Feature Engineering

Name

Getting Started

- Colab - get notebook from our gitmystuff repository
- Save a Copy in Drive
- Remove Copy of
- Edit name
- Take attendance
- Clean up Colab Notebooks folder
- Submit shared link

Feature Engineering Defined

Feature engineering is the process of transforming raw data into meaningful features that can be used by machine learning models to improve their accuracy by selecting, manipulating, and creating relevant input variables from the original data, essentially making the data more suitable for prediction tasks; it's considered a crucial step in the machine learning workflow, often involving techniques like data scaling, encoding categorical variables, and combining features to reveal patterns that might not be readily apparent in the raw data.

Key points about feature engineering: • Goal: To create features that best represent the underlying patterns in the data, allowing the machine learning model to learn more effectively and make better predictions.

- Process steps:
    - Data cleaning: Handling missing values, outliers, and inconsistencies.
    - Feature selection: Choosing the most relevant variables from the data.
    - Feature transformation: Applying mathematical operations like scaling, normalization, or log transformation to features.
    - Feature creation: Combining existing features to create new, potentially more predictive features.
    - Encoding: Transforming categorical data into numerical representations.
- Importance: Feature engineering can significantly impact the performance of a machine learning model, often making the difference between good and poor predictions.

Derived Variables

age and salary usually are correlated but ambition can create outliers because a younger person can make a million off a great idea or an older person may be an artist etc.

Ambition = YearsExperience / Age

The concept of derived variables can be used, in certain ways, to mitigate the effects of multicollinearity. However, it's crucial to understand that it's not a universal "fix," and it needs to be applied carefully. Here's how it works:

How Derived Variables Can Help:

- **Combining Correlated Variables:**
  - If you have two or more highly correlated variables that represent a similar underlying concept, you can create a single derived variable that captures that concept.
  - For example, if you have variables for "height in inches" and "height in centimeters," which are perfectly correlated, you could create a single "height" variable.
  - This reduces the redundancy and eliminates the direct multicollinearity between the original variables.
- **Creating Ratio or Index Variables:**
  - Sometimes, multicollinearity arises from using raw values of related variables. You can create ratio or index variables that represent the relationship between those variables.
  - For instance, instead of using "income" and "number of people in household" separately, you could create a "income per capita" variable.
  - This can reduce multicollinearity by focusing on the relative relationship rather than the absolute values.
- **Principal Component Analysis (PCA):**
  - PCA is a more advanced technique that creates derived variables called principal components. These components are linear combinations of the original variables and are designed to be uncorrelated.
  - PCA effectively transforms the original correlated variables into a new set of uncorrelated variables, addressing multicollinearity.
  - This is a form of creating derived variables, that are then used in place of the original variables.

Important Considerations:

- **Information Loss:**
  - Combining variables or creating indices can lead to some loss of information. You need to carefully consider whether the derived variable adequately captures the information you need.

- **Interpretability:**
  - Derived variables can sometimes be more difficult to interpret than the original variables. This is especially true with techniques like PCA, where the principal components may not have a clear real-world meaning.
- **Domain Knowledge:**
  - The decision of how to create derived variables to address multicollinearity should be guided by domain knowledge. You need to understand the relationships between your variables and choose transformations that make sense in the context of your problem.
- **Not a Universal Solution:**
  - Derived variables are not always the best solution for multicollinearity. Sometimes, simpler approaches like removing one of the correlated variables or using regularization techniques may be more appropriate.

In summary:

Derived variables can be a valuable tool for addressing multicollinearity, particularly when you can combine correlated variables or create meaningful ratios or indices. However, it's essential to use them judiciously and consider the potential trade-offs.

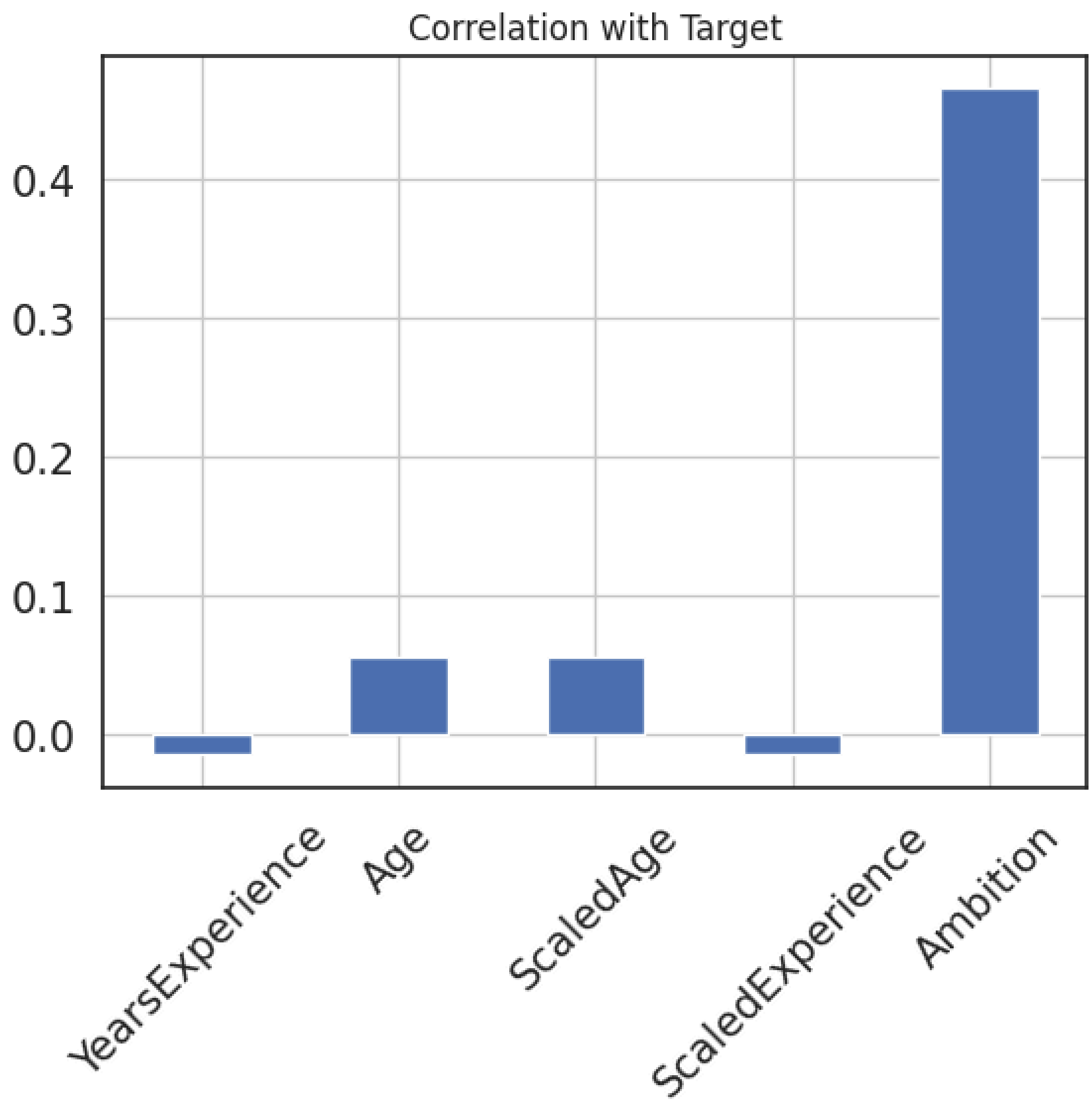
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler

def calculate_ambition(df, age_weight=0.5, experience_weight=0.5, salary_weight=0.5):
    """
    Calculates an "ambition" score based on age, experience, and salary.

    Args:
        df (pd.DataFrame): DataFrame with 'Age', 'YearsExperience', and 'Salary' columns
        age_weight (float): Weight for age in the ambition calculation.
        experience_weight (float): Weight for years of experience.
        salary_weight (float): Weight for salary.

    Returns:
        pd.DataFrame: DataFrame with an added 'Ambition' column.
    """

    if not all(col in df.columns for col in ['Age', 'YearsExperience', 'Salary']):
      raise ValueError("DataFrame must contain 'Age', 'YearsExperience', and 'Salary' co

    # Standardize the features
    scaler = StandardScaler()
    scaled_features = scaler.fit_transform(df[['Age', 'YearsExperience', 'Salary']])
    scaled_df = pd.DataFrame(scaled_features, columns=['ScaledAge', 'ScaledExperience',
    df = pd.concat([df.reset_index(drop=True), scaled_df], axis=1) #reset index to avoid

    # Calculate the ambition score
    df['Ambition'] = (
```

```
        (df['ScaledAge'] * age_weight) +
        (df['ScaledExperience'] * experience_weight) +
        (df['ScaledSalary'] * salary_weight)
    )

    return df

# Example Usage (with your sample data):
data = {'YearsExperience': [1.1, 1.3, 1.5, 2.0, 2.2],
        'Age': [21.0, 21.5, 21.7, 22.0, 22.2],
        'Salary': [39343, 46205, 37731, 43525, 39891]}
df = pd.DataFrame(data)

df = calculate_ambition(df)
df.head()
```

	YearsExperience	Age	Salary	ScaledAge	ScaledExperience	ScaledSalary	Ambition
0	1.1	21.0	39343	-1.632052	-1.248040	-0.647214	-1.763653
1	1.3	21.5	46205	-0.432014	-0.768025	1.577827	0.188894
2	1.5	21.7	37731	0.048002	-0.288009	-1.169913	-0.704961
3	2.0	22.0	43525	0.768025	0.912029	0.708822	1.194438
4	2.2	22.2	39891	1.248040	1.392045	-0.469522	1.085281

```
# correlation matrix
sns.set(style="white")

# compute the correlation matrix
corr = df.corr()

# generate a mask for the upper triangle
mask = np.zeros_like(corr, dtype=bool)
mask[np.triu_indices_from(mask)] = True

# set up the matplotlib figure
f, ax = plt.subplots()

# generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5}, annot=True);
```


```
# showing correlation of multiple features with one target
df.drop(['ScaledSalary', 'Salary'], axis=1).corrwith(df['Salary']).plot.bar(
        title = "Correlation with Target", fontsize = 15,
        rot = 45, grid = True);
```



Example of Engineering a Feature by Transforming its Values

Logarithm and Moore's Law

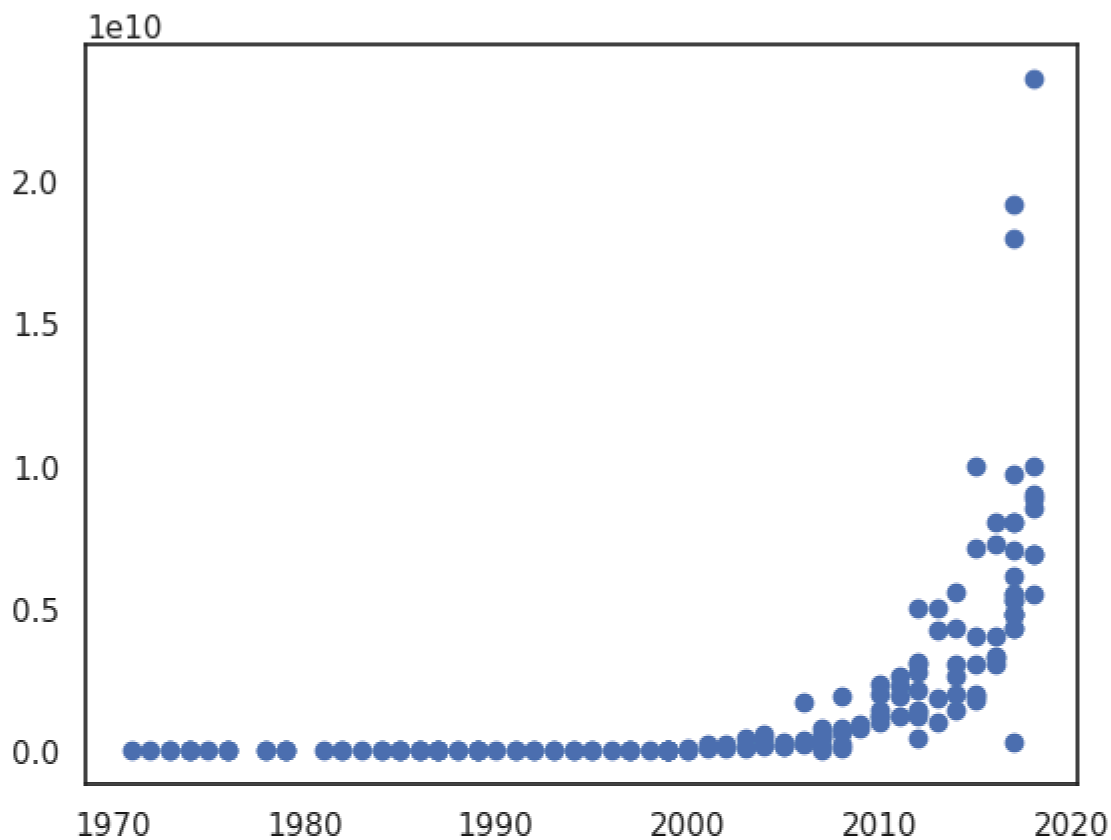
Moore's law is the observation that the number of transistors in a dense integrated circuit (IC) doubles about every two years. Moore's law is an observation and projection of a historical trend. Rather than a law of physics, it is an empirical relationship linked to gains from experience in production.

https://en.wikipedia.org/wiki/Moore's_law

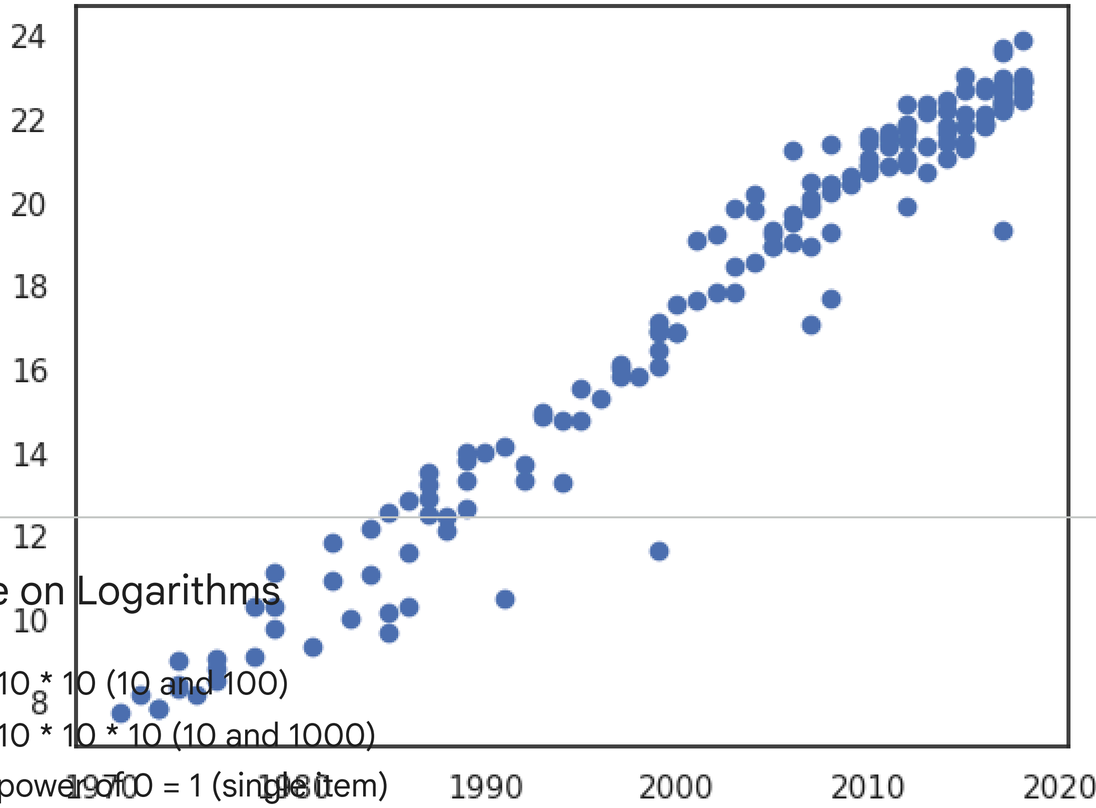
```
# get the data
import pandas as pd

moores = pd.read_csv('https://raw.githubusercontent.com/lazyprogrammer/machine_learning_e
moores.columns = ['year', 'transistors']
moores.head()
```

	year	transistors
0	1971	2300
1	1972	3500
2	1973	2500
3	1973	2500
4	1974	4100

```
# plot the data
import matplotlib.pyplot as plt

plt.scatter(moores['year'], moores['transistors']);
```




```
# apply log to transistors
import numpy as np

moores['log_trans'] = np.log(moores.transistors)
plt.scatter(moores['year'], moores['log_trans']);
```



More on Logarithms

- 10 * 10 (10 and 100)
- 10 * 10 * 10 (10 and 1000)
- power of 0 = 1 (single item)
- power of 1 = 10
- power of 3 = thousand
- power of 6 = million
- power of 9 = billion
- power of 12 = trillion
- power of 23 = number of molecules in a dozen grams of carbon
- power of 80 = number of molecules in the universe

A 0 to 80 scale took us from a single item to the number of things in the universe.

https://betterexplained.com/articles/using-logs-in-the-real-world/

Imputation

Mean, Median, Mode Imputation

- Mean if normal
- Median if skewed
- Used for MCAR

```
# get data
import pandas as pd

houses = pd.read_csv('https://raw.githubusercontent.com/gitmystuff/Datasets/main/house-p
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(
    houses.drop('SalePrice', axis=1),
    houses['SalePrice'],
    test_size=0.25,
    random_state=42)

X_train.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandConto
1023	1024	120	RL	43.0	3182	Pave	NaN	Reg	l
810	811	20	RL	78.0	10140	Pave	NaN	Reg	l
1384	1385	50	RL	60.0	9060	Pave	NaN	Reg	l
626	627	20	RL	NaN	12342	Pave	NaN	IR1	l
813	814	20	RL	75.0	9750	Pave	NaN	Reg	l

5 rows × 80 columns

```
# find nulls
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1095 entries, 1023 to 1126
Data columns (total 80 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             1095 non-null   int64  
 1   MSSubClass     1095 non-null   int64  
 2   MSZoning       1095 non-null   object 
 3   LotFrontage    895 non-null    float64
 4   LotArea        1095 non-null   int64  
 5   Street         1095 non-null   object 
 6   Alley          70 non-null     object 
 7   LotShape       1095 non-null   object 
 8   LandContour    1095 non-null   object 
 9   Utilities      1095 non-null   object 
 10  LotConfig      1095 non-null   object 
 11  LandSlope      1095 non-null   object 
 12  Neighborhood   1095 non-null   object 
 13  Condition1     1095 non-null   object 
 14  Condition2     1095 non-null   object 
 15  BldgType       1095 non-null   object 
 16  HouseStyle     1095 non-null   object 
 17  OverallQual    1095 non-null   int64  
 18  OverallCond    1095 non-null   int64  
 19  YearBuilt      1095 non-null   int64  
 20  YearRemodAdd   1095 non-null   int64  
 21  RoofStyle      1095 non-null   object 
 22  RoofMatl       1095 non-null   object 
 23  Exterior1st    1095 non-null   object 
 24  Exterior2nd    1095 non-null   object 
 25  MasVnrType     456 non-null    object 
 26  MasVnrArea     1091 non-null   float64
 27  ExterQual      1095 non-null   object 
```

```
 28  ExterCond      1095 non-null   object
 29  Foundation     1095 non-null   object
 30  BsmtQual       1068 non-null   object
 31  BsmtCond       1068 non-null   object
 32  BsmtExposure   1068 non-null   object
 33  BsmtFinType1   1068 non-null   object
 34  BsmtFinSF1     1095 non-null   int64
 35  BsmtFinType2   1068 non-null   object
 36  BsmtFinSF2     1095 non-null   int64
 37  BsmtUnfSF      1095 non-null   int64
 38  TotalBsmtSF    1095 non-null   int64
 39  Heating        1095 non-null   object
 40  HeatingQC      1095 non-null   object
 41  CentralAir     1095 non-null   object
 42  Electrical     1094 non-null   object
 43  1stFlrSF       1095 non-null   int64
 44  2ndFlrSF       1095 non-null   int64
 45  LowQualFinSF   1095 non-null   int64
 46  GrLivArea      1095 non-null   int64
 47  BsmtFullBath   1095 non-null   int64
 48  BsmtHalfBath   1095 non-null   int64
 49  FullBath       1095 non-null   int64
 50  HalfBath       1095 non-null   int64
 51  BedroomAbvGr   1095 non-null   int64
 52  KitchenAbvGr   1095 non-null   int64
```

```
X_train.isnull().sum()
```

	0
Id	0
MSSubClass	0
MSZoning	0
LotFrontage	200
LotArea	0
...	...
MiscVal	0
MoSold	0
YrSold	0
SaleType	0
SaleCondition	0

80 rows × 1 columns

dtype: int64

```
for feat in X_train.columns:
    if X_train[feat].isnull().any():
        print(feat, X_train[feat].isnull().sum())
```

```
LotFrontage 200
Alley 1025
MasVnrType 639
MasVnrArea 4
BsmtQual 27
BsmtCond 27
BsmtExposure 27
BsmtFinType1 27
BsmtFinType2 27
Electrical 1
FireplaceQu 512
GarageType 58
GarageYrBlt 58
GarageFinish 58
GarageQual 58
GarageCond 58
PoolQC 1089
Fence 877
MiscFeature 1052
```

```
nulls = [feat for feat in X_train.columns if X_train[feat].isnull().any()]
nulls
```

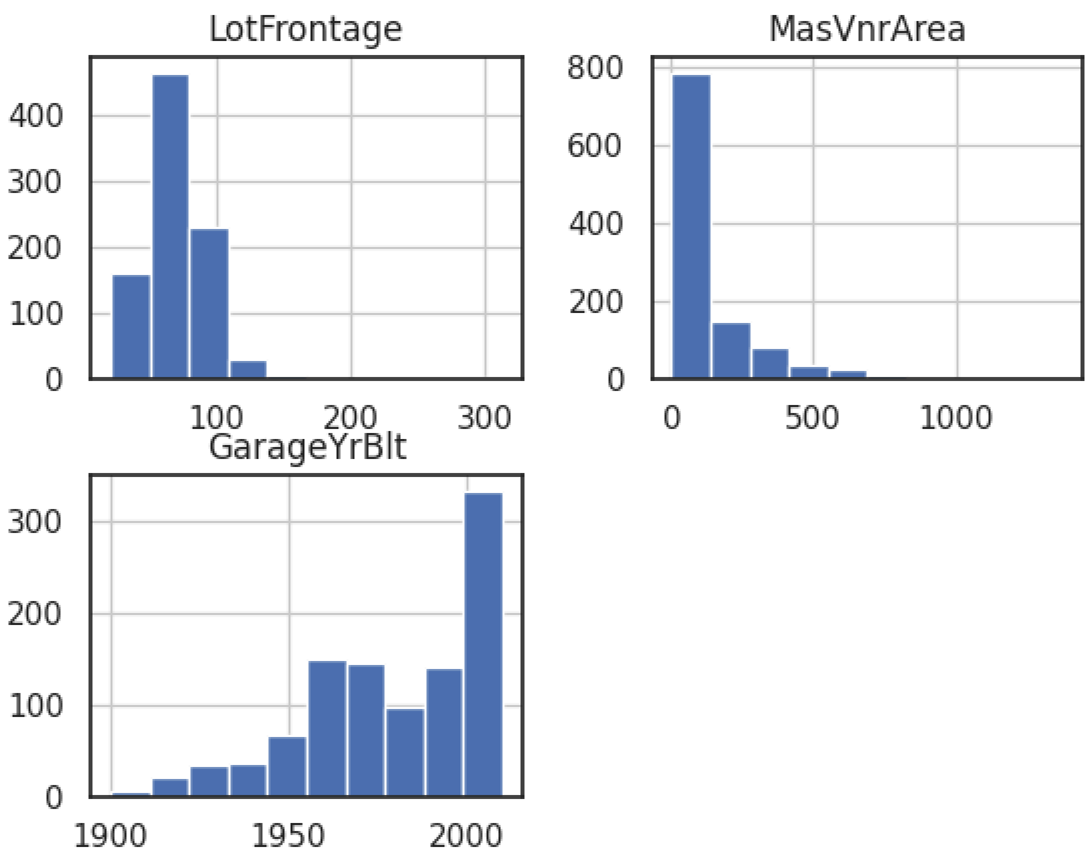
```
['LotFrontage',
 'Alley',
 'MasVnrType',
 'MasVnrArea',
 'BsmtQual',
 'BsmtCond',
 'BsmtExposure',
 'BsmtFinType1',
 'BsmtFinType2',
 'Electrical',
 'FireplaceQu',
 'GarageType',
 'GarageYrBlt',
 'GarageFinish',
 'GarageQual',
 'GarageCond',
 'PoolQC',
 'Fence',
 'MiscFeature']
```

```
# example of some nulls
X_train[['LotFrontage', 'MasVnrArea', 'GarageYrBlt']].isnull().sum()
```

	0
LotFrontage	200
MasVnrArea	4
GarageYrBlt	58

dtype: int64

```
X_train[['LotFrontage', 'MasVnrArea', 'GarageYrBlt']].hist();
```




```
# fill na with mean median mode
mmm = pd.DataFrame(columns = ['LotFrontage', 'MasVnrArea', 'GarageYrBlt'])
mmm['LotFrontage'] = X_train['LotFrontage'].fillna(round(X_train['LotFrontage'].mean(), :
mmm['MasVnrArea'] = X_train['MasVnrArea'].fillna(X_train['MasVnrArea'].median())
mmm['GarageYrBlt'] = X_train['GarageYrBlt'].fillna(X_train['GarageYrBlt'].mode()[0])
mmm.isnull().sum()
```

	0
LotFrontage	0
MasVnrArea	0
GarageYrBlt	0

dtype: int64

Arbitrary Constants

- Discovers if MNAR
- Goal is to flag missing values
- Use values not in distribution
- Importance of missingness if present
- Depends on the model (Linear models maybe not because more arbitrary values in distribution, Trees maybe)

We don't want to impute mean, median, etc because it looks like the data. We want to emphasize the missing data because we believe it's missing not at random.

```
# recall missing values (non-null)
print(X_train.shape)
print(X_train[nulls].info())
```

```
(1095, 80)
<class 'pandas.core.frame.DataFrame'>
Index: 1095 entries, 1023 to 1126
Data columns (total 19 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   LotFrontage   895 non-null    float64
 1   Alley         70 non-null     object 
 2   MasVnrType    456 non-null    object 
 3   MasVnrArea    1091 non-null   float64
 4   BsmtQual      1068 non-null   object 
 5   BsmtCond      1068 non-null   object 
 6   BsmtExposure  1068 non-null   object 
 7   BsmtFinType1  1068 non-null   object 
 8   BsmtFinType2  1068 non-null   object 
 9   Electrical    1094 non-null   object 
 10  FireplaceQu   583 non-null    object 
 11  GarageType    1037 non-null   object 
 12  GarageYrBlt   1037 non-null   float64
 13  GarageFinish  1037 non-null   object 
 14  GarageQual    1037 non-null   object 
 15  GarageCond    1037 non-null   object 
 16  PoolQC        6 non-null      object 
 17  Fence         218 non-null    object 
 18  MiscFeature   43 non-null     object 
dtypes: float64(3), object(16)
memory usage: 171.1+ KB
None
```

```
X_train['GarageType'].fillna('Missing', inplace=True)
X_train['GarageType'].value_counts()
```

```
<ipython-input-31-8ae2afe1a60e>:1: FutureWarning: A value is trying to be set on a copy of
The behavior will change in pandas 3.0. This inplace method will never work because the ir

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col:


  X_train['GarageType'].fillna('Missing', inplace=True)
```

