MODELOS Y SIMULACION

En esta clase vamos a ver Bias - Variance Trade Off y exploraremos las tecnicas de Cross Validation

In [438]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
```

In [439]:
```
#Traemos el dataset de Boston y armaremos nuestro modelo con todas las variables
boston = pd.read_csv('boston.csv')
boston.head()
```

Out[439]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

In [440]:
```
X = boston['LSTAT']
y = boston['MEDV']
X
```

Out[440]:
```
0      4.98
1      9.14
2      4.03
3      2.94
4      5.33
       ...
501    9.67
502    9.08
503    5.64
504    6.48
505    7.88
Name: LSTAT, Length: 506, dtype: float64
```

In [ ]:

In [441]:
```
# Forma 1 de hacerlo
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Forma 2 de hacerlo
'''
df = boston[['LSTAT', 'MEDV']]
train = df.sample(frac=0.8,random_state=200) #random state is a seed value
test = df.drop(train.index)
X_train = train['LSTAT']
y_train = train['MEDV']
X_test = test['LSTAT']
y_test = test['MEDV']
'''
```

Out[441]:
```
"\ndf = boston[['LSTAT', 'MEDV']]\ntrain = df.sample(frac=0.8,random_state=200) #random state is a seed value\n
test = df.drop(train.index)\nX_train = train['LSTAT']\ny_train = train['MEDV']\nX_test = test['LSTAT']\ny_test
= test['MEDV']\n"
```

In [442]:
```
print('Size of dataset:',len(boston),'| Size of Train:',X_train.size,'| Size of
Test:',X_test.size)
```

```
Size of dataset: 506 | Size of Train: 404 | Size of Test: 102
```

In [443]:
```
#print('Size of dataset:',len(boston),'| Size of Train:',len(train),'| Size of Test:',len(test))
```

In [444]:
```
print("Train: ", X_train,"\nTest: ", X_test)
```

```
Train:  477    24.91
15      8.47
332     7.83
423    23.29
19     11.28
       ...
106    18.66
270    13.00
348     5.99
435    23.27
102    10.63
Name: LSTAT, Length: 404, dtype: float64
Test:  173     9.04
274     3.53
491    18.07
72      5.52
452    17.27
       ...
412    34.37
436    18.05
411    21.22
86     12.86
75      8.94
Name: LSTAT, Length: 102, dtype: float64
```

Bias - Variance Trade Off

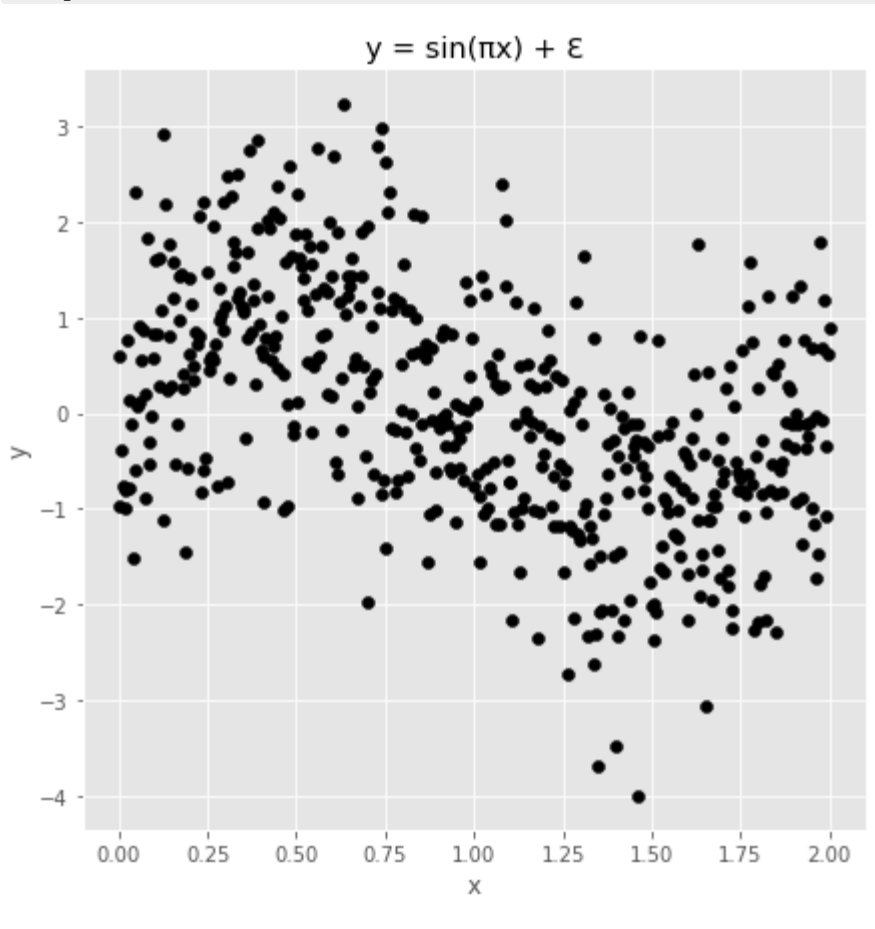
Ajustaremos nuestro modelo hasta que el error del modelo entrenado con train tienda a 0 y analizaremos el error que retornan las predicciones

Vamos a usar un ejemplo: https://towardsdatascience.com/bias-variance-trade-off-with-python-example-6519d2084be4 Ya que es nuestras librerias hacen malas predicciones

In [445]:
```
import numpy as np
import matplotlib.pyplot as plt
from random import randint
from math import pi as PI
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

plt.style.use('ggplot')
```

In [446]:
```
# number of observations
NUM_OBS = 500

# predictors
x = np.linspace(0, 2, num = NUM_OBS)
# noise
eps = np.random.normal(0, 1, NUM_OBS)
# outcome
y = np.sin(PI*x) + eps

fig = plt.figure(figsize=(7,7))
ax = plt.axes()
ax.set_title('y = sin(πx) + Ɛ')
ax.set_xlabel("x")
ax.set_ylabel("y")

ax.scatter(x, y, c = 'k')
```

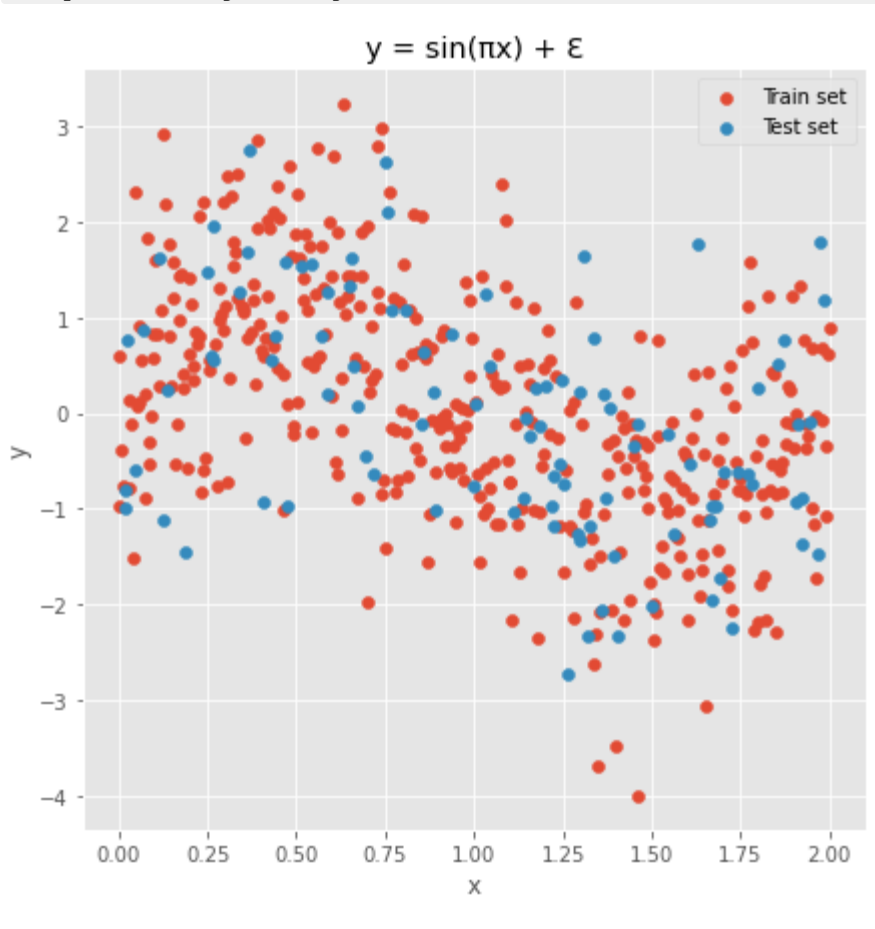
Out[446]:
```
<matplotlib.collections.PathCollection at 0x7fec015acc10>
```



In [447]:
```
# armamos los dataset de training y test en proporcion 80-20
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 1)
```

In [448]:
```
# grafiquemos la dispersion de puntos para cada set
fig = plt.figure(figsize = (7,7))
ax = plt.axes()
ax.set_title('y = sin(πx) + Ɛ')
ax.set_xlabel("x")
ax.set_ylabel("y")

ax.scatter(x_train, y_train, label = "Train set")
ax.scatter(x_test, y_test, label = "Test set")
ax.legend()
```

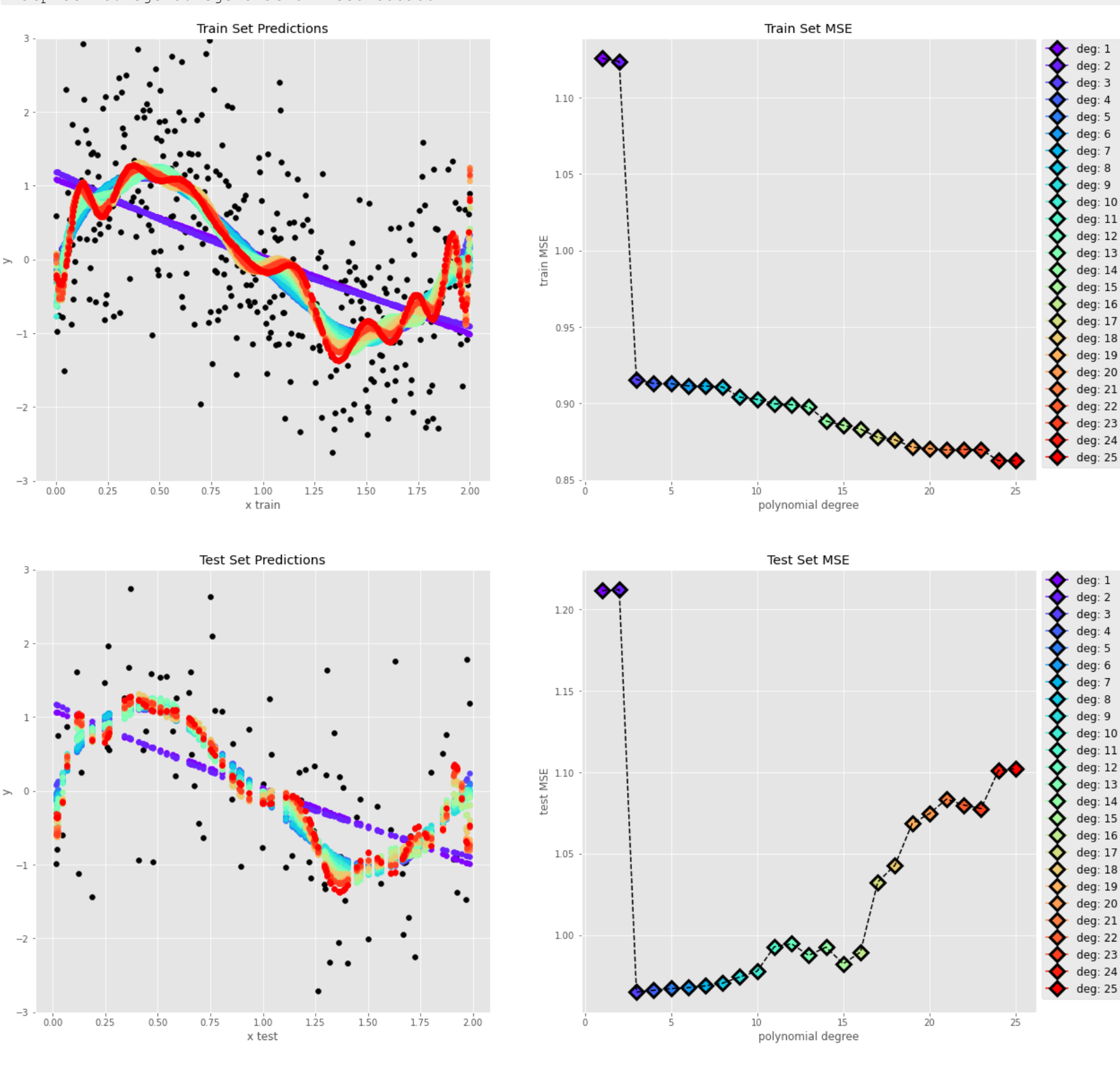
Out[448]:
```
<matplotlib.legend.Legend at 0x7fec0184e730>
```



In [449]:
```
# number of polynomial models to investigate
NUM_MODELS = 25
# seteamos los colores para cada grado de polinomio
colors = iter(plt.cm.rainbow(np.linspace(0, 1, NUM_MODELS)))
# creamos los graficos
fig, axs = plt.subplots(2, 2, figsize = (20,20))
# titulos
axs[0, 0].set_title('Train Set Predictions')
axs[1, 0].set_title('Test Set Predictions')
axs[0, 1].set_title('Train Set MSE')
axs[1, 1].set_title('Test Set MSE')
# etiquetas de los ejes
axs[0, 0].set_xlabel("x train")
axs[0, 0].set_ylabel("y")
axs[0, 0].set_ylim([-3, 3])

axs[1, 0].set_xlabel("x test")
axs[1, 0].set_ylabel("y")
axs[1, 0].set_ylim([-3, 3])

axs[0, 1].set_xlabel("polynomial degree")
axs[0, 1].set_ylabel("train MSE")

axs[1, 1].set_xlabel("polynomial degree")
axs[1, 1].set_ylabel("test MSE")
# dispersion de puntos
axs[0,0].scatter(x_train, y_train, c = 'k', label = "y train")
axs[1,0].scatter(x_test, y_test, c = 'k', label = "y test")

#para almacenar los mse que salgan de cada modelo
train_mse = []
test_mse = []

for k in range(0, NUM_MODELS):

    c = next(colors)

    # polyfit nos da los coeficientes para cada grado
    # k-th degree polynomial coefficients
    fit_coeff = np.polyfit(x_train, y_train, deg = k+1)

    # nos da los valores y para cada coeficiente
    # train and test k-th degree polynomial fit
    y_train_pred = np.polyval(fit_coeff, x_train)
    y_test_pred = np.polyval(fit_coeff, x_test)

    # plot train and test k-th degree polynomial fit
    axs[0,0].scatter(x_train, y_train_pred, color = c, label = "deg: {}".format(k+1))
    axs[1,0].scatter(x_test, y_test_pred, color = c, label = "deg: {}".format(k+1))

    # obtengo los errores
    # train and test MSE of k-th degree polynomial fit
    iter_train_mse = mean_squared_error(y_train_pred, y_train)
    iter_test_mse = mean_squared_error(y_test_pred, y_test)

    train_mse.append(iter_train_mse)
    test_mse.append(iter_test_mse)

    # plot train and test MSE of k-th degree polynomial fit
    axs[0,1].plot(k+1,
                  iter_train_mse,
                  color = c,
                  label = "deg: {}".format(k+1),
                  marker = 'D',
                  markersize = 12,
                  markeredgecolor = 'black',
                  markeredgewidth = 3)
    axs[1,1].plot(k+1,
                  iter_test_mse,
                  color = c,
                  label = "deg: {}".format(k+1),
                  marker = 'D',
                  markersize = 12,
                  markeredgecolor = 'black',
                  markeredgewidth = 3)

# plot dashed line to interpolate MSE measures
axs[0,1].plot(range(1,NUM_MODELS+1), train_mse, 'k--')
axs[1,1].plot(range(1,NUM_MODELS+1), test_mse, 'k--')

# draw legends
axs[0,1].legend(loc = "upper right",
                bbox_to_anchor = (1.2, 1.01),
                prop = {'size': 12})
axs[1,1].legend(loc = "upper right",
                bbox_to_anchor = (1.2, 1.01),
                prop = {'size': 12})
```

