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

View license

0 stars 5 forks

Star Watch

Code Issues Pull requests Actions Projects Security Insights

solution

This branch is 12 commits ahead, 15 commits behind master.

hoffm386 revise coefficient values in solution markdown ... on Jun 17 16

View code

README.md

Linear Transformations - Lab

Introduction

In this lab, you'll practice your linear transformation skills!

Objectives

You will be able to:

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

Ames Housing Data

Let's look at the Ames Housing data, where each record represents a home sale:

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('seaborn')

ames = pd.read_csv('ames.csv', index_col=0)
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	MSZoning	LotFrontage	LotArea	Street	Alley	Lo
Id							
1	60	RL	65.0	8450	Pave	NaN	Re
2	20	RL	80.0	9600	Pave	NaN	Re
3	60	RL	68.0	11250	Pave	NaN	IR1
4	70	RL	60.0	9550	Pave	NaN	IR1
5	60	RL	84.0	14260	Pave	NaN	IR1
...	...	...	...	...	...	...	...
1456	60	RL	62.0	7917	Pave	NaN	Re
1457	20	RL	85.0	13175	Pave	NaN	Re
1458	70	RL	66.0	9042	Pave	NaN	Re
1459	20	RL	68.0	9717	Pave	NaN	Re

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	Lo
Id							
1460	20	RL	75.0	9937	Pave	NaN	Re

1460 rows × 80 columns

We'll use this subset of features. These are specifically the *continuous numeric* variables, which means that we'll hopefully have meaningful mean values.

From the data dictionary (`data_description.txt`):

```
LotArea: Lot size in square feet

MasVnrArea: Masonry veneer area in square feet

TotalBsmtSF: Total square feet of basement area

GrLivArea: Above grade (ground) living area square feet

GarageArea: Size of garage in square feet
```

```
ames = ames[[
    "LotArea",
    "MasVnrArea",
    "TotalBsmtSF",
    "GrLivArea",
    "GarageArea",
    "SalePrice"
]].copy()
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>

	LotArea	MasVnrArea	TotalBsmtSF	GrLivArea	GarageArea	SalePr
Id						
1	8450	196.0	856	1710	548	20850
2	9600	0.0	1262	1262	460	18150
3	11250	162.0	920	1786	608	22350
4	9550	0.0	756	1717	642	14000
5	14260	350.0	1145	2198	836	25000
...	...	...	...	...	...	...
1456	7917	0.0	953	1647	460	17500
1457	13175	119.0	1542	2073	500	21000
1458	9042	0.0	1152	2340	252	26650
1459	9717	0.0	1078	1078	240	14212
1460	9937	0.0	1256	1256	276	14750

1460 rows × 6 columns

We'll also drop any records with missing values for any of these features:

```
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;
}
```

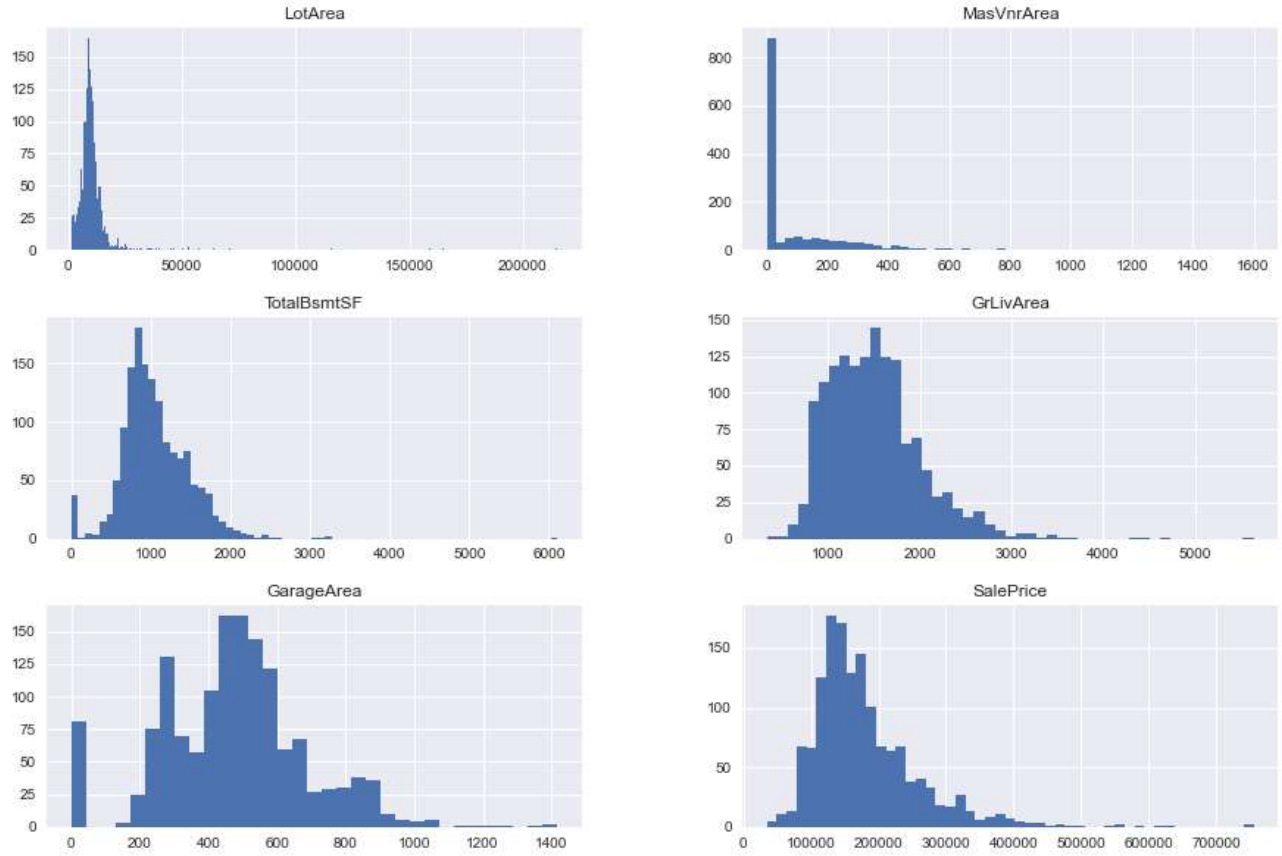
</style>

	LotArea	MasVnrArea	TotalBsmtSF	GrLivArea	GarageArea	SalePr
Id						
1	8450	196.0	856	1710	548	20850
2	9600	0.0	1262	1262	460	18150
3	11250	162.0	920	1786	608	22350
4	9550	0.0	756	1717	642	14000
5	14260	350.0	1145	2198	836	25000
...	...	...	...	...	...	...
1456	7917	0.0	953	1647	460	17500
1457	13175	119.0	1542	2073	500	21000
1458	9042	0.0	1152	2340	252	26650
1459	9717	0.0	1078	1078	240	14212
1460	9937	0.0	1256	1256	276	14750

1452 rows × 6 columns

And plot the distributions of the un-transformed variables:

