AASD 4000
Machine Learning - I

Applied AI Solutions Developer Program

Module 07
Data Preprocessing

Vejey Gandyer

Agenda

- Data Preprocessing
- Importance of Data Preprocessing
- Feature Scaling
- Feature Standardization
- Feature Normalization
- Label Encoding
- One Hot Encoding
- Imputation
- Task: Data Preprocessor

Data Preprocessing

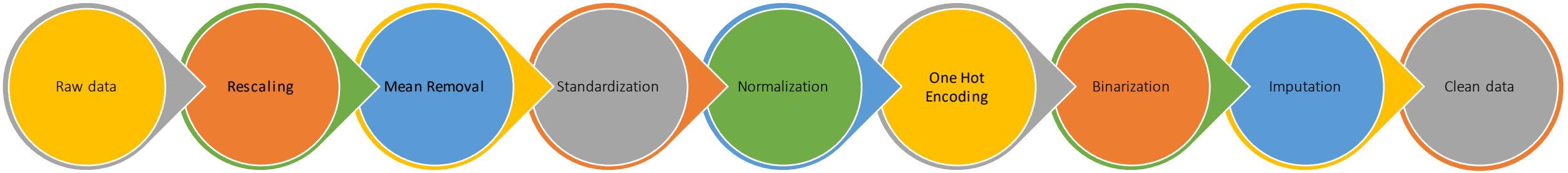
What is it?

What is Data Preprocessing ?

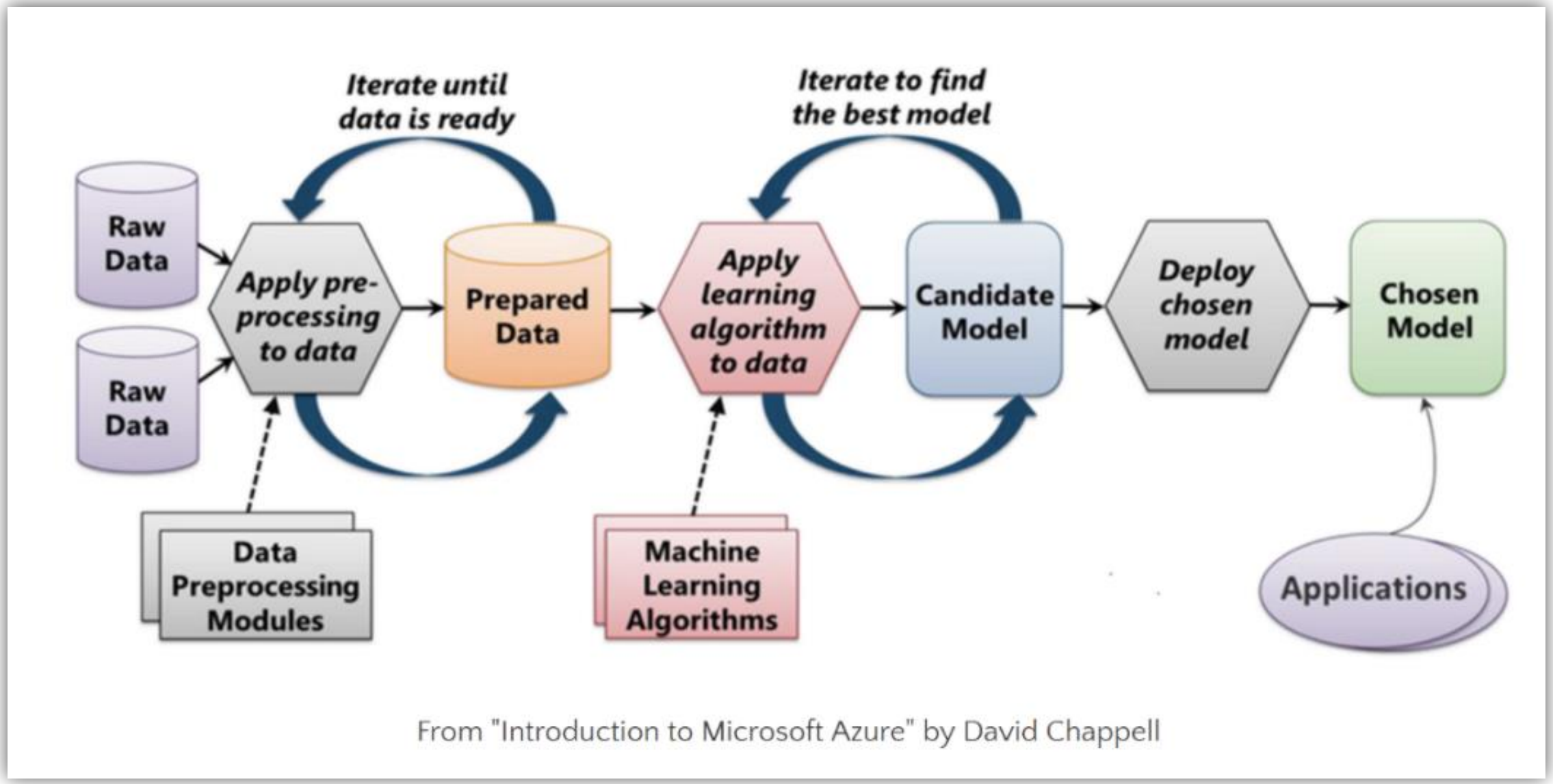
Data Preprocessing is a technique that is used to convert the raw data into a clean data set.

Real-world data: Incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors

What is Data Preprocessing ?



Machine Learning Process



From "Introduction to Microsoft Azure" by David Chappell

Importance of Data Preprocessing

Why it is needed?

Importance of Data Preprocessing

Most of the time, we don’t get quality data.

Data

- Missing
- Noisy
- Inconsistent values

These can potentially reduce the accuracy of the model

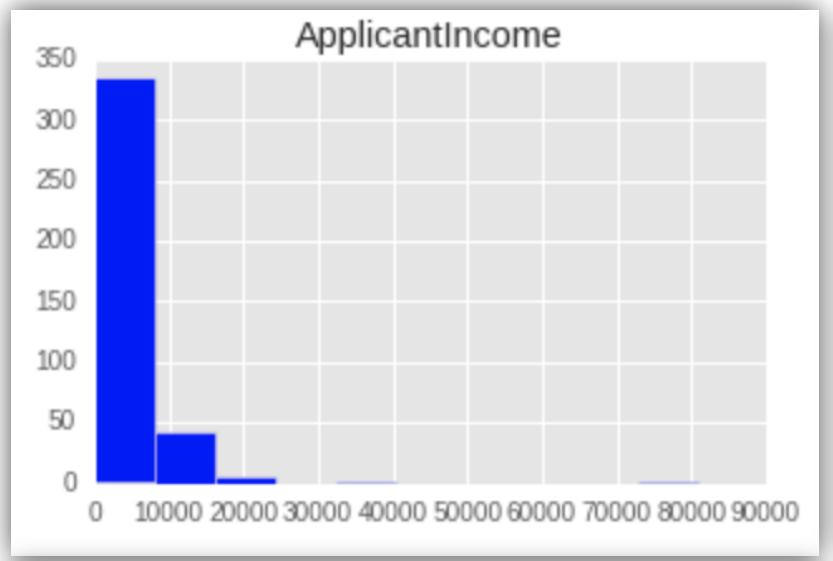
Dataset – Loan Prediction dataset

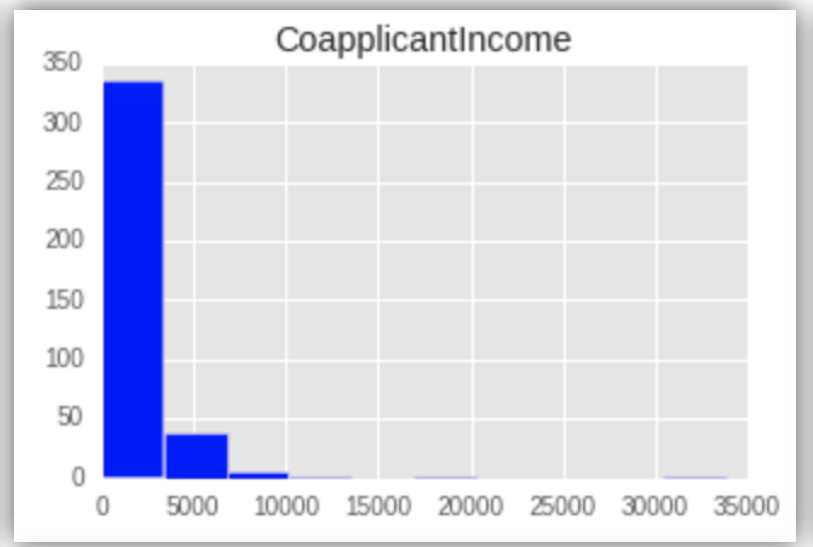
	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Credit_History	Property_Area
15	LP001032	Male	No	0	Graduate	No	1.0	Urban
248	LP001824	Male	Yes	1	Graduate	No	1.0	Semiurban
590	LP002928	Male	Yes	0	Graduate	No	1.0	Semiurban
246	LP001814	Male	Yes	2	Graduate	No	1.0	Urban
388	LP002244	Male	Yes	0	Graduate	No	1.0	Urban

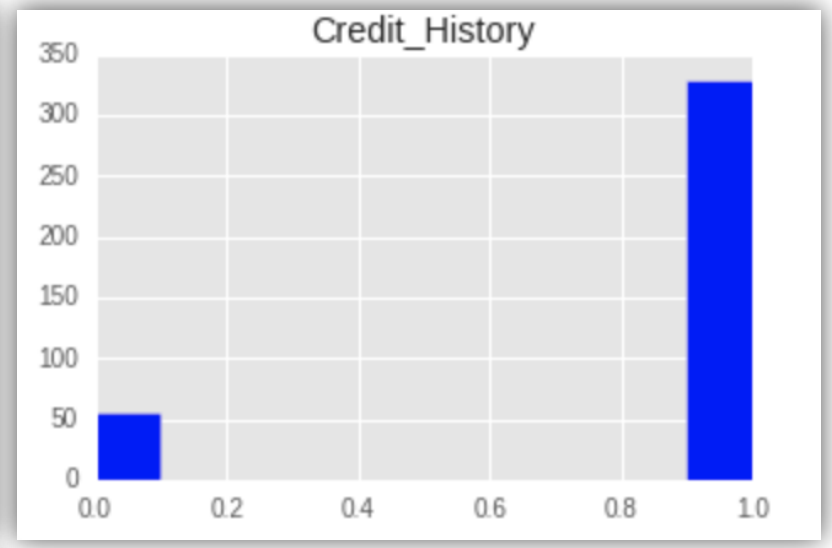
	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
15	4950	0.0	125.0	360.0
248	2882	1843.0	123.0	480.0
590	3000	3416.0	56.0	180.0
246	9703	0.0	112.0	360.0
388	2333	2417.0	136.0	360.0

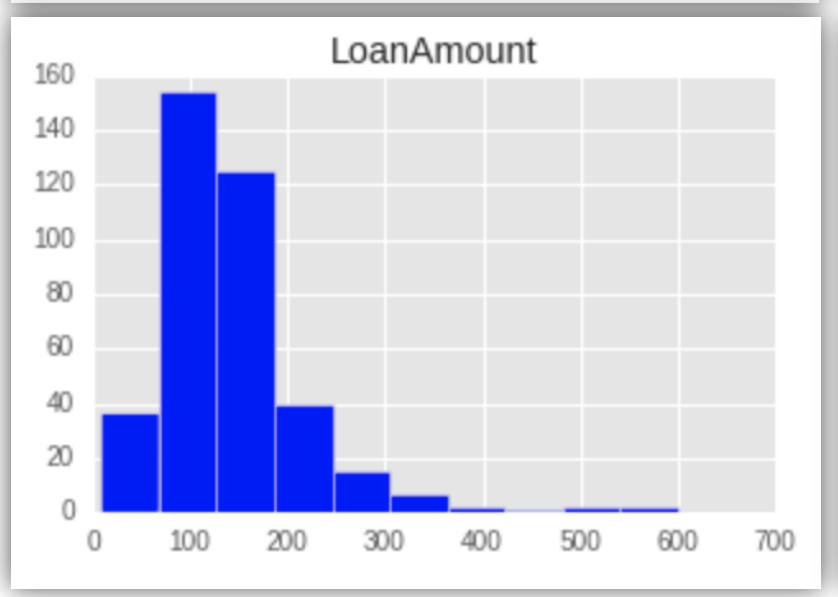
https://github.com/subashgandyer/datasets/blob/main/loan_prediction.zip

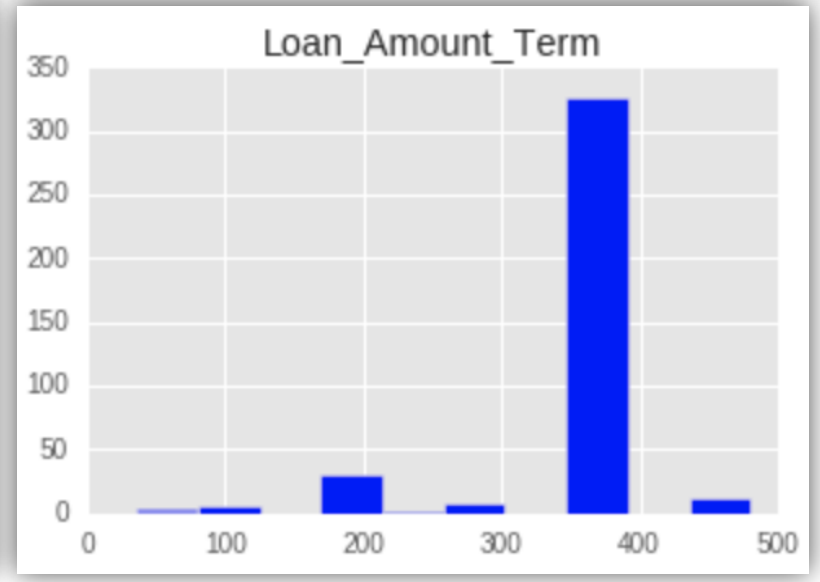
Dataset – Loan Prediction dataset











Feature Scaling

MinMaxScaler

Scales down all the features to a same range between 0 and 1

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

MinMaxScaler

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

```
# Scaling down both train and test data set
X_train_minmax=min_max.fit_transform(X_train[['ApplicantIncome', 'CoapplicantIncome',
                'LoanAmount', 'Loan_Amount_Term', 'Credit_History']])
X_test_minmax=min_max.fit_transform(X_test[['ApplicantIncome', 'CoapplicantIncome',
                'LoanAmount', 'Loan_Amount_Term', 'Credit_History']])

# KNN Model
knn=KNeighborsClassifier(n_neighbors=5)

# Fit the model
knn.fit(X_train_minmax,Y_train)

# Checking the model's accuracy
accuracy_score(Y_test,knn.predict(X_test_minmax))
```

Exercise: Build a Feature Scaler for Logistic Regression model

Use the previous code scripts in creating a Feature Scaler tool for Logistic Regression model

https://github.com/subashgandyer/datasets/blob/main/loan_prediction.zip

Feature Standardization

scale() - Feature Standardization

Scales down all the features to a standard normal distribution with zero mean and one Standard deviation

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

scale

```
from sklearn.preprocessing import scale
X_train_scale=scale(X_train[['ApplicantIncome', 'CoapplicantIncome',
                    'LoanAmount', 'Loan_Amount_Term', 'Credit_History']])
X_test_scale=scale(X_test[['ApplicantIncome', 'CoapplicantIncome',
                     'LoanAmount', 'Loan_Amount_Term', 'Credit_History']])

# Fitting logistic regression on our standardized data set
from sklearn.linear_model import LogisticRegression
log=LogisticRegression(penalty='l2',C=.01)
log.fit(X_train_scale,Y_train)

# Checking the model's accuracy
accuracy_score(Y_test,log.predict(X_test_scale))
```

StandardScaler

```
from sklearn import preprocessing
import numpy as np
X_train = np.array([[ 1., -1.,  2.],
                    [ 2.,  0.,  0.],
                    [ 0.,  1., -1.]])
scaler = preprocessing.StandardScaler().fit(X_train)
scaler
```

```
scaler.mean_


scaler.scale_
```

```
X_scaled = scaler.transform(X_train)
X_scaled
```

Pipeline

```
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

X, y = make_classification(random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
pipe = make_pipeline(StandardScaler(), LogisticRegression())
pipe.fit(X_train, y_train)  # apply scaling on training data

pipe.score(X_test, y_test)
```

Exercise: Build a Feature Scaler for Support Vector Machine model

Use the previous code scripts in creating a Feature Scaler tool for Support Vector Machine model

https://github.com/subashgandyer/datasets/blob/main/loan_prediction.zip

Feature Normalization

Normalization

Normalization is the process of scaling individual samples to have unit norm.

This process can be useful if you plan to use a quadratic form such as the dot-product or any other kernel to quantify the similarity of any pair of samples.

```
X = [[ 1., -1.,  2.],
     [ 2.,  0.,  0.],
     [ 0.,  1., -1.]]
X_normalized = preprocessing.normalize(X, norm='l2')
```

Categorical Variables

Label Encoding

Label Encoding

Categorical features should be transformed into numerical values

ML models need numeric arrays as inputs not strings

LabelEncoder encodes labels with value between 0 and n_classes – 1 with respect to alphabetical order

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
```

Label Encoding

```
# Import label encoder
from sklearn import preprocessing
```

```
label_encoder = preprocessing.LabelEncoder()
```

```
data['Country']= label_encoder.fit_transform(data['Country'])
print(data.head())
```

Country	Age	Salary
India	44	72000
US	34	65000
Japan	46	98000
US	35	45000
Japan	23	34000

Country	Age	Salary
0	44	72000
2	34	65000
1	46	98000
2	35	45000
1	23	34000

Challenges:

India < Japan < USA

Exercise: Build a Label Encoding Preprocessing task

https://github.com/subashgandyer/datasets/blob/main/loan_prediction.zip

One-hot Encoding

One-hot Encoding

Transforms each categorical feature with n possible values into n binary features with only one active

One-hot Encoding

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

```
# creating one hot encoder object
onehotencoder = OneHotEncoder()
```

```
X = onehotencoder.fit_transform(df[['Property_Area']]).toarray()
```

```
df2 = pd.DataFrame(X)
df2_new = pd.concat([df,df2], axis=1)
```

```
#droping the Property_Area column
df= df.drop(['Property_Area'], axis=1)
```

Credit_History	Property_Area	0	1	2
1	Urban	0.0	0.0	1.0
1	Semiurban	0.0	1.0	0.0
1	Semiurban	0.0	1.0	0.0
1	Urban	0.0	0.0	1.0
1	Urban	0.0	0.0	1.0
...	...	...	...	...
0	Rural	1.0	0.0	0.0
1	Rural	1.0	0.0	0.0
1	Rural	1.0	0.0	0.0
1	Semiurban	0.0	1.0	0.0
1	Semiurban	0.0	1.0	0.0

One-hot Encoding - get_dummies

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