```
/Users/tomasnozica/opt/anaconda3/lib/python3.8/site-packages/IPython/core/interactiveshell.py:3437: RankWarnin
g: Polyfit may be poorly conditioned
  exec(code_obj, self.user_global_ns, self.user_ns)
/Users/tomasnozica/opt/anaconda3/lib/python3.8/site-packages/IPython/core/interactiveshell.py:3437: RankWarnin
g: Polyfit may be poorly conditioned
  exec(code_obj, self.user_global_ns, self.user_ns)
/Users/tomasnozica/opt/anaconda3/lib/python3.8/site-packages/IPython/core/interactiveshell.py:3437: RankWarnin
g: Polyfit may be poorly conditioned
  exec(code_obj, self.user_global_ns, self.user_ns)
/Users/tomasnozica/opt/anaconda3/lib/python3.8/site-packages/IPython/core/interactiveshell.py:3437: RankWarnin
g: Polyfit may be poorly conditioned
  exec(code_obj, self.user_global_ns, self.user_ns)
/Users/tomasnozica/opt/anaconda3/lib/python3.8/site-packages/IPython/core/interactiveshell.py:3437: RankWarnin
g: Polyfit may be poorly conditioned
  exec(code_obj, self.user_global_ns, self.user_ns)
/Users/tomasnozica/opt/anaconda3/lib/python3.8/site-packages/IPython/core/interactiveshell.py:3437: RankWarnin
g: Polyfit may be poorly conditioned
  exec(code_obj, self.user_global_ns, self.user_ns)
/Users/tomasnozica/opt/anaconda3/lib/python3.8/site-packages/IPython/core/interactiveshell.py:3437: RankWarnin
g: Polyfit may be poorly conditioned
  exec(code_obj, self.user_global_ns, self.user_ns)
/Users/tomasnozica/opt/anaconda3/lib/python3.8/site-packages/IPython/core/interactiveshell.py:3437: RankWarnin
g: Polyfit may be poorly conditioned
  exec(code_obj, self.user_global_ns, self.user_ns)
```

Out[449]:
```
<matplotlib.legend.Legend at 0x7fec015d5a00>
```



In [450]:
```
print("Best fit polynomial degree: ", test_mse.index(min(test_mse)))
```

```
Best fit polynomial degree:  2
```

Usamos este espacio para agradecerle a Nicolo Cosimo Albanese por haber hecho este artículo ya que estaba dificil armar esta seccion práctica

Cross Validation

Modelo 1 : Y = B1 * X1

Modelo 2 : Y = B2 * X2

Modelo 3 : Y = B1 * X1 + B2 * X2

In [451]:
```
mse = {}
```

MODELO 1

In [452]:
```
import sklearn
boston = pd.read_csv('boston.csv')
```

In [453]:
```
X = boston[['LSTAT']]
y = boston[['MEDV']]
```

In [454]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

In [455]:
```
model_1 = linear_model.LinearRegression().fit(X_train, y_train)
scores = cross_val_score(model_1, X, y, cv=5, scoring='neg_mean_squared_error')
abs(scores)
```

Out[455]:
```
array([23.55825   , 41.82157437, 73.99360893, 50.50118016, 23.21775049])
```

¿ Por que neg MSE ? https://stackoverflow.com/questions/48244219/is-sklearn-metrics-mean-squared-error-the-larger-the-better-negated

In [456]:
```
mse['model1'] = abs(scores.mean())
```

MODELO 2

In [457]:
```
X = boston[['RM']]
y = boston[['MEDV']]
```

In [458]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

In [459]:
```
model_2 = linear_model.LinearRegression().fit(X_train, y_train)
scores = cross_val_score(model_2, X, y, cv=5, scoring='neg_mean_squared_error')
abs(scores)
```

Out[459]:
```
array([ 10.11581024,  33.25013827,  39.73491255, 106.6526283 ,
        73.84557039])
```

In [460]:
```
mse['model2'] = abs(scores.mean())
```

MODELO 3

In [461]:
```
X = boston[['RM', 'LSTAT']]
y = boston[['MEDV']]
```

In [462]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

In [463]:
```
model_3 = linear_model.LinearRegression().fit(X_train, y_train)
scores = cross_val_score(model_3, X, y, cv=5, scoring='neg_mean_squared_error')
abs(scores)
```

Out[463]:
```
array([11.78879381, 28.97215949, 47.84395882, 71.77053822, 36.6099291 ])
```

In [464]:
```
mse['model3'] = abs(scores.mean())
```

In [469]:
```
mse
```

Out[469]:
```
{'model1': 42.6184727894106,
 'model2': 52.719811950004285,
 'model3': 39.39707588807394}
```