Monte_Carlo_gruppo_1

April 15, 2020

1 Analisi del valore del titolo Google e simulazione Monte Carlo

Matteo Stefanini, Dario Castelluccio, Richard Aldeghi Gennaio-Aprile 2020

Dati scaricati a gennaio 2020 da Yahoo Finance

https://finance.yahoo.com/

Valori di inizio mese.

Periodo considerato: maggio 2015 a gennaio 2020

Questo notebook è stato sviluppato per effettuare la previsione del valore di Google e sviluppare una simulazione Monte Carlo come discusso nel seguito.

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

#Disattivazione messaggi durante esecuzione
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
goog = pd.read_csv('GOOG_2015_2020.csv',sep=",")

#Mostra prime 15 righe
goog.head(15)
```

Out[3]:
```
          Date        Open        High         Low       Close   Adj Close  \
0   2015-05-01  538.429993  544.190002  521.085022  532.109985  532.109985
1   2015-06-01  536.789978  543.739990  520.500000  520.510010  520.510010
2   2015-07-01  524.729980  678.640015  515.179993  625.609985  625.609985
3   2015-08-01  625.340027  674.900024  565.049988  637.609985  637.609985
4   2015-09-01  602.359985  650.900024  589.380005  608.419983  608.419983
5   2015-10-01  608.369995  730.000000  599.849976  710.809998  710.809998
6   2015-11-01  711.059998  762.708008  705.849976  742.599976  742.599976
7   2015-12-01  747.109985  779.979980  724.169983  758.880005  758.880005
8   2016-01-01  743.000000  752.000000  673.260010  742.950012  742.950012
9   2016-02-01  750.460022  789.869995  663.059998  697.770020  697.770020
10  2016-03-01  703.619995  757.880005  685.340027  744.950012  744.950012
11  2016-04-01  738.599976  769.900024  689.000000  693.010010  693.010010
```

```
12  2016-05-01  697.630005  739.729980  689.010010  735.719971  735.719971
13  2016-06-01  734.530029  737.210022  663.283997  692.099976  692.099976
14  2016-07-01  692.200012  778.549988  688.215027  768.789978  768.789978

      Volume
0   31865900
1   34320500
2   63319000
3   61434900
4   47955000
5   49438600
6   33934800
7   43100300
8   46561200
9   64367000
10  41742400
11  42154000
12  34996500
13  39362500
14  29914800
```

```
In [4]: #lunghezza dati
        len(goog)
```

```
Out[4]: 58
```

1.0.1 Elenco delle variabili nel dataset

Mostra il nome delle colonne del dataset.

```
In [5]: goog.columns
```

```
Out[5]: Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'], dtype='object')
```

1.0.2 Creazione della struttura dati serie storica (Time Series)

Prende colonna "date" e crea un oggetto date-time.

```
In [7]: goog_time = pd.to_datetime(goog['Date'])
        goog_time.head(5)
```

```
Out[7]: 0   2015-05-01
        1   2015-06-01
        2   2015-07-01
        3   2015-08-01
        4   2015-09-01
        Name: Date, dtype: datetime64[ns]
```

Crea oggetto "series", ovvero l'oggetto serie storiche.

```
In [8]: goog_ope = pd.Series(goog['Open'])
        #goog_ope.index =goog['Date']
        goog_ope.index = range(0,58)

        #prime dieci osservazioni
        goog_ope.head(10)
```

```
Out[8]: 0    538.429993
        1    536.789978
        2    524.729980
        3    625.340027
        4    602.359985
        5    608.369995
        6    711.059998
        7    747.109985
        8    743.000000
        9    750.460022
        Name: Open, dtype: float64
```

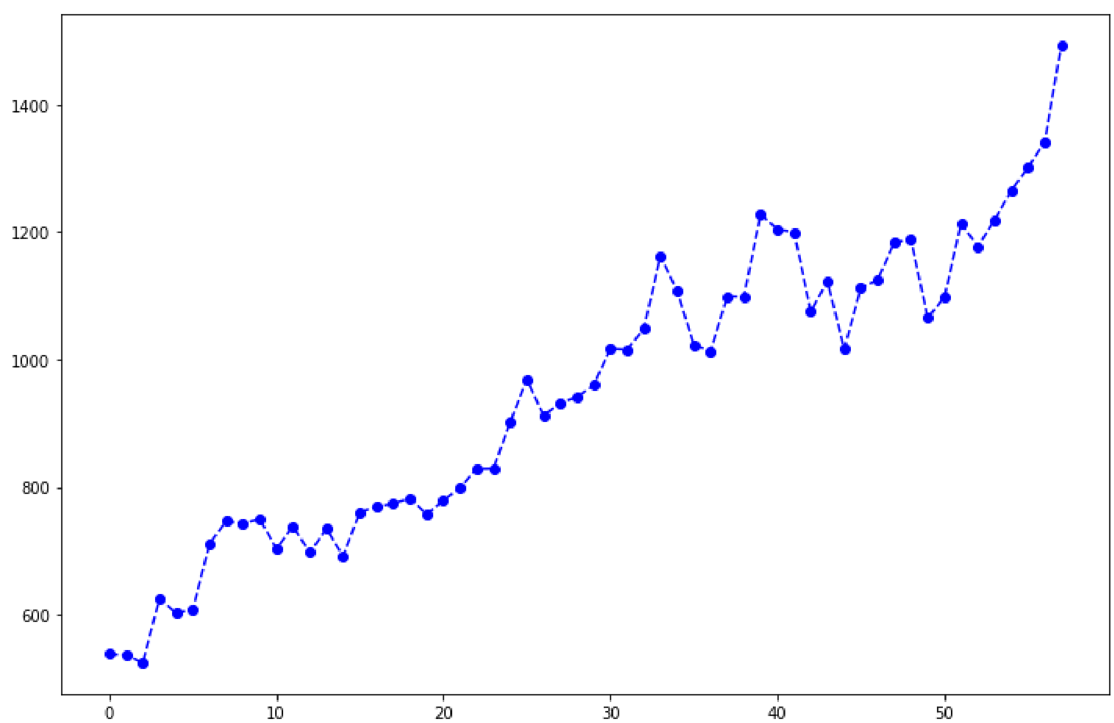
L'oggetto contiene numeri rappresentati in virgola mobile, come il float in C.

1.0.3 La serie storica prezzi all'apertura

"plt." è la libreria matlib ".figure" crea il canvas (la tela) ".plot" significa "disegna in piani cartesiani"

```
In [9]: fig =plt.figure(figsize = (12,8))
        goog_ope.plot(linestyle='--',color='b',marker="o")
```

