Linear Regression

In 1991, Orley Ashenfelter, an economics professor at Princeton University, stunned the wine world with a bold prediction. He predicted that the 1990 vintage of Bordeaux wines would be the "wine of the century," even better than the prized 1961 vintage. Furthermore, he made this prediction without tasting even a drop of the wine, which had been placed in oak barrels just months earlier.

How did Ashenfelter predict the quality of the wine without tasting it? He used data on past vintages to come up with the following formula for predicting wine quality:
$$\begin{align} \widehat{\text{wine quality}} = -7.8 &+ 0.62 \cdot (\text{average summer temperature}) \nonumber \\ &+ 0.0012 \cdot (\text{winter rainfall}) \nonumber \\ &- 0.0037 \cdot (\text{harvest rainfall}) \nonumber \\ &+ 0.024 \cdot (\text{age of the wine}) \label{eq:ashenfelter} \end{align}$$

The variable on the left-hand side of this expression, wine quality, is what we are trying to predict and is called the *target* (or *label*). (The hat symbol over "wine quality" indicates that the values are predicted instead of observed.) The variables on the right-hand side, such as "average summer temperature" and "harvest rainfall," are called *features* and are the inputs used to predict the target. Although Ashenfelter had no way of knowing the quality of the 1990 wines, he did have the values of the features in 1990, so to make a prediction, all he had to do was plug those values into the equation above. In this way, he arrived at the following prediction for the quality of the 1990 Bordeaux, after they had been aged for 31 years (like the 1961 Bordeaux had been at the time):
$$\begin{align} -7.8 &+ 0.62 \cdot (18.7) \nonumber \\ &+ 0.0012 \cdot (468) \nonumber \\ &- 0.0037 \cdot (80) \nonumber \\ &+ 0.024 \cdot (31) = 4.8. \label{eq:ashenfelter_1990} \end{align}$$

For comparison, the quality of the prized 1961 vintage was 4.6.

You can imagine the uproar from wine experts, who had spent years refining their palates to distinguish good wines from bad. Robert Parker, the most influential wine critic in America, called Ashenfelter's predictions "ludicrous and absurd", comparing him to a "movie critic who never goes to see the movie but tells you how good it is based on the actors and the director." It did not help that Ashenfelter had also openly challenged Parker's rating of the 1986 Bordeaux. Parker thought they would be "very good and sometimes exceptional." But according to Ashenfelter's formula, the low summer temperatures and high harvest rainfalls in 1986 doomed the vintage.

Who was right? Thirty years later, Robert Parker ranks the 1986 Bordeaux well, but the 1990 Bordeaux wines are exceptional, with three of the six wines scoring a 98 on a 100-point scale.

We will reproduce Ashenfelter's analysis, which is an example of *machine learning*. Machine learning is concerned with the general problem of how to use data to make predictions. The process of producing a model like Ashenfelter's from data is called *fitting* a model (although the terms *training* or *learning* are also used), and the data that is used to fit the model is the *training data*.

Getting Familiar with the Data

First, we read in the historical data that Ashenfelter used. The observational unit in this data set is the vintage, so we index this `DataFrame` by the year.

In [1]:

```
import pandas as pd
data_dir = ""
bordeaux_df = pd.read_csv("bordeaux.csv",index_col="year")
bordeaux_df.head()
```

Out[1]:

year	price	summer	har	sep	win	age
1952	37.0	17.1	160	14.3	600	40
1953	63.0	16.7	80	17.3	690	39
1955	45.0	17.1	130	16.8	502	37
1957	22.0	16.1	110	16.2	420	35
1958	18.0	16.4	187	19.1	582	34

The **price** column is in 1981 dollars, normalized so that the 1961 Bordeaux has a price of 100. Price is a reasonable proxy for the quality of the wine. The **summer** column contains the average summer temperature (in degrees Celsius), while the **har** and **win** columns contain the harvest and winter rainfalls (in millimeters). The **sep** column stores the average temperature in September, which Ashenfelter did not include in his model.

Let us also take a peek at the end of this `DataFrame`.

In [2]:

```
bordeaux_df.tail()
```

Out[2]:

year	price	summer	har	sep	win	age
1987	NaN	17.0	115	18.9	452	5
1988	NaN	17.1	59	16.8	808	4
1989	NaN	18.6	82	18.4	443	3
1990	NaN	18.7	80	19.3	468	2
1991	NaN	17.7	183	20.4	570	1

We see that the `DataFrame` also contains data for vintages where the price is missing (including 1990, the vintage for which Ashenfelter made his prediction). In fact, prices are only available up to 1980, as it takes several years before wine quality can be estimated with much reliability), so only part of the `DataFrame` can be used for training. The rest of the data, where the features are known but the target is not, is called the *test data*. Machine learning fits a model to the training data, which is then used to predict the targets in the test data. The following code splits the `DataFrame` into the training and test sets.

In [3]:

```
bordeaux_train = bordeaux_df.loc[:1980].copy()
bordeaux_test = bordeaux_df.loc[1981:].copy()
```

Warm-Up: A Model with One Feature

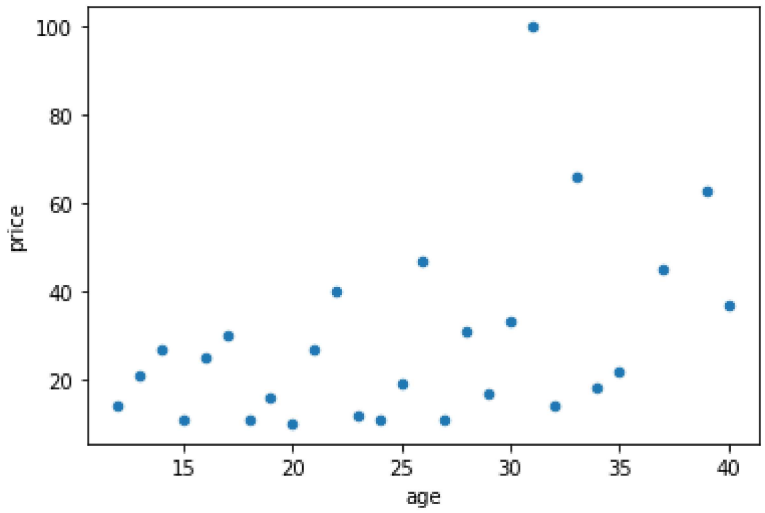
Before fitting a model that uses all of the features, we first consider a model that uses only the age of the wine to predict the price. That is, we fit a model of the form \begin{equation} \widehat{\text{price}} = b + c \cdot \text{age}, \label{eq:simple_linear_regression} \end{equation} where b and c are numbers that we will learn from the training data. Models of the form above are called *linear regression* models. (The way in which this model is "linear" will become apparent in a moment.) This model only involves two variables, **age** and **price**, so we can visualize the data easily using a scatterplot (see Chapter 3).

In [4]:

```
bordeaux_train.plot.scatter(x="age", y="price")
```

Out[4]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bdc3aafc48>
```



Now, to fit models like the above to the training data, we use the scikit-learn package, which was used in Chapter 3 for transforming variables and calculating distances. However, its main purpose is to fit machine learning models, including linear regression. All models in scikit-learn are used in essentially the same way, following the three-step pattern:

1. Declare the model.
2. Fit the model to training data.
3. Use the model to predict on test data.

In the case of the linear regression model above, the code is as follows.

In [5]:

```
from sklearn.linear_model import LinearRegression

X_train = bordeaux_train[["age"]]
X_test = bordeaux_test[["age"]]
y_train = bordeaux_train["price"]

model = LinearRegression()
model.fit(X=X_train, y=y_train)
model.predict(X=X_test)
```

Out[5]:

```
array([12.41648163, 11.26046336, 10.1044451 ,  8.94842683,  7.79240856,
        6.6363903 ,  5.48037203,  4.32435376,  3.1683355 ,  2.01231723,
        0.85629897])
```

The parameters of `.fit()` are `X` for the features and `y` for the targets, which are assumed to be 2-D and 1-D arrays of numbers, respectively. So even when there is only one feature, as in this case, we still need to supply a 2-D array with one column---hence, the double brackets around `"age"` when defining `X_train` and `X_test`.

By contrast, `.predict()` only has one parameter, `X` for the features. That is because its job is to predict the targets `y` for the given features. Note that the predictions will always be returned in the form of `numpy` arrays, no matter the type of the input data---so although we supplied `pandas` objects, `sklearn` still returned the predicted values as `numpy` arrays. The predictions are in the same order as the rows of `X`.

Because there are only two variables involved, the model above is a rare example of a machine learning model we can visualize. A general way to do this is to generate a fine grid of `X` values using `np.linspace()` and call `model.predict()` to get the predicted target at each of these values. We can then use these predictions to draw a curve which depicts the predicted value of `y` at each value of `X`. In the code below, we put the predictions in a `pandas` `Series`, indexed by the `X` values, and then call `.plot.line()`.

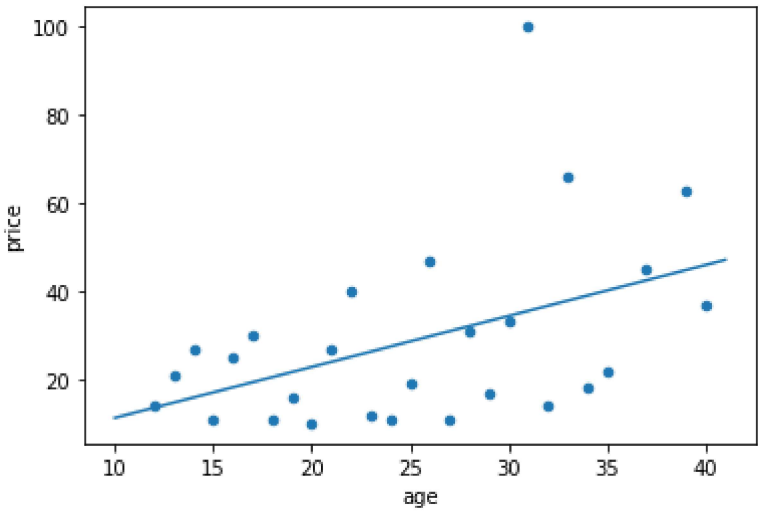
In [6]:

```
import numpy as np

X_new = pd.DataFrame()
# create a sequence of 200 evenly spaced numbers from 10 to 41
X_new["age"] = np.linspace(10, 41, num=200)

print (model.predict(X_new))
print ("***")
# create a Series out of the predicted values
# (trailing underscore indicates fitted values)
y_new_ = pd.Series(
    model.predict(X_new), # y values in Series.plot.line()
    index=X_new["age"]    # x values in Series.plot.line()
)

# plot the data, then the model
bordeaux_train.plot.scatter(x="age", y="price")
y_new_.plot.line()
```

```
[11.26046336 11.44054661 11.62062986 11.80071311 11.98079635 12.1608796
 12.34096285 12.5210461  12.70112934 12.88121259 13.06129584 13.24137909
 13.42146233 13.60154558 13.78162883 13.96171208 14.14179532 14.32187857
 14.50196182 14.68204507 14.86212831 15.04221156 15.22229481 15.40237806
 15.5824613  15.76254455 15.9426278  16.12271105 16.30279429 16.48287754
 16.66296079 16.84304404 17.02312728 17.20321053 17.38329378 17.56337703
 17.74346027 17.92354352 18.10362677 18.28371002 18.46379326 18.64387651
 18.82395976 19.00404301 19.18412625 19.3642095  19.54429275 19.724376
 19.90445924 20.08454249 20.26462574 20.44470899 20.62479223 20.80487548
 20.98495873 21.16504198 21.34512522 21.52520847 21.70529172 21.88537497
 22.06545821 22.24554146 22.42562471 22.60570796 22.7857912  22.96587445
 23.1459577  23.32604095 23.50612419 23.68620744 23.86629069 24.04637394
 24.22645718 24.40654043 24.58662368 24.76670693 24.94679017 25.12687342
 25.30695667 25.48703992 25.66712316 25.84720641 26.02728966 26.20737291
 26.38745615 26.5675394  26.74762265 26.9277059  27.10778914 27.28787239
 27.46795564 27.64803889 27.82812213 28.00820538 28.18828863 28.36837188
 28.54845512 28.72853837 28.90862162 29.08870487 29.26878812 29.44887136
 29.62895461 29.80903786 29.98912111 30.16920435 30.3492876  30.52937085
 30.7094541  30.88953734 31.06962059 31.24970384 31.42978709 31.60987033
 31.78995358 31.97003683 32.15012008 32.33020332 32.51028657 32.69036982
 32.87045307 33.05053631 33.23061956 33.41070281 33.59078606 33.7708693
 33.95095255 34.1310358  34.31111905 34.49120229 34.67128554 34.85136879
 35.03145204 35.21153528 35.39161853 35.57170178 35.75178503 35.93186827
 36.11195152 36.29203477 36.47211802 36.65220126 36.83228451 37.01236776
 37.19245101 37.37253425 37.5526175  37.73270075 37.912784   38.09286724
 38.27295049 38.45303374 38.63311699 38.81320023 38.99328348 39.17336673
 39.35344998 39.53353322 39.71361647 39.89369972 40.07378297 40.25386621
 40.43394946 40.61403271 40.79411596 40.9741992  41.15428245 41.3343657
 41.51444895 41.69453219 41.87461544 42.05469869 42.23478194 42.41486518
 42.59494843 42.77503168 42.95511493 43.13519817 43.31528142 43.49536467
 43.67544792 43.85553116 44.03561441 44.21569766 44.39578091 44.57586415
 44.7559474  44.93603065 45.1161139  45.29619714 45.47628039 45.65636364
 45.83644689 46.01653013 46.19661338 46.37669663 46.55677988 46.73686312
 46.91694637 47.09702962]
***
```

Out[6]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bdc63f8e48>
```



The resulting plot is shown above. Notice that the curve is a straight line, which is why this model is called *linear* regression. In hindsight, this is obvious from the model equation: b is simply the intercept and c the slope of this line. All linear regression does is choose the intercept and slope to minimize the total squared distance between the points and the line---that is, between the observed and predicted prices. In mathematical terms, b and c are chosen to minimize \begin{align} & \text{sum of } (\text{price} - \widehat{\text{price}})^2 &= & & \text{sum of } (\text{price} - (b + c\cdot \text{age}))^2 \\ & \text{over training data } & & & \text{ over training data}. \end{align} Since `sklearn` does this optimization for us, it is not necessary to understand the details of this process to extract useful insights out of linear regression. However, the math is explained in the appendix of this lesson for those who are curious.

What to Do about Nonlinearity

One question is whether the relationship between age and price is truly linear. In the graph above, it seems that the points deviate more from the line when prices are high than when they are low. To correct this, we need to spread out low prices and rein in high prices. Previously, we learned that this can be achieved by applying a log transformation to the prices. Let's add a column to the training data for the log-price.

In [7]:

```
bordeaux_train["log(price)"] = np.log(bordeaux_train["price"])
```

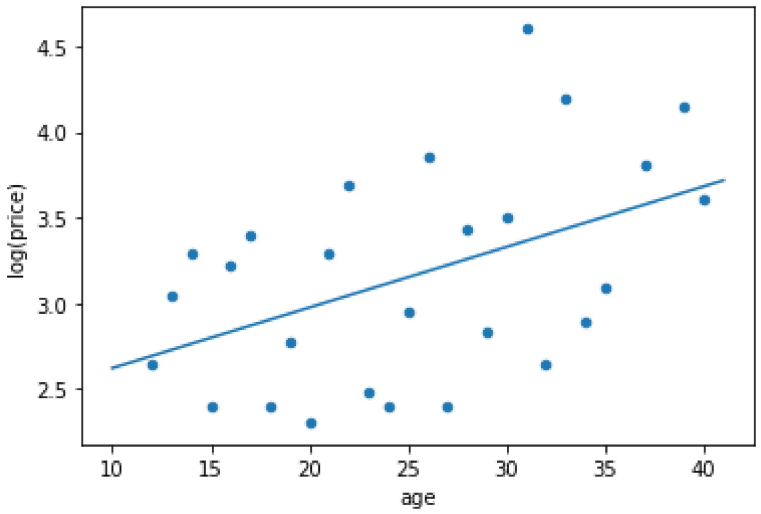
Now, we will fit a linear regression model to predict this new target. That is, in contrast to the previous model, we now fit the model \begin{equation} \widehat{\text{log(price)}} = b + c \cdot \text{age}, \label{eq:simple_linear_regression_log} \end{equation} where b and c are chosen to minimize \begin{equation} \text{sum of } (\text{log(price)} - \widehat{\text{log(price)}})^2 \text{ over training data} \end{equation} over the training data. The code below fits this model.

In [8]:

```
log_price_model = LinearRegression()
log_price_model.fit(X=bordeaux_train[["age"]],
                    y=bordeaux_train["log(price)"])

X_new = pd.DataFrame()
X_new["age"] = np.linspace(10, 41, num=200)
y_new_ = pd.Series(
    log_price_model.predict(X_new),
    index=X_new["age"]
)

bordeaux_train.plot.scatter(x="age", y="log(price)")
y_new_.plot.line()
```

Out[8]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bdc648c548>
```



The points are more evenly spread out when the target is log-price instead of price. For this reason, Ashenfelter chose log-price to be the measure of "wine quality" in his linear regression model.

Fitting Ashenfelter's Model

We are now ready to reproduce Ashenfelter's analysis. To do so, we will need to fit a linear regression model that predicts the log-price from the average summer temperature, winter rainfall, harvest rainfall, and the age of the wine. In other words, the model is of the form \begin{align} \widehat{\text{log(price)}} = b &+ c_1 \cdot (\text{average summer temperature}) \nonumber \\ &+ c_2 \cdot (\text{winter rainfall}) \nonumber \\ &+ c_3 \cdot (\text{harvest rainfall}) \nonumber \\ &+ c_4 \cdot (\text{age of the wine}), \label{eq:multiple_linear_regression} \end{align} where b, c_1, c_2, c_3, c_4 are chosen to minimize \begin{equation} \text{sum of } (\text{log(price)} - \widehat{\text{log(price)}})^2 \text{ over training data.} \end{equation} This is still a *linear regression* model, albeit a more complicated one.

The code to fit this model is the natural extension of the code we wrote to fit the earlier models in this lesson. Instead of passing `bordeaux_train[["age"]]` for `X`, we now supply a `DataFrame` containing all of the features we want to be in the model.

In [9]:

```
ashen_model = LinearRegression()
ashen_model.fit(
    X=bordeaux_train[["summer", "win", "har", "age"]],
    y=bordeaux_train["log(price)"]
)
```

Out[9]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fals
e)
```

This model is much harder to visualize, since it involves five variables: four features, plus the target. Nevertheless, we can obtain predictions from it just as we did with the simpler models above. We just need to supply the values of all of the features in the model, in the same order as in the training data.

In [10]:

```
ashen_model.predict(
    X=bordeaux_test[["summer", "win", "har", "age"]]
)
```

Out[10]:

```
array([3.17926885, 3.4231464 , 3.71919787, 2.83391541, 3.48195778,
       2.4330387 , 2.91879638, 3.5924235 , 3.97294747, 4.04789338,
       3.14087609])
```

Communication Corner: Interpreting the Model

Even though we cannot visualize Ashenfelter's model, we can still interpret the model by examining the values of the *intercept* b and the *coefficients* c_1, c_2, c_3, c_4.

The coefficients are saved in the `.coef_` attribute, after the model has been fitted. (As above, the trailing underscore in `.coef_` reminds us that these are fitted values.)

In [11]:

```
ashen_model.coef_
```

Out[11]:

```
array([ 0.61871092,  0.00119721, -0.00374825,  0.02435187])
```

These coefficients are in the same order as the columns of `X`. So 0.61871092 is the coefficient for **summer**, 0.00119721 the coefficient for **win**, and so on. If you compare these values with the model at the beginning of this lesson, you will see that they are exactly the coefficients that Ashenfelter obtained.

A positive coefficient means that the predicted target *increases* as that feature increases, while a negative coefficient means that it *decreases* as that feature increases. Since **win** has a positive coefficient (0.0012) and **har** has a negative coefficient (-0.0037), we conclude from the model that Bordeaux wines tend to be best when winter rainfall is high and harvest rainfall is low.

Another essential component of a linear regression model is the *intercept*, which is stored in the `.intercept_` attribute, separately from the coefficients.

In [12]:

```
ashen_model.intercept_
```

Out[12]:

```
-7.831137841446707
```

In principle, the intercept is the predicted value when all of the features are equal to 0. However, this interpretation is often purely hypothetical, since it may be impossible for some features to be 0. For example, to interpret the intercept of -7.8 in the model above, we would have to set **summer** equal to 0. That is, we would have to imagine a summer in Bordeaux, France where the average temperature was 0°C (i.e., freezing), which would be so catastrophic that the quality of red wine would be the least of our worries!

Exercises

Exercises 1-3 ask you to fit linear regression models to the Ames housing data set (AmesHousing.txt), which contains information about homes in Ames, Iowa.

1. Fit a linear regression model that predicts the price of a home (**SalePrice**) using square footage (**Gr Liv Area**) as the only feature. Then, make a graph of the fitted model (this is possible because there is only one feature in this model). Do this the way we did it in the lesson, by creating a grid of `X` values and calling `model.predict()` on those `X` values.

In [13]:

```
import pandas as pd
df = pd.read_csv('AmesHousing.txt', sep="\t")
df
```

Out[13]:

	Order	PID	MS SubClass	MS Zoning	Lot Frontage	Lot Area	Street	Alley	Lot Shape	Land Contour	...
0	1	526301100	20	RL	141.0	31770	Pave	NaN	IR1	Lvl	...
1	2	526350040	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	...
2	3	526351010	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	...
3	4	526353030	20	RL	93.0	11160	Pave	NaN	Reg	Lvl	...
4	5	527105010	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	...
...	...	...	...	...	...	...	...	...	...	...	...
2925	2926	923275080	80	RL	37.0	7937	Pave	NaN	IR1	Lvl	...
2926	2927	923276100	20	RL	NaN	8885	Pave	NaN	IR1	Low	...
2927	2928	923400125	85	RL	62.0	10441	Pave	NaN	Reg	Lvl	...
2928	2929	924100070	20	RL	77.0	10010	Pave	NaN	Reg	Lvl	...
2929	2930	924151050	60	RL	74.0	9627	Pave	NaN	Reg	Lvl	...

2930 rows × 82 columns

In [14]:

```
AH_train = df.loc[:2919].copy()
AH_test = df.loc[2920:].copy()

print (AH_train)
```

```
      Order        PID  MS SubClass MS Zoning  Lot Frontage  Lot Area Street
\
0         1  526301100           20        RL         141.0     31770   Pave
1         2  526350040           20        RH          80.0     11622   Pave
2         3  526351010           20        RL          81.0     14267   Pave
3         4  526353030           20        RL          93.0     11160   Pave
4         5  527105010           60        RL          74.0     13830   Pave
...     ...        ...          ...       ...           ...       ...    ...
2915   2916  923227100           20        RL          80.0     13384   Pave
2916   2917  923228130          180        RM          21.0      1533   Pave
2917   2918  923228180          160        RM          21.0      1533   Pave
2918   2919  923228210          160        RM          21.0      1526   Pave
2919   2920  923228260          160        RM          21.0      1936   Pave

     Alley Lot Shape Land Contour  ... Pool Area Pool QC  Fence Misc Feature
\
0      NaN       IR1          Lvl  ...         0     NaN    NaN          NaN
1      NaN       Reg          Lvl  ...         0     NaN  MnPrv          NaN
2      NaN       IR1          Lvl  ...         0     NaN    NaN         Gar2
3      NaN       Reg          Lvl  ...         0     NaN    NaN          NaN
4      NaN       IR1          Lvl  ...         0     NaN  MnPrv          NaN
...    ...       ...          ...  ...       ...     ...    ...          ...
2915   NaN       Reg          Lvl  ...         0     NaN    NaN          NaN
2916   NaN       Reg          Lvl  ...         0     NaN    NaN          NaN
2917   NaN       Reg          Lvl  ...         0     NaN    NaN          NaN
2918   NaN       Reg          Lvl  ...         0     NaN  GdPrv          NaN
2919   NaN       Reg          Lvl  ...         0     NaN    NaN          NaN

     Misc Val Mo Sold Yr Sold Sale Type  Sale Condition  SalePrice
0           0       5    2010       WD           Normal     215000
1           0       6    2010       WD           Normal     105000
2       12500       6    2010       WD           Normal     172000
3           0       4    2010       WD           Normal     244000
4           0       3    2010       WD           Normal     189900
...       ...     ...     ...       ...              ...        ...
2915        0       5    2006       WD           Normal     140000
2916        0       8    2006       WD          Abnorml      92000
2917        0      12    2006       WD          Abnorml      87550
2918        0       6    2006       WD           Normal      79500
2919        0       6    2006       WD           Normal      90500

[2920 rows x 82 columns]
```

In [15]:

```
from sklearn.linear_model import LinearRegression

x_train = AH_train[["Gr Liv Area"]]
x_test = AH_test[["Gr Liv Area"]]
y_train = AH_train['SalePrice']


model = LinearRegression()
model.fit(X = x_train, y = y_train)
model.predict(X = x_test)
```

Out[15]:

```
array([135284.0903835 , 206357.08655893, 206357.08655893, 139083.59017904,
       150035.08958972, 125338.3409187 , 114051.59152606, 121650.59111715,
       168473.8385975 , 236753.08492326])
```

In [16]:

```
model.coef_
```

Out[16]:

```
array([111.74999399])
```

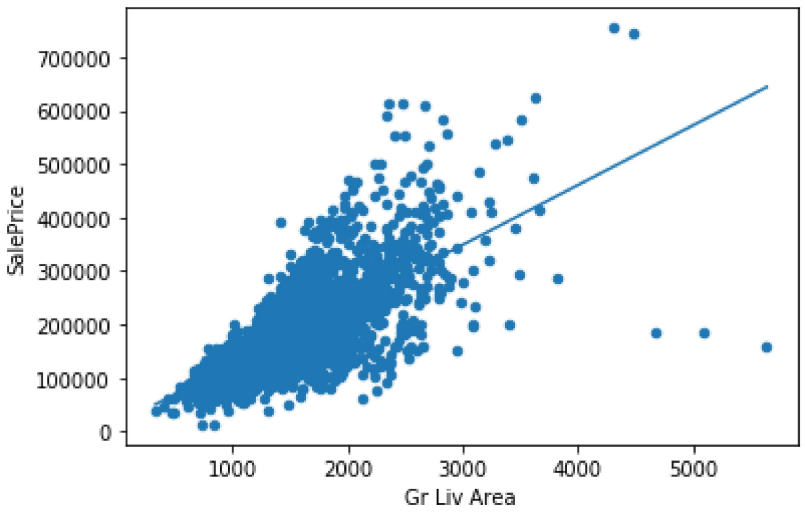
In [17]:

```
import numpy as np
from pandas import Series

X_new = pd.DataFrame()
X_new['Gr Liv Area'] = np.linspace(334, 5642, num = 2)

print(X_new['Gr Liv Area'])
print(type(X_new))

y_new = pd.Series(
model.predict(X_new),
index=X_new['Gr Liv Area']
)# My error appear in this line

AH_train.plot.scatter(x = "Gr Liv Area", y = "SalePrice")
y_new.plot.line()
```

```
0     334.0
1    5642.0
Name: Gr Liv Area, dtype: float64
<class 'pandas.core.frame.DataFrame'>
```

Out[17]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bdc64f29c8>
```



2. There is another way to graph a fitted linear regression model: extract the intercept and coefficient and draw a line with that intercept and slope. Verify that this gives the same graph as Exercise 2. (**SKIP THIS ONE**)

3. Fit a linear regression model that predicts the price of a home using square footage, number of bedrooms (**Bedroom AbvGr**), number of full bathrooms (**Full Bath**), and number of half bathrooms (**Half Bath**). Interpret the coefficients. Then, use your fitted model to predict the price of a home that is 1500 square feet, with 3 bedrooms, 2 full baths, and 1 half bath.

In [18]:

```
from sklearn.linear_model import LinearRegression

x_train = AH_train[["Gr Liv Area", "Bedroom AbvGr", "Full Bath", "Half Bath"]]
x_test1 = AH_test[["Gr Liv Area", "Bedroom AbvGr", "Full Bath", "Half Bath"]]
y_train = AH_train['SalePrice']


model1 = LinearRegression()
model1.fit(X = x_train, y = y_train)
model1.predict(X = x_test1)
```

Out[18]:

```
array([113939.4337988 , 184432.72266503, 184432.72266503, 143346.65901861,
        98099.38966255, 102030.39469077, 120140.00402597,  98131.27850635,
       177681.50650511, 248007.40130143])
```

In [19]:

```
model1.coef_
```

Out[19]:

```
array([   118.15503589, -30043.26796024,  26783.19491319,   1393.24091369])
```

In [20]:

```
model1.intercept_
```

Out[20]:

```
46867.5026591653
```

In [21]:

```
# Cost prediction
cost = (118 * 1500) + (30043 * 3) + (26783 * 2) + (1393 * 1.5)
cost
```

Out[21]:

```
322784.5
```

Predicted Price of Home: $322,784.50

Exercises 4-5 ask you to fit linear regression models to the tips data (tips.csv), which contains information about tips collected by a waiter.

4. Suppose you want to predict how much a male diner will tip on a Sunday bill of \$40.00. Fit a linear regression model to the tips data to answer this question. (Hint: You will need to convert categorical variables to quantitative variables. asZaqAZ)

In [22]:

```
import pandas as pd
import numpy as np

tips_df = pd.read_csv("tips.csv")
tips_df
```

Out[22]:

	obs	totbill	tip	sex	smoker	day	time	size
0	1	16.99	1.01	F	No	Sun	Night	2
1	2	10.34	1.66	M	No	Sun	Night	3
2	3	21.01	3.50	M	No	Sun	Night	3
3	4	23.68	3.31	M	No	Sun	Night	2
4	5	24.59	3.61	F	No	Sun	Night	4
...	...	...	...	...	...	...	...	...
239	240	29.03	5.92	M	No	Sat	Night	3
240	241	27.18	2.00	F	Yes	Sat	Night	2
241	242	22.67	2.00	M	Yes	Sat	Night	2
242	243	17.82	1.75	M	No	Sat	Night	2
243	244	18.78	3.00	F	No	Thu	Night	2

244 rows × 8 columns

In [23]:

```
above_40 = tips_df[tips_df["totbill"] > 40]
above_40
```

Out[23]:

	obs	totbill	tip	sex	smoker	day	time	size
59	60	48.27	6.73	M	No	Sat	Night	4
95	96	40.17	4.73	M	Yes	Fri	Night	4
102	103	44.30	2.50	F	Yes	Sat	Night	3
142	143	41.19	5.00	M	No	Thu	Day	5
156	157	48.17	5.00	M	No	Sun	Night	6
170	171	50.81	10.00	M	Yes	Sat	Night	3
182	183	45.35	3.50	M	Yes	Sun	Night	3
184	185	40.55	3.00	M	Yes	Sun	Night	2
197	198	43.11	5.00	F	Yes	Thu	Day	4
212	213	48.33	9.00	M	No	Sat	Night	4

In [24]:

```
only_M = above_40[above_40['sex'] == 'M']
only_M
```

Out[24]:

	obs	totbill	tip	sex	smoker	day	time	size
59	60	48.27	6.73	M	No	Sat	Night	4
95	96	40.17	4.73	M	Yes	Fri	Night	4
142	143	41.19	5.00	M	No	Thu	Day	5
156	157	48.17	5.00	M	No	Sun	Night	6
170	171	50.81	10.00	M	Yes	Sat	Night	3
182	183	45.35	3.50	M	Yes	Sun	Night	3
184	185	40.55	3.00	M	Yes	Sun	Night	2
212	213	48.33	9.00	M	No	Sat	Night	4

In [25]:

```
only_Sun = only_M[only_M['day'] == 'Sun']
only_Sun
```

Out[25]:

	obs	totbill	tip	sex	smoker	day	time	size
156	157	48.17	5.0	M	No	Sun	Night	6
182	183	45.35	3.5	M	Yes	Sun	Night	3
184	185	40.55	3.0	M	Yes	Sun	Night	2

In [26]:

```
AH_train2 = only_Sun.loc[:182].copy()
AH_test2 = only_Sun.loc[184:].copy()

print (AH_train2)
```

```
     obs  totbill  tip sex smoker  day   time  size
156  157    48.17  5.0   M     No  Sun  Night     6
182  183    45.35  3.5   M    Yes  Sun  Night     3
```

In [27]:

```
from sklearn.linear_model import LinearRegression

x_train2 = AH_train2[["totbill"]]
x_test2 = AH_test2[["totbill"]]
y_train2 = AH_train2['tip']


model2 = LinearRegression()
model2.fit(X = x_train2, y = y_train2)
model2.predict(X = x_test2)
```

Out[27]:

```
array([0.94680851])
```

In [28]:

```
model2.coef_
```

Out[28]:

```
array([0.53191489])
```

In [29]:

```
model2.intercept_
```

Out[29]:

```
-20.622340425531902
```

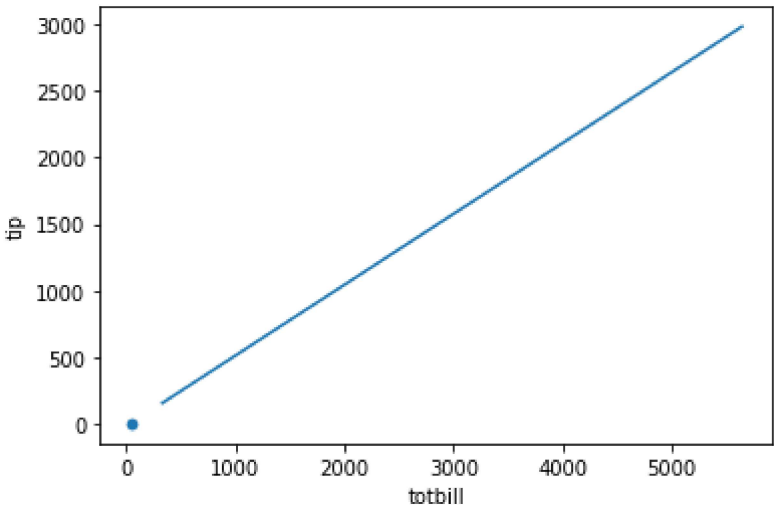
In [30]:

```
import numpy as np
from pandas import Series

X_new = pd.DataFrame()
X_new['totbill'] = np.linspace(334, 5642, num = 2)

print(X_new['totbill'])
print(type(X_new))

y_new = pd.Series(
model2.predict(X_new),
index=X_new['totbill']
)# My error appear in this line

AH_train2.plot.scatter(x = "totbill", y = "tip")
y_new.plot.line()
```

```
0     334.0
1    5642.0
Name: totbill, dtype: float64
<class 'pandas.core.frame.DataFrame'>
```

Out[30]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bdc659a808>
```



Prediction: A male would pay around 5% of their of their $40 bill or more on Sunday.

