Scaling Numerical Inputs

Most Models Expect Numerical Inputs

- Most modern ML models (e.g., random forests, SVMs, neural networks) operate on **numerical inputs**.
- If your input is already numeric, you can often use it **as-is**.
- Scaling your data can still have benefits!

Why Scaling is Desirable

- Many ML optimizers are tuned for inputs in the **[−1, 1] range**.
- Scaling can:
  - Improve convergence speed.
  - Reduce computational cost.
  - Increase floating point precision.

Why [−1, 1]?

- Gradient descent slows down when the **loss function curvature** is high.
- Features with **larger magnitude** → **larger gradients** → unstable weight updates.
- Scaling helps:
  - Ensure **faster, more stable** training.

Code Example: Training Speed

```
from sklearn import datasets, linear_model
import timeit

diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)
raw = diabetes_X[:, None, 2]
max_raw = max(raw)
min_raw = min(raw)
scaled = (2*raw - max_raw - min_raw)/(max_raw - min_raw)

def train_raw():
    linear_model.LinearRegression().fit(raw, diabetes_y)

def train_scaled():
    linear_model.LinearRegression().fit(scaled, diabetes_y)

raw_time = timeit.timeit(train_raw, number=1000)
scaled_time = timeit.timeit(train_scaled, number=1000)
```

- Result: ~30% speedup just by scaling **one** feature.

More Reasons to Scale

- Some algorithms (e.g., **k-means**, **regularized regression**) are sensitive to **relative magnitudes**.
- Example:
  - Euclidean distance in k-means favors large-magnitude features.
  - L1/L2 penalties shrink weights more aggressively for smaller-magnitude features.

Linear Scaling Methods

1. Min-Max Scaling

```
x_scaled = (2*x - max_x - min_x)/(max_x - min_x)
```

Or just use

```
sklearn.preprocessing.MinMaxScaler
```

- Maps input to $[-1, 1]$.
- **Problem**: min/max are often **outliers** → range compression.

Your task

- Open the starter notebook and load housing data
- Apply a min max scaler to the column `total_rooms`
- Verify the transformed column is in range [-1,1]

2. Clipping with Min-Max Scaling

- Use **reasonable bounds** instead of min and max, not outliers.
- First, clip using e.g. `np.clip`
- Then, use Min-Max Scaling to [-1, 1].
- Treats outliers as –1 or 1.

Warning:
This is not the same as

```
sklearn.preprocessing.MinMaxScaler(clip=True)
```

Your task

- Plot a histogram of `total_rooms`
- Estimate reasonable bounds visually
- Clip the column
- Apply a min max scaler
- Plot a histogram of the transformed column
- If you are quick, find out what `clip=True` does in Scikit Learn's Min Max Scaler

3. Z-Score Normalization

```
x_scaled = (x - mean_x)/std_x
```

Or just use

```
sklearn.preprocessing.StandardScaler
```

- Addresses the problem of outliers without requiring prior knowledge of reasonable range
- Centers data to zero mean, unit variance.
- Effective for **normally distributed** features.
- Not bounded in [−1, 1], but most values lie within.

Your task

- Apply z-score normalization to the column `total_rooms`
- Plot a histogram of the transformed column

4. Winsorizing

- Clip values using **empirical percentiles** (e.g., 10th and 90th).
- Then apply **min-max scaling**.
- Like clipping, but based on **data distribution**.

Summary of Linear Scaling Methods

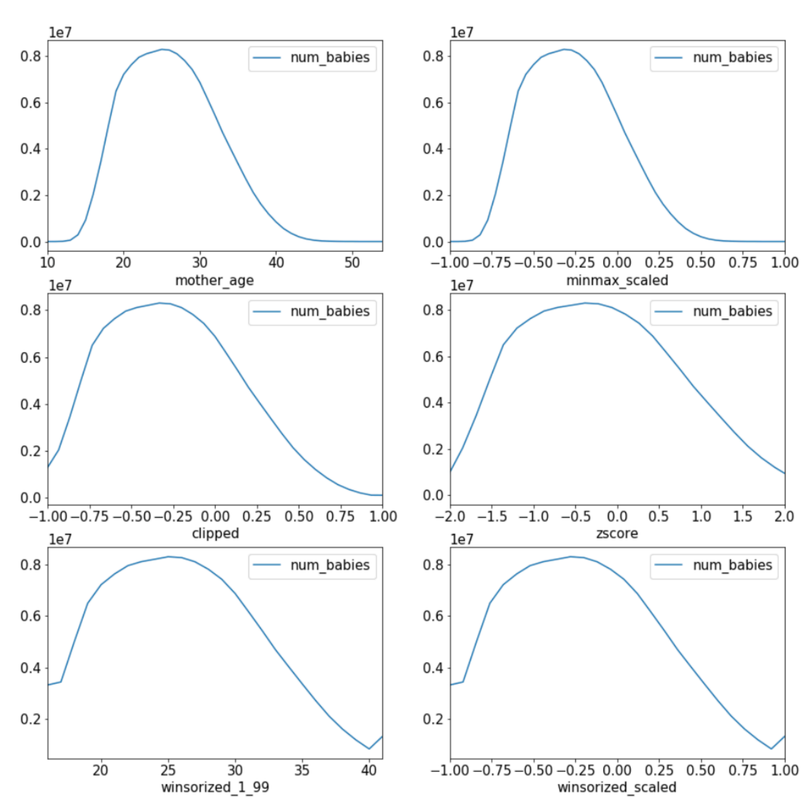
Method	Best For	Notes
Min-Max	Uniform distributions	Sensitive to outliers
Clipping	Uniform + outliers	Requires good thresholds
Z-Score	Normal distributions	Unbounded output
Winsorizing	Any distribution	Data-driven outlier handling

Don't Throw Away Outliers

- Valid outliers (e.g., age = 50) should be **handled**, not discarded.
- Clipping preserves their presence as –1 or 1.
- Throw away only **invalid inputs** (e.g., negative ages).

Example: Mother Age Feature

- **Min-Max**: keeps scale but keeps sparse edge values.
- **Clipping**: rolls up extreme values, needs tuning.
- **Winsorizing**: percentile-based, but also needs tuning.
- **Z-Score**: spreads out values, best for bell-shaped distribution.



Watch Out for Training-Serving Skew

- **Scaling must be consistent** between training and production.
- Never recompute statistics (mean, min, max) during serving!

Good Practice:

- Learn scaling parameters (e.g., min/max or mean/std) on the **training set**.
- **Store** them with the model.
- **Reuse** them for all future inputs in production.
- Scikit Learn's API has this built in, using the `.fit()` and `.transform()` paradigmas

Key Takeaways

- Always consider scaling for numeric inputs.
- Choose a scaling method based on:
  - The feature distribution.
  - Model sensitivity.
- Never discard **valid outliers**.
- Avoid **training-serving skew** by using saved transformation parameters.

Counts

Dealing with Counts

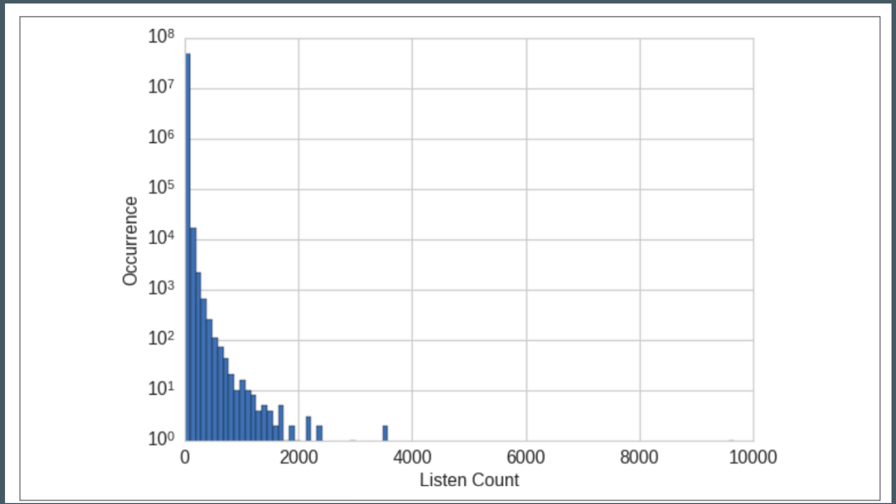
- Counts can quickly accumulate without bound
- Very likely to contain a few extreme values

Three options

- Keep raw numbers
- Convert into binary values to indicate presence
- Bin into coarser granularity

Dealing with Counts: Binarization

Example: How often does a song get listendend to?

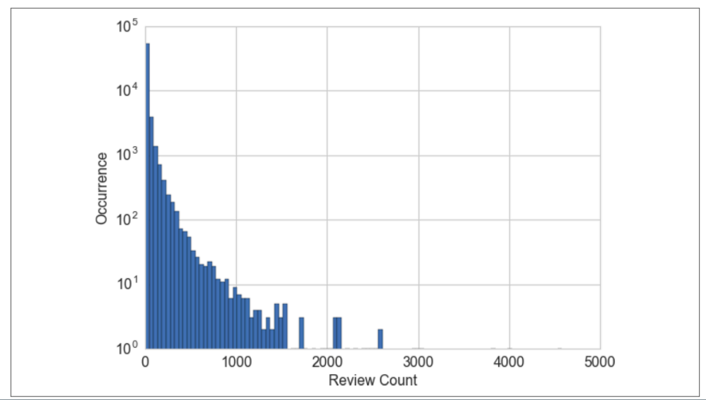


- 99% of listen counts are 24 and lower
- Raw listen counts does not seem to be robust indicator of taste
- Someone who listens to a song 20 times does notlike it twice as much as someone else who listens to it 10 times

Solution: Clip everything larger than 1 to 1

Dealing with Counts: Binning

Example: Review counts of businesses on Yelp (logarithmic y-scale)



Solution:

- We group the counts into bins
- Map a continuous number to a discrete one
- Think of the discretized numbers as an ordered sequence of bins that represent a measure of intensity

Two basic options: Fixed width binning and Quantile binning

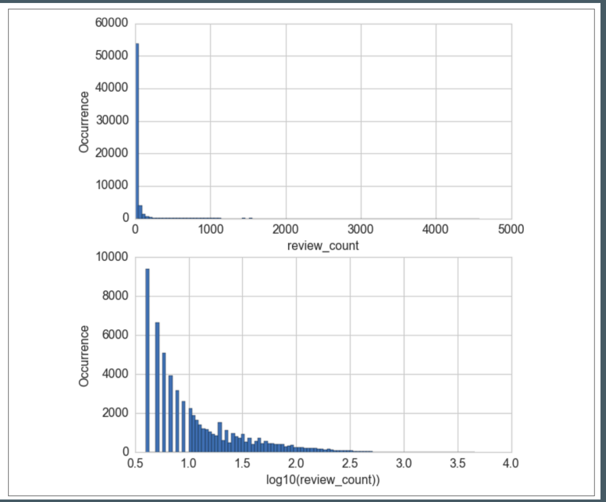
Your task

- Research the pandas `qcut` method
- Replace the counts in the Yelp dataset by there deciles

Log Transformation

Features of the Log Transformation

- Is a powerful tool for dealing with positive numbers with a heavy-tailed distribution
- Compresses the long tail in the high end of the distribution into a shorter tail
- Expands the low end into a longer head



Your task

- Reproduce the graph of the log transformation for the counts in the Yelp dataset

