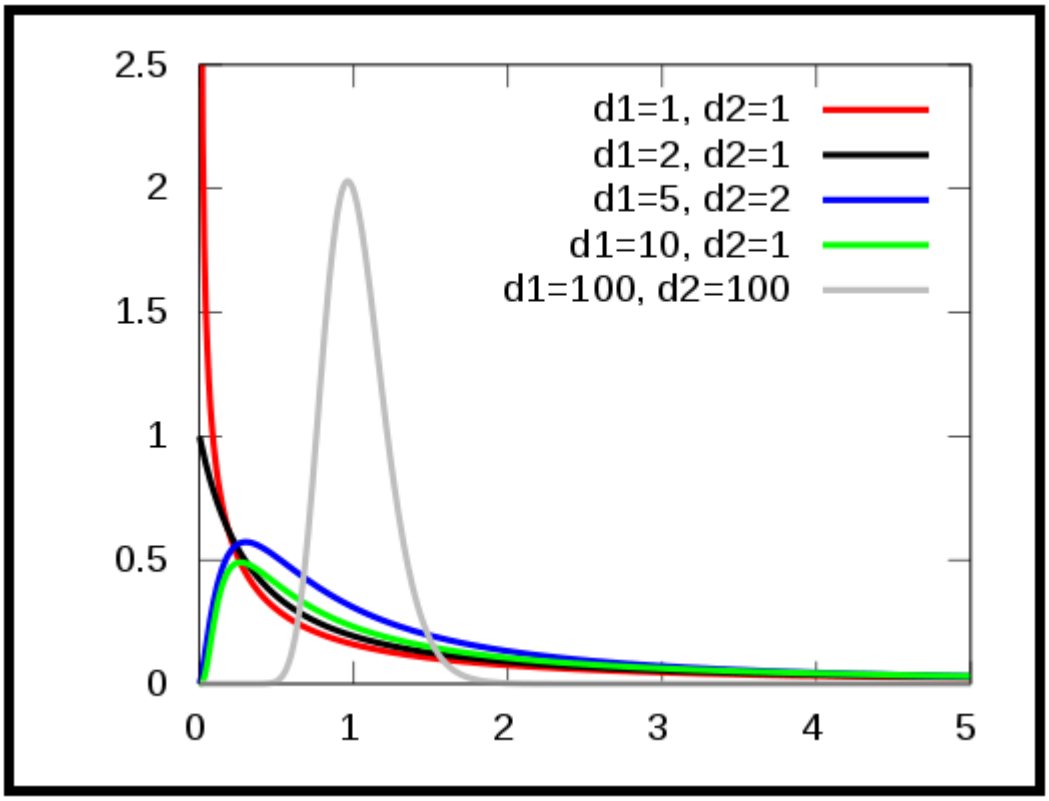


The F-distribution

The F-Test for Regression Analysis

How to use it, how to interpret its results

Sachin Date Follow
Oct 27, 2019 · 11 min read ★

The F-test, when used for regression analysis, lets you compare two competing regression models in their ability to "explain" the variance in the dependent variable.

The F-test is used primarily in ANOVA and in regression analysis. We'll study its use in *linear* regression.

. . .

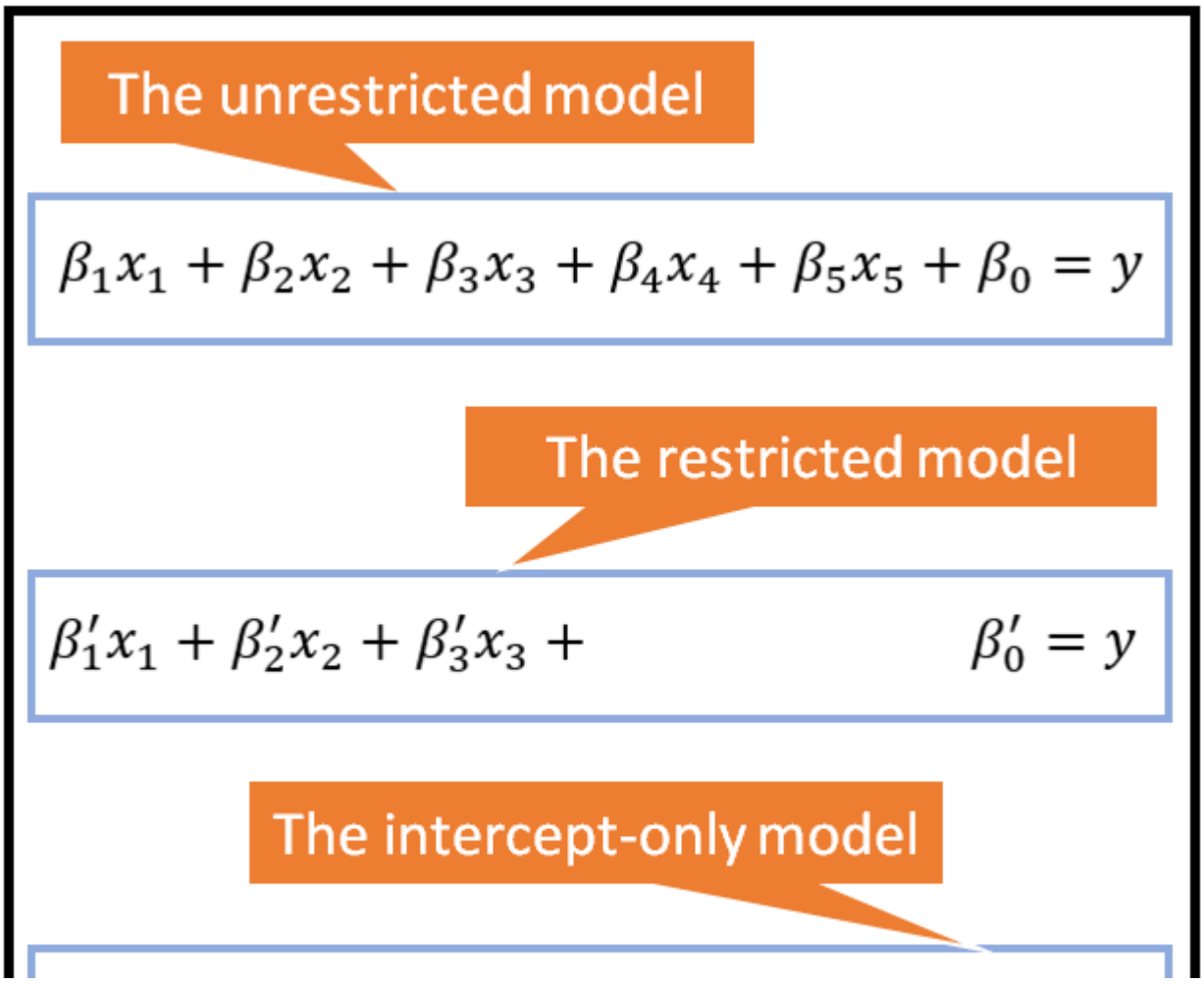
Why use the F-test in regression analysis

In linear regression, the F-test can be used to answer the following questions:

- Will you be able to improve your linear regression model by making it more complex i.e. by adding more linear regression variables to it?
- If you already have a complex regression model, would you be better off trading your complex model with the intercept-only model (which is the simplest linear regression model you can build)?

The second question is a special case of the first question. In both cases, the two models are said to be **nested**. The simpler model is called the **restricted model.** It is as if we are restricting it to use fewer regression variables. The complex model is called the **unrestricted model.** It contains all the variables of the restricted model and at least one more variable.

The restricted model is said to be nested within the unrestricted model.



$$\beta_0'' = y$$

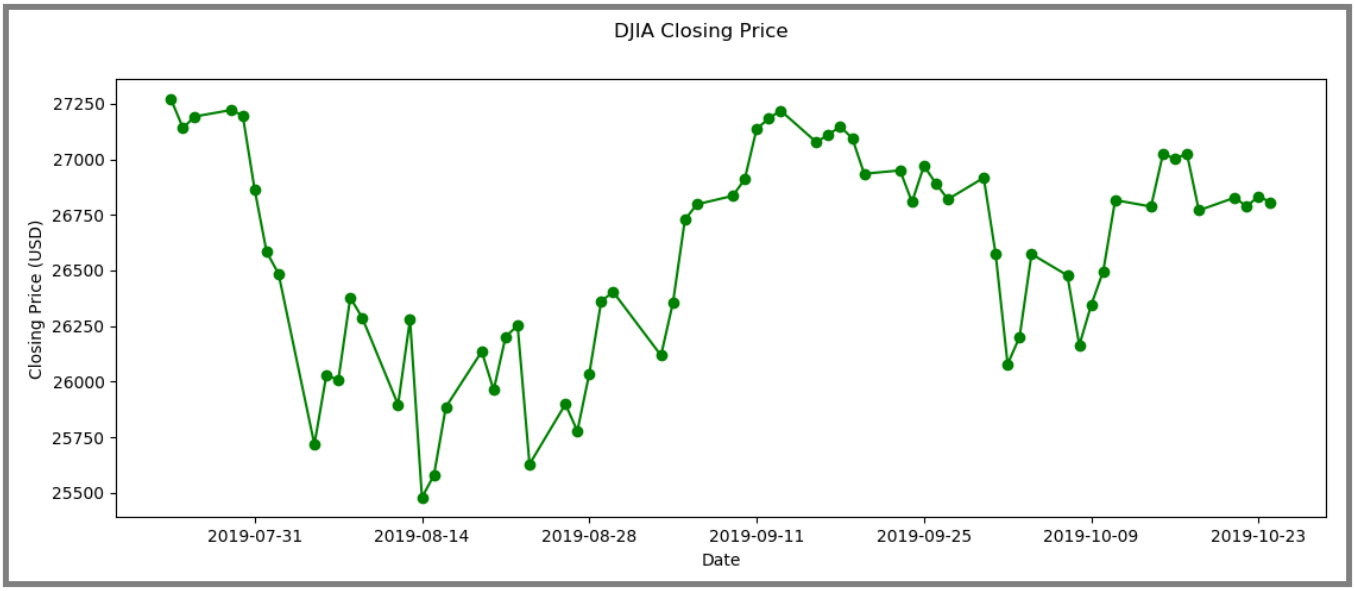
Linear regression models: unrestricted, restricted and intercept-only (restricted)

Let's explore the use of the F-test using a real-world time series example. We'll start by building an intercept-only model —the restricted model.

