In [ ]:
```
from sklearn.datasets import load_boston
import pandas as pd
import matplotlib.pyplot as plt
```

In [ ]:
```
boston_dataset = load_boston()
```

In [ ]:
```
type(boston_dataset) # Bunch means in a very simple way...Data and Meta Data
```

In [ ]:
```
#The dir() function returns all properties and methods of the specified object
dir(boston_dataset)
```

In [ ]:
```
print(boston_dataset.DESCR)
```

In [ ]:
```
print(boston_dataset.data)
```

In [ ]:
```
print(boston_dataset.feature_names)
```

In [ ]:
```
print(boston_dataset.filename)
```

In [ ]:
```
data = pd.DataFrame(data = boston_dataset.data, columns = boston_dataset.feature_names)
```

In [ ]:
```
data.head()
```

In [ ]:
```
data.tail()
```

In [ ]:
```
print(boston_dataset.target)
```

In [ ]:
```
data['PRICE'] = boston_dataset.target
```

In [ ]:
```
data.head()
```

In [ ]:
```
data.tail()
```

In [ ]:
```
data.count()
```

In [ ]:
```
pd.isnull(data)
```

In [ ]:
```
pd.isnull(data).any()
```

In [ ]:
```
data.info()
```

In [ ]:
```
plt.hist(data['PRICE'])
plt.show()
```

In [ ]:
```
plt.hist(data['PRICE'], bins =30)
plt.show()
```

In [ ]:
```
plt.figure(figsize= (10,6))
plt.hist(data['PRICE'], bins =50, ec = 'black', color = '#800000', alpha = 0.7)# bins 5
0...setting color by  hexcode
plt.xlabel('Price in 1000\'s')
plt.ylabel('Nr of Houses')
plt.show()
```

In [ ]:
```
import seaborn as sns
```

In [ ]:
```
sns.distplot(data['PRICE'])
plt.show()
```

In [ ]:
```
plt.figure(figsize= (10,6))
sns.distplot(data['PRICE'], color = 'red', bins = 50)
plt.show()
```

In [ ]:
```
plt.figure(figsize= (10,6))
sns.distplot(data['PRICE'], color = 'red', bins = 50, hist = False)
plt.show()
```

In [ ]:
```
plt.figure(figsize= (10,6))
sns.distplot(data['PRICE'], color = 'red', bins = 50, kde = False) # KDE = Kernel Densit
y Estimates
plt.show()
```

In [ ]:
```
plt.figure(figsize= (10,6))
plt.hist(data['RM'], ec = 'black', color = 'pink')# removed bins
plt.xlabel('Average Number of Rooms')
plt.ylabel('Nr of Houses')
plt.show()
```

In [ ]:
```
data['RM'].mean()
```

In [ ]:
```
plt.figure(figsize= (10,6))
plt.hist(data['RAD'], ec = 'black', color = 'green')# using automatic binning.....
plt.xlabel('Accessibility to Highways')
plt.ylabel('Nr of Houses')
plt.show()
```

In [ ]:
```
data['RAD']
```

In [ ]:
```
data['RAD'].value_counts()
```

In [ ]:
```
plt.figure(figsize= (10,6))
plt.hist(data['RAD'], ec = 'black', bins = 24, color = 'green')# using automatic binnin
g.....
plt.xlabel('Accessibility to Highways')
plt.ylabel('Nr of Houses')
plt.show()
```

In [ ]:
```
accessibility_as_whole = data['RAD']
print(accessibility_as_whole)
```

In [ ]:
```
accessibility = data['RAD'].value_counts()
```

In [ ]:
```
(type(accessibility))
```

In [ ]:
```
print(accessibility)
```

In [ ]:
```
print(accessibility)
```

In [ ]:
```
accessibility.index
```

In [ ]:
```
plt.figure(figsize= (10,6))
plt.bar(accessibility.index, accessibility )
plt.show()
```

In [ ]:
```
plt.figure(figsize= (10,6))
plt.bar(accessibility.index, accessibility, color ='brown', ec = 'blue', width = 0.5)# c
hanging the width, color and ec
plt.xlabel('Accessibility index to Highways')
plt.ylabel('Nr of Houses')
plt.show()
```

In [ ]:
```
data['CHAS'].value_counts()
```

In [ ]:
```
# smallest value of a particular column in our DataFrame.
data['PRICE'].min() #Remember this was in 1978....
```

In [ ]:
```
data['PRICE'].max()
```

In [ ]:
```
data['PRICE'].mean()
```

In [ ]:
```
data.min()
```

In [ ]:
```
data.max()
```

In [ ]:
```
data.mean()
```

In [ ]:
```
data.median()
```

Calculating correlation

$$\rho_{xy} = Corr(X, Y)$$

$$-1.0 \leq \rho_{xy} \leq +1.0$$

In [ ]:
```
data['PRICE'].corr(data['RM'])
```

In [ ]:
```
data['PRICE'].corr(data['PTRATIO'])
```

In [ ]:
```
data.corr()
```

In [ ]:
```
type(data.corr())
```

In [ ]:
```
import numpy as np
```

In [ ]:
```
masker = np.zeros_like(data.corr())
```

In [ ]:
```
masker
```

In [ ]:
```
type(masker)
```

In [ ]:
```
triangle_indices = np.triu_indices_from(masker)
```

In [ ]:
```
triangle_indices
```

In [ ]:
```
masker[triangle_indices] = True
```

In [ ]:
```
masker
```

In [ ]:
```
plt.figure(figsize =(16,10))
sns.heatmap(data.corr())
plt.show()
```

In [ ]:
```
plt.figure(figsize =(10,8))# reduced the size
sns.heatmap(data.corr(),mask = masker) # to mask half of the chart
plt.xticks(fontsize = 10)#adjusted the fonts on the x axis
plt.yticks(fontsize = 10)# adjusted the font on the y axis
plt.show()
```

In [ ]:
```
plt.figure(figsize =(10,8))# reduced the size
sns.heatmap(data.corr(), mask = masker, annot = True) # annotating the values on the hal
f chart
plt.xticks(fontsize = 10)#adjusted the fonts on the x axis
plt.yticks(fontsize = 10)# adjusted the font on the y axis
plt.show()
```

In [ ]:
```
plt.figure(figsize =(16, 10)) # increased the size
sns.heatmap(data.corr(), mask = masker, annot = True, annot_kws ={'size':14} ) # Increas
ing the font size on the annotations
plt.xticks(fontsize = 10)#adjusted the fonts on the x axis
plt.yticks(fontsize = 10)# adjusted the font on the y axis
plt.show()
```

Advanced visualizations using the scatter plot

In [ ]:
```
nox_dis_corr = round(data['NOX'].corr(data['DIS']), 3)
plt.figure(figsize = (16, 8))
plt.title (f'Distance from Employment Centres vs Pollution , Corr ({nox_dis_corr})',
                       fontsize = 25, color = 'green'  )

plt.scatter(x=data['DIS'], y =data['NOX'], alpha = 0.5, color = 'red', s= 80)
plt.xlabel( 'Distance from Employment Centres...DIS', fontsize =14, color = 'black' )
plt.ylabel( 'Level of Pollutants Nitric Oxide...NOX', fontsize =14, color = 'black' )
plt.show()
```

Using Seaborn to carry out the plotting

In [ ]:
```
sns.set()
sns.set_style('dark')
sns.jointplot(x=data['DIS'], y =data['NOX'], color = 'red', joint_kws = {'alpha' : 0.5})
plt.show()
```

In [ ]:
```
sns.set() # to reset any previous styling
sns.set_style('dark')# to set style.
sns.scatterplot(x=data['DIS'], y =data['NOX'], color = 'red', alpha = 0.5) # joint plot
 creates  scatter plot
plt.show()
```

In [ ]:
```
sns.set() # to reset any previous styling
sns.set_style('darkgrid')# to set style.
plt.figure(figsize = (16, 8))
sns.scatterplot(x=data['DIS'], y =data['NOX'], color = 'blue', alpha = 0.5) # joint plot
creates  scatter plot
plt.show()
```

In [ ]:
```
sns.set()
sns.set_style('dark')
sns.jointplot(x=data['DIS'], y =data['NOX'], color = 'red', joint_kws = {'alpha' : 0.5},
kind = 'hex')
plt.show()
```

Visualizing the RAD vs TAX relationship

In [ ]:
```
data['RAD'].corr(data['TAX'])
```

In [ ]:
```
sns.set() # to reset any previous styling
sns.set_style('dark')# to set style.
sns.jointplot(x=data['TAX'], y =data['RAD'], color = 'darkred', joint_kws = {'alpha' :
0.5} ) # jointplot gives  scatter plot
plt.show()
```

Plotting the linear regression of TAX vs RAD....LMPLOT

In [ ]:
```
sns.lmplot(x ='TAX', y = 'RAD', data = data)
plt.show()
```

In [ ]:
```
sns.lmplot(x ='TAX', y = 'RAD', data = data, height =10, aspect = 2)
plt.show()
```

In [ ]:
```
# All defaults regressions with lmplot......
# height = 10 is optional
sns.lmplot(x ='RM', y = 'PRICE', data = data, height =10, aspect = 2, ci = None )# Confi
dence interval set to none....
plt.show()
```

Scatter plot using matplotlib, RM vs PRICE

In [ ]:
```
rm_price_corr = round(data['RM'].corr(data['PRICE']), 3)
plt.figure(figsize = (16, 8))
plt.title (f'Rooms  vs Price , Corr ({rm_price_corr})',
                        fontsize = 25, color = 'green'  )
plt.scatter(x=data['RM'], y =data['PRICE'], alpha = 0.5, color = 'red', s= 80)
plt.xlabel( 'RM ....No of ROOMs ', fontsize =14, color = 'green' )
plt.ylabel( 'House price...PRICE', fontsize =14, color = 'green' )
plt.show()
```

Getting all the plots at one go....using seaborn

In [ ]:
```
#sns.pairplot(data)
#plt.show()
```

In [ ]:
```
#sns.pairplot(data , kind = 'reg')
#plt.show()
```

In [ ]:
```
#sns.pairplot(data, kind='reg', plot_kws={'line_kws':{'color': 'red'}})
#plt.show()
```

Shuffling and splitting the data into training and testing

Explain random_state

In [ ]:
```
my_X =[1,2,3,4,5,6,7,8,9,0]
```

In [ ]:
```
my_y = ['cat', 'dog', 'pen', 'car', 'toy', 'ram', 'jam', 'sam', 'ham','kom' ]
```

In [ ]:
```
from sklearn.model_selection import train_test_split
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(my_X, my_y)
```

In [ ]:
```
X_train
```

In [ ]:
```
y_train
```

In [ ]:
```
X_test
```

In [ ]:
```
y_test
```

test_size demo is below

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(my_X, my_y,
                                                    test_size=0.2)
```

In [ ]:
```
X_train
```

In [ ]:
```
y_train
```

In [ ]:
```
X_test
```

In [ ]:
```
y_test
```

Randomness....

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(my_X, my_y,
                                                    test_size=0.2, random_state=0)
```

In [ ]:
```
X_train
```

In [ ]:
```
y_train
```

In [ ]:
```
X_test
```

In [ ]:
```
y_test
```

In [ ]:
```
prices = data['PRICE']
features = data.drop('PRICE', axis=1)
```

In [ ]:
```
prices
```

In [ ]:
```
features
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(features, prices,
                                                    test_size=0.2, random_state=10)
```

In [ ]:
```
X_train
```

In [ ]:
```
y_train
```

In [ ]:
```
X_test
```

In [ ]:
```
y_test
```

In [ ]:
```
# checking the  % of training set
round(len(X_train)/len(features),2)*100
```

Running the multivariable regression

$$\hat{y} = \theta_0 + \theta_1 x$$

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$$

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$$

$$\hat{PRICE} = \theta_0 + \theta_1 RM + \theta_2 NOX + \ldots + \theta_{13} LSTAT$$

In [ ]:
```
from sklearn.linear_model import LinearRegression
```

In [ ]:
```
regr = LinearRegression()
```

In [ ]:
```
regr.fit(X_train, y_train)
```

In [ ]:
```
pd.DataFrame(data=regr.coef_, index=X_train.columns, columns=['coef'])
```

