HORIZON
SCHOOL OF DIGITAL TECHNOLOGIES

Data Preparation

Khouloud Chalbi

khouloud.chelbi@horizon-university.tn

HORIZON
SCHOOL OF DIGITAL TECHNOLOGIES

HORIZON
SCHOOL OF DIGITAL TECHNOLOGIES

Outline