. . .

A brief look at the intercept-only model

The following time series shows the daily closing price of the Dow Jones Industrial Average over a 3-month period.



Dow Jones Industrial Average closing price over a 3-month period

Suppose we wish to create a regression model for this time series. But we don't know what factors influence the Closing Price. Neither do we want to assume any inflation, trend or seasonality in the data set.

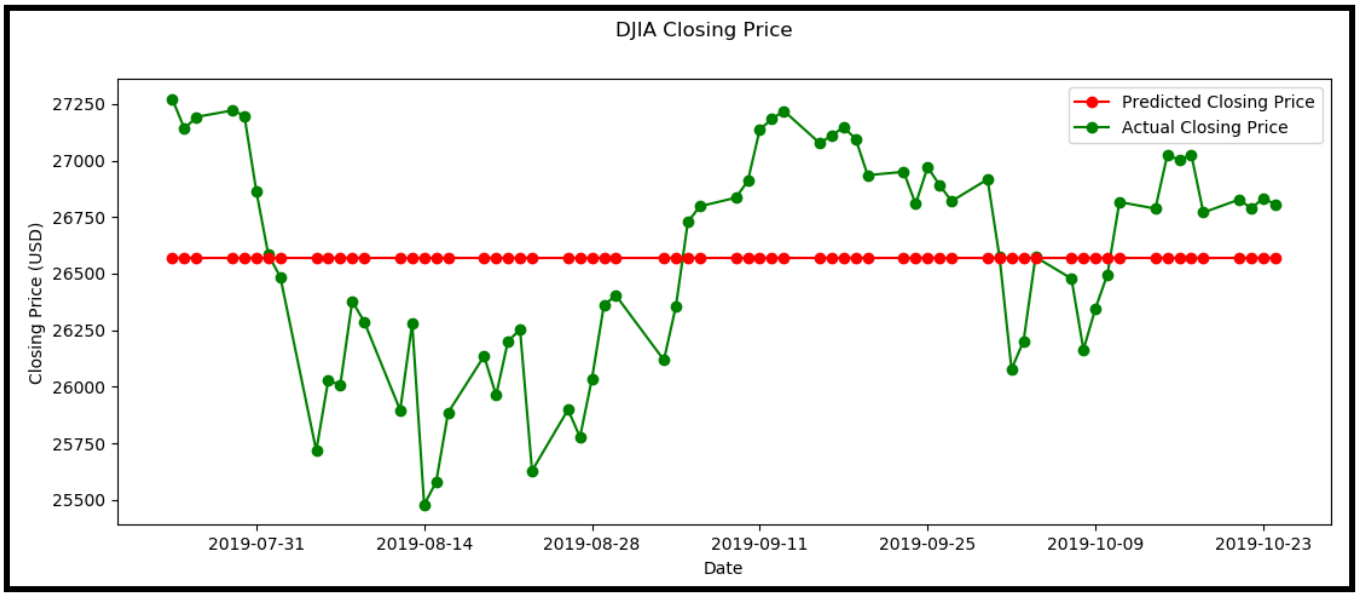
In the absence of any assumptions about inflation, trend, seasonality or the presence of explanatory variables, the best we can do is the intercept-only model (sometimes known as the **mean model**). It takes on the following form for our time series example:

β_0 = sample mean

$$Closing\ Price = \beta_0$$

Intercept-only model for the DJIA data set

In the intercept-only model, all forecasts take the value of the intercept Beta_0. The following plot shows the fitted intercept-only model against the backdrop of the actual time series:



Actual and predicted Closing Price of DJIA using the mean model

Here is the Python code to produce the above results:

Import all the required packages:

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

Read the data set into a Pandas Data Frame:

```
df = pd.read_csv('djia.csv', header=0, infer_datetime_format=True,
parse_dates=[0], index_col=[0])
```

Calculate the sample mean and set all the predicted values to this mean value:

```
mean = round(df['Closing Price'].mean(),2)

y_pred = np.full(len(df['Closing Price']), mean)
```

Plot the actual and the predicted values:

```
fig = plt.figure()

fig.suptitle('DJIA Closing Price')

actual, = plt.plot(df.index, df['Closing Price'], 'go-',
label='Actual Closing Price')

predicted, = plt.plot(df.index, y_pred, 'ro-', label='Predicted
Closing Price')

plt.xlabel('Date')

plt.ylabel('Closing Price (USD)')

plt.legend(handles=[predicted, actual])

plt.show()
```

Can we do any better than the mean model? Perhaps we can. Let's try to develop a competing, unrestricted model for this time series.

. . .

A competing model

Suppose by means of some analysis, we have deduced that today's value of the DJIA Closing Price may turn out to be a good predictor of tomorrow's Closing Price.