```
ames.hist(figsize=(15,10), bins="auto");
```



Step 1: Build an Initial Linear Regression Model

`SalePrice` should be the target, and all other columns in `ames` should be predictors.

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

<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>

	LotArea	MasVnrArea	TotalBsmtSF	GrLivArea	GarageArea
Id					
1	8450	196.0	856	1710	548
2	9600	0.0	1262	1262	460
3	11250	162.0	920	1786	608
4	9550	0.0	756	1717	642
5	14260	350.0	1145	2198	836
...	...	...	...	...	...
1456	7917	0.0	953	1647	460
1457	13175	119.0	1542	2073	500
1458	9042	0.0	1152	2340	252
1459	9717	0.0	1078	1078	240
1460	9937	0.0	1256	1256	276

1452 rows × 5 columns

```
import statsmodels.api as sm

initial_model = sm.OLS(y, sm.add_constant(X))
initial_results = initial_model.fit()

print(initial_results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              SalePrice   R-squared:                       0.676
Model:                            OLS   Adj. R-squared:                  0.675
Method:                 Least Squares   F-statistic:                     603.0
Date:                Wed, 18 May 2022   Prob (F-statistic):               0.00
Time:                        19:14:48   Log-Likelihood:                -17622.
No. Observations:                1452   AIC:                         3.526e+04
Df Residuals:                    1446   BIC:                         3.529e+04
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
```

```
------------------------------------------------------------------------------
const       -1.525e+04   4145.934     -3.677      0.000   -2.34e+04   -7113.396
LotArea         0.2568      0.125      2.056      0.040       0.012       0.502
MasVnrArea     55.0481      7.427      7.412      0.000      40.480      69.616
TotalBsmtSF    44.1640      3.324     13.286      0.000      37.643      50.685
GrLivArea      63.8443      2.772     23.030      0.000      58.406      69.282
GarageArea     93.4629      6.795     13.755      0.000      80.134     106.792
==============================================================================
Omnibus:                      817.744   Durbin-Watson:                   1.991
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            77147.499
Skew:                          -1.709   Prob(JB):                         0.00
Kurtosis:                      38.546   Cond. No.                     5.09e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 5.09e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Step 2: Evaluate Initial Model and Interpret Coefficients

Describe the model performance overall and interpret the meaning of each predictor coefficient. Make sure to refer to the explanations of what each feature means from the data dictionary!

▶ **Answer (click to reveal)**

Step 3: Express Model Coefficients in Metric Units

Your stakeholder gets back to you and says this is great, but they are interested in metric units.

Specifically they would like to measure area in square meters rather than square feet.

Report the same coefficients, except using square meters. You can do this by building a new model, or by transforming just the coefficients.

The conversion you can use is **1 square foot = 0.092903 square meters**.

```
# New model approach
X_metric = X.copy()

# All of the features are measured in square feet, so apply the same transformation
```

```
for col in X_metric.columns:
    X_metric[col] = X_metric[col] * 0.092903

# One of the features has "SF" in it which is no longer accurate
X_metric.rename(columns={"TotalBsmtSF": "TotalBsmtArea"}, inplace=True)

X_metric
```

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

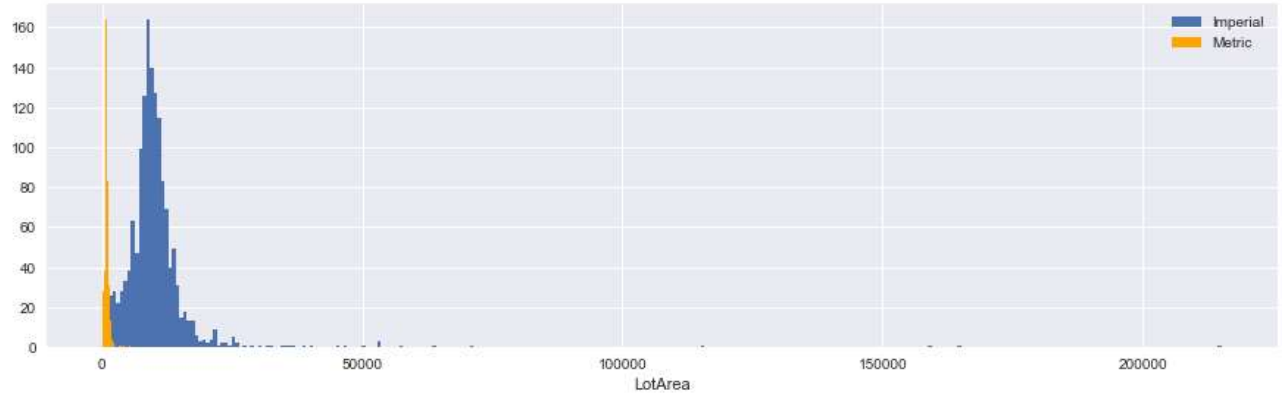
```
.dataframe tbody tr th {
    vertical-align: top;
}

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

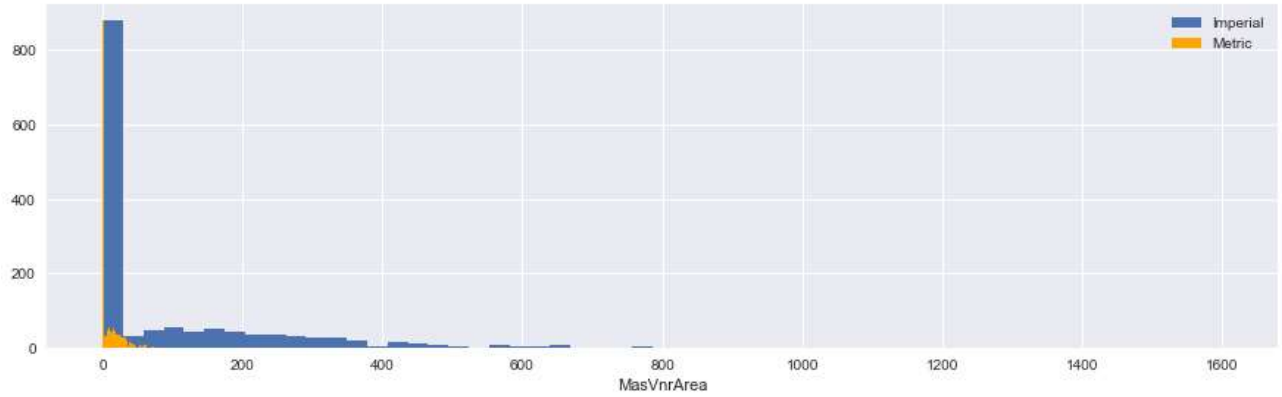
</style>

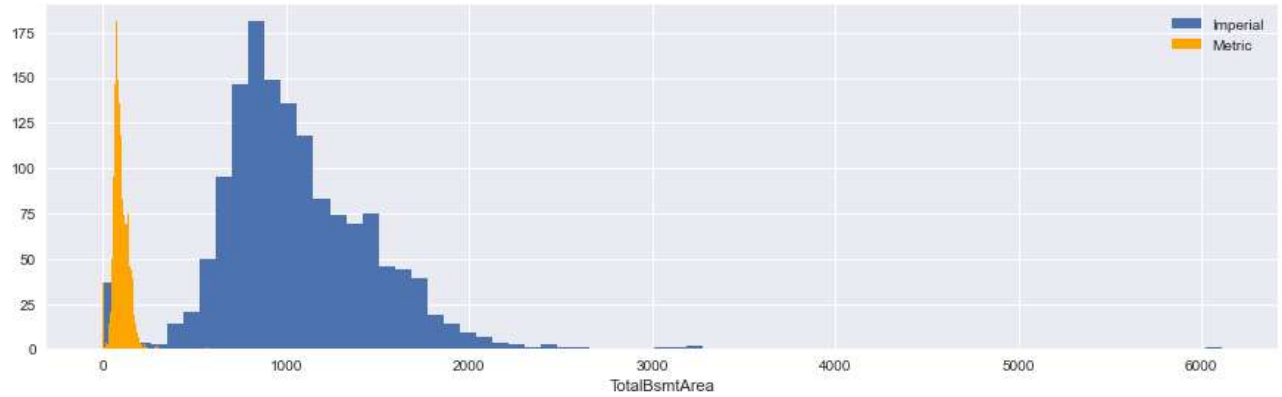
	LotArea	MasVnrArea	TotalBsmtArea	GrLivArea	GarageArea
Id					
1	785.030350	18.208988	79.524968	158.864130	50.910844
2	891.868800	0.000000	117.243586	117.243586	42.735380
3	1045.158750	15.050286	85.470760	165.924758	56.485024
4	887.223650	0.000000	70.234668	159.514451	59.643726
5	1324.796780	32.516050	106.373935	204.200794	77.666908
...	...	...	...	...	...
1456	735.513051	0.000000	88.536559	153.011241	42.735380
1457	1223.997025	11.055457	143.256426	192.587919	46.451500
1458	840.028926	0.000000	107.024256	217.393020	23.411556
1459	902.738451	0.000000	100.149434	100.149434	22.296720
1460	923.177111	0.000000	116.686168	116.686168	25.641228

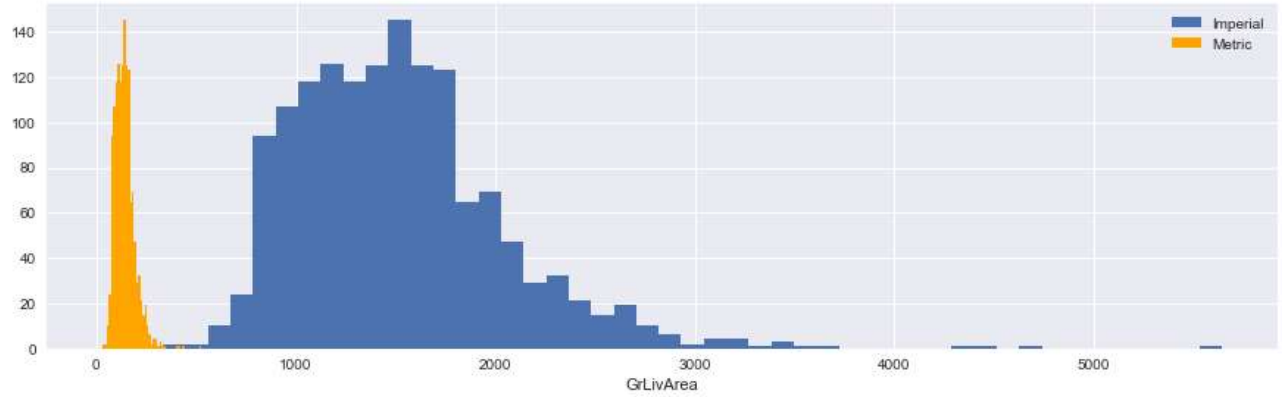
1452 rows × 5 columns

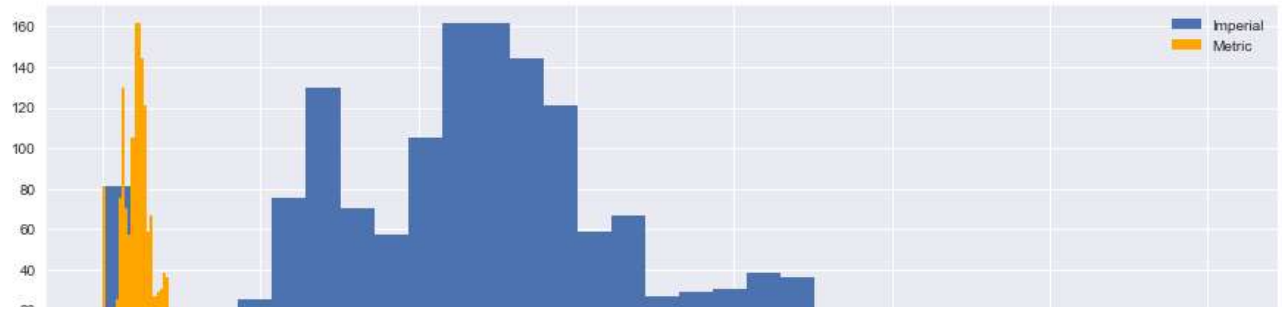
```
fig, axes = plt.subplots(nrows=5, figsize=(15, 25))