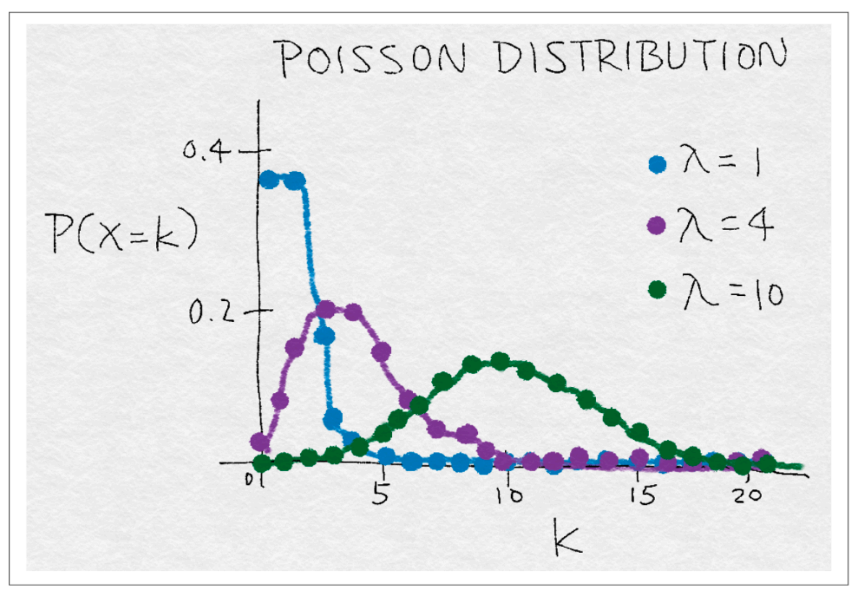
Power Transforms

Nonlinear Transformations and Techniques

What if the data is skewed, not uniformly distributed, and not normally distributed?

- Apply **nonlinear transformations** before scaling
- Common choices:
  - `log(x)`
  - `x^0.25`, `sqrt(x)`, `x^2`, etc.
  - `sigmoid(x)`
- Goals:
  - make the distribution **bell-shaped** or **uniform**
  - Overall goal: stabilize variance (homoscedastic)

