```
!pip3 install scikit-learn --user --upgrade
```

```
Requirement already satisfied: scikit-learn in /root/.local/lib/python3.7/site-packag
Requirement already satisfied: numpy>=1.14.6 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: threadpoolctl>=2.0.0 in /root/.local/lib/python3.7/sit
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.7/dist-packages
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split,  KFold, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression, Ridge, RidgeCV, Lasso, LassoCV

from scipy import stats


Slump = pd.read_csv('slump_test.data')
Slump.head()
```

	No	Cement	Slag	Fly ash	Water	SP	Coarse Aggr.	Fine Aggr.	SLUMP(cm)	FLOW(cm)	Com
0	1	273.0	82.0	105.0	210.0	9.0	904.0	680.0	23.0	62.0	
1	2	163.0	149.0	191.0	180.0	12.0	843.0	746.0	0.0	20.0	
2	3	162.0	148.0	191.0	179.0	16.0	840.0	743.0	1.0	20.0	
3	4	162.0	148.0	190.0	179.0	19.0	838.0	741.0	3.0	21.5	
4	5	154.0	112.0	144.0	220.0	10.0	923.0	658.0	20.0	64.0	

```
Slump.describe()
```

	No	Cement	Slag	Fly ash	Water	SP	Coa Ag
count	103.000000	103.000000	103.000000	103.000000	103.000000	103.000000	103.000
mean	52.000000	229.894175	77.973786	149.014563	197.167961	8.539806	883.978
std	29.877528	78.877230	60.461363	85.418080	20.208158	2.807530	88.391
min	1.000000	137.000000	0.000000	0.000000	160.000000	4.400000	708.000
25%	26.500000	152.000000	0.050000	115.500000	180.000000	6.000000	819.500
50%	52.000000	248.000000	100.000000	164.000000	196.000000	8.000000	879.000

Apartado a)

Quitamos las dos variables de salida que no usaremos

```
Slump = Slump.drop(['SLUMP(cm)','FLOW(cm)'], axis = 1)
Slump.shape
```

```
(103, 9)
```

Una vez tenemos una unica variable de salida separamos las variables de salida y de entrada.

```
X = Slump.loc[:, Slump.columns != 'Compressive Strength (28-day)(Mpa)']
Y = Slump['Compressive Strength (28-day)(Mpa)']
```

Dividimos los datos en dos conjuntos, un 60% de entrenamiento y un 40% de test

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.4, random_state=42)
```

Escalamos los datos de entrenamiento y los datos de test con el objetivo de estandarizarlos, ya que nuestros procedimientos se van a basar en que los datos se asemejan a una distribución normal.

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X_test)
```

Apartado b)

En primer lugar inicializamos la regresión lineal. Entrenamos la regresión lineal con los datos de entrenamiento utilizando el método *fit*. Dado que no queremos usar nuestros datos de entrenamiento porque podria producirse overfitting, y tampoco queremos usar nuestros datos de test por que los vamos a necesitar al final para validar nuestro modelo, usamos *Cross Validation* para comprobar como de bueno es nuestro modelo.

Tras hacer la media de los resultados de la *Cross Validation* obtenemos una R2 de 0.7229.

```
lr = LinearRegression()
lr.fit(X_train_scaled,Y_train)
folds_r2 = cross_val_score(lr, X_train_scaled,Y_train, cv=5, scoring='r2')
lr_r2 = np.mean(folds_r2)
folds_r2, lr_r2
```

```
(array([0.58401637, 0.64284091, 0.81470389, 0.87579358, 0.6971585 ]),
 0.7229026501401414)
```

Ahora usamos los datos de test para computar el R2 score y obtenemos un R2 de 0.9017

```
r2 = lr.score(X_test_scaled,Y_test)

print('R2 score with test data: {}'.format(r2))
```

```
R2 score with test data: 0.9017024036224461
```

Apartado c)

Hemos obtenido metricas bastante buenas con la regresión linear, ahora vamos a ajustar las regresiones Ridge y LASSO para ver si obtenemos mejores métricas.