To test this theory, we will develop a linear regression model consisting of a single regression variable. This variable will be the time lagged value of the time series. The following Python code illustrates the regression process:

Import the required packages:

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
```

Read the data set into a Pandas Data Frame:

```
df = pd.read_csv('djia.csv', header=0, infer_datetime_format=True,
parse_dates=[0], index_col=[0])
```

Add the time-lagged column:

```
df['CP_LAGGED'] = df['Closing Price'].shift(1)
```

Here are the first few rows of the modified Data Frame. The first row contains a NaN as there is nothing to lag that value with:

```
            Closing Price    CP_LAGGED
Date
2019-07-24    27269.97070          NaN
2019-07-25    27140.98047  27269.97070
2019-07-26    27192.44922  27140.98047
2019-07-29    27221.34961  27192.44922
2019-07-30    27198.01953  27221.34961
```

Let's remove the first row to get rid of the NaN:

```
df_lagged = df.drop(df.index[0])
```

Next let's create our training and test data sets:

```
split_index = round(len(df_lagged)*0.8)

split_date = df_lagged.index[split_index]

df_train = df_lagged.loc[df_lagged.index <= split_date].copy()

df_test = df_lagged.loc[df_lagged.index > split_date].copy()

X_train = df_train['CP_LAGGED'].values

#Add a placeholder for the constant so that model computes an
intercept value. The OLS regression equation will take the form: y =
Beta_0 + Beta_1*x
X_train = sm.add_constant(X_train)

y_train = df_train['Closing Price'].values

X_test = df_test['CP_LAGGED'].values

#Add a placeholder for the constant
X_test = sm.add_constant(X_test)

y_test = df_test['Closing Price'].values
```

Construct and fit the OLS (**O**rdinary **L**east **S**quares) regression model to the time series data set:

```
ols_model = sm.OLS(y_train,X_train)

ols_results = ols_model.fit()
```

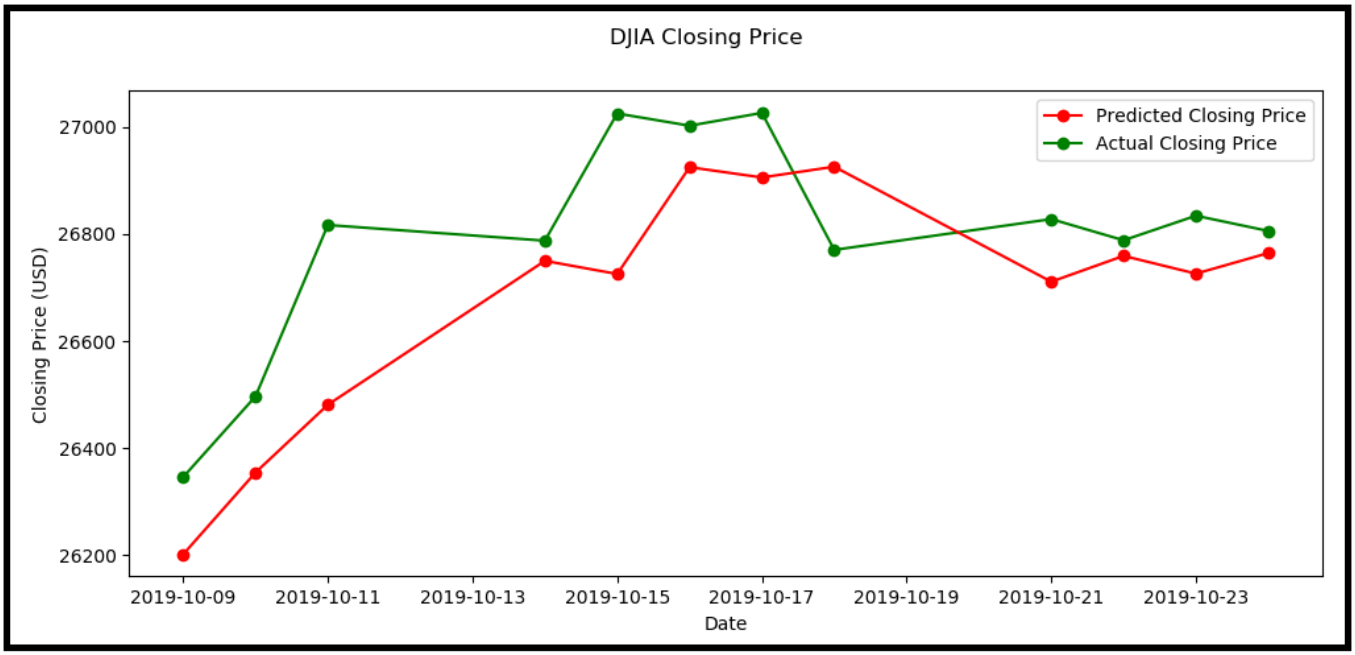
Use the fitted model to make predictions on the training and testing data sets:

```
y_pred_train = ols_results.predict(X_train)

y_pred_test = ols_results.predict(X_test)
```

Plot the model's performance against the *test data set*:

```
fig = plt.figure()

fig.suptitle('DJIA Closing Price')

actual, = plt.plot(df_test.index, y_test, 'go-', label='Actual
Closing Price')

predicted, = plt.plot(df_test.index, y_pred_test, 'ro-',
label='Predicted Closing Price')

plt.xlabel('Date')

plt.ylabel('Closing Price (USD)')

plt.legend(handles=[predicted, actual])

plt.show()
```

The results look like this:



Predicted versus actual Closing Price of DJIA using the OLS regression model on the test data set

At first glance, this model's performance looks much better than what we got from the mean model. But closer inspection reveals that at each time step, the model has simply learned to predict what is essentially the previously observed value offset by a certain amount.

But still, this lagged variable model *may* be statistically better performing than the intercept-only model in explaining the amount of variance in Closing Price. We will use the F-test to determine if this is true.

. . .

The testing approach

Our testing approach is going to be as follows:

We start with two hypotheses:

- **H_0: The Null hypothesis:** The lagged-variable model does not explain the variance in the DJIA Closing Price any better than the intercept only model.
- **H_1: The alternate hypothesis:** The lagged-variable model does a better job (in a statistically significant way) of explaining the variance in the DJIA Closing Price than the intercept only model.

We will use the F-test on the two models: the intercept-only model and the lagged variable model to determine if:

- The null hypothesis can be rejected (and the alternate hypothesis accepted) within some margin of error, OR
- The null hypothesis should be accepted.

. . .

A step-by-step procedure for using the F-test

To accomplish the above goals, we will follow these steps:

1. Formulate the test statistic for the F-test a.k.a. the **F-statistic**.
2. Identify the **P**robability **D**ensity **F**unction of the random variable that the F-statistic represents *under the assumption that the null hypothesis is true*.
3. Plug in the values into the formula for the F-statistic and calculate the corresponding probability value using the **P**robability **D**ensity **F**unction found in step 2. This is the probability of observing the F-statistic value *assuming that the null hypothesis is true*.
4. If the probability found in step 3 is less than the error threshold such as 0.05, reject the null hypothesis and accept the alternate hypothesis at a confidence level of (1.0 — error threshold), for e.g. 1–0.05 = 0.95 (i.e. 95% confidence level). Otherwise, accept the null hypothesis with a probability of error equal to the threshold error, for e.g. at 0.05 or 5%.

Let's dive into these steps.

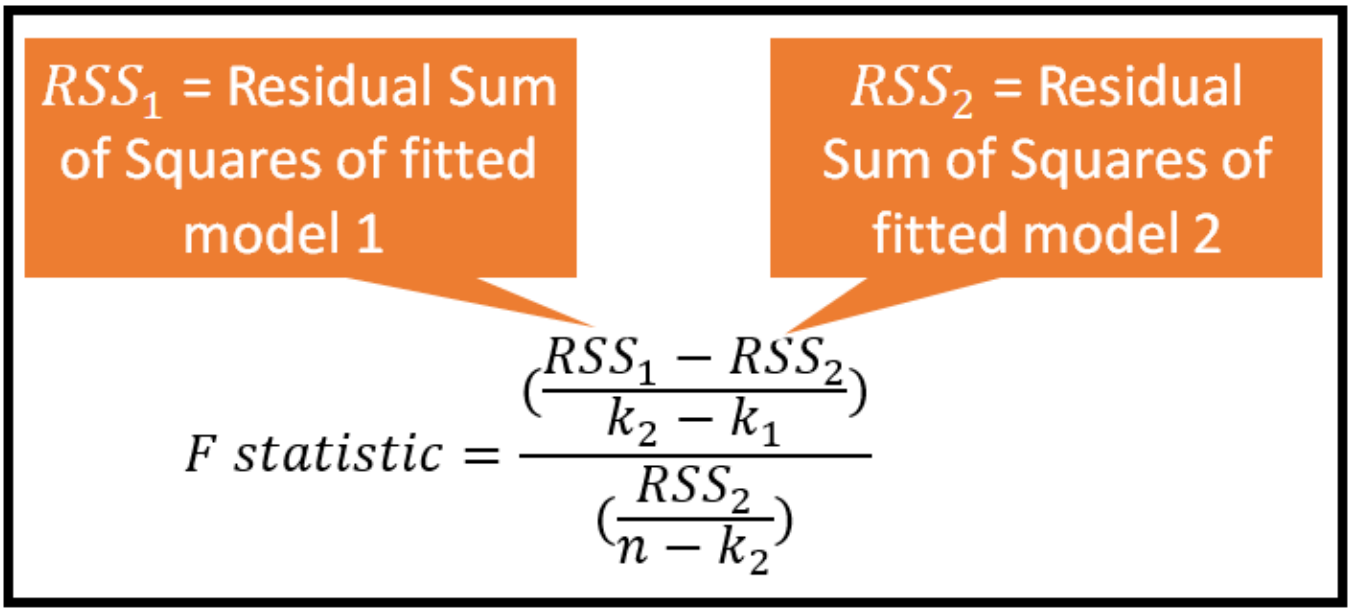
STEP 1: Developing the intuition for the test statistic

Recollect that the F-test measures how much better a complex model is as compared to a simpler version of the same model in its ability to explain the variance in the dependent variable.

Consider two regression models 1 and 2:

- Let Model 1 has k_1 parameters. Model 2 has k_2 parameters.
- Let $k\_1 < k\_2$
- Thus, Model 1 is the simpler version of model 2. i.e. model 1 is the restricted model and model 2 is the unrestricted model. Model 1 can be nested within model 2.
- Let RSS_1 and RSS_2 be the sum of squares of residual errors after Model 1 and Model 2 are fitted to the same data set.
- Let n be the number of data samples.

With the above definitions in place, the test statistic of the F-test for regression can be expressed as a ratio as follows:



Formula for the F-statistic when applied to regression analysis

The F-statistic formula lets you calculate how much of the variance in the dependent variable, the simpler model is *not* able to explain as compared to the complex model, expressed as a fraction of the unexplained variance from the complex model.

In regression analysis, the mean squared error of the fitted model is an excellent measure of unexplained variance. Which explains the RSS terms in the numerator and the denominator.

The numerator and the denominator are suitably scaled using the corresponding available degrees of freedom.

The F-statistic is itself a random variable.

Let's determine which **P**robability **D**ensity **F**unction the F-statistic obeys.

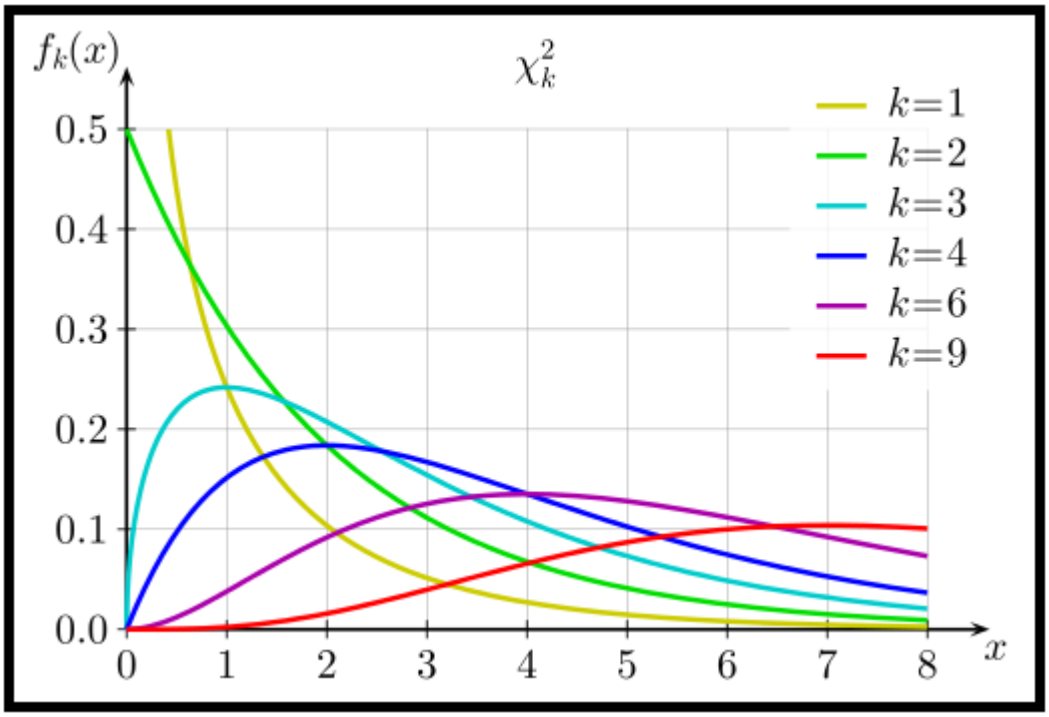
STEP 2: Identifying the Probability Density Function of the F-statistic

Notice that both the numerator and denominator of the test statistic contain sums of squares of residual errors. Also recollect that in regression, a residual error happens to be a random variable with some probability density (or probability mass) function, i.e. a

PDF or PMF depending on whether it is continuous or discrete. In this case we are concerned with finding the PDF of the F-statistic.

If we assume that the residual errors from the two models are 1) independent and 2) normally distributed, which incidentally happen to be requirements of **O**rdinary **L**east **S**quares regression, then it can be seen that the numerator and denominator of the F-statistic formula contain sums of squares of independent, normally distributed random variables.

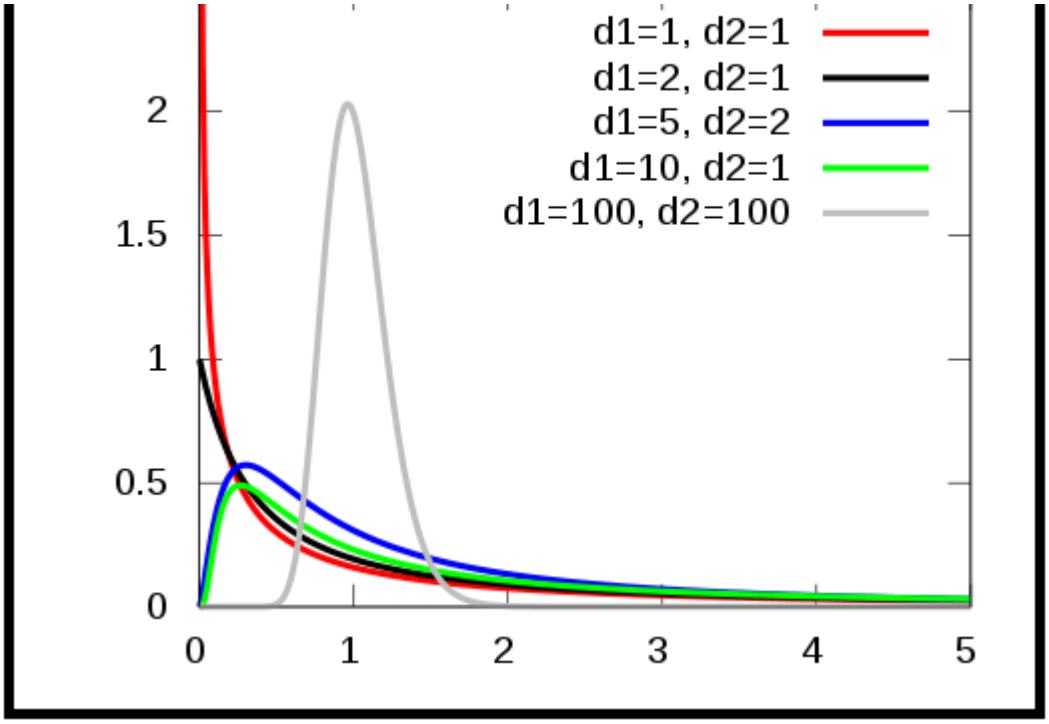
It can be proved that the sum of squares of k independent, standard normal random variables follow the PDF of the Chi-squared(k) distribution.



PDF of the Chi-Squared distribution (Source: Wikipedia CC BY 3.0)

Thus the numerator and denominator of the F-statistic formula can be shown to each obey scaled versions of two chi-squared distributions.

