



Putting it all
together

Typical machine learning process



Data



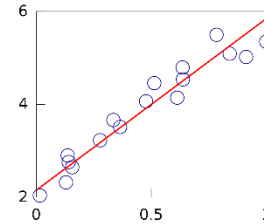
Data Analysis



Feature Engineering



Variable selection



Machine Learning Model building

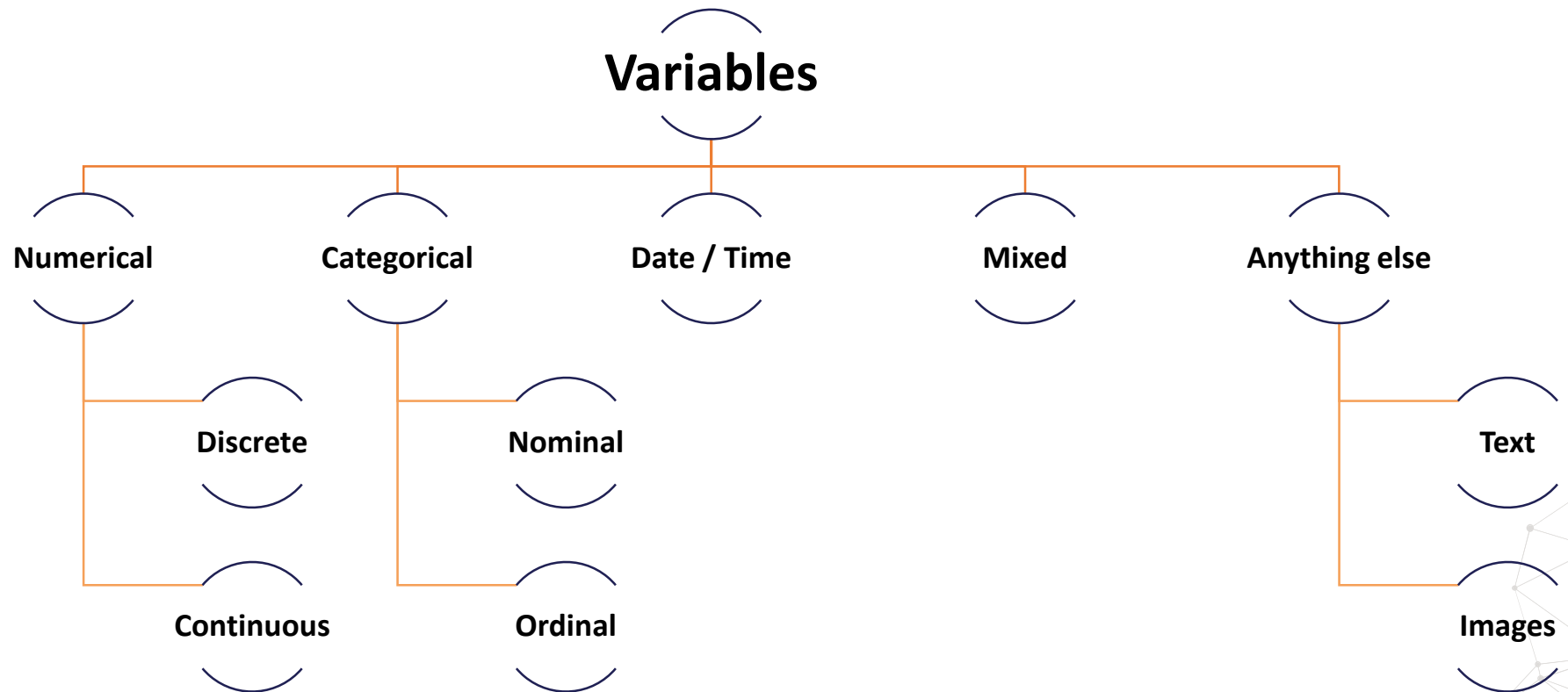


Deployment

Data Analysis → Variable Types



Data Analysis



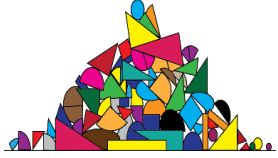
Data Analysis → Variable Characteristics



Data Analysis



Feature engineering steps



Feature
Engineering

- Feature Creation:
 - Extracting Features from Dates
 - Extracting Features from Mixed Variables
 - Missing Data Imputation
 - Categorical Variable Encoding
 - Numerical Variable Transformation
 - Discretisation
 - Outlier Handling
 - Feature Scaling
 - And more...

Jupyter notebook



- Hard to understand
- Hard to reproduce
- Hard to score new data
- Hard to deploy

Contents [] to a t

- 1 Regression
- 2 In this demo
- 3 House Price dataset
 - 3.1 Load Dataset
 - 3.2 Types of variables (ps)
 - 3.2.1 Find categorical vs
 - 3.2.2 Find temporal vars
 - 3.2.3 Find discrete vars
 - 3.3 Types of problems with
 - 3.3.1 Missing values
 - 3.3.2 Outliers and skew
 - 3.3.3 Monotonicity between
 - 3.3.4 Number of obs
 - 3.3.5 Separate train and test
 - 3.3.6 Temporal variables (ps)
 - 3.3.7 Missing data imputation
 - 3.3.8 Continuous variable
 - 4 Putting it all together

There are 18 numerical and continuous variables.