Ridge

Tras ajustar una Ridge regresion y validarla con cross variation, obtenemos una R2 de 0,7486, la cual es un poco mejor que la obtenida con regresión lineal

```
lambdas = [1e-10,1e-5,1e-4,1e-3,1e-2,0.1, 0.5,1,5,10,50,100]
ridge_cv =RidgeCV(alphas=lambdas,cv=5)
ridge_cv.fit(X_train_scaled,Y_train)

print('Best lambda:', ridge_cv.alpha_, 'R2 score:', ridge_cv.best_score_)
```

```
Best lambda: 1.0 R2 score: 0.748682166266127
```

Tras ajustar nuestra Ridge regresión la usamos para obtener el R2 de los datos de test. Y obtenemos un R2 de 0,9025. Un valor ligeramente más alto que el obtenido con la regresión lineal.

```
r2 = ridge_cv.score(X_test_scaled,Y_test)

print('R2 score with test data: {}'.format(r2))
```

```
R2 score with test data: 0.9025280603882707
```

LASSO

Repetimos el proceso con la regresión LASSO, y tras validarlocon la cross validation, obtenemos un R2 de 0,7450.

```
lasso_cv =LassoCV(alphas=lambdas,cv=5, max_iter=10000)
lasso_cv.fit(X_train_scaled,Y_train)

lasso_r2 =  np.mean(cross_val_score(lasso_cv, X_train_scaled,Y_train))

print('Best lambda:', lasso_cv.alpha_, 'R2 score:',lasso_r2)
```

```
Best lambda: 0.1 R2 score: 0.7449988240516696
```

Ahora usamos la regresión LASSO para calcular el R2 obtenido con los datos de test y nos queda un R2 de 0,9053. El más alto obtenido en todas las regresiones.

```
r2 = lasso_cv.score(X_test_scaled,Y_test)

print('R2 score with test data: {}'.format(r2))
```

```
R2 score with test data: 0.9052695202306292
```

Obtenemos unos resultados ligeramente mejores en Lasso

Apartado d)

Volvemos a ajustar las regresiones tratadas en el apartado anterior pero esta vez pasando el parametro positive como *True* para forzar que todos los pesos sean positivos. Tras volver a

calcular las métricas obtenemos los siguientes resultados para los datos de entrenamiento y los de test.

- Regresión lineal: 0.7442 , 0.8686
- Ridge: 0.8414 , 0.8731
- LASSO: 0.7442 , 0.8686

Vemos que los mejores resultados se obtienen con la Ridge Regresión

Regresión Lineal

```
lr_pos = LinearRegression(positive=True)
lr_pos.fit(X_train_scaled,Y_train)
folds_r2 = cross_val_score(lr_pos, X_train_scaled,Y_train, cv=5, scoring='r2')
lr_r2 = np.mean(folds_r2)
lr_r2
```

```
0.7442173617531336
```

```
r2 = lr_pos.score(X_test_scaled,Y_test)

print('R2 score with test data: {}'.format(r2))
```

```
R2 score with test data: 0.8686195510553445
```

LASSO

```
lasso_cv_pos =LassoCV(alphas=lambdas,cv=5, max_iter=10000, positive=True)
lasso_cv_pos.fit(X_train_scaled,Y_train)
lasso_r2 =  np.mean(cross_val_score(lasso_cv_pos, X_train_scaled,Y_train))

print('Best lambda:', lasso_cv_pos.alpha_, 'R2 score:',lasso_r2)
```

```
Best lambda: 1e-10 R2 score: 0.7442173617547451
```

```
r2 = lasso_cv_pos.score(X_test_scaled,Y_test)

print('R2 score with test data: {}'.format(r2))
```

```
R2 score with test data: 0.868619551058308
```

Ridge

```
#ridge_pos =Ridge(alpha = 100, positive=True)
ridge_pos =Ridge(alpha = ridge_cv.alpha_, positive=True)
ridge_pos.fit(X_train_scaled,Y_train)

print('Best lambda:',ridge_cv.alpha_, 'R2 score:', ridge_pos.score(X_train_scaled,Y_train)
```

```
Best lambda: 1.0 R2 score: 0.8414967141332247
```

```
r2 = ridge_pos.score(X_test_scaled,Y_test)

print('R2 score with test data: {}'.format(r2))
```

```
R2 score with test data: 0.8730990901932585
```

