METIS

Day 8: Feature Engineering & Cross Validation

John Navarro
john.navarro@thisismetis.com
https://www.linkedin.com/in/johnnavarro/

METIS

Goals for Today

- Articulate why feature scaling is important, and be able to do it using sklearn
- Use a variety of transformation methods to reduce non-normality in features
- Handle categorical features when building a machine learning model
- Handle null/missing values when building a machine learning model
- Use cross validation to accurately estimate model performance
- Articulate the strengths and weaknesses of basic cross validation
- Use cross validation to choose optimal model parameters by searching across many models simultaneously

METIS

Feature Engineering

- Transforming the values at our disposal into a representation that a given machine learning algorithm can use

- Creating new, derived features from the available features (using domain expertise) and use them when training a model

- Dealing with missing values, as most ML algorithms can't handle unknown values. They must be filled in.

METIS

Feature Engineering

- **Feature scaling**: making it so that all columns (features) range over the same values is very helpful for many (but not all) machine learning models. In some cases, scaling along samples is useful, as well.

- **Turning categorical values into numerical values**: machine learning algorithms only understand numbers, not categories.

- **Handling missing values**: Models break when you give them `NaNs`, what are some strategies to replace `NaNs` with "good" (i.e. useful) numbers?

METIS

Build the Classifier

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X, y)
```

Build the Classifier

```
Training data:
    id  length  mass  rings
0   0     0.9   0.1     40
1   1     0.3   0.2     50
2   2     0.6   0.8     60

Single test sample:
    length  mass  rings
0    0.59  0.79   54.9
```

Which class is it?

METIS

Build the Classifier

```
print("Prediction: ",knn.predict(test))
>> Prediction:  [1]
```

METIS

What happened?

METIS

Scale all the features

METIS

When you're creating a scaling object, you should first "**fit**" it to the **training data**, then **transform** both the **training and testing data** using the "fit" scaler.

If you try to fit the training and testing data separately, you will get inaccurate results.

METIS

Standard Scaling

$$zscore(x_i) = \frac{x_i - \mu}{\sigma}$$

```
scaled_feature = (feature - column_mean) / standard_deviation
```

METIS

Standard Scaling

$$zscore(x_i) = \frac{x_i - \mu}{\sigma}$$

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)

or
X_scaled = scaler.fit_transform(X)
```

METIS

Standard Scaling

```
original values:
 [[  0.9   0.1  40. ]
 [  0.3   0.2  50. ]
 [  0.6   0.8  60. ]]

Mean of each column:
 [  0.6  0.3667  50.]

SD of each column:
 [ 0.2449  0.3091  8.165 ]
```

```
scaled values:
 [[ 1.2247 -0.8627 -1.2247]
 [-1.2247 -0.5392  0.    ]
 [ 0.      1.4018  1.2247]]

Means of scaled data, per column:
 [ 0. -0.  0.]

SD's of scaled data, per column:
 [ 1.  1.  1.]
```

METIS

Min/Max Scaling

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

```
normed_feature = (feature - col_min) / (col_max - col_min)
```

Min/Max Scaling

```
from sklearn.preprocessing import MinMaxScaler
minmax= MinMaxScaler()

minmax.fit(X)
X_scaled_minmax = minmax.transform(X)

or

X_scaled_minmax = minmax.fit_transform(X)
```

MinMax Scaling

```
original values:
 [[  0.9   0.1  40. ]
 [  0.3   0.2  50. ]
 [  0.6   0.8  60. ]]

Mean of each column:
 [  0.6  0.3667  50.]

SD of each column:
 [ 0.2449  0.3091  8.165 ]
```

```
scaled values:
 [[ 1.       0.       0.     ]
 [ 0.       0.1429  0.5    ]
 [ 0.5     1.       1.     ]]
Means of scaled data, per column:
[ 0.5    0.381  0.5  ]

SD's of scaled data, per column:
[ 0.4082  0.4416  0.4082]
```

METIS

The Dataset

- Label (Target)
- Alcohol
- Malic acid
- Ash
- Alcalinity of ash
- Magnesium
- Total phenols
- Flavonoids
- Nonflavanoid phenols
- Proanthocyanins
- Color intensity
- Hue
- OD280/OD315 of diluted wines
- Proline

Exercise

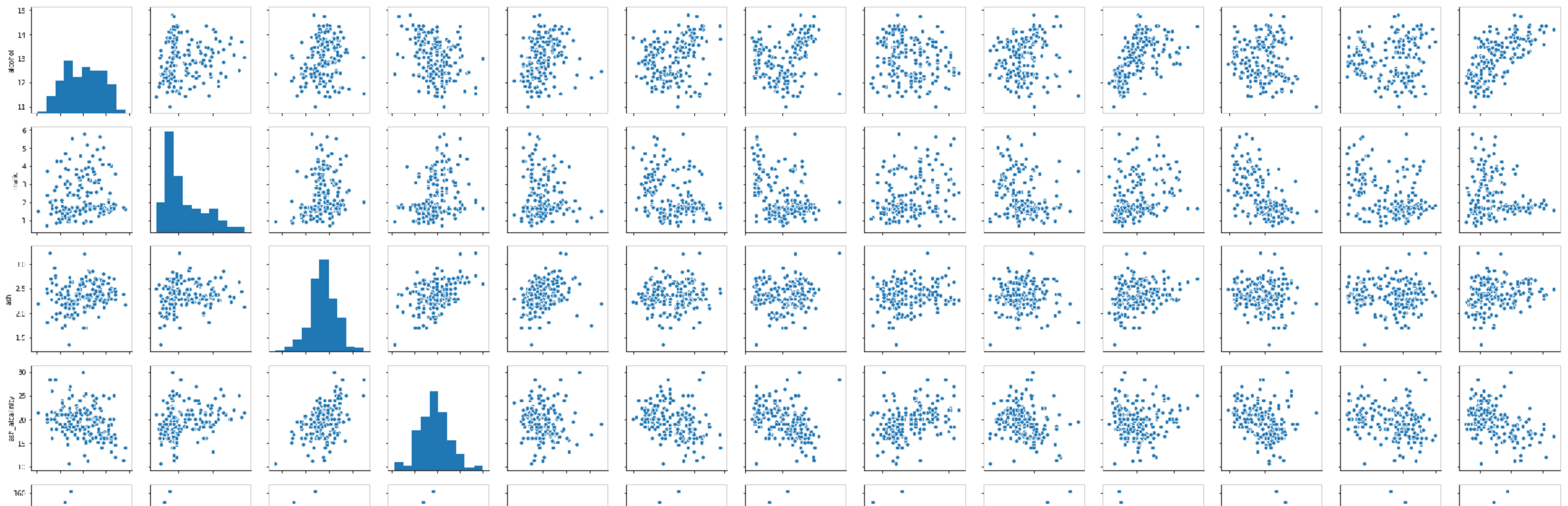
- Using the wine dataset, build a knn model with 3 nearest neighbors to predict the wine's `label` from the remaining columns. Don't scale the data, and check the test error when using 80/20 train/test split. (use `KNeighborsClassifier(n_neighbors=3)`)

- Build the same model, with the same train/test split, but scale the data using `StandardScaler` and `MinMaxScaler`. What is the test error for each of these scaled datasets (for `train_test_split use random_state=1234` for reproducibility)?

METIS

Minimizing Skew

Many machine learning methods rely on the assumption that feature values are distributed normally and have a symmetrical shape. However, this is not usually (almost never) the case.

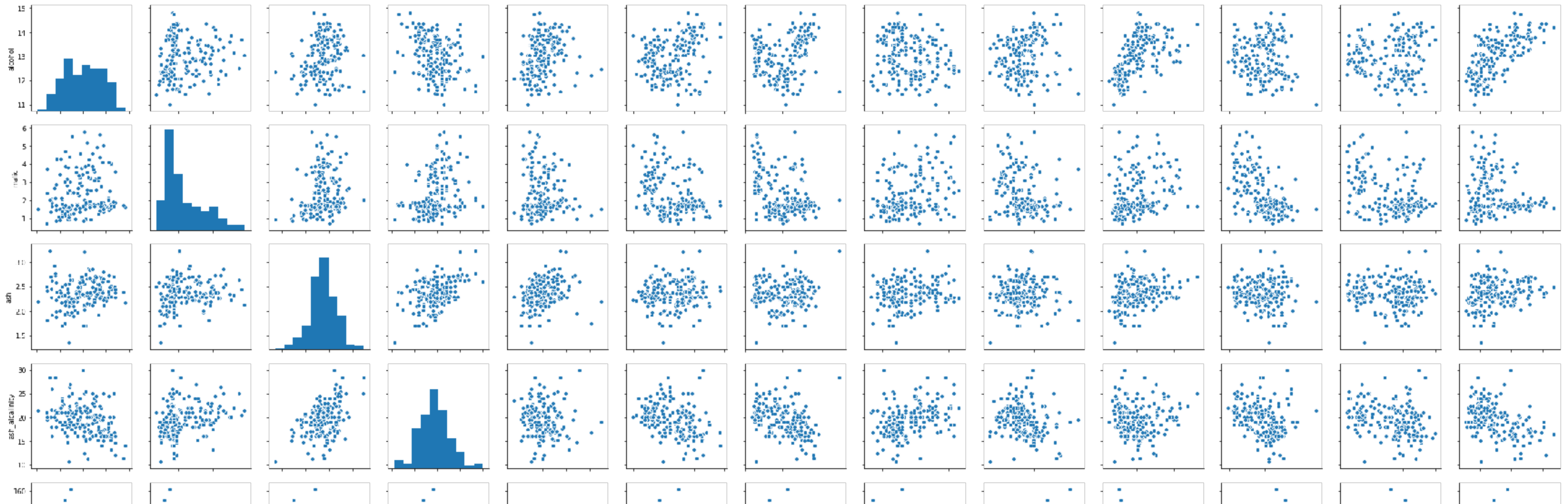


METIS

Minimizing Skew

$$Skew = \frac{m^3}{\sigma^3} \text{ where } m^3 = \frac{\sum_{i=1}^{n}(X_i - \mu_x)^3}{n}$$

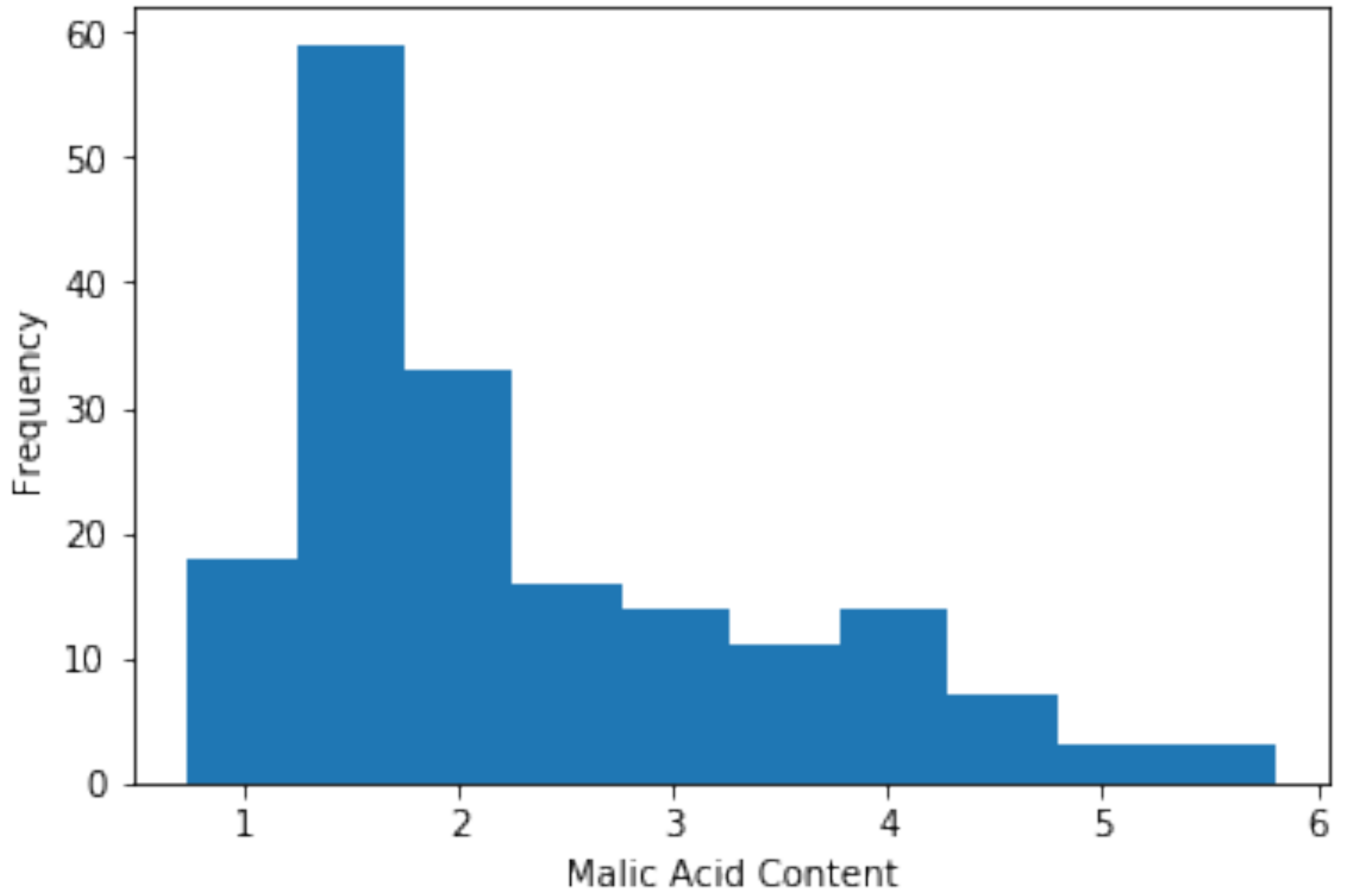
```
data['column'].skew()
```



METIS

Minimizing Skew

Many machine learning methods rely on the assumption that feature values are distributed normally and have a symmetrical shape. However, this is not usually (almost never) the case.



METIS

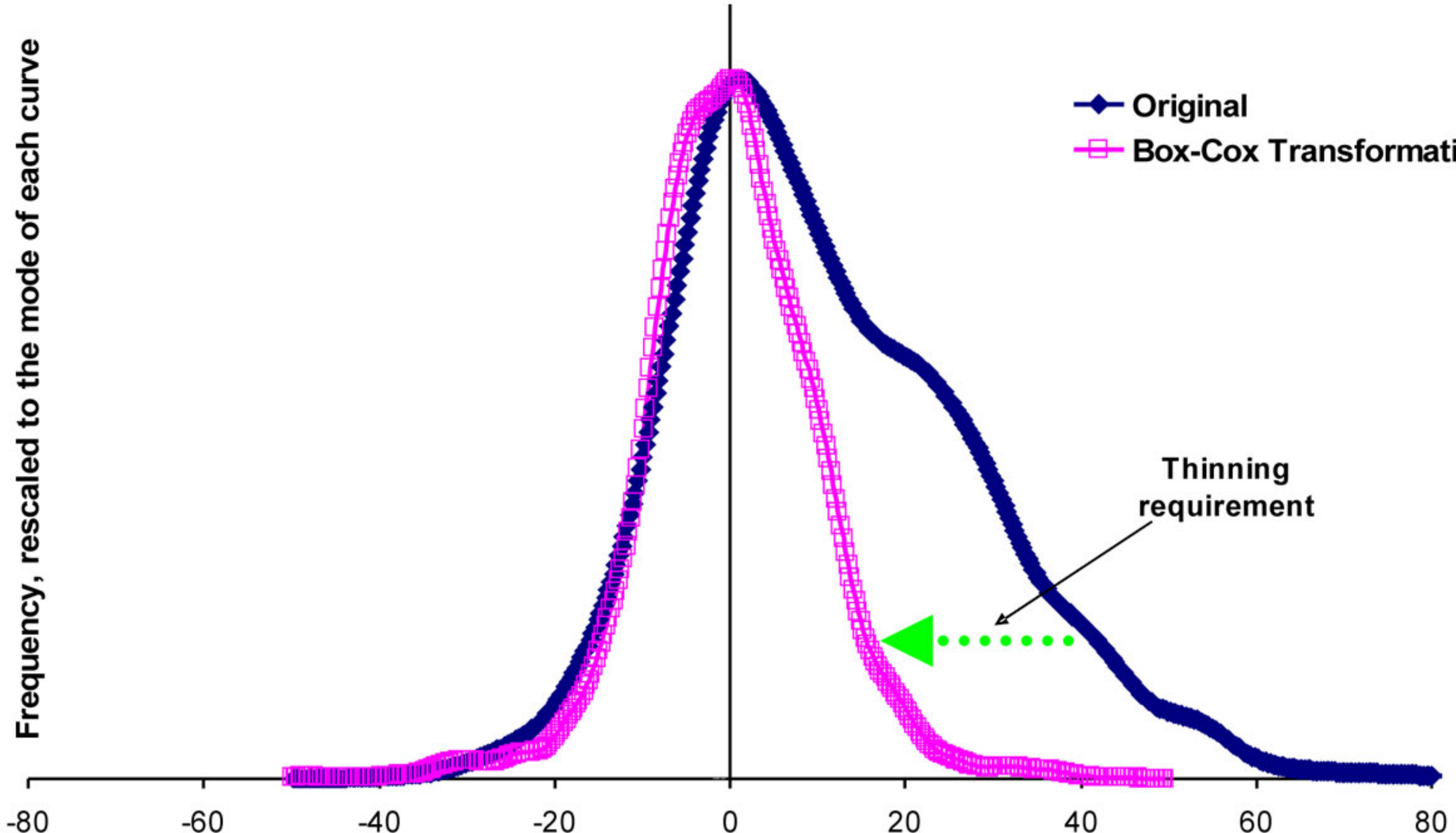
Minimizing Skew

1. **Square root transformation:** take the square root of each value
2. **Logarithmic transformation:** take the natural logarithm of each value
3. **Box-Cox transformation:** Use the Box-Cox calculation to "figure out" the optimal power (exponent) to transform your data

METIS

Minimizing Skew

The Box-Cox transformation, **also known as the Power Transformation**, is a very common way to "find" the best way to transform your data automatically.



METIS

Minimizing Skew

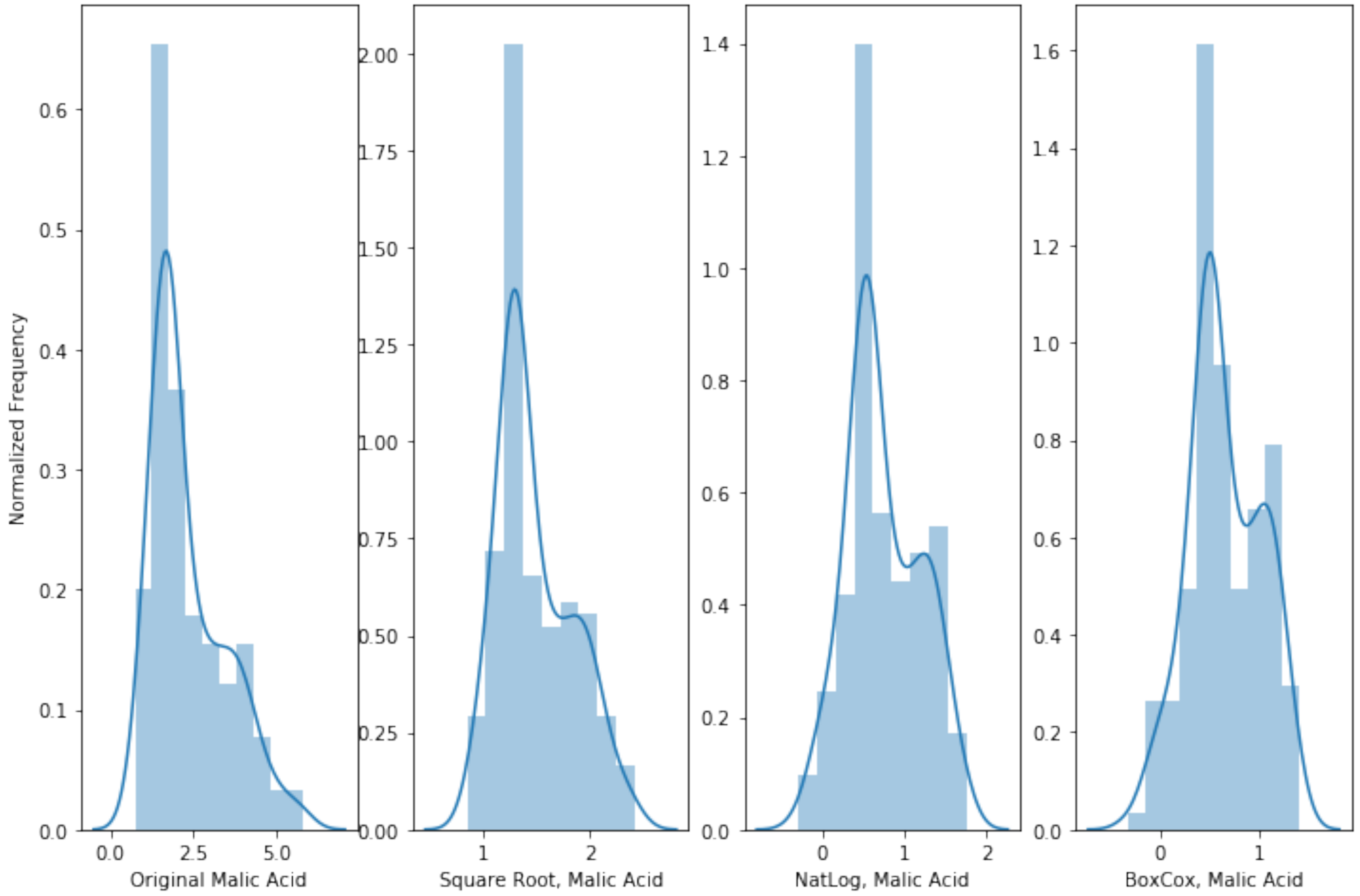
The Box-Cox transformation, **also known as the Power Transformation**, is a very common way to "find" the best way to transform your data automatically.

```
sqrt_malic = np.sqrt(wine_data.malic)
log_malic = np.log(wine_data.malic)
boxcox_malic,power_val = stats.boxcox(wine_data.malic)
```

Note: All of these transformations require that your data is positive to begin with. If you have negative values in your data, you must scale it to be all positive values first (using MinMax Scaler, for example).

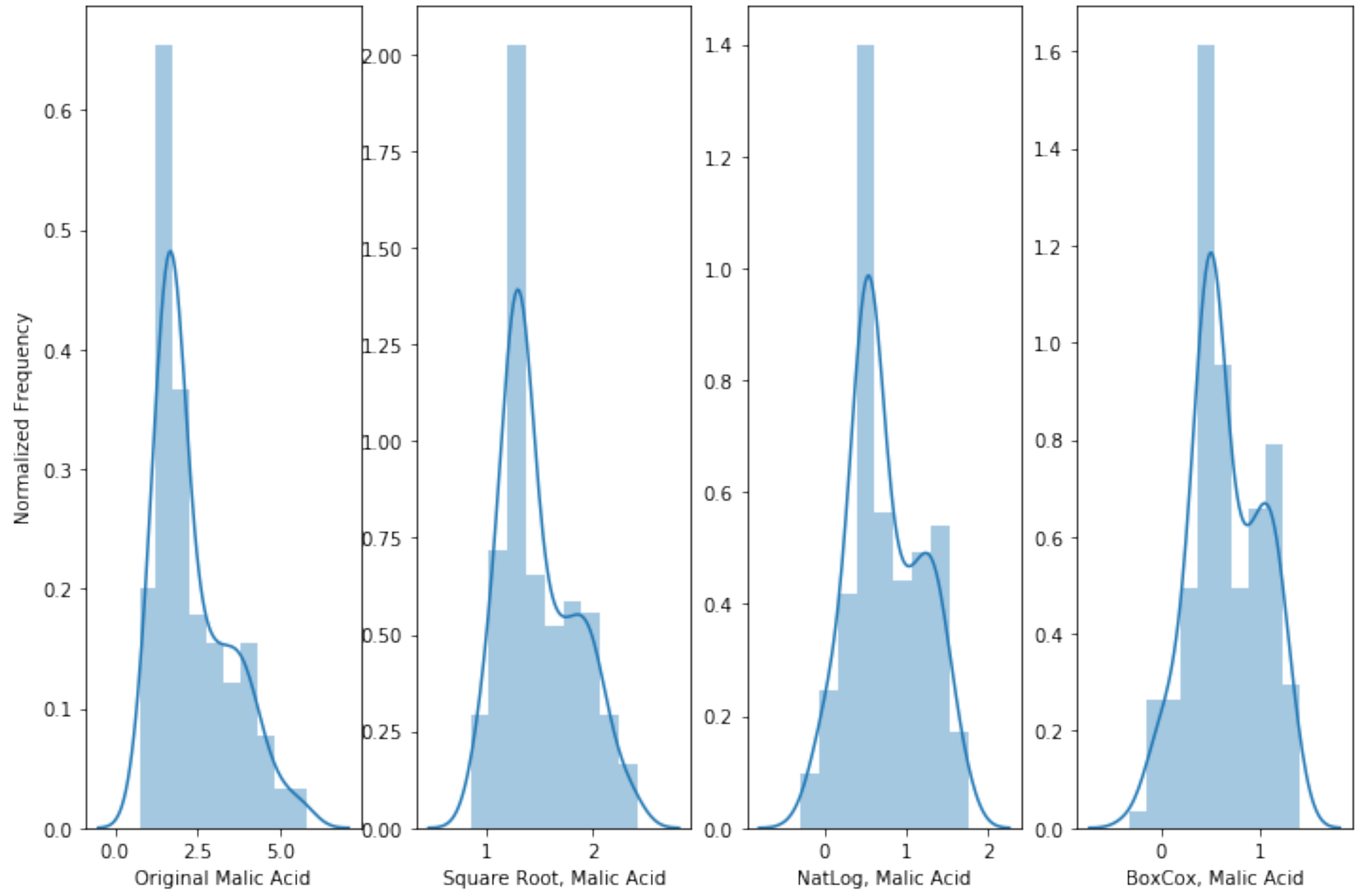
METIS

Minimizing Skew



METIS

Minimizing Skew



```
[('original data skew', 1.0308694978039965),
 ('square_root skew', 0.668562089904606),
 ('natural log skew', 0.272293752908186 3),
 ('box-cox skew', 0.026826808320270 62)]
```

METIS

Exercise

- Look at the individual variable histograms in the wine dataset using `sns.distplot` and compute each feature's original skew (not the targets) using `stats.skew` (a value very far from 0, to either side, is very bad). Which variables are candidates for one of these transformations?

- Transform those columns you think warrant a transformation.

- Compare the test-set error of a Logistic Regression Classifier (`LogisticRegression`) before/after applying your transformations on the dataset (use the `random_state=123` in `train_test_split` to compare identical splits of the data across both transformations).

METIS

Categorical Features

- **Ordered categories:** transform them to sensible numeric values (example: small=1, medium=2, large=3)
- **Unordered categories**: use dummy encoding

METIS

The Dataset

Sex: M, F, and I (infant)

Length: Longest shell measurement

Diameter: Perpendicular to length

Height: Height with with meat in shell

Whole weight: Whole abalone weight

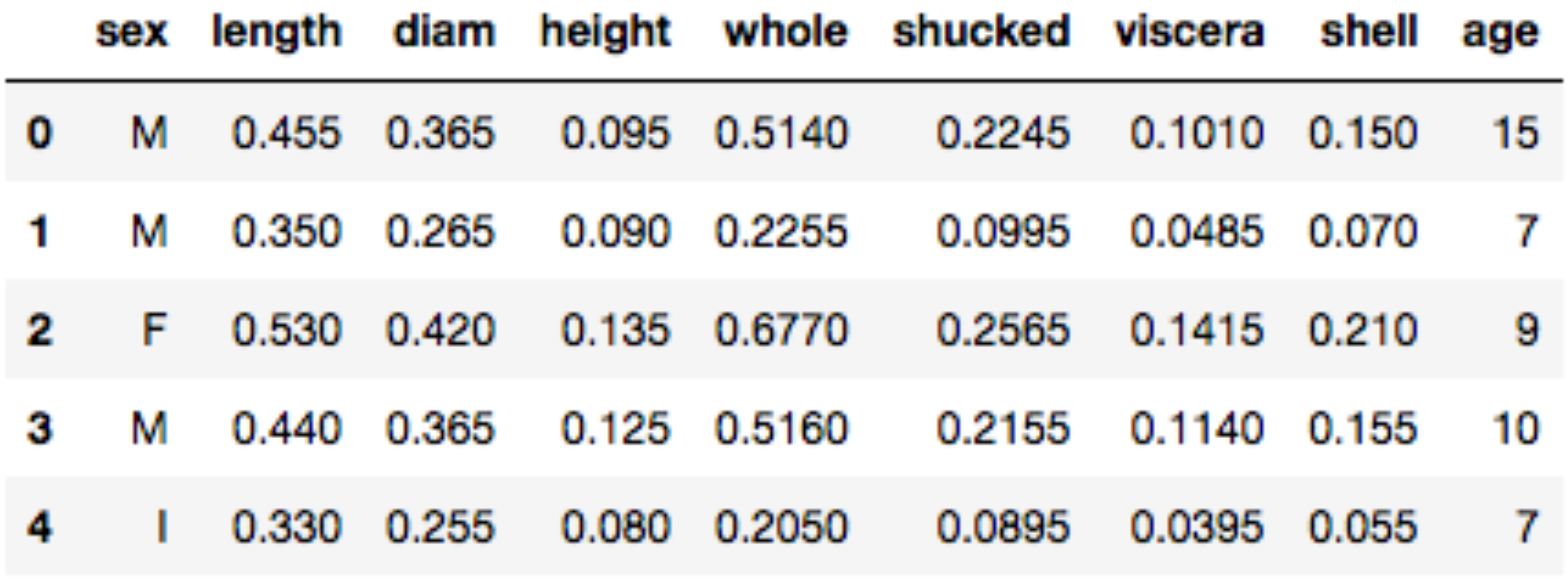
Shucked weight: Weight of meat only

Viscera weight: Gut weight (after bleeding)

Shell weight: Weight after being dried

Rings: This value +1.5 gives the abalone's age in years

Categorical Features

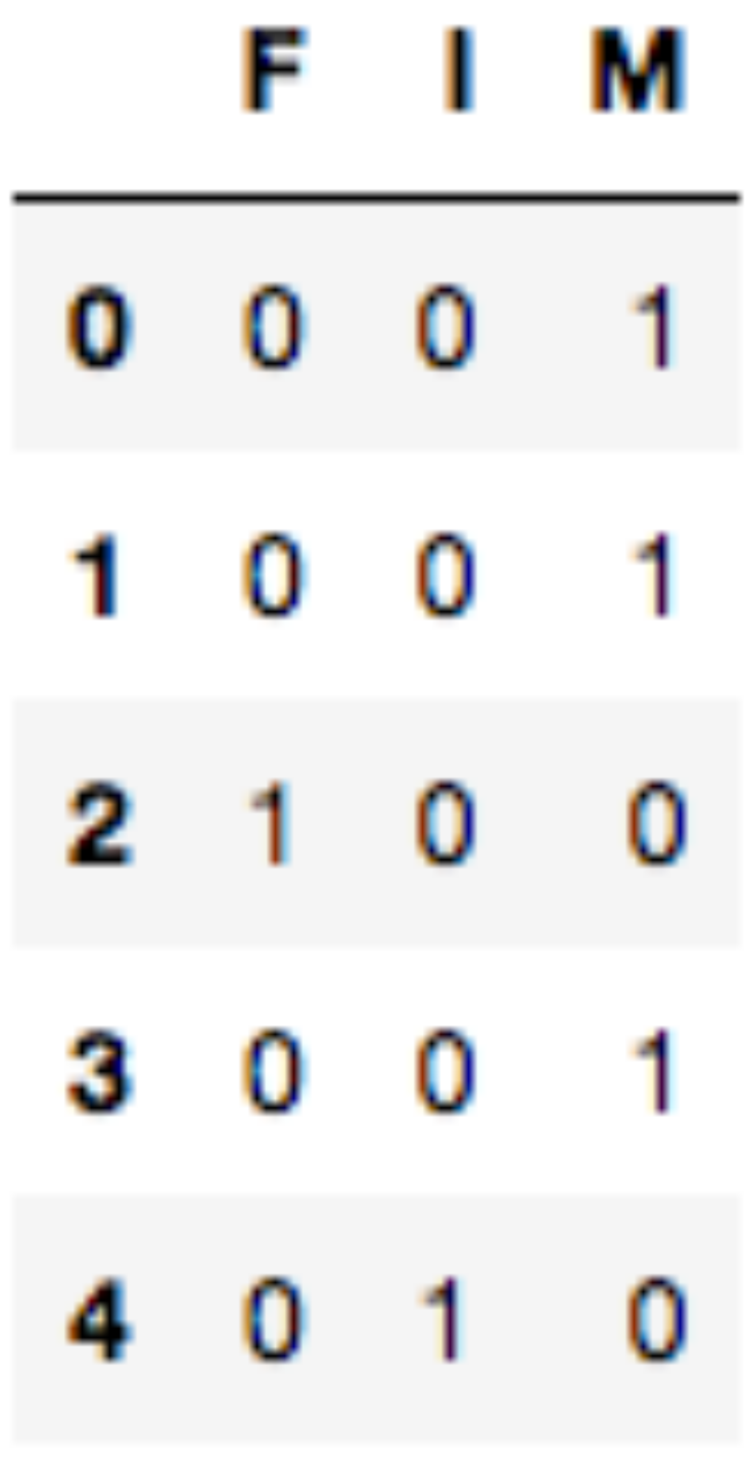


	sex	length	diam	height	whole	shucked	viscera	shell	age
0	M	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	15
1	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	7
2	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	9
3	M	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	10
4	I	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	7

METIS

Categorical Features

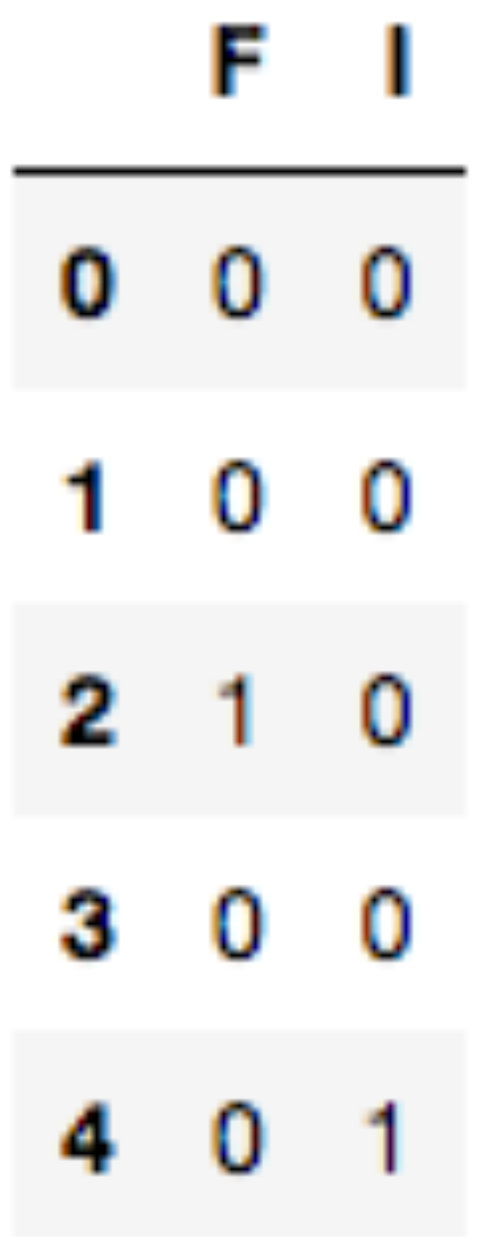
```
sex_dummies = pd.get_dummies(abalone_data.sex)
sex_dummies.head()
```



	F	I	M
0	0	0	1
1	0	0	1
2	1	0	0
3	0	0	1
4	0	1	0

Categorical Features

```
sex_dummies = pd.get_dummies(abalone_data.sex)
sex_dummies = sex_dummies[["F","I"]]
sex_dummies.head()
```



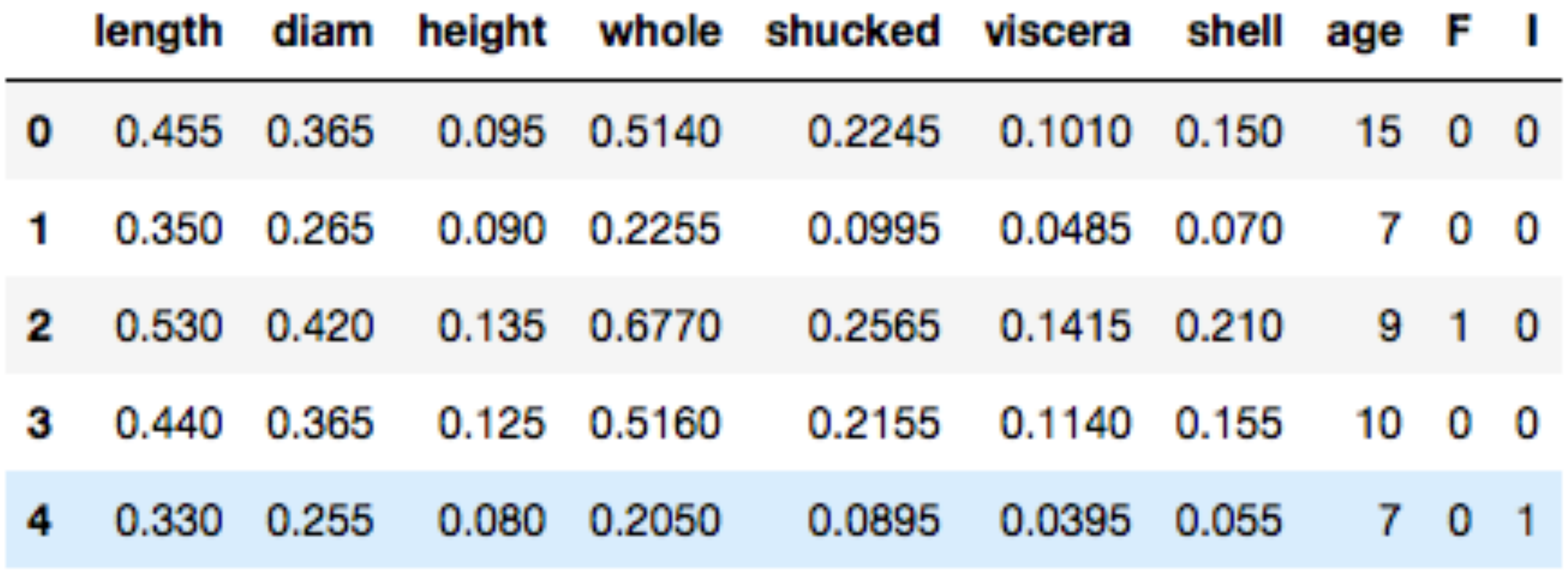
	F	I
0	0	0
1	0	0
2	1	0
3	0	0
4	0	1

Here is how we interpret the encoding:

- F is encoded as F=1 and I=0
- I is encoded as F=0 and I=1
- M is encoded as F=0 and I=0

Categorical Features

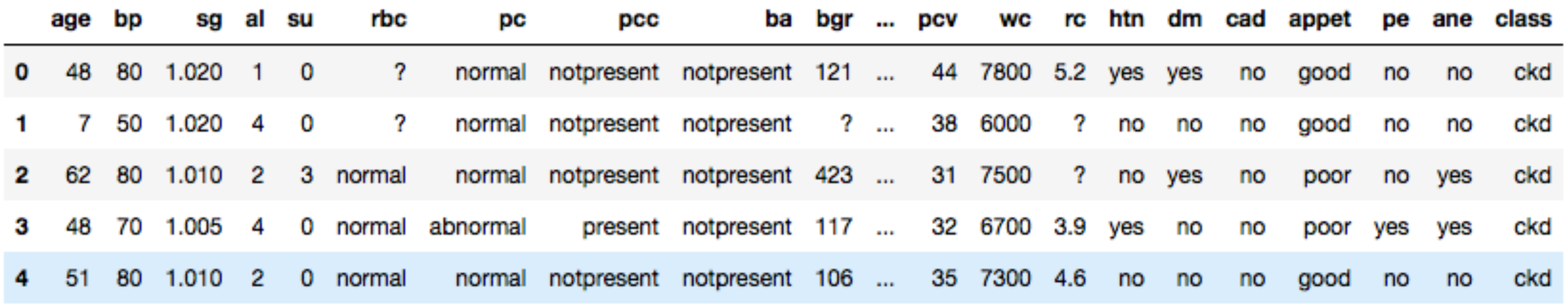
```
abalone_data = pd.concat([abalone_data,sex_dummies],axis=1)
abalone_data.drop("sex",inplace=True,axis=1)
```



	length	diam	height	whole	shucked	viscera	shell	age	F	I
0	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	15	0	0
1	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	7	0	0
2	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	9	1	0
3	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	10	0	0
4	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	7	0	1

METIS

Missing Data



	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	class
0	48	80	1.020	1	0	?	normal	notpresent	notpresent	121	...	44	7800	5.2	yes	yes	no	good	no	no	ckd
1	7	50	1.020	4	0	?	normal	notpresent	notpresent	?	...	38	6000	?	no	no	no	good	no	no	ckd
2	62	80	1.010	2	3	normal	normal	notpresent	notpresent	423	...	31	7500	?	no	yes	no	poor	no	yes	ckd
3	48	70	1.005	4	0	normal	abnormal	present	notpresent	117	...	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
4	51	80	1.010	2	0	normal	normal	notpresent	notpresent	106	...	35	7300	4.6	no	no	no	good	no	no	ckd

METIS

Missing Data

```
kidney_data = pd.read_csv("<file>",
header=None,
na_values="?",
names=kidney_columns)
```

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	class
0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	121.0	...	44.0	7800.0	5.2	yes	yes	no	good	no	no	ckd
1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	NaN	...	38.0	6000.0	NaN	no	no	no	good	no	no	ckd
2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	423.0	...	31.0	7500.0	NaN	no	yes	no	poor	no	yes	ckd
3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	117.0	...	32.0	6700.0	3.9	yes	no	no	poor	yes	yes	ckd
4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	106.0	...	35.0	7300.0	4.6	no	no	no	good	no	no	ckd

METIS

Missing Data

```
kidney_data.isnull().sum()

age          9
bp          12
sg          47
al          46
su          49
rbc        152
pc          65
...
```

METIS

Missing Data

Option 1: Drop null rows

```
kidney_data_nonnull = kidney_data.dropna()

>>Fraction of data kept: 0.395
```

METIS

Missing Data

Option 2: Impute missing values:

We can **impute** (fill in) the data on a per-column basis. The imputation strategy for categorical columns is usually one of the following:

1. Fill in with the most common categorical value
2. Fill in with a special "missing" category

METIS

Missing Data

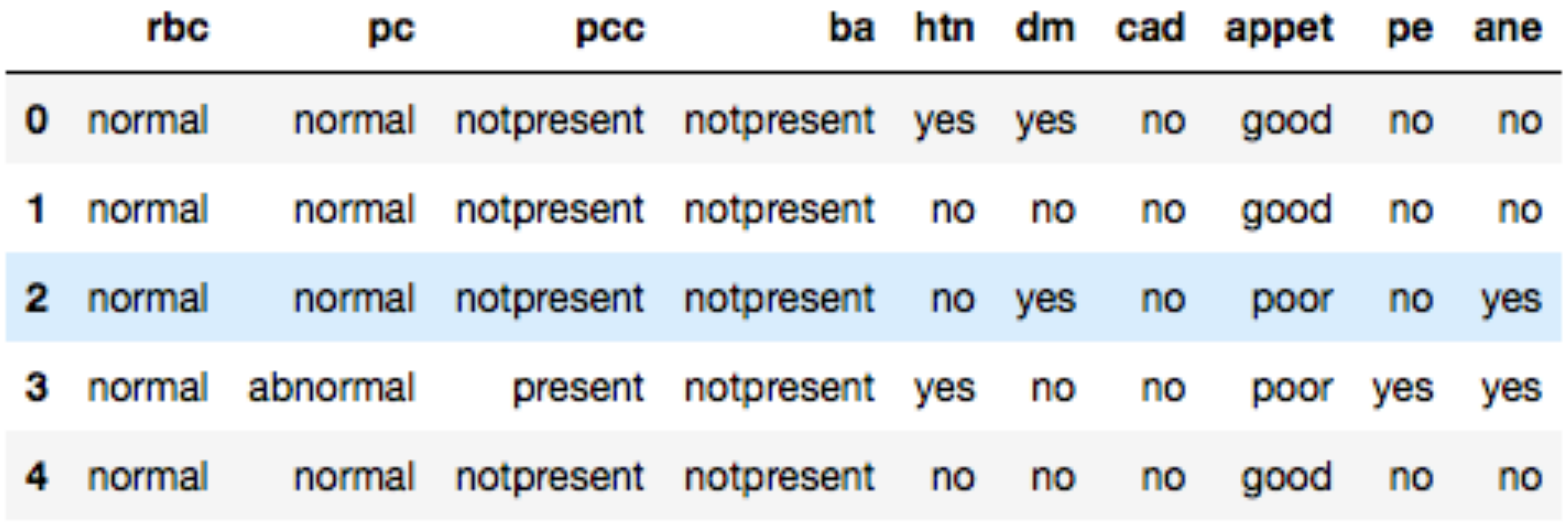
```
def get_most_frequent_value(my_column):
    return my_column.value_counts().index[0]

most_frequent_values_per_column =
kidney_data[kidney_columns[14:-1]].apply(get_most_frequen
t_value,axis=0)

categorical_most_frequent =
kidney_data[kidney_columns[14:-1]].fillna(most_frequent_v
alues_per_column,axis=0)
```

METIS

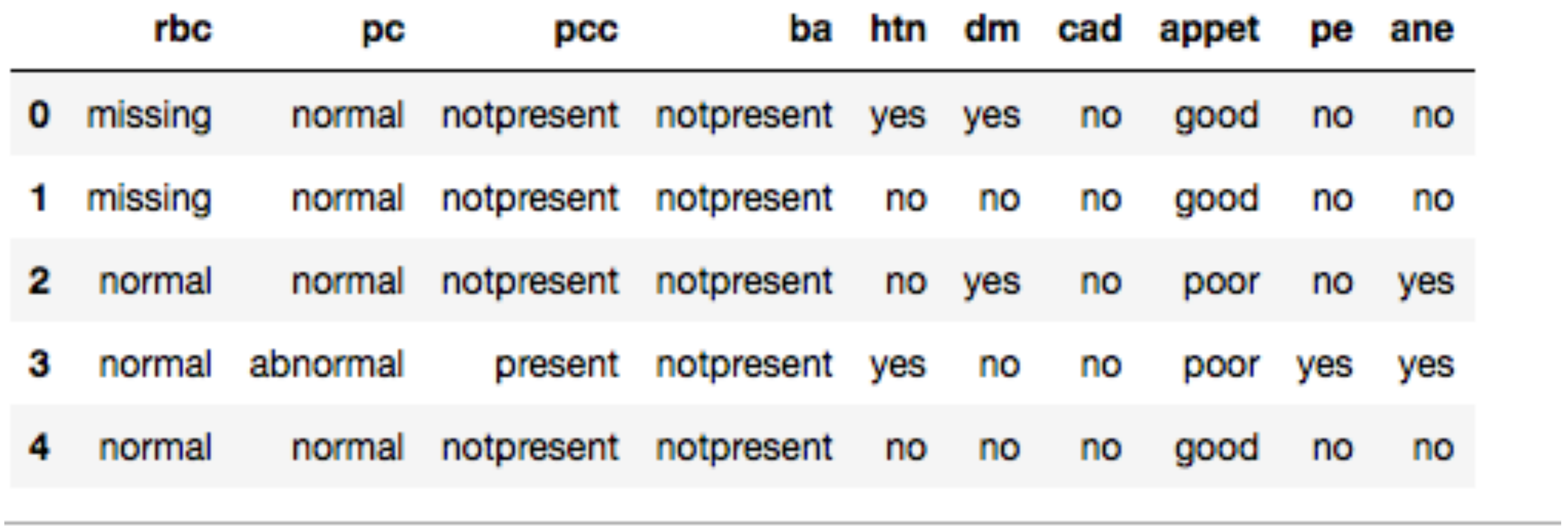
Missing Data



	rbc	pc	pcc	ba	htn	dm	cad	appet	pe	ane
0	normal	normal	notpresent	notpresent	yes	yes	no	good	no	no
1	normal	normal	notpresent	notpresent	no	no	no	good	no	no
2	normal	normal	notpresent	notpresent	no	yes	no	poor	no	yes
3	normal	abnormal	present	notpresent	yes	no	no	poor	yes	yes
4	normal	normal	notpresent	notpresent	no	no	no	good	no	no

Missing Data

```
special_missing_category =
kidney_data[kidney_columns[14:-1]].fillna("missing")
```



	rbc	pc	pcc	ba	htn	dm	cad	appet	pe	ane
0	missing	normal	notpresent	notpresent	yes	yes	no	good	no	no
1	missing	normal	notpresent	notpresent	no	no	no	good	no	no
2	normal	normal	notpresent	notpresent	no	yes	no	poor	no	yes
3	normal	abnormal	present	notpresent	yes	no	no	poor	yes	yes
4	normal	normal	notpresent	notpresent	no	no	no	good	no	no

METIS

Missing Data

For numerical columns, there are 3 common strategies for filling in missing values:

1. Fill in using the mean

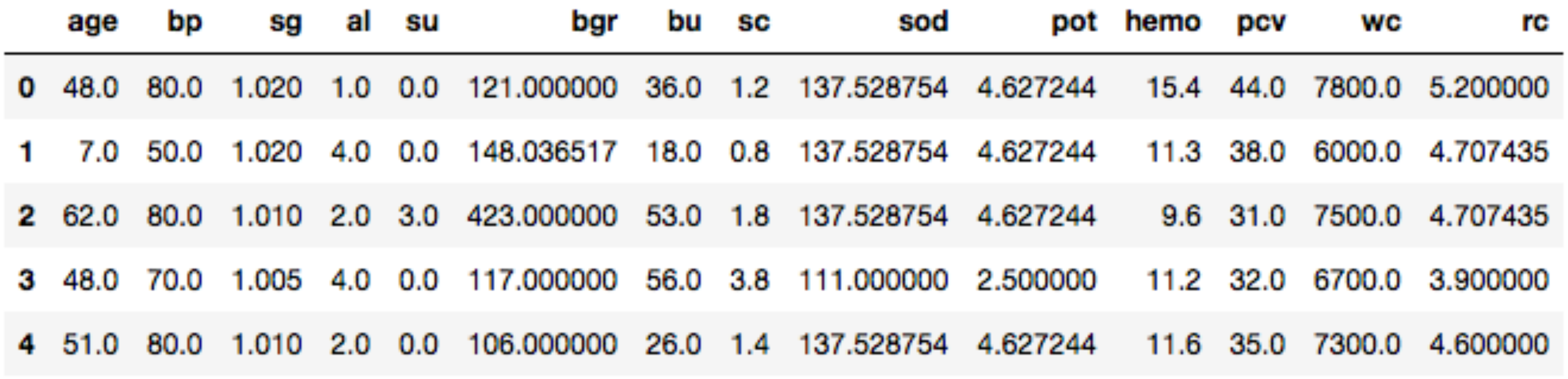
2. Fill in using the median (when many outliers are present)

3. Fill in with some default value (e.g. 0).

METIS

Missing Data

```
mean_per_column = kidney_data[kidney_columns[:14]].apply(lambda x: x.mean(),axis=0)
numeric_mean_filled = kidney_data[kidney_columns[:14]].fillna(mean_per_column,axis=0)
```

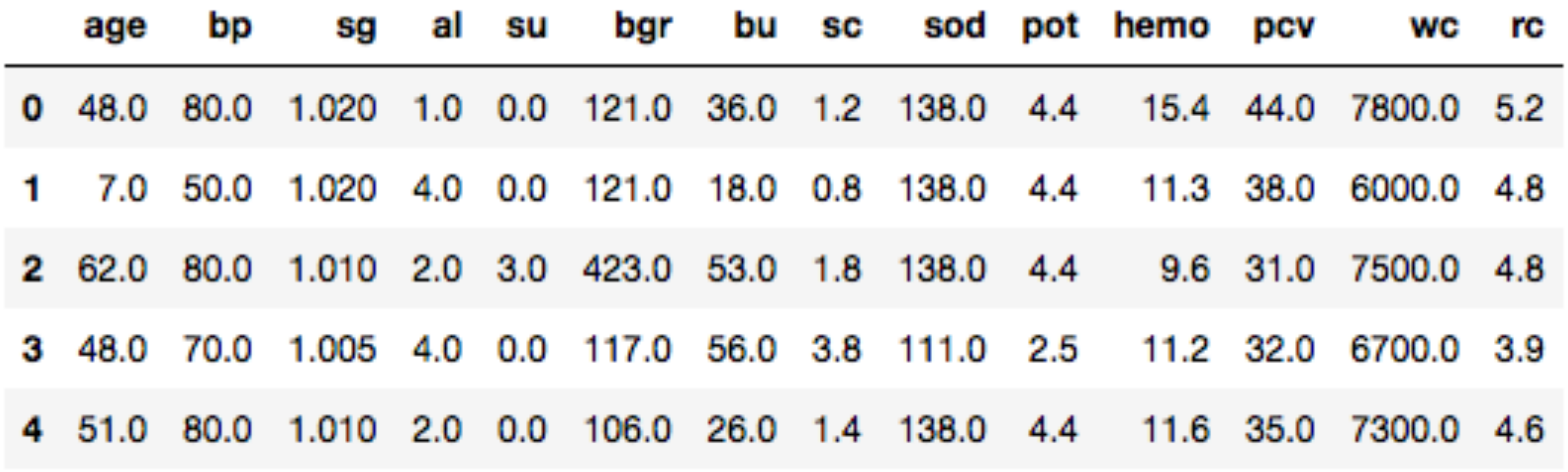


	age	bp	sg	al	su	bgr	bu	sc	sod	pot	hemo	pcv	wc	rc
0	48.0	80.0	1.020	1.0	0.0	121.000000	36.0	1.2	137.528754	4.627244	15.4	44.0	7800.0	5.200000
1	7.0	50.0	1.020	4.0	0.0	148.036517	18.0	0.8	137.528754	4.627244	11.3	38.0	6000.0	4.707435
2	62.0	80.0	1.010	2.0	3.0	423.000000	53.0	1.8	137.528754	4.627244	9.6	31.0	7500.0	4.707435
3	48.0	70.0	1.005	4.0	0.0	117.000000	56.0	3.8	111.000000	2.500000	11.2	32.0	6700.0	3.900000
4	51.0	80.0	1.010	2.0	0.0	106.000000	26.0	1.4	137.528754	4.627244	11.6	35.0	7300.0	4.600000

