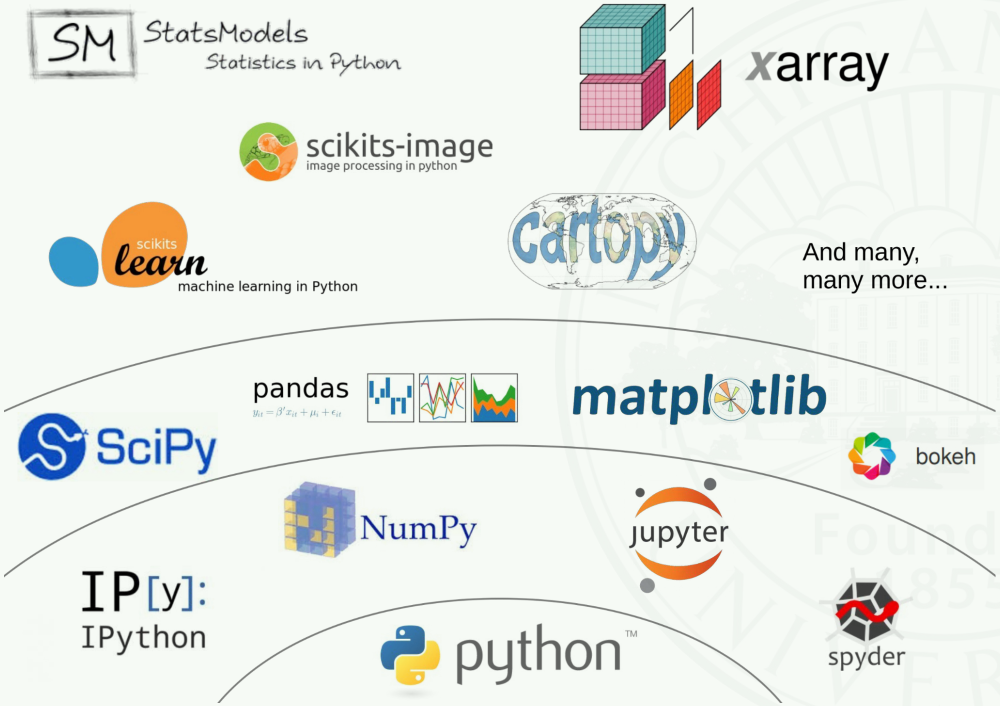
# Scaling, Encoding and Imputing

**Prof. Murillo**
Computational Mathematics, Science and Engineering
Michigan State University

# Where are we?




We have explored:

- what data science is
- types of data
- some of the Python ecosystem for data science
  - Pandas
  - Seaborn
  - Streamlit

# Detailed Processing of Data

Data preprocessing:

- scaling
  - normalizing
  - transforming
- encoding
- imputing

# Scaling

Raw data typically contains numerical values that are in different ranges.

| | mpg | cylinders | displacement | horsepower | weight | acceleration | model_year | origin | name |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 18.0 | 8 | 307.0 | 130.0 | 3504 | 12.0 | 70 | usa | chevrolet chevelle malibu |
| 1 | 15.0 | 8 | 350.0 | 165.0 | 3693 | 11.5 | 70 | usa | buick skylark 320 |
| 2 | 18.0 | 8 | 318.0 | 150.0 | 3436 | 11.0 | 70 | usa | plymouth satellite |
| 3 | 16.0 | 8 | 304.0 | 150.0 | 3433 | 12.0 | 70 | usa | amc rebel sst |
| 4 | 17.0 | 8 | 302.0 | 140.0 | 3449 | 10.5 | 70 | usa | ford torino |
| 5 | 15.0 | | | | | | | | ord galaxie 500 |
| 6 | 14.0 | | | | | | | | hevrolet impala |
| 7 | 14.0 | | | | | | | | lymouth fury iii |
| 8 | 14.0 | | | | | | | | ontiac catalina |
| 9 | 15.0 | | | | | | | | mbassador dpl |

```
1 df_m.describe()
```

| | mpg | cylinders | displacement | horsepower | weight | acceleration | model_year |
|---|---|---|---|---|---|---|---|
| count | 398.000000 | 398.000000 | 398.000000 | 392.000000 | 398.000000 | 398.000000 | 398.000000 |
| mean | 23.514573 | 5.454774 | 193.425879 | 104.469388 | 2970.424623 | 15.568090 | 76.010050 |
| std | 7.815984 | 1.701004 | 104.269838 | 38.491160 | 846.841774 | 2.757689 | 3.697627 |
| min | 9.000000 | 3.000000 | 68.000000 | 46.000000 | 1613.000000 | 8.000000 | 70.000000 |
| 25% | 17.500000 | 4.000000 | 104.250000 | 75.000000 | 2223.750000 | 13.825000 | 73.000000 |
| 50% | 23.000000 | 4.000000 | 148.500000 | 93.500000 | 2803.500000 | 15.500000 | 76.000000 |
| 75% | 29.000000 | 8.000000 | 262.000000 | 126.000000 | 3608.000000 | 17.175000 | 79.000000 |
| max | 46.600000 | 8.000000 | 455.000000 | 230.000000 | 5140.000000 | 24.800000 | 82.000000 |

When modeling, the `weight` column may dominate the other features.

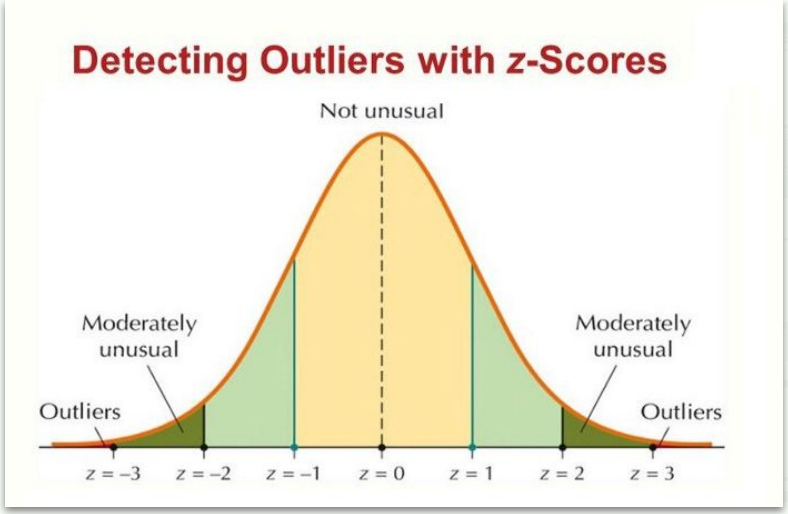
# Z Scaling

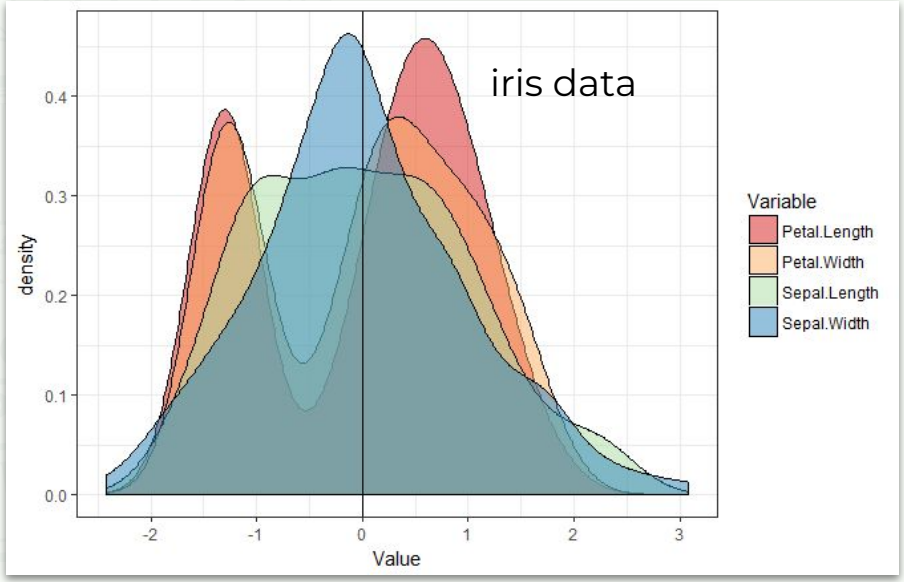
The standard way to scale data is through the *z* (standard) score:

$$z = \frac{x - \mu}{\sigma}$$

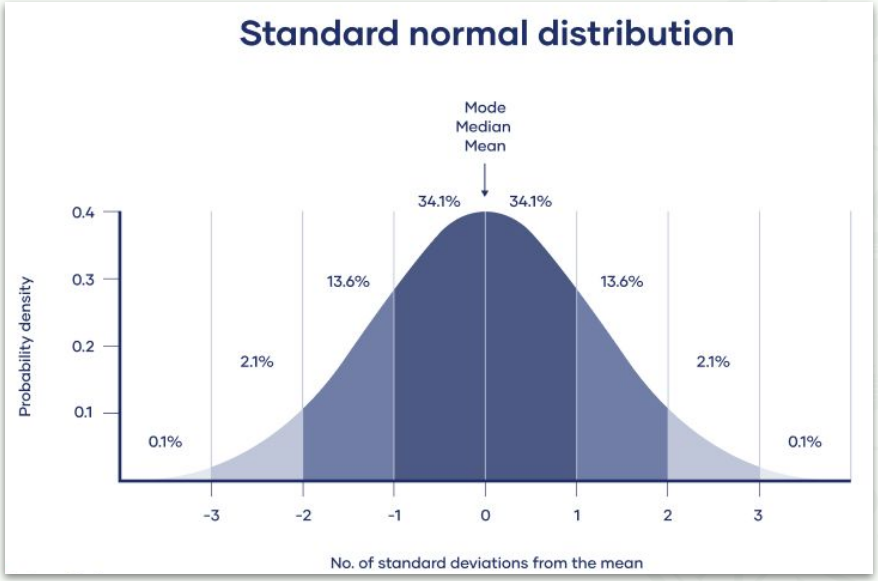
$\mu$ = mean

$\sigma$ = standard deviation



# Z Scaling: Connection to Gaussian/Normal




$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2}$$

Although there is this interesting connection to the normal distribution, we apply the z-score to any distribution.

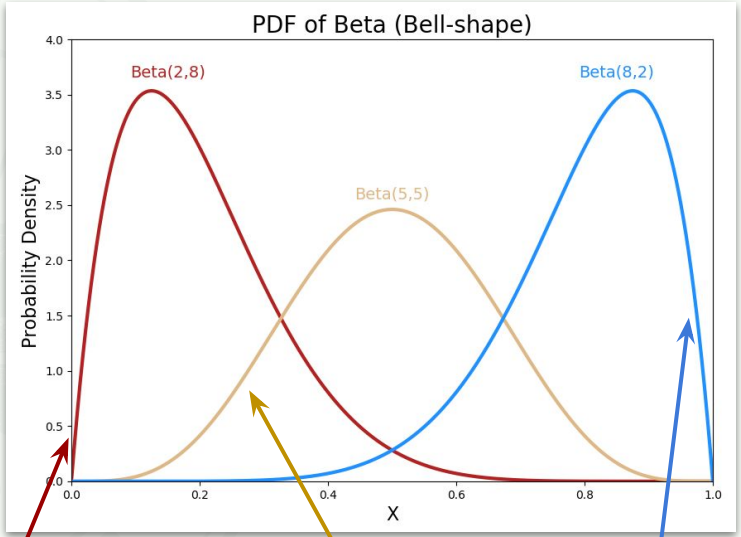
# z-scaling Ranking Example

A committee is formed to read and rank submitted proposals.

Rankings are data used by a Program Officer to select proposals to be funded.

The z-score rescales these distributions so that they have comparable meanings.

Problem: people have personal approaches to ranking.

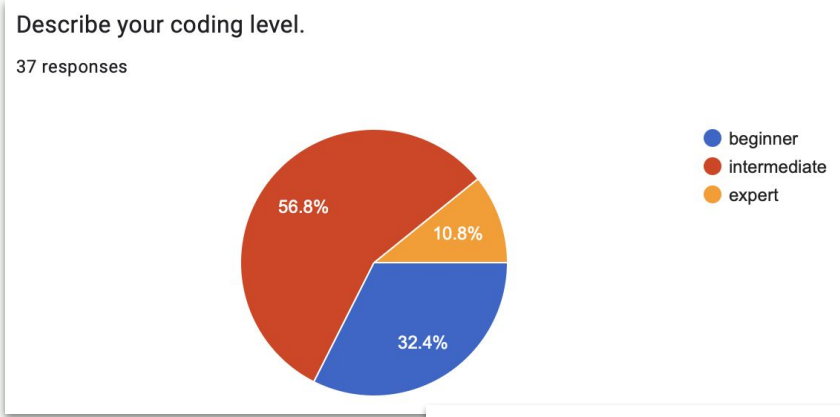


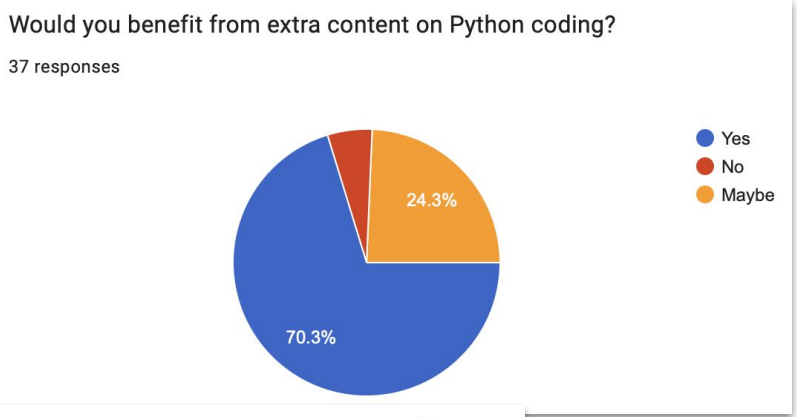

Tough person gives proposals low scores.

Balanced person gives a range of scores.

Nice person gives proposals high scores.

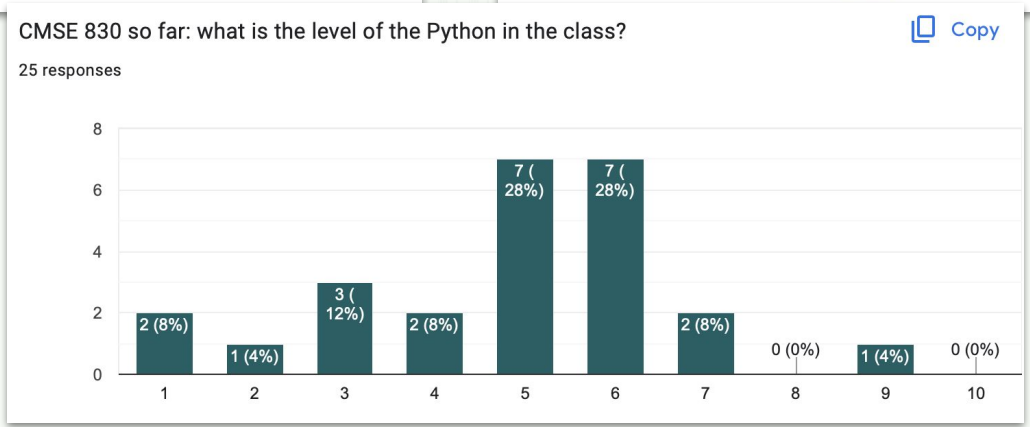
# z-scaling Python Skill Example




The z-score could "re-normalize" the data.

But, I didn't collect names, so we can't do it with this dataset.



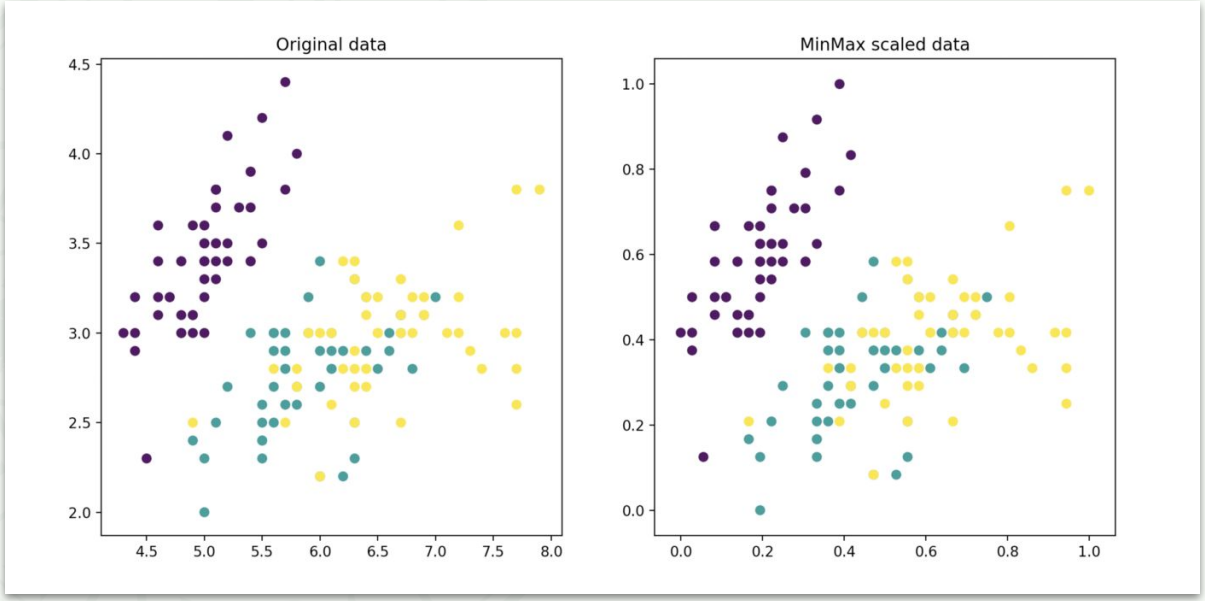
# There are Many Scaling Methods

**MinMax** scaler maps the data into *[0,1]*.

This can be advantageous when you can't handle negative values.

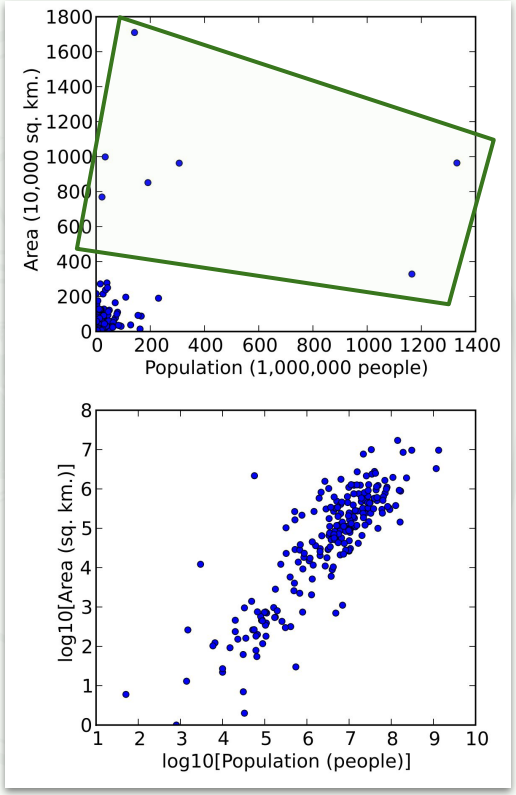
For example, you will take the logarithm of the data.

How you scale/normalize ( or not) is problem dependent.



# Transforming

Transforming is a related operation, and might be used with scaling.

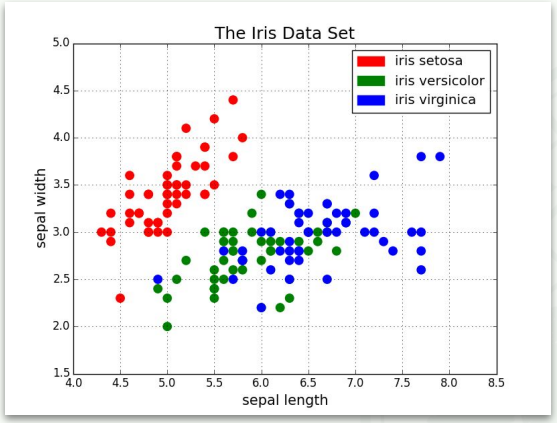



models will be dominated by these values

*z*-scaling doesn't help here

log-log transformed: much more balanced

# Encoding




How do we handle categorical data?

Encoding maps the category onto a numerical value.

There are many ways to do this - very problem dependent.

- **label encoder**: encode target labels with *[0,N-1]* *(lossy)*
- **ordinal encoder**: features converted to ordinal numbers *[0,N-1]*
- **one-hot encoder**: creates binary feature for each category

# Compare Label and One-Hot Encoding

## Label Encoding

| Food Name | Categorical # | Calories |
|---|---|---|
| Apple | 1 | 95 |
| Chicken | 2 | 231 |
| Broccoli | 3 | 50 |

## One Hot Encoding

| Apple | Chicken | Broccoli | Calories |
|---|---|---|---|
| 1 | 0 | 0 | 95 |
| 0 | 1 | 0 | 231 |
| 0 | 0 | 1 | 50 |

# Missing Data: What To Do???

Let's examine the mpg dataset:

| | mpg | cylinders | displacement | horsepower | weight | acceleration | model_year | origin | name |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 18.0 | 8 | 307.0 | 130.0 | 3504 | 12.0 | 70 | usa | chevrolet chevelle malibu |
| 1 | 15.0 | 8 | 350.0 | 165.0 | 3693 | 11.5 | 70 | usa | buick skylark 320 |
| 2 | 18.0 | 8 | 318.0 | 150.0 | 3436 | 11.0 | 70 | usa | plymouth satellite |
| 3 | 16.0 | 8 | 304.0 | 150.0 | 3433 | 12.0 | 70 | usa | amc rebel sst |
| 4 | 17.0 | 8 | 302.0 | 140.0 | 3449 | 10.5 | 70 | usa | ford torino |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 393 | 27.0 | 4 | 140.0 | 86.0 | 2790 | 15.6 | 82 | usa | ford mustang gl |
| 394 | 44.0 | 4 | 97.0 | 52.0 | 2130 | 24.6 | 82 | europe | vw pickup |
| 395 | 32.0 | 4 | 135.0 | 84.0 | 2295 | 11.6 | 82 | usa | dodge rampage |
| 396 | 28.0 | 4 | 120.0 | 79.0 | 2625 | 18.6 | 82 | usa | ford ranger |
| 397 | 31.0 | 4 | 119.0 | 82.0 | 2720 | 19.4 | 82 | usa | chevy s-10 |

398 rows × 9 columns

# .describe()

| | mpg | cylinders | displacement | horsepower | weight | acceleration | model_year |
|---|---|---|---|---|---|---|---|
| count | 398.000000 | 398.000000 | 398.000000 | 392.000000 | 398.000000 | 398.000000 | 398.000000 |
| mean | 23.514573 | 5.454774 | 193.425879 | 104.469388 | 2970.424623 | 15.568090 | 76.010050 |
| std | 7.815984 | 1.701004 | 104.269838 | 38.491160 | 846.841774 | 2.757689 | 3.697627 |
| min | 9.000000 | 3.000000 | 68.000000 | 46.000000 | 1613.000000 | 8.000000 | 70.000000 |
| 25% | 17.500000 | 4.000000 | 104.250000 | 75.000000 | 2223.750000 | 13.825000 | 73.000000 |
| 50% | 23.000000 | 4.000000 | 148.500000 | 93.500000 | 2803.500000 | 15.500000 | 76.000000 |
| 75% | 29.000000 | 8.000000 | 262.000000 | 126.000000 | 3608.000000 | 17.175000 | 79.000000 |
| max | 46.600000 | 8.000000 | 455.000000 | 230.000000 | 5140.000000 | 24.800000 | 82.000000 |

# .info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64  
 2   displacement  398 non-null    float64
 3   horsepower    392 non-null    float64
 4   weight        398 non-null    int64  
 5   acceleration  398 non-null    float64
 6   model_year    398 non-null    int64  
 7   origin        398 non-null    object 
 8   name          398 non-null    object 
dtypes: float64(4), int64(3), object(2)
memory usage: 28.1+ KB
```

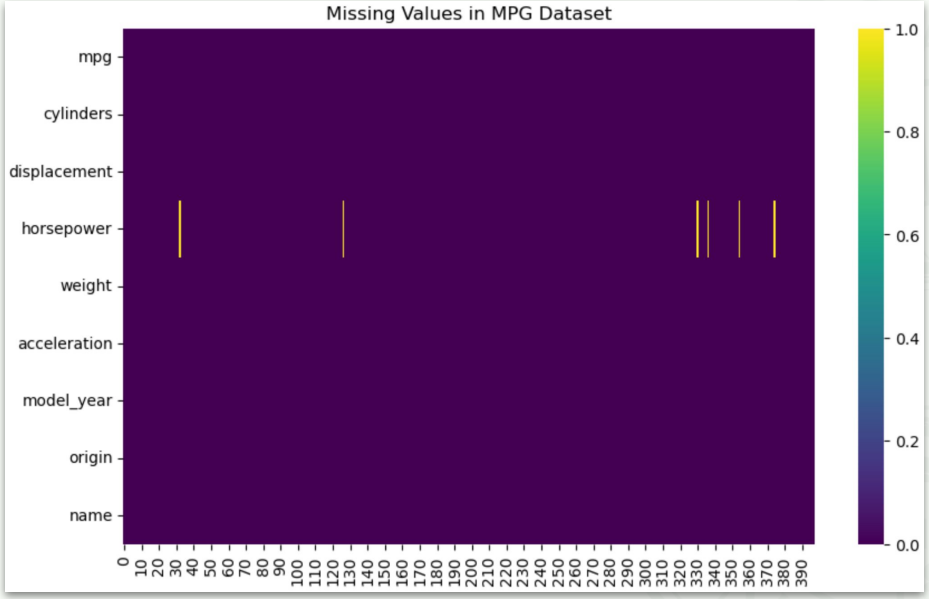
```
df_m[df_m["horsepower"].isna()]
```

| | mpg | cylinders | displacement | horsepower | weight | acceleration | model_year | origin | name |
|---|---|---|---|---|---|---|---|---|---|
| 32 | 25.0 | 4 | 98.0 | NaN | 2046 | 19.0 | 71 | usa | ford pinto |
| 126 | 21.0 | 6 | 200.0 | NaN | 2875 | 17.0 | 74 | usa | ford maverick |
| 330 | 40.9 | 4 | 85.0 | NaN | 1835 | 17.3 | 80 | europe | renault lecar deluxe |
| 336 | 23.6 | 4 | 140.0 | NaN | 2905 | 14.3 | 80 | usa | ford mustang cobra |
| 354 | 34.5 | 4 | 100.0 | NaN | 2320 | 15.8 | 81 | europe | renault 18i |
| 374 | 23.0 | 4 | 151.0 | NaN | 3035 | 20.5 | 82 | usa | amc concord dl |

In this scenario, we are likely to use "**listwise deletion**": remove the rows that contain a NaN.

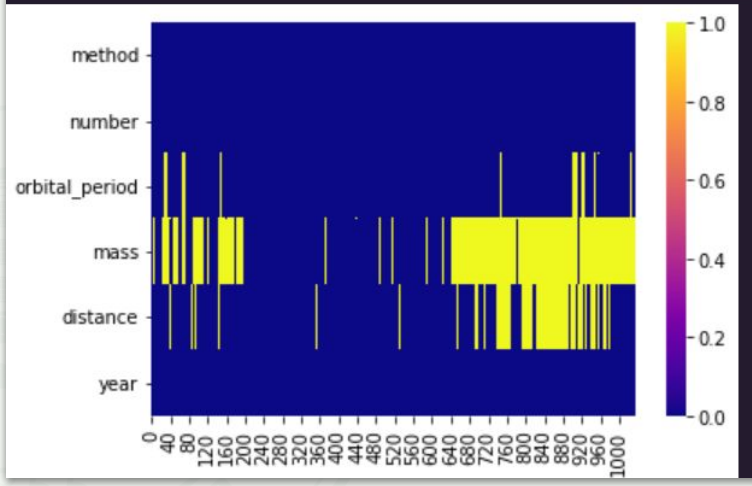
```
df_m.dropna()
```

# Visualize Missing Data: Part of the EDA Process




Seaborn: mpg

```
1 sns.heatmap(df_pl.isna().transpose(),cmap="plasma");
```




Seaborn: planets

# Pairwise Deletion

**List wise deletion**

| Gender | Manpower | Sales |
|---|---|---|
| M | 25 | 343 |
| F | . | 280 |
| M | 33 | 332 |
| M | . | 272 |
| F | 25 | . |
| M | 29 | 326 |
| | 26 | 259 |
| M | 32 | 297 |

**Pair wise deletion**

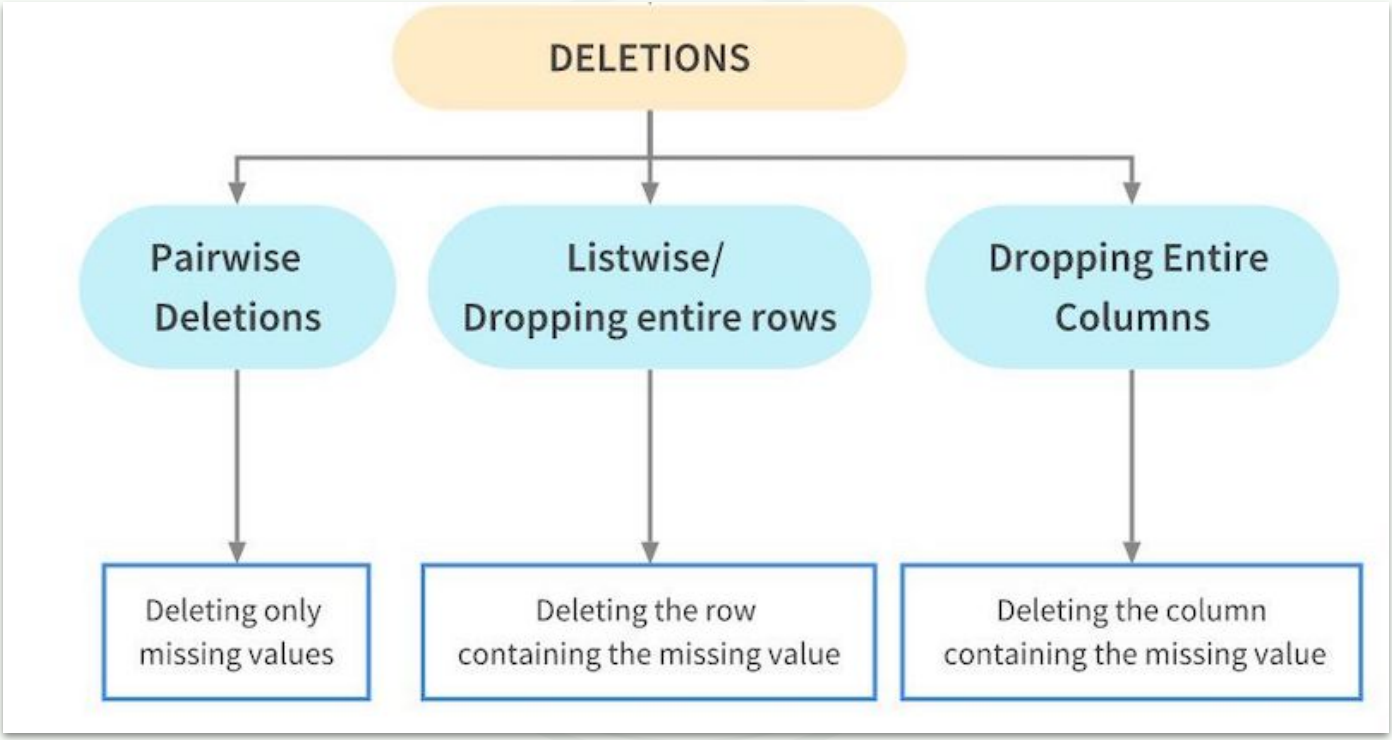
| Gender | Manpower | Sales |
|---|---|---|
| M | 25 | 343 |
| F | . | 280 |
| M | 33 | 332 |
| M | . | 272 |
| F | 25 | . |
| M | 29 | 326 |
| | 26 | 259 |
| M | 32 | 297 |

There is no reason to delete useful data in a row if you are not using the column with the NaN.

Each column now has a different number of rows, and care must be taken when computing statistics.

*Note: This can create impossible correlation matrices!*

# Really Bad? Drop the Column



# Are the missing values random?

| | mpg | cylinders | displacement | horsepower | weight | acceleration | model_year | origin | name |
|---|---|---|---|---|---|---|---|---|---|
| 32 | 25.0 | 4 | 98.0 | NaN | 2046 | 19.0 | 71 | usa | ford pinto |
| 126 | 21.0 | 6 | 200.0 | NaN | 2875 | 17.0 | 74 | usa | ford maverick |
| 330 | 40.9 | 4 | 85.0 | NaN | 1835 | 17.3 | 80 | europe | renault lecar deluxe |
| 336 | 23.6 | 4 | 140.0 | NaN | 2905 | 14.3 | 80 | usa | ford mustang cobra |
| 354 | 34.5 | 4 | 100.0 | NaN | 2320 | 15.8 | 81 | europe | renault 18i |
| 374 | 23.0 | 4 | 151.0 | NaN | 3035 | 20.5 | 82 | usa | amc concord dl |

None of the missing values are from Japan.

If we drop these rows we are biasing the dataset toward Japan.

And, we are losing useful data on the other features.

# Types of Missing Data

**missingness**: the manner in which data are missing

- Missing Completely at Random (MCAR)
- Missing at Random (MAR)
- Missing Not at Random (MNAR)

→ delete rows and/or columns

less data, poorer statistics

*Why is the data missing?!*

# Missingness Examples

**MCAR:** A survey where some respondents forgot to answer a question purely by chance.

**MAR:** In a health survey, younger people might be less likely to disclose their income than older people. If age is recorded, then the missingness of income is MAR with respect to age. *(most common)*

**MNAR:** In a survey about mental health, individuals with severe depression might be less likely to respond. The missingness in this case is related to the unobserved severity of depression.

# Missingness Diagram

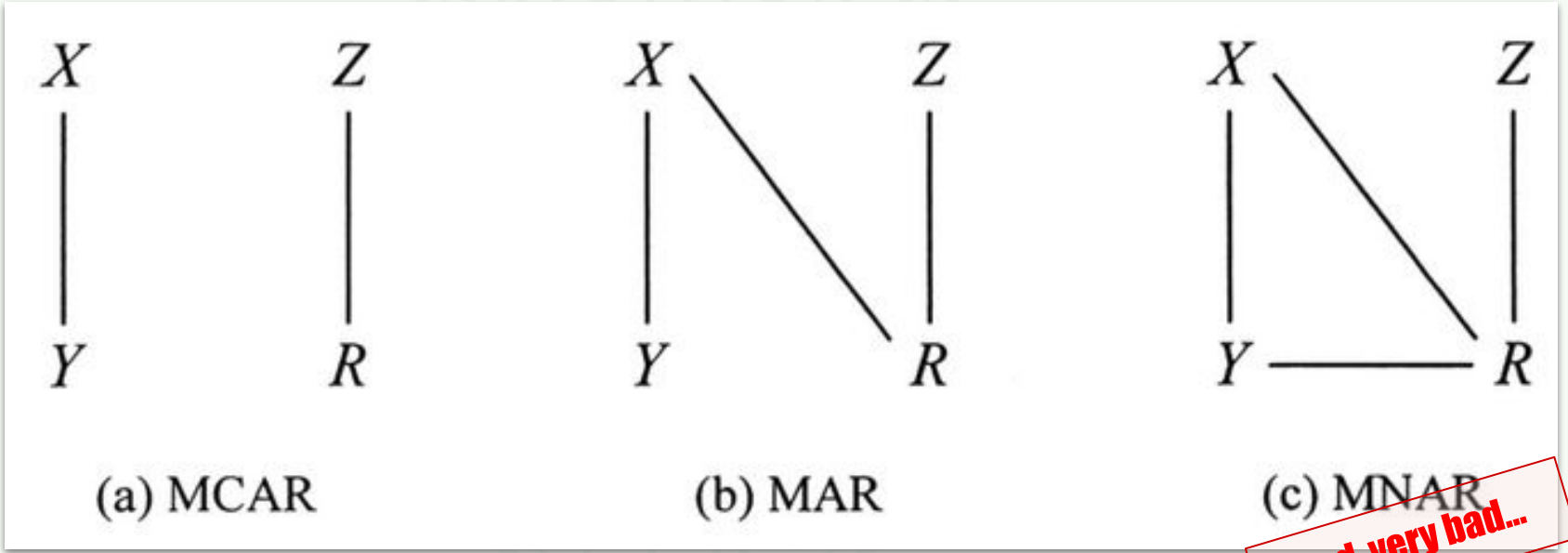
missingness *R* is caused by *Z*, which is something not in our dataset

missingness *R* is caused by *X*, which is something we know about

missingness *R* is caused by *Y*, which means the data is biased

*X* completely observed

*Y* partially observed; *R* are missing values




bad, very bad...

# Imputation

**imputation**: assignment of a missing value

When we are not MCAR, we need to replace the missing values with something *reasonable*.

With **imputation**, we do not drop anything, we replace the `NaN`s with something.

# Mean Imputation/Substitution

Most obvious idea: replace missing values with the average (mean/median/etc) of the values you do have.

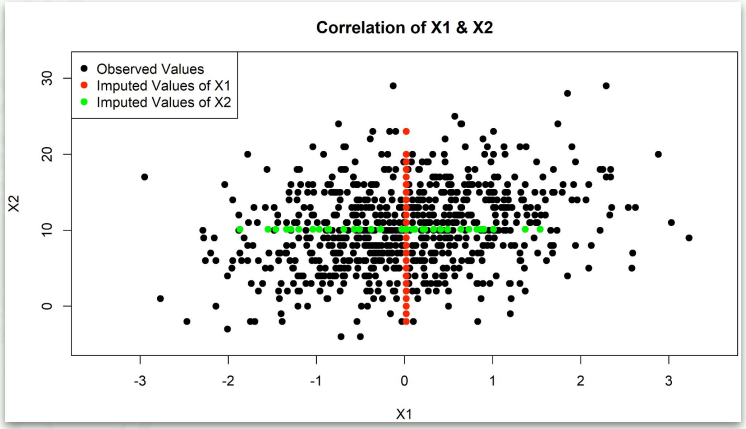
**Easy!** **Right?**

| Price |
|---|
| 100 |
| 90 |
| 50 |
| 40 |
| 20 |
| 100 |
| |
| 60 |
| 120 |
| |
| 200 |

Mean = 86.66

Median = 90

| Price |
|---|
| 100 |
| 90 |
| 50 |
| 40 |
| 20 |
| 100 |
| 86.66 |
| 60 |
| 120 |
| 86.66 |
| 200 |

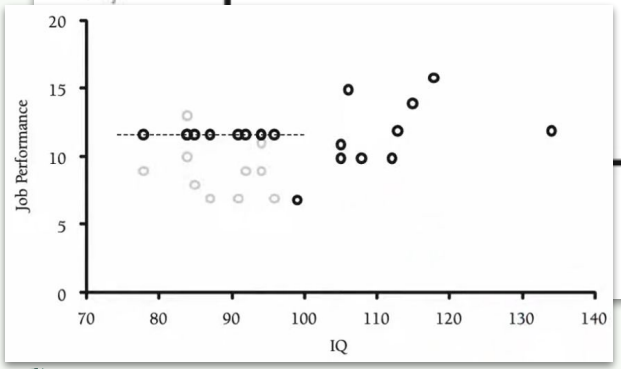



beware...