With a little bit of math, it can also be shown that the ratio of two suitably scaled Chi-squared distributed random variables is itself a random variable that follows the **F-distribution**, whose PDF is shown below.



PDF of the F-distribution (Source: Wikipedia CC BY SA 4.0)

In other words:

If the random variable ***X*** has the PDF of the F-distribution with parameters *d_1* and *d_2*, i.e. :

$$If\ X \sim F(d_1, d_2)$$

then, ***X*** can be shown to be expressed as the ratio of two suitably scaled random variables ***X_1*** and ***X_2***, each of which has the PDF of a Chi-squared distribution. i.e. :

$$X = \frac{X_1/d_1}{X_2/d_2}$$
$$Where:$$
$$X_1 \sim \chi^2_{d_1} \; and \; X_2 \sim \chi^2_{d_2}$$

An F-distributed random variable X, expressed as the ratio of two scaled Chi-squared distributed random variables X_1 and X_2

Now recollect that k_1 and k_2 are the number of variables in the simple and complex models M1 and M2 introduced earlier, and *n* is the number of data samples.

Substitute d_1 and d_2 as follows:

d_1 = (k_2 — k_1) which is the difference in degrees of freedom of the residuals of the two models M1 and M2 to be compared, and

d_2 = (n — k_2) which is the degrees of freedom of the residuals of the complex model M2,

With these substitutions, we can rewrite the F-distribution's formula as follows:

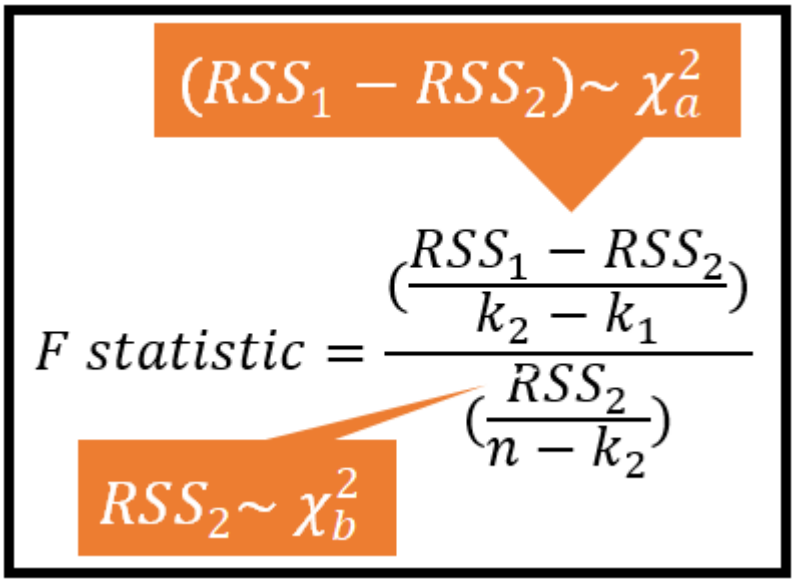
$$X = \frac{\frac{X_1}{(k_2 - k_1)}}{\frac{X_2}{(n - k_2)}}$$

$$Where:$$

$$X_1 \sim \chi^2_{(k_2 - k_1)} and\ X_2 \sim \chi^2_{(n - k_2)}$$

Alternate formula for the F-distribution's PDF

Let's compare the above formula with the formula for the F-statistic (reproduced below), where we know that the numerator and denominator contain suitably scaled PDFs of Chi-squared distributions:



Formula for the F-test's test statistic

Comparing these two formulae, it is clear that:

1. The degree of freedom '*a*' of the Chi-squared distribution in the numerator is (k1 — k2).
2. The degree of freedom '*b*' of the Chi-squared distribution in the denominator is (n — k2).
3. The test statistic of the F-test has the same PDF as that of the F-distribution.

In other words, the F-statistic follows the F-distribution.

STEP 3: Calculating the value of the F-statistic

If you use *statsmodels*'s OLS estimator, this step is a one-line operation. All you need to do is print *OLSResults.summary()* and you will get:

1. The value of the F-statistic and,
2. The corresponding 'p' value, i.e. the probability of encountering this value, from the F-distribution's PDF.

The *statsmodels* library will do the grunt work of both computations.