for index, col in enumerate(X):
    ax = axes[index]
    ax.hist(X[col], bins="auto", label="Imperial")
    if col == "TotalBsmtSF":
        col = "TotalBsmtArea"
    ax.hist(X_metric[col], bins="auto", label="Metric", color="orange")
    ax.set_xlabel(col)
    ax.legend()
```


Imperial
Metric
LotArea


Imperial
Metric
MasVnrArea


Imperial
Metric
TotalBsmtArea


Imperial
Metric
GrLivArea


Imperial
Metric

```
metric_model = sm.OLS(y, sm.add_constant(X_metric))
metric_results = metric_model.fit()

metric_results.params
```

```
const           -15246.083611
LotArea              2.763844
MasVnrArea         592.532631
TotalBsmtArea      475.377849
GrLivArea          687.214850
GarageArea        1006.027003
dtype: float64
```

```
# Transforming initial coefficients approach
# (using [1:] to skip over the intercept)
initial_results.params[1:] / 0.092903
```

```
LotArea           2.763844
MasVnrArea      592.532631
TotalBsmtSF     475.377849
GrLivArea       687.214850
GarageArea     1006.027003
dtype: float64
```

▶ **Answer (click to reveal)**

Step 4: Center Data to Provide an Interpretable Intercept

Your stakeholder is happy with the metric results, but now they want to know what's happening with the intercept value. Negative \$17k for a home with zeros across the board...what does that mean?

Center the data so that the mean is 0, fit a new model, and report on the new intercept.

(It doesn't matter whether you use data that was scaled to metric units or not. The intercept should be the same either way.)

```
X_centered = X_metric.copy()

for col in X_centered.columns:
    X_centered[col] = X_centered[col] - X_centered[col].mean()
```

```
X_centered
```

<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;
}
```

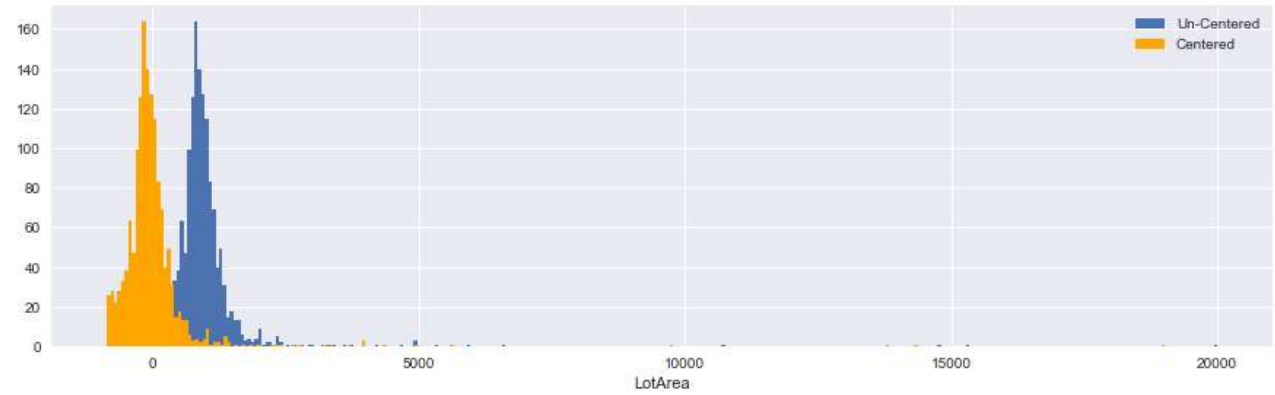
</style>

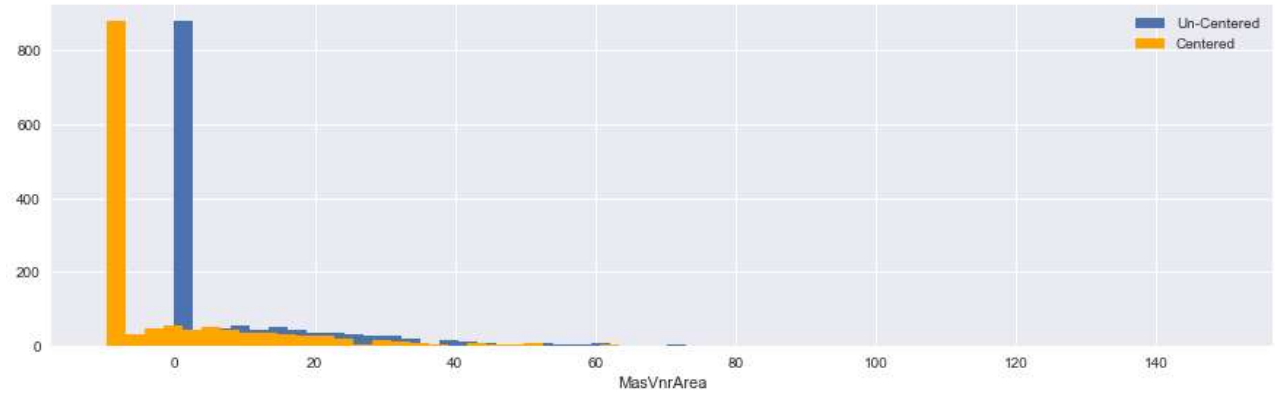
	LotArea	MasVnrArea	TotalBsmtArea	GrLivArea	GarageArea
Id					
1	-191.127128	8.576316	-18.566396	18.200478	7.016480
2	-84.288678	-9.632672	19.152222	-23.420066	-1.158984
3	69.001272	5.417614	-12.620604	25.261106	12.590660
4	-88.933828	-9.632672	-27.856696	18.850799	15.749362
5	348.639302	22.883378	8.282571	63.537142	33.772544
...	...	...	...	...	...
1456	-240.644427	-9.632672	-9.554805	12.347589	-1.158984
1457	247.839547	1.422785	45.165062	51.924267	2.557136
1458	-136.128552	-9.632672	8.932892	76.729368	-20.482808
1459	-73.419027	-9.632672	2.058070	-40.514218	-21.597644
1460	-52.980367	-9.632672	18.594804	-23.977484	-18.253136

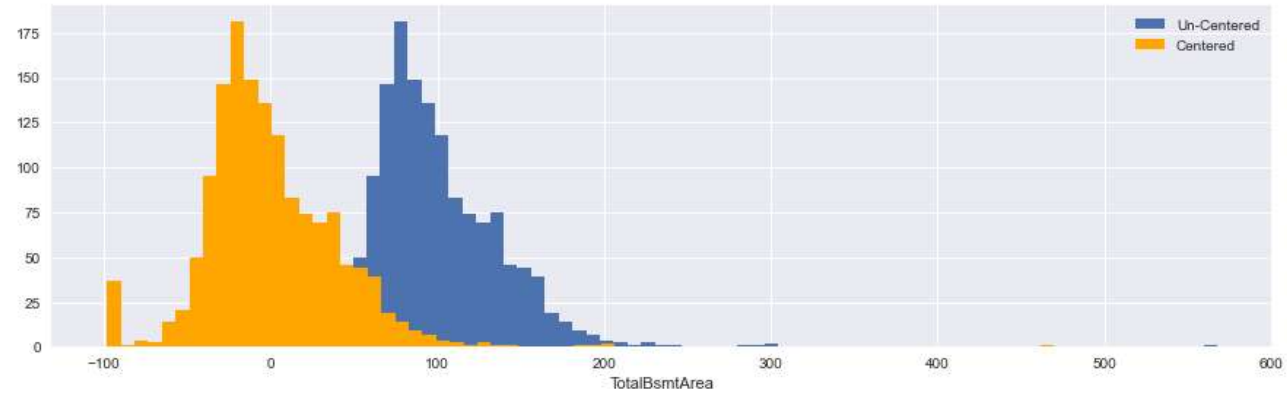
1452 rows × 5 columns

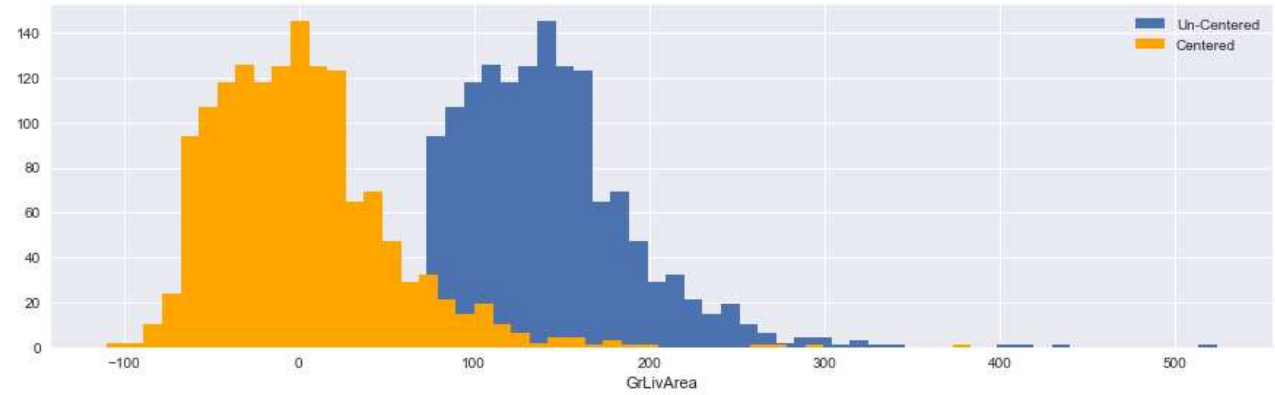
```
fig, axes = plt.subplots(nrows=5, figsize=(15, 25))

for index, col in enumerate(X_metric):
    ax = axes[index]
    ax.hist(X_metric[col], bins="auto", label="Un-Centered")
```

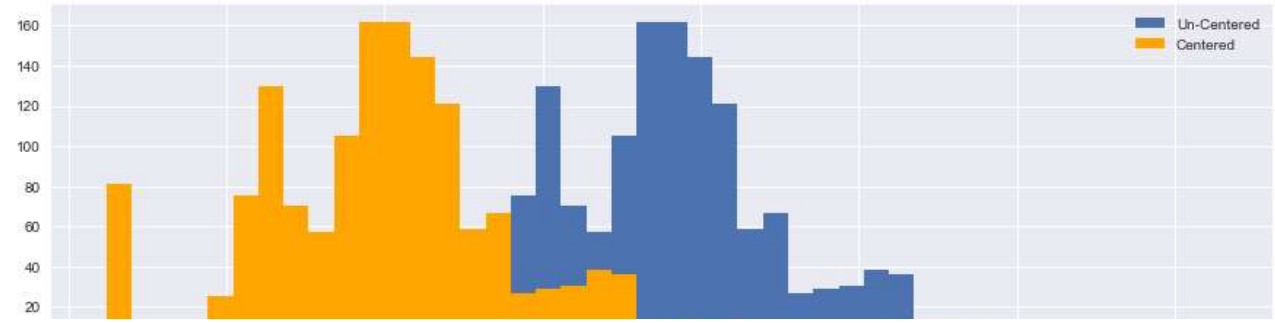
```
        ax.hist(X_centered[col], bins="auto", label="Centered", color="orange")
        ax.set_xlabel(col)
        ax.legend()
```


Un-Centered
Centered
LotArea


Un-Centered
Centered
MasVnrArea


Un-Centered
Centered
TotalBsmtArea


Un-Centered
Centered
GrLivArea


Un-Centered
Centered

```
centered_model = sm.OLS(y, sm.add_constant(X_centered))
centered_results = centered_model.fit()

centered_results.params["const"]
```

```
180615.06336088135
```

▶ Answer (click to reveal)

Step 5: Identify the "Most Important" Feature

Finally, either build a new model with transformed coefficients or transform the coefficients from the Step 4 model so that the most important feature can be identified.

