learn-co-curriculum / dsc-log-transformations-lab Public

View license

0 stars 5 forks

Star Watch

Code Issues Pull requests Actions Projects Security Insights

solution

This branch is 3 commits ahead, 5 commits behind master.

hoffm386 add hint about years ... on Jul 5 4

View code

README.md

Log Transformations - Lab

Introduction

It's time to practice some logarithmic transformations on the Ames Housing dataset!

Objectives

You will be able to:

- Determine if a log transformation would be useful for a specific model or set of data
- Apply log transformations to independent and dependent variables in linear regression
- Interpret the coefficients of variables that have been transformed using a log transformation

Ames Housing Data

Below we load the numeric features from the Ames Housing dataset into a dataframe. We also drop any rows with missing data.

```
import pandas as pd
ames = pd.read_csv("ames.csv", index_col=0)
ames = ames.select_dtypes("number")
ames.dropna(inplace=True)
ames
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	Yea
Id						
1	60	65.0	8450	7	5	200
2	20	80.0	9600	6	8	197
3	60	68.0	11250	7	5	200
4	70	60.0	9550	7	5	191
5	60	84.0	14260	8	5	200
...	...	...	...	...	...	...
1456	60	62.0	7917	6	5	199
1457	20	85.0	13175	6	6	197
1458	70	66.0	9042	7	9	194
1459	20	68.0	9717	5	6	195

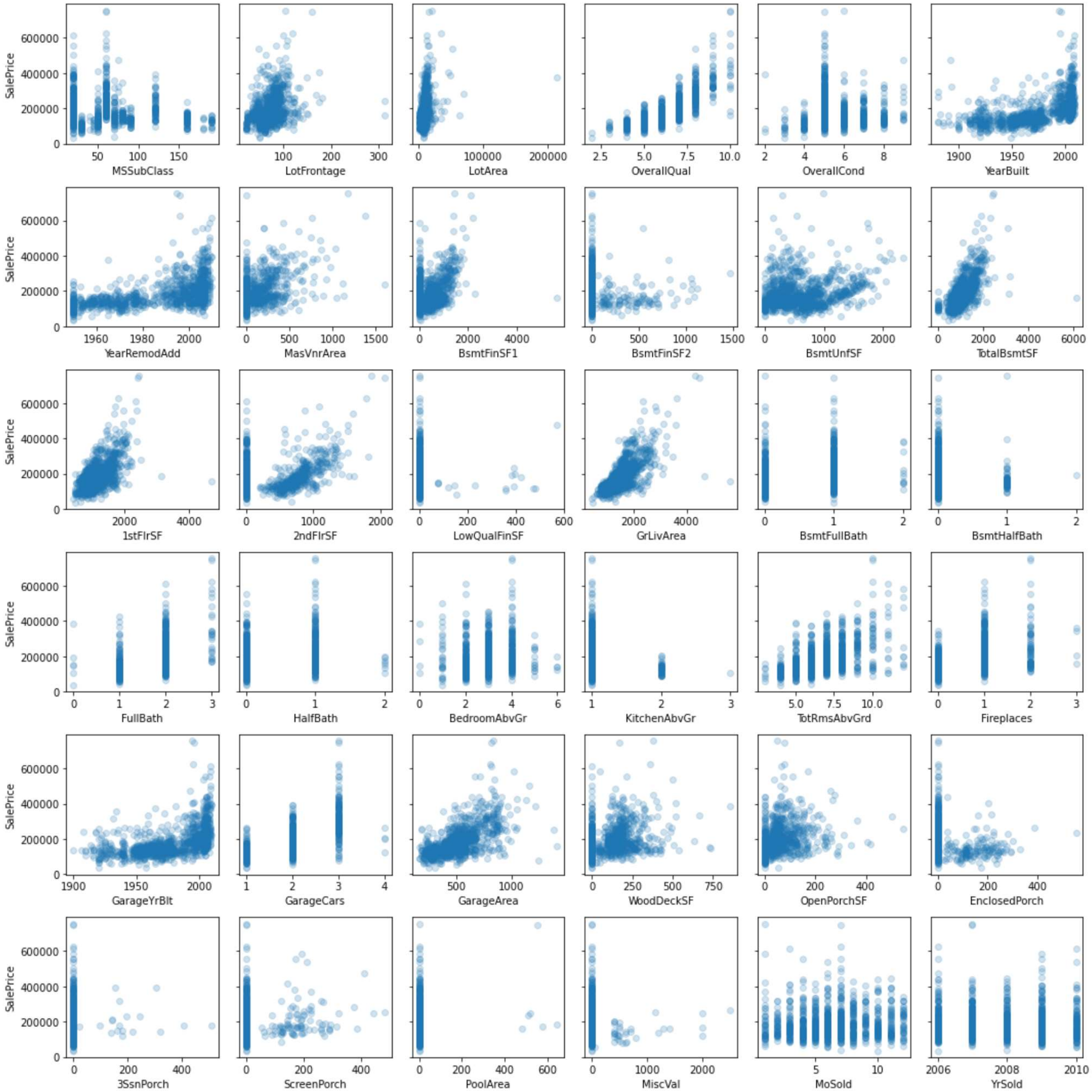
	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	Yea
Id						
1460	20	75.0	9937	5	6	196

1121 rows × 37 columns

Identify Good Candidates for Log Transformation

Below we plot each of the potential numeric features against `SalePrice` :

```
import matplotlib.pyplot as plt
import numpy as np

y = ames["SalePrice"]
X = ames.drop("SalePrice", axis=1)

fig, axes = plt.subplots(nrows=6, ncols=6, figsize=(15,15), sharey=True)

for i, column in enumerate(X.columns):
    # Locate applicable axes
    row = i // 6
    col = i % 6
    ax = axes[row][col]

    # Plot feature vs. y and label axes
    ax.scatter(X[column], y, alpha=0.2)
    ax.set_xlabel(column)
    if col == 0:
        ax.set_ylabel("SalePrice")

fig.tight_layout()
```

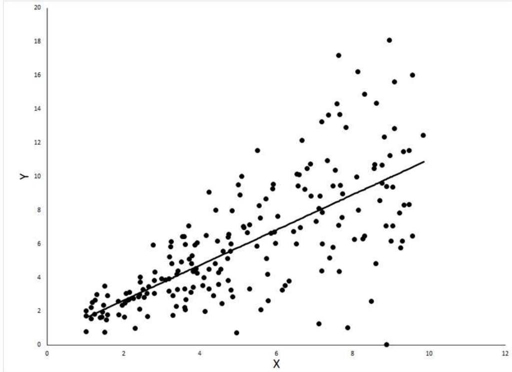


Let's say we want to build a model with **at least one log-transformed feature** as well as a **log-transformed target**

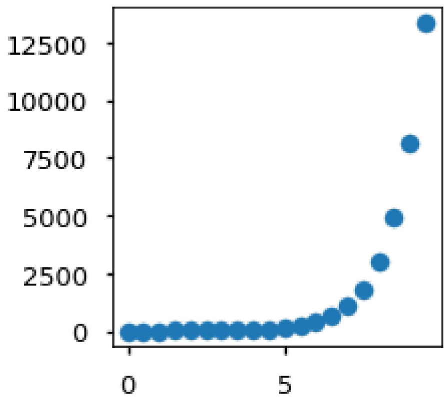
Do you see any features that look like good candidates for this type of transformation?

For reference, a good candidate for this might look like any of these three graphs:

Skbkekas, CC BY-SA 3.0, via Wikimedia Commons



Derek Zelmer, UCSA



Try to find one feature that resembles each of these shapes.

Because this is real-world messy data, none of them are going to match perfectly, and that's ok!

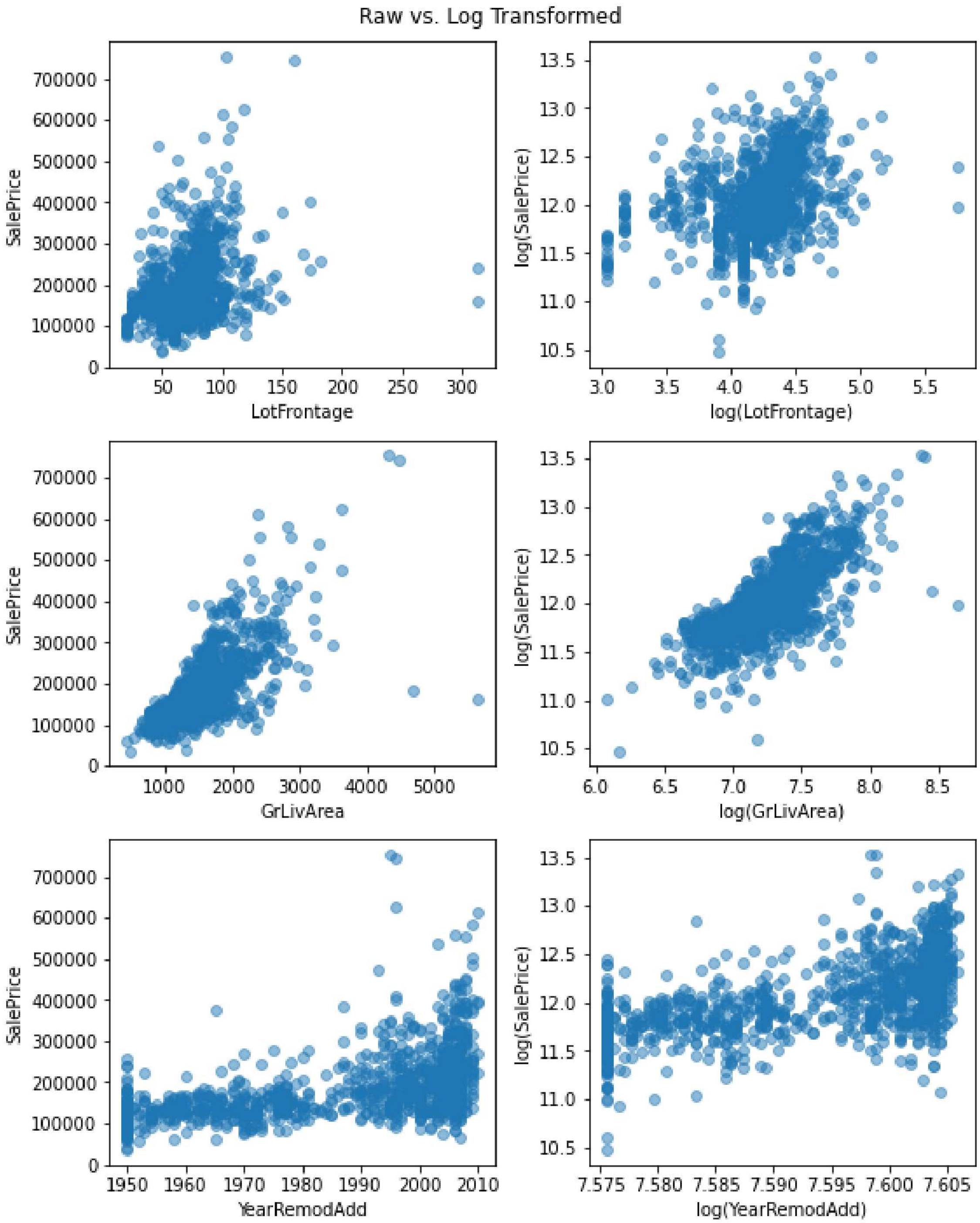
```
"""
LotFrontage resembles the first graph, GrLivArea resembles the second one, and
YearRemodAdd resembles the third one. So these are all potential candidates for
log transformation.
"""
```

Plot Log Transformed Versions of Features

For each feature that you identified as a good candidate for log transformation, plot the feature vs. `SalePrice` as well as the log transformed feature vs. log transformed `SalePrice`.

```
import numpy as np
candidates = ["LotFrontage", "GrLivArea", "YearRemodAdd"]

fig, axes = plt.subplots(ncols=2, nrows=len(candidates), figsize=(8,10))
```

```python
for i, column in enumerate(candidates):
    # Plot raw version
    left_ax = axes[i][0]
    left_ax.scatter(ames[column], y, alpha=0.5)
    left_ax.set_xlabel(column)
    left_ax.set_ylabel("SalePrice")

    # Plot log transformed version
    right_ax = axes[i][1]
    right_ax.scatter(np.log(ames[column]), np.log(y), alpha=0.5)
    right_ax.set_xlabel(f"log({column})")
    right_ax.set_ylabel("log(SalePrice)")

fig.suptitle("Raw vs. Log Transformed")

fig.tight_layout()
```



Do the transformed relationships look more linear? If so, they should be included in the model.

Build a Model with Log-Transformed Features and Target

Data Preparation

Choose up to 3 of the features you investigated, and set up an X dataframe containing the log-transformed versions of these features as well as a y series containing the log-transformed version of the target.

▶ Hint (click to reveal)

```
# We are going to use all 3 of the candidates graphed above
X_log = X[candidates].copy()

X_log.describe()
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	LotFrontage	GrLivArea	YearRemodAdd
count	1121.000000	1121.000000	1121.000000
mean	70.665477	1531.411240	1985.683318
std	24.266812	523.723899	21.025974
min	21.000000	438.000000	1950.000000
25%	60.000000	1155.000000	1966.000000
50%	70.000000	1479.000000	1995.000000
75%	80.000000	1776.000000	2005.000000
max	313.000000	5642.000000	2010.000000

```
# However YearRemodAdd is tricky. A 1% increase in the year means
# roughly a 20-year increase for 20th-21st century dates

# That would result in a very large, difficult-to-interpret coefficient

# Based on the .describe() call above, we see that the latest remodel
# year was 2010

# So let's try subtracting 1910 from YearRemodAdd. So now a 0 means 1910
```

```
# and 1 year = 1% increase
X_log["YearRemodAdd"] = X_log["YearRemodAdd"] - 1910
X_log.describe()
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	LotFrontage	GrLivArea	YearRemodAdd
count	1121.000000	1121.000000	1121.000000
mean	70.665477	1531.411240	75.683318
std	24.266812	523.723899	21.025974
min	21.000000	438.000000	40.000000
25%	60.000000	1155.000000	56.000000
50%	70.000000	1479.000000	85.000000
75%	80.000000	1776.000000	95.000000
max	313.000000	5642.000000	100.000000

```
# Go through and log transform all columns
for column in X_log.columns:
    X_log[f"log_{column}"] = np.log(X_log[column])
    X_log.drop(column, axis=1, inplace=True)

X_log
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}
```

```
.dataframe thead th {
    text-align: right;
}
```

</style>

	log_LotFrontage	log_GrLivArea	log_YearRemodAdd
Id			
1	4.174387	7.444249	4.532599
2	4.382027	7.140453	4.189655
3	4.219508	7.487734	4.521789
4	4.094345	7.448334	4.094345
5	4.430817	7.695303	4.499810
...	...	...	...
1456	4.127134	7.406711	4.499810
1457	4.442651	7.636752	4.356709
1458	4.189655	7.757906	4.564348
1459	4.219508	6.982863	4.454347
1460	4.317488	7.135687	4.007333

1121 rows × 3 columns

```
y_log = np.log(y)
y_log.name = "log_SalePrice"
y_log
```

```
Id
1       12.247694
2       12.109011
3       12.317167
4       11.849398
5       12.429216
          ...
1456    12.072541
1457    12.254863
1458    12.493130
1459    11.864462
```

```
1460    11.901583
Name: log_SalePrice, Length: 1121, dtype: float64
```

Modeling

Now build a StatsModels OLS model with a log-transformed target as well as log-transformed features.

```
import statsmodels.api as sm

model = sm.OLS(y_log, sm.add_constant(X_log))
results = model.fit()
```

Model Evaluation and Interpretation

How did the model perform? How might we interpret its coefficients? Create as many cells as needed.

```
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:          log_SalePrice   R-squared:                       0.688
Model:                            OLS   Adj. R-squared:                  0.687
Method:                 Least Squares   F-statistic:                     820.6
Date:                Mon, 13 Jun 2022   Prob (F-statistic):          8.34e-282
Time:                        12:10:13   Log-Likelihood:                 98.731
No. Observations:                1121   AIC:                            -189.5
Df Residuals:                    1117   BIC:                            -169.4
Df Model:                           3
Covariance Type:            nonrobust
=====================================================================================
                       coef    std err          t      P>|t|      [0.025
0.975]
---------------------------------------------------------------------------------
---
const                4.2715      0.160     26.740      0.000       3.958
4.585
log_LotFrontage      0.1755      0.020      8.651      0.000       0.136
0.215
log_GrLivArea        0.6705      0.023     29.320      0.000       0.626
0.715
log_YearRemodAdd     0.5041      0.022     23.014      0.000       0.461
```

```
0.547
==============================================================================
Omnibus:                      124.227   Durbin-Watson:                   2.076
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              380.156
Skew:                          -0.549   Prob(JB):                     2.82e-83
Kurtosis:                       5.633   Cond. No.                         230.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

```python
# Just for demonstration purposes, to show that we can use the % approximation
# because the betas are small enough:
for column in results.params.index[1:]:
    print(f"""
    {column}
    Approximation:      {results.params[column]}
    More precise value: {(np.exp(results.params[column] * np.log(1.01)) - 1) * 100}
    """)
```

```
    log_LotFrontage
    Approximation:      0.17553002535033485
    More precise value: 0.17481079893331142


    log_GrLivArea
    Approximation:      0.6704791556946889
    More precise value: 0.6693793387537061


    log_YearRemodAdd
    Approximation:      0.5040970917054829
    More precise value: 0.5028533695270898
```

```python
"""
The model explained about 69% of the variance in SalePrice

All coefficients were statistically significant

For each increase of 1% in lot frontage, we see an associated increase
of about 0.2% in sale price
```

```
For each increase of 1% in above-grade living area, we see an associated
increase of about 0.7% in sale price

For each increase of 1 year since 1910 in remodel year, we see an
associated increase of about 0.5% in sale price
"""
```

Summary

Now you have practiced modeling with log transformations! This is a subtle, messy process, so don't be discouraged if this was a tricky lab.

Releases

No releases published

Packages

No packages published

Languages

- **Jupyter Notebook** 100.0%