```
print(ols_results.summary())
```

This prints the following:

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.728
Model:                            OLS   Adj. R-squared:                  0.723
Method:                 Least Squares   F-statistic:                     136.7
Date:                Sat, 26 Oct 2019   Prob (F-statistic):           4.84e-16
Time:                        19:47:09   Log-Likelihood:                -370.13
No. Observations:                  53   AIC:                             744.3
Df Residuals:                      51   BIC:                             748.2
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       4233.3797   1905.887      2.221      0.031     407.153    8059.607
x1             0.8396      0.072     11.690      0.000       0.695       0.984
==============================================================================
Omnibus:                       17.469   Durbin-Watson:                   2.129
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               20.971
Skew:                          -1.320   Prob(JB):                     2.79e-05
Kurtosis:                       4.589   Cond. No.                     1.38e+06
==============================================================================
```

Output of OLSResults.summary()

STEP 4: Determining if the null hypothesis can be accepted

Since *OLSResults.summary()* prints out the probability of occurrence of the F-statistic under the assumption that the null hypothesis is true, we only need to compare this probability with our threshold alpha value. In our example, the p value returned by *.summary()* is 4.84E-16 which is an exceedingly small number. Much smaller than even alpha = 0.01. Thus, there is much less than 1% chance that the F-statistic of 136.7 could have occurred by chance under the assumption of a valid Null hypothesis.

Thus we reject the Null hypothesis and accept the alternate hypothesis H_1 that the complex model, i.e. the lagged variable model, in spite of its obvious flaws, is able to explain the variance in the dependent variable Closing Price better than the intercept-only model.

Here is the complete Python source code shown in this article:

```
1   import pandas as pd
```

```
import numpy as np
import matplotlib.pyplot as plt

#Create a pandas DataFrame for the djia data set.
df = pd.read_csv('djia.csv', header=0, infer_datetime_format=True, parse_dates=[0], index_col=[0

#################################
######## THE MEAN MODEL ########
#################################

mean = round(df['Closing Price'].mean(),2)
y_pred = np.full(len(df['Closing Price']), mean)

fig = plt.figure()
fig.suptitle('DJIA Closing Price')
actual, = plt.plot(df.index, df['Closing Price'], 'go-', label='Actual Closing Price')
predicted, = plt.plot(df.index, y_pred, 'ro-', label='Predicted Closing Price')
plt.xlabel('Date')
plt.ylabel('Closing Price (USD)')
plt.legend(handles=[predicted, actual])
plt.show()

##########################################
######## THE LAGGED VARIABLE MODEL ########
##########################################

import pandas as pd
import numpy as np
import statsmodels.api as sm

#Read the data set into a Pandas Data Frame
df = pd.read_csv('djia.csv', header=0, infer_datetime_format=True, parse_dates=[0], index_col=[0

#Add the lagged column
df['CP_LAGGED'] = df['Closing Price'].shift(1)

#Let's remove the first row as it contains an NaN:
df_lagged = df.drop(df.index[0])

split_index = round(len(df_lagged)*0.8)
split_date = df_lagged.index[split_index]
df_train = df_lagged.loc[df_lagged.index <= split_date].copy()
df_test = df_lagged.loc[df_lagged.index > split_date].copy()
X_train = df_train['CP_LAGGED'].values
#Add a placeholder for the constant so that model computes an intercept value i.e. the regressio
```

```
#Add a placeholder for the constant so that model computes an intercept value i.e. the regressio
X_train = sm.add_constant(X_train)
y_train = df_train['Closing Price'].values
X_test = df_test['CP_LAGGED'].values
#Add a placeholder for the constant so that model computes an intercept value
X_test = sm.add_constant(X_test)
y_test = df_test['Closing Price'].values


#Construct and fit the OLS regression model:
ols_model = sm.OLS(y_train,X_train)
ols_results = ols_model.fit()

#Use the fitted model to make predictions on the training and testing data sets:
y_pred_train = ols_results.predict(X_train)
y_pred_test = ols_results.predict(X_test)

#Use the fitted model to make predictions on the training and testing data sets:
y_pred_train = ols_results.predict(X_train)
y_pred_test = ols_results.predict(X_test)

#Plot the model's performance against the test data set:
fig = plt.figure()
fig.suptitle('DJIA Closing Price')
actual, = plt.plot(df_test.index, y_test, 'go-', label='Actual Closing Price')
predicted, = plt.plot(df_test.index, y_pred_test, 'ro-', label='Predicted Closing Price')
plt.xlabel('Date')
plt.ylabel('Closing Price (USD)')
plt.legend(handles=[predicted, actual])
plt.show()

#print the summary of regression results
print(ols_results.summary())
```

f_test.py hosted with ♥ by GitHub view raw

The data file containing the DJIA closing prices is over here.

. . .

Conclusion

- The F-test can be used in regression analysis to determine whether a complex model is better than a simpler version of the same model in explaining the variance in the dependent variable.
- The test statistic of the F-test is a random variable whose **P**robability **D**ensity **F**unction is the F-distribution *under the assumption that the null hypothesis is true.*
- The testing procedure for the F-test for regression is identical in its structure to that of other parametric tests of significance such as the t-test.

. . .

Thanks for reading! I write about topics in data science, with a focus on time series analysis and forecasting.

If you liked this article, please follow me at Sachin Date to receive tips, how-tos and programming advice on topics devoted to time series analysis and forecasting.

Data Science | Regression | Machine Learning | Programming | Statistics