QQ-plot

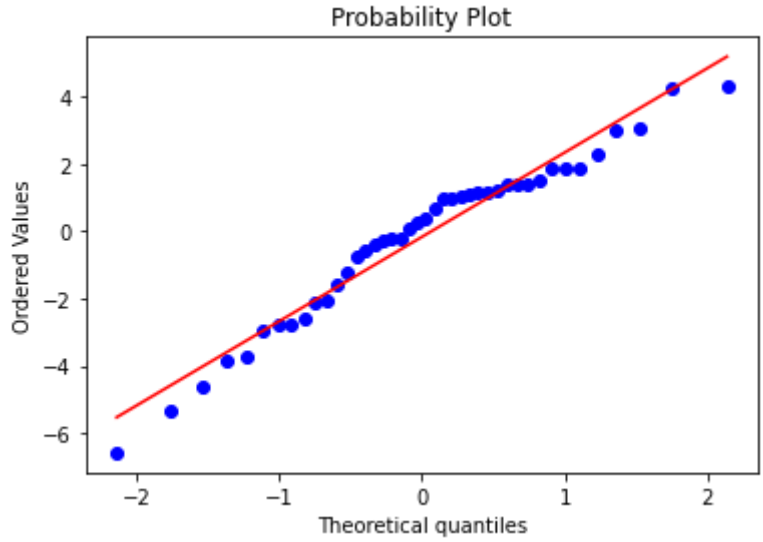
Para cada uno de nuestros modelos calculamos sus residuos y realizamos las *QQ-plot*. Aunque ninguna de las gráficas se aleja demasiado de una distribución Gaussiana, observamos que las gráficas de los modelos con pesos positivos se aproximan mucho más a una distribucion normal.

```
def calcResiduals(modelo,x,y):
  prediccion = modelo.predict(x)
  return y - prediccion
```

```
residuosLasso = calcResiduals(lasso_cv,X_test_scaled,Y_test)
residuosRidge = calcResiduals(ridge_cv,X_test_scaled,Y_test)
residuos = calcResiduals(lr,X_test_scaled,Y_test)
residuosLasso_pos = calcResiduals(lasso_cv_pos,X_test_scaled,Y_test)
residuosRidge_pos = calcResiduals(ridge_pos,X_test_scaled,Y_test)
residuos_pos = calcResiduals(lr_pos,X_test_scaled,Y_test)
```

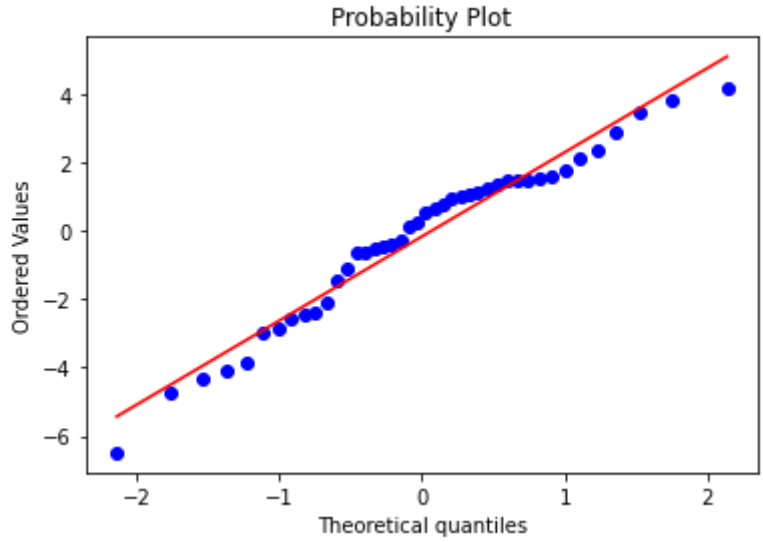
Residuos Regresión Lineal

```
stats.probplot(residuos, dist="norm", plot=plt);
```



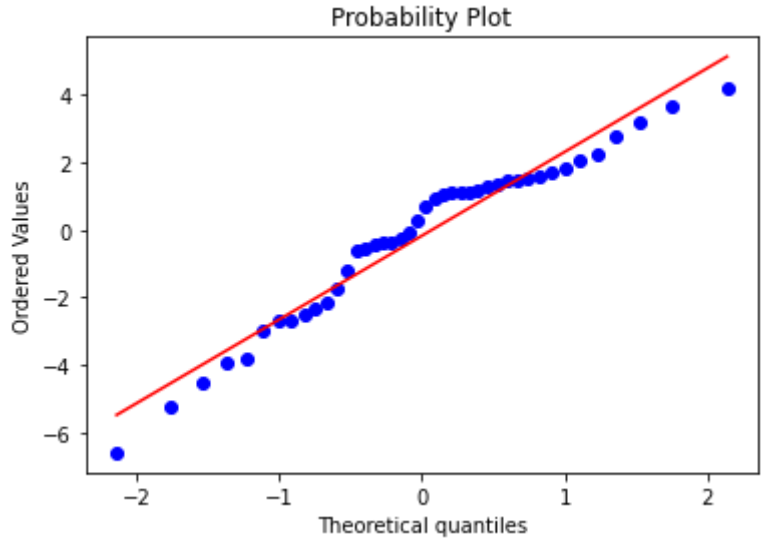
Residuos Lasso

```
stats.probplot(residuosLasso, dist="norm", plot=plt);
```



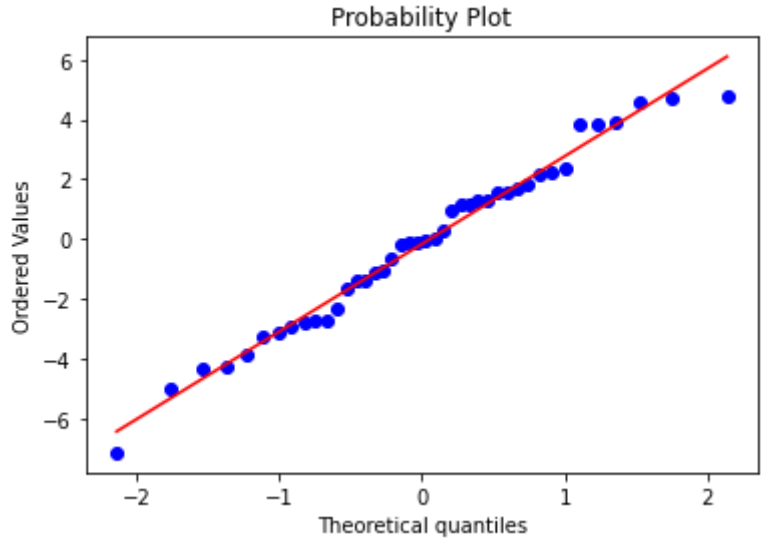
Residuos Ridge

```
stats.probplot(residuosRidge, dist="norm", plot=plt);
```



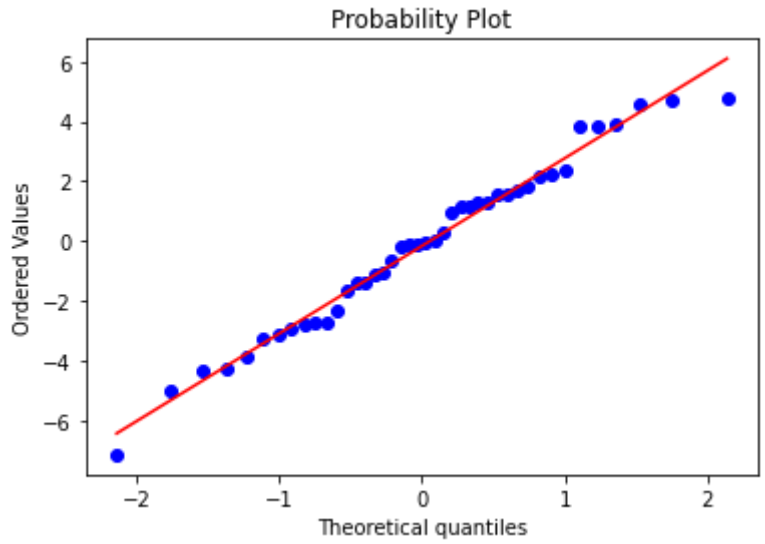
Residuos Regresión Lineal (pesos positivos)

```
stats.probplot(residuos_pos, dist="norm", plot=plt);
```