METIS

Missing Data

```
median_per_column = kidney_data[kidney_columns[:14]].apply(lambda x: x.median(),axis=0)
numeric_mean_filled = kidney_data[kidney_columns[:14]].fillna(median_per_column,axis=0)
```

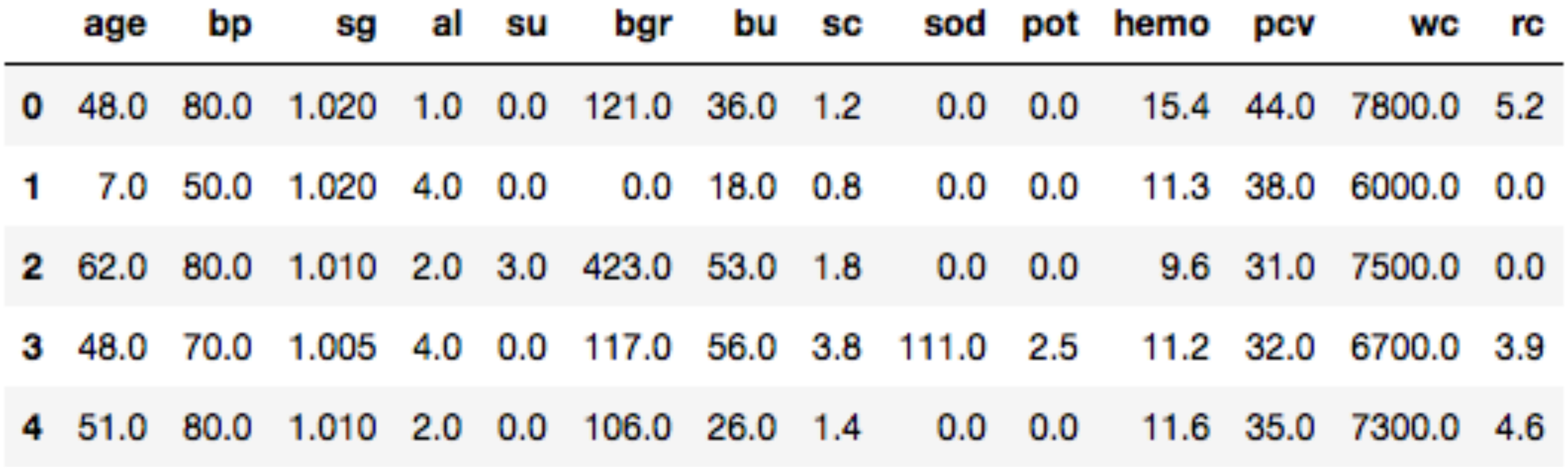


	age	bp	sg	al	su	bgr	bu	sc	sod	pot	hemo	pcv	wc	rc
0	48.0	80.0	1.020	1.0	0.0	121.0	36.0	1.2	138.0	4.4	15.4	44.0	7800.0	5.2
1	7.0	50.0	1.020	4.0	0.0	121.0	18.0	0.8	138.0	4.4	11.3	38.0	6000.0	4.8
2	62.0	80.0	1.010	2.0	3.0	423.0	53.0	1.8	138.0	4.4	9.6	31.0	7500.0	4.8
3	48.0	70.0	1.005	4.0	0.0	117.0	56.0	3.8	111.0	2.5	11.2	32.0	6700.0	3.9
4	51.0	80.0	1.010	2.0	0.0	106.0	26.0	1.4	138.0	4.4	11.6	35.0	7300.0	4.6

METIS

Missing Data

```
default_value_per_column = kidney_data[kidney_columns[:14]].fillna(0.0)
```



	age	bp	sg	al	su	bgr	bu	sc	sod	pot	hemo	pcv	wc	rc
0	48.0	80.0	1.020	1.0	0.0	121.0	36.0	1.2	0.0	0.0	15.4	44.0	7800.0	5.2
1	7.0	50.0	1.020	4.0	0.0	0.0	18.0	0.8	0.0	0.0	11.3	38.0	6000.0	0.0
2	62.0	80.0	1.010	2.0	3.0	423.0	53.0	1.8	0.0	0.0	9.6	31.0	7500.0	0.0
3	48.0	70.0	1.005	4.0	0.0	117.0	56.0	3.8	111.0	2.5	11.2	32.0	6700.0	3.9
4	51.0	80.0	1.010	2.0	0.0	106.0	26.0	1.4	0.0	0.0	11.6	35.0	7300.0	4.6

METIS

Exercise

Using the kidney dataset:

1. Create a complete, filled in dataset of non-missing values using mean imputation per numeric column, most frequent value imputation for the categorical values, convert all of the categorical columns into numerical columns using `get_dummies`

2. Do the same thing using median imputation for each numeric column.

3. Compare test set errors when building a Logistic Regression model (`LogisticRegression()`) on the data and using train/test split (`train_test_split(random_state=123)`), predicting the `class` column. Which imputation method seems to perform better on this dataset?

Polynomial Features

```
from sklearn.preprocessing import PolynomialFeatures
poly_fit_3 = PolynomialFeatures(degree=3)

fitted_degree3_numeric_kidneys =
poly_fit_3.fit_transform(kidney_data_filled[kidney_data_numeric_columns])

array([[  1.0000e+00,   4.8000e+01,   8.0000e+01, ...,   3.1637e+08,
          2.1091e+05,   1.4061e+02],
       [  1.0000e+00,   7.0000e+00,   5.0000e+01, ...,   1.7280e+08,
          1.3824e+05,   1.1059e+02],
       [  1.0000e+00,   6.2000e+01,   8.0000e+01, ...,   2.7000e+08,
          1.7280e+05,   1.1059e+02],
       [  1.0000e+00,   4.8000e+01,   7.0000e+01, ...,   1.7507e+08,
          1.0191e+05,   5.9319e+01],
       [  1.0000e+00,   5.1000e+01,   8.0000e+01, ...,   2.4513e+08,
          1.5447e+05,   9.7336e+01]])
```

Polynomial Features

This transformation generates every possible pairwise combination of all of the numeric columns in the kidney dataset upto a degree of 3.

So, if we had 3 columns,X,Y,Z, this transformation would generate:

$$(1, X, Y, Z, XY, XZ, YZ, X^2, Y^2, Z^2, X^2Y, X^2Z, \ldots, XYZ, X^3, Y^3, Z^3)$$

However, this is complete overkill. What we usually want to generate is just the interactions (XY.YZ, etc. terms) not all of the polynomial degrees. In that case, we simply pass an extra Boolean parameter to the `PolynomialFeatures` function:

METIS

Polynomial Features

```
poly_fit_3_interact = PolynomialFeatures(degree=3,interaction_only=True)
interactions_only_kidney_columns =
poly_fit_3_interact.fit_transform(kidney_data_filled[kidney_data_numeric_columns])[:,
1:]
```

METIS

Exercise

1. Generate the interactions-only degree-2 polynomial-fit features for the kidney_data null-filled dataset on the numeric columns only.

2. Use train_test_split to predict the class with/without these polynomial features using `LogisticRegression()`. What happens to train/test accuracy?

METIS

Cross Validation

Motivation: We need a way to choose between several different machine learning models (or multiple versions of the same model). Our goal is **to estimate likely performance of a model on out-of-sample data**.

METIS

Cross Validation

Initial approach: Train/test split

- Split the dataset into two pieces, so that the model can be trained and tested on **different data**.

- **Testing accuracy** is a better estimate than training accuracy of out-of-sample performance (because testing on the same data that you used to train the model causes **overfitting** and worse out-of-sample performance).

- However, just using one train/test split provides a **high variance** estimate since changing which observations happen to be in the testing set can significantly change testing accuracy.

What would be a better approach?

METIS

Cross Validation

Better approach: Create a bunch of train/test splits, calculate the testing accuracy for each, and average the results together.

METIS

Cross Validation

1. Split the dataset into K **equal** partitions (or "folds").

2. Use fold 1 as the **testing set** and the union of the other folds as the **training set**.

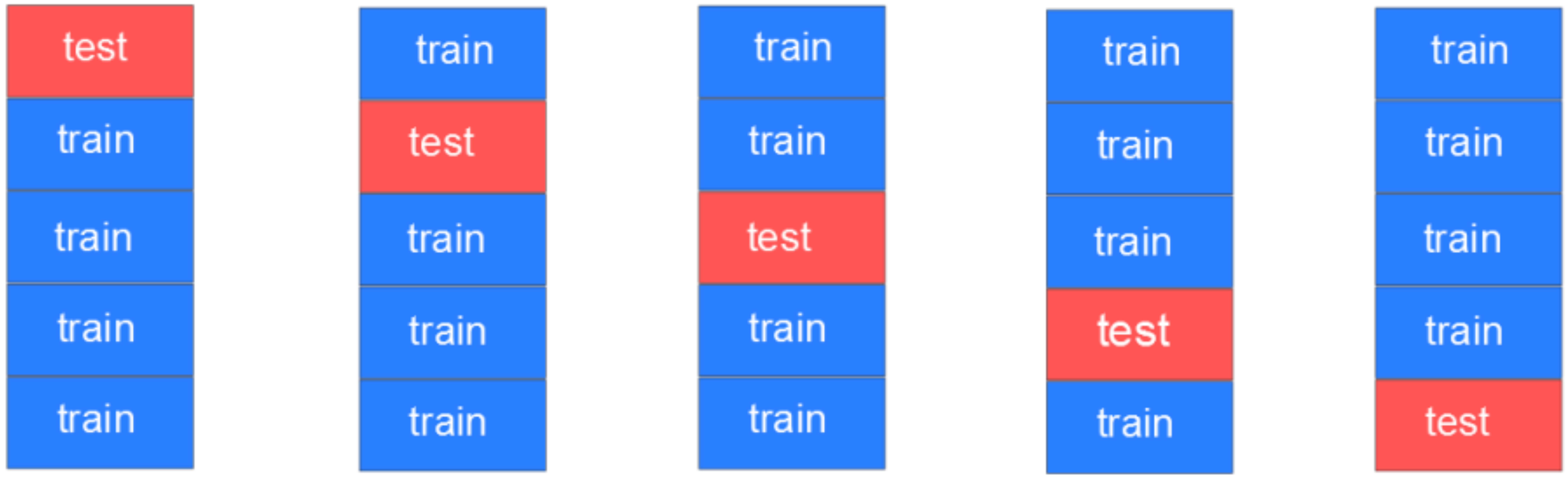
3. Calculate testing accuracy.

4. Repeat steps 2 and 3 K times, using a **different fold** as the testing set each time.

5. Use the **average testing accuracy** as the estimate of out-of-sample accuracy.

METIS

Cross Validation



METIS

Cross Validation

```
from sklearn.model_selection import KFold,
cross_val_score
kf = KFold(n_splits=5, shuffle=False)


from sklearn.cross_validation import cross_val_score,
cross_val_predict
from sklearn import metrics
scores = cross_val_score(<model>, <features>, <target>,
cv=6)
```

METIS

Exercise

Compute cross-validated mean/std of accuracies when doing 20, 40, 50-fold cross-validation using `cross_val_score()`. What happens to the mean and standard deviation of the accuracies as you increase the number of folds?

METIS

Cross Validation

Here are the advantages of **cross-validation:**

- More accurate estimate of out-of-sample accuracy
- More "efficient" use of data (every observation is used for both training and testing)

However, there are some advantages to using **train/test split:**

- Runs K times faster than K-fold cross-validation
- Simpler to examine the detailed results of the testing process

METIS

Cross Validation

We can improve this basic cross-validation approach a bit more as follows:

Repeated cross-validation

- Repeat cross-validation multiple times (with **different random splits** of the data) and average the results
- More reliable estimate of out-of-sample performance by **reducing the variance** associated with a single trial of cross-validation

Creating a hold-out set

- "Hold out" a portion of the data **before** beginning the model building process
- Locate the best model using cross-validation on the remaining data, and test it **using the hold-out set**
- More reliable estimate of out-of-sample performance since hold-out set is **truly out-of-sample**

Feature engineering and selection within cross-validation iterations

- Normally, feature engineering and selection occurs **before** cross-validation
- Instead, you can perform all feature engineering and selection **within each cross-validation iteration**
- This gives us a more reliable estimate of out-of-sample performance since it **better mimics** the application of the model to out-of-sample data

Cross Validation

```
from sklearn.model_selection import GridSearchCV
```

METIS

Cross Validation

```
# try max_depth=2
rf_2 = RandomForestClassifier(max_depth=2, random_state=1)
print("Cross-validated mean accuracy for depth
2:",cross_val_score(rf_2, X, y, cv=10, scoring='accuracy').mean())

# try max_depth=3
rf_3 = RandomForestClassifier(max_depth=3, random_state=1)
print("Cross-validated mean accuracy for depth
3:",cross_val_score(rf_3, X, y, cv=10, scoring='accuracy').mean())

Cross-validated mean accuracy for depth 2: 0.761290322581
Cross-validated mean accuracy for depth 3: 0.787096774194
```

METIS

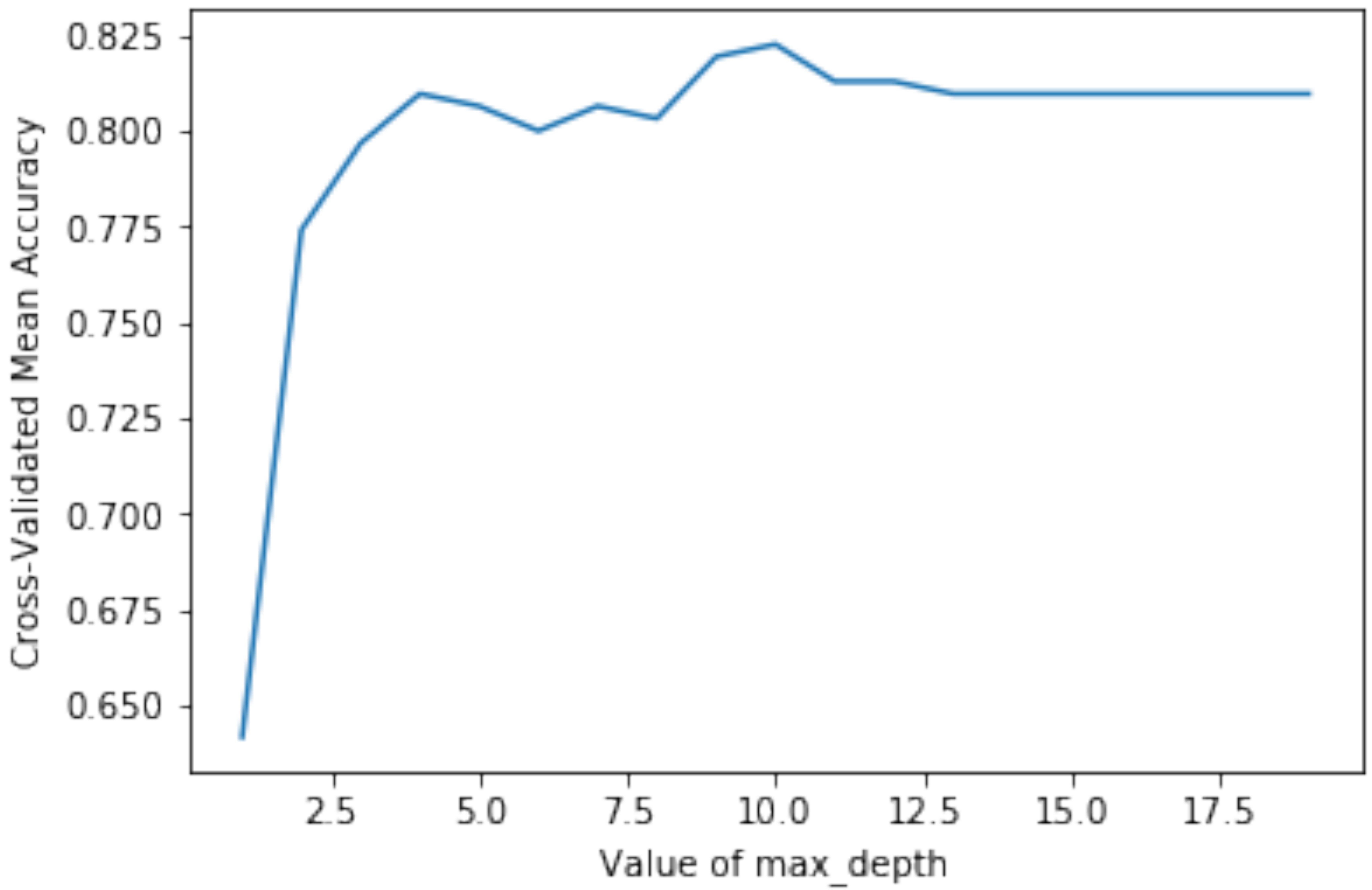
Cross Validation

```
# use GridSearchCV to automate the search across depths 1-10
rf_grid =
RandomForestClassifier(n_estimators=50,random_state=1,n_jobs=-1)
#50 trees
max_depth_range = range(1, 20)
param_grid = dict(max_depth=max_depth_range)
print(param_grid)
grid = GridSearchCV(rf_grid, param_grid, cv=10, scoring='accuracy')
grid.fit(X, y)

# store the results of the grid search
grid.grid_scores_
grid_mean_scores = [result.mean_validation_score for result in
grid.grid_scores_]
```

METIS

Cross Validation



METIS

Exercise

Using the kidney dataset, use and test several different imputation methods, different feature transformations, and different model parameters using `GridSearchCV` for the kidney dataset.

This will get you to use everything we've learned today on one dataset with several kinds of missing values, and using both categorical and numeric values.

Don't forget to transform the categories using `pd.get_dummies()` once you've filled in the missing data!

Remember there are other parameters you can tune for random forests than just the maximum depth. Look at the random forest lesson or the random forest documentation.

Try to be systematic in your exploration. Your goal is to make as robust a model as you can (lowest average cross-validated test error).