```
# creating one hot encoder object
onehotencoder = OneHotEncoder()
```

```
df3=pd.get_dummies(df[["Property_Area"]])
```

```
df3_new=pd.concat([df,df3],axis=1)
```

```
del df3_new['Property_Area']
df3_new
```

Property_Area_Rural	Property_Area_Semiurban	Property_Area_Urban
0	0	1
0	1	0
0	1	0
0	0	1
0	0	1
...	...	...
1	0	0
1	0	0
1	0	0
0	1	0
0	1	0

Exercise: Build a One Hot Encoding Preprocessing task

https://github.com/subashgandyer/datasets/blob/main/loan_prediction.zip

Data Preprocessing

Hands-on

Data Preprocessing Task

Dataset: simple toy data with 10 entries

https://github.com/subashgandyer/datasets/blob/main/raw_data.csv

Data Dictionary:

Country - Country of Origin

Age - Age of the customer

Salary - Salary of the customer

Married - Marital Status

Create a fresh Python Notebook and follow along the simple data preprocessing (categorical features, missing values) process.

Imputation

Imputation

Process of dealing with missing values

1. Discard entire rows or columns containing missing values

2. Impute (infer them from the known part of the data) the missing values

Univariate Imputation - SimpleImputer

Multivariate Imputation - IterativeImputer

Univariate Imputation - SimpleImputer

Missing values can be imputed from

- a constant value
- statistics (mean, median, …)

https://github.com/subashgandyer/datasets/blob/main/heart_disease.csv

Univariate Imputation - SimpleImputer

```
import numpy as np
from sklearn.impute import SimpleImputer
imp = SimpleImputer(missing_values=np.nan, strategy='mean')
imp.fit([[1, 2], [np.nan, 3], [7, 6]])
```

```
X = [[np.nan, 2], [6, np.nan], [7, 6]]
print(imp.transform(X))
```

```
[[4.        2.       ]
 [6.        3.666...]
 [7.        6.       ]]
```

Exercise: Build a SimpleImputer Preprocessing task

Assignment: In the same notebook, some are hands-on in-class exercises and the last portion is left as an assignment

https://github.com/subashgandyer/datasets/blob/main/heart_disease.csv

Multivariate Imputation - IterativeImputer

Models each feature with missing values as a function of other features, uses that estimate for imputation

Iterative approach:

At each step, one feature column is designated as output y and other features are treated as inputs X

A regressor is fit on (X, y) for known y

Regressor model is used to predict the missing values of y

Repeat the above steps until all features are considered and for max_iter repetitions

Multivariate Imputation - IterativeImputer

```
import numpy as np
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
imp = IterativeImputer(max_iter=10, random_state=0)
imp.fit([[1, 2], [3, 6], [4, 8], [np.nan, 3], [7, np.nan]])
```

```
X_test = [[np.nan, 2], [6, np.nan], [np.nan, 6]]
# the model learns that the second feature is double the first
print(np.round(imp.transform(X_test)))
```

```
[[ 1.  2.]
 [ 6. 12.]
 [ 3.  6.]]
```

Exercise: Build a IterativeImputer Preprocessing task

https://github.com/subashgandyer/datasets/blob/main/heart_disease.csv

Discretization

Discretization

Quantization or Binning provides a way to partition continuous features into discrete values

KBinsDiscretizer

Discretizes features into k bins

```
X = np.array([[ -3., 5., 15 ],
              [  0., 6., 14 ],
              [  6., 3., 11 ]])
est = preprocessing.KBinsDiscretizer(n_bins=[3, 2, 2], encode='ordinal').fit(X)
```

Binarization

Process of thresholding numerical features to get boolean values

```
X = [[ 1., -1.,  2.],
     [ 2.,  0.,  0.],
     [ 0.,  1., -1.]]

binarizer = preprocessing.Binarizer().fit(X)
binarizer



binarizer.transform(X)
```

```
binarizer = preprocessing.Binarizer(threshold=1.1)
binarizer.transform(X)
```

Further Reading

Scikit-learn documentation

https://scikit-learn.org/stable/modules/impute.html#impute