In [ ]:
```
print('Intercept', regr.intercept_)
```

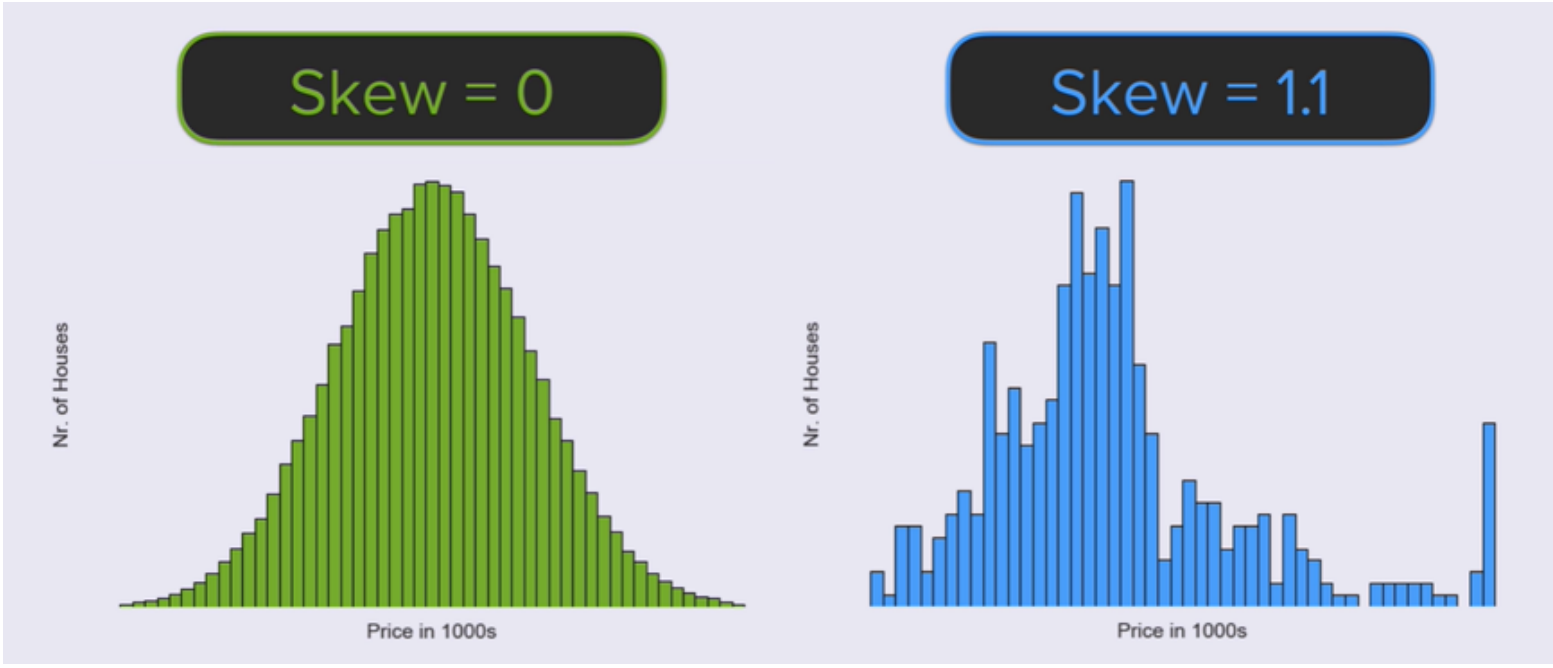
In [ ]:
```
# Calculating the regression score using the training data.......
print('Training data r-squared:', regr.score(X_train, y_train))
```

In [ ]:
```
# calculating the regression score using the testing data.......
print('Test data r-squared:', regr.score(X_test, y_test))
```

Data Transformations

In [ ]:
```
plt.figure(figsize= (16,8))
plt.hist(data['PRICE'], bins =50, ec = 'black', color = 'blue', alpha = 0.7)# bins 50...
setting color by  hexcode
plt.xlabel('Price in 1000\'s')
plt.ylabel('Nr of Houses')
plt.show()
```

In [ ]:
```
# Measuring the skew!!!
```



What is a transformation?

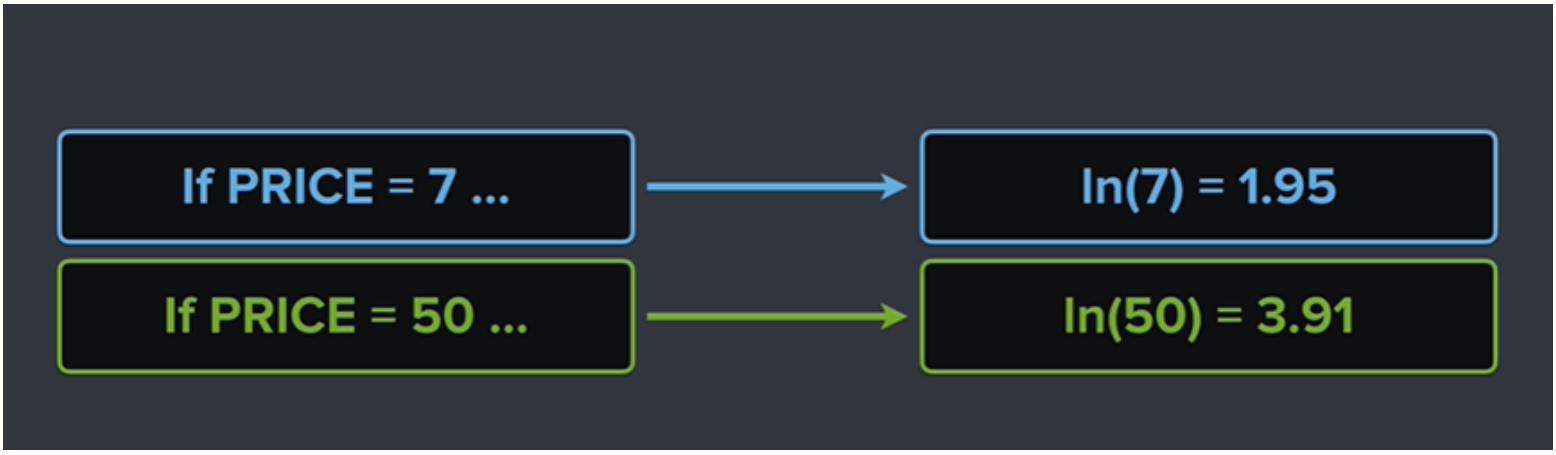
Multiplying all the prices by 5 or dividing all the prices by 2

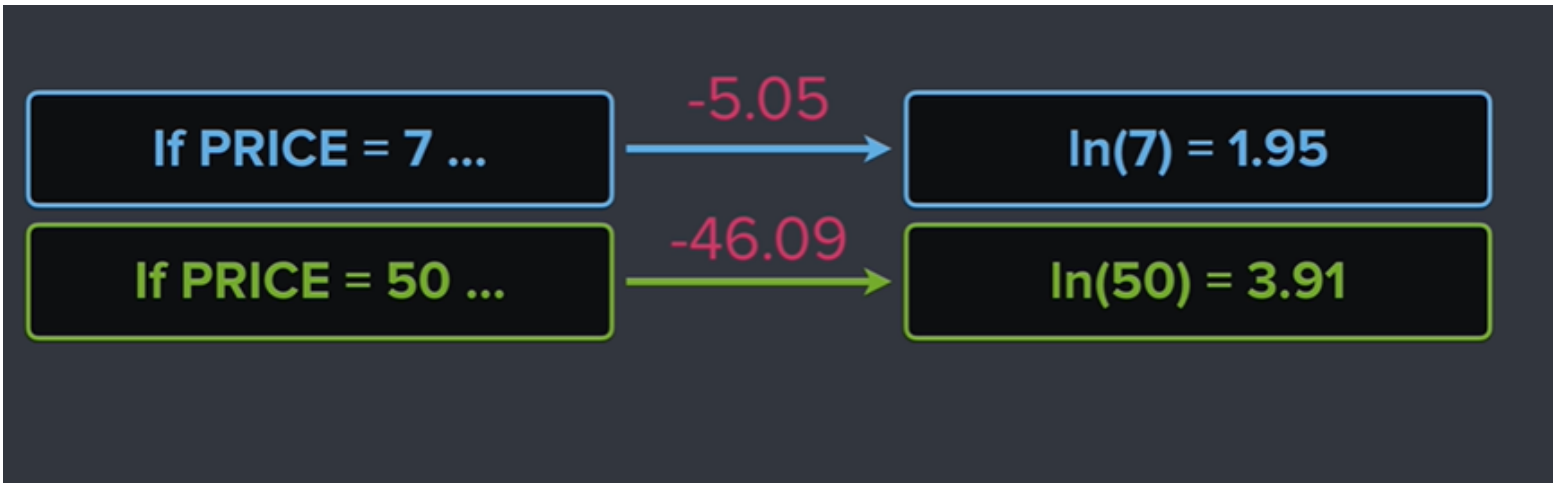
Applying some sort of calculation to all the prices in the data set

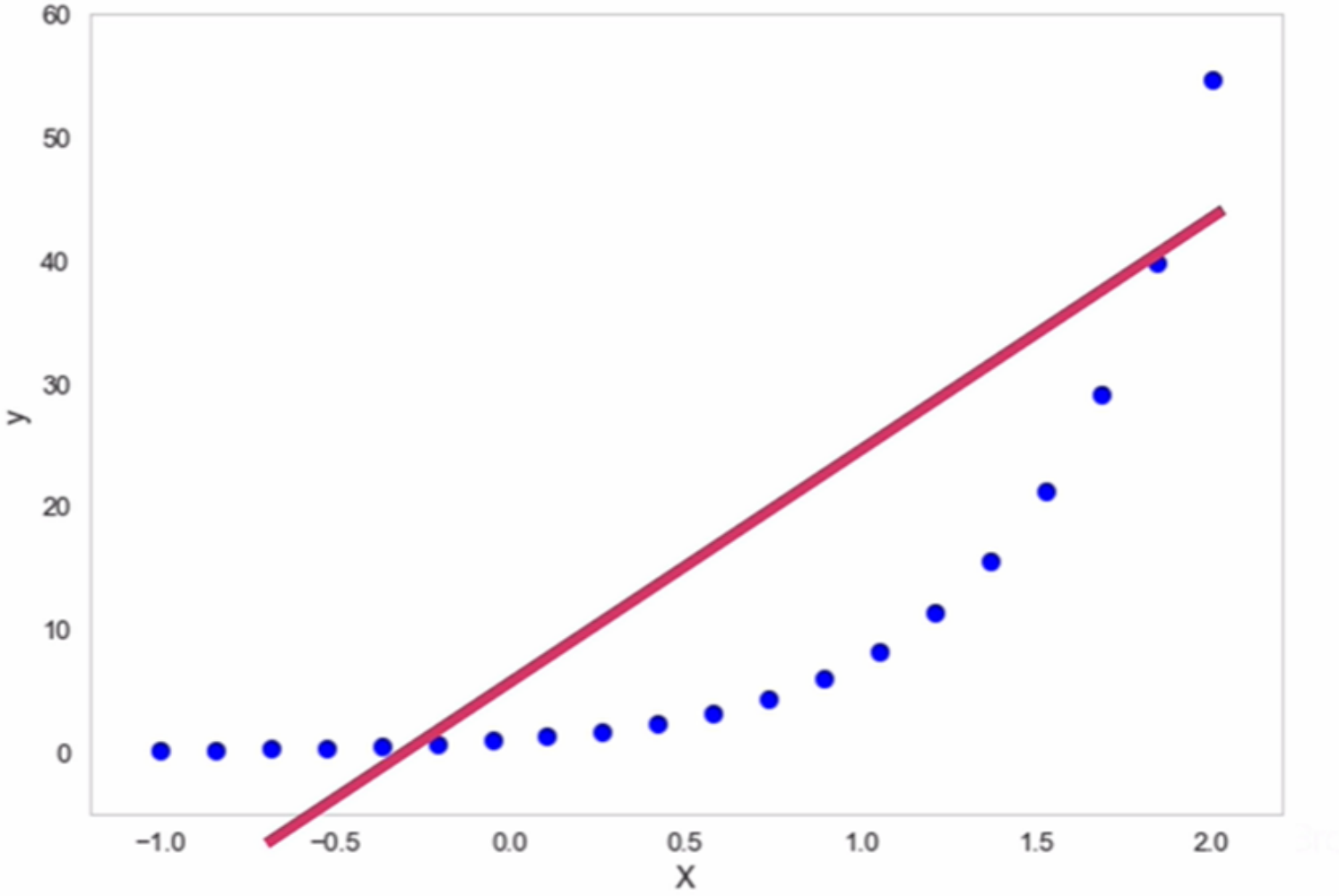
Here, we have to do something to shift the prices on the right tail towards the middle

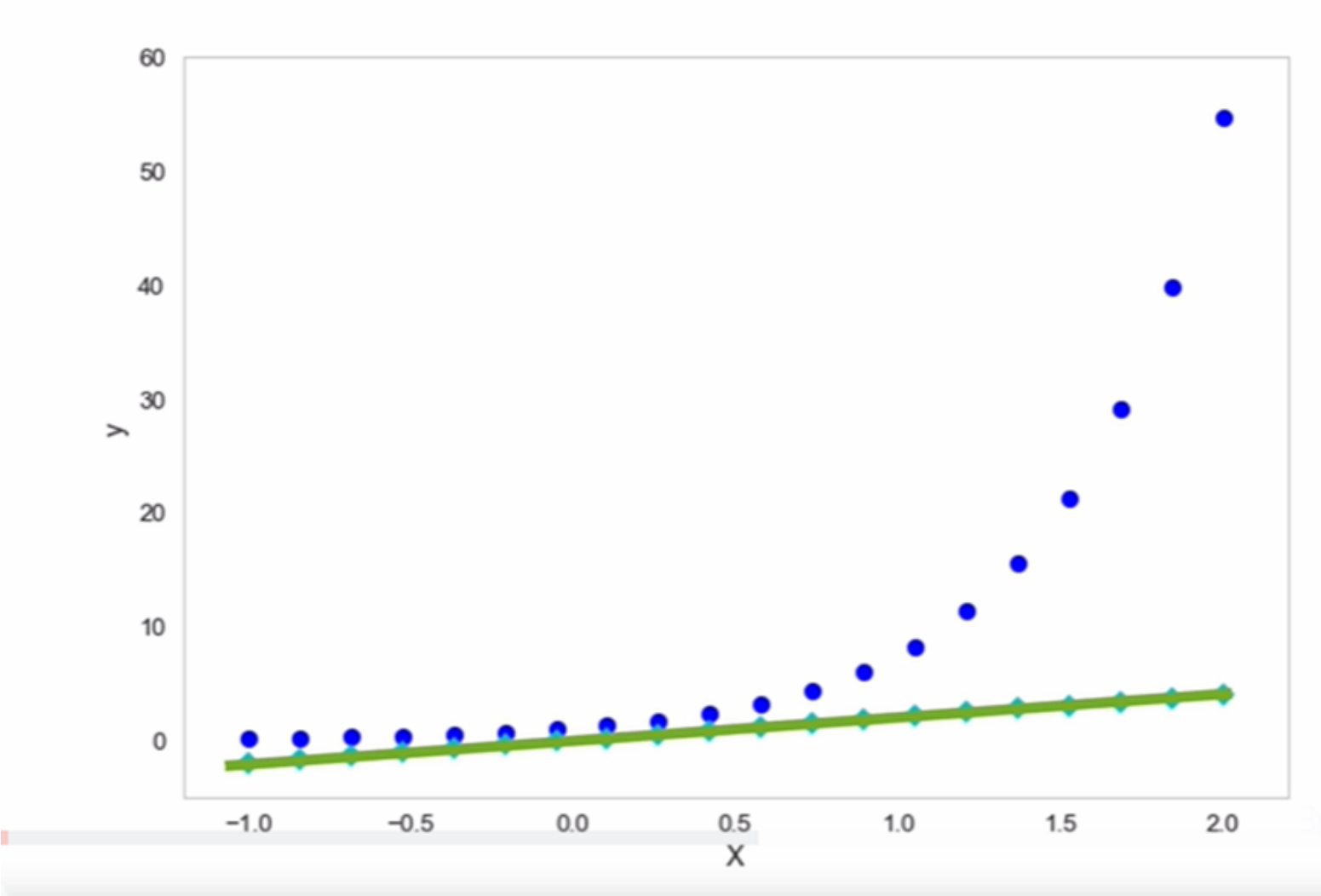
To do this we will use a log transformation!