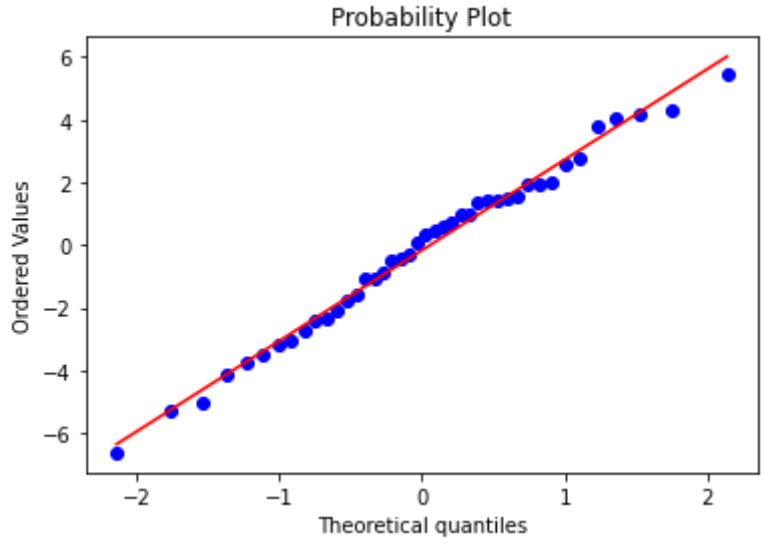
Residuos Lasso (pesos positivos)

```
stats.probplot(residuosLasso_pos, dist="norm", plot=plt);
```



Residuos Ridge (pesos positivos)

```
stats.probplot(residuosRidge_pos, dist="norm", plot=plt);
```



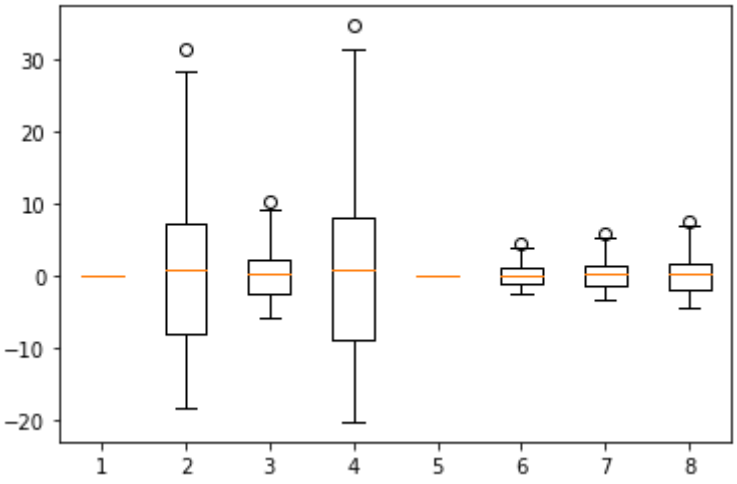
Apartado e)

Hemos elegido el modelo que usa Ridge con los pesos positivos ya que, como la diferencia en R2 entre todos los modelos no era lo suficientemente significativa, hemos escogido, de los que más se ajustaban a una distribución normal, el que tenía mayor R2.

```
weights = ridge_pos.coef_

result = weights * X_train_scaled.reshape(-1,1)

plt.boxplot(result);
```



Hemos computado los efectos utilizando los datos escalados y así poder ver que la diferencia entre los atributos no sea tan grande. Computamos el primer ejemplo del conjunto de test y observamos que para la gran mayoría de atributos se acerca bastante a la media a excepción de *Slag*.

```
primero = X_test_scaled[0]
res = weights * primero
res
```

```
array([-0.        , 11.64313056, -3.76019523,  0.62991925, -0.        ,
       -1.57827151, -0.25454443,  1.0809621 ])
```