Even though all of the features are measured in area, they are different kinds of area (e.g. lot area vs. masonry veneer area) that are not directly comparable as-is. So apply **standardization** (dividing predictors by their standard deviations) and identify the feature with the highest standardized coefficient as the "most important".

```
# New model approach
X_standardized = X_centered.copy()

# We have already subtracted the mean, just need to divide by std
for col in X_standardized.columns:
    X_standardized[col] = X_standardized[col] / X_standardized[col].std()

X_standardized
```

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

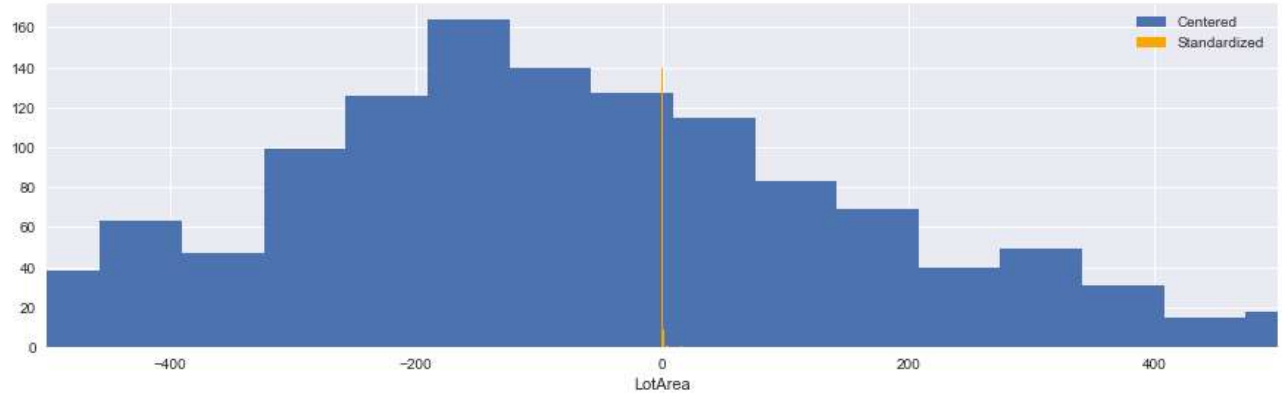
```
.dataframe tbody tr th {
    vertical-align: top;
}

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

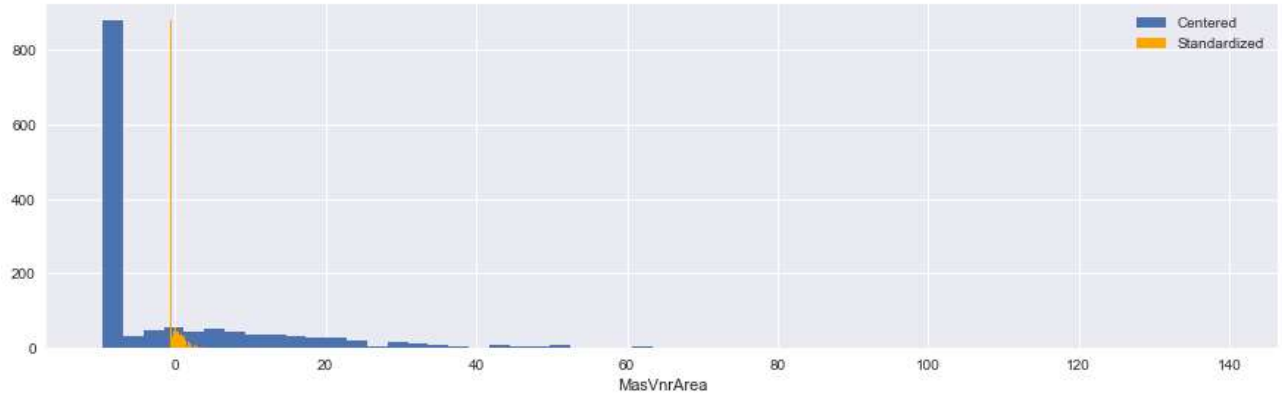
</style>

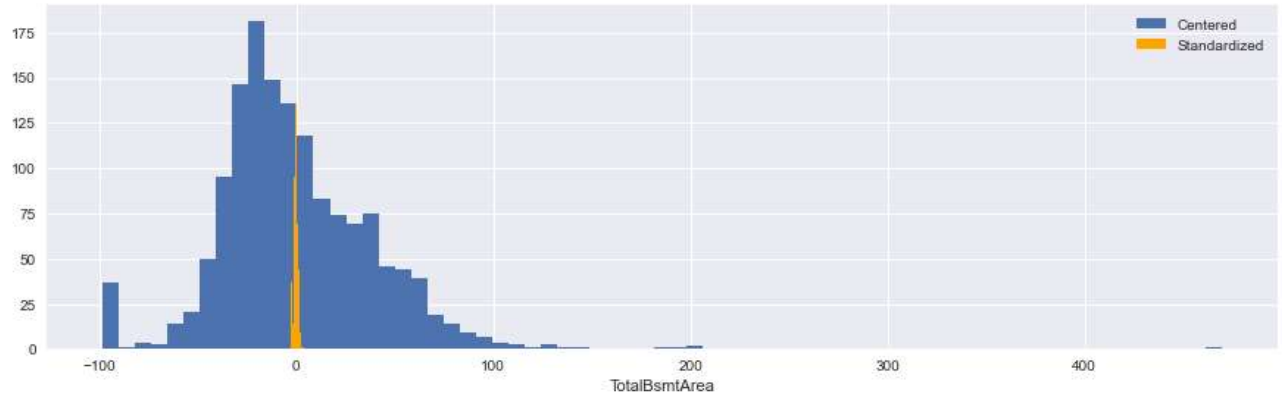
	LotArea	MasVnrArea	TotalBsmtArea	GrLivArea	GarageArea
Id					
1	-0.205943	0.509840	-0.456148	0.372713	0.352744
2	-0.090822	-0.572637	0.470541	-0.479601	-0.058266
3	0.074350	0.322063	-0.310069	0.517302	0.632979
4	-0.095828	-0.572637	-0.684396	0.386031	0.791778
5	0.375664	1.360357	0.203490	1.301127	1.697870
...	...	...	...	...	...
1456	-0.259298	-0.572637	-0.234747	0.252857	-0.058266
1457	0.267051	0.084581	1.109636	1.063316	0.128557
1458	-0.146681	-0.572637	0.219467	1.571280	-1.029746
1459	-0.079110	-0.572637	0.050564	-0.829659	-1.085793
1460	-0.057087	-0.572637	0.456846	-0.491016	-0.917652

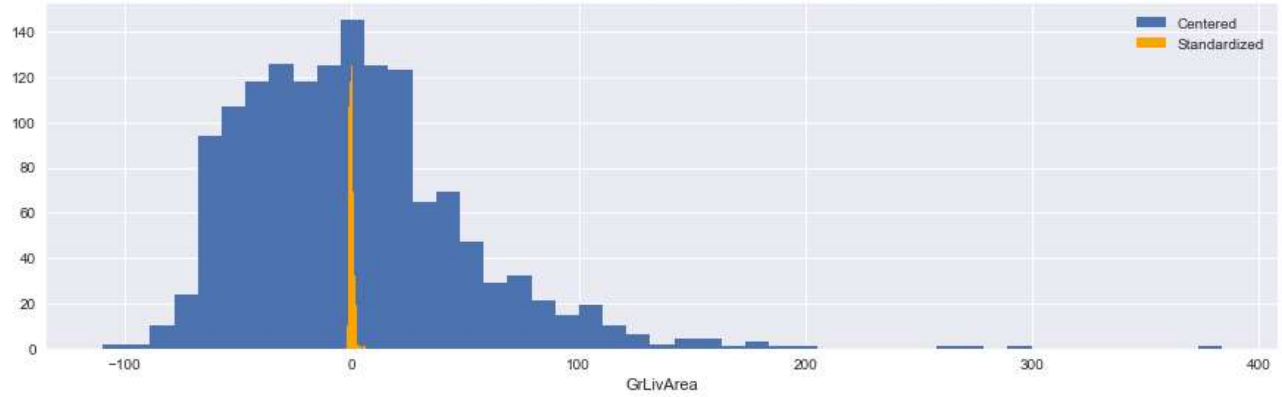
1452 rows × 5 columns

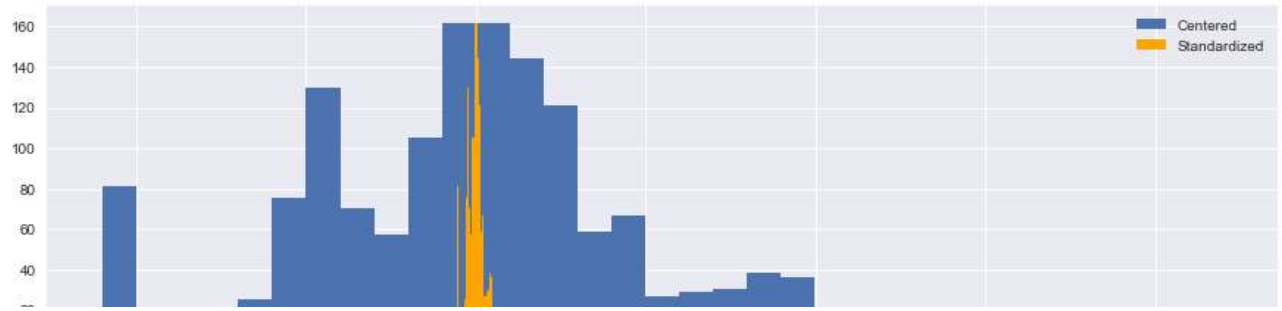
```
fig, axes = plt.subplots(nrows=5, figsize=(15, 25))