What is a log transformation?


If PRICE = 7 ...
ln(7) = 1.95
If PRICE = 50 ...
ln(50) = 3.91


If PRICE = 7 ...
-5.05
ln(7) = 1.95
If PRICE = 50 ...
-46.09
ln(50) = 3.91


60
50
40
30
20
10
0
y
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
X



LOG TRANSFORMATIONS

In [ ]:
```
y_log = np.log(data['PRICE'])# We used the log function from numpy....we did not import
 log from math.....
```

In [ ]:
```
y_log.head()
```

In [ ]:
```
y_log.tail()
```

In [ ]:
```
y_log.skew() # for comparision
```

In [ ]:
```
data['PRICE'].skew() # for comparision.........
```

In [ ]:
```
# plotting after log transformation of the prices.....
sns.set()
plt.figure(figsize=(20,6))
sns.distplot(y_log)
plt.title(f'Log price: The skew is {round(y_log.skew(),2)}', fontsize = 25)# note the f
 string used to print the y_log.skew()
plt.xlabel('LOG PRICES ', fontsize =14, color = 'green' )
plt.show()
```

In [ ]:
```
# plotting before the log transformation.............
skew=round(data['PRICE'].skew(),2)
sns.set()
plt.figure(figsize=(20,6))
sns.distplot(data['PRICE'])
plt.title(f'Actual price: The skew is {skew}', fontsize = 25)
plt.xlabel( 'ACTUAL PRICES ', fontsize =14, color = 'green' )
plt.show()
```

Plotting to compare using the untransformed data...LSTAT vs PRICE

In [ ]:
```
sns.lmplot(x='LSTAT', y='PRICE', data=data, height=10, aspect =2,
           scatter_kws={'alpha': 0.6}, line_kws={'color':'darkred'})
plt.show()
```

In [ ]:
```
transformed_data = features # new data frame
transformed_data['LOG_PRICE'] = y_log # added the log price to the new data frame called
transformed data
transformed_data
```

In [ ]:
```
sns.lmplot(x='LSTAT', y='LOG_PRICE', data=transformed_data, height =10,aspect = 2,
           scatter_kws={'alpha': 0.6}, line_kws={'color':'cyan'})
plt.show()
```

Regression with LOG PRICES.....

$$\log_e(12) \approx 2.485$$

$$e^{2.485} \approx 12$$

$$\hat{PRICE} = \theta_0 + \theta_1 RM + \theta_2 NOX + \theta_3 DIS + \theta_4 CHAS + \ldots + \theta_{13} LSTAT$$

$$\log(\hat{PRICE}) = \theta_0 + \theta_1 RM + \theta_2 NOX + \theta_3 DIS + \theta_4 CHAS + \ldots + \theta_{13} LSTAT$$

In [ ]:
```
prices = np.log(data['PRICE']) # Use log prices
features = data.drop('PRICE', axis=1)

X_train, X_test, y_train, y_test = train_test_split(features, prices,
                                                    test_size=0.2, random_state=10)

regr = LinearRegression()
regr.fit(X_train, y_train)

print('Training data r-squared:', regr.score(X_train, y_train))
print('Test data r-squared:', regr.score(X_test, y_test))


pd.DataFrame(data=regr.coef_, index=X_train.columns, columns=['coef'])
```

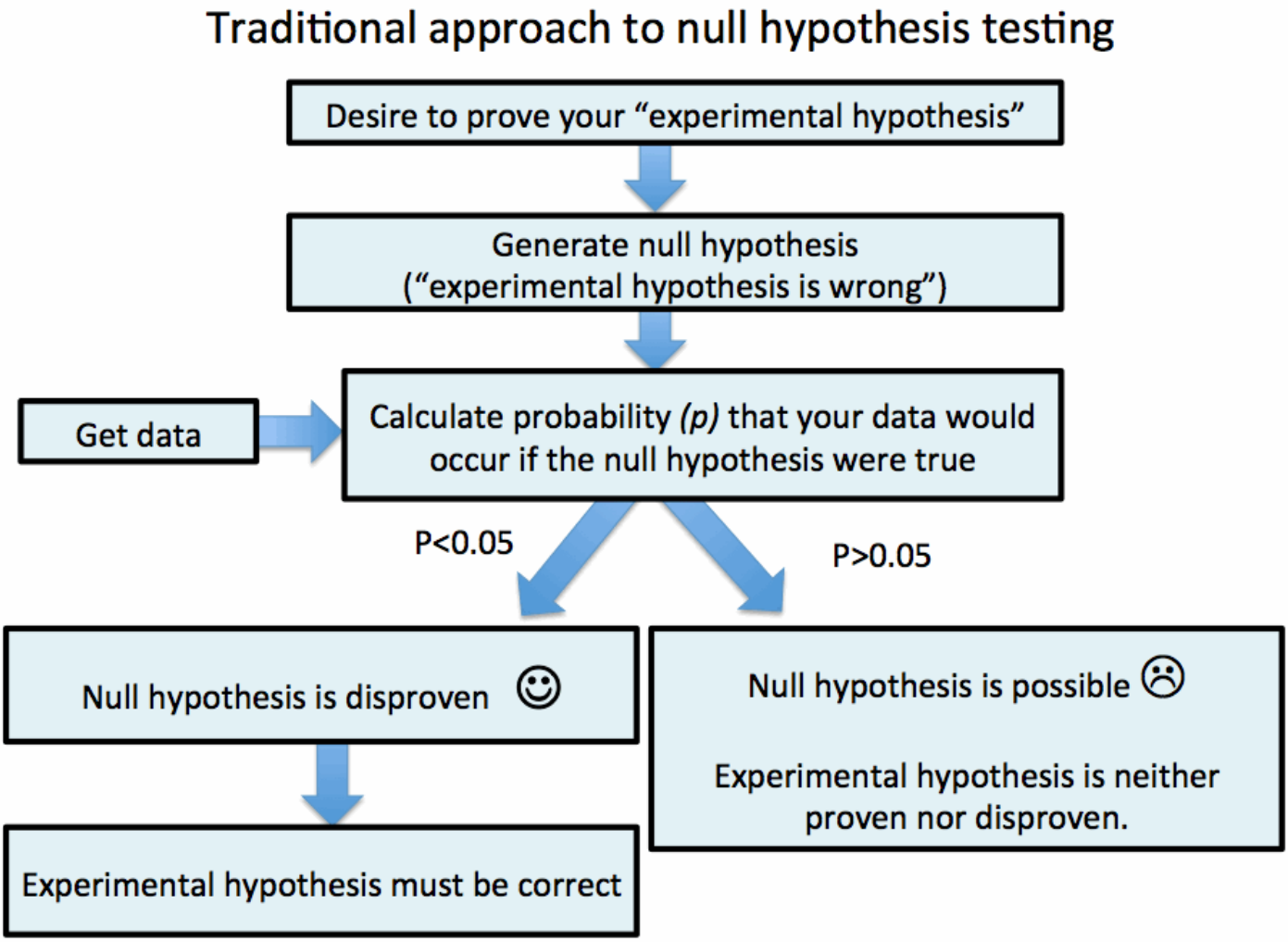
In [ ]:
```
print('Intercept', regr.intercept_)
```

In [ ]:
```
# converting to an actual number.............

np.e**0.080475
```

statsmodels

statsmodels is a Python module that provides classes and functions for the estimation of many different statistical models, as well as for conducting statistical tests, and statistical data exploration. An extensive list of result statistics are available for each estimator. The results are tested against existing statistical packages to ensure that they are correct. The package is released under the open source Modified BSD (3-clause) license. The online documentation is hosted at statsmodels.org.



In [ ]:
```
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [ ]:
```
results = sm.OLS(y_train, sm.add_constant(X_train)).fit()
print(results.params, '\n')
print(results.pvalues,'\n')
```

In [ ]:
```
pd.DataFrame({'coef': results.params, 'p-value': round(results.pvalues, 3)})
```

Testing for multicollinearity using VIF

Variance Inflation Factor (VIF)

- Diagnostic for multicollinearity
- Describes the amount of an X that is explained by the other X's in the model
- If VIF is high, then it suggests that the covariate should not be added.
- Why?
  - it is redundant
  - it adds variance to the model
  - it creates 'instability' in the estimation

$$TAX = \alpha_0 + \alpha_1 RM + \alpha_2 NOX + \ldots + \alpha_{12} LSTAT$$

$$VIF_{TAX} = \frac{1}{(1 - R^2_{TAX})}$$

If VIF of any feature is > 10 we have a problem!!

Formal check for multicollinearity

VIF, spits out a number which quantifies the severity of multicollinearity

We start out by obtaining a regression model for the suspected feature with the other features

endog, exog, what's that?

statsmodels is using `endog` and `exog` as names for the data, the observed variables that are used in an estimation problem. Other names that are often used in different statistical packages or text books are, for example,

endog	exog
y	x
y variable	x variable
left hand side (LHS)	right hand side (RHS)
dependent variable	independent variable
regressand	regressors
outcome	design
response variable	explanatory variable

In [ ]:
```
variance_inflation_factor(exog=sm.add_constant(X_train).values, exog_idx=10)
```

In [ ]:
```
X_incl_const = sm.add_constant(X_train)
for i in range(len(X_incl_const.columns)):# running it for length of columns of X_incl_c
onst
    print(variance_inflation_factor(exog=X_incl_const.values, exog_idx=i),'\n')
```

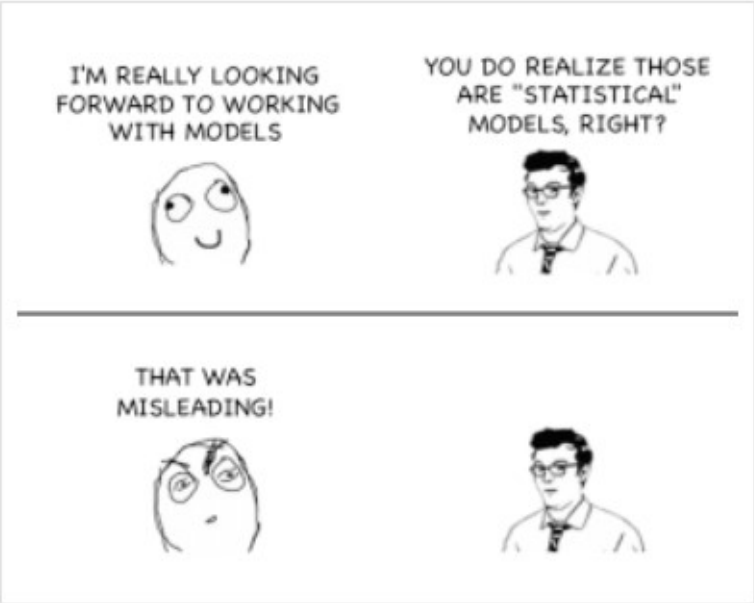
In [ ]:
```
vif = [] # empty list
for i in range(X_incl_const.shape[1]):# accessing the number of columns uisng the .shape
[1] for columns
    vif.append(variance_inflation_factor(exog=X_incl_const.values, exog_idx=i))
print(vif)
```

In [ ]:
```
type(vif)
```

In [ ]:
```
pd.DataFrame({'coef_name': X_incl_const.columns, 'vif': np.around(vif, 2)})
```

Model Simplification & the BIC

What Is Bayesian Information Criterion?



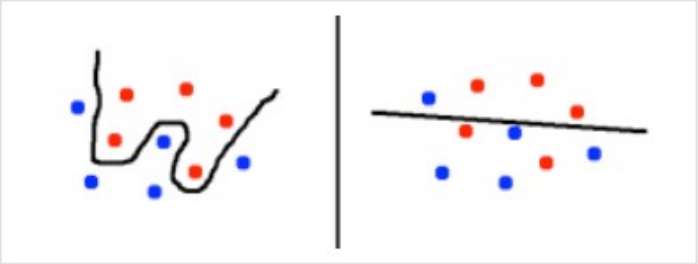
Let's say you have a bunch of datapoints and you want to come up with a nice model for them. We want this model to satisfy all the points in the best possible way. If we do this, then we will be able to use a mathematical formula to extract information about unknown points. At the same time, we should make sure that we don't overfit our model to these datapoints. If we overfit our model, then it will tune itself too much to our datapoints and perform poorly on unknown data. So how we pick the best model? Where do we draw the line?

First of all, what's this "model" you keep talking about?

Before we proceed, we need to understand what we mean by "model" in this context. Here, we are talking about a statistical model, which is basically a generalization of the observed data. Let's take a very simple example to understand what this means. You randomly survey 1000 people from India and Brazil, and ask them what their favorite sport is. Once you have that information, you will obtain a set of values mapping those people to their favorite sport. Now, if you extract a statistical model from this data, you will see that people who like cricket are most likely from India and people who like football are most likely from Brazil. Bear in mind that this is a statistical model, which means it's just a probability and not the absolute truth! Let's say you want use this model to predict the nationality of a person based on his/her favorite sport. Now, you ask an unknown person what his favorite sport is and he says football. If you enter this value into your model, it will say that this person is most likely to be from Brazil.

This model seems fine to me! Why do we need to "pick" a model?

This is where things get interesting. The example we just discussed was a very simple example. We just considered a simple probability to see what's more likely in that case. But real world is not so nice and simple! In the real world, we'll have 10,000 questions and 200 countries. Now that's a lot of data. We cannot use that simple model that will give us the right answer.



Consider the figure on the left. The curvy line separates the two sets of points really well. The accuracy is high and we are all happy. But the set of datapoints we have is a small set. This means that we don't really know what the real world data actually looks like. We just hope that these points are good enough to be representative of the real world data and build a model based on that. So if we fine-tune it too much by making it very curvy, we might actually perform really badly on unknown data.

On the other hand, the straight line separator might generalize very well because we are not fine tuning it at all. But at the same time, it doesn't really separate the two sets of points that well. So the best answer lies somewhere between these two extremes. There are a lot of models between these two extremes. The question is, how do we evaluate these models and pick the best one?

Say hello to Bayesian Information Criterion

This is where Bayesian Information Criterion (BIC) comes in handy. It is a method to choose the best model among a finite set of models. As we add more parameters to a model, the accuracy increases. But at the same time, it will also increase the chances of overfitting. So BIC solves this problem by penalizing the "curviness" of a model. That way, we are not blindsighted by the accuracy provided by a particular model. We will not be going into the mathematical details of BIC, because that's not why we are here. We just want to understand how it selects the best model. Let's see if we can do that without any equations. Having said that, the mathematical formulation is actually really nice. So if you want to take a peek, go ahead and google it. It's a lot of fun!

The curviness of a model depends on the number of parameters we add to that model. So in order to select the best model, we need to define a function which takes this into account. BIC can basically measure the efficiency of a model in terms of its ability to predict the data. BIC is defined in terms of the likelihood function, the number of parameters, and the number of datapoints. The model with the lowest BIC is the best one. So once we build a set of models, we evaluate the BIC for each of them and pick the one with the lowest BIC.

Original model with log prices and all features....MODEL1

In [ ]:
```
X_incl_const = sm.add_constant(X_train)
model = sm.OLS(y_train, X_incl_const)
results_1 = model.fit()
org_coef = pd.DataFrame({'coef': results_1.params, 'p-value': round(results_1.pvalues, 3
)})

