Análisis de regresiónes

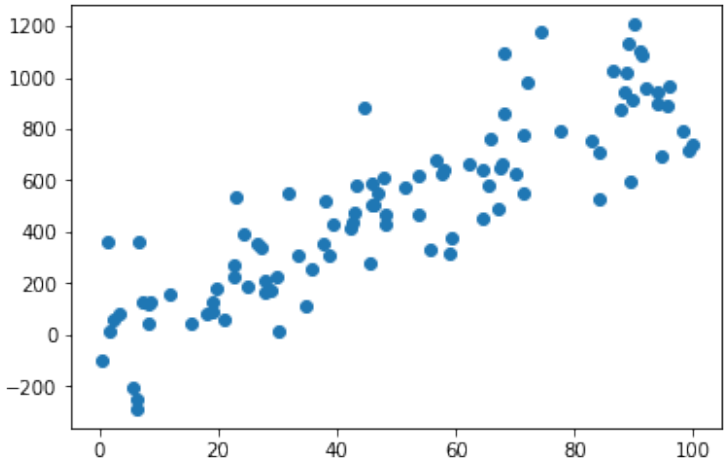
1 Análisis de regresiónes (lineal, polinómica, Ridge y Lasso)

1.1 Regresión Lineal

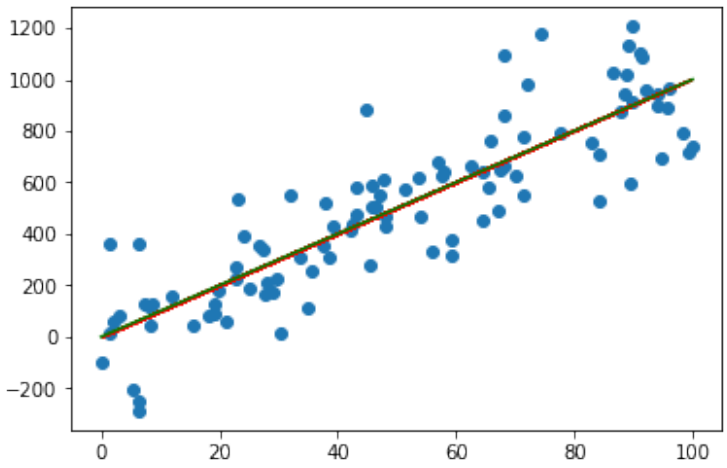
código utizado obtenido de: http://machinelearningparatodos.com/regresion-lineal-en-python/

In [6]:
```
import numpy as np
import random
from sklearn import linear_model
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
%matplotlib inline

# Generador de distribución de datos para regresión lineal simple
def generador_datos_simple(beta, muestras, desviacion):
# Genero n (muestras) valores de x aleatorios entre 0 y 100
    x = np.random.random(muestras) * 100
# Genero un error aleatorio gaussiano con desviación típica (desviacion)
    e = np.random.randn(muestras) * desviacion
# Obtengo el y real como x*beta + error
    y = x * beta + e
    return x.reshape((muestras,1)), y.reshape((muestras,1))

# Parámetros de la distribución
desviacion = 200
beta = 10
n = 100
x, y = generador_datos_simple(beta, n, desviacion)

# Represento los datos generados
plt.scatter(x, y)
plt.show()

# Creo un modelo de regresión lineal
modelo = linear_model.LinearRegression()

# Entreno el modelo con los datos (X,Y)
modelo.fit(x, y)
# Ahora puedo obtener el coeficiente b_1
```

```
print ('Coeficiente beta1: ', modelo.coef_[0])

# Podemos predecir usando el modelo
y_pred = modelo.predict(x)

# Por último, calculamos el error cuadrático medio y el estadístico R^2
print ('Error cuadrático medio: %.2f' % mean_squared_error(y, y_pred))
print ('Estadístico R_2: %.2f' % r2_score(y, y_pred))

# Representamos el ajuste (rojo) y la recta Y = beta*x (verde)
plt.scatter(x, y)
plt.plot(x, y_pred, color='red')
x_real = np.array([0, 100])
y_real = x_real*beta
plt.plot(x_real, y_real, color='green')
plt.show()
```

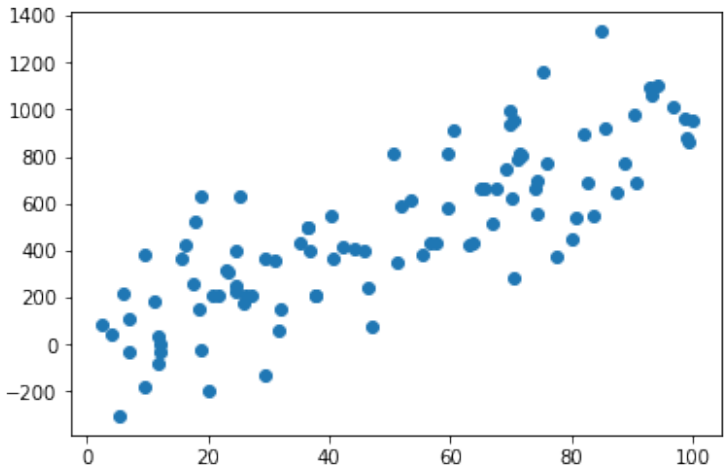



```
Coeficiente beta1:  [10.0329068]
Error cuadrático medio: 27994.72
Estadístico R_2: 0.76
```

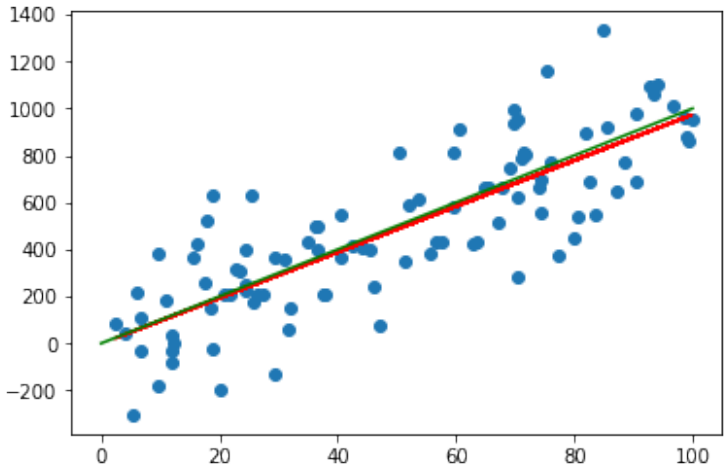


In [7]:
```
import numpy as np
import random
from sklearn import linear_model
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
%matplotlib inline

# Generador de distribución de datos para regresión lineal simple
def generador_datos_simple(beta, muestras, desviacion):
# Genero n (muestras) valores de x aleatorios entre 0 y 100
    x = np.random.random(muestras) * 100
# Genero un error aleatorio gaussiano con desviación típica (desviacion)
    e = np.random.randn(muestras) * desviacion
# Obtengo el y real como x*beta + error
    y = x * beta + e
    return x.reshape((muestras,1)), y.reshape((muestras,1))

# Parámetros de la distribución
desviacion = 200
beta = 10
n = 100
x, y = generador_datos_simple(beta, n, desviacion)

# Represento los datos generados
plt.scatter(x, y)
plt.show()

# Creo un modelo de regresión lineal
modelo = linear_model.LinearRegression()

# Entreno el modelo con los datos (X,Y)
```

```
modelo.fit(x, y)
# Ahora puedo obtener el coeficiente b_1
print ('Coeficiente beta1: ', modelo.coef_[0])

# Podemos predecir usando el modelo
y_pred = modelo.predict(x)

# Por último, calculamos el error cuadrático medio y el estadístico R^2
print ('Error cuadrático medio: %.2f' % mean_squared_error(y, y_pred))
print ('Estadístico R_2: %.2f' % r2_score(y, y_pred))

# Representamos el ajuste (rojo) y la recta Y = beta*x (verde)
plt.scatter(x, y)
plt.plot(x, y_pred, color='red')
x_real = np.array([0, 100])
y_real = x_real*beta
plt.plot(x_real, y_real, color='green')
plt.show()
```

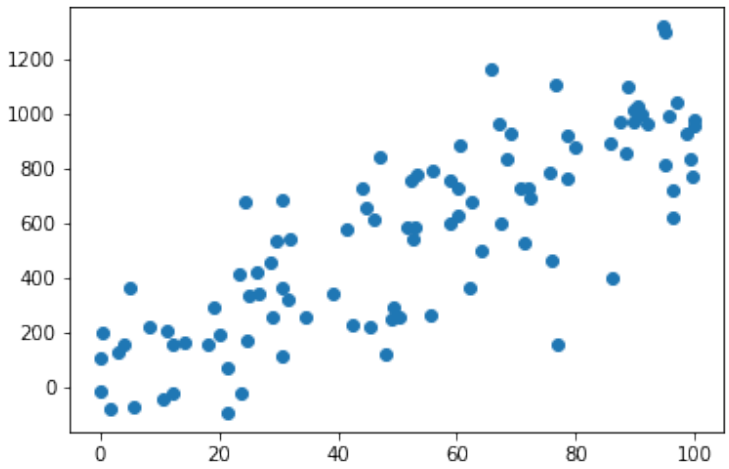



```
Coeficiente beta1:  [9.75709895]
Error cuadrático medio: 38332.93
Estadístico R_2: 0.67
```

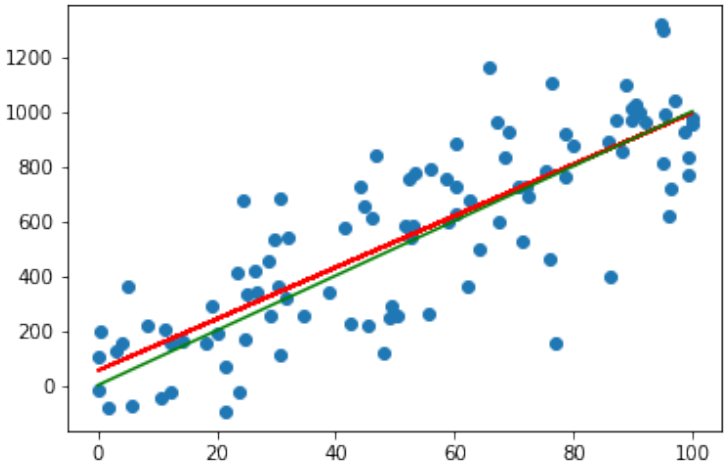


In [8]:
```
import numpy as np
import random
from sklearn import linear_model
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
%matplotlib inline

# Generador de distribución de datos para regresión lineal simple
def generador_datos_simple(beta, muestras, desviacion):
# Genero n (muestras) valores de x aleatorios entre 0 y 100
    x = np.random.random(muestras) * 100
# Genero un error aleatorio gaussiano con desviación típica (desviacion)
    e = np.random.randn(muestras) * desviacion
# Obtengo el y real como x*beta + error
    y = x * beta + e
    return x.reshape((muestras,1)), y.reshape((muestras,1))

# Parámetros de la distribución
desviacion = 200
beta = 10
n = 100
x, y = generador_datos_simple(beta, n, desviacion)

# Represento los datos generados
plt.scatter(x, y)
plt.show()

# Creo un modelo de regresión lineal
modelo = linear_model.LinearRegression()

# Entreno el modelo con los datos (X,Y)
```

```
modelo.fit(x, y)
# Ahora puedo obtener el coeficiente b_1
print ('Coeficiente beta1: ', modelo.coef_[0])

# Podemos predecir usando el modelo
y_pred = modelo.predict(x)

# Por último, calculamos el error cuadrático medio y el estadístico R^2
print ('Error cuadrático medio: %.2f' % mean_squared_error(y, y_pred))
print ('Estadístico R_2: %.2f' % r2_score(y, y_pred))

# Representamos el ajuste (rojo) y la recta Y = beta*x (verde)
plt.scatter(x, y)
plt.plot(x, y_pred, color='red')
x_real = np.array([0, 100])
y_real = x_real*beta
plt.plot(x_real, y_real, color='green')
plt.show()
```

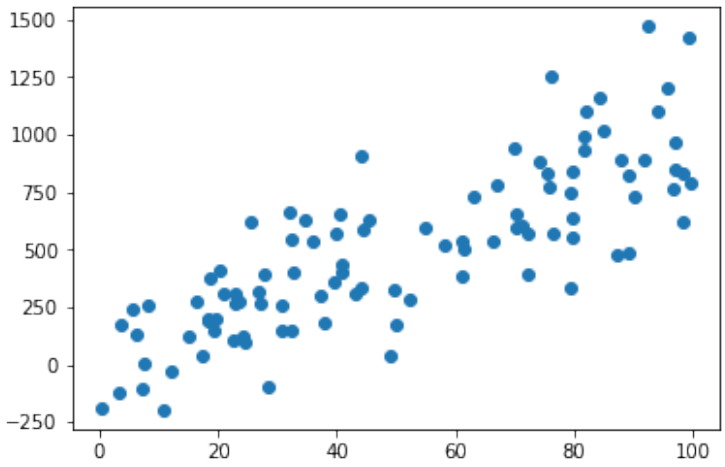



```
Coeficiente beta1:  [9.41198251]
Error cuadrático medio: 40540.33
Estadístico R_2: 0.66
```

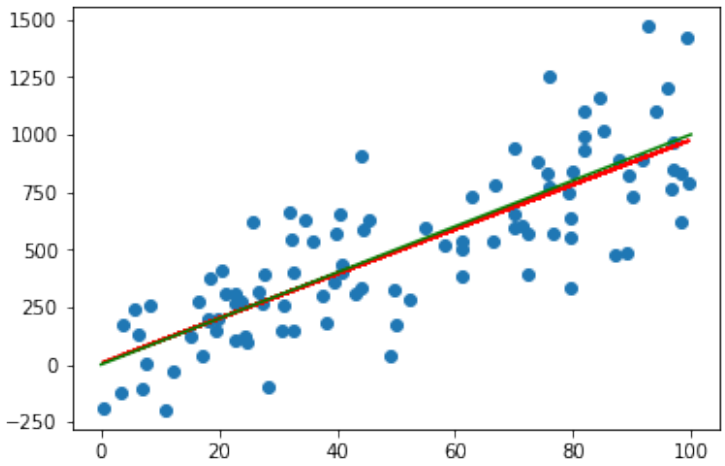


In [9]:
```
import numpy as np
import random
from sklearn import linear_model
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
%matplotlib inline

# Generador de distribución de datos para regresión lineal simple
def generador_datos_simple(beta, muestras, desviacion):
# Genero n (muestras) valores de x aleatorios entre 0 y 100
    x = np.random.random(muestras) * 100
# Genero un error aleatorio gaussiano con desviación típica (desviacion)
    e = np.random.randn(muestras) * desviacion
# Obtengo el y real como x*beta + error
    y = x * beta + e
    return x.reshape((muestras,1)), y.reshape((muestras,1))

# Parámetros de la distribución
desviacion = 200
beta = 10
n = 100
x, y = generador_datos_simple(beta, n, desviacion)

# Represento los datos generados
plt.scatter(x, y)
plt.show()

# Creo un modelo de regresión lineal
modelo = linear_model.LinearRegression()

# Entreno el modelo con los datos (X,Y)
```

```
modelo.fit(x, y)
# Ahora puedo obtener el coeficiente b_1
print ('Coeficiente beta1: ', modelo.coef_[0])

# Podemos predecir usando el modelo
y_pred = modelo.predict(x)

# Por último, calculamos el error cuadrático medio y el estadístico R^2
print ('Error cuadrático medio: %.2f' % mean_squared_error(y, y_pred))
print ('Estadístico R_2: %.2f' % r2_score(y, y_pred))

# Representamos el ajuste (rojo) y la recta Y = beta*x (verde)
plt.scatter(x, y)
plt.plot(x, y_pred, color='red')
x_real = np.array([0, 100])
y_real = x_real*beta
plt.plot(x_real, y_real, color='green')
plt.show()
```

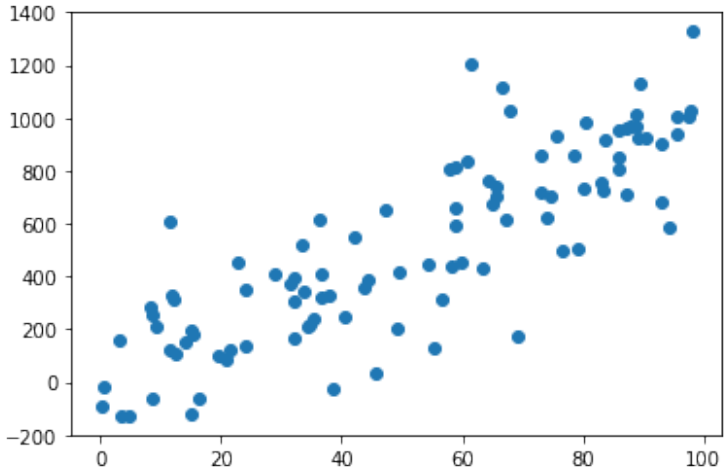



```
Coeficiente beta1:  [9.70398119]
Error cuadrático medio: 43090.87
Estadístico R_2: 0.66
```

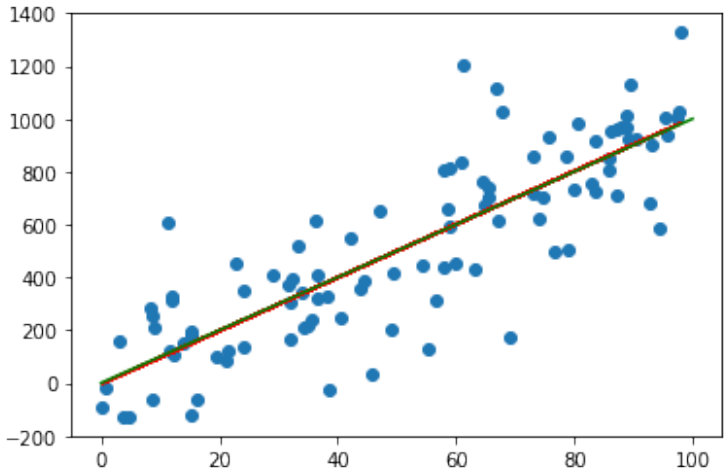



```
In [10]: import numpy as np
         import random
         from sklearn import linear_model
         from sklearn.metrics import mean_squared_error, r2_score
         import matplotlib.pyplot as plt
         %matplotlib inline

         # Generador de distribución de datos para regresión lineal simple
         def generador_datos_simple(beta, muestras, desviacion):
         # Genero n (muestras) valores de x aleatorios entre 0 y 100
             x = np.random.random(muestras) * 100
         # Genero un error aleatorio gaussiano con desviación típica (desviacion)
             e = np.random.randn(muestras) * desviacion
         # Obtengo el y real como x*beta + error
             y = x * beta + e
             return x.reshape((muestras,1)), y.reshape((muestras,1))

         # Parámetros de la distribución
         desviacion = 200
         beta = 10
         n = 100
         x, y = generador_datos_simple(beta, n, desviacion)

         # Represento los datos generados
         plt.scatter(x, y)
         plt.show()

         # Creo un modelo de regresión lineal
         modelo = linear_model.LinearRegression()

         # Entreno el modelo con los datos (X,Y)
```

```
modelo.fit(x, y)
# Ahora puedo obtener el coeficiente b_1
print ('Coeficiente beta1: ', modelo.coef_[0])

# Podemos predecir usando el modelo
y_pred = modelo.predict(x)

# Por último, calculamos el error cuadrático medio y el estadístico R^2
print ('Error cuadrático medio: %.2f' % mean_squared_error(y, y_pred))
print ('Estadístico R_2: %.2f' % r2_score(y, y_pred))

# Representamos el ajuste (rojo) y la recta Y = beta*x (verde)
plt.scatter(x, y)
plt.plot(x, y_pred, color='red')
x_real = np.array([0, 100])
y_real = x_real*beta
plt.plot(x_real, y_real, color='green')
plt.show()
```




```
Coeficiente beta1:  [10.0771589]
Error cuadrático medio: 36523.49
Estadístico R_2: 0.70
```



1.2 Regresión Polinómica

código utizado obtenido de:

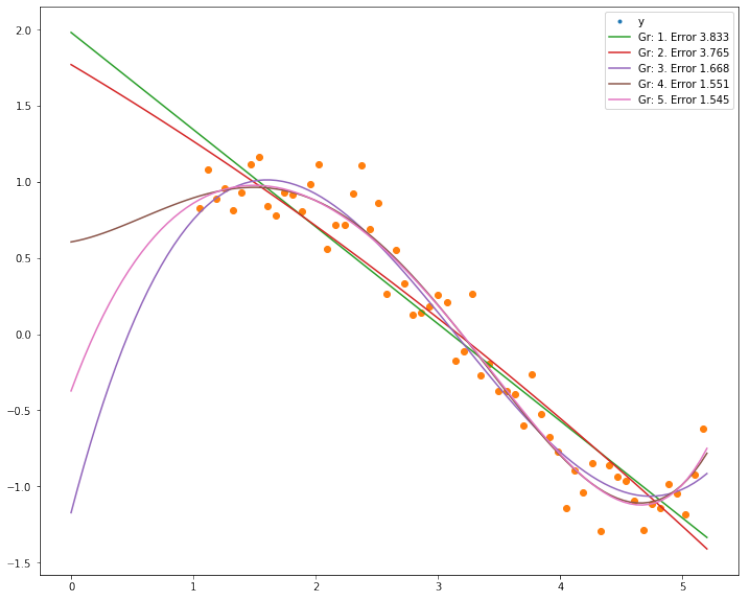
https://es.stackoverflow.com/questions/153866/regresi%C3%B3n-polin%C3%B3mica-en-python?rq=1

https://www.analyticsvidhya.com/blog/2016/01/complete-tutorial-ridge-lasso-regression-python/

```
In [11]: import numpy as np
         import pandas as pd
         import random
         import matplotlib.pyplot as plt
         %matplotlib inline
         from matplotlib.pylab import rcParams
         rcParams['figure.figsize'] = 12, 10

         # Define la matriz de entrada con ángulos de 60 a 300 grados convertidos a radianes
         x = np.array([i*np.pi/180 for i in range(60,300,4)])
         y = np.sin(x) + np.random.normal(0,0.15,len(x))
         data = pd.DataFrame(np.column_stack([x,y]),columns=['x','y'])
         plt.plot(data['x'],data['y'],'.')

         # Calcular ajustes para diferentes grados
         sols = {}
         for grado in range(1,6):
           z = np.polyfit(x, y, grado, full=True)
           sols[grado] = z

         # Imprimir datos
         plt.plot(x, y, 'o')
```

```
# Imprimir curvas de ajuste
xp = np.linspace(0, 5.2, 100)
for grado, sol in sols.items():
  coefs, error, *_ = sol
  p = np.poly1d(coefs)
  plt.plot(xp, p(xp), "-", label="Gr: %s. Error %.3f" % (grado, error) )
plt.legend()
```

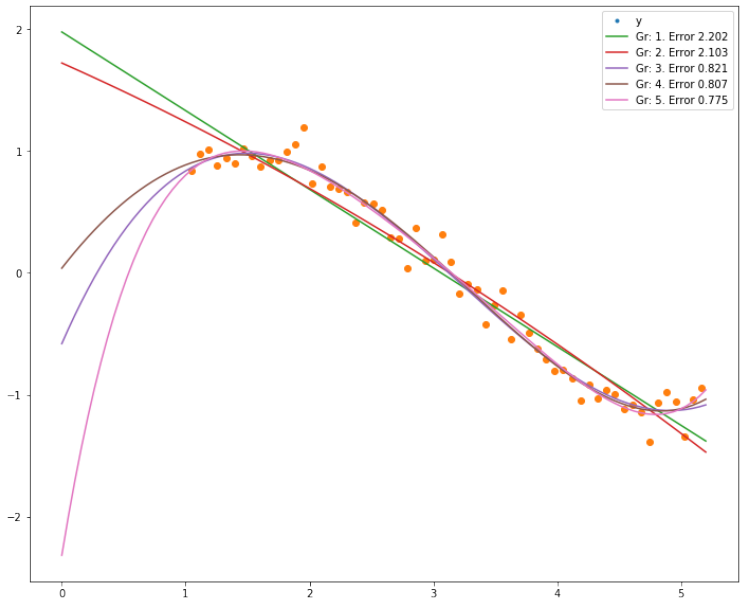
Out[11]: <matplotlib.legend.Legend at 0x2365ace0ef0>



In [12]:
```
import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 12, 10

# Define la matriz de entrada con ángulos de 60 a 300 grados convertidos a radianes
x = np.array([i*np.pi/180 for i in range(60,300,4)])
y = np.sin(x) + np.random.normal(0,0.15,len(x))
data = pd.DataFrame(np.column_stack([x,y]),columns=['x','y'])
plt.plot(data['x'],data['y'],'.')

# Calcular ajustes para diferentes grados
sols = {}
for grado in range(1,6):
  z = np.polyfit(x, y, grado, full=True)
```

```
    sols[grado] = z

# Imprimir datos
plt.plot(x, y, 'o')

# Imprimir curvas de ajuste
xp = np.linspace(0, 5.2, 100)
for grado, sol in sols.items():
    coefs, error, *_ = sol
    p = np.poly1d(coefs)
    plt.plot(xp, p(xp), "-", label="Gr: %s. Error %.3f" % (grado, error) )
plt.legend()
```

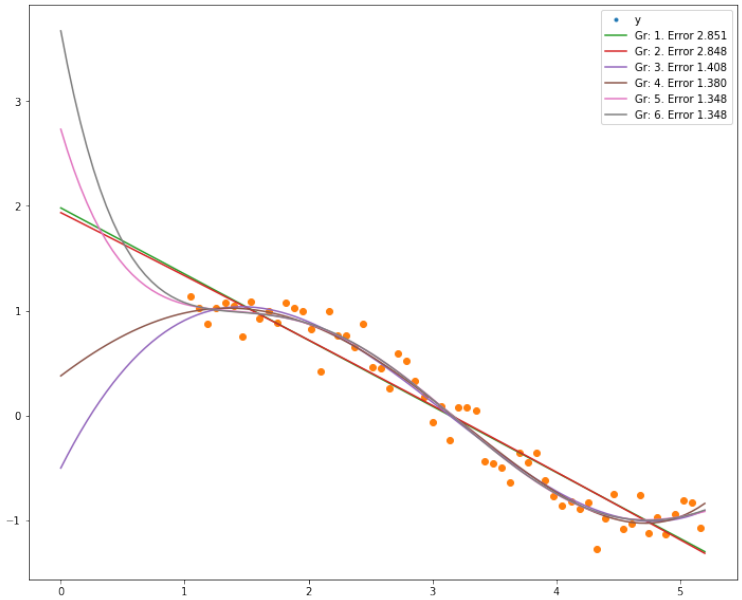
Out[12]: <matplotlib.legend.Legend at 0x2365b186f98>



In [13]:
```
import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 12, 10

# Define la matriz de entrada con ángulos de 60 a 300 grados convertidos a radianes
x = np.array([i*np.pi/180 for i in range(60,300,4)])
y = np.sin(x) + np.random.normal(0,0.15,len(x))
data = pd.DataFrame(np.column_stack([x,y]),columns=['x','y'])
```

```
plt.plot(data['x'],data['y'],'.')

# Calcular ajustes para diferentes grados
sols = {}
for grado in range(1,7):
  z = np.polyfit(x, y, grado, full=True)
  sols[grado] = z

# Imprimir datos
plt.plot(x, y, 'o')

# Imprimir curvas de ajuste
xp = np.linspace(0, 5.2, 100)
for grado, sol in sols.items():
  coefs, error, *_ = sol
  p = np.poly1d(coefs)
  plt.plot(xp, p(xp), "-", label="Gr: %s. Error %.3f" % (grado, error) )
plt.legend()
```

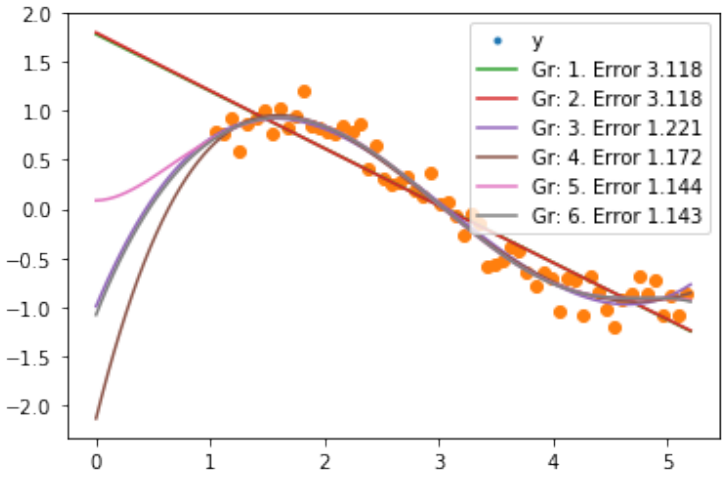
Out[13]: <matplotlib.legend.Legend at 0x2365a2acf60>



In [16]:
```
import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt
%matplotlib inline
```

```
# Define la matriz de entrada con ángulos de 60 a 300 grados convertidos a radianes
x = np.array([i*np.pi/180 for i in range(60,300,4)])
y = np.sin(x) + np.random.normal(0,0.15,len(x))
data = pd.DataFrame(np.column_stack([x,y]),columns=['x','y'])
plt.plot(data['x'],data['y'],'.')

# Calcular ajustes para diferentes grados
sols = {}
for grado in range(1,7):
  z = np.polyfit(x, y, grado, full=True)
  sols[grado] = z

# Imprimir datos
plt.plot(x, y, 'o')

# Imprimir curvas de ajuste
xp = np.linspace(0, 5.2, 100)
for grado, sol in sols.items():
  coefs, error, *_ = sol
  p = np.poly1d(coefs)
  plt.plot(xp, p(xp), "-", label="Gr: %s. Error %.3f" % (grado, error) )
plt.legend()
```

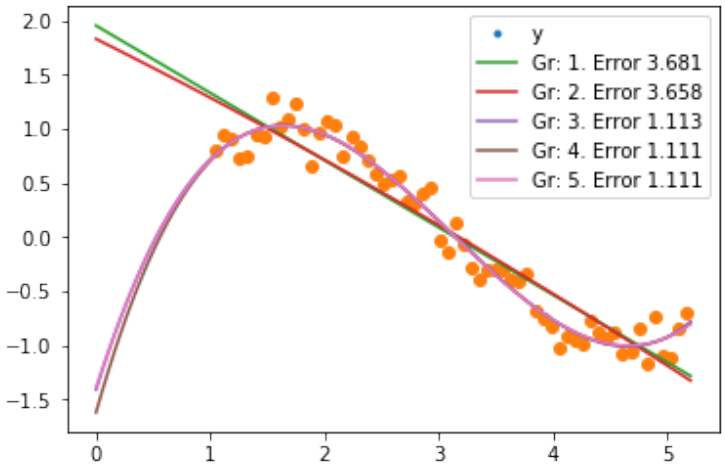
Out[16]: <matplotlib.legend.Legend at 0x2365b269550>



In [18]:
```
import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt
%matplotlib inline
```

```
# Define la matriz de entrada con ángulos de 60 a 300 grados convertidos a radianes
x = np.array([i*np.pi/180 for i in range(60,300,4)])
y = np.sin(x) + np.random.normal(0,0.15,len(x))
data = pd.DataFrame(np.column_stack([x,y]),columns=['x','y'])
plt.plot(data['x'],data['y'],'.')

# Calcular ajustes para diferentes grados
sols = {}
for grado in range(1,6):
  z = np.polyfit(x, y, grado, full=True)
  sols[grado] = z

# Imprimir datos
plt.plot(x, y, 'o')

# Imprimir curvas de ajuste
xp = np.linspace(0, 5.2, 100)
for grado, sol in sols.items():
  coefs, error, *_ = sol
  p = np.poly1d(coefs)
  plt.plot(xp, p(xp), "-", label="Gr: %s. Error %.3f" % (grado, error) )
plt.legend()
```

Out[18]: <matplotlib.legend.Legend at 0x2365b389cf8>



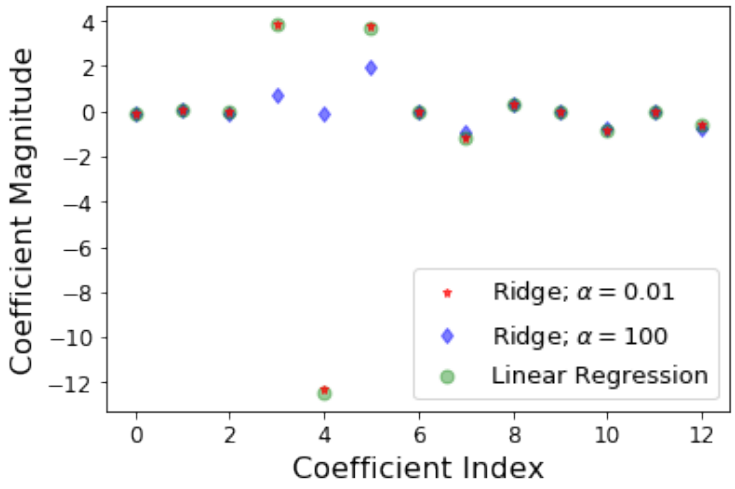
1.3 Regresión Ridge

código utizado obtenido de:

https://towardsdatascience.com/ridge-and-lasso-regression-a-complete-guide-with-python-scikit-learn-e20e34bcbf0b

In [2]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib
matplotlib.rcParams.update({'font.size': 12})
from sklearn.datasets import load_boston
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
boston=load_boston()
boston_df=pd.DataFrame(boston.data,columns=boston.feature_names)
#print boston_df.info()
# add another column that contains the house prices which in scikit learn datasets are c
boston_df['Price']=boston.target
#print boston_df.head(3)
newX=boston_df.drop('Price',axis=1)
print (newX[0:3]) # check
newY=boston_df['Price']
#print type(newY)# pandas core frame
X_train,X_test,y_train,y_test=train_test_split(newX,newY,test_size=0.3,random_state=3)
print (len(X_test), len(y_test))
lr = LinearRegression()
lr.fit(X_train, y_train)
rr = Ridge(alpha=0.01) # higher the alpha value, more restriction on the coefficients; l
# restricted and in this case linear and ridge regression resembles
rr.fit(X_train, y_train)
rr100 = Ridge(alpha=100) #  comparison with alpha value
rr100.fit(X_train, y_train)
train_score=lr.score(X_train, y_train)
test_score=lr.score(X_test, y_test)
Ridge_train_score = rr.score(X_train,y_train)
Ridge_test_score = rr.score(X_test, y_test)
Ridge_train_score100 = rr100.score(X_train,y_train)
Ridge_test_score100 = rr100.score(X_test, y_test)
print ("linear regression train score:", train_score)
print ("linear regression test score:", test_score)
print ("ridge regression train score low alpha:", Ridge_train_score)
print ("ridge regression test score low alpha:", Ridge_test_score)
print ("ridge regression train score high alpha:", Ridge_train_score100)
print ("ridge regression test score high alpha:", Ridge_test_score100)
plt.plot(rr.coef_,alpha=0.7,linestyle='none',marker='*',markersize=5,color='red',label=r
plt.plot(rr100.coef_,alpha=0.5,linestyle='none',marker='d',markersize=6,color='blue',lab
plt.plot(lr.coef_,alpha=0.4,linestyle='none',marker='o',markersize=7,color='green',label
plt.xlabel('Coefficient Index',fontsize=16)
plt.ylabel('Coefficient Magnitude',fontsize=16)
plt.legend(fontsize=13,loc=4)
plt.show()
```