```
Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb243be3250>
```



1.0.4

2 La retta che spiega i dati osservati nel tempo

x_j è il mese jesimo considerato;
y_j è il valore di Google in borsa il primo del mese jesimo considerato;

Dobbiamo trovare i coefficienti "migliori" del fascio di infinite rette possibili: la **retta di regressione**.

β_0 è la coordinata all'origine;
β_1 è la il coeeficiente angolare;
e_j è lo scostamento dalla retta di regressione dovuto a fluttuazione specifica di quel mese;
Il modello per l'osservazione del mese j-esimo è:

$$y_j = \beta_0 + \beta_1 \cdot x_j + e_j$$

2.1 Come assegnare ai paremetri incogniti un valore?

La migliore retta secondo i "minimi quadrati" si ottiene minimizzando $Q(\beta_0, \beta_1)$ rispetto ai parametri beta.
L'errore globale di sostituzione delle osservazioni originali con il valore corrispondente che giace sulla retta è:

$$Q(\beta_0, \beta_1) = \sum_j (y_j - \beta_0 - \beta_1 \cdot x_j)^2$$

Per poter avere una quantità globale di errore, elevo al quadrato perchè così posso sommare le quantità di errore senza che si elidano (per poter togliere il segno per fare la somma).

2.1.1 La retta di regressione: un modello statistico

Funzione che permette di creare l'indice di ciascun mese (sintassi della libreria).

```
In [10]: import numpy as np
         import statsmodels.api as sm
         mesi = range(58)
         goog_data = sm.add_constant(mesi)
         # print(goog_data)
```

".fit" calcola il ".summary" riassuno del modello.

```
In [11]: # Stima  OLS model: i minimi quadrati ordinari
         mod1 = sm.OLS(goog_ope, goog_data)
         res = mod1.fit()          #calcolo il beta0 e il beta1 migliore
         print(res.summary())

         #Mesi osservati -58-
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   Open   R-squared:                       0.920
Model:                            OLS   Adj. R-squared:                  0.919
Method:                 Least Squares   F-statistic:                     645.4
Date:                Wed, 15 Apr 2020   Prob (F-statistic):           2.02e-32
Time:                        14:18:22   Log-Likelihood:                -324.15
No. Observations:                  58   AIC:                             652.3
Df Residuals:                      56   BIC:                             656.4
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        572.0737     17.072     33.510      0.000     537.875     606.273
x1            13.1219      0.517     25.405      0.000      12.087      14.157
==============================================================================
Omnibus:                        3.339   Durbin-Watson:                   0.790
Prob(Omnibus):                  0.188   Jarque-Bera (JB):                2.384
Skew:                           0.413   Prob(JB):                        0.304
Kurtosis:                       3.550   Cond. No.                         65.3
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

const: $\beta_0 = 572.0737$
x1: $\beta_1 = 13.1219$
Stampa parametri a parte.

```
In [12]: # stima dei parametri del modello
         print('Parameters:\n\n',res.params)
```

```
Parameters:

 const    572.073746
x1        13.121883
dtype: float64
```

```
In [13]: # errore standard della stima
         print('Standard errors:\n\n ', res.bse)
```

```
Standard errors:

  const    17.071913
x1        0.516501
dtype: float64
```

```
In [14]: # valori predetti dal modello (punti sulla retta per ogni mese considerato)
         print('Predicted values: ', res.predict())
```

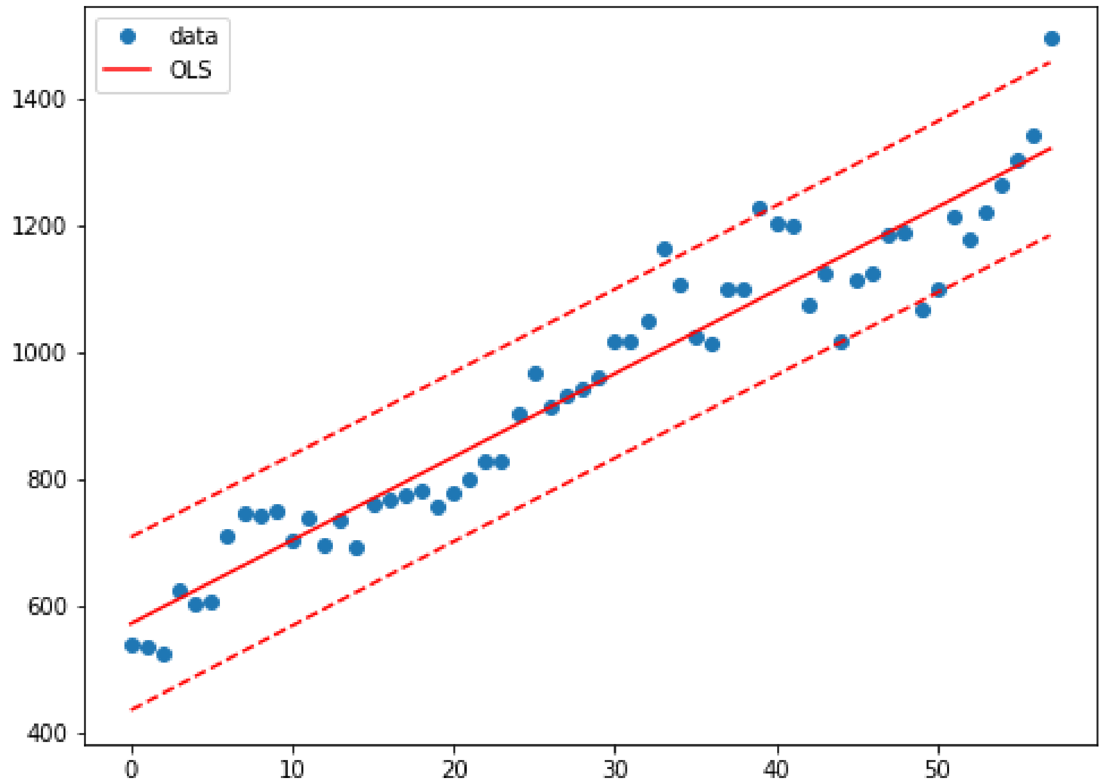
```
Predicted values:  [ 572.07374635  585.1956295   598.31751264  611.43939578  624.56127892
  637.68316207  650.80504521  663.92692835  677.04881149  690.17069464
  703.29257778  716.41446092  729.53634406  742.65822721  755.78011035
  768.90199349  782.02387663  795.14575978  808.26764292  821.38952606
  834.5114092   847.63329235  860.75517549  873.87705863  886.99894177
  900.12082492  913.24270806  926.3645912   939.48647434  952.60835749
  965.73024063  978.85212377  991.97400691 1005.09589005 1018.2177732
 1031.33965634 1044.46153948 1057.58342262 1070.70530577 1083.82718891
 1096.94907205 1110.07095519 1123.19283834 1136.31472148 1149.43660462
 1162.55848776 1175.68037091 1188.80225405 1201.92413719 1215.04602033
 1228.16790348 1241.28978662 1254.41166976 1267.5335529  1280.65543605
 1293.77731919 1306.89920233 1320.02108547]
```

Per ogni mese "Xj" ottengo "YJ", il valore che giace sulla retta: il valore atteso secondo il modello.