# Mean Imputation/Substitution

Mean imputation is possibly the worst missing data handling method available.

# Regression (Simple)



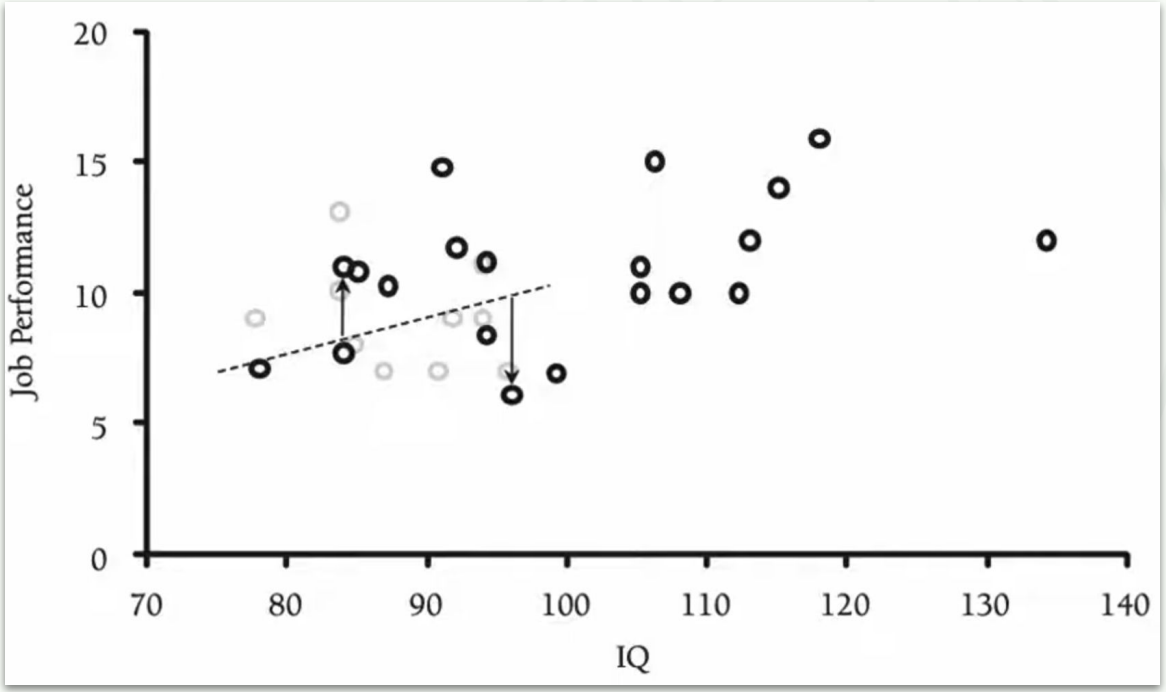

Using regression captures the trend that is lost when we use mean substitution.

However, an important message is lost!

Yes, there is a slight upward trend, but mostly there is no correlation.

# Stochastic Regression

In addition to the regression line, we can estimate the variance.
from the variance, we can add noise.

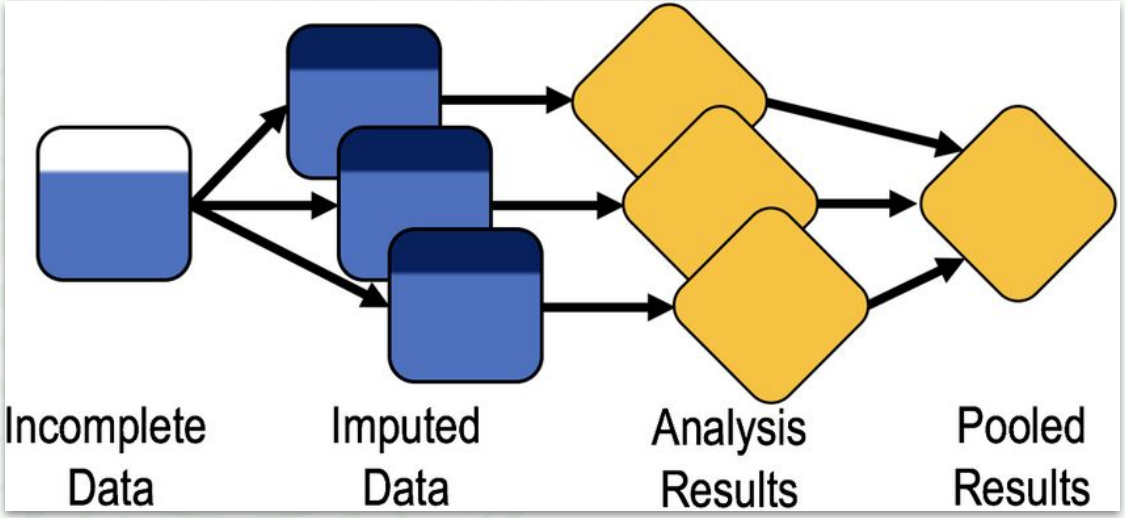



This method repairs most of the problems with previously-discussed methods.

# Multiple Imputation

**Note:** the regression can be generated many times!

This means one can obtain many predictions, giving a confidence interval.



# Modern Approaches

Modern imputation methods employ machine learning.

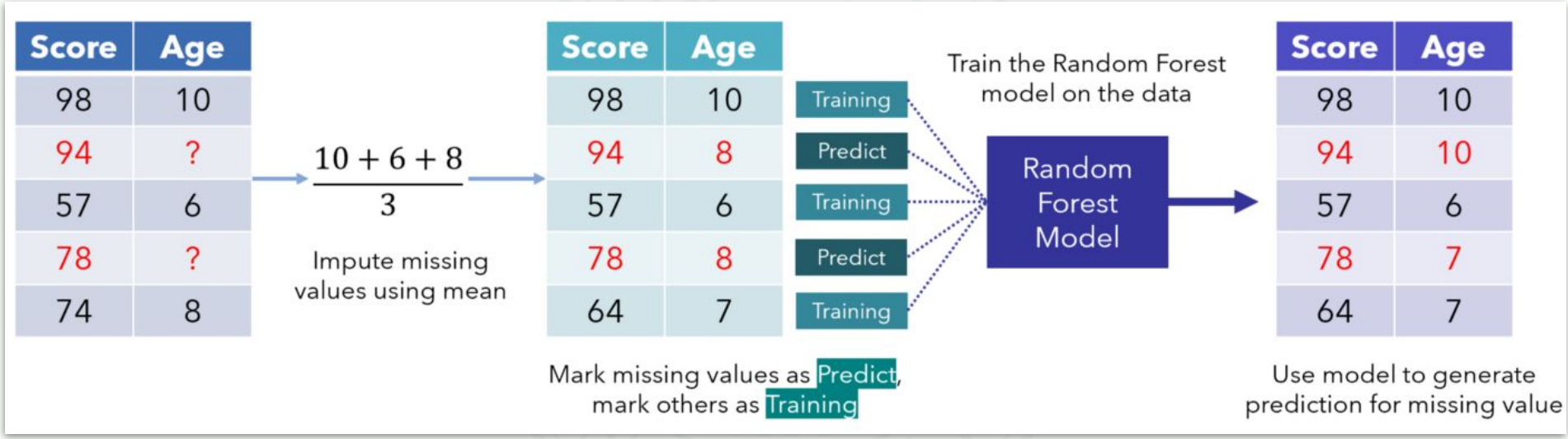
We already saw regression: machine learning supplies a vast set of methods for this.

Deep learning is the best, when there is enough data.

Let's examine KNN (K nearest neighbors) methods (there are *many* variants) .

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 28 | 9.0 | 8 | 304.0 | 193.0 | 4732 | 18.5 | 70 | usa | hi 1200d |
| 29 | 27.0 | 4 | 97.0 | 88.0 | 2130 | 14.5 | 71 | japan | datsun pl510 |
| 30 | 28.0 | 4 | 140.0 | 90.0 | 2264 | 15.5 | 71 | usa | chevrolet vega 2300 |
| 31 | 25.0 | 4 | 113.0 | 95.0 | 2228 | 14.0 | 71 | japan | toyota corona |
| 32 | 25.0 | 4 | 98.0 | NaN | 2046 | 19.0 | 71 | usa | ford pinto |
| 33 | 19.0 | 6 | 232.0 | 100.0 | 2634 | 13.0 | 71 | usa | amc gremlin |
| 34 | 16.0 | 6 | 225.0 | 105.0 | 3439 | 15.5 | 71 | usa | plymouth satellite custom |

# Modern Approaches: Alternate ML Approach



# Wed: ICA on Libraries