print('BIC with all features and the log price is      :' , results_1.bic, '\n')
print('r-squared is with all features and log price is :' , results_1.rsquared, '\n')
print(org_coef)    # the DataFrame with all the coef's and P values
```

Model after dropping the INDUS feature ....MODEL 2

In [ ]:
```
X_incl_const = sm.add_constant(X_train)
X_incl_const = X_incl_const.drop(['INDUS'], axis =1)
model = sm.OLS(y_train, X_incl_const)
results_2 = model.fit()
coef_minus_indus = pd.DataFrame({'coef': results_2.params, 'p-value': round(results_2.pv
alues, 3)})

print('BIC without INDUS features and the log price is      :' , results_2.bic, '\n')
print('r-squared without INDUS feature  and log price is    :' , results_2.rsquared, '\n
')
print(coef_minus_indus)
```

Model after dropping the INDUS and AGE feature....MODEL 3

In [ ]:
```
X_incl_const = sm.add_constant(X_train)
X_incl_const = X_incl_const.drop(['INDUS','AGE'], axis =1)
model = sm.OLS(y_train, X_incl_const)
results_3 = model.fit()
coef_minus_indus_age = pd.DataFrame({'coef': results_3.params, 'p-value': round(results_
3.pvalues, 3)})


print('BIC without INDUS and AGE features and the log price is      :' , results_3.bic,
'\n')
print('r-squared  without INDUS and AGE features and log price is   :' , results_3.rsqua
red,'\n')
print(coef_minus_indus_age)
```

In [ ]:
```
frames = [org_coef, coef_minus_indus, coef_minus_indus_age]
pd.concat(frames, axis=1, sort = False) # gives Future Warning without the sort
```

In [ ]:
```
print(f'BIC WITH ALL::   {results_1.bic}: W/O INDUS {results_2.bic}: W/O INDUS AND AGE
{results_3.bic} ')
```

In [ ]:
```
print(f'RSQD:: WITH ALL {(results_1.rsquared)}: W/O INDUS {results_2.rsquared}: W/O INDU
S,AGE {results_3.rsquared}')
```

Residual and Residual plots

Equation to predict a property price

Our model after dropping two features

$$\log(\hat{PRICE}) = \theta_0 + \theta_1 RM + \theta_2 NOX + \theta_3 DIS + \theta_4 CHAS + \ldots + \theta_{11} LSTAT$$

