Практический пример на простую линейную регрессию

Для прогнозирования средних январских температур в будущем и средних январских температур до 1895 года воспользуемся методом простой линейной регрессии и данными средней январской температуры в Нью-Йорке с 1895 по 2018 год.

При помощи метода, называемого простой линейной регрессией, построим прогнозы, выявляющие линейные отношения между месяцами (январь каждого года) и средней температурой в Нью-Йорке. Для заданной коллекции значений, представляющих независимую переменную (комбинация «месяц/год») и зависимую переменную (средняя температура за этот месяц/год), простая линейная регрессия описывает отношение между этими переменными прямой линией — регрессионной прямой.

Данные мы будем брать с сайта: https://www.ncdc.noaa.gov/cag/ (https://www.ncdc.noaa.gov/cag/)

In [9]:
```
nyc.head()
```

Out[9]:

	Date	Value	Anomaly
0	189501	34.2	-3.2
1	189601	34.7	-2.7
2	189701	35.5	-1.9
3	189801	39.6	2.2
4	189901	36.4	-1.0

- Date — значение в форме 'YYYYMM' (например, '201801'). Часть MM всегда содержит 01, потому что мы загружали данные только за январь каждого года.
- Value — температура по Фаренгейту в формате с плавающей точкой.
- Anomaly — разность между значением для заданной даты и средними значениями для всех дат. В нашем примере значение Anomaly не используется, поэтому мы его проигнорируем.

Скоро мы воспользуемся Seaborn для графического представления пар DateValue и регрессионной прямой. При отображении данных из DataFrame Seaborn помечает оси графика именами столбцов DataFrame. Для удобочитаемости переименуем столбец 'Value' в 'Temperature':

In [10]:
```
nyc.columns = ['Date', 'Temperature', 'Anomaly']
```

Seaborn помечает деления на оси x значениями Date. Поскольку в примере обрабатываются только январские данные, метки оси x будут лучше читаться без обозначения 01 (для января); удалим его из Date. Сначала проверим тип столбца:

In [11]:
```
nyc.Date.dtype
```

Out[11]:
```
dtype('int64')
```

Значения являются целыми числами, поэтому мы можем разделить их на 100 для отсечения двух последних цифр. Вспомните, что каждый столбец в DataFrame представляет собой коллекцию Series. Вызов метода floordiv коллекции Series выполняет целочисленное деление с каждым элементом Series:

In [12]:
```
nyc.Date = nyc.Date.floordiv(100)
nyc.head(3)
```

Out[12]:

	Date	Temperature	Anomaly
0	1895	34.2	-3.2
1	1896	34.7	-2.7
2	1897	35.5	-1.9

Чтобы быстро получить некоторые статистики для температур из набора данных, вызовем describe для столбца Temperature. В коллекции присутствуют 124 наблюдения, среднее значение наблюдений равно 37.60, а наименьшее и наибольшее наблюдение равны 26.10 и 47.60 градусам соответственно:

In [13]:
```
pd.set_option('precision', 2)
```

In [14]:
```
nyc.Temperature.describe()
```

Out[14]:
```
count    124.00
mean      37.60
std        4.54
min       26.10
25%       34.58
50%       37.60
75%       40.60
max       47.60
Name: Temperature, dtype: float64
```

Библиотека SciPy (Scientific Python) широко применяется в инженерных, научных и математических вычислениях на языке Python. Ее модуль *stats* предоставляет функцию *linregress*, которая вычисляет *наклон* и *точку пересечения* регрессионной прямой для заданного набора точек данных:

In [15]:
```
from scipy import stats
linear_regression = stats.linregress(x=nyc.Date, y=nyc.Temperature)
```

Функция linregress получает два одномерных массива1 одинаковой длины, представляющих координаты x и y точек данных. Ключевые аргументы x и y представляют независимые и зависимые переменные соответственно. Объект, возвращаемый linregress, содержит угол наклона и точку пересечения регрессионной прямой:

In [16]:
```
linear_regression.slope
```

Out[16]:
```
0.014771361132966163
```

In [17]:
```
linear_regression.intercept
```

Out[17]:
```
8.694993233674289
```

Эти значения можно объединить с уравнением простой линейной регрессии для прямой линии, y = mx + b при прогнозировании средней январской температуры в Нью-Йорке для заданного года. Спрогнозируем среднюю температуру по Фаренгейту за январь 2019 года. В следующих вычислениях linear_regression.slope соответствует m, 2019 соответствует x (значение, для которого прогнозируется температура), а linear_regression.intercept соответствует b:

In [18]:
```
linear_regression.slope * 2019 + linear_regression.intercept
```

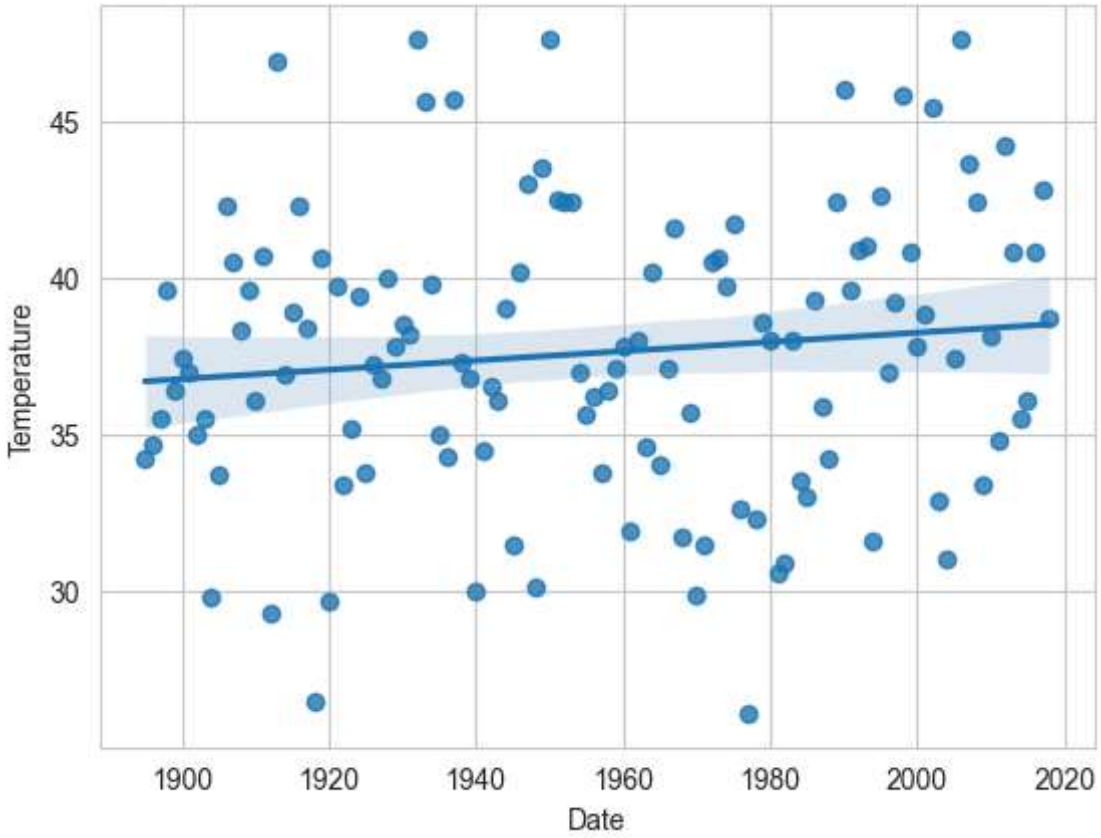
Out[18]:
```
38.51837136113297
```

Также по формуле можно приближенно оценить, какой могла быть средняя температура до 1895 года. Например, оценка средней температуры за январь 1890 года может быть получена следующим образом:

In [19]:
```
linear_regression.slope * 1890 + linear_regression.intercept
```

Out[19]:
```
36.612865774980335
```

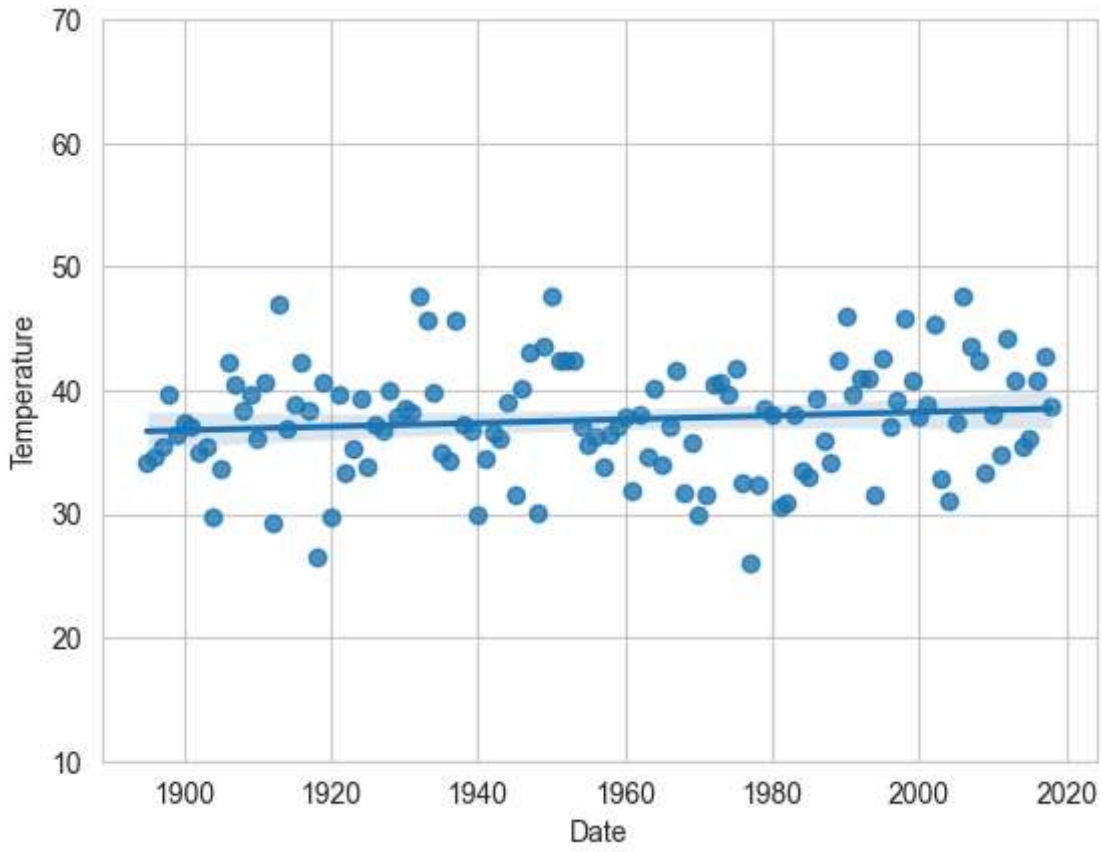
In [20]:
```
import seaborn as sns
sns.set_style('whitegrid')
axes = sns.regplot(x=nyc.Date, y=nyc.Temperature)
```



Наклон регрессионной прямой (подъем от левой части к правой) указывает на то, что в последние 124 года средняя температура повышалась. На графике ось y представляет температурный диапазон между минимумом 26.1 и максимумом 47.6 по Фаренгейту, в результате чего наблюдается значительный разброс данных вокруг регрессионной прямой, из-за которого труднее выявить линейную связь. Эта проблема типична для визуализаций в области аналитики данных. Если оси на графике отражают разные типы данных (даты и температуры в данном случае), то как разумно определить их относительные масштабы? На предыдущем графике все определяется исключительно высотой графика — Seaborn и Matplotlib автоматически масштабируют оси на основании диапазонов значений данных. Ось значений y можно масштабировать, чтобы подчеркнуть линейность отношения. В следующем примере ось y была масштабирована от 21,5-градусного диапазона до 60-градусного диапазона (от 10 до 70 градусов):

In [21]:
```
import seaborn as sns
sns.set_style('whitegrid')
axes = sns.regplot(x=nyc.Date, y=nyc.Temperature)
axes.set_ylim(10, 70)
```

Out[21]: (10.0, 70.0)



Assignment for class

Now, you should make it all for Los Angeles using ave_hi_la_jan_1895-2018.csv file and for Austin using ave_hi_austin_jan_1895-2018.csv file.

In [ ]: