UGBA88 Lab K

April 23, 2020

UGBA 88 DATA & DECISIONS
Haas School of Business
University of California, Berkeley
Spring 2020

WITHOUT DATA
YOU'RE JUST ANOTHER
PERSON WITH AN OPINION

1 Lab K: Non-Linear Regression

Address each data retrieval and analysis section prompt with python code. Answer each discussion section prompt with a few thoughtful sentences.

1.1 Setup

[1]:
```
# Import some useful functions
from numpy import *
from numpy.random import *
from datascience import *

# Import more useful functions for linear regression
from statsmodels.formula.api import *

# Customize look of graphics
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
%matplotlib inline

# Force display of all values
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

# Handle some obnoxious warning messages
import warnings
warnings.filterwarnings("ignore")
```

1.2 Boeing

Your company wants to buy a used Boeing 747. This is the most popular airplane on the planet so many used 747s are available.

The data here describe 288 recent sales of Boeing 747s. For each sale, you have the transaction price (in millions $) and the total number of miles traveled at time of sale (in millions).

1.2.1 Retrieve Data

[2]:
```
# Retrieve data from file 'Airplane_Purchases.csv'.  Show the first few sales.
data = Table().read_table("Airplane_Purchases.csv")
data
```

[2]:
```
mileage | price
65.3896 | 58.7512
28.5512 | 162.88
8.49768 | 309.676
66.8998 | 64.5737
45.697  | 84.7735
59.1262 | 71.2479
22.3089 | 151.944
61.093  | 84.5521
51.7692 | 86.7699
61.3561 | 94.4638
… (278 rows omitted)
```

1.2.2 Analysis

Linear Model

[3]:
```
# Build a linear regression model to predict price based on mileage.
# Show the model goodness of fit (R^2).
# Show the model parameter values (intercept and coefficient).
model = ols("price ~ mileage", data).fit()
model.rsquared
model.params
```

[3]:
```
0.8593451142881448
```

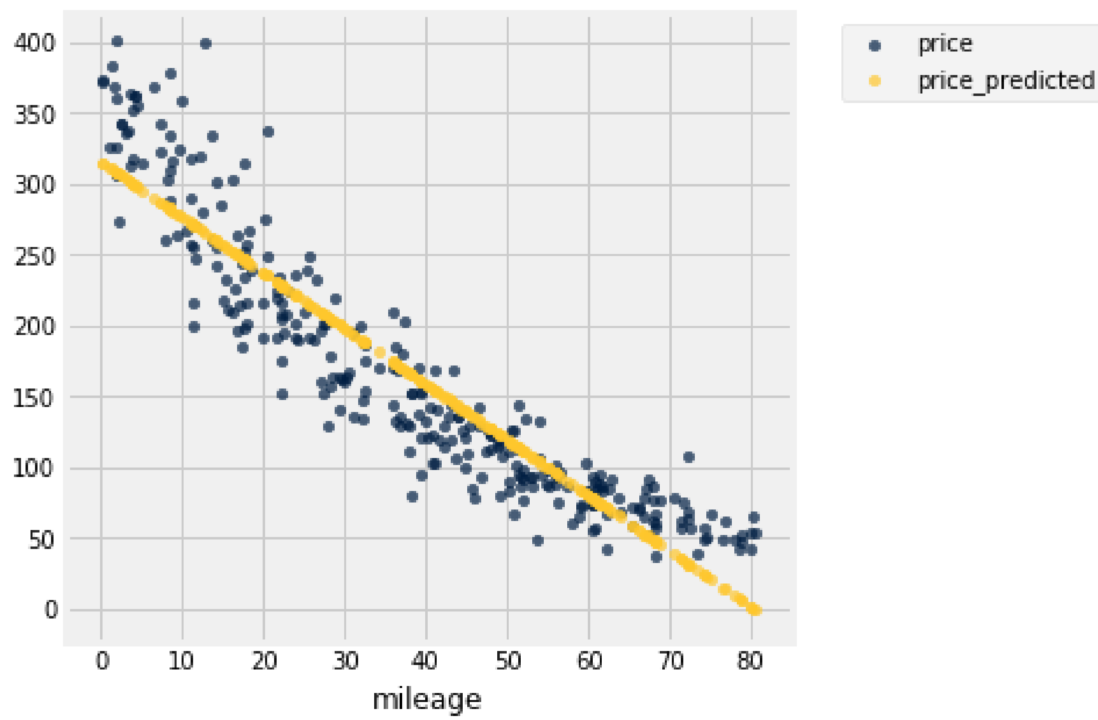
[3]:
```
Intercept    315.161411
mileage       -3.922520
dtype: float64
```

[5]:
```
# Use the model to predict the sale prices.
# Show the first few predictions.
data = data.with_column("price_predicted", model.predict(data))
```

```
data
```

[5]:
```
mileage | price   | price_predicted
65.3896 | 58.7512 | 58.6693
28.5512 | 162.88  | 203.169
8.49768 | 309.676 | 281.829
66.8998 | 64.5737 | 52.7458
45.697  | 84.7735 | 135.914
59.1262 | 71.2479 | 83.2378
22.3089 | 151.944 | 227.654
61.093  | 84.5521 | 75.5228
51.7692 | 86.7699 | 112.096
61.3561 | 94.4638 | 74.4911
… (278 rows omitted)
```

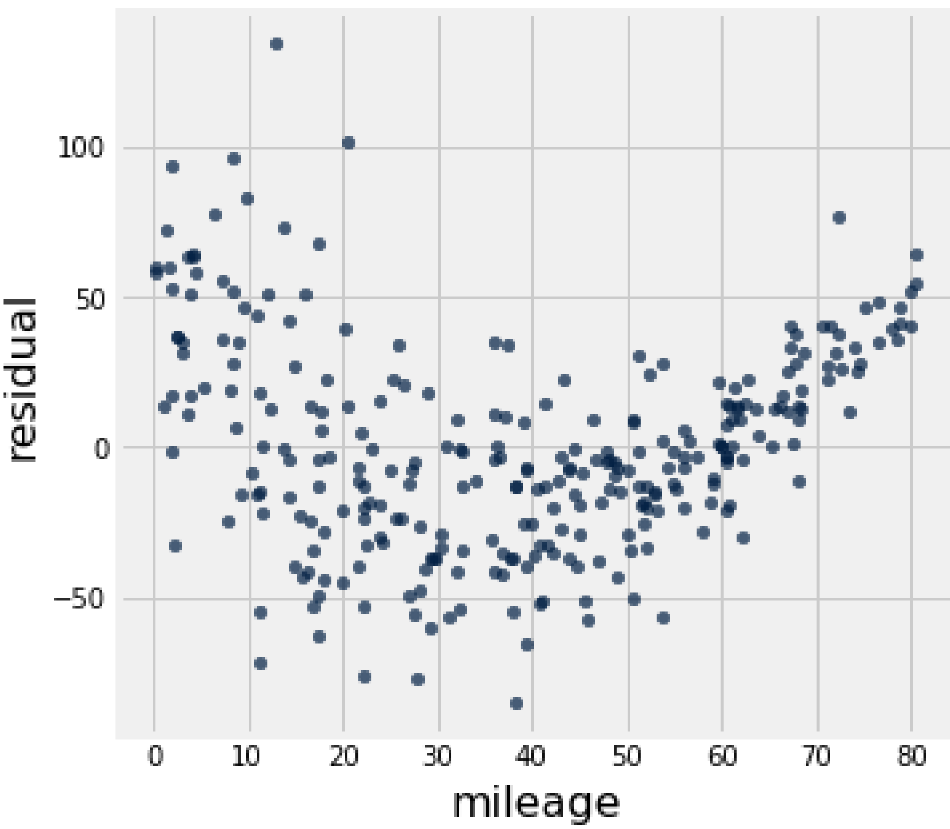
[7]:
```
# Visualize the model performance as a scatterplot of actual price and␣
↪predicted price vs. mileage.
data.scatter("mileage")
```



[8]:
```
# Show the first few residuals.
data = data.with_column("residual", model.resid)
data
```

[8]:
```
mileage | price   | price_predicted | residual
65.3896 | 58.7512 | 58.6693         | 0.0819827
28.5512 | 162.88  | 203.169         | -40.289
8.49768 | 309.676 | 281.829         | 27.8467
66.8998 | 64.5737 | 52.7458         | 11.8279
45.697  | 84.7735 | 135.914         | -51.1406
59.1262 | 71.2479 | 83.2378         | -11.9899
22.3089 | 151.944 | 227.654         | -75.71
61.093  | 84.5521 | 75.5228         | 9.02928
51.7692 | 86.7699 | 112.096         | -25.3257
61.3561 | 94.4638 | 74.4911         | 19.9727
… (278 rows omitted)
```

[10]:
```
# Visualize the residuals as a scatterplot.
data.select("mileage","residual").scatter("mileage")
```



[12]:
```
# Use the model to predict the price of a Boeing 747 that has travelled 50␣
↪million miles.
my_price_predicted = model.predict(Table().with_column("mileage", 50))
my_price_predicted
```

[12]:
```
0    119.035426
dtype: float64
```

Log-Linear Model

[13]:
```
# Reset dataset to include mileage and price variables only.
data = data.select("mileage", "price")
data
```

[13]:
```
mileage | price
65.3896 | 58.7512
28.5512 | 162.88
8.49768 | 309.676
66.8998 | 64.5737
45.697  | 84.7735
59.1262 | 71.2479
22.3089 | 151.944
61.093  | 84.5521
51.7692 | 86.7699
61.3561 | 94.4638
… (278 rows omitted)
```

[14]:
```
# Add a variable for log price.  You can use log() function.
# Show the first few observations of the resulting dataset.
data = data.with_column("log_price", log(data.column("price")))
data
```

[14]:
```
mileage | price   | log_price
65.3896 | 58.7512 | 4.07331
28.5512 | 162.88  | 5.09301
8.49768 | 309.676 | 5.73553
66.8998 | 64.5737 | 4.16781
45.697  | 84.7735 | 4.43998
59.1262 | 71.2479 | 4.26617
22.3089 | 151.944 | 5.02351
61.093  | 84.5521 | 4.43737
51.7692 | 86.7699 | 4.46326
61.3561 | 94.4638 | 4.54822
… (278 rows omitted)
```

[15]:
```
# Build a linear regression model to predict log price based on mileage.
# Show the model goodness of fit (R^2).
# Show the model parameter values (intercept and coefficient).
model = ols("log_price ~ mileage", data).fit()
model.rsquared
model.params
```

[15]: 0.9123608561236595

[15]:
```
Intercept    5.915149
mileage     -0.025407
dtype: float64
```

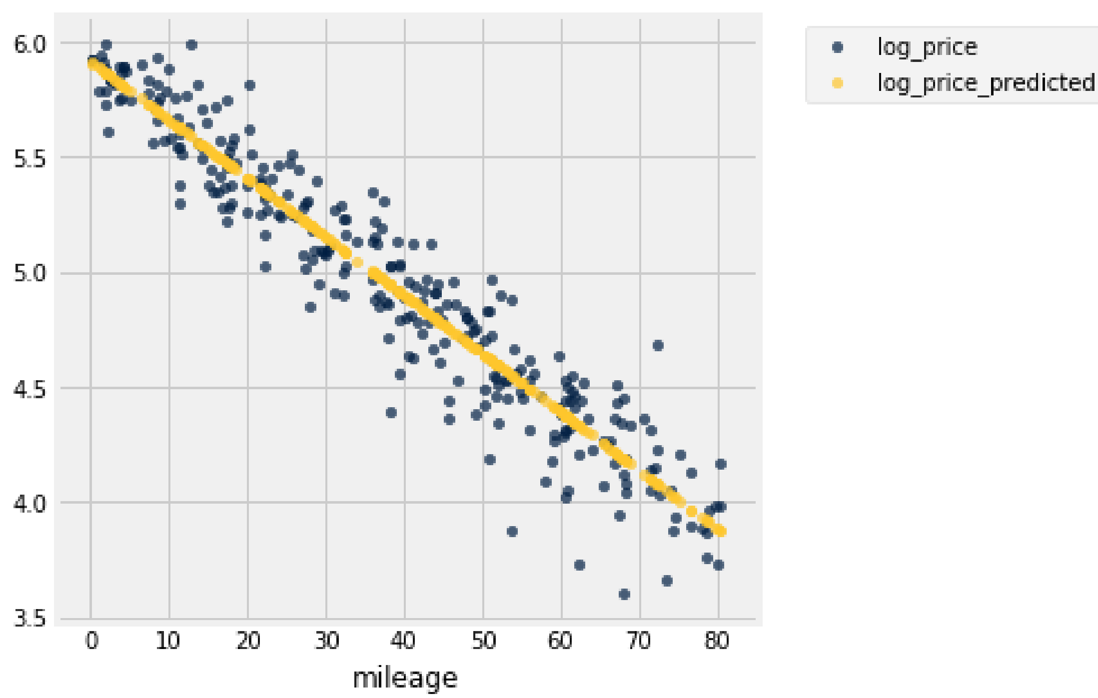
[16]:
```
# Use the model to predict the sale log prices.
# Show the first few predictions.
data = data.with_column("log_price_predicted", model.predict(data))
data
```

[16]:

mileage	price	log_price	log_price_predicted
65.3896	58.7512	4.07331	4.25377
28.5512	162.88	5.09301	5.18973
8.49768	309.676	5.73553	5.69924
66.8998	64.5737	4.16781	4.2154
45.697	84.7735	4.43998	4.75411
59.1262	71.2479	4.26617	4.4129
22.3089	151.944	5.02351	5.34834
61.093	84.5521	4.43737	4.36293
51.7692	86.7699	4.46326	4.59982
61.3561	94.4638	4.54822	4.35625

… (278 rows omitted)

[17]:
```
# Visualize the model performance as a scatterplot of actual log price and␣
↪predicted log price vs. mileage.
data.select("mileage", "log_price", "log_price_predicted").scatter("mileage")
```



[18]:

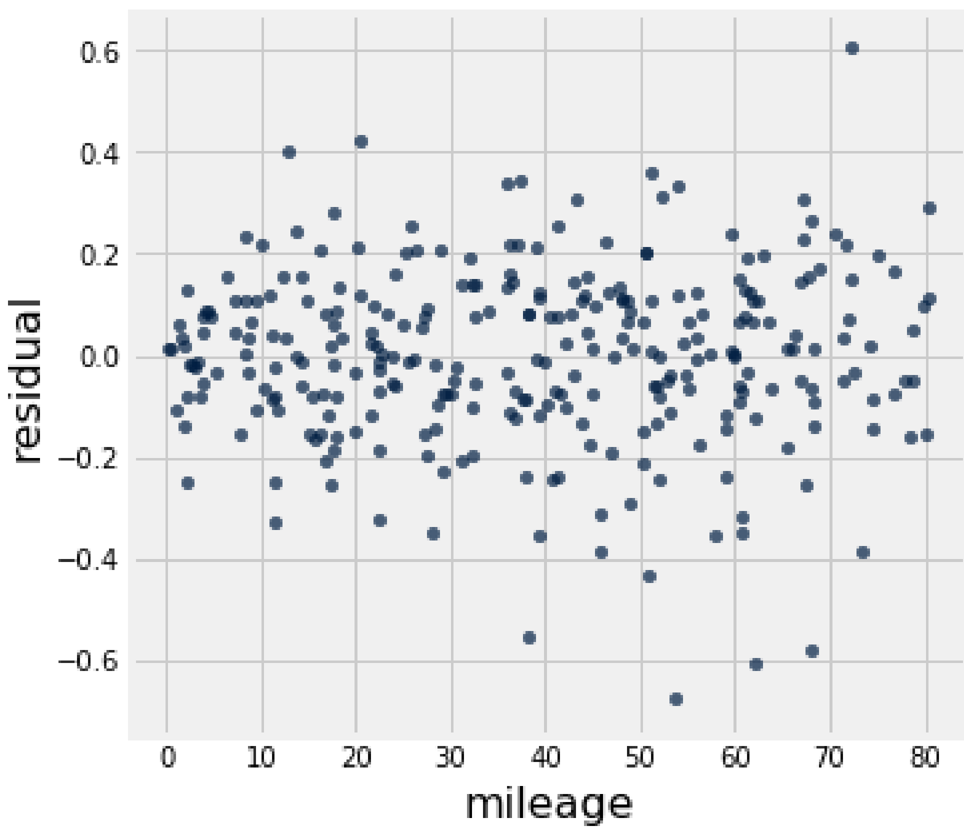
```
# Show the first few residuals.
data = data.with_column("residual", model.resid)
data
```

[18]:

```
mileage | price    | log_price | log_price_predicted | residual
65.3896 | 58.7512 | 4.07331   | 4.25377             | -0.180453
28.5512 | 162.88  | 5.09301   | 5.18973             | -0.0967234
8.49768 | 309.676 | 5.73553   | 5.69924             | 0.0362814
66.8998 | 64.5737 | 4.16781   | 4.2154              | -0.0475901
45.697  | 84.7735 | 4.43998   | 4.75411             | -0.314123
59.1262 | 71.2479 | 4.26617   | 4.4129              | -0.146738
22.3089 | 151.944 | 5.02351   | 5.34834             | -0.324822
61.093  | 84.5521 | 4.43737   | 4.36293             | 0.0744365
51.7692 | 86.7699 | 4.46326   | 4.59982             | -0.136565
61.3561 | 94.4638 | 4.54822   | 4.35625             | 0.191968
… (278 rows omitted)
```

[20]:

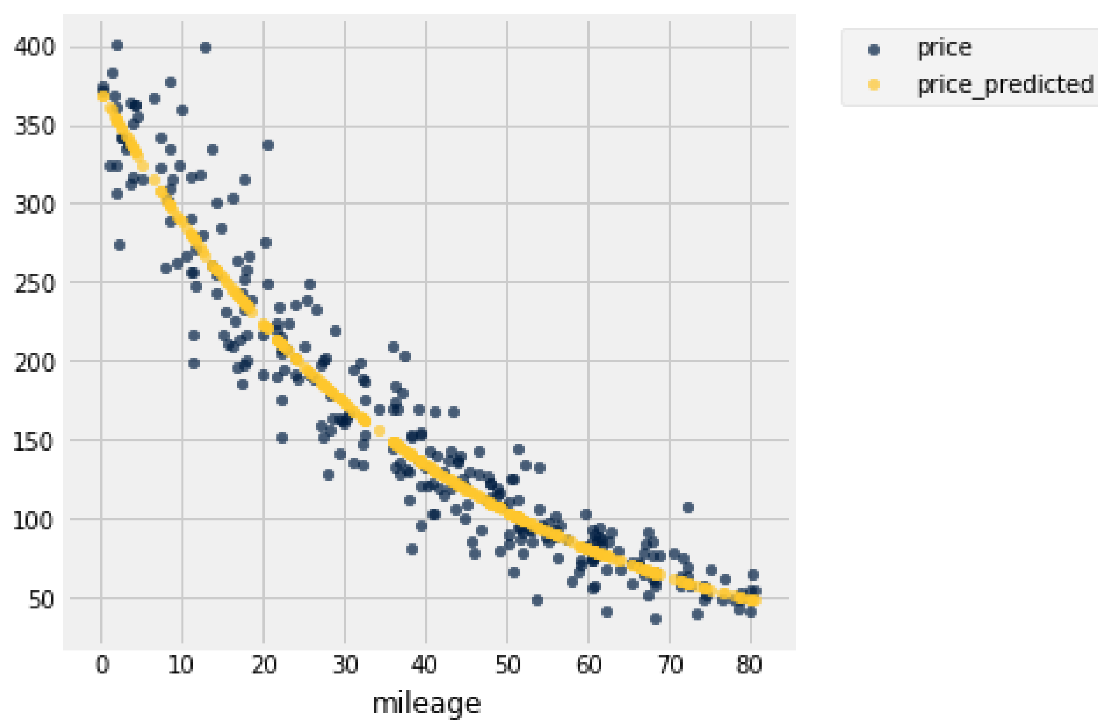
```
# Visualize the residuals as a scatterplot.
data.select("mileage", "residual").scatter("mileage")
```



[22]:
```
# Predict the sale prices.  Show the first few predictions. You can use the␣
↪exp() function.
data = data.with_column("price_predicted", exp(data.
↪column("log_price_predicted")))
data
```

[22]:
```
mileage | price    | log_price | log_price_predicted | residual    |
price_predicted
65.3896 | 58.7512 | 4.07331   | 4.25377             | -0.180453  | 70.3699
28.5512 | 162.88  | 5.09301   | 5.18973             | -0.0967234 | 179.421
8.49768 | 309.676 | 5.73553   | 5.69924             | 0.0362814  | 298.642
66.8998 | 64.5737 | 4.16781   | 4.2154              | -0.0475901 | 67.721
45.697  | 84.7735 | 4.43998   | 4.75411             | -0.314123  | 116.06
59.1262 | 71.2479 | 4.26617   | 4.4129              | -0.146738  | 82.5087
22.3089 | 151.944 | 5.02351   | 5.34834             | -0.324822  | 210.258
61.093  | 84.5521 | 4.43737   | 4.36293             | 0.0744365  | 78.4868
51.7692 | 86.7699 | 4.46326   | 4.59982             | -0.136565  | 99.4669
61.3561 | 94.4638 | 4.54822   | 4.35625             | 0.191968   | 77.9641
… (278 rows omitted)
```

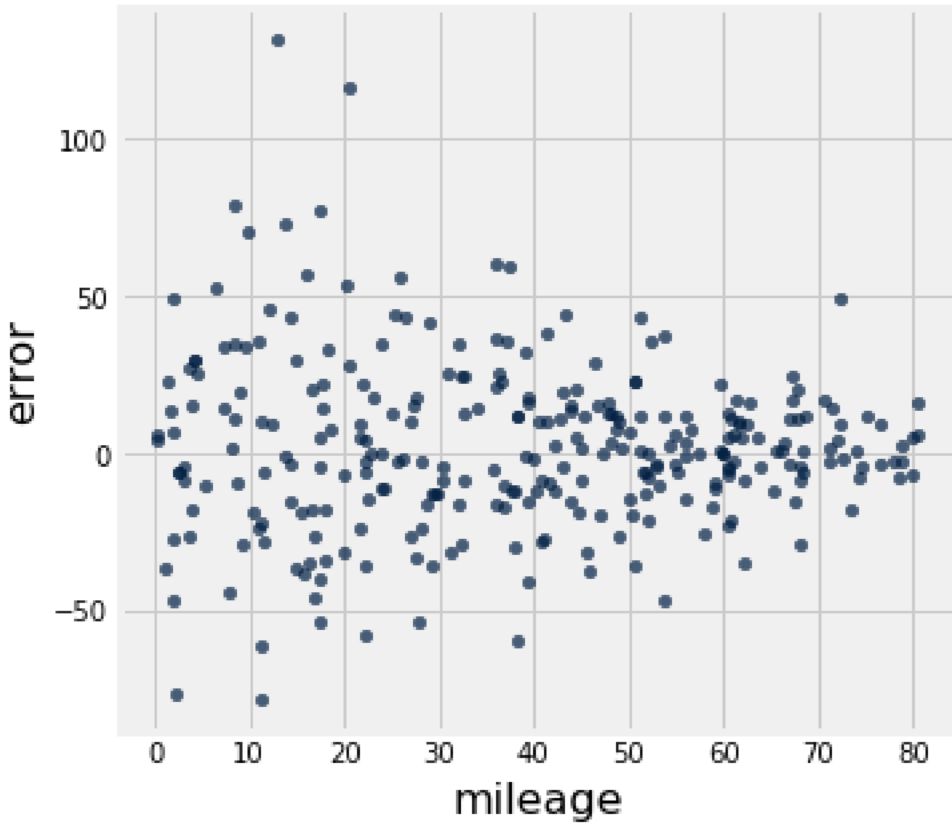
[24]:
```
# Visualize the model performance as a scatterplot of actual price and␣
↪predicted price vs. mileage.
data.select("mileage", "price", "price_predicted").scatter("mileage")
```



[25]:
```
# Show the errors between the sale actual prices and predicted prices.
data = data.with_column("error", data.column("price") - data.
→column("price_predicted"))
data
```

[25]:
```
mileage | price   | log_price | log_price_predicted | residual   |
price_predicted | error
65.3896 | 58.7512 | 4.07331   | 4.25377             | -0.180453  | 70.3699
| -11.6186
28.5512 | 162.88  | 5.09301   | 5.18973             | -0.0967234 | 179.421
| -16.5413
8.49768 | 309.676 | 5.73553   | 5.69924             | 0.0362814  | 298.642
| 11.0341
66.8998 | 64.5737 | 4.16781   | 4.2154              | -0.0475901 | 67.721
| -3.14737
45.697  | 84.7735 | 4.43998   | 4.75411             | -0.314123  | 116.06
| -31.2863
59.1262 | 71.2479 | 4.26617   | 4.4129              | -0.146738  | 82.5087
| -11.2608
22.3089 | 151.944 | 5.02351   | 5.34834             | -0.324822  | 210.258
| -58.3139
61.093  | 84.5521 | 4.43737   | 4.36293             | 0.0744365  | 78.4868
| 6.06522
51.7692 | 86.7699 | 4.46326   | 4.59982             | -0.136565  | 99.4669
| -12.697
61.3561 | 94.4638 | 4.54822   | 4.35625             | 0.191968   | 77.9641
| 16.4997
… (278 rows omitted)
```

[26]:
```
# Visualize the errors as a scatterplot.
data.select("mileage", "error").scatter("mileage")
```



[28]:
```
# Use the model to predict the price of a Boeing 747 that has travelled 50␣
↪million miles.
my_log_price_predicted = model.predict(Table().with_column('mileage', 50))
my_price_predicted = exp(my_log_price_predicted)
my_price_predicted
```

[28]:
```
0    104.040122
dtype: float64
```

Linear-Log Model

[29]:
```
# Reset dataset to include mileage and price variables only.
data = data.select("mileage", "price")
data
```

[29]:
```
mileage | price
65.3896 | 58.7512
28.5512 | 162.88
8.49768 | 309.676
66.8998 | 64.5737
```

```
45.697  | 84.7735
59.1262 | 71.2479
22.3089 | 151.944
61.093  | 84.5521
51.7692 | 86.7699
61.3561 | 94.4638
… (278 rows omitted)
```

[30]:
```
# Add a variable for log mileage.  You can use log() function.
# Show the first few observations of the resulting dataset.
data = data.with_column("log_mileage", log(data.column("mileage")))
data
```

[30]:
```
mileage | price   | log_mileage
65.3896 | 58.7512 | 4.18036
28.5512 | 162.88  | 3.3517
8.49768 | 309.676 | 2.13979
66.8998 | 64.5737 | 4.2032
45.697  | 84.7735 | 3.82203
59.1262 | 71.2479 | 4.07967
22.3089 | 151.944 | 3.10499
61.093  | 84.5521 | 4.1124
51.7692 | 86.7699 | 3.9468
61.3561 | 94.4638 | 4.11669
… (278 rows omitted)
```

[31]:
```
# Build a linear regression model to predict price based on log mileage.
# Show the model goodness of fit (R^2).
# Show the model parameter values (intercept and coefficient).
model = ols("price ~ log_mileage", data).fit()
model.rsquared
model.params
```

[31]:
```
0.8152573793319022
```

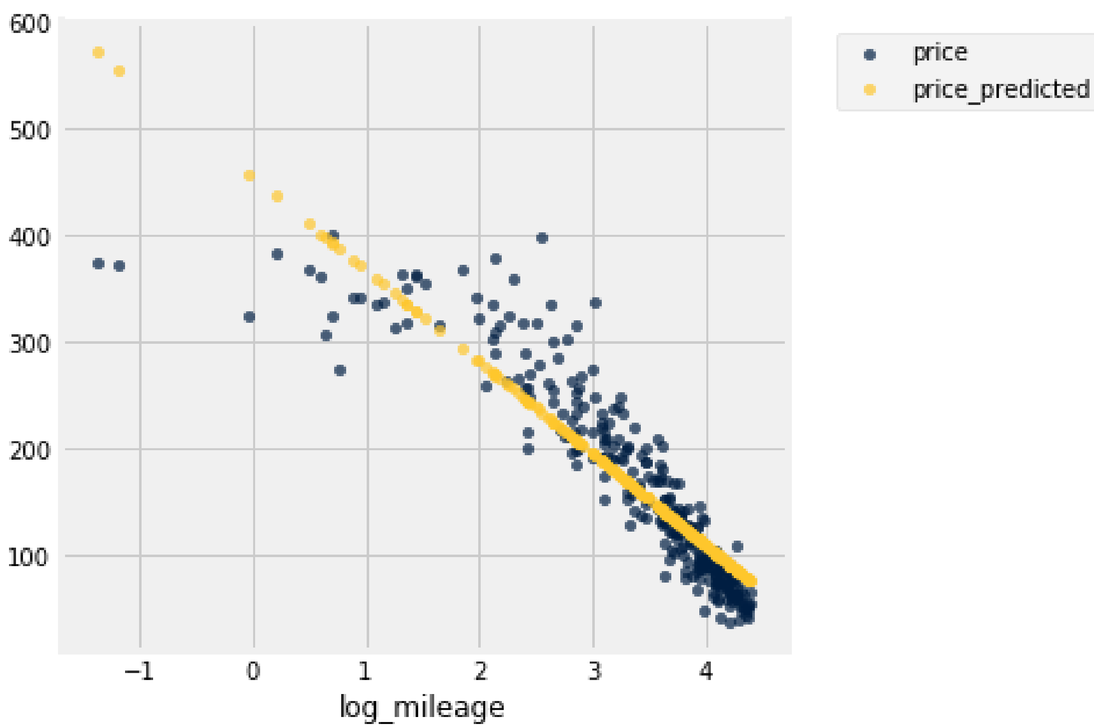
[31]:
```
Intercept      453.880889
log_mileage    -86.289307
dtype: float64
```

[32]:
```
# Use the model to predict the sale prices.
# Show the first few predictions.
data = data.with_column("price_predicted", model.predict(data))
data
```

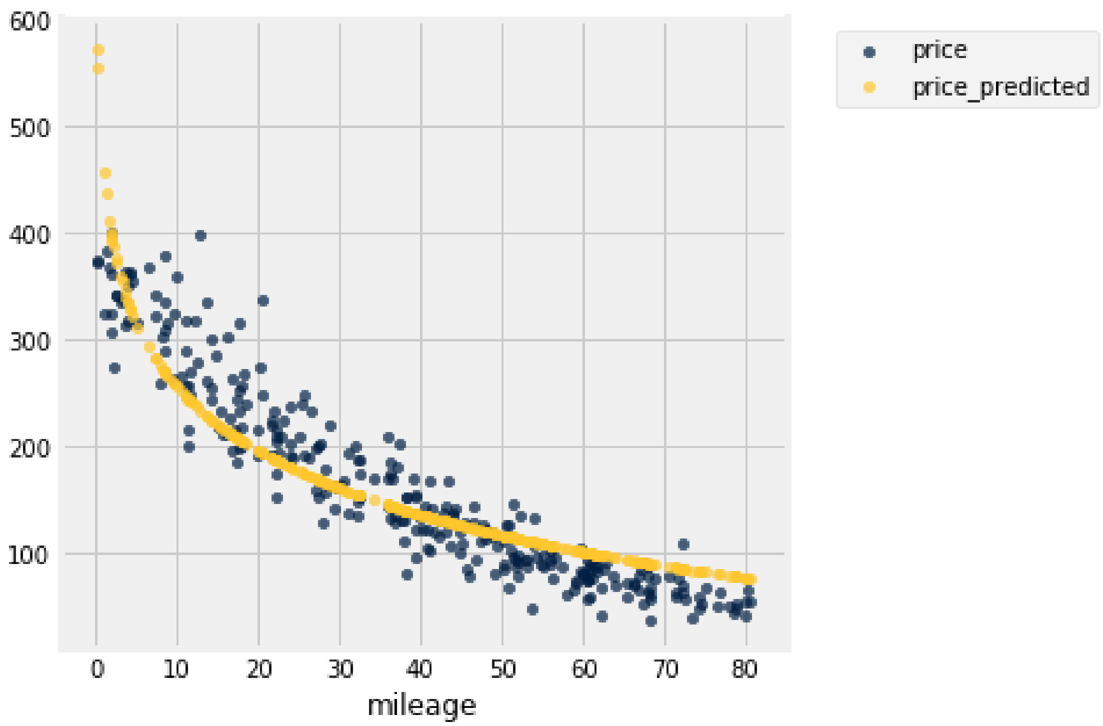
[32]:
```
mileage | price   | log_mileage | price_predicted
65.3896 | 58.7512 | 4.18036     | 93.1602
28.5512 | 162.88  | 3.3517      | 164.665
```

```
8.49768 | 309.676 | 2.13979     | 269.24
66.8998 | 64.5737 | 4.2032      | 91.1901
45.697  | 84.7735 | 3.82203     | 124.08
59.1262 | 71.2479 | 4.07967     | 101.849
22.3089 | 151.944 | 3.10499     | 185.954
61.093  | 84.5521 | 4.1124      | 99.0249
51.7692 | 86.7699 | 3.9468      | 113.315
61.3561 | 94.4638 | 4.11669     | 98.6542
… (278 rows omitted)
```

[33]:
```
# Visualize the model performance as a scatterplot of actual price and␣
↪predicted price vs. log mileage.
data.select("log_mileage", "price", "price_predicted").scatter("log_mileage")
```



[34]:
```
# Visualize the model performance as a scatterplot of actual price and␣
↪predicted price vs. mileage.
data.select("mileage", "price", "price_predicted").scatter("mileage")
```



[37]:
```
# Use the model to predict the price of a Boeing 747 that has travelled 50␣
↪million miles.
my_price_predicted = model.predict(Table().with_column('log_mileage', log(50)))
my_price_predicted
```

[37]:
```
0    116.315137
dtype: float64
```

Log-Log Model

[40]:
```
# Reset dataset to include mileage and price variables only.
data = data.select("mileage", "price")
data
```

[40]:
```
mileage | price
65.3896 | 58.7512
28.5512 | 162.88
8.49768 | 309.676
66.8998 | 64.5737
45.697  | 84.7735
59.1262 | 71.2479
22.3089 | 151.944
61.093  | 84.5521
```

```
51.7692 | 86.7699
61.3561 | 94.4638
… (278 rows omitted)
```

[42]:
```
# Add a variable for log mileage.  You can use log() function.
# Add a variable for log price.  You can use log() function.
# Show the first few observations of the resulting dataset.
data = data.with_columns("log_mileage", log(data.column("mileage")),␣
↪"log_price", log(data.column("price")))
data
```

[42]:
```
mileage | price   | log_mileage | log_price
65.3896 | 58.7512 | 4.18036     | 4.07331
28.5512 | 162.88  | 3.3517      | 5.09301
8.49768 | 309.676 | 2.13979     | 5.73553
66.8998 | 64.5737 | 4.2032      | 4.16781
45.697  | 84.7735 | 3.82203     | 4.43998
59.1262 | 71.2479 | 4.07967     | 4.26617
22.3089 | 151.944 | 3.10499     | 5.02351
61.093  | 84.5521 | 4.1124      | 4.43737
51.7692 | 86.7699 | 3.9468      | 4.46326
61.3561 | 94.4638 | 4.11669     | 4.54822
… (278 rows omitted)
```

[43]:
```
# Build a linear regression model to predict log price based on log mileage.
# Show the model goodness of fit (R^2).
# Show the model parameter values (intercept and coefficient).
model = ols("log_price ~ log_mileage", data).fit()
model.rsquared
model.params
```

[43]:
```
0.6991840078120621
```

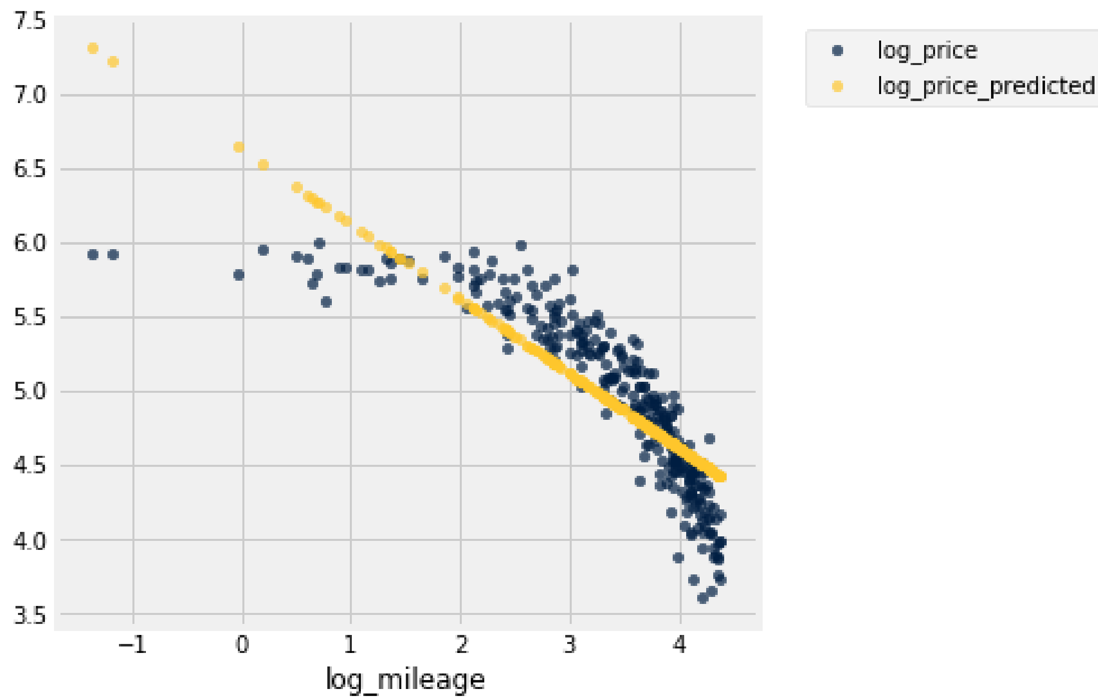
[43]:
```
Intercept      6.624381
log_mileage   -0.502345
dtype: float64
```

[45]:
```
# Use the model to predict the sale log prices.
# Show the first few predictions.
data = data.with_columns("log_price_predicted", model.predict(data))
data
```

[45]:
```
mileage | price   | log_mileage | log_price | log_price_predicted
65.3896 | 58.7512 | 4.18036     | 4.07331   | 4.5244
28.5512 | 162.88  | 3.3517      | 5.09301   | 4.94067
8.49768 | 309.676 | 2.13979     | 5.73553   | 5.54947
66.8998 | 64.5737 | 4.2032      | 4.16781   | 4.51293
```

```
45.697  | 84.7735 | 3.82203     | 4.43998    | 4.7044
59.1262 | 71.2479 | 4.07967     | 4.26617    | 4.57498
22.3089 | 151.944 | 3.10499     | 5.02351    | 5.06461
61.093  | 84.5521 | 4.1124      | 4.43737    | 4.55854
51.7692 | 86.7699 | 3.9468      | 4.46326    | 4.64173
61.3561 | 94.4638 | 4.11669     | 4.54822    | 4.55638
… (278 rows omitted)
```

[46]:
```
# Visualize the model performance as a scatterplot of actual log price and␣
↪predicted log price vs. log mileage.
data.select("log_mileage", "log_price", "log_price_predicted").
↪scatter("log_mileage")
```

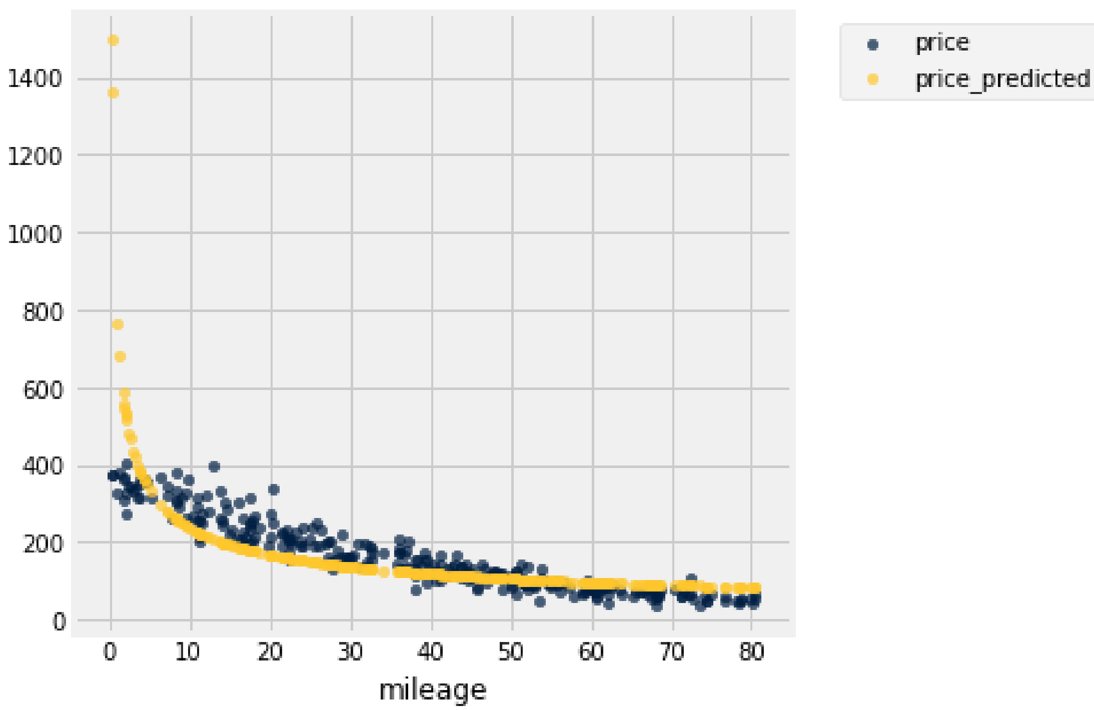


[48]:
```
# Predict the sale prices.  You can use the exp() function.  Show the first few␣
↪predictions.
data = data.with_column("price_predicted", exp(data.
↪column("log_price_predicted")))
data
```

[48]:
```
mileage | price   | log_mileage | log_price | log_price_predicted |
price_predicted
65.3896 | 58.7512 | 4.18036     | 4.07331   | 4.5244              | 92.2403
28.5512 | 162.88  | 3.3517      | 5.09301   | 4.94067             | 139.864
8.49768 | 309.676 | 2.13979     | 5.73553   | 5.54947             | 257.101
```

```
66.8998 | 64.5737 | 4.2032      | 4.16781    | 4.51293              | 91.1884
45.697  | 84.7735 | 3.82203     | 4.43998    | 4.7044               | 110.432
59.1262 | 71.2479 | 4.07967     | 4.26617    | 4.57498              | 97.026
22.3089 | 151.944 | 3.10499     | 5.02351    | 5.06461              | 158.318
61.093  | 84.5521 | 4.1124      | 4.43737    | 4.55854              | 95.444
51.7692 | 86.7699 | 3.9468      | 4.46326    | 4.64173              | 103.724
61.3561 | 94.4638 | 4.11669     | 4.54822    | 4.55638              | 95.2383
… (278 rows omitted)
```

[49]:
```
# Visualize the model performance as a scatterplot of actual price and
↪predicted price vs. mileage.
data.select("mileage", "price", "price_predicted").scatter("mileage")
```



[55]:
```
# Use the model to predict the price of a Boeing 747 that has travelled 50
↪million miles.
my_log_price_predicted = model.predict(Table().with_column("log_mileage",
↪log(50)))
my_price_predicted = exp(my_price_predicted)
my_price_predicted
```

[55]:
```
0    105.551294
dtype: float64
```

1.2.3 Discussion

Assume you are the Director of Procurement, responsible for purchasing a jet plane for your company. A Boeing 747 with 50 million miles has become available. How much do you think it's worth?

What does your analysis tell you about how to make business decisions?

I think it is worth around 105 million dollars based on the log-log analysis, as that prediction/model seems to fit the data the best based off the scatterplots and the range of residuals/errors. My business analysis tells me that I should look through different angles of analysis before making a decision, as different models can yield different results.

1.3 Boeing revisited

Your company wants to buy a used Boeing 747. This is the most popular airplane on the planet so many used 747s are available.

The data here describe 288 recent sales of Boeing 747s. For each sale, you have the transaction price (in millions $) and the total number of miles traveled at time of sale (in millions).

1.3.1 Retrieve Data

[57]:
```
# Retrieve data from file 'Airplane_Purchases.csv'.  Show the first few sales.
data = Table().read_table("Airplane_Purchases.csv")
data
```

[57]:
```
mileage | price
65.3896 | 58.7512
28.5512 | 162.88
8.49768 | 309.676
66.8998 | 64.5737
45.697  | 84.7735
59.1262 | 71.2479
22.3089 | 151.944
61.093  | 84.5521
51.7692 | 86.7699
61.3561 | 94.4638
… (278 rows omitted)
```

1.3.2 Analysis: Polynomial Model

[59]:
```
# Add a variable for square mileage.  You can use **2 operation.
# Show the first few observations of the resulting dataset.
data = data.with_columns("square_mileage", data.column("mileage")**2)
data
```

```
[59]: mileage | price   | square_mileage
      65.3896 | 58.7512 | 4275.81
      28.5512 | 162.88  | 815.173
      8.49768 | 309.676 | 72.2105
      66.8998 | 64.5737 | 4475.58
      45.697  | 84.7735 | 2088.21
      59.1262 | 71.2479 | 3495.91
      22.3089 | 151.944 | 497.687
      61.093  | 84.5521 | 3732.36
      51.7692 | 86.7699 | 2680.06
      61.3561 | 94.4638 | 3764.57
      … (278 rows omitted)
```

```
[61]: # Build a linear regression model to predict price based on mileage and quare␣
       ↪mileage.
      # Show the model goodness of fit (R^2).
      # Show the model parameter values (intercept and coefficient).
      model = ols("price ~ mileage + square_mileage", data).fit()
      model.rsquared
      model.params
```

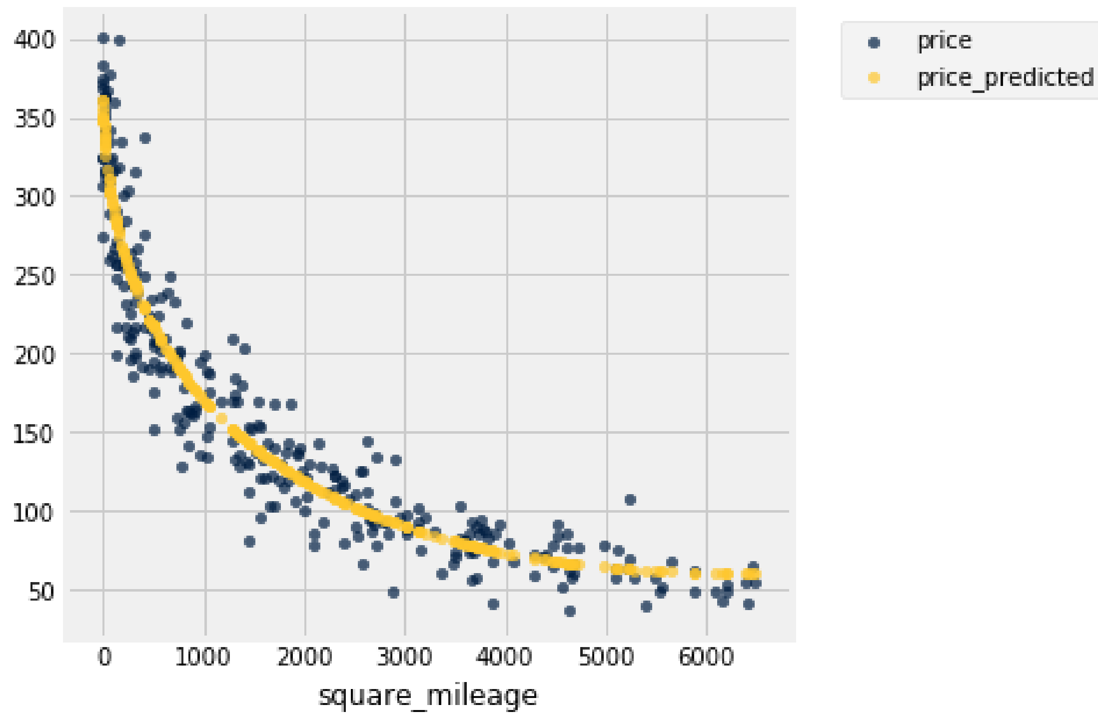
```
[61]: 0.9147580938011316
```

```
[61]: Intercept          363.394929
      mileage             -7.608377
      square_mileage       0.047687
      dtype: float64
```

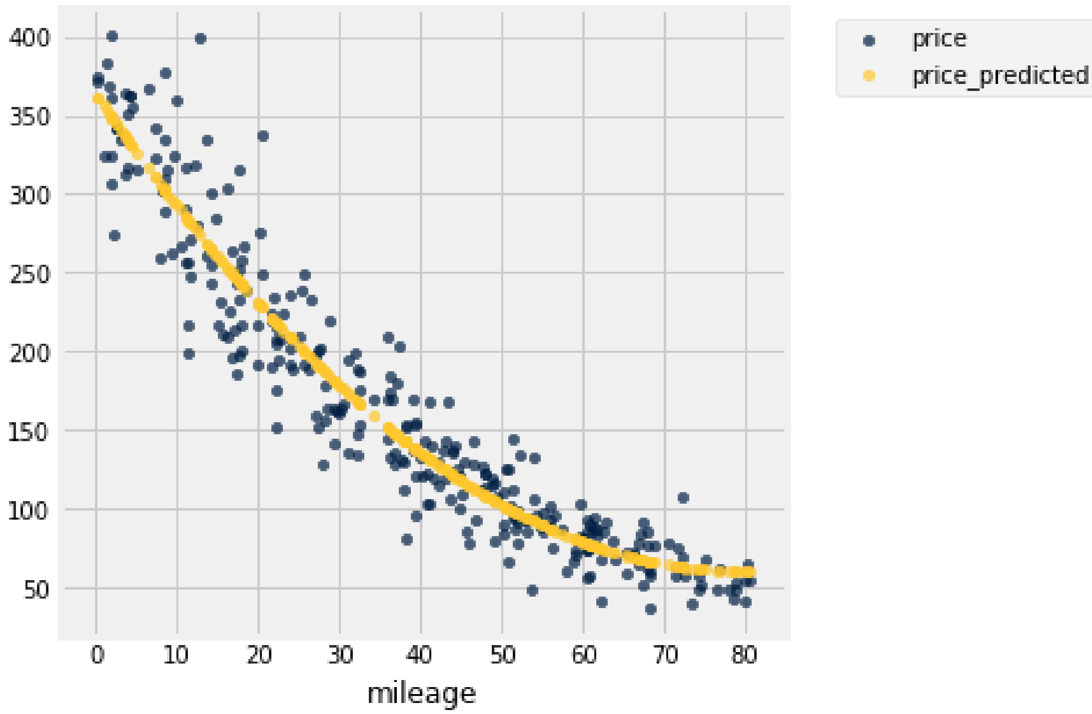
```
[64]: # Use the model to predict the sale prices.
      # Show the first few predictions.
      data = data.with_column("price_predicted", model.predict(data))
      data
```

```
[64]: mileage | price   | square_mileage | price_predicted
      65.3896 | 58.7512 | 4275.81        | 69.7867
      28.5512 | 162.88  | 815.173        | 185.04
      8.49768 | 309.676 | 72.2105        | 302.185
      66.8998 | 64.5737 | 4475.58        | 67.8237
      45.697  | 84.7735 | 2088.21        | 115.296
      59.1262 | 71.2479 | 3495.91        | 80.2503
      22.3089 | 151.944 | 497.687        | 217.394
      61.093  | 84.5521 | 3732.36        | 76.5615
      51.7692 | 86.7699 | 2680.06        | 97.3191
      61.3561 | 94.4638 | 3764.57        | 76.0962
      … (278 rows omitted)
```

[67]:
```
# Visualize the model performance as a scatterplot of actual price and␣
↪predicted price vs. square mileage.
data.select("price", "price_predicted", "square_mileage").
↪scatter("square_mileage")
```



[68]:
```
# Visualize the model performance as a scatterplot of actual price and␣
↪predicted price vs. mileage.
data.select("mileage", "price", "price_predicted").scatter("mileage")
```



[80]:
```
# Use the model to predict the price of a Boeing 747 that has travelled 50␣
↪million miles.
my_predicted_price = model.predict(Table().with_columns("mileage", 50,␣
↪"square_mileage", 50**2))
my_predicted_price
```

[80]:
```
0    102.193869
dtype: float64
```

1.3.3 Discussion

Assume you are the Director of Procurement, responsible for purchasing a jet plane for your company. A Boeing 747 with 50 million miles has become available. How much do you think it's worth?

What does your analysis tell you about how to make business decisions?

I think the plane is worth around 102 million dollars, based off the model predictions. This prediction seems accurate as it is similar to the earlier predictions made based off differing models, and we are basing it off mileage and square mileage. My analysis tells me that I should always try to find new angles of analysis to deepen my intuition and understanding of data in order to make the best possible business decisions.

 Document revised April 13, 2020