Multiple Linear Regression

libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

# regression module
from sklearn import linear_model
```

Load the data

In [2]:
```
data = pd.read_csv('Multiple_linear_regression.csv')

# Let's explore the top 5 rows of the df
data.head()
```

Out[2]:

	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 [3]:
```
# descriptive statistics
data.describe()
```

Out[3]:

	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
max	2050.000000	3.000000	3.810000

multiple regression

Declare the dependent and independent variables

In [4]:
```
# There are two independent variables: 'SAT' and 'Rand 1,2,3'
x = data[['SAT','Rand 1,2,3']]

# and a single depended variable: 'GPA'
y = data['GPA']
```

p-values in sklearn

In [5]:
```
# 'stat' module from scipy.stats
import scipy.stats as stat

# Here's the full source code of the ORIGINAL class: https://github.com/scikit-learn/scikit-learn/blob/7b136e9/


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[1])
        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 [6]:
```
# create the regression everything is the same
reg_with_pvalues = LinearRegression()
reg_with_pvalues.fit(x,y)
```

Out[6]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

In [7]:
```
# The difference is that we can check what's contained in the local variable 'p' in an instance of the LinearRe
reg_with_pvalues.p
```

Out[7]:
```
array([0.        , 0.75717067])
```

In [8]:
```
# 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 of the regression
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 [9]:
```
# This result is identical to the one from StatsModels
reg_summary
```

Out[9]:

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