Simple Linear Regression

Import the Relevant Libraries

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

from sklearn.linear_model import LinearRegression
```

Load the Data

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

Out[2]:

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

Create the Regression

Declare the Dependent and Independent Variables

In [3]:
```
x = data['SAT']
y = data['GPA']
```

In [4]:
```
x.shape
```

Out[4]: (84,)

In [5]:
```
y.shape
```

Out[5]: (84,)

The Regression Itself

In [6]:
```
reg = LinearRegression()
```

In [7]:
```
reg.fit(x, y)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-7-98bf9091ae0e> in <module>
----> 1 reg.fit(x, y)

~\anaconda3\lib\site-packages\sklearn\linear_model\_base.py in fit(self, X, y, sample_we
ight)
    516         accept_sparse = False if self.positive else ['csr', 'csc', 'coo']
    517
--> 518         X, y = self._validate_data(X, y, accept_sparse=accept_sparse,
    519                                    y_numeric=True, multi_output=True)
    520

~\anaconda3\lib\site-packages\sklearn\base.py in _validate_data(self, X, y, reset, valid
ate_separately, **check_params)
    431                 y = check_array(y, **check_y_params)
    432             else:
--> 433                 X, y = check_X_y(X, y, **check_params)
    434             out = X, y
    435

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in inner_f(*args, **kwargs)
     61             extra_args = len(args) - len(all_args)
     62             if extra_args <= 0:
---> 63                 return f(*args, **kwargs)
     64
     65             # extra_args > 0

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in check_X_y(X, y, accept_spar
se, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, mult
i_output, ensure_min_samples, ensure_min_features, y_numeric, estimator)
    812         raise ValueError("y cannot be None")
    813
--> 814     X = check_array(X, accept_sparse=accept_sparse,
    815                     accept_large_sparse=accept_large_sparse,
    816                     dtype=dtype, order=order, copy=copy,

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in inner_f(*args, **kwargs)
     61             extra_args = len(args) - len(all_args)
     62             if extra_args <= 0:
---> 63                 return f(*args, **kwargs)
     64
     65             # extra_args > 0

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in check_array(array, accept_s
parse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, e
nsure_min_samples, ensure_min_features, estimator)
    635             # If input is 1D raise error
    636             if array.ndim == 1:
--> 637                 raise ValueError(
    638                     "Expected 2D array, got 1D array instead:\narray={}.\n"
    639                     "Reshape your data either using array.reshape(-1, 1) if "

ValueError: Expected 2D array, got 1D array instead:
array=[1714 1664 1760 1685 1693 1670 1764 1764 1792 1850 1735 1775 1735 1712
 1773 1872 1755 1674 1842 1786 1761 1722 1663 1687 1974 1826 1787 1821
 2020 1794 1769 1934 1775 1855 1880 1849 1808 1954 1777 1831 1865 1850
```

```
 1966 1702 1990 1925 1824 1956 1857 1979 1802 1855 1907 1634 1879 1887
 1730 1953 1781 1891 1964 1808 1893 2041 1893 1832 1850 1934 1861 1931
 1933 1778 1975 1934 2021 2015 1997 2020 1843 1936 1810 1987 1962 2050].
Reshape your data either using array.reshape(-1, 1) if your data has a single feature or
array.reshape(1, -1) if it contains a single sample.
```

In [8]:
```
x_matrix = x.values.reshape(-1, 1)
x_matrix.shape
```

Out[8]:
```
(84, 1)
```

In [9]:
```
reg.fit(x_matrix, y)
```

Out[9]:
```
LinearRegression()
```

R-Squared

In [10]:
```
reg.score(x_matrix, y)
```

Out[10]:
```
0.40600391479679765
```

Coefficients

In [11]:
```
reg.coef_
```

Out[11]:
```
array([0.00165569])
```

Intercept

In [12]:
```
reg.intercept_
```

Out[12]:
```
0.2750402996602803
```

Making Predictions

In [13]:
```
new_data = pd.DataFrame(data = [1740, 1760], columns = ['SAT'])
new_data
```

Out[13]:

	SAT
0	1740
1	1760

In [14]:
```
reg.predict(new_data)
```

Out[14]:
```
array([3.15593751, 3.18905127])
```

In [15]:

```
new_data['Predicted_GPA'] = reg.predict(new_data)
new_data
```

Out[15]:

	SAT	Predicted_GPA
0	1740	3.155938
1	1760	3.189051

In [16]:

```
plt.scatter(x, y)
yhat = reg.coef_ * x_matrix + reg.intercept_
# yhat = 0.0017 * x + 0.275
fig = plt.plot(x, yhat, lw = 4, c = 'orange', label = 'regression line')
plt.xlabel('SAT', fontsize = 20)
plt.ylabel('GPA', fontsize = 20)
plt.show()
```