Example: Data from multiple Poisson distributions

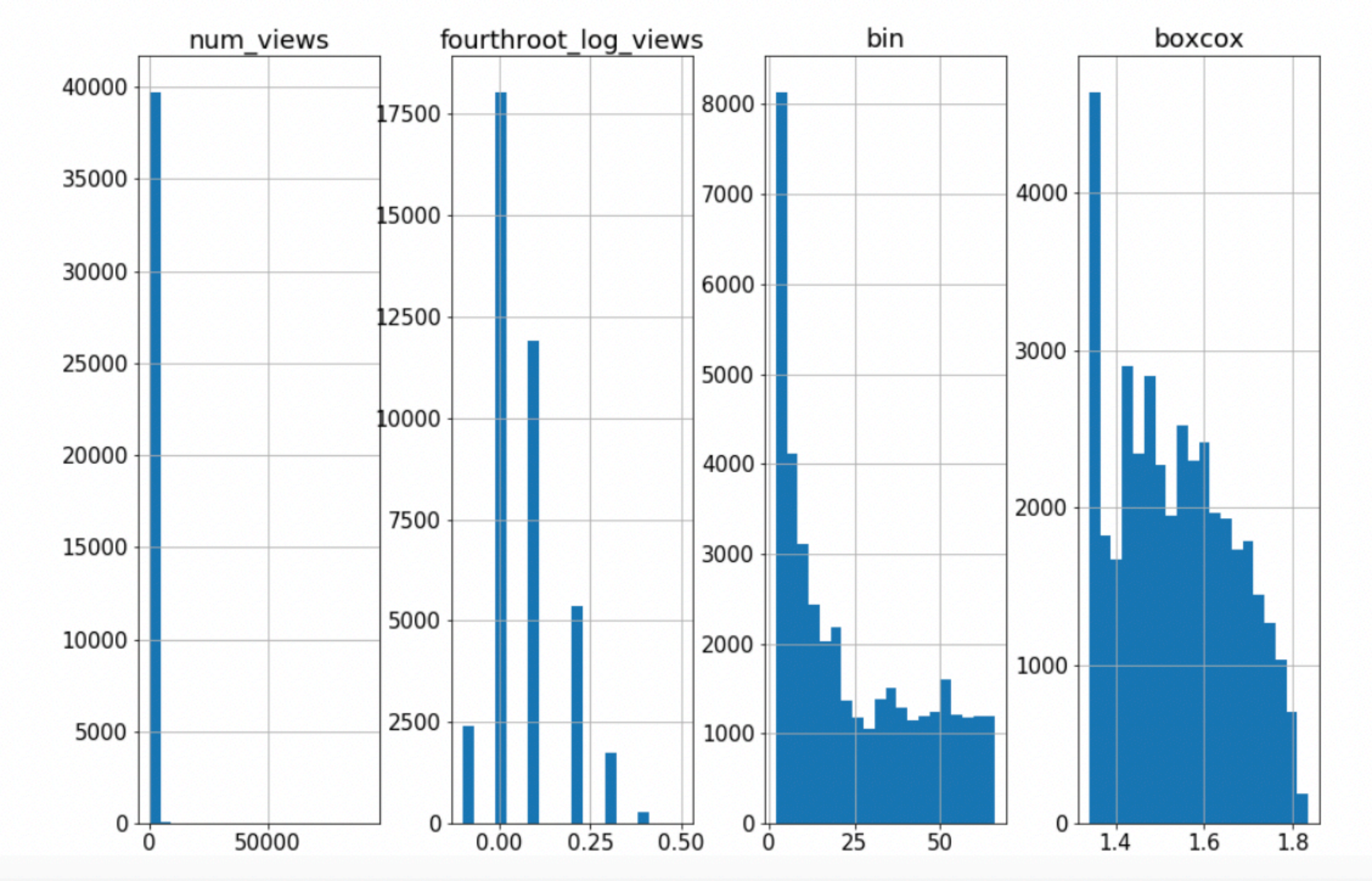


If the data comes from multiple Poisson distributions with different means, the variance will not be stable

Real life example: Wikipedia page views

The variance among large page views will be significantly different from variance amoung small page views

Example: Wikipedia Page Views



Result: roughly bell-shaped distribution

Box-Cox Transform

- Parametric nonlinear transform
- Equalizes **variance across ranges**
- Useful when variance depends on magnitude (heteroscedasticity)

Example: The variance among rarely viewed Wikipedia pages will be much smaller than the variance among frequently viewed pages

Python Example:

```python
from scipy.stats import boxcox
traindf[' boxcox'], est_lambda = boxcox(traindf['num_views'])
evaldf['boxcox'] = boxcox(evaldf['num_views'], est_lambda)
```

Your task

- Apply a Boxcox transform to the column `total_rooms` in the housing dataset
- Plot a histogram before and after

Other kinds of transformations

Logit Transformations

Use when input values are constrained to the **(0, 1)** range.

- Applies the inverse of the sigmoid function:

```
logit(x) = log(x / (1 - x))
```

- Useful for probabilities, normalized scores, ratios, etc.

Why?

- Expands small values close to 0 or 1.
- Makes input more **symmetric and linear**

⚠️ Requires: `0 < x < 1` (apply clipping if needed)

Hinge Functions for Splitting Numeric Features

Hinge functions divide numerical inputs into regions using a threshold:

```
f(x) = max(0, x - c)   # right hinge
f(x) = max(0, c - x)   # left hinge
```

- `c` is a **cutoff value** (e.g., age = 40)
- Turns one continuous feature into **piecewise-linear segments**

Why?

- Allows models to capture different behavior above/below a threshold
- Especially useful in linear models, decision trees

Can be combined with other transforms for rich feature sets

Handling Arrays of Numbers

Sometimes a feature is a **list of numbers**, e.g.:

```
[2100, 15200, 230000, 1200, 300, 532100]
```

Sales of prior books on a topic

Problem: array is of **variable length**

Converting Arrays to Fixed-Length Features

Common strategies:

1. Bulk statistics: mean, median, min, max, count
2. Empirical percentiles: 10th, 25th, 75th...
3. Fixed-length truncation/padding: e.g., last 3 values

Choose based on:

- Whether ordering matters
- Whether long tails or extreme values are important