Data cleaning：

注意：若需要train/test split则先split后再对train data进行处理。为了避免data leakage，不论是scaling还是imputation，都只能fit train data，对于test data只进行transformation不参与fit。

**Check column data type：**

1. sf\_permits.dtypes 观察每一个column data type
2. pd.to\_numeric()，pd.to\_datetime()，df[“a”].astype(str)

landslides['date\_parsed'] = pd.to\_datetime(landslides['date'], format="%m/%d/%y")

landslides['date\_parsed'] = pd.to\_datetime(landslides['Date'], infer\_datetime\_format=True)

提取具体年月日：

day\_of\_month\_landslides = landslides['date\_parsed'].dt.day

（ 或者在导入时就设置好

df = pd.read\_csv('data/data\_4.csv', parse\_dates={ 'date': ['year', 'month', 'day'] }) ）

**Inconsistent Data Entry：**

对于文字类型data column，应观察其value并判断是否有data entry problem，寻找和修正的具体步骤如下：

1. 观察值a=df[“a”].unique(), a.sort(), a
2. 整体改变大小写，strip空格 df[“a”]= df[“a”].str.lower(),

df[“a”]= df[“a”].str.strip()

1. 确定要对标的值(即最终保留值)，查找与对标值相似的值

Import fuzzywuzzy

From fuzzywuzzy import process

Matches=fuzzywuzzy.process.extract(“对标值”，df[“a”]，limit=10，scorer=fuzzywuzzy.fuzz.token\_sort\_ratio)

Matches

通过观察matches中与对标值相似的值和其score，决定一个score threshold(min\_ratio)，使得threshold之上的值全部改为对标值

1. def replace\_matches\_in\_column(df, column, string\_to\_match, min\_ratio = 47):

# get a list of unique strings

strings = df[column].unique()

# get the top 10 closest matches to our input string

matches = fuzzywuzzy.process.extract(string\_to\_match, strings,

limit=10, scorer=fuzzywuzzy.fuzz.token\_sort\_ratio)

# only get matches with a ratio >min\_ratio

close\_matches = [matches[0] for matches in matches if matches[1] >= min\_ratio]

# get the rows of all the close matches in our dataframe

rows\_with\_matches = df[column].isin(close\_matches)

# replace all rows with close matches with the input matches

df.loc[rows\_with\_matches, column] = string\_to\_match

# let us know the function's done

print("All done!")

1. call the function

replace\_matches\_in\_column(df=professors, column='Country',

string\_to\_match="usa",min\_ratio=75)

1. 最后再inspect一下改变后的column value，

Professor[“country”].unique()

**Missing values：**

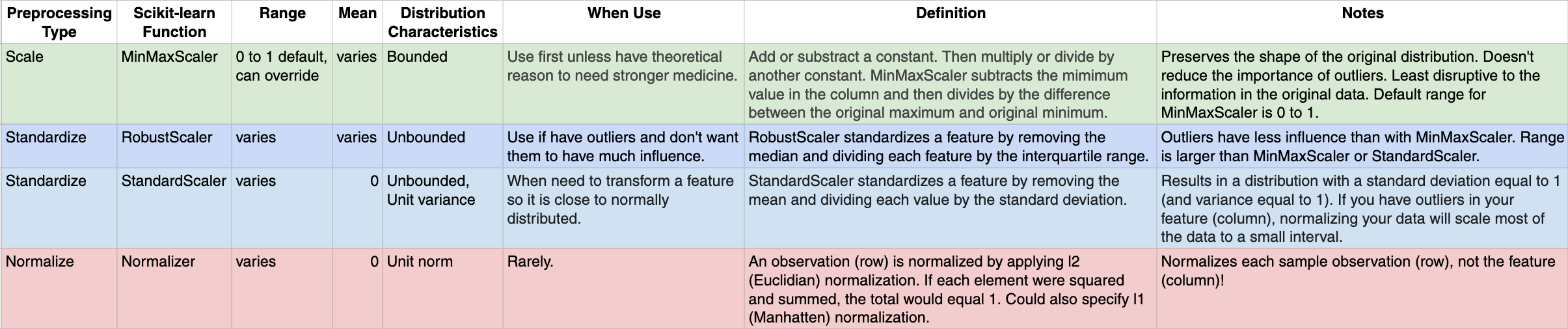
1. 先看missing数量，df.isnull().sum()
2. 确定missing的原因，是因为缺失数据还是因为本身就没有数据（无小孩家的小孩身高）
3. 决定是drop掉，还是impute
4. 如果drop，drop row还是column；如果impute，用什么impute，mean/std/mode/zero还是对于logical order的row data用ffill/bfilll（ffill: propagate last valid observation forward to next valid backfill / bfill: use next valid observation to fill gap）

Ps：用ffill时第一行可能有missing，用bfill则最后一行可能有missing

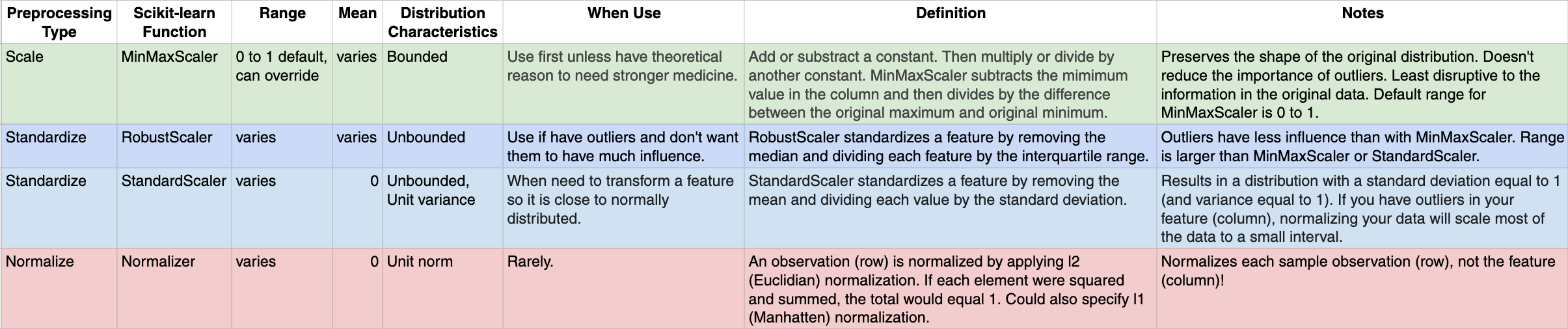
**Scaling/Standardizing/Boxcox(range and distribution):**

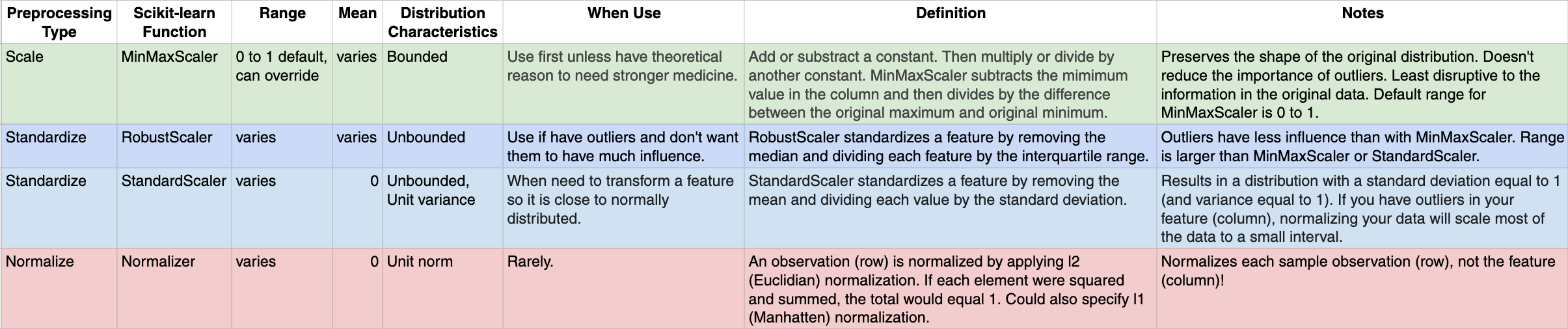
**注意一些value不应该被scale，比如id和经纬度**

1. 一些算法比如KNN，SVM，LDA，PCA会受到scale的影响，所以数据需要先调整至统一的scale下。MinMaxScaler不改变distribution只改变数据range到（0,1）之间，若希望最大程度保留数据原本distribution建议用MinMaxScaler



1. StandardScaler 使数据统一mean为0，std为1，如果需要数据的distribution接近于normal distribution则推荐用，为第二首选项





1. 若需要数据非常接近与normal distribution，则可以使用BoxCox transformation：

from scipy import stats，

normalized\_data = stats.boxcox(original\_data)[0]

忽略return的第二个output

注意boxcox只接受正数，所以建议先minmaxscale再boxcox

Feature engineering：

**Creating features:**

一共五种创建新feature的方法

1. mathematical transform
2. 对其他column的总结性column，如前五个column为事故类型，可创建一个column记录有无事故，或者事故数量大于三的row

roadway\_features = ["Amenity", "Bump", "Crossing", "GiveWay",

"Junction", "NoExit", "Railway", "Roundabout", "Station", "Stop",

"TrafficCalming", "TrafficSignal"]

accidents["RoadwayFeatures"] = accidents[roadway\_features].sum(axis=1)

or：

components = [ "Cement", "BlastFurnaceSlag", "FlyAsh", "Water",

"Superplasticizer", "CoarseAggregate", "FineAggregate"]

concrete["Components"] = concrete[components].gt(0).sum(axis=1)

1. Building-Up and Breaking-Down Features

以下都可以break down，可以全部split保存也可以只抽取重要的info

ID numbers: '123-45-6789'

Phone numbers: '(999) 555-0123'

Street addresses: '8241 Kaggle Ln., Goose City, NV'

Internet addresses: 'http://www.kaggle.com

Product codes: '0 36000 29145 2'

Dates and times: 'Mon Sep 30 07:06:05 2013'

customer[["Type", "Level"]] = ( *# Create two new features*

customer["Policy"] *# from the Policy feature*

.str *# through the string accessor*

.split(" ", expand=True) *# by splitting on " "*

*# and expanding the result into separate columns*

)

Building-up

autos["make\_and\_style"] = autos["make"] + "\_" + autos["body\_style"]

1. Group Transforms

将同一种类的row的numerical value进行aggregate，但是不压缩row，用transform去将aggregate的value填充到各个row

customer["AverageIncome"] = (

customer.groupby("State") *# for each state*

["Income"] *# select the income*

.transform("mean") *# and compute its mean*

)

Or：

customer["StateFreq"] = (

customer.groupby("State")

["State"]

.transform("count")

/ customer.State.count()

)

如果要考虑到train\_test\_split, 则

*# Create the average claim amount by coverage type, on the training set*

df\_train["AverageClaim"] = df\_train.groupby("Coverage")["ClaimAmount"].transform("mean")

*# Merge the values into the validation set*

df\_valid = df\_valid.merge(

df\_train[["Coverage", "AverageClaim"]].drop\_duplicates(),

on="Coverage",

how="left"

)

**K-means Clustering:**

1.可利用k-means clustering为observations分配不同的clusters（类似于bining），将clusterid当做一项categorical feature

*# Create cluster feature*

kmeans = KMeans(n\_clusters=6)

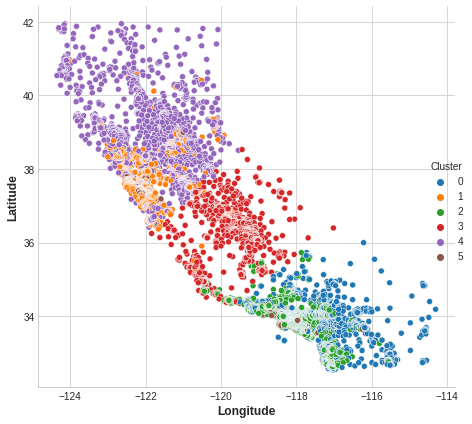
X["Cluster"] = kmeans.fit\_predict(X)

X["Cluster"] = X["Cluster"].astype("category")

sns.relplot(

x="Longitude", y="Latitude", hue="Cluster", data=X, height=6,

);



**Mutual Information:**

根据feature与target的关系程度建立ranking排名.

并可以根据排名选择部分关系程度高的feature进行建模，增强模型可解释性，降低计算复杂度，减少过拟合。

MI与correlation不一样，correlation只能寻找linear relationship而mutual information可以寻找任何种类的关系，但其限制是只能观察两者之间的关系(feature和target)，而有些features需要与其他features结合才能对target提供有效信息。

具体步骤：

1. mutual information前提是dataframe为numeric value，因此categorical feature需要进行encoding(onehot/pd.get\_dummies使feature增多)。

记录下categorical features的名字并且确认其dtype均为int。

X = df.copy()

y = X.pop("price")

from sklearn.preprocessing import LabelEncoder

*# Label encoding for categoricals*

label\_encoder=LabelEncoder()

for colname **in** X.select\_dtypes("object"):

X[colname] = label\_encoder.fit\_transform(X[colname])

*# All discrete features should now have integer dtypes (double-check this before using MI!)*

discrete\_features = X.dtypes == int

1. 假如target 为continuous就用mutual\_info\_regression, 为discrete就用mutual\_info\_classif

from sklearn.feature\_selection import mutual\_info\_regression

def make\_mi\_scores(X, y, discrete\_features):

mi\_scores = mutual\_info\_regression(X, y, discrete\_features=discrete\_features)

mi\_scores = pd.Series(mi\_scores, name="MI Scores", index=X.columns)

mi\_scores = mi\_scores.sort\_values(ascending=False)

return mi\_scores

mi\_scores = make\_mi\_scores(X, y, discrete\_features)

curb\_weight 1.526026

highway\_mpg 0.958583

length 0.615287

bore 0.496247

Name: MI Scores, dtype: float64

1. 作图以便于观察

def plot\_mi\_scores(scores):

scores = scores.sort\_values(ascending=True)

width = np.arange(len(scores))

ticks = list(scores.index)

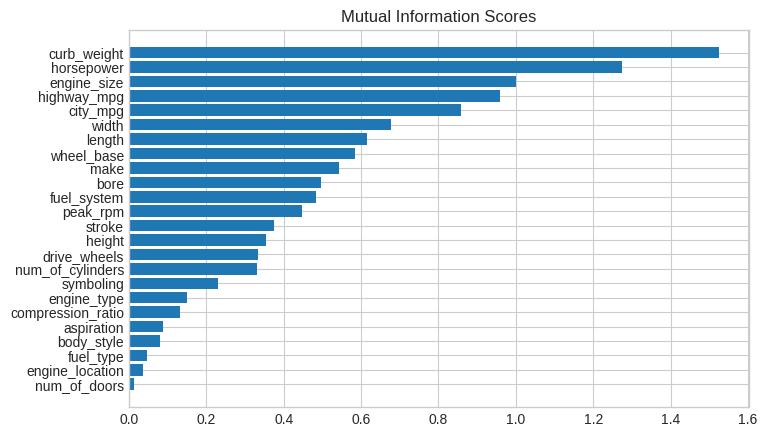
plt.barh(width, scores)

plt.yticks(width, ticks)

plt.title("Mutual Information Scores")

plt.figure(dpi=100, figsize=(8, 5))

plot\_mi\_scores(mi\_scores)



**PCA:**

PCA可以帮助投影高纬度数据至低纬度，便于可视化观察数据pattern，也可以用来制造informative features

注意：在使用PCA之前数据必须先被standardized

PCA分score和loading，score指的是PCA fit\_transform原数据之后得到降噪后的新数据，其dimension为（observations，number of PCs）; loading指的是每个PC由各个features构成的比例，其dimension为（features，number of PCs）

要确定number of PCs应首先确定一个threshold for total explained variance。

用PCA时你的principal components可以是有很多个，但是要在2D图上能visualize出来的只有两个explained variance最多的主PC，分别代表x轴和y轴。在此图上，features可以以射线轴的方式表达，数据以点的方式表达。