plt.scatter(x=sun.SUNACTIVITY, y=predict_sunspots)
```

```
<matplotlib.collections.PathCollection at 0x742b30fa0d30>
```



Exponential smoothing

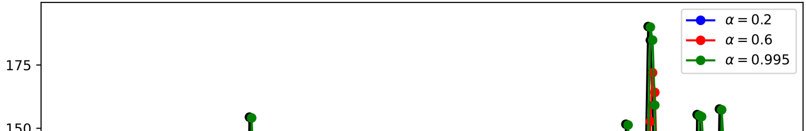
```
from statsmodels.tsa.api import ExponentialSmoothing,
 ↪  SimpleExpSmoothing, Holt

fit1 = SimpleExpSmoothing(sun,
 ↪  initialization_method="heuristic").fit(smoothing_level=
 ↪  optimized=False)
fcast1 = fit1.forecast(3).rename(r"$\alpha=0.2$")
fit2 = SimpleExpSmoothing(sun,
 ↪  initialization_method="heuristic").fit(smoothing_level=
 ↪  optimized=False)
fcast2 = fit2.forecast(3).rename(r"$\alpha=0.6$")
fit3 = SimpleExpSmoothing(sun,
 ↪  initialization_method="estimated").fit()
fcast3 = fit3.forecast(3).rename(r"$\alpha=%s$" %
 ↪  fit3.model.params["smoothing_level"])
```

Exponential smoothing

```
plt.plot(sun, marker="o", color="black")
plt.plot(fit1.fittedvalues, marker="o", color="blue")
(line1,) = plt.plot(fcast1, marker="o", color="blue")
plt.plot(fit2.fittedvalues, marker="o", color="red")
(line2,) = plt.plot(fcast2, marker="o", color="red")
plt.plot(fit3.fittedvalues, marker="o", color="green")
(line3,) = plt.plot(fcast3, marker="o", color="green")
plt.legend([line1, line2, line3], [fcast1.name, fcast2.name, fcast3.name])
```

```
<matplotlib.legend.Legend at 0x742b30e776a0>
```



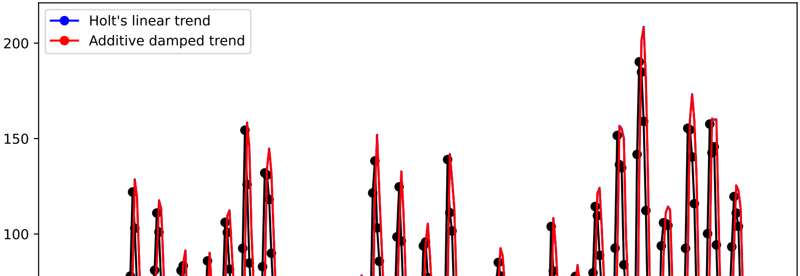
Exponential smoothing

```
fit1 = Holt(sun,
↪  initialization_method="estimated").fit(smoothing_level=
↪  smoothing_trend=0.2, optimized=False)
fcast1 = fit1.forecast(5).rename("Holt's linear
↪  trend")
fit2 = Holt(sun, damped_trend=True,
↪  initialization_method="estimated").fit(smoothing_level=
↪  smoothing_trend=0.2)
fcast2 = fit2.forecast(5).rename("Additive damped
↪  trend")
```

Exponential smoothing

```
plt.plot(sun, marker="o", color="black")
plt.plot(fit1.fittedvalues, color="blue")
(line1,) = plt.plot(fcast1, marker="o", color="blue")
plt.plot(fit2.fittedvalues, color="red")
(line2,) = plt.plot(fcast2, marker="o", color="red")
plt.legend([line1, line2], [fcast1.name,
 ↪  fcast2.name])
```

```
<matplotlib.legend.Legend at 0x742b348342e0>
```



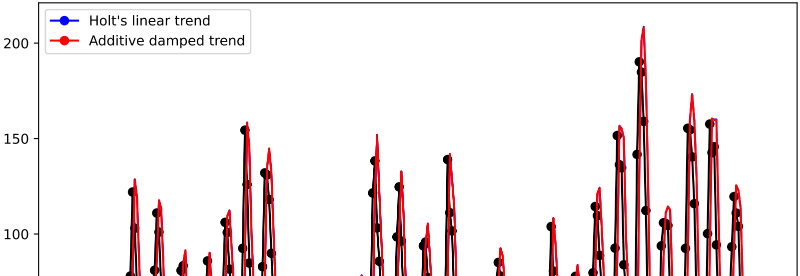
Exponential smoothing

```
fit1 = Holt(sun,
↪  initialization_method="estimated").fit(smoothing_level=
↪  smoothing_trend=0.2, optimized=False)
fcast1 = fit1.forecast(5).rename("Holt's linear
↪  trend")
fit2 = Holt(sun, damped_trend=True,
↪  initialization_method="estimated").fit(smoothing_level=
↪  smoothing_trend=0.2)
fcast2 = fit2.forecast(5).rename("Additive damped
↪  trend")
```

Exponential smoothing