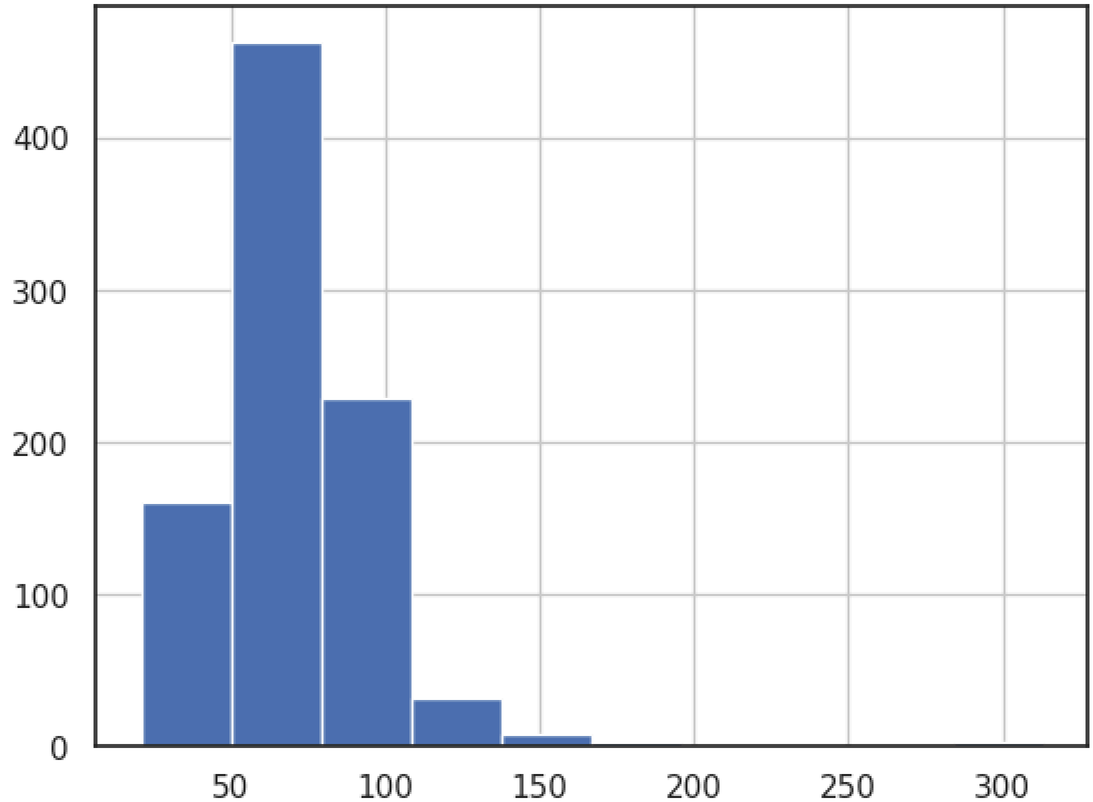
	count
GarageType	
Attchd	651
Detchd	289
BuiltIn	69
Missing	58
Basment	15
CarPort	7
2Types	6

dtype: int64

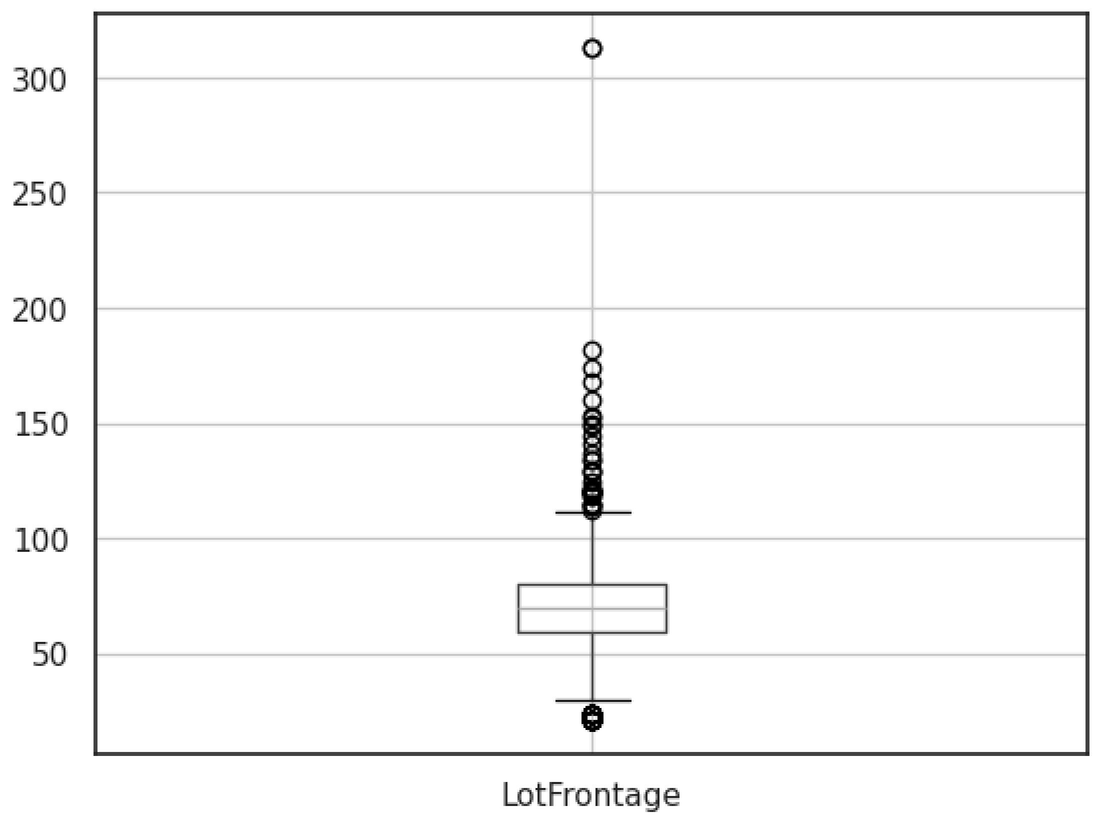
End of Distribution

- If normal we can use -3, 3 standard deviations
- If skewed we can use IQR proximity rule (iqr x 1.5, or iqr x 3)
- Flag the missing value where observations are rarely represented
- Used in finances

```
# histogram of LotFrontage
X_train['LotFrontage'].hist();
```




```
# boxplot of LotFrontage
X_train.boxplot('LotFrontage');
```

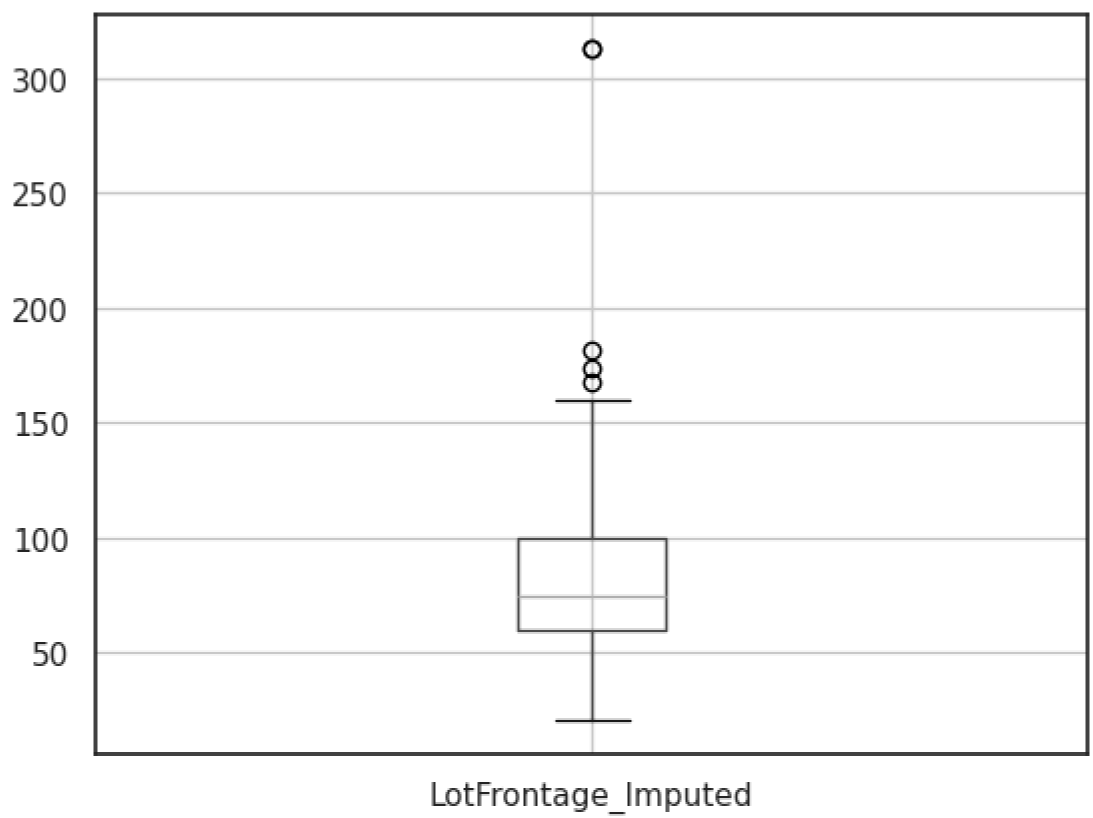



```
# iqr as na
iqr = X_train['LotFrontage'].quantile(0.75) - X_train['LotFrontage'].quantile(0.25)
end_of_distribution = X_train['LotFrontage'].quantile(0.75) + (1.5 * iqr)
X_train['LotFrontage_Imputed'] = X_train['LotFrontage'].fillna(end_of_distribution)
```

```
print(end_of_distribution)
print(X_train['LotFrontage_Imputed'])
```

```
111.5
1023     43.0
810      78.0
1384     60.0
626     111.5
813      75.0
        ...
1095     78.0
1130     65.0
1294     60.0
860      55.0
1126     53.0
Name: LotFrontage_Imputed, Length: 1095, dtype: float64
```

```
X_train.boxplot('LotFrontage_Imputed');
```



Categorical Encoding

- Sklearn One Hot Encoding
- Dummy Trap
- Pandas get_dummies
- Labelizer
- Weight of Evidence
- Frequency Encoding

Categorical Data

- Nominal (Cat or Dog)
- Ordinal (Grades)
- Works better for limited labels in a category
- Engineer features with many labels

Multicollinearity

- Predictors need to be independent of each other
- https://www.theanalysisfactor.com/multicollinearity-explained-visually/
- https://statisticsbyjim.com/regression/multicollinearity-in-regression-analysis/
- Cats_and_Dogs = [Cat, Dog, Dog, Cat, Cat, Dog]
- Cats = [1, 0, 0, 1, 1, 0]
- Dogs = [0, 1, 1, 0, 0, 1]

Mismatch in Training and Test

- Some labels in the train set don't show up in the test set

https://towardsdatascience.com/beware-of-the-dummy-variable-trap-in-pandas-727e8e6b8bde

One Hot Encoder

```
# sklearn OneHotEncoder
# https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.h
# https://stackoverflow.com/questions/50473381/scikit-learns-labelbinarizer-vs-onehotenc
import numpy as np
import pandas as pd
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import LabelBinarizer

pets = ['dog', 'cat', 'cat', 'dog', 'turtle', 'cat', 'cat', 'turtle', 'dog', 'cat']
print('cat = 0; dog = 1; turtle = 2')
le = LabelEncoder()
int_values = le.fit_transform(pets)
print('Pets:', pets)
print('Label Encoder:', int_values)
int_values = int_values.reshape(len(int_values), 1)
print(pd.Series(pets))

ohe = OneHotEncoder(sparse_output=False)
ohe = ohe.fit_transform(int_values)
print('One Hot Encoder:\n', ohe)

lb = LabelBinarizer()
print('Label Binarizer:\n', lb.fit_transform(int_values))
```

```
cat = 0; dog = 1; turtle = 2
Pets: ['dog', 'cat', 'cat', 'dog', 'turtle', 'cat', 'cat', 'turtle', 'dog', 'cat']
Label Encoder: [1 0 0 1 2 0 0 2 1 0]
```

```
0       dog
1       cat
2       cat
3       dog
4    turtle
5       cat
6       cat
7    turtle
8       dog
9       cat
dtype: object
One Hot Encoder:
 [[0. 1. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]]
Label Binarizer:
 [[0 1 0]
 [1 0 0]
 [1 0 0]
 [0 1 0]
 [0 0 1]
 [1 0 0]
 [1 0 0]
 [0 0 1]
 [0 1 0]
 [1 0 0]]
```

```
pets = pd.DataFrame(pd.Series(pets), columns=['Pets'])
pets.head()
```

	Pets
0	dog
1	cat
2	cat
3	dog
4	turtle

```
ohe = OneHotEncoder(sparse_output=False)
ohe_pets = ohe.fit_transform(pets)
pets_df = pd.DataFrame(ohe_pets, columns=ohe.get_feature_names_out(['Pets']))
pets_df
```

	Pets_cat	Pets_dog	Pets_turtle
0	0.0	1.0	0.0
1	1.0	0.0	0.0
2	1.0	0.0	0.0
3	0.0	1.0	0.0
4	0.0	0.0	1.0
5	1.0	0.0	0.0
6	1.0	0.0	0.0
7	0.0	0.0	1.0
8	0.0	1.0	0.0
9	1.0	0.0	0.0

```
# pip install Faker
```

```
import numpy as np
import pandas as pd
from faker import Faker
fake = Faker()

output = []
for x in range(100):
  sex = np.random.choice(['egg', 'seed'], p=[0.5, 0.5])
  output.append(
    {
        'sex': sex,
        'brain_wave': np.random.choice(['BETA', 'ALPHA', 'THETA']),
        'given_name': fake.first_name_female() if sex=='egg' else fake.first_name_male(),
        'surname': fake.last_name(),
        'space_zone': fake.zipcode(),
        })

df = pd.DataFrame(output)
print(df.shape)
print(df.info())
df.head()
```

```
(100, 5)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   sex         100 non-null    object
 1   brain_wave  100 non-null    object
 2   given_name  100 non-null    object
 3   surname     100 non-null    object
 4   space_zone  100 non-null    object
dtypes: object(5)
memory usage: 4.0+ KB
None
```

	sex	brain_wave	given_name	surname	space_zone
0	seed	ALPHA	Douglas	Yang	78986
1	egg	THETA	Morgan	Murphy	63501
2	seed	BETA	Kurt	Hardin	22381
3	egg	BETA	Briana	Schneider	52701
4	seed	ALPHA	James	Nelson	19191

```
dummy_cols = ['sex', 'space_zone', 'brain_wave']
df_dummies = pd.get_dummies(df, columns=dummy_cols)

print(df_dummies.shape)
df_dummies.head()
```

```
(100, 107)
```

	given_name	surname	sex_egg	sex_seed	space_zone_02187	space_zone_04186	space_zone
0	Douglas	Yang	False	True	False	False	
1	Morgan	Murphy	True	False	False	False	
2	Kurt	Hardin	False	True	False	False	
3	Briana	Schneider	True	False	False	False	
4	James	Nelson	False	True	False	False	

5 rows × 107 columns

Dummy Trap

The Dummy Variable Trap occurs when two or more dummy variables created by one-hot encoding are highly correlated (multi-collinear). This means that one variable can be predicted from the others, making it difficult to interpret predicted coefficient variables in regression models. In other words, the individual effect of the dummy variables on the prediction model can not be interpreted well because of multicollinearity.

https://www.learndatasci.com/glossary/dummy-variable-trap/

```
pets_df.corr()
```

	Pets_cat	Pets_dog	Pets_turtle
Pets_cat	1.000000	-0.654654	-0.500000
Pets_dog	-0.654654	1.000000	-0.327327
Pets_turtle	-0.500000	-0.327327	1.000000

```
ohe = OneHotEncoder(drop='first', sparse_output=False)
ohe_pets = ohe.fit_transform(pets)
pets_df = pd.DataFrame(ohe_pets, columns=ohe.get_feature_names_out(['Pets']))
pets_df
```

	Pets_dog	Pets_turtle
0	1.0	0.0
1	0.0	0.0
2	0.0	0.0
3	1.0	0.0
4	0.0	1.0
5	0.0	0.0
6	0.0	0.0
7	0.0	1.0
8	1.0	0.0
9	0.0	0.0

```
pets_df.corr()
```

	Pets_dog	Pets_turtle
Pets_dog	1.000000	-0.327327
Pets_turtle	-0.327327	1.000000

Day of Week Encoding

- https://mikulskibartosz.name/time-in-machine-learning

Get Dummies

```
# using pandas get_dummies
import pandas as pd

X_dummy = pd.get_dummies(X_train[['GarageType', 'GarageQual']], drop_first=True)
y_dummy = pd.get_dummies(X_test[['GarageType', 'GarageQual']], drop_first=True)
print(X_dummy.shape)
print(y_dummy.shape)
```

```
(1095, 10)
(365, 7)
```

```
# using one hot encoder
from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder(drop='first', sparse_output=False)

ohe_train = ohe.fit_transform(X_train[['GarageType', 'GarageQual']].dropna())
ohe_train = pd.DataFrame(ohe_train, columns=ohe.get_feature_names_out(['GarageType', 'Ga
print(ohe_train.shape)
ohe_train.head()
```

```
(1037, 9)
```

	GarageType_Attchd	GarageType_Basment	GarageType_BuiltIn	GarageType_CarPort	GarageTy
0	1.0	0.0	0.0	0.0	
1	1.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	
3	1.0	0.0	0.0	0.0	
4	1.0	0.0	0.0	0.0	

```
# ohe is already trained
ohe_test = ohe.transform(X_test[['GarageType', 'GarageQual']].dropna())
ohe_test = pd.DataFrame(ohe_test, columns=ohe_train.columns)
print(ohe_test.shape)
ohe_test.head()
```

```
(342, 9)
```

	GarageType_Attchd	GarageType_Basment	GarageType_BuiltIn	GarageType_CarPort	GarageTy
0	1.0	0.0	0.0	0.0	
1	1.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	
4	1.0	0.0	0.0	0.0	

One Hot Encoding Alternatives

For features with many labels

- https://medium.com/analytics-vidhya/stop-one-hot-encoding-your-categorical-variables-bbb0fba89809
- https://medium.com/swlh/stop-one-hot-encoding-your-categorical-features-avoid-curse-of-dimensionality-16743c32cea4
- https://towardsdatascience.com/all-about-categorical-variable-encoding-305f3361fd02 (frequency and mean encoding)

```
# review features with multiple labels
# identify features with more than 5 features
mult_labels = []
freq_feats = []

for val in X_train.columns.sort_values():
  if val in nulls:
    print(val, len(X_train[val].dropna().unique()))
    mult_labels.append(val)
    if len(X_train[val].dropna().unique()) > 4:
      freq_feats.append(val)

print(mult_labels)
print(freq_feats)
```

```
Alley 2
BsmtCond 4
BsmtExposure 4
BsmtFinType1 6
BsmtFinType2 6
BsmtQual 4
Electrical 4
Fence 4
FireplaceQu 5
GarageCond 5
GarageFinish 3
GarageQual 5
GarageType 7
```

```
GarageYrBlt 94
LotFrontage 105
MasVnrArea 278
MasVnrType 3
MiscFeature 4
PoolQC 3
['Alley', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2', 'BsmtQual', 'Electri
['BsmtFinType1', 'BsmtFinType2', 'FireplaceQu', 'GarageCond', 'GarageQual', 'GarageType',
```

```
# fill frequency
print(X_train['GarageType'].value_counts())
for feat in freq_feats:
    freq = X_train.groupby(feat).size()/len(X_train)
    X_train[feat] = X_train[feat].map(freq)
    freq = X_test.groupby(feat).size()/len(X_test)
    X_test[feat] = X_test[feat].map(freq)

print(X_train['GarageType'].value_counts())
print(X_train['GarageType'].value_counts(normalize=True))
```

```
GarageType
Attchd     651
Detchd     289
BuiltIn     69
Missing     58
Basment     15
CarPort      7
2Types       6
Name: count, dtype: int64
GarageType
0.594521    651
0.263927    289
0.063014     69
0.052968     58
0.013699     15
0.006393      7
0.005479      6
Name: count, dtype: int64
GarageType
0.594521    0.594521
0.263927    0.263927
0.063014    0.063014
0.052968    0.052968
0.013699    0.013699
0.006393    0.006393
0.005479    0.005479
Name: proportion, dtype: float64
```

Bi-Label Mapping

```
# get and train test split data
import seaborn as sns
from sklearn.model_selection import train_test_split

titanic = sns.load_dataset('titanic')
```

```
X_train, X_test, y_train, y_test = train_test_split(titanic.drop(['survived'], axis=1), t
X_train.head()
```

	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	de
298	1	male	NaN	0	0	30.5000	S	First	man	True	
884	3	male	25.00	0	0	7.0500	S	Third	man	True	N
247	2	female	24.00	0	2	14.5000	S	Second	woman	False	N
478	3	male	22.00	0	0	7.5208	S	Third	man	True	N
305	1	male	0.92	1	2	151.5500	S	First	child	False	

```
titanic.nunique()
```

	0
survived	2
pclass	3
sex	2
age	88
sibsp	7
parch	7
fare	248
embarked	3
class	3
who	3
adult_male	2
deck	7
embark_town	3
alive	2
alone	2

dtype: int64

```
# bi-label mapping
# whatever you do for X_train, do for X_test
X_train['sex'] = X_train['sex'].map({'male':0,'female':1})
X_test['sex'] = X_test['sex'].map({'male':0,'female':1})
X_train.head()
```

	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck
298	1	0	NaN	0	0	30.5000	S	First	man	True	C
884	3	0	25.00	0	0	7.0500	S	Third	man	True	NaN
247	2	1	24.00	0	2	14.5000	S	Second	woman	False	NaN
478	3	0	22.00	0	0	7.5208	S	Third	man	True	NaN
305	1	0	0.92	1	2	151.5500	S	First	child	False	C

Encoding Order

- Bilabel Mapping (2 labels)
- Frequency (5+ labels)
- One Hot Encoding (3 - 5 labels)

Outliers

- Treat outliers as missing data and impute accordingly
- Impose min max values
- Take care of altering meaningful data
- Outliers should be detected and removed from train only

https://www.projectpro.io/recipes/deal-with-outliers-in-python

- Drop
- Mark
- Rescale

```
# get data
from sklearn.datasets import fetch_california_housing

housing = fetch_california_housing()
print(housing.DESCR)
```

```
.. _california_housing_dataset:

California Housing dataset
--------------------------

**Data Set Characteristics:**

:Number of Instances: 20640

:Number of Attributes: 8 numeric, predictive attributes and the target

:Attribute Information:
    - MedInc        median income in block group
    - HouseAge      median house age in block group
```

```
    - AveRooms      average number of rooms per household
    - AveBedrms     average number of bedrooms per household
    - Population    block group population
    - AveOccup      average number of household members
    - Latitude      block group latitude
    - Longitude     block group longitude

:Missing Attribute Values: None

This dataset was obtained from the StatLib repository.
https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html

The target variable is the median house value for California districts,
expressed in hundreds of thousands of dollars ($100,000).

This dataset was derived from the 1990 U.S. census, using one row per census
block group. A block group is the smallest geographical unit for which the U.S.
Census Bureau publishes sample data (a block group typically has a population
of 600 to 3,000 people).

A household is a group of people residing within a home. Since the average
number of rooms and bedrooms in this dataset are provided per household, these
columns may take surprisingly large values for block groups with few households
and many empty houses, such as vacation resorts.

It can be downloaded/loaded using the
:func:`sklearn.datasets.fetch_california_housing` function.

.. rubric:: References

- Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions,
  Statistics and Probability Letters, 33 (1997) 291-297
```

```
# get keys
housing.keys()
```

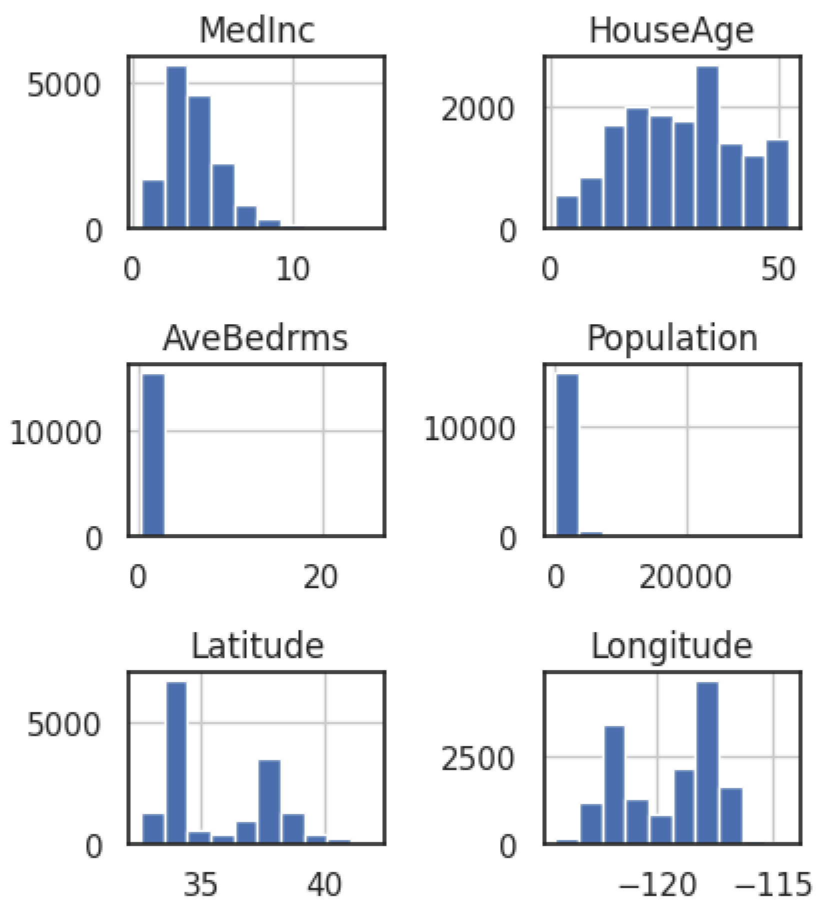
```
dict_keys(['data', 'target', 'frame', 'target_names', 'feature_names', 'DESCR'])
```

```
# create housing dataframe
import pandas as pd

housing_df = pd.DataFrame(housing.data, columns=housing.feature_names)
housing_df['MedHouseVal'] = housing.target
housing_df.head()
```

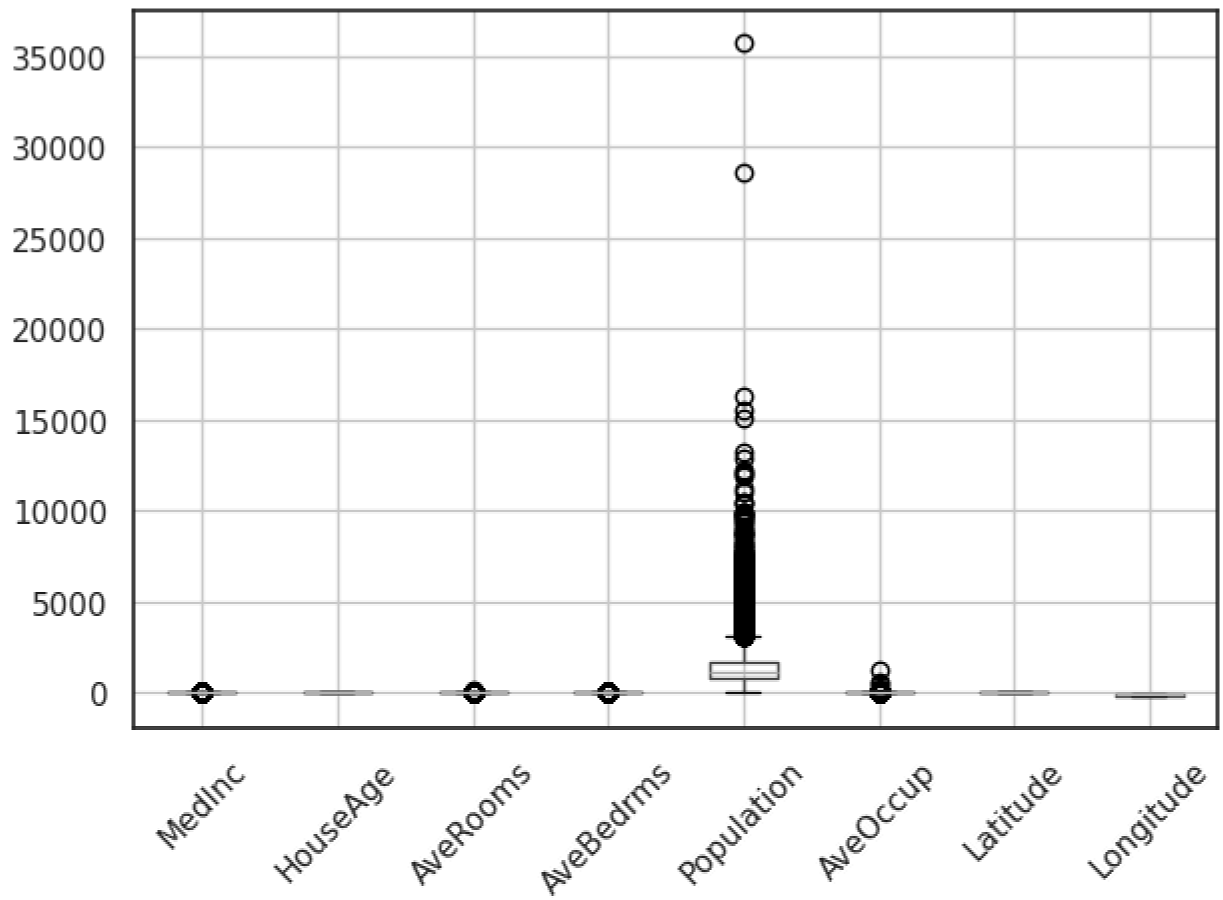
	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHo
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	

```
# train test split
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    housing_df.drop('MedHouseVal', axis=1),
    housing_df['MedHouseVal'],
    test_size=0.25,
    random_state=42)
```

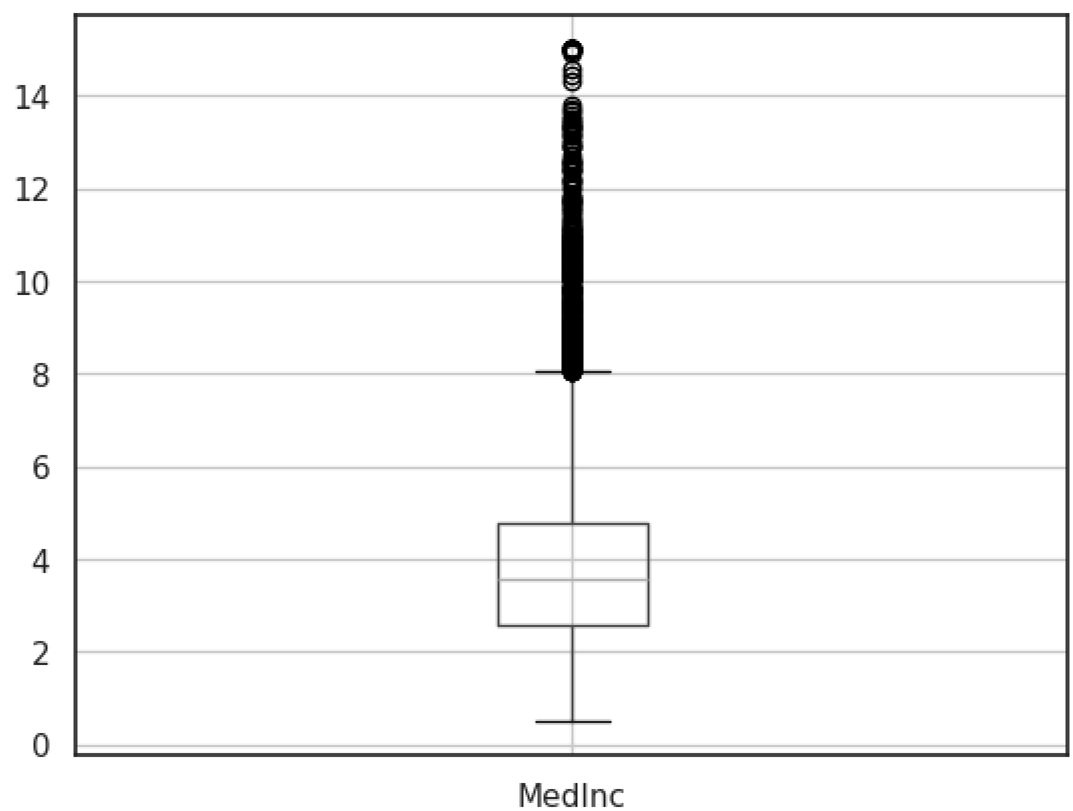
```
# histograms
import matplotlib.pyplot as plt

X_train.hist()
plt.tight_layout()
```



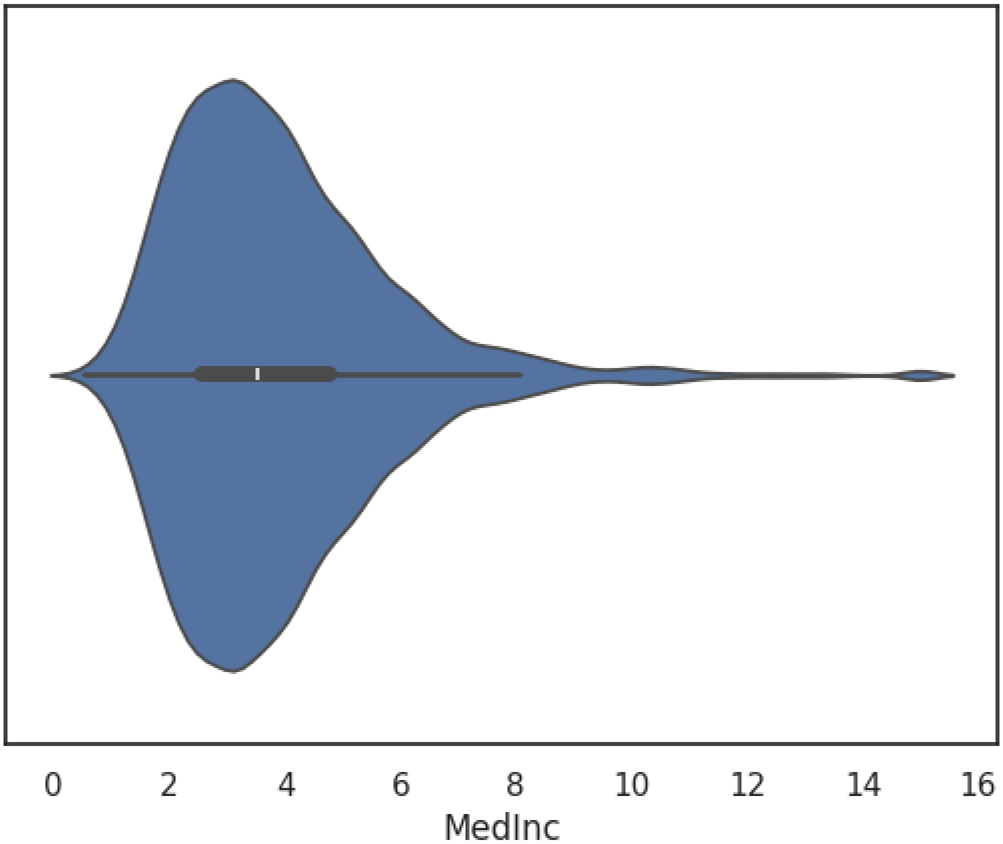

```
# boxplots
X_train.boxplot(rot=45)
plt.tight_layout()
```



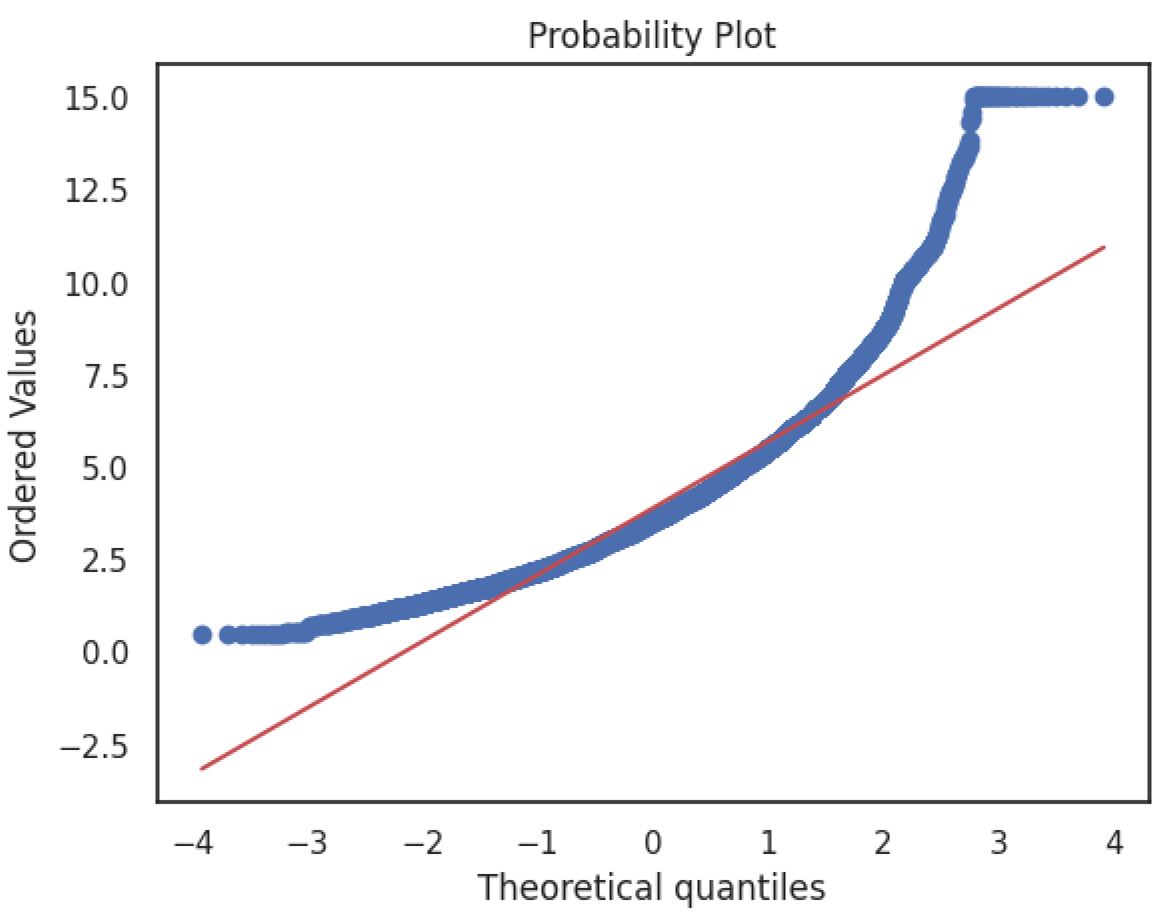

```
X_train.boxplot('MedInc');
```




```
import seaborn as sns

sns.violinplot(x=X_train['MedInc']);
```

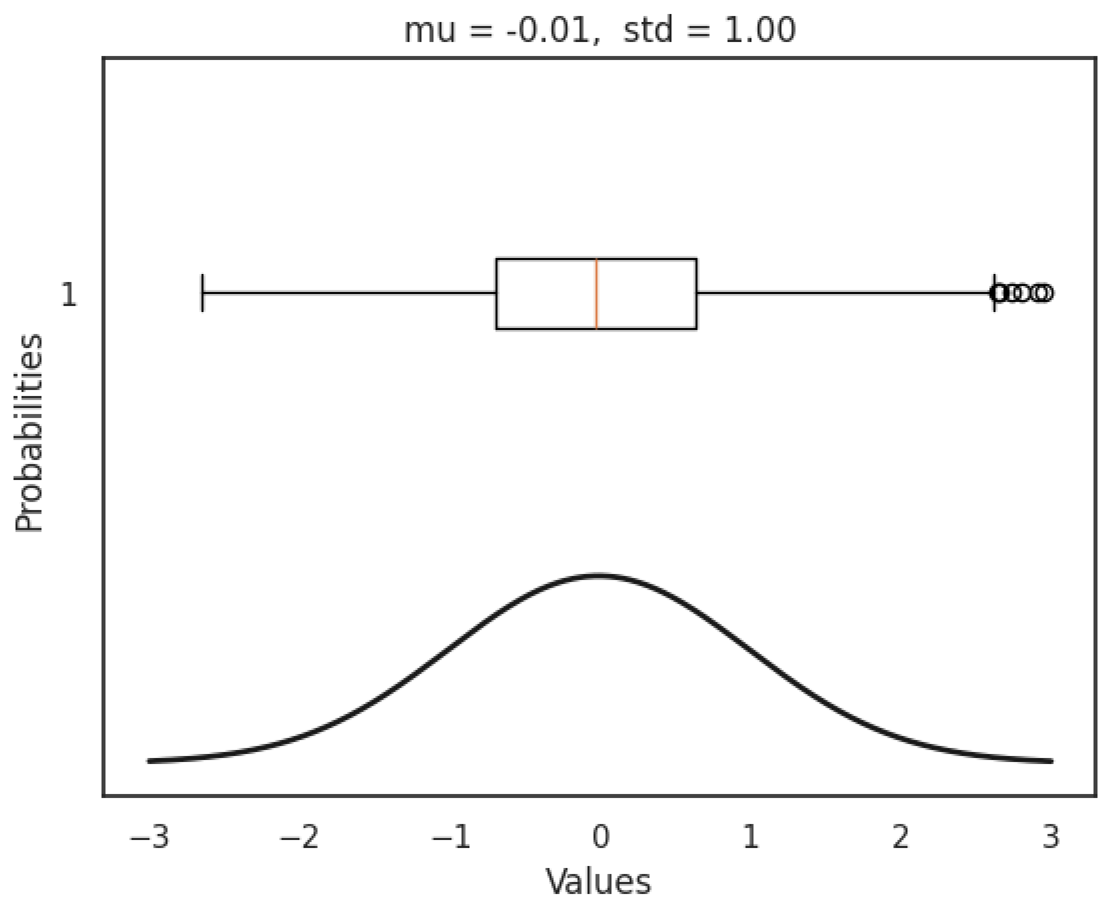



```
# prob plot
import scipy.stats as stats

stats.probplot(X_train['MedInc'], plot=plt);
```

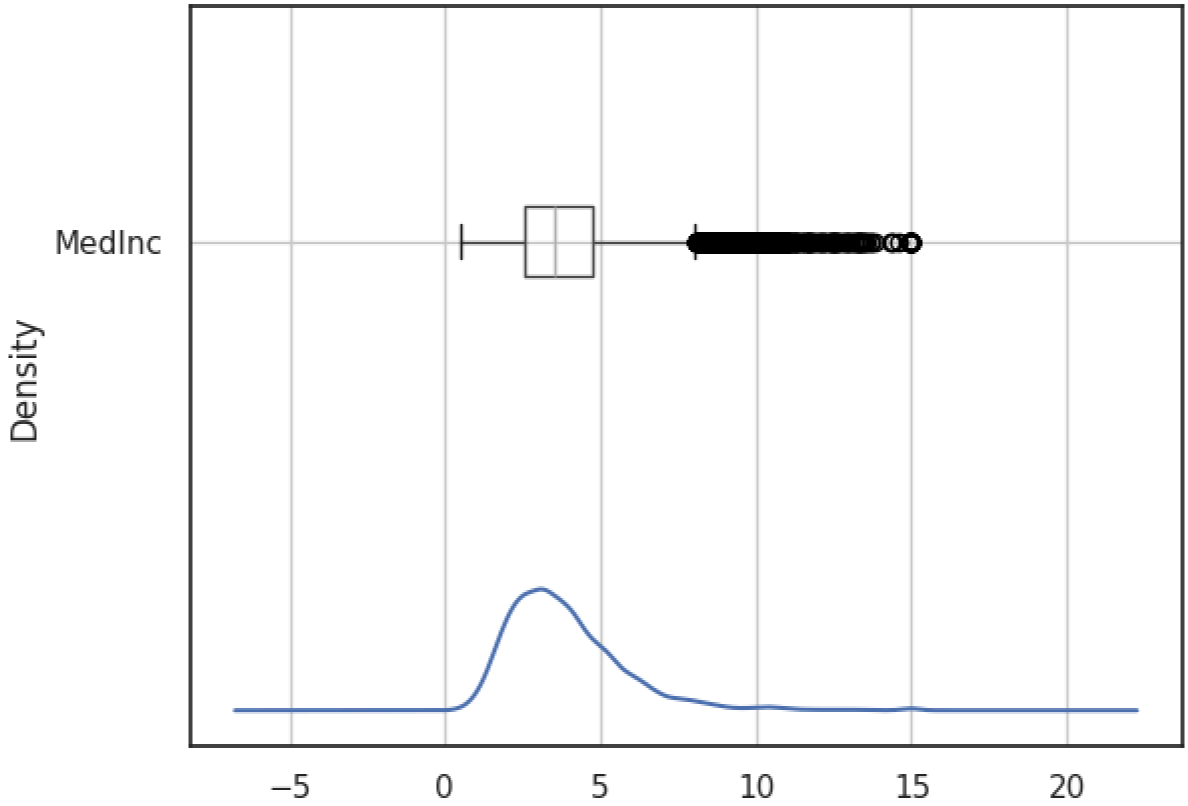


Boxplot and Normal Curve Review

```
# compare boxplot with normal distribution
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

data = stats.norm.rvs(size=1000)
mu, std = stats.norm.fit(data)

x = np.linspace(-3, 3, 1000)
p = stats.norm.pdf(x, mu, std)
plt.plot(x, p, 'k', linewidth=2)
plt.boxplot(data, vert=False)
plt.xlabel('Values')
plt.ylabel('Probabilities')
plt.title(f'mu = {mu:.2f},  std = {std:.2f}')

plt.show()
```




```
# compare with AveBedrms
X_train['MedInc'].plot.kde()
X_train.boxplot('MedInc', vert=False);
```




```
# find iqr and inner outer boundaries
q1 = X_train['MedInc'].quantile(0.25)
q3 = X_train['MedInc'].quantile(0.75)
iqr = q3 - q1

lower_inner_fence = q1 - (1.5 * iqr)
upper_inner_fence = q3 + (1.5 * iqr)
lower_outer_fence = q1 - (1.5 * iqr)
upper_outer_fence = q3 + (1.5 * iqr)

print(f'Q1: {q1:.2f} - Q3: {q3:.2f}')
```

```
Q1: 2.57 - Q3: 4.76
```

```
# print outliers by feature
for feat in X_train._get_numeric_data().columns[1:]:
    q1 = X_train[feat].quantile(0.25)
    q3 = X_train[feat].quantile(0.75)
    iqr = q3 - q1
    lower_fence = (q1 - 1.5 * iqr)
    upper_fence = (q3 + 1.5 * iqr)
    lower_count = X_train[feat][X_train[feat] < lower_fence].count()
    upper_count = X_train[feat][X_train[feat] > upper_fence].count()
    print(f'{feat} outliers = {lower_count + upper_count}: lower_fence: {lower_fence}, u
```

```
HouseAge outliers = 0: lower_fence: -10.5, upper_fence: 65.5, lower_count: 0, upper_count:
AveRooms outliers = 393: lower_fence: 2.037789153370117, upper_fence: 8.470351411049805, l
AveBedrms outliers = 1084: lower_fence: 0.866268363721791, upper_fence: 1.24046847804609
Population outliers = 887: lower_fence: -618.625, upper_fence: 3134.375, lower_count: 0, u
AveOccup outliers = 544: lower_fence: 1.1554832731192153, upper_fence: 4.554741594313875,
Latitude outliers = 0: lower_fence: 28.269999999999996, upper_fence: 43.39, lower_count: 0
```

```
Longitude outliers = 0: lower_fence: -127.48499999999999, upper_fence: -112.3250000000002
```

```
# check our numbers
X_train['MedInc'].describe()
```

	MedInc
count	15480.000000
mean	3.878314
std	1.903788
min	0.499900
25%	2.566925
50%	3.543900
75%	4.762500
max	15.000100

dtype: float64

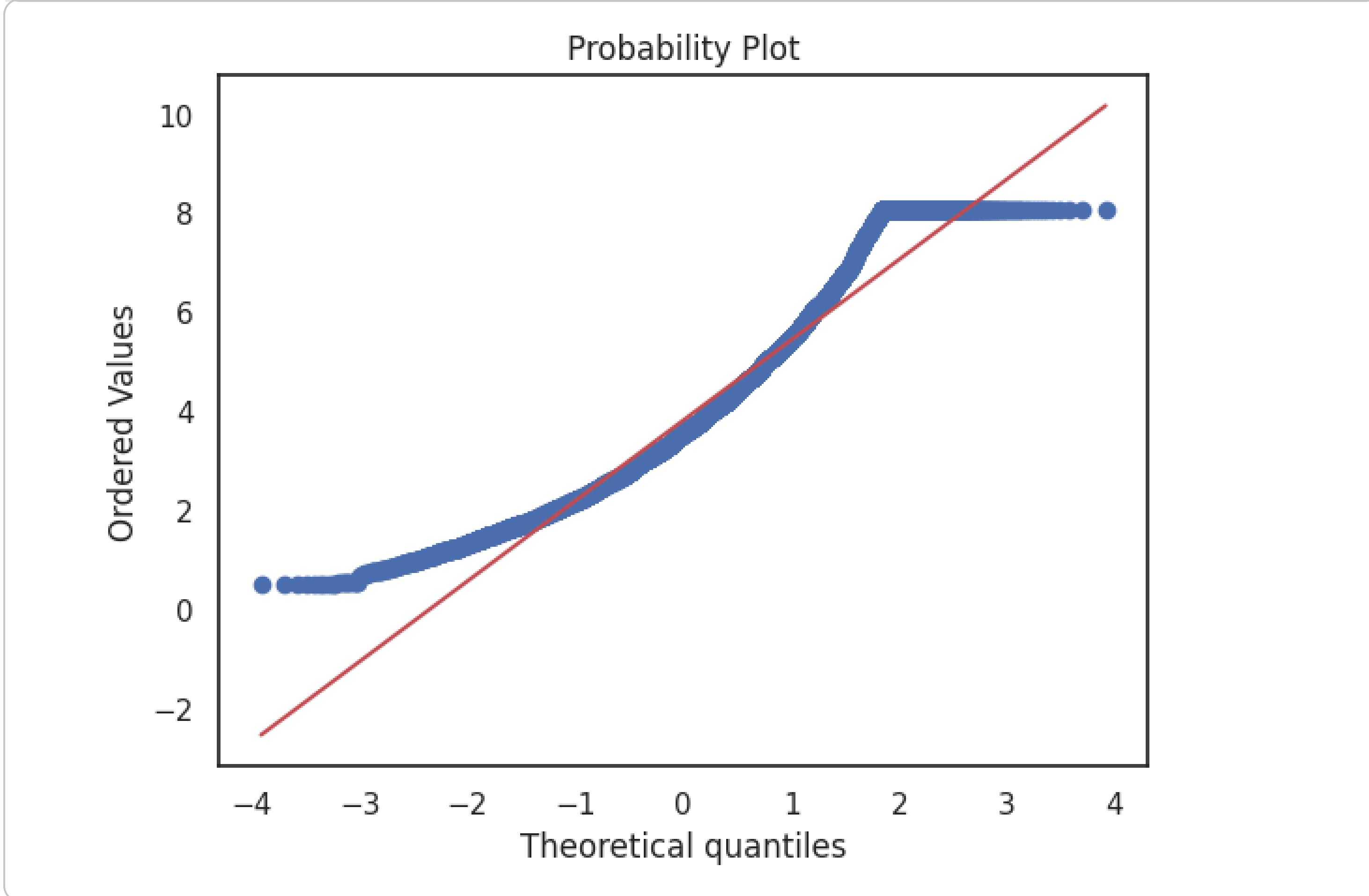
Outlier Trimming

```
# flag the rows with outliers
import numpy as np

outliers = np.where(X_train['MedInc'] < lower_inner_fence, True,
                   np.where(X_train['MedInc'] > upper_inner_fence, True, False))

X_train_trimmed = X_train.loc[outliers]
print(X_train.shape, X_train_trimmed.shape)
```

```
(15480, 8) (505, 8)
```

IQR Proximity Rule Capping

```
# cap outliers
import scipy.stats as stats

X_train['capped'] = np.where(X_train['MedInc'] < lower_inner_fence, lower_inner_fence,
                   np.where(X_train['MedInc'] > upper_inner_fence, upper_inner_fence, X_t

stats.probplot(X_train['capped'], plot=plt);
```



Scaling

- Coefficients of linear models are influenced by the scale of the feature
- Features with larger scales dominate smaller scales
- Some algorithms, like PCA, require features to be centered at 0

https://www.atoti.io/articles/when-to-perform-a-feature-scaling/

- from sklearn.preprocessing import MinMaxScaler
- from sklearn.preprocessing import minmax_scale
- from sklearn.preprocessing import MaxAbsScaler
- from sklearn.preprocessing import StandardScaler
- from sklearn.preprocessing import RobustScaler
- from sklearn.preprocessing import Normalizer
- from sklearn.preprocessing import QuantileTransformer
- from sklearn.preprocessing import PowerTransformer

https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html

Standardization

$z = \frac{(x - \bar{x})}{\sigma}$

- Centers data around 0
- Scales the std to 1
- Preserves original shape
- Preserves outliers

```
X_train.drop('capped', axis=1, inplace=True)
X_train.describe()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup
count	15480.000000	15480.000000	15480.000000	15480.000000	15480.000000	15480.000000
mean	3.878314	28.595995	5.435598	1.096881	1427.497287	3.106660
std	1.903788	12.611330	2.421650	0.438804	1142.930862	11.955834
min	0.499900	1.000000	0.888889	0.333333	3.000000	0.692308
25%	2.566925	18.000000	4.450000	1.006593	788.750000	2.430205
50%	3.543900	29.000000	5.232331	1.049346	1167.000000	2.817672
75%	4.762500	37.000000	6.058141	1.100143	1727.000000	3.280020
max	15.000100	52.000000	141.909091	25.636364	35682.000000	1243.333333

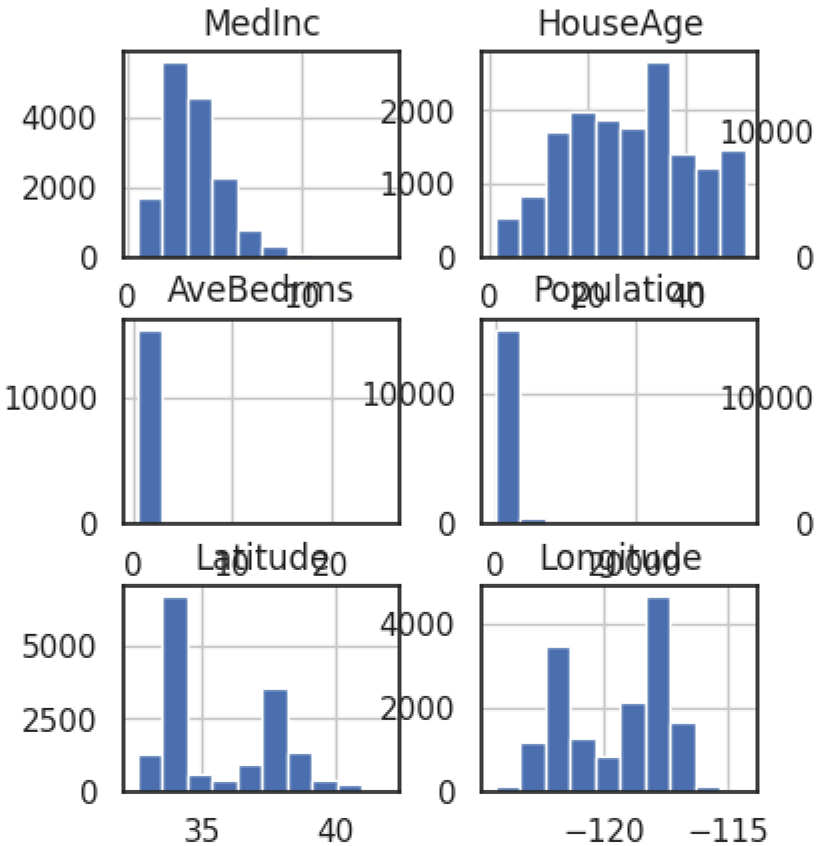
Characteristics of X_train

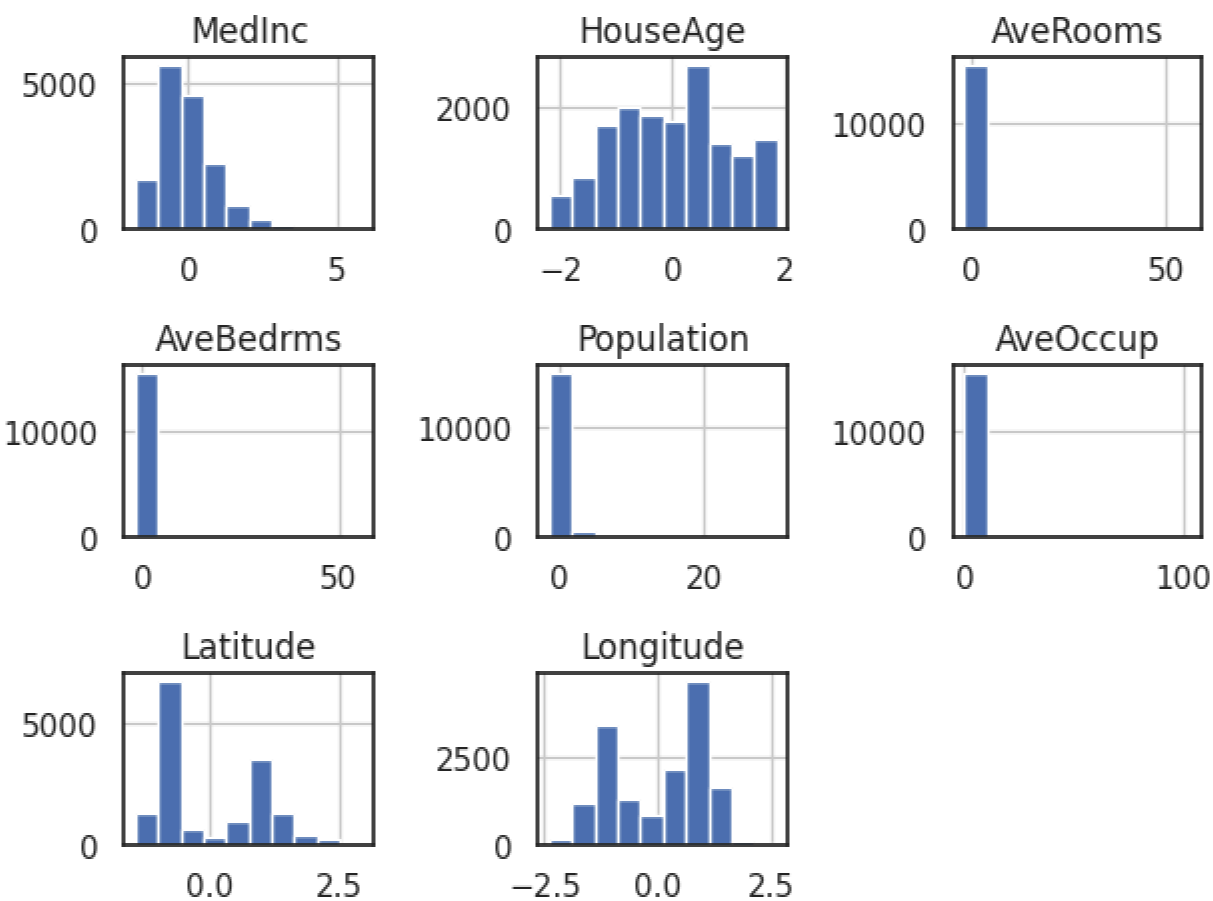
- Mean values not centered around 0
- Std not 1
- Features have various magnitudes

```
# standardize features
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X_train)
standardized_X = scaler.transform(X_train)
standardized_yX = scaler.transform(X_test) # we use the scaler that was trained on the X_
X_train_standardized = pd.DataFrame(standardized_X, columns=X_train.columns)
X_train_standardized.describe()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup
count	1.548000e+04	1.548000e+04	1.548000e+04	1.548000e+04	1.548000e+04	1.548000e+04
mean	2.074711e-16	-1.232434e-16	-1.620294e-16	7.435912e-17	-8.996536e-17	1.055716e-17
std	1.000032e+00	1.000032e+00	1.000032e+00	1.000032e+00	1.000032e+00	1.000032e+00
min	-1.774632e+00	-2.188261e+00	-1.877586e+00	-1.740123e+00	-1.246395e+00	-2.019458e-01
25%	-6.888537e-01	-8.402236e-01	-4.070076e-01	-2.057655e-01	-5.588859e-01	-5.658128e-02
50%	-1.756629e-01	3.203613e-02	-8.394015e-02	-1.083316e-01	-2.279278e-01	-2.417208e-02

```
# compare histograms
X_train.hist()
X_train_standardized.hist()
plt.tight_layout();
```





MinMaxScaling (Normalization)

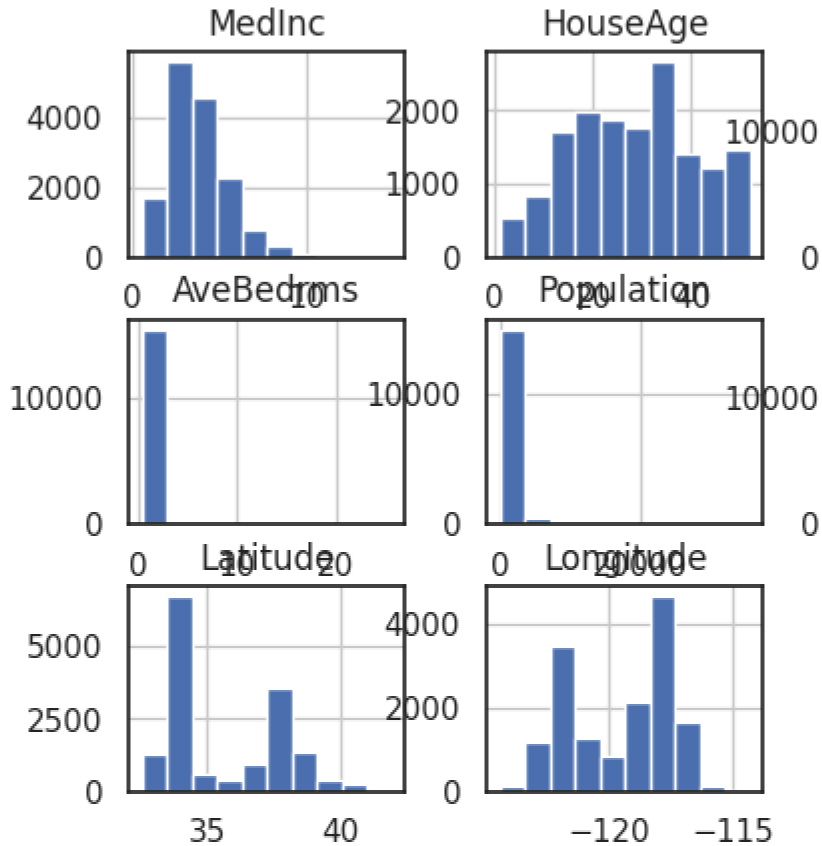
- Does not center the mean around 0
- Std (variance) differ

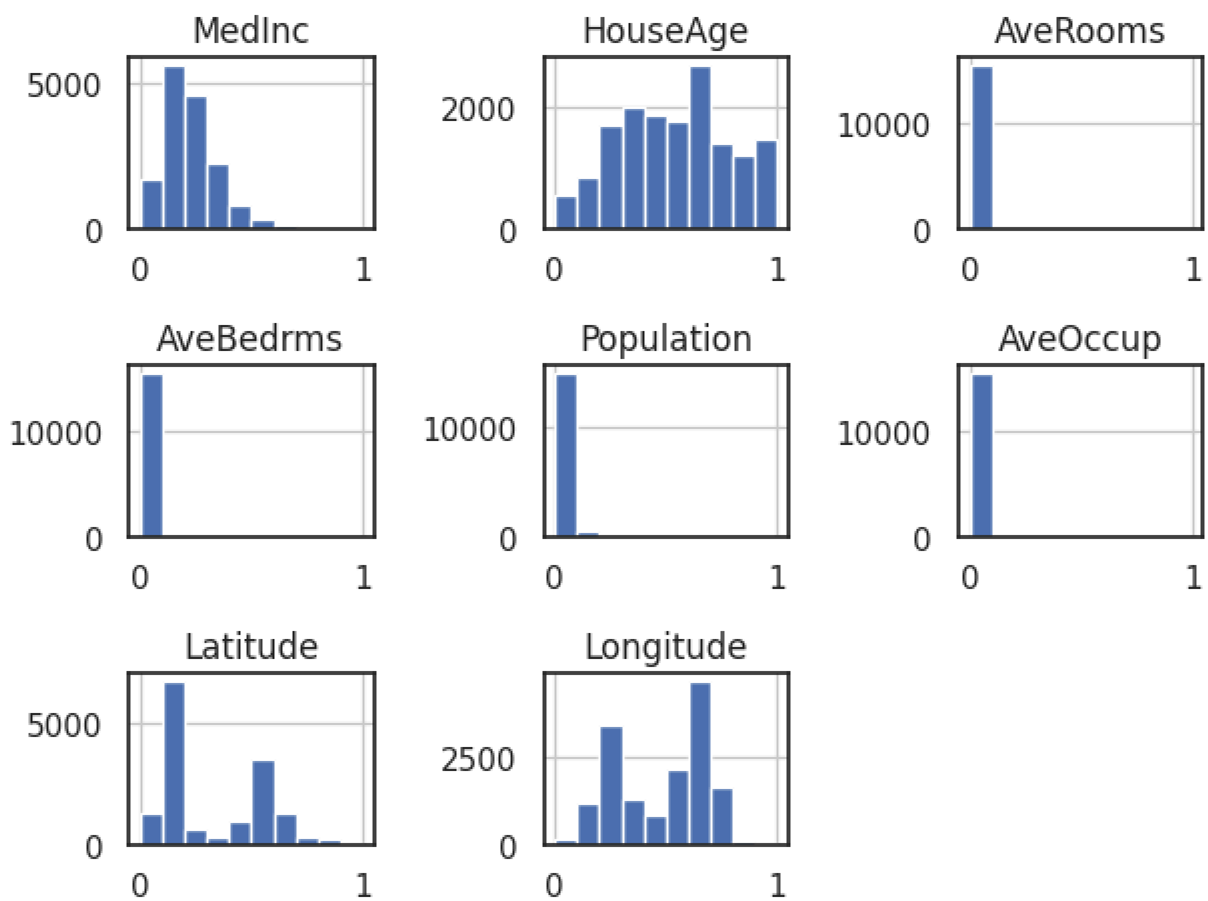
- May not preserve original shape
- 0 to 1 range
- Sensitive to outliers

```
# minmax scaling
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
scaler.fit(X_train)
minmax = scaler.transform(X_train)
# don't forget X_test
X_train_minmax = pd.DataFrame(minmax, columns=X_train.columns)
X_train_minmax.describe()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	
count	15480.000000	15480.000000	15480.000000	15480.000000	15480.000000	15480.000000	1
mean	0.232991	0.541098	0.032242	0.030176	0.039925	0.001943	
std	0.131294	0.247281	0.017172	0.017342	0.032034	0.009621	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.142551	0.333333	0.025252	0.026608	0.022023	0.001399	
50%	0.209928	0.549020	0.030800	0.028298	0.032624	0.001710	
75%	0.293968	0.705882	0.036656	0.030305	0.048320	0.002082	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

```
# visual comparison
X_train.hist()
X_train_minmax.hist()
plt.tight_layout();
```





Mean Normalization

- Centers the mean at 0
- Std (variance) will differ

- May alter original distribution
- -1 to 1 range
- Preserves outliers

```
# find the means
means = X_train.mean(axis=0)
means
```

	0
MedInc	3.878314
HouseAge	28.595995
AveRooms	5.435598
AveBedrms	1.096881
Population	1427.497287
AveOccup	3.106660
Latitude	35.646720
Longitude	-119.583736

dtype: float64

```
# find the ranges
ranges = X_train.max(axis=0) - X_train.min(axis=0)
ranges
```

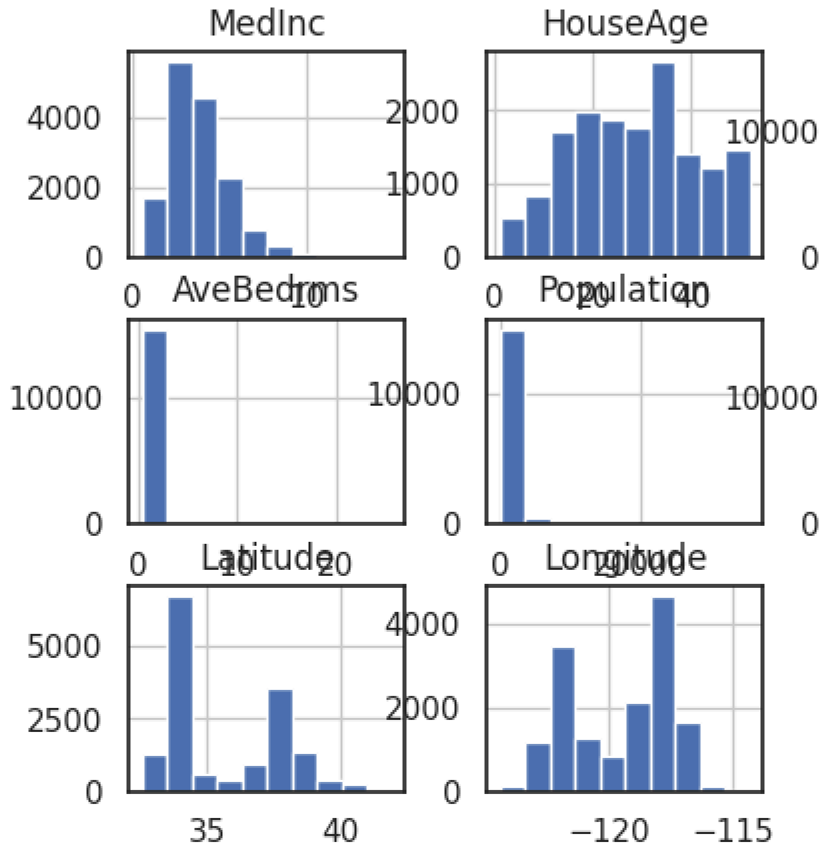
	0
MedInc	14.500200
HouseAge	51.000000
AveRooms	141.020202
AveBedrms	25.303030
Population	35679.000000
AveOccup	1242.641026
Latitude	9.400000
Longitude	10.040000

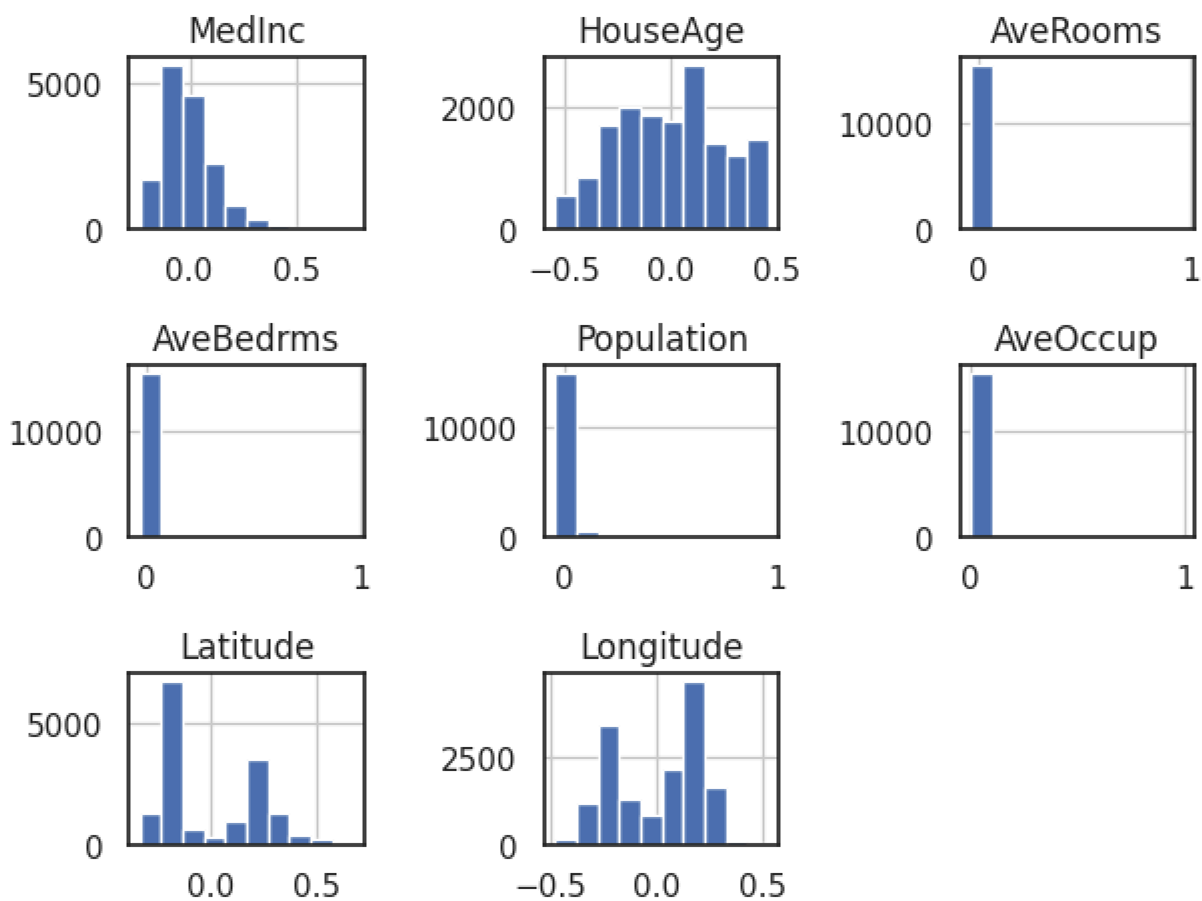
dtype: float64

```
# mean scale the data
X_train_meanscale = (X_train - means) / ranges
# don't forget X_test
X_train_meanscale.describe()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup
count	1.548000e+04	1.548000e+04	1.548000e+04	1.548000e+04	1.548000e+04	1.548000e+04
mean	2.673715e-17	-3.287637e-17	-2.782730e-18	1.290957e-18	-2.868793e-18	9.682178e-20
std	1.312939e-01	2.472810e-01	1.717237e-02	1.734195e-02	3.203371e-02	9.621310e-03
min	-2.329909e-01	-5.410979e-01	-3.224155e-02	-3.017614e-02	-3.992537e-02	-1.942920e-03
25%	-9.043938e-02	-2.077646e-01	-6.989058e-03	-3.568259e-03	-1.790261e-02	-5.443684e-04
50%	-2.306272e-02	7.921670e-03	-1.441404e-03	-1.878620e-03	-7.301138e-03	-2.325595e-04

```
# visual comparison
X_train.hist()
X_train_meanscale.hist()
plt.tight_layout();
```





RobustScaler

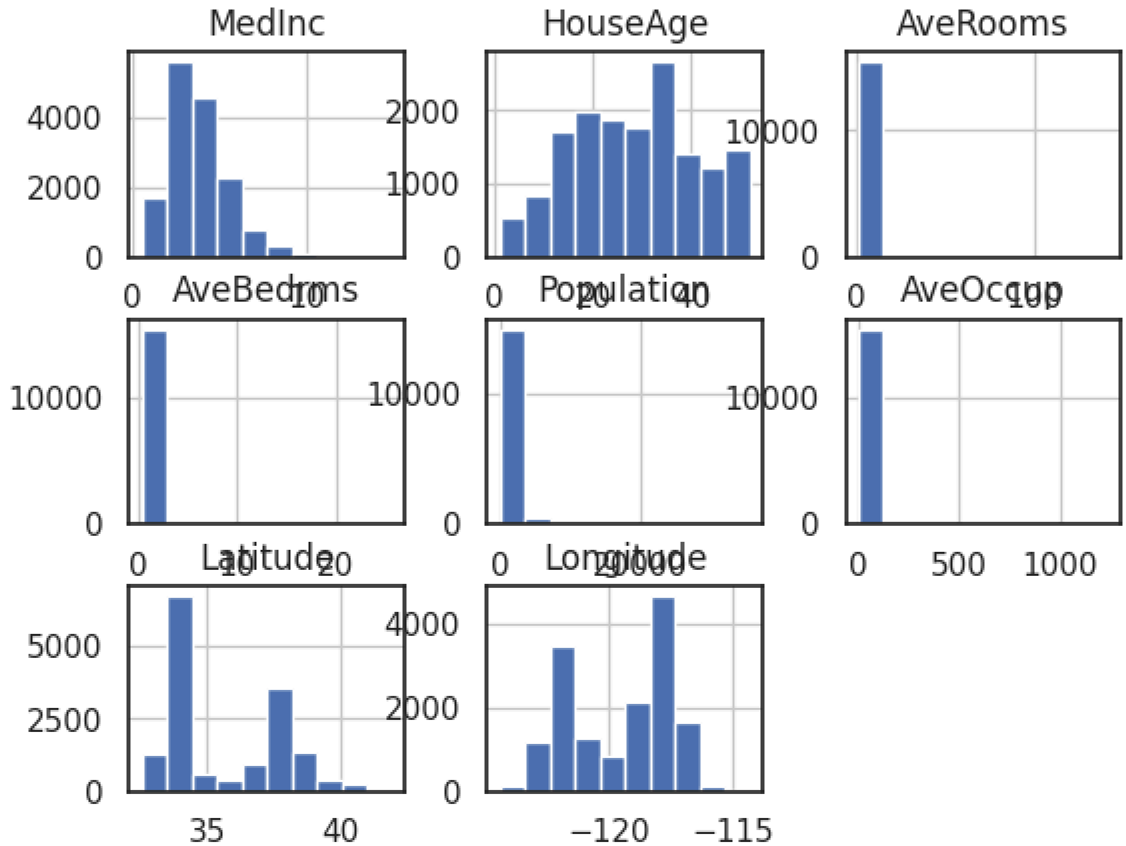
- Replaces median with iqr
- Variance varies

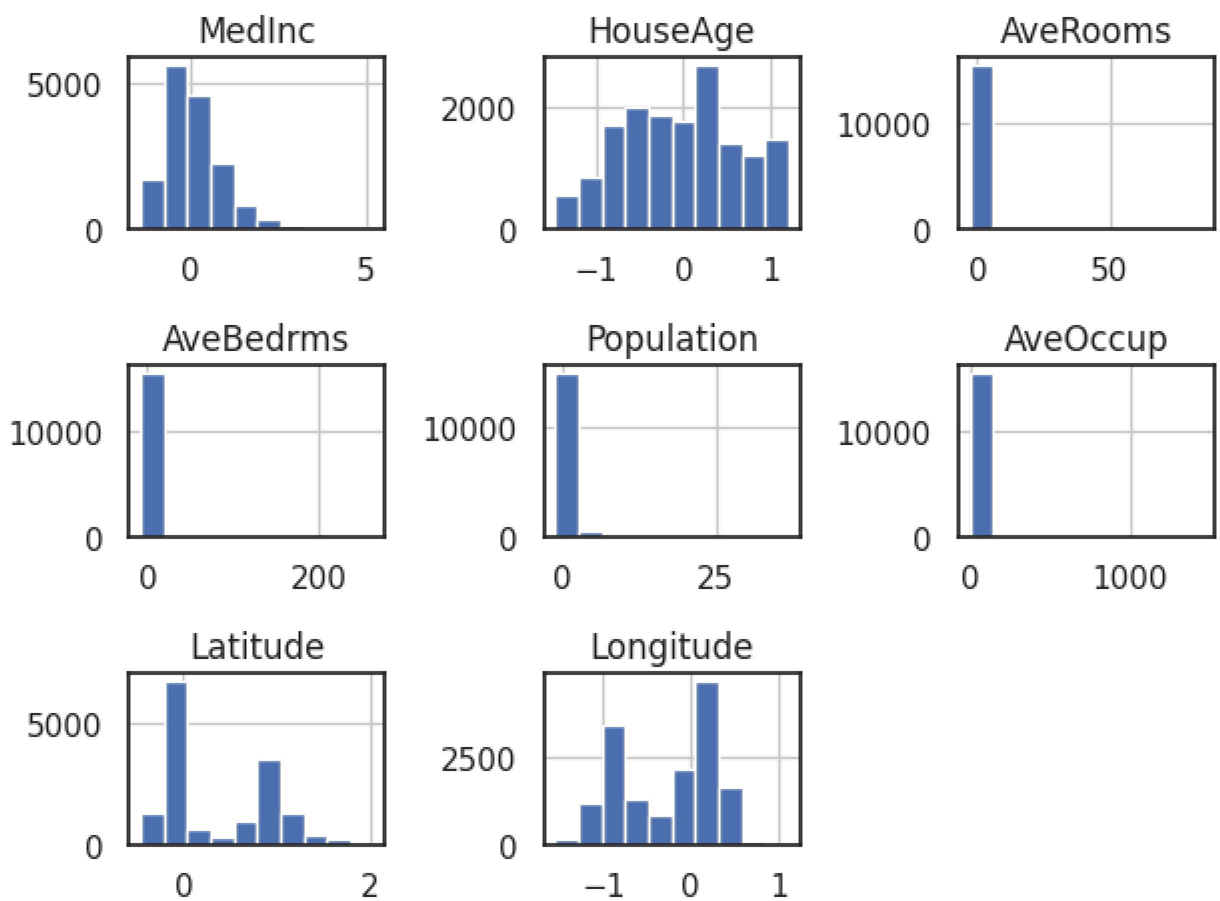
- May not preserve distribution
- Min max varies
- Robust to outliers https://www.statisticshowto.com/robust-statistics/

```
# robust scaler
from sklearn.preprocessing import RobustScaler

scaler = RobustScaler()
scaler.fit(X_train)
robust = scaler.transform(X_train)
# don't forget X_test
X_train_robust = pd.DataFrame(robust, columns=X_train.columns)
X_train_robust.describe()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup
count	15480.000000	15480.000000	1.548000e+04	15480.000000	15480.000000	1.548000e+04
mean	0.152313	-0.021263	1.263989e-01	0.508122	0.277642	3.400601e-01
std	0.867102	0.663754	1.505870e+00	4.690579	1.218152	1.406876e+01
min	-1.386425	-1.473684	-2.700910e+00	-7.653798	-1.240608	-2.500974e+00
25%	-0.444975	-0.578947	-4.864819e-01	-0.457006	-0.403144	-4.559425e-01
50%	0.000000	0.000000	-2.761531e-16	0.000000	0.000000	-2.612859e-16
75%	0.555025	0.421053	5.135181e-01	0.542994	0.596856	5.440575e-01
max	5.217859	1.210526	8.499056e+01	262.822125	36.786571	1.459749e+03

```
# visual comparison
X_train.hist()
X_train_robust.hist()
plt.tight_layout();
```





PowerTransformers

```
# PowerTransformer scaler for outliers
from sklearn.preprocessing import PowerTransformer
```

```
feat_scales = []
```