How to (Properly) Include p-Values With sklearn

Import the Relevant Libraries

In [10]:
```
# We Will Need NumPy, pandas, matplotlib, and seaborn
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

# We Will Also Need the Actual Regression (Machine Learning) Module
from sklearn import linear_model
```

Load the Data

In [11]:
```
data = pd.read_csv('MultipleRegression.csv')

# Let's Explore the Top 5 Rows of the Data Frame
data.head()
```

Out[11]:

	SAT	Rand 1,2,3	GPA
0	1714	1	2.40
1	1664	3	2.52
2	1760	3	2.54
3	1685	3	2.74
4	1693	2	2.83

In [12]:
```
# This Method Provides Very Nice Descriptive Statistics
data.describe()
```

Out[12]:

	SAT	Rand 1,2,3	GPA
count	84.000000	84.000000	84.000000
mean	1845.273810	2.059524	3.330238
std	104.530661	0.855192	0.271617
min	1634.000000	1.000000	2.400000
25%	1772.000000	1.000000	3.190000
50%	1846.000000	2.000000	3.380000
75%	1934.000000	3.000000	3.502500

	SAT	Rand 1,2,3	GPA
max	2050.000000	3.000000	3.810000

Create the First Multiple Regression

Declare the Dependent and Independent Variables

In [19]:
```
# There Are Two Independent Variables: 'SAT' and 'Rand 1,2,3'
x = data[['SAT','Rand 1,2,3']]

# There Is a Single Dependedent Variable: 'GPA'
y = data['GPA']
```

How to (Properly) Include p-Values In sklearn

In [20]:
```
# Since the p-Values Are Obtained Through Certain Statistics, We Need the 'stat' Module
import scipy.stats as stat

# Since We Are Using an Object-Oriented Language Such as Python, We Can Simply Define O
# By Typing the Code Below, We Ovewrite a Part of the Class With One That Includes p-Va

class LinearRegression(linear_model.LinearRegression):
    """
    LinearRegression class after sklearn's, but calculate t-statistics
    and p-values for model coefficients (betas).
    Additional attributes available after .fit()
    are `t` and `p` which are of the shape (y.shape[1], X.shape[1])
    which is (n_features, n_coefs)
    This class sets the intercept to 0 by default, since usually we include it
    in X.
    """

    # Nothing Changes in __init__
    def __init__(self, fit_intercept = True, normalize = False, copy_X = True,
                 n_jobs = 1):
        self.fit_intercept = fit_intercept
        self.normalize = normalize
        self.copy_X = copy_X
        self.n_jobs = n_jobs


    def fit(self, X, y, n_jobs = 1):
        self = super(LinearRegression, self).fit(X, y, n_jobs)

        # Calculate SSE (Sum of Squared Errors) and SE (Standard Error)
        sse = np.sum((self.predict(X) - y) ** 2, axis = 0) / float(X.shape[0] - X.shape
        se = np.array([np.sqrt(np.diagonal(sse * np.linalg.inv(np.dot(X.T, X))))])

        # Compute the t-Statistic For Each Feature
        self.t = self.coef_ / se

        # Find the p-Value For Each Feature
        self.p = np.squeeze(2 * (1 - stat.t.cdf(np.abs(self.t), y.shape[0] - X.shape[1]
        return self
```

In [21]:
```
# When We Create the Regression, Everything Is the Same
reg_with_pvalues = LinearRegression()
reg_with_pvalues.fit(x, y)
```

Out[21]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

In [16]:
```
# The Difference Is That We Can Check What's Contained In the Local Variable 'p' In an
reg_with_pvalues.p
```

Out[16]:
```
array([0.        , 0.75717067])
```

In [17]:
```
# Let's Create a New Data Frame With the Names of the Features
reg_summary = pd.DataFrame([['SAT'], ['Rand 1,2,3']], columns = ['Features'])

# Then, We Create and Fill a Second Column Called 'Coefficients' With the Coefficients
reg_summary['Coefficients'] = reg_with_pvalues.coef_

# Finally, We Add the p-Values We Just Calculated
reg_summary['p-values'] = reg_with_pvalues.p.round(3)
```

In [18]:
```
# This Result Is Identical to the One From StatsModels
reg_summary
```

Out[18]:

	Features	Coefficients	p-values
0	SAT	0.001654	0.000
1	Rand 1,2,3	-0.008270	0.757