```
plt.plot(sun, marker="o", color="black")
plt.plot(fit1.fittedvalues, color="blue")
(line1,) = plt.plot(fcast1, marker="o", color="blue")
plt.plot(fit2.fittedvalues, color="red")
(line2,) = plt.plot(fcast2, marker="o", color="red")
plt.legend([line1, line2], [fcast1.name,
 ↪  fcast2.name])
```

```
<matplotlib.legend.Legend at 0x742b34853e80>
```



Spatiotemporal modelling

Basic geometric objects

- Fundamental geometric objects that can be used in Python with Shapely module
- The most fundamental geometric objects are Points, Lines and Polygons which are the basic ingredients when working with spatial data in vector format

Geometric Objects consist of coordinate *tuples* where:

- *Point:* Represents a single point in space. Points can be either two-dimensional (x, y) or three dimensional (x, y, z)
- *LineString:* Represents a sequence of points joined together to form a line. Hence, a line consist of a list of at least two coordinate *tuples*
- *Polygon:* Represents a filled area that consists of a list of at least three coordinate *tuples* that forms the outer ring and a (possible) list of hole polygons

Basic geometric objects

```
from shapely.geometry import Point, LineString,
↪  Polygon

point1 = Point(2.2, 4.2)
print(type(point1))
```

```
<class 'shapely.geometry.point.Point'>
```

Basic geometric objects

```
poly = Polygon([(2.2, 4.2), (7.2, -25.1), (9.26,
 ↪  -2.456)])
# Attributes:

# Get the centroid of the Polygon
poly_centroid = poly.centroid

# Get the area of the Polygon
poly_area = poly.area

# Get the bounds of the Polygon (i.e. bounding box)
poly_bbox = poly.bounds

# Get the exterior of the Polygon
poly_ext = poly.exterior

# Get the length of the exterior
poly_ext_length = poly_ext.length
```

Geolocalisation with Python and OpenStreetMaps

```
from geopandas.tools import geocode

aules =
↪  pd.read_csv("/home/dmorina/Insync/dmorina@ub.edu/OneDri
↪  Biz/Docència/UB/2023-2024/PyEcon/3. Python for
↪  data analysis/examples/aules.csv")
aules['address'] = aules['aulacarrer'] + ' ' +
↪  aules['aulanum'] + ', ' + aules['aulacp'] + ' ' +
↪  aules['població_aula'] + ', Spain'
location = geocode(aules['address'])
aules['address'] = location['address']
join = location.merge(aules, on='address')
```

Geolocalisation with Python and OpenStreetMaps

```
join.head()
```

	geometry	address
0	POINT (0.56637 41.18341)	Rambla de Catalunya, 43791, Ascó, Ca
1	POINT (2.17570 41.43344)	21-23, Carrer de Deià, 08016, Carrer de
2	POINT (2.78371 41.79036)	Carrer de la Costa Brava, 17411, Vidre
3	POINT (3.18028 42.26463)	Riera Ginjolers, 17480, Roses, Cataluny
4	POINT (2.49749 42.19768)	Escola Llar Lluís Maria Mestres i Martí,

Geolocalisation with Python and OpenStreetMaps

Export to a shapefile

```
# Output file path
outfp =
 ↪  r"/home/dmorina/Insync/dmorina@ub.edu/OneDrive
 ↪  Biz/Docència/UB/2023-2024/PyEcon/3. Python for
 ↪  data analysis/examples/aules.shp"

# Save to Shapefile
location.to_file(outfp)
```

Representing spatial data

Loading map data

```
import geopandas as gpd

fp = "/home/dmorina/Insync/dmorina@ub.edu/OneDrive
↪ Biz/Docència/UB/2023-2024/PyEcon/3. Python for
↪ data analysis/examples/"
data = gpd.read_file(fp)
data.head()
```

	NAME	Income	NoSchool	NoSchoolSE	IncomeSE	geome
0	Aroostook	21024	0.013387	0.001407	250.909	POLY
1	Somerset	21025	0.005212	0.001150	390.909	POLY
2	Piscataquis	21292	0.006338	0.002129	724.242	POLY
3	Penobscot	23307	0.006845	0.001025	242.424	POLY
4	Washington	20015	0.004782	0.000966	327.273	POLY

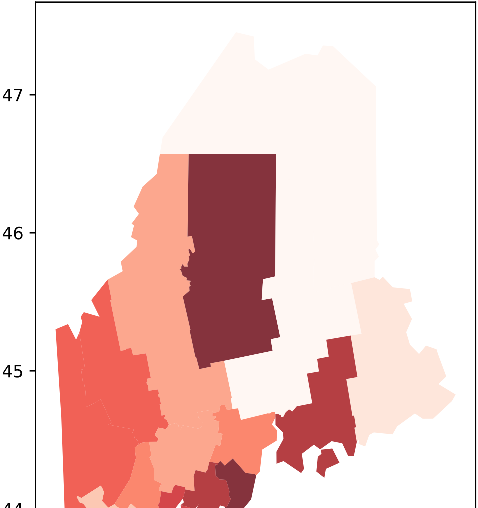
Representing spatial data