```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0

   PTRATIO       B  LSTAT
0     15.3  396.90   4.98
1     17.8  396.90   9.14
2     17.8  392.83   4.03
152 152
linear regression train score: 0.7419034960343789
linear regression test score: 0.7146895989294312
ridge regression train score low alpha: 0.7419030253527293
ridge regression test score low alpha: 0.7145115044376253
ridge regression train score high alpha: 0.7172809669938278
ridge regression test score high alpha: 0.6805838894730996
```



1.4 Regresión Lasso

código utizado obtenido de:

https://towardsdatascience.com/ridge-and-lasso-regression-a-complete-guide-with-python-scikit-learn-e20e34bcbf0b

```
In [3]: import math
        import matplotlib.pyplot as plt
        import pandas as pd
        import numpy as np
        # difference of lasso and ridge regression is that some of the coefficients can be zero
        # completely neglected
```

```
from sklearn.linear_model import Lasso
from sklearn.linear_model import LinearRegression
from sklearn.datasets import load_breast_cancer
from sklearn.cross_validation import train_test_split
cancer = load_breast_cancer()
#print cancer.keys()
cancer_df = pd.DataFrame(cancer.data, columns=cancer.feature_names)
#print cancer_df.head(3)
X = cancer.data
Y = cancer.target
X_train,X_test,y_train,y_test=train_test_split(X,Y, test_size=0.3, random_state=31)
lasso = Lasso()
lasso.fit(X_train,y_train)
train_score=lasso.score(X_train,y_train)
test_score=lasso.score(X_test,y_test)
coeff_used = np.sum(lasso.coef_!=0)
print ("training score:", train_score)
print ("test score: ", test_score)
print ("number of features used: ", coeff_used)
lasso001 = Lasso(alpha=0.01, max_iter=10e5)
lasso001.fit(X_train,y_train)
train_score001=lasso001.score(X_train,y_train)
test_score001=lasso001.score(X_test,y_test)
coeff_used001 = np.sum(lasso001.coef_!=0)
print ("training score for alpha=0.01:", train_score001)
print ("test score for alpha =0.01: ", test_score001)
print ("number of features used: for alpha =0.01:", coeff_used001)
lasso00001 = Lasso(alpha=0.0001, max_iter=10e5)
lasso00001.fit(X_train,y_train)
train_score00001=lasso00001.score(X_train,y_train)
test_score00001=lasso00001.score(X_test,y_test)
coeff_used00001 = np.sum(lasso00001.coef_!=0)
print ("training score for alpha=0.0001:", train_score00001)
print ("test score for alpha =0.0001: ", test_score00001)
print ("number of features used: for alpha =0.0001:", coeff_used00001)
lr = LinearRegression()
lr.fit(X_train,y_train)
lr_train_score=lr.score(X_train,y_train)
lr_test_score=lr.score(X_test,y_test)
print ("LR training score:", lr_train_score)
print ("LR test score: ", lr_test_score)
plt.subplot(1,2,1)
plt.plot(lasso.coef_,alpha=0.7,linestyle='none',marker='*',markersize=5,color='red',labe
plt.plot(lasso001.coef_,alpha=0.5,linestyle='none',marker='d',markersize=6,color='blue',

plt.xlabel('Coefficient Index',fontsize=16)
plt.ylabel('Coefficient Magnitude',fontsize=16)
plt.legend(fontsize=13,loc=4)
```

```
        plt.subplot(1,2,2)
        plt.plot(lasso.coef_,alpha=0.7,linestyle='none',marker='*',markersize=5,color='red',labe
        plt.plot(lasso001.coef_,alpha=0.5,linestyle='none',marker='d',markersize=6,color='blue',
        plt.plot(lasso00001.coef_,alpha=0.8,linestyle='none',marker='v',markersize=6,color='blac
        plt.plot(lr.coef_,alpha=0.7,linestyle='none',marker='o',markersize=5,color='green',label
        plt.xlabel('Coefficient Index',fontsize=16)
        plt.ylabel('Coefficient Magnitude',fontsize=16)
        plt.legend(fontsize=13,loc=4)
        plt.tight_layout()
        plt.show()
```

```
training score: 0.5600974529893079
test score:  0.5832244618818156
number of features used:  4
training score for alpha=0.01: 0.7037865778498829
test score for alpha =0.01:  0.6641831577726228
number of features used: for alpha =0.01: 10
training score for alpha=0.0001: 0.7754092006936697
test score for alpha =0.0001:  0.7318608210757909
number of features used: for alpha =0.0001: 22
LR training score: 0.7842206194055069
LR test score:  0.7329325010888686
```