for index, col in enumerate(X_standardized):
    ax = axes[index]
    ax.hist(X_centered[col], bins="auto", label="Centered")
    ax.hist(X_standardized[col], bins="auto", label="Standardized", color="orange")
    ax.set_xlabel(col)
    ax.legend()

# Manually adjust LotArea axis because otherwise standardized data is invisible
# (LotArea has a large standard deviation)
axes[0].set_xlim(-500, 500);
```


Centered
Standardized
LotArea


Centered
Standardized
MasVnrArea


Centered
Standardized
TotalBsmtArea


Centered
Standardized
GrLivArea


Centered
Standardized

```
standardized_model = sm.OLS(y, sm.add_constant(X_standardized))
standardized_results = standardized_model.fit()

standardized_results.params
```

```
const            180615.063361
LotArea            2565.014370
MasVnrArea         9967.343227
TotalBsmtArea     19349.103860
GrLivArea         33558.347891
GarageArea        20011.010509
dtype: float64
```

```
# Transforming initial coefficients approach
# (using [1:] to skip over the intercept)
for param in centered_results.params.index[1:]:
    transformed_val = centered_results.params[param] * X_centered[param].std()
    print(f"{param:18}{round(transformed_val, 5):>12}")
```

```
LotArea             2565.01437
MasVnrArea          9967.34323
TotalBsmtArea      19349.10386
GrLivArea          33558.34789
GarageArea         20011.01051
```

▶ Answer (click to reveal)

Summary

Great! You've now got some hands-on practice transforming data and interpreting the results!

Releases

No releases published

Packages

No packages published

Contributors 6

Languages

- **Jupyter Notebook** 94.7%
- **Python** 5.3%