PROYECTO 2 - Carlos Barrera

Regresión por Descenso Gradiente con condición de Armijo

Importamos las librerías

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

Leemos el archivo CSV y definimos los vectores X y Y

In [45]:
```
wine=pd.read_csv('winequality-red.csv')
cols = ["fixed_acidity", "alcohol", "chlorides"]
X=wine.as_matrix(columns=cols)
Y=np.matrix(wine.density.as_matrix(),float).T
Ygraf=wine.density.as_matrix()
```

Definimos las funciones

In [47]:
```
def fBeta(X,Y,Beta):
    'Es el desarrollo de la función que queremos minimizar'
    return
np.dot(Beta.T,np.dot(np.dot(X.T,X),Beta))-2*np.dot(Y.T,np.dot(X,Beta))+n
p.dot(Y.T,Y)
```

In [48]:
```
def gradiente(X,Y,Beta):
    'Nos da el gradiente de la función fBeta'
    g=2*np.dot(np.dot(X.T,X),Beta)-2*np.dot(X.T,Y)
    return g
```

In [54]:
```
def beta_alfamas1(X,Y,Beta):
    'Calcula un nuevo vector Beta acercándose al mínimo local'
    alfa=1.
    tau = 1/2.
    g1=fBeta(X,Y,Beta+alfa*(-gradiente(X,Y,Beta)))
    while (fBeta(X,Y,Beta+alfa*(-gradiente(X,Y,Beta))) >
fBeta(X,Y,Beta)+tau*alfa*np.dot(gradiente(X,Y,Beta).T,-gradiente(X,Y,Bet
a))):
        alfa= alfa*tau
    return Beta-alfa*gradiente(X,Y,Beta)
```

In [55]:
```
def tolerancia(g):
    'Define la tolerancia que nos indicará cuando nos hemos acercado lo
 suficiente a la condición de que grad(fBeta)=0'
    not_zero = False
    val_min = 1e-2
    for gi in g:
        if (np.abs(gi) > val_min):
            not_zero = True
    return not_zero
```

In [68]:
```
def buscaminimo(X,Y):
    'Itera generando un vector beta cada vez más cercano al vector que m
inimice la función hasta que se cumple la tolerancia de grad(fBeta)=0'
    Beta=np.matrix(np.zeros(X.shape[1])).T
    while (tolerancia(gradiente(X,Y,Beta))):
        Beta = beta_alfamas1(X,Y,Beta)
    return Beta
```

In [80]:
```
def imprime(X, Y, Beta, columna, vector_unos):
    'Nos permite imprimir un vector Xn contra el vector Y'
    plt.figure()
    mensaje = cols[columna] + " vs density"
    if(vector_unos):
        mensaje += " with ones vector"
        columna += 1
    plt.plot(X[:,columna], Y, 'o')
    x = np.linspace(0,max(X[:,columna]),100) # 100 linearly spaced numbe
rs
    if(vector_unos):
        y= Beta[0,0]+x*Beta[columna,0]
    else:
        y = x*Beta[columna,0]
    plt.plot(x, y, 'o')
    plt.title(mensaje)
    plt.show()
```

Obtenemos el vector β que minimiza la distancia entre $X\beta$ y Y

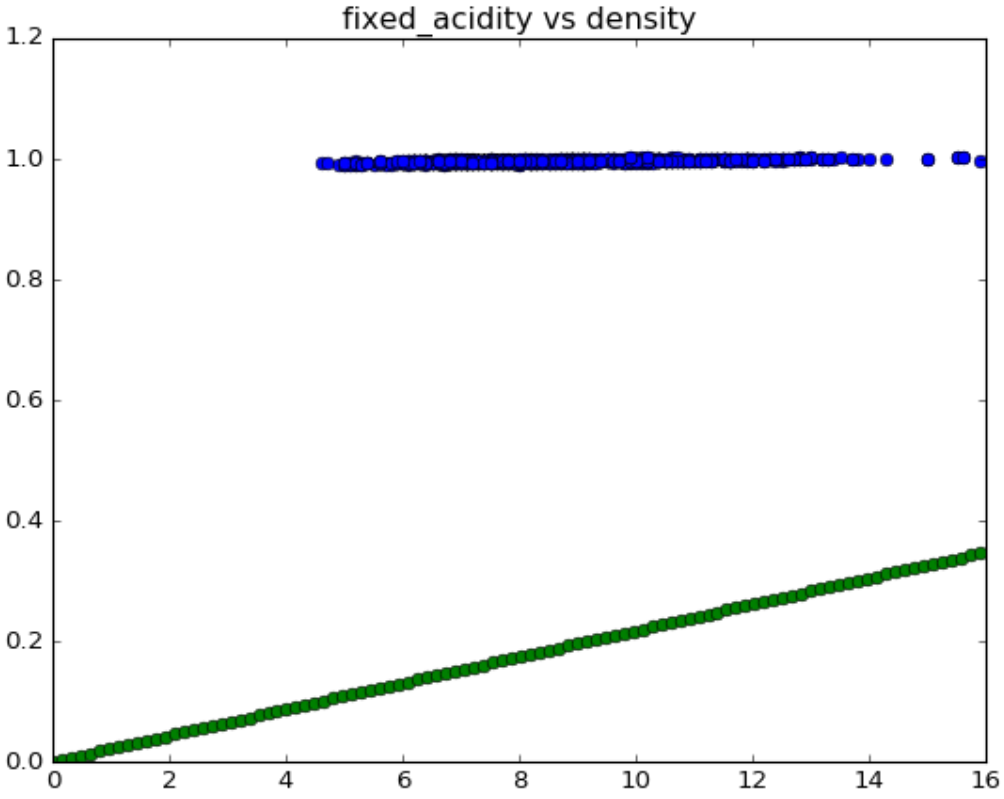
In [59]:
```
myBeta=buscaminimo(X,Y)
```

In [92]:
```
print(myBeta)
```

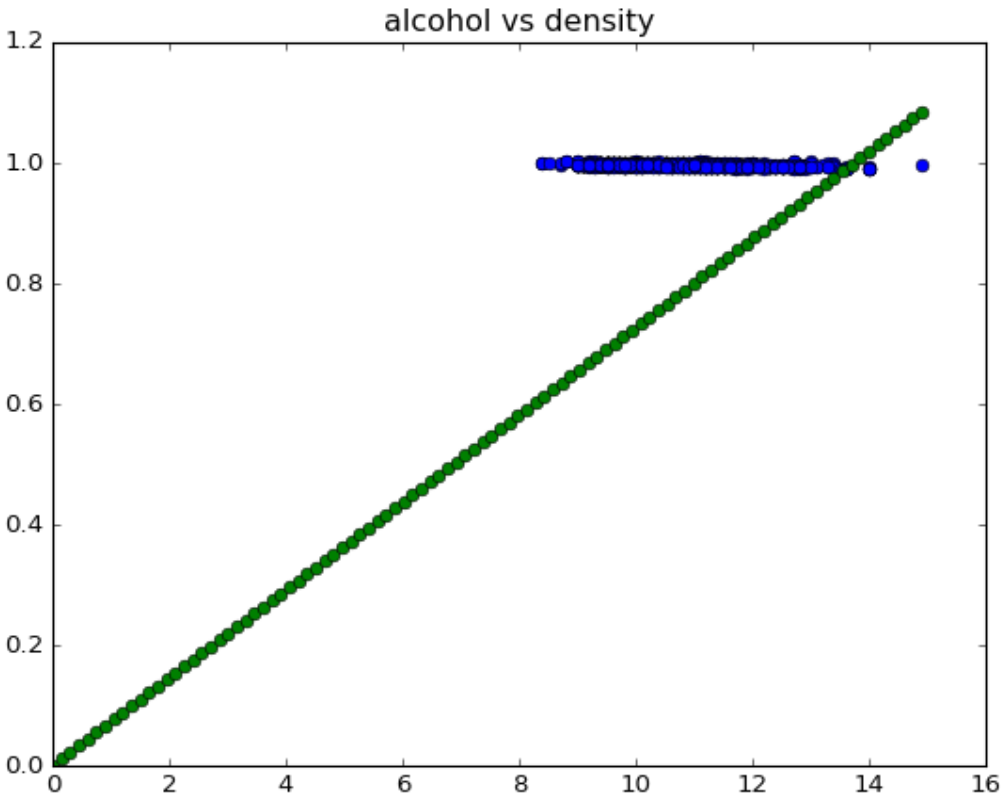
```
[[ 0.02176165]
 [ 0.07281368]
 [ 0.5668443 ]]
```

Imprimimos las gráficas comparando los vectores X_n vs Y añadiendo la linea de tendencia $f(x) = \beta_n x$

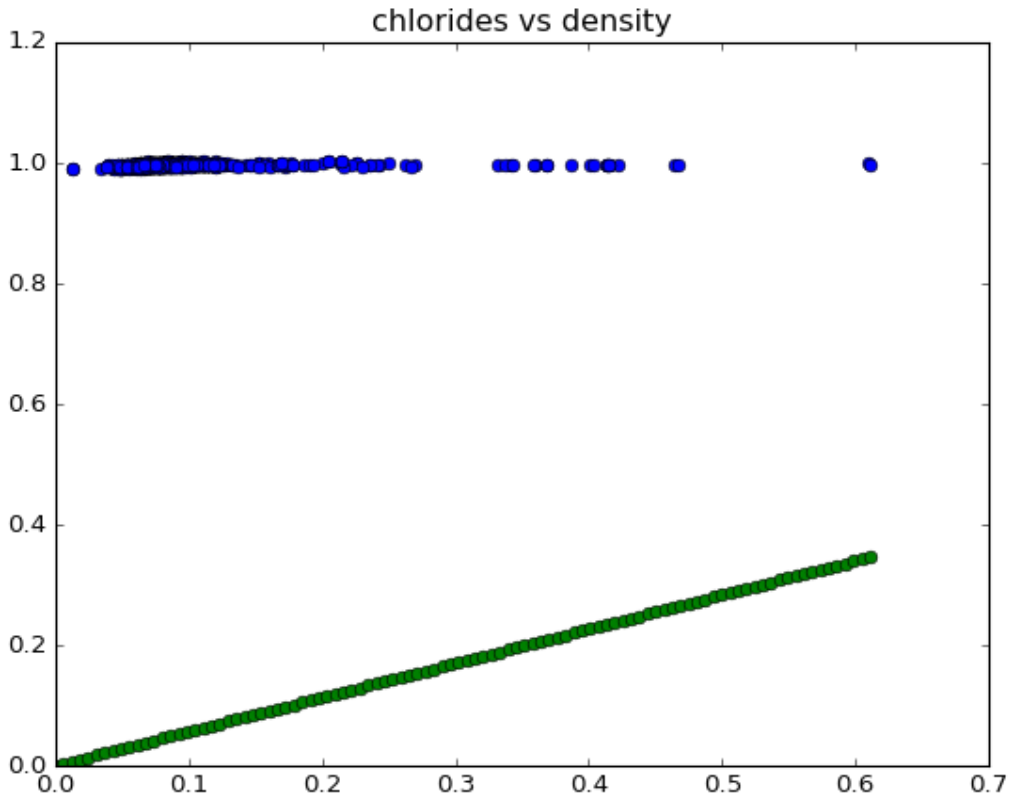
In [82]:
```
imprime(X, Ygraf, myBeta, 0, False)
```



In [83]:
```
imprime(X, Ygraf, myBeta, 1, False)
```



In [84]:
```
imprime(X, Ygraf, myBeta, 2, False)
```



Agregamos una columna de unos a la matriz X y volvemos a calcular β

In [66]:
```
X2 = np.c_[np.ones(X.shape[0]), X]
```

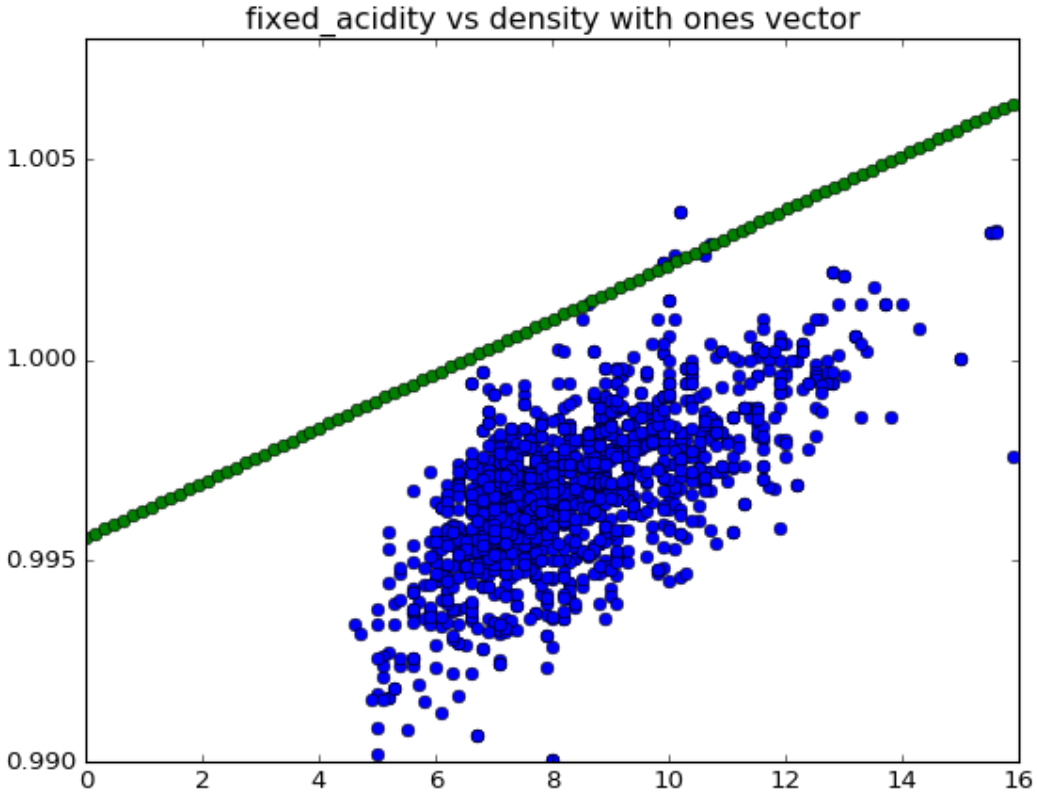
In [90]:
```
myBeta2=buscaminimo(X2,Y)
```

In [91]:
```
print(myBeta2)
```

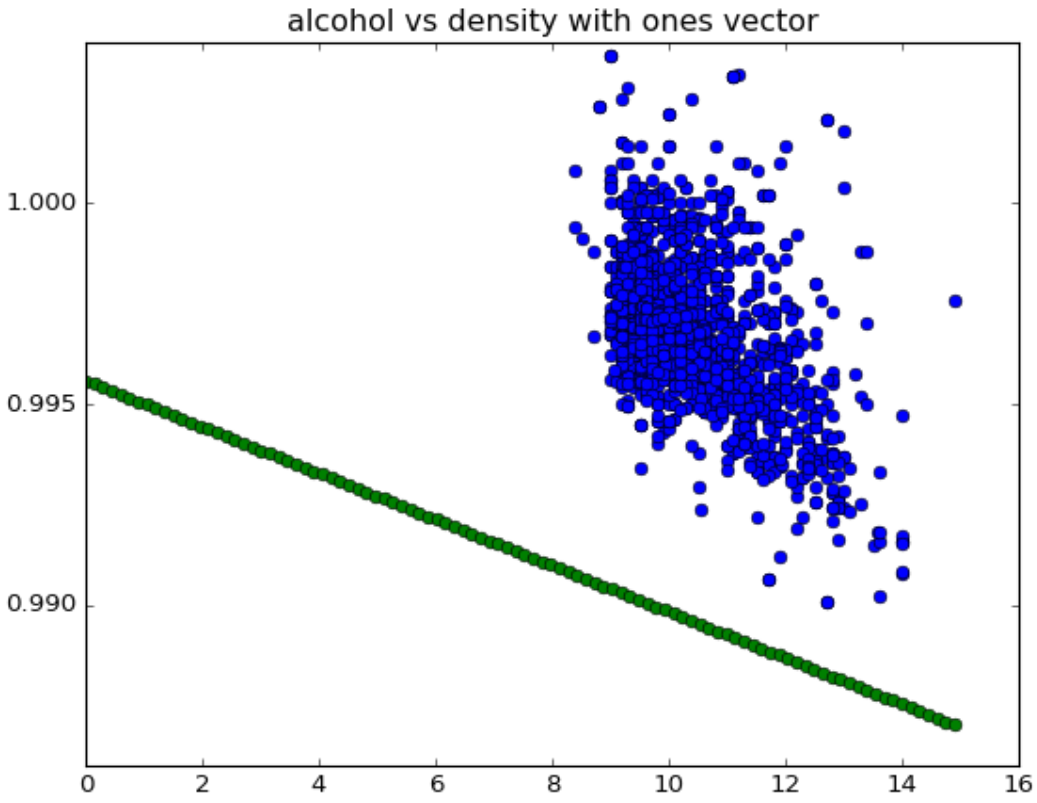
```
[[  9.95582566e-01]
 [  6.78822004e-04]
 [ -5.75192383e-04]
 [  1.72059991e-02]]
```

Imprimimos las nuevas gráficas X_n vs Y añadiendo la linea de tendencia $f(x) = \beta_0 + \beta_n x$

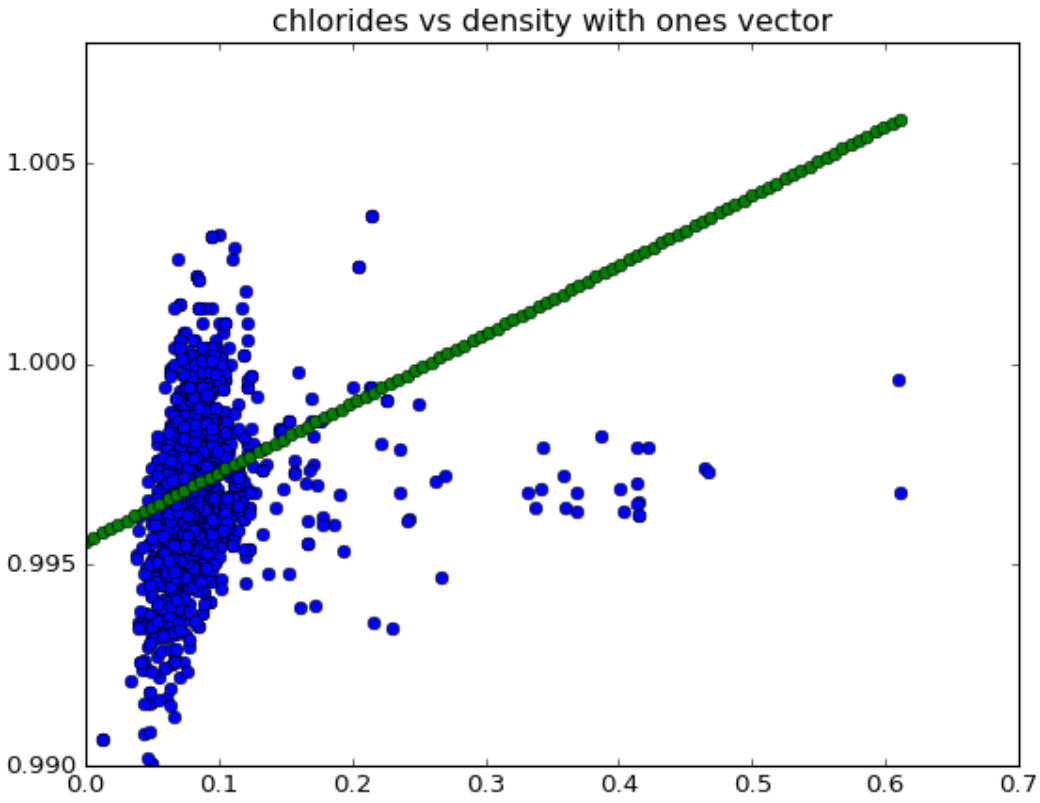
In [93]:
```
imprime(X2, Ygraf, myBeta2, 0, True)
```



In [94]:
```
imprime(X2, Ygraf, myBeta2, 1, True)
```



In [95]:
```
imprime(X2, Ygraf, myBeta2, 2, True)
```



In [ ]:
```

```

Regresión por Mínimos Cuadrados (SVD)

Importamos las librerías

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

Leemos el archivo CSV y definimos los vectores X y Y

In [3]:
```
wine=pd.read_csv('winequality-red.csv')
cols = ["fixed_acidity", "alcohol", "chlorides"]
X=wine.as_matrix(columns=cols)
Y=np.matrix(wine.density.as_matrix(),float).T
Ygraf=wine.density.as_matrix()
```

Definimos las funciones

In [110]:
```
def seudoinversa(X):
    'Calcula la seudoinversa de la matriz X'
    U, S, VT = np.linalg.svd(np.dot(X.T,X))
    SeudoS = sigmainversa(S, VT.shape[0])
    return np.dot(VT.T, np.dot(SeudoS, U.T))
```

In [5]:
```
def sigmainversa (s, m):
    'Regresa la matriz inversa de sigma truncando a cero valores demasia
do pequeños'
    S = np.zeros([m, m])
    for idx, val in enumerate(s):
        if (val > 1e-12 and idx < m):
            S[idx][idx] = 1.0/val
    return S
```

In [6]:
```
def betafinal(X,Y):
    'Regresa el vector Beta obtenido de la multiplicación de (XT*X)^-1*X
T*Y'
    return np.dot(seudoinversa(X), np.dot(X.T,Y))
```

In [ ]:
```
def imprime(X, Y, Beta, columna, vector_unos):
    'Nos permite imprimir un vector Xn contra el vector Y'
    plt.figure()
    mensaje = cols[columna] + " vs density"
    if(vector_unos):
        mensaje += " with ones vector"
        columna += 1
    plt.plot(X[:,columna], Y, 'o')
    x = np.linspace(0,max(X[:,columna]),100) # 100 linearly spaced numbe
rs
    if(vector_unos):
        y= Beta[0,0]+x*Beta[columna,0]
    else:
        y = x*Beta[columna,0]
    plt.plot(x, y, 'o')
    plt.title(mensaje)
    plt.show()
```

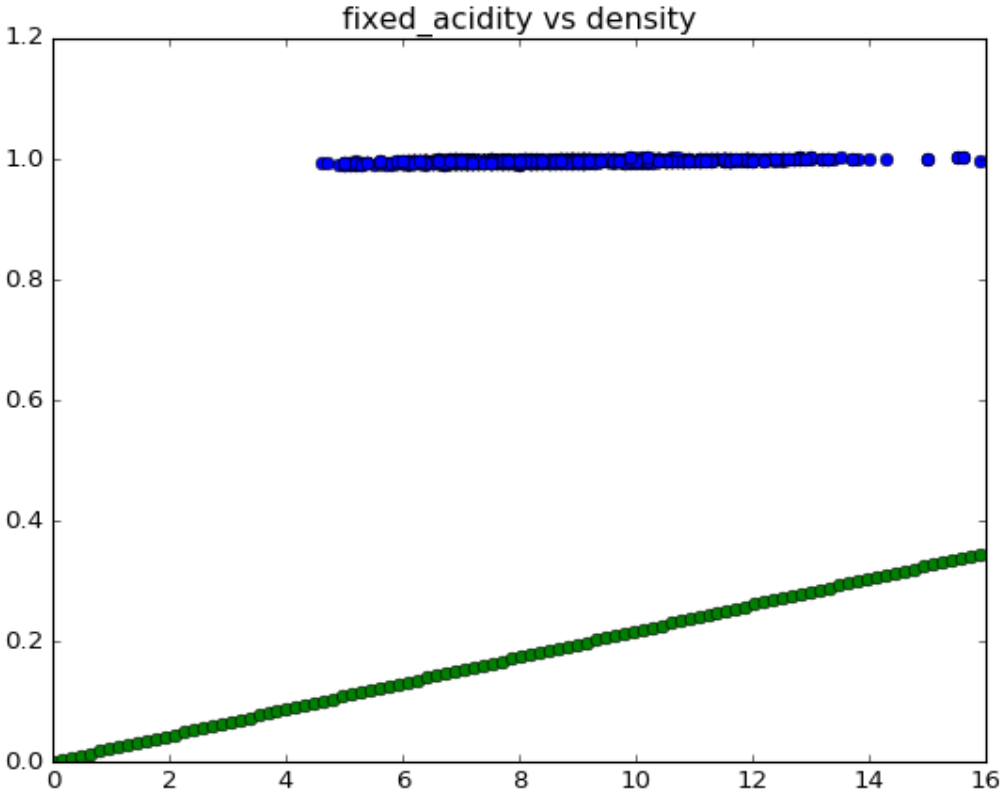
Obtenemos el vector β que minimiza la distancia entre $X\beta$ y Y

In [111]:
```
myBeta = betafinal(X,Y)
print (myBeta)
```

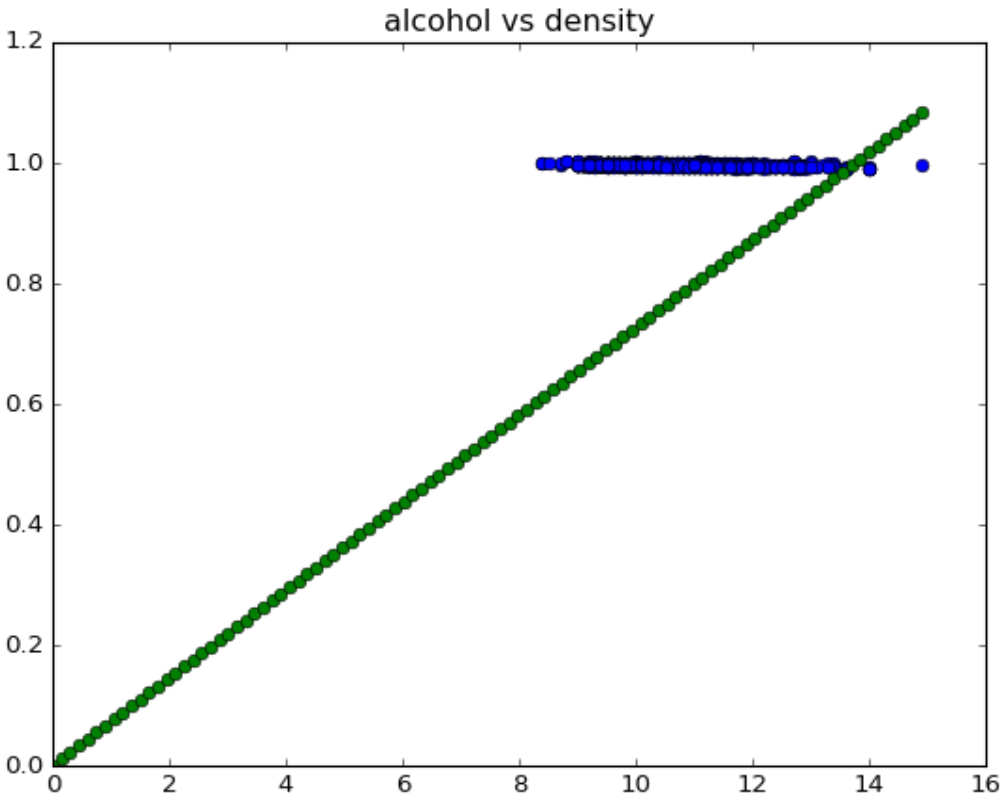
```
[[ 0.02167594]
 [ 0.07277042]
 [ 0.58035115]]
```

Imprimimos las gráficas comparando los vectores X_n vs Y añadiendo la linea de tendencia $f(x) = \beta_n x$

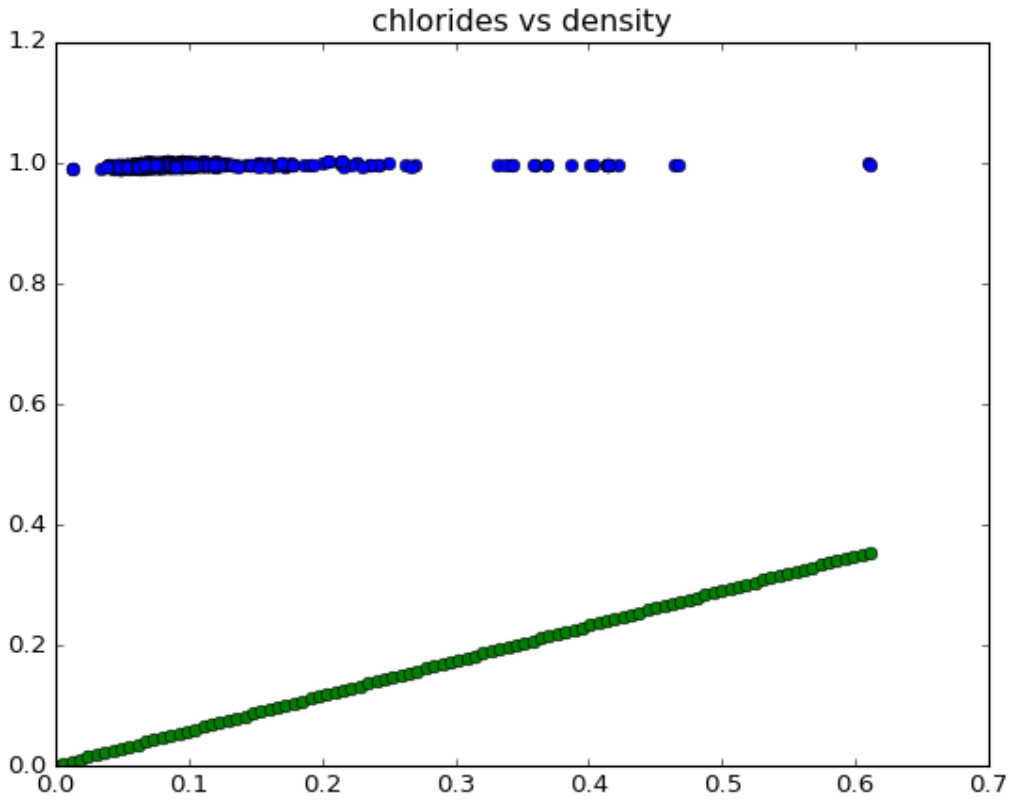
In [63]:
```
imprime(X, Ygraf, myBeta, 0, False)
```



In [64]:
```
imprime(X, Ygraf, myBeta, 1, False)
```



In [65]:
```
imprime(X, Ygraf, myBeta, 2, False)
```



Agregamos una columna de unos a la matriz X y volvemos a calcular β

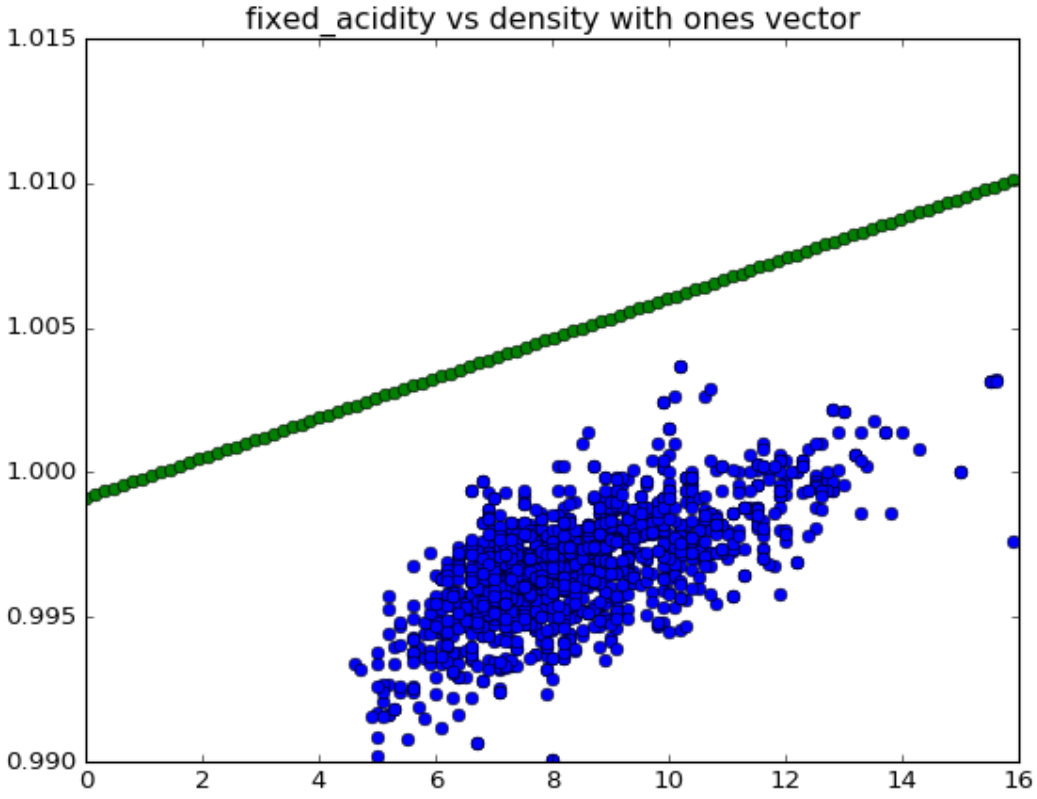
In [38]:
```
X2 = np.c_[np.ones(X.shape[0]), X]
```

In [113]:
```
myBeta2 = betafinal(X2,Y)
print (myBeta2)
```

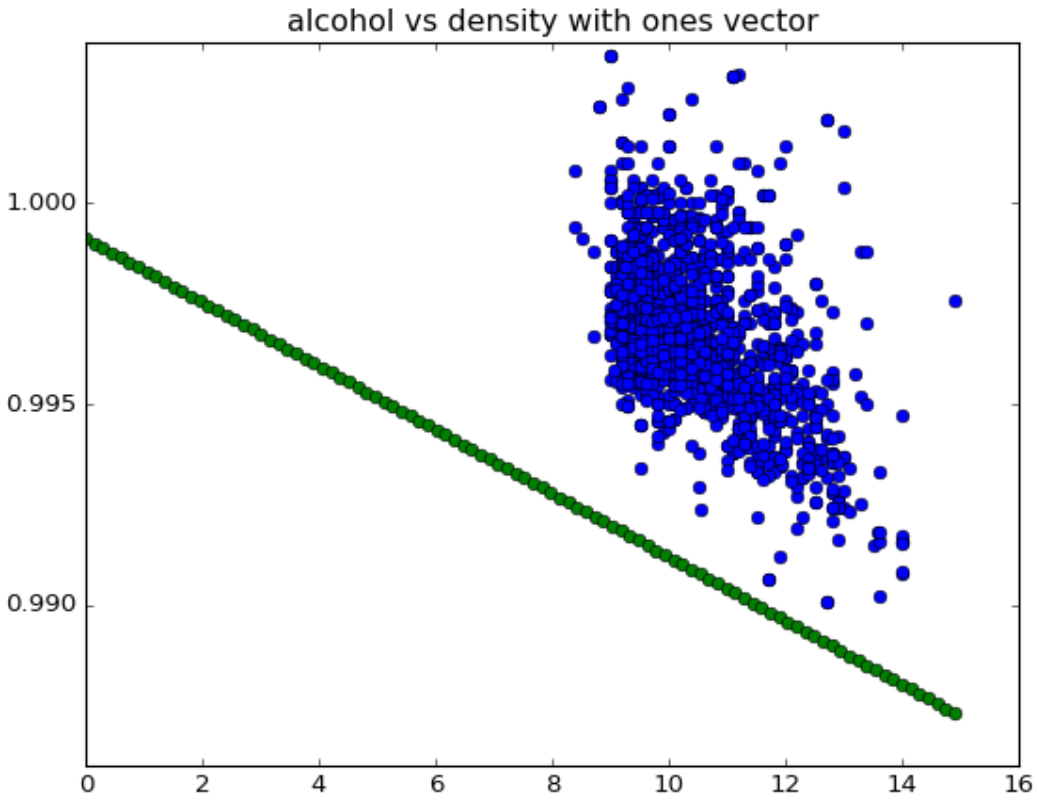
```
[[  9.99122292e-01]
 [  6.89968044e-04]
 [ -7.92786330e-04]
 [  1.68409049e-03]]
```

Imprimimos las nuevas gráficas X_n vs Y añadiendo la linea de tendencia $f(x) = \beta_0 + \beta_n x$

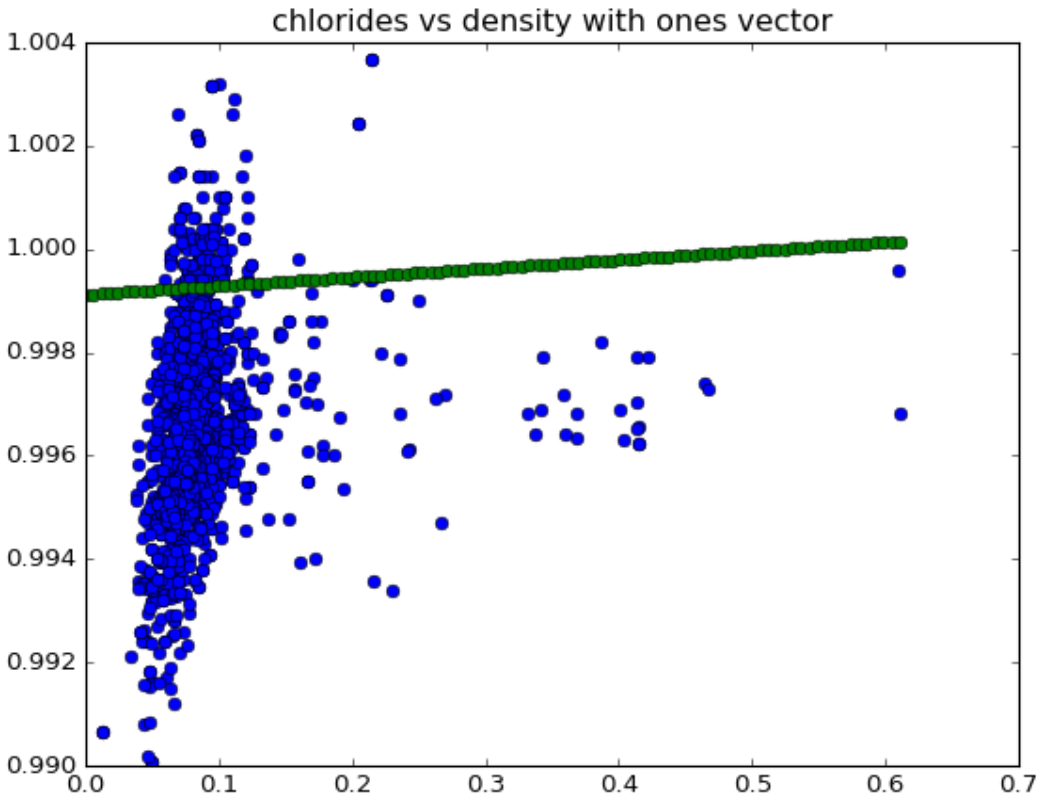
In [66]:
```
imprime(X2, Ygraf, myBeta2, 0, True)
```



In [67]:
```
imprime(X2, Ygraf, myBeta2, 1, True)
```



In [68]:
```
imprime(X2, Ygraf, myBeta2, 2, True)
```



Análisis del error de aproximación

Si $X\beta \neq Y$, entonces el modelo de regresión tendrá un error de aproximación pues Y se encuentra fuera del espacio columna de X.

Una vez realizada la regresión obtenemos que $Y_i = (\sum_{j=1}^{m} x_{ij}\beta_i) + \epsilon_i$ donde ϵ_i es el error de cada observación, por lo tanto, podemos encontrar el vector de errores ϵ restando el vector aproximado $X\beta$ al vector Y es decir $\epsilon = Y - X\beta$

In [69]:
```
Yaprox = np.dot(X, myBeta)
```

In [91]:
```
epsilon = Y - Yaprox
```

Para calcular la norma cuadrada de $Y - X\beta$ (Que es el valor que queríamos minimizar) solamente tenemos que calcular el producto punto $\langle \epsilon, \epsilon \rangle$

In [96]:
```
norma_cuadrada = np.dot(epsilon.T, epsilon)
print(norma_cuadrada)
```

```
[[ 11.44796078]]
```

Para comprobar que el vector ϵ es ortogonal al vector $X\beta$, se debe comprobar que $\langle \epsilon, X\beta \rangle = 0$

In [97]:
```
prueba = np.dot(epsilon.T, Yaprox)
print(prueba)
```

```
[[ -5.86730664e-12]]
```

Como podemos ver $\langle \epsilon, X\beta \rangle \cong 0 \therefore \epsilon \perp X\beta$.

Esto es importante ya que si consideramos ahora el hecho de que el error de proyección es ortogonal a la proyección de un vector sobre un hiperplano, se observa que $X\beta = proy_{span(X)} Y$ lo cual tiene sentido pues la proyección de un vector minimiza la distancia entre él y el hiperplano sobre el cual se proyecta.

Efecto de agregar la columna de unos

Como podemos observar en las gráficas, al hacer la regresión sin contar con la columna de unos, obliga a la linea de tendencia a pasar por el origen, lo cual genera una diferencia considerable entre dicha línea y las observaciones.

Al añadir esta columna, estamos representando un valor constante, mismo que al resolver la regresión lineal, nos da el *intercepto de regresión* el cual indica la media de la variable Y cuando X es cero.

Como podemos observar en las gráficas, al no forzar que la linea de tendencia pase por el origen, dicha línea describe de una manera mucho más precisa la relación enre los vectores X_n y Y

In [ ]:
```

```