2.1.2 Plot della retta stimata con i punti originali in assi cartesiani

Sintassi della libreria. Tra le due rette tratteggiate è raccolta la quasi totalità dei punti.

```
In [15]: from statsmodels.sandbox.regression.predstd import wls_prediction_std

         prstd, iv_l, iv_u = wls_prediction_std(res)
         fig, ax = plt.subplots(figsize=(8,6))
         ax.plot(mesi, goog_ope, 'o', label="data")
         ax.plot(mesi, res.fittedvalues, 'r-', label="OLS")
         ax.plot(mesi, iv_u, 'r--')
         ax.plot(mesi, iv_l, 'r--')
         ax.legend(loc='best');
```




```
In [16]: mesi
```

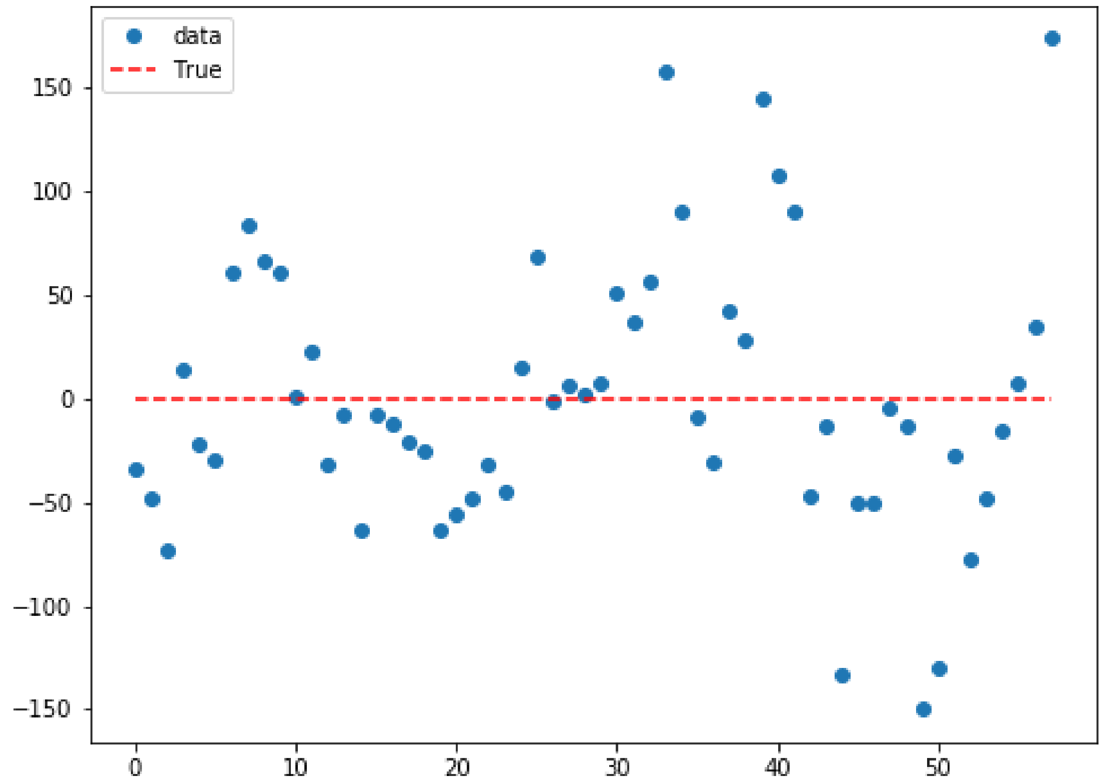
```
Out[16]: range(0, 58)
```

2.1.3 Residuo j-esimo

$$e_j = (y_j - \beta_0 - \beta_1 \cdot x_j)$$

ovvero valore osservato j-esimo meno valore che giace sulla retta al-jesimo mese (valore previsto).

```
In [17]: # residui  e_1, e_2 , ...., e_j;..., e_n

         valori_e = goog_ope - res.fittedvalues
         fig, ax = plt.subplots(figsize=(8,6))
         retta_oriz = 0.0*goog_ope
         ax.plot(mesi, valori_e, 'o', label="data")
         ax.plot(mesi, retta_oriz, 'r--', label="True")
         #ax.plot(mesi, iv_l, 'r--')
         ax.legend(loc='best');
```



La retta tratteggiata è la retta di riferimento.

I residui si distribuiscono a grupi sopra e sotto la retta rossa di riferimento: come la traccia di un serpente!!! La retta non è un buon modello!!!
I residui, cioè il rumore, non è "bianco" ma mostra una struttura, ovvero una regolarità!

"Bianco" indica l'alternarsi dei residui lungo i mesi è del tutto casuale.
"struttura" va tolta dai residui e inclusa nel modello.

3

3.0.1 Regressione polinomiale: quando una retta non basta!

x_j è il mese jesimo considerato;
${x_j}^2$ è il mese jesimo considerato elevato al quadrato;
${x_j}^3, {x_j}^4, {x_j}^5$ sono le potenze successive;

y_j è il valore di Google in borsa il primo del mese jesimo considerato;

Dobbiamo trovare i coefficienti "migliori" delle infinite funzioni possibili: **il polinomio di regressione**.

β_0 è la coordinata all'origine;
β_1 è la il coefficiente del termine lineare;
β_2 è la il coefficiente del termine di secondo grado;
β_i è la il coefficiente del termine di grado *i*esimo;

e_j è lo scostamento dell'osservazione y_j dalla retta di regressione;
Il modello per l'osservazione del mese j-esimo è:

$$y_j = \beta_0 + \beta_1 \cdot x_j + \beta_2 \cdot {x_j}^2 + \beta_3 \cdot {x_j}^3 + \beta_4 \cdot {x_j}^4 + \beta_5 \cdot {x_j}^5 + e_j$$

```
-> Sono curve definite da polinomi posibili di 5ř grado.
```

Creazione dataframe.

```
In [18]: Gdf =pd.DataFrame({"Y":goog_ope,
                           "Mesi": mesi,
                           "Mesi2": [aux**2 for aux in range(58)],
                           "Mesi3": [aux**3 for aux in range(58)],
                           "Mesi4": [aux**4 for aux in range(58)],
                           "Mesi5": [aux**5 for aux in range(58)]
                          })
         Gdf.head(6)
```

Out[18]:

	Y	Mesi	Mesi2	Mesi3	Mesi4	Mesi5
0	538.429993	0	0	0	0	0
1	536.789978	1	1	1	1	1
2	524.729980	2	4	8	16	32
3	625.340027	3	9	27	81	243
4	602.359985	4	16	64	256	1024
5	608.369995	5	25	125	625	3125

4

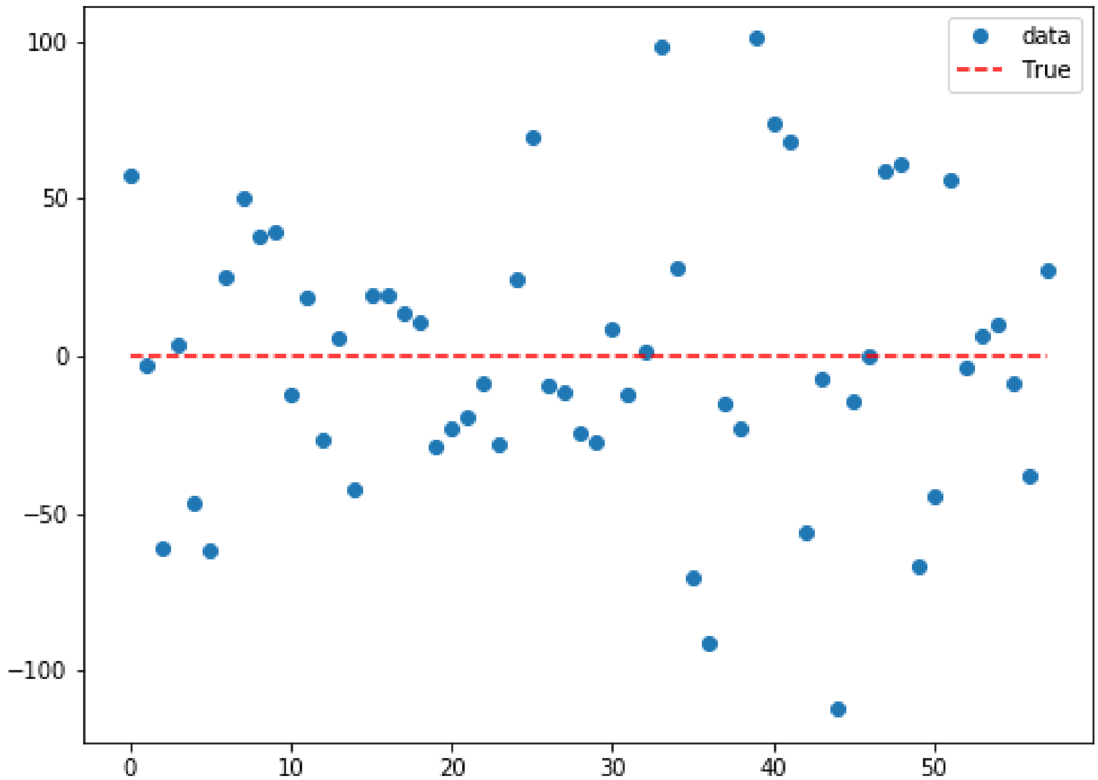
```
In [19]: from patsy import dmatrices
         y, X = dmatrices('Y ~ Mesi+Mesi2+Mesi3+Mesi4+Mesi5', data=Gdf, return_type='dataframe
         mod = sm.OLS(y, X)     # Describe model

         res2 = mod.fit()       # Fit model

         print(res2.summary())    # Summarize model
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      Y   R-squared:                       0.963
Model:                            OLS   Adj. R-squared:                  0.959
Method:                 Least Squares   F-statistic:                     269.4
Date:                Wed, 15 Apr 2020   Prob (F-statistic):           6.79e-36
Time:                        14:19:21   Log-Likelihood:                -301.98
No. Observations:                  58   AIC:                             616.0
Df Residuals:                      52   BIC:                             628.3
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
```

```
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    481.2602     31.829     15.120      0.000     417.390     545.130
Mesi          64.7449     11.626      5.569      0.000      41.416      88.073
Mesi2         -6.9857      1.292     -5.408      0.000      -9.578      -4.394
Mesi3          0.3543      0.058      6.106      0.000       0.238       0.471
Mesi4         -0.0074      0.001     -6.599      0.000      -0.010      -0.005
Mesi5       5.455e-05   7.86e-06      6.939      0.000    3.88e-05    7.03e-05
==============================================================================
Omnibus:                        0.184   Durbin-Watson:                   1.547
Prob(Omnibus):                  0.912   Jarque-Bera (JB):                0.026
Skew:                           0.052   Prob(JB):                        0.987
Kurtosis:                       2.995   Cond. No.                     1.03e+09
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.03e+09. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Sintassi

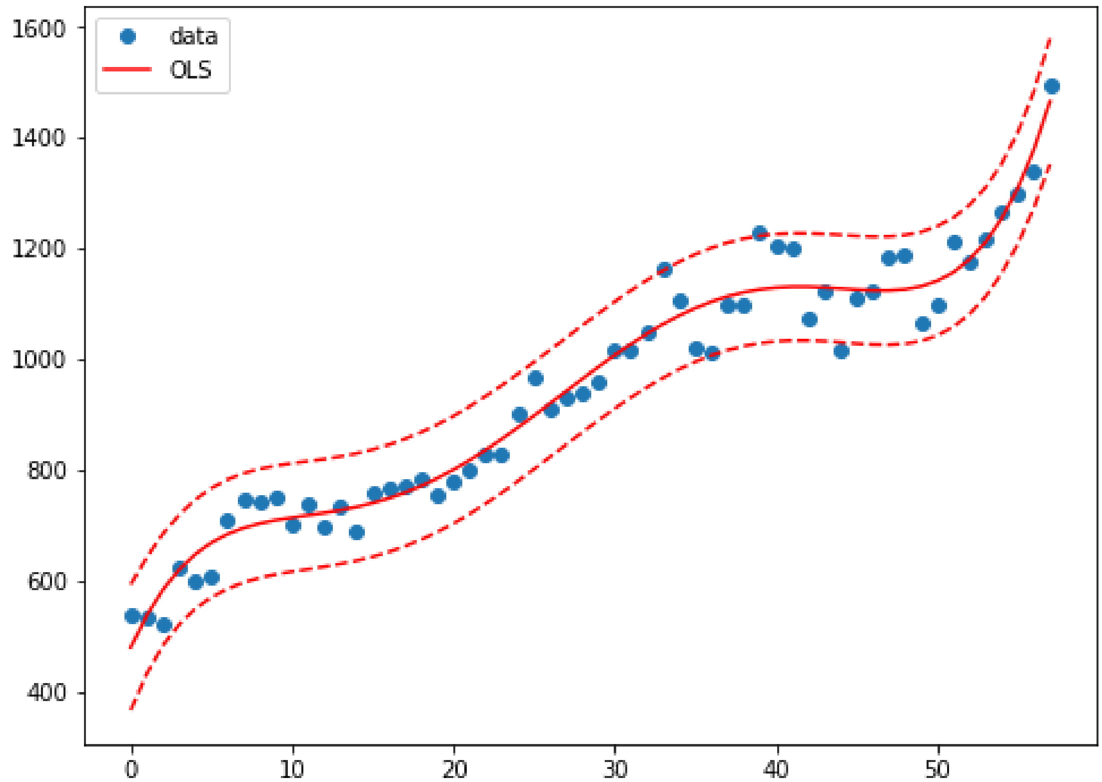
```
In [20]: valori_e2 = Gdf['Y'] - res2.fittedvalues
         fig, ax = plt.subplots(figsize=(8,6))
         retta_oriz = 0.0*goog_ope
         ax.plot(mesi, valori_e2, 'o', label="data")
         ax.plot(mesi, retta_oriz, 'r--', label="True")
         #ax.plot(mesi, iv_l, 'r--')
         ax.legend(loc='best');
```



I residui sopra riportati in grafico non mostrano più la struttura presente quando il modello è una retta.

Abbiamo tolto l'ondulazione dei residui e messa nel modello.

```
In [21]: prstd, iv_l, iv_u = wls_prediction_std(res2)
         #y_true = np.dot(goog_data, res.params)
         fig, ax = plt.subplots(figsize=(8,6))

         ax.plot(mesi, goog_ope, 'o', label="data")
         #ax.plot(mesi, y_true, 'b-', label="True")
         ax.plot(mesi, res2.fittedvalues, 'r-', label="OLS")
         ax.plot(mesi, iv_u, 'r--')
         ax.plot(mesi, iv_l, 'r--')
         ax.legend(loc='best');
```



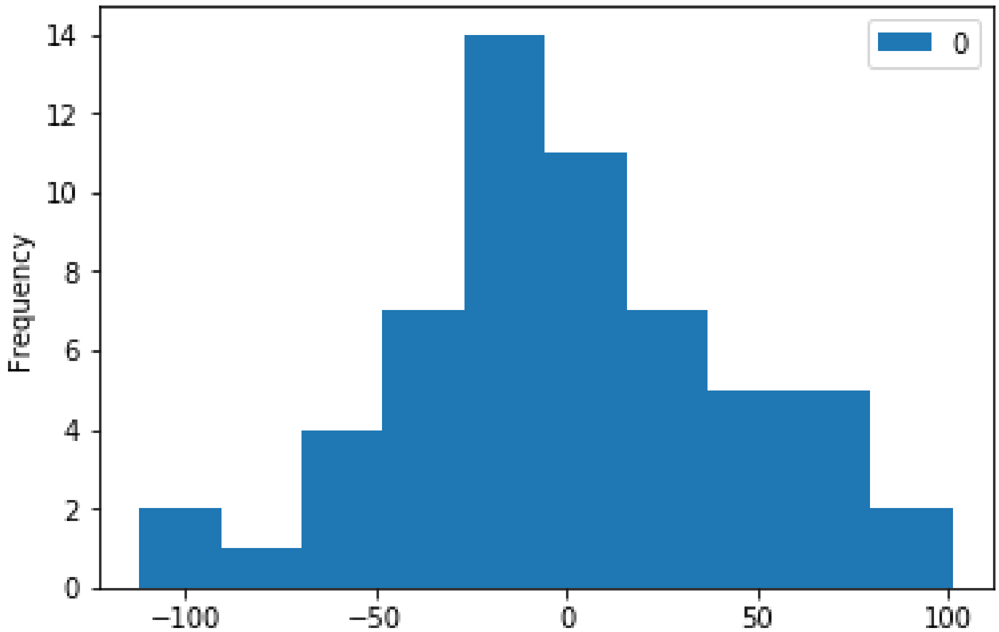
58 residui -> 58 osservazioni
Il valore cresce nel tempo ma non segue precisamente una retta!

```
In [22]: # pd.DataFrame({'residui':valori_e2}).var()
         pd.DataFrame({'residui':valori_e2}).describe()
```

```
Out[22]:             residui
         count   5.800000e+01
         mean    9.132306e-07
         std     4.453815e+01
         min    -1.120868e+02
         25%    -2.586021e+01
         50%    -3.160732e+00
         75%     2.516145e+01
         max     1.010026e+02
```

```
In [23]: # Istogramma dei residui
         pd.DataFrame(valori_e2).plot.hist()
```

```
Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb23d1312d0>
```



Conto quanti residui stanno in ciascun intervallo di base, facciamo quindi un istrogramma (sommando tutte le altezze otteniamo 58).

```
In [26]: # deviazione standard
         devstd2 = pd.DataFrame({'residui':valori_e2}).describe()['residui'][2]
         devstd2

Out[26]: 44.53814713156571
```

4.0.1

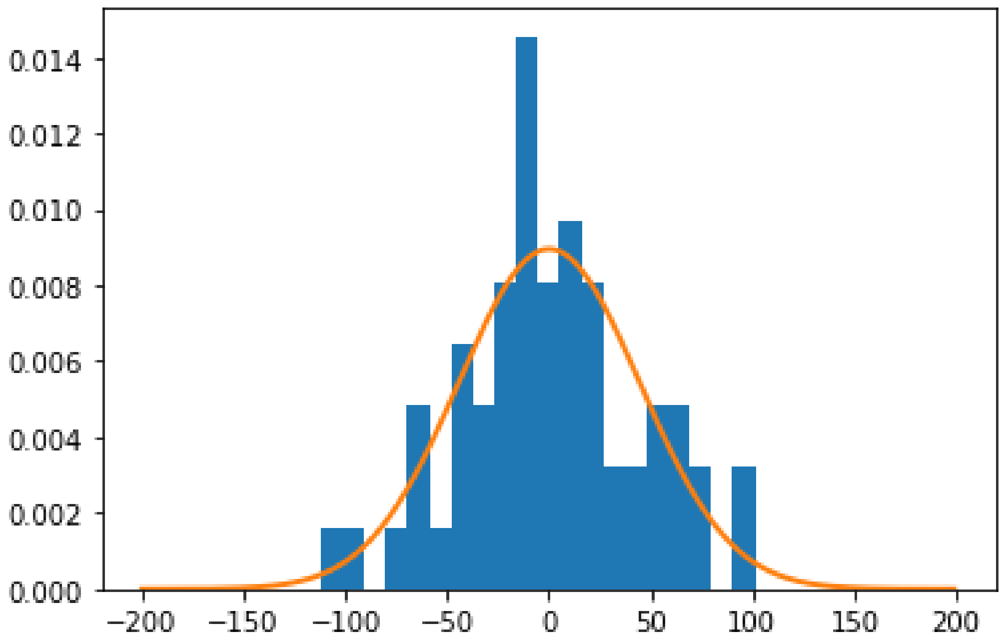
4.0.2

4.1 Il modello matematico noto come funzione di densità di probabilità normale è adeguato?

Sintassi codice per disegno.

```
In [24]: #import pandas as pd
         import matplotlib.pyplot as plt

         Maxv=120

         #plt.subplot(211)
         #plt.hist(valori_e2,bins=20, range=(-Maxv, Maxv), color='g')
         #plt.xlabel("Residui")
         #pd.DataFrame({'residui':valori_e2}).residui.plot(kind='kde', ax=ax, secondary_y=True
```

```
import scipy.stats as ss
import numpy as np

griglia = [aux for aux in range(-200,200,1)]
loc, sigma=0, devstd2
dataNOR =ss.norm.pdf(griglia,loc=loc,scale=sigma)

myHist = plt.hist(valori_e2, 20, density=True)

#x = np.linspace(-200,200)
h = plt.plot(griglia, dataNOR, lw=2)
plt.show()
```



Una funzione matematica detta di densità di probabilità è una curva che approssima, eventualmente, un istogramma di frequenze relative. L'istogramma di frequenze relative assomiglia al modello teorico a campana Gaussiana ... ma non troppo.

4.1.1 Generazione di un campione casuale dalla Gaussiana via simulazione Monte Carlo

"np" è libreria.

"random.normal" è l'estrazione in modo casuale.

Abbiamo generato 10000 residui.

```
In [27]: # Curva teorica Normale (Gaussiana)
         comp1 = np.random.normal(0, devstd2, size=10000)

         values = pd.Series(comp1)
         values.describe()
```

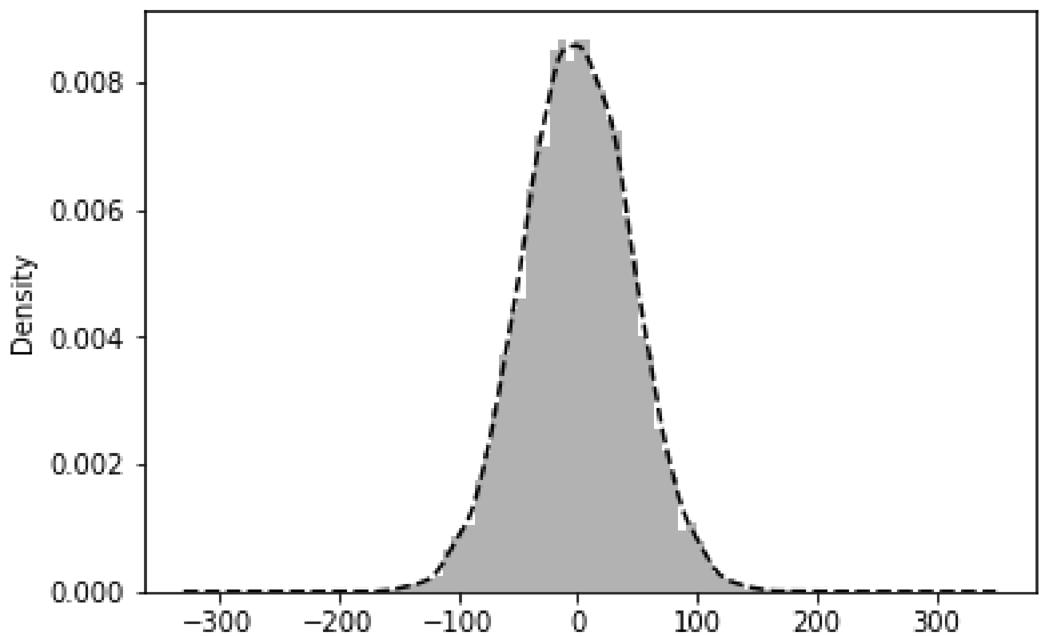
```
Out[27]: count    10000.000000
         mean        -0.318178
         std         44.995107
         min       -159.299987
         25%        -30.901456
         50%         -0.462965
         75%         30.534306
         max        179.686229
         dtype: float64
```

5

5.0.1 Sovraimposizione della Gaussiana al campione simulato

```
In [28]: values.hist(bins=50, alpha=0.3, color='k', density=True)
         values.plot(kind='kde', style='k--')
```

```
Out[28]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb23cdf7bd0>
```




```
In [29]: # Stima puntuale dei parametri della curva
         res2.params
```

```
Out[29]: Intercept    481.260198
         Mesi          64.744925
         Mesi2         -6.985727
         Mesi3          0.354270
```

```
Mesi4          -0.007431
Mesi5           0.000055
dtype: float64
```

-> sono i $_beta$

```
In [30]: # dataset futuro
         grigliapre = [aux*1.0 for aux in range(0,64)]
         grigliapre2 = [aux**2.0 for aux in range(0,64)]
         grigliapre3 = [aux**3.0 for aux in range(0,64)]
         grigliapre4 = [aux**4.0 for aux in range(0,64)]
         grigliapre5 = [aux**5.0 for aux in range(0,64)]

         datPred=pd.DataFrame({ 'Mesi': grigliapre,
                     'Mesi2': grigliapre2,
                     'Mesi3': grigliapre3,
                     'Mesi4': grigliapre4,
                     'Mesi5': grigliapre5
                     })

         matPred=sm.add_constant(datPred)
         matPred.head(5)
```

Out[30]:

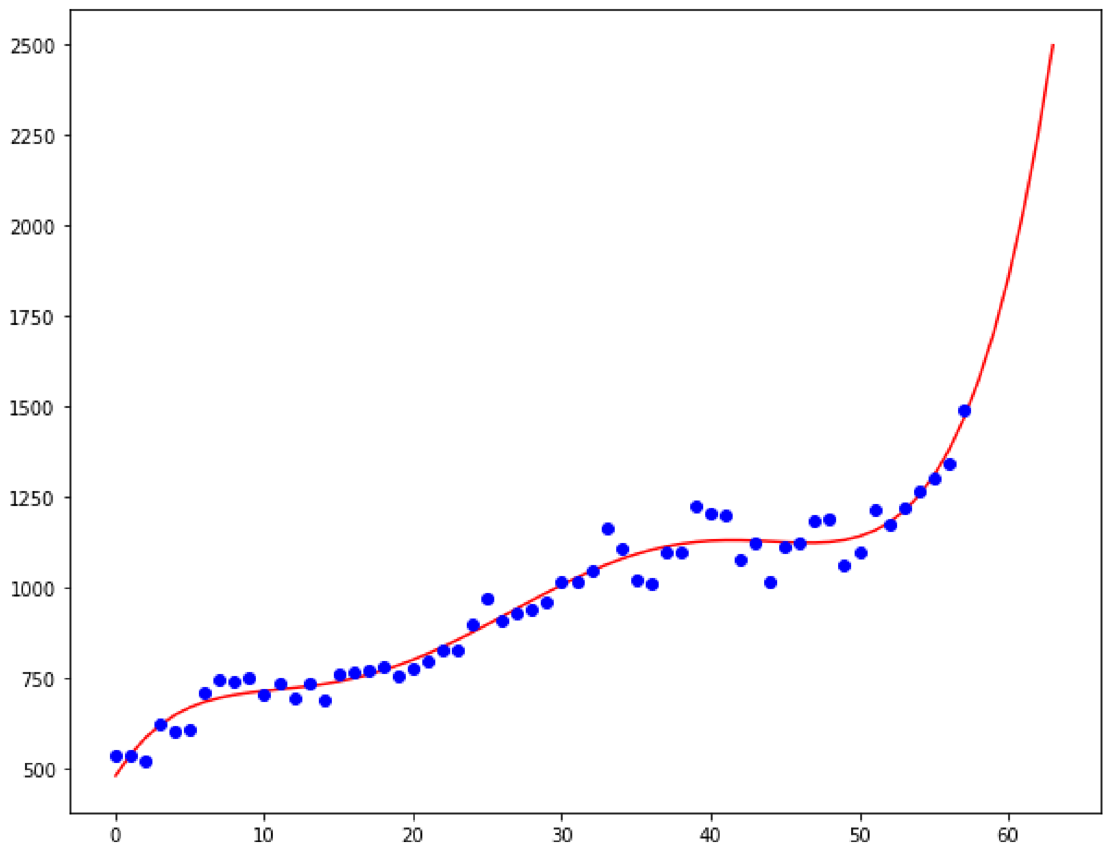
	const	Mesi	Mesi2	Mesi3	Mesi4	Mesi5
0	1.0	0.0	0.0	0.0	0.0	0.0
1	1.0	1.0	1.0	1.0	1.0	1.0
2	1.0	2.0	4.0	8.0	16.0	32.0
3	1.0	3.0	9.0	27.0	81.0	243.0
4	1.0	4.0	16.0	64.0	256.0	1024.0

```
In [31]: attese_futuro = np.dot( matPred,res2.params)
         attese_futuro
```

```
Out[31]: array([ 481.26019824,  539.36628964,  585.5241536 ,  621.60008625,
                 649.2951368 ,  670.15165358,  685.55982989,  696.76425003,
                 704.8704352 ,  710.85138948,  715.55414578,  719.70631179,
                 723.9226159 ,  728.71145321,  734.48143143,  741.54791688,
                 750.1395804 ,  760.40494329,  772.41892334,  786.18938069,
                 801.66366384,  818.73515558,  837.24981895,  857.01274317,
                 877.79468964,  899.33863782,  921.36633126,  943.58482349,
                 965.69302402,  987.38824424, 1008.37274342, 1028.36027463,
                1047.08263071, 1064.29619021, 1079.78846335, 1093.38463797,
                1104.95412547, 1114.41710679, 1121.75107833, 1126.99739791,
                1130.26783075, 1131.75109538, 1131.71940962, 1130.5350365 ,
                1128.65683028, 1126.64678231, 1125.17656705, 1125.034088  ,
                1127.13002365, 1132.50437342, 1142.33300365, 1157.9341935 ,
                1180.77518095, 1212.47870871, 1254.82957022, 1309.78115554,
                1379.46199736, 1466.18231692, 1572.44056997, 1700.92999271,
                1854.54514777, 2036.38847013, 2249.7768131 , 2498.24799425])
```

In [32]:
```
# Attesa valori futuri non ragionevole !!! Modello globale 'rigido' tempi remoti e vi
fig, ax = plt.subplots(figsize=(10,8))
#ax.plot(mesi, y_true, 'b-', label="True")
ax.plot(grigliapre, attese_futuro, 'r-', label="OLS")
ax.plot(mesi, goog_ope, 'o', color='b',label="data")
#ax.plot(mesi, iv_u, 'r--')
#ax.plot(mesi, iv_l, 'r--')
#ax.legend(loc='best');
```

Out[32]:
```
[<matplotlib.lines.Line2D at 0x7fb23cd99750>]
```



5.0.2

5.1

5.2 Quanto vale l'incertezza intorno al valore atteso futuro?

Componenti dell'incertezza sul valore futuro:

1. I parametri inseriti nel calcolo della curva sono stime del loro "vero" valore incognito, sono quindi affetti da incertezza.

2. Esiste una fluttuazione anche nel futuro mese di interesse.

E' possibile eseguire una simulazione Monte Carlo per estrarre casualmente da distribuzioni Gaussiane delle realizzazioni dei parametri e di calcolare i valori attesi futuri per ciascuna tupla simulata in un intorno plausibile dei valori ottenuti con la stima puntuale.

5.3

5.4

5.5 Fluttuazioni scambiabili

L'idea è che le fluttuazioni sono casuali nel senso che quelle future "assomigliano" a quelle passate (sono scambiabili).

Allora simulando un campione Monte Carlo dalla distribuzione futura dei valori previsti sommeremo a ciascun valore simulato uno dei residui ottenuti dal nostro campione originale dopo avere stimato i parametri del modello.

```
In [33]: import random
         campio1 =np.random.choice([1,5,1,0], size=10, replace=True, p=None)
         campio2 =np.random.choice([1,5,1,0], size=10, replace=True, p=None)
         campio3 =np.random.choice([1,5,1,0], size=10, replace=True, p=None)

         [campio1,
          campio2,
          campio3]

Out[33]: [array([1, 0, 5, 1, 1, 1, 1, 1, 0, 1]),
          array([0, 5, 5, 0, 0, 5, 1, 1, 5, 1]),
          array([1, 5, 0, 1, 0, 1, 0, 1, 0, 5])]

In [34]: residui_mod_2 = [ aux for aux in valori_e2]
         campio_fut_residuo =np.random.choice(valori_e2, size=10000, replace=True, p=None)
         campio_fut_residuo[0:10]

Out[34]: array([  98.31379479,  101.00260209,    5.81857579,   69.61137418,
                 -3.74515195,   50.34573497,   50.34573497,   13.84505671,
                -22.85366584, -112.08682328])
```

5.6

5.7 Campione Monte Carlo di ipotetici valori attesi futuri

L'incertezza sul valore predetto dovuta ai parametri segue una legge Normale.

Con un esperimento Monte Carlo campioniamo il valore atteso definito dalla curva per 10000 volte, sommiamo il residuo, positivo o negativo che sia dalla precednete simulazione e vediamo 95 volte su cento entro quale intervallo di valori la previsione del prossimo mese è situata.

In [35]:
```
# stime puntuali dei parametri
res2.params
```

Out[35]:
```
Intercept    481.260198
Mesi          64.744925
Mesi2         -6.985727
Mesi3          0.354270
Mesi4         -0.007431
Mesi5          0.000055
dtype: float64
```

In [36]:
```
# errori standard delle stime puntuali
res2.bse
```

Out[36]:
```
Intercept    31.829089
Mesi         11.625618
Mesi2         1.291810
Mesi3         0.058022
Mesi4         0.001126
Mesi5         0.000008
dtype: float64
```

In [37]:
```
# valore atteso

matPre=np.column_stack((1,59,59**2,59**3,59**4,59**5) )

attesa59=matPre.dot(res2.params)
attesa59[0]
```

Out[37]:
```
1700.9299927103857
```

In [38]:
```
import scipy as sp
covb = res2.cov_params()
prediction_var = res2.mse_resid + (matPre * np.dot(covb,matPre.T).T).sum(1)
prediction_std = np.sqrt(prediction_var)
tppf = sp.stats.t.ppf(0.975, res2.df_resid)
prediction_std
```

Out[38]:
```
array([74.06481216])
```

In [39]:
```
# simulazione Monte Carlo incertezza previsione (attesa)
err_prev= prediction_std
scostamenti_attesa= err_prev* np.random.standard_t(57,10000)
pd.DataFrame(scostamenti_attesa).describe()
```

Out[39]:
```
                  0
count  10000.000000
mean       0.220699
std       75.509557
```

```
min     -280.328318
25%      -50.897361
50%       -0.059315
75%       51.049516
max      262.607893
```

In [40]:
```
# errore complessivo
errore_totale =scostamenti_attesa +campio_fut_residuo
[min(errore_totale),max(errore_totale)]
```

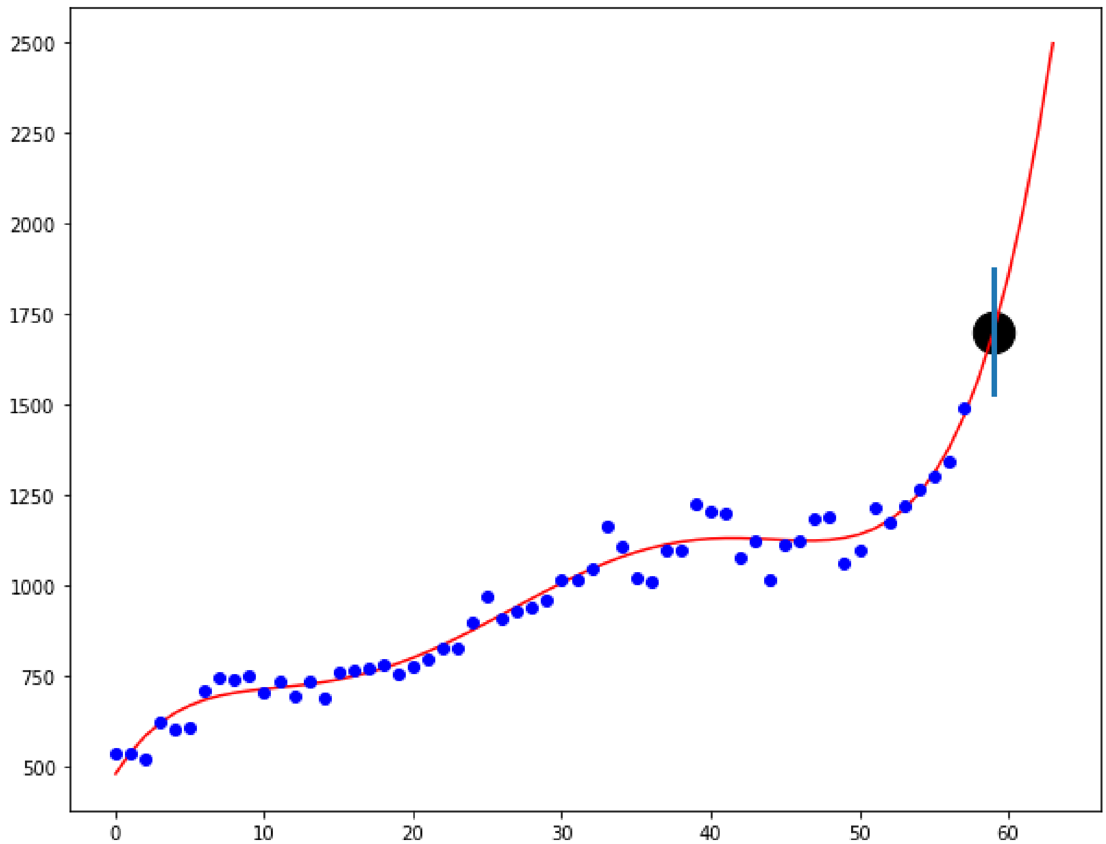
Out[40]:
```
[-351.0527732280508, 306.35892489954284]
```

In [41]:
```
# quantili di interesse
estremi_intervallo = [np.quantile(errore_totale, 0.025,
               axis=None, out=None, overwrite_input=False, interpolation='linear', kee
np.quantile(errore_totale, 0.975,
               axis=None, out=None, overwrite_input=False, interpolation='linear', kee
]

[estremi_intervallo,attesa59[0]+estremi_intervallo[0],attesa59[0]+estremi_intervallo[
```

Out[41]:
```
[[-168.82755914261733, 172.24609920107608],
 1532.1024335677685,
 1873.1760919114618]
```

In [42]:
```
fig, ax = plt.subplots(figsize=(10,8))
ax.plot(grigliapre, attese_futuro, 'r-', label="OLS")
ax.plot(mesi, goog_ope, 'o', color='b',label="data")
plt.scatter(59, attesa59, s=500,marker='o', c='black')
plt.plot((59,59), (attesa59[0]+estremi_intervallo[0],
                   attesa59[0]+estremi_intervallo[1]), linewidth=3)
```

Out[42]:
```
[<matplotlib.lines.Line2D at 0x7fb23d249dd0>]
```



Il punto nero rappresenta la previsione del prossimo mese e la barra verticale l'incertezza abbinata alla previsione.