5. Fit a linear regression model, with no intercept, that predicts the tip from the total bill. That is, we want our predictions to be of the form $$ \widehat{\text{tip}} = c \cdot (\text{total bill}). $$ where c is some coefficient to be learned from the training data.

(*Hint:* `LinearRegression()` has a parameter, `fit_intercept=` , which is `True` by default.)

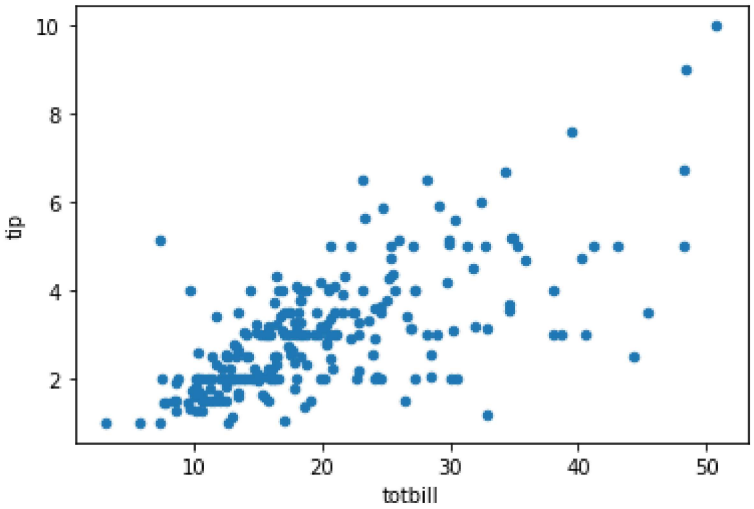
Plot the data and the fitted model. In practical terms, what assumption is being made when we fit a model with no intercept?

In [31]:

```
tips_df.plot.scatter(x="totbill", y="tip")
```

Out[31]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bdc65f79c8>
```



In [32]:

```
AH_train = tips_df.loc[:242].copy()
AH_test = tips_df.loc[243:].copy()

print (AH_train)
```

```
     obs  totbill   tip sex smoker  day   time  size
0      1    16.99  1.01   F     No  Sun  Night     2
1      2    10.34  1.66   M     No  Sun  Night     3
2      3    21.01  3.50   M     No  Sun  Night     3
3      4    23.68  3.31   M     No  Sun  Night     2
4      5    24.59  3.61   F     No  Sun  Night     4
..   ...      ...   ...  ..    ...  ...    ...   ...
238  239    35.83  4.67   F     No  Sat  Night     3
239  240    29.03  5.92   M     No  Sat  Night     3
240  241    27.18  2.00   F    Yes  Sat  Night     2
241  242    22.67  2.00   M    Yes  Sat  Night     2
242  243    17.82  1.75   M     No  Sat  Night     2

[243 rows x 8 columns]
```

In [33]:

```
from sklearn.linear_model import LinearRegression

x_train = AH_train[["totbill"]]
x_test = AH_test[["totbill"]]
y_train = AH_train['tip']

model3 = LinearRegression(fit_intercept=False)
model3.fit(X = x_train, y = y_train)
model3.predict(X = x_test)
```

Out[33]:

```
array([2.69835813])
```

In [34]:

```
model3.coef_
```

Out[34]:

```
array([0.14368254])
```

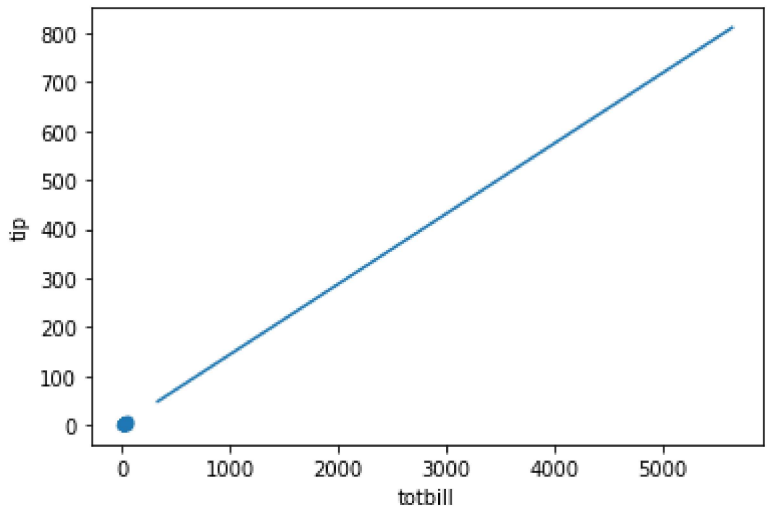
In [35]:

```
import numpy as np
from pandas import Series

X_new = pd.DataFrame()
X_new['totbill'] = np.linspace(334, 5642, num = 2)

print(X_new['totbill'])
print(type(X_new))

y_new = pd.Series(
model3.predict(X_new),
index=X_new['totbill']
)# My error appear in this line

AH_train.plot.scatter(x = "totbill", y = "tip")
y_new.plot.line()
```

```
0     334.0
1    5642.0
Name: totbill, dtype: float64
<class 'pandas.core.frame.DataFrame'>
```

Out[35]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bdc6668a88>
```



Answer: I assume that a model with no intercept forces the regression line to go through the origin at y=0.

In [ ]: