Making Predictions with the Standardized Coefficients

Libraries

In [1]:
```python
# For these lessons 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()

# and of course the actual regression (machine learning) module
from sklearn.linear_model import LinearRegression
```

Load the data

In [2]:
```python
# Load the data from a .csv in the same folder
data = pd.read_csv('1.02. 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]:
```python
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

Create the multiple linear regression

Declare the dependent and independent variables

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

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

Standardization

In [5]:
```python
# Import the preprocessing module
from sklearn.preprocessing import StandardScaler
```

In [6]:
```python
# Create an instance of the StandardScaler class
scaler = StandardScaler()
```

In [7]:
```python
scaler.fit(x)
```

Out[7]:
```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

In [8]:
```python
# The actual scaling of the data is done through the method 'transform()'
# Let's store it in a new variable, named appropriately
x_scaled = scaler.transform(x)
```

In [9]:
```python
# The result is an ndarray
x_scaled
```

Out[9]:
```
array([[-1.26338288, -1.24637147],
       [-1.74458431,  1.10632974],
       [-0.82067757,  1.10632974],
       [-1.54247971,  1.10632974],
       [-1.46548748, -0.07002087],
       [-1.68684014, -1.24637147],
       [-0.78218146, -0.07002087],
       [-0.78218146, -1.24637147],
       [-0.51270866, -0.07002087],
       [ 0.04548499,  1.10632974],
       [-1.06127829,  1.10632974],
       [-0.67631715, -0.07002087],
       [-1.06127829, -1.24637147],
       [-1.28263094,  1.10632974],
       [-0.6955652 , -0.07002087],
       [ 0.25721362, -0.07002087],
       [-0.86879772,  1.10632974],
       [-1.64834403, -0.07002087],
       [-0.03150724,  1.10632974],
       [-0.57045283,  1.10632974],
       [-0.81105355,  1.10632974],
       [-1.18639066,  1.10632974],
       [-1.75420834,  1.10632974],
       [-1.52323165, -1.24637147],
       [ 1.23886453, -1.24637147],
       [-0.18549169, -1.24637147],
       [-0.5608288 , -1.24637147],
       [-0.23361183,  1.10632974],
       [ 1.68156984, -1.24637147],
       [-0.4934606 , -0.07002087],
       [-0.73406132, -1.24637147],
       [ 0.85390339, -1.24637147],
       [-0.67631715, -1.24637147],
       [ 0.09360513,  1.10632974],
       [ 0.33420585, -0.07002087],
       [ 0.03586096, -0.07002087],
       [-0.35872421,  1.10632974],
       [ 1.04638396,  1.10632974],
       [-0.65706909,  1.10632974],
       [-0.13737155, -0.07002087],
       [ 0.18984542,  1.10632974],
       [ 0.04548499, -1.24637147],
       [ 1.1618723 ,  1.10632974],
       [-1.37887123, -1.24637147],
       [ 1.39284898, -1.24637147],
       [ 0.76728713, -0.07002087],
       [-0.20473975, -0.07002087],
       [ 1.06563201, -1.24637147],
       [ 0.11285319, -1.24637147],
       [ 1.28698467,  1.10632974],
       [-0.41646838,  1.10632974],
       [ 0.09360513, -1.24637147],
       [ 0.59405462, -0.07002087],
       [-2.03330517, -0.07002087],
       [ 0.32458182, -1.24637147],
       [ 0.40157405, -1.24637147],
       [-1.10939843, -0.07002087],
       [ 1.03675993, -1.24637147],
       [-0.61857297, -0.07002087],
       [ 0.44007016, -0.07002087],
       [ 1.14262424, -1.24637147],
       [-0.35872421,  1.10632974],
       [ 0.45931822,  1.10632974],
       [ 1.88367444,  1.10632974],
       [ 0.45931822, -1.24637147],
       [-0.12774752, -0.07002087],
       [ 0.04548499,  1.10632974],
       [ 0.85390339, -0.07002087],
       [ 0.15134931, -0.07002087],
       [ 0.8250313 ,  1.10632974],
       [ 0.84427936,  1.10632974],
       [-0.64744506, -1.24637147],
       [ 1.24848856, -1.24637147],
       [ 0.85390339,  1.10632974],
       [ 1.69119387,  1.10632974],
       [ 1.6334497 ,  1.10632974],
       [ 1.46021718, -1.24637147],
       [ 1.68156984, -0.07002087],
       [-0.02188321,  1.10632974],
       [ 0.87315144,  1.10632974],
       [-0.33947615, -1.24637147],
       [ 1.3639769 ,  1.10632974],
       [ 1.12337618, -1.24637147],
       [ 1.97029069, -0.07002087]])
```

Regression with scaled features

In [10]:
```python
# Creating a regression
reg = LinearRegression()

# inputs are the 'scaled inputs'
reg.fit(x_scaled,y)
```

Out[10]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

In [11]:
```python
# coefficients
reg.coef_
```

Out[11]:
```
array([ 0.17181389, -0.00703007])
```

In [12]:
```python
# intercept
reg.intercept_
```

Out[12]:
```
3.330238095238095
```

Creating a summary table

In [13]:
```python
# create a new data frame with the names of the features
reg_summary = pd.DataFrame([['Bias'],['SAT'],['Rand 1,2,3']], columns=['Features'])

# create and fill a second column, called 'Weights' with the coefficients of the regression
# Since the standardized coefficients are called 'weights' in ML, this is a much better word choice for our cas
reg_summary['Weights'] = reg.intercept_, reg.coef_[0], reg.coef_[1]
```

In [14]:
```python
reg_summary
```

Out[14]:

	Features	Weights
0	Bias	3.330238
1	SAT	0.171814
2	Rand 1,2,3	-0.007030

Making predictions with the standardized coefficients (weights)

In [15]:
```python
# a new dataframe with 2 *new* observations
new_data = pd.DataFrame(data=[[1700,2],[1800,1]],columns=['SAT','Rand 1,2,3'])
new_data
```

Out[15]:

	SAT	Rand 1,2,3
0	1700	2
1	1800	1

In [16]:
```python
# We can make a prediction for a whole dataframe (not a single value)
reg.predict(new_data)
```

Out[16]:
```
array([295.39979563, 312.58821497])
```

In [17]:
```python
# Our model is expecting SCALED features (features of different magnitude)
# In fact we must transform the 'new data' in the same way as we transformed the inputs we train the model on
# Luckily for us, this information is contained in the 'scaler' object
# We simply transform the 'new data' using the relevant method
new_data_scaled = scaler.transform(new_data)

# Let's check the result
new_data_scaled
```

Out[17]:
```
array([[-1.39811928, -0.07002087],
       [-0.43571643, -1.24637147]])
```

In [18]:
```python
# Finally we make a prediction using the scaled new data
reg.predict(new_data_scaled)
```

Out[18]:
```
array([3.09051403, 3.26413803])
```

What if we removed the 'Random 1,2,3' variable?

In [20]:
```python
# Theory suggests that features with very small weights could be removed and the results should be identical
# Moreover, we proved in 2-3 different ways that 'Rand 1,2,3' is an irrelevant feature
# Let's create a simple linear regression (simple, because there is a single feature) without 'Rand 1,2,3'
reg_simple = LinearRegression()

# Once more, we must reshape the inputs into a matrix, otherwise we will get a compatibility error
# Note that instead of standardizing again, I'll simply take only the first column of x
x_simple_matrix = x_scaled[:,0].reshape(-1,1)

# Finally, we fit the regression
reg_simple.fit(x_simple_matrix,y)
```

Out[20]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

In [21]:
```python
# In a similar manner to the cell before, we can predict only the first column of the scaled 'new data'
# Note that we also reshape it to be exactly the same as x
reg_simple.predict(new_data_scaled[:,0].reshape(-1,1))
```

Out[21]:
```
array([3.08970998, 3.25527879])
```