Perfect! Now we have inspected and have a view of the different types of variables that we have in the house price dataset. Let's move on to understand the types of problems that these variables have.

3.3 Types of problems within the variables (section 3)

3.3.1 Missing values

```
In [12]: # Let's output variables with NA and the percentage of NA
for var in data.columns:
    print(var, data[var].isnull().sum(), data[var].isnull().sum()/len(data))
```

LotFrontage 0.173972682739726
Alley 0.937612128701212
MasVnrType 0.884784238478423
MasVnrArea 0.884784238478423
BsmtQual 0.8253446075344608
BsmtCond 0.8253446075344608
BsmtExposure 0.8253446075344608
BsmtFinType1 0.8253446075344608
BsmtFinType2 0.8253446075344608
Electrical 0.888888888888889
FireplaceQu 0.472602739726027
GarageType 0.854794238479423
GarageYrBlt 0.854794238479423
GarageCars 0.854794238479423
GaragePkg 0.854794238479423
PoolQC 0.952054794238479
Fence 0.887534246575342
MiscFeature 0.363636363636364

3.3.2 Outliers and distributions

```
In [13]: # Let's make boxplots to visualize outliers in the continuous variables
# and Histograms to get an idea of the distribution
for var in numerical:
    plt.figure(figsize=(6,4))
    plt.subplot(1, 2, 1)
    fig = data.boxplot(column=var)
    fig.set_title(var)
    fig.set_ylabel(var)
    plt.subplot(1, 2, 2)
    fig = data.hist(column=var)
    fig.set_title('Histogram of %s' % var)
    fig.set_ylabel('Number of houses')
    plt.show()
```

The majority of the continuous variables seem to contain outliers. In addition, the majority of the variables are not normally distributed. As we are planning to build linear regression, we need to tackle these to improve the model performance. To tackle the 2 aspects together, I will do discretisation. I will follow discretisation with encoding of the intervals following the target mean, as we do in the Discretisation plus encoding lecture in section 5.

3.3.3 Outliers in discrete variables

Now let's identify outliers in the discrete variables. I will call outliers those values that are present in less than 5 % of the houses. This is exactly the same as finding rare labels in categorical variables. Discrete variables can be pre-processed / engineered as if they were categorical. Keep this in mind.

```
In [14]: # Let's identify outliers in the discrete variables
for var in categorical:
    counts = data[var].value_counts()
    p = counts / len(data)
    plt.figure(figsize=(6,4))
    plt.subplot(1, 2, 1)
    fig = data.boxplot(column=var)
    fig.set_title(var)
    fig.set_ylabel(var)
    plt.subplot(1, 2, 2)
    fig = p.plot(kind='bar')
    fig.set_title('Percentage of observations per label')
    fig.set_ylabel('Percentage of observations per label')
    plt.show()
```

Most of the discrete variables show values that are shared by a tiny proportion of houses in the dataset.

3.4 Monotonicity between discrete variables and target values

```
In [15]: # Let's plot the median sale price per value of the discrete variable
for var in categorical:
    data.groupby(var)['SalePrice'].median().plot()
    plt.title('Median house Price per label')
    plt.show()
```

Pipeline



- Clear
- Concise
- Reproducible
- Able to score new data
- Easy to deploy



Pipeline

```
price_pipe = pipe([
    # add a binary variable to indicate missing information for the 2 variables below
    ('continuous_var_imputer', msi.AddNaNBinaryImputer(variables = ['LotFrontage', 'GarageYrBlt'])),

    # replace NA by the median in the 3 variables below, they are numerical
    ('continuous_var_median_imputer', msi.MeanMedianImputer(imputation_method='median', variables = ['LotFrontage', 'GarageYrBlt', 'MasVnrArea'])),

    # replace NA by adding the label "Missing" in categorical variables (transformer will skip those variables where there is no NA)
    ('categorical_imputer', msi.CategoricalVariableImputer(variables = categorical)),

    # there were a few variables in the submission dataset that showed NA, but these variables did not show NA in the train set.
    # to handle those, I will add an additional step here
    ('additional_median_imputer', msi.MeanMedianImputer(imputation_method='median', variables = numerical)),

    # discretise numerical variables using trees
    ('numerical_tree_discretiser', dsc.DecisionTreeDiscretiser(cv = 3, scoring='neg_mean_squared_error', variables = numerical, regression=True)),

    # remove rare labels in categorical and discrete variables
    ('rare_label_encoder', ce.RareLabelCategoricalEncoder(tol = 0.03, n_categories=1, variables = categorical+discrete)),

    # encode categorical variables using the target mean
    ('categorical_encoder', ce.MeanCategoricalEncoder(variables = categorical+discrete))
])
```


Pipeline

train the pipeline

```
price_pipe.fit(X_train, y_train)
```

score data

```
price_pipe.transform(X_train)
```

```
price_pipe.transform(X_test)
```

```
price_pipe.transform(live_data)
```

THANK YOU

www.trainindata.com