```
data.crs
```

```
<Geographic 2D CRS: EPSG:4326>
Name: WGS 84
Axis Info [ellipsoidal]:
- Lat[north]: Geodetic latitude (degree)
- Lon[east]: Geodetic longitude (degree)
Area of Use:
- name: World.
- bounds: (-180.0, -90.0, 180.0, 90.0)
Datum: World Geodetic System 1984 ensemble
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
```

Representing spatial data

```
my_map = data.plot(column="IncomeSE", linewidth=0.8,
 ↪  cmap="Reds", scheme="quantiles", k=9, alpha=0.8)
plt.show()
```



Spatial regression

```python
import numpy as np
from pysal.model import spreg

# Fit OLS model
m1 = spreg.OLS(y=np.array(data[["IncomeSE"]]),
 ↪  x=np.array(data[["NoSchoolSE"]]))
print(m1.summary)
```

```
REGRESSION RESULTS
------------------

SUMMARY OF OUTPUT: ORDINARY LEAST SQUARES
-----------------------------------------
Data set            :     unknown
Weights matrix      :        None
Dependent Variable  :     dep_var                Number of
Mean dependent var  :    458.7501                Number of
S.D. dependent var  :    145.9563                Degrees of
```

Running R code from Python

R code in Python

```python
import rpy2.robjects as robjects
from rpy2.robjects import r, pandas2ri
from rpy2.robjects.packages import importr
pandas2ri.activate()

# import the jags package
R2jags = importr('R2jags')
```

Introduction to Bayesian statistics

Bayesian thinking

- Parameters are not single values anymore, but probability distributions
- We establish a belief about our data (prior)
- We update the model in accordance to our beliefs and the available data

Example

We have a dataset on Spanish high speed rail tickets pricing. Assuming that the price distribution is Gaussian, we aim to estimate the population mean μ and standard deviation σ

```
import pandas as pd
import os

os.chdir("/home/dmorina/Insync/dmorina@ub.edu/OneDrive
 ↪  Biz/Docència/UB/2023-2024/PyEcon/3. Python for
 ↪  data analysis/examples/")
renfe = pd.read_csv("renfe.csv")
renfe.head()
```

	insert_date	origin	destination	start_date
0	2019-04-22 08:00:25	MADRID	SEVILLA	2019-04-28 08:30:0
1	2019-04-22 10:03:24	MADRID	VALENCIA	2019-05-20 06:45:0
2	2019-04-25 19:19:46	MADRID	SEVILLA	2019-05-29 06:20:0
3	2019-04-24 06:21:57	SEVILLA	MADRID	2019-05-03 08:35:0

Example

Choices of priors:

- μ, population mean. We know that train ticket price can not be lower than 0 or higher than 300, so a reasonable choice might be a uniform distribution between 0 and 300. If other reliable prior information is available, we could use it!
- σ, standard deviation of a population. Can only be positive, so a uniform distribution between 0 and 300 is also suitable. Again, very wide.

Choices for ticket price likelihood function:

- y is an observed variable representing the data that comes from a normal distribution with the parameters μ and σ.
- Draw 1000 posterior samples using NUTS sampling.

Example

```
robjects.globalenv["renfe"] = renfe
robjects.r('''
model <- function() {
  # likelihood
  for (i in 1:N) {
    y[i] ~ dnorm(mu, tau)
  }
  # priors
  mu ~ dunif(0, 300)
  sigma ~ dunif(0, 300)
  tau <- 1/(sigma*sigma)
}

data <- list(y=renfe$price, N=length(renfe$price))
params <- c("mu", "sigma")
fit <- jags(data = data, inits = NULL,
 ↪  parameters.to.save = params, model.file = model,
            n.chains = 5, n.iter = 100, n.burnin =
```

Example

```
r_f = robjects.globalenv['fit']
print(r_f)
```

```
Inference for Bugs model at "/tmp/RtmpfUXeBQ/modela4b63cb3
 5 chains, each with 100 iterations (first 10 discarded), 
 n.sims = 45 iterations saved
           mu.vect   sd.vect       2.5%        25%
mu          74.673    20.811     63.176     63.367     63.
sigma       53.946    37.042     24.121     24.328     30.
deviance 257573.817 24571.282 237858.948 237860.170 240175.
              97.5%  Rhat n.eff
mu          126.741 0.945    45
sigma       126.492 0.951    45
deviance 305019.181 0.949    45

For each parameter, n.eff is a crude measure of effective 
and Rhat is the potential scale reduction factor (at conve
```

Example

```
from rpy2.robjects import rl
import rpy2.robjects.lib.ggplot2 as gp

robjects.r('''fitMCMC <- as.mcmc(fit)''')
r_f = robjects.globalenv['fitMCMC']
r = robjects.r
r.X11()

r.layout(r.matrix(robjects.IntVector([1,2,3,2]),
↪  nrow=2, ncol=2))
r.plot(r.fitMCMC)
```

```
<rpy2.rinterface_lib.sexp.NULLType object at 0x742b1c5118c0
```