$$\hat{y} = \theta_0 + \theta_1 RM + \theta_2 NOX + \theta_3 DIS + \theta_4 CHAS + \ldots + \theta_{11} LSTAT$$

Predicted house price (of course log(price) in this case

$$y \neq \hat{y}$$

Residual

$$r = y - \hat{y}$$

$$r = y - \hat{y}$$

Residuals

Residuals

Difference btw the target value and the predicted value.

Can we get predictive information from the residuals???

Since we have 404 Target values in our training dataset we have 404 predicted or fitted values as well. This means we have 404 residuals. How do the residuals look like as a group.

hmm ... the world is such a complex place...
but maybe a simple linear model is good enough...

??????????!!!!!!!!!

Can we find some patterns in our residuals??

$$r = y - \hat{y}$$

Residuals

Residuals

Difference btw the target value and the predicted value.

Can we get predictive information from the residuals???

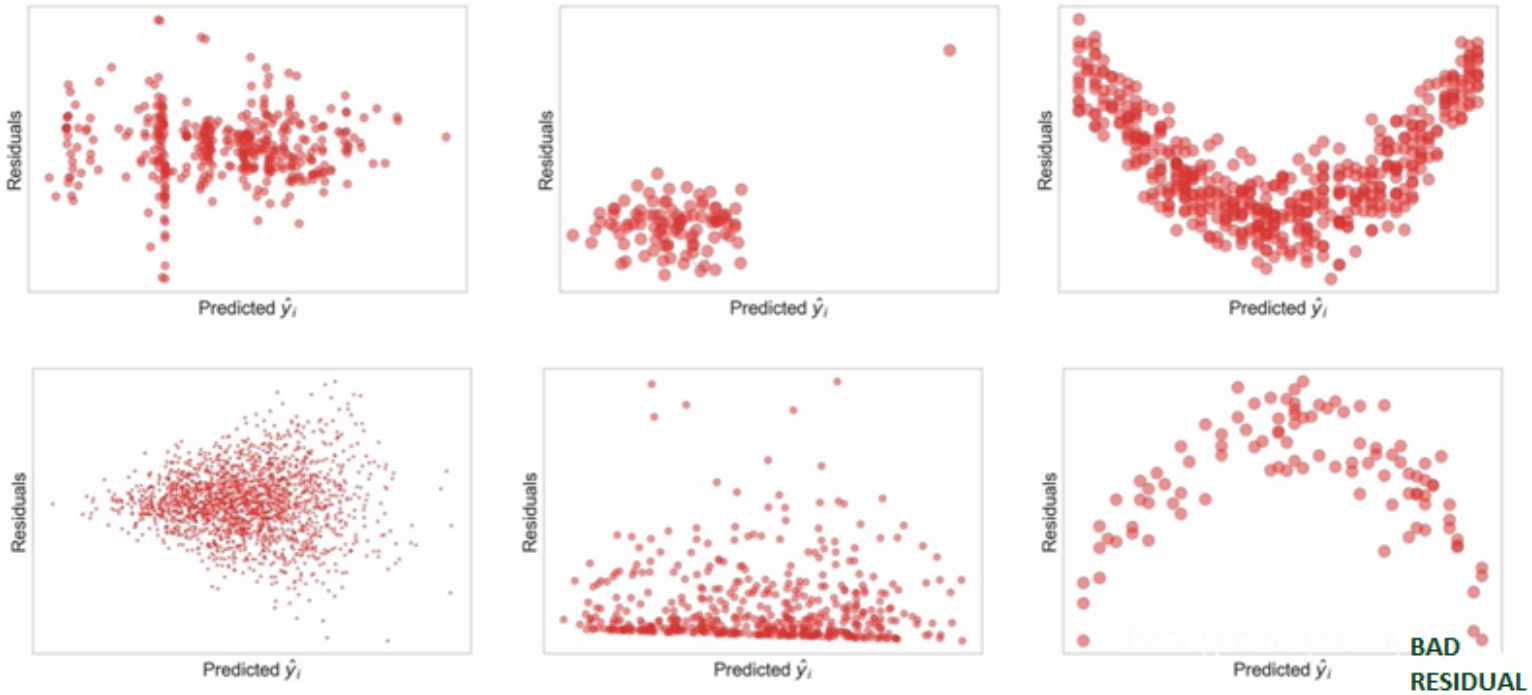
Since we have 404 Target values in our training dataset we have 404 predicted or fitted values as well. This means we have 404 residuals. How do the residuals look like as a group.

hmm ... the world is such a complex place...
but maybe a simple linear model is good enough...

??????????!!!!!!!!!

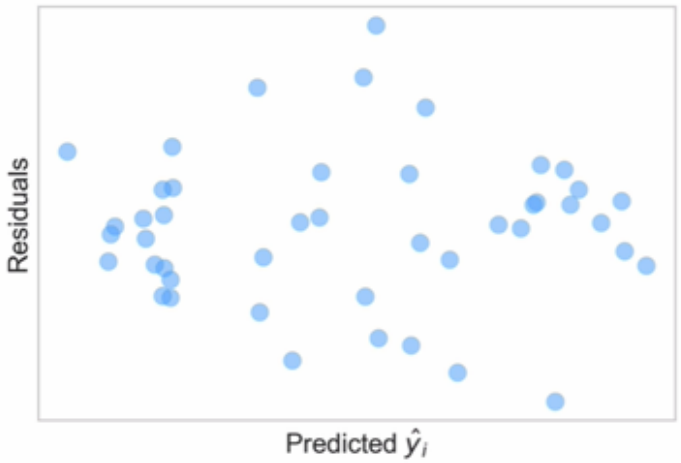
Can we find some patterns in our residuals??

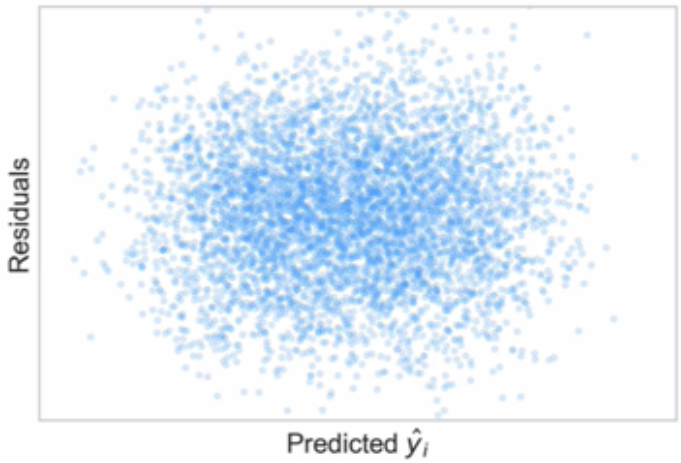
Any non-random pattern indicates there is an issue



BAD RESIDUAL PLOTS

RANDOM RESIDUAL PLOTS!!!!





Less Data Points

More Data Points

Residuals

Difference btw the target value and the predicted value.

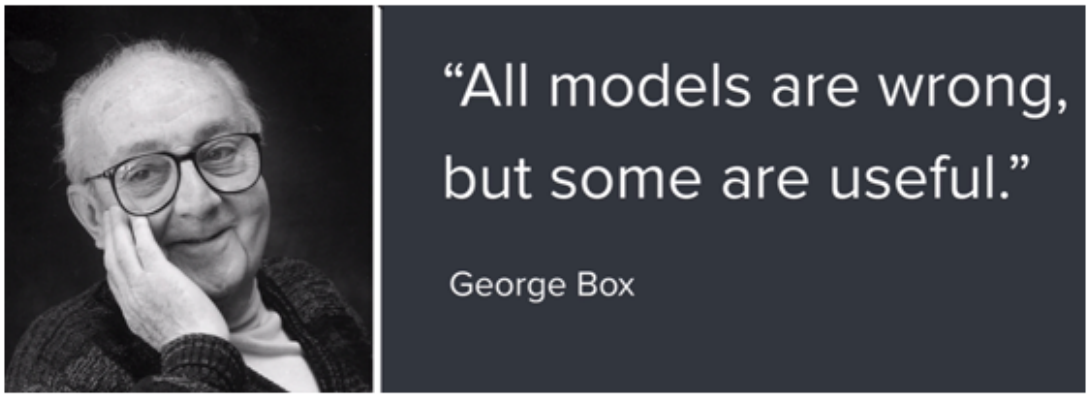
Used to check if assumptions hold and model is valid.

Residuals should be random (i.e., no pattern).

Residuals should be normally distributed.

"...the statistician knows...that in nature there never was a normal distribution, there never was a straight line, yet with normal and linear assumptions, known to be false, he can often derive results which match, to a useful approximation, those found in the real world."

George Box (JASA, 1976, Vol. 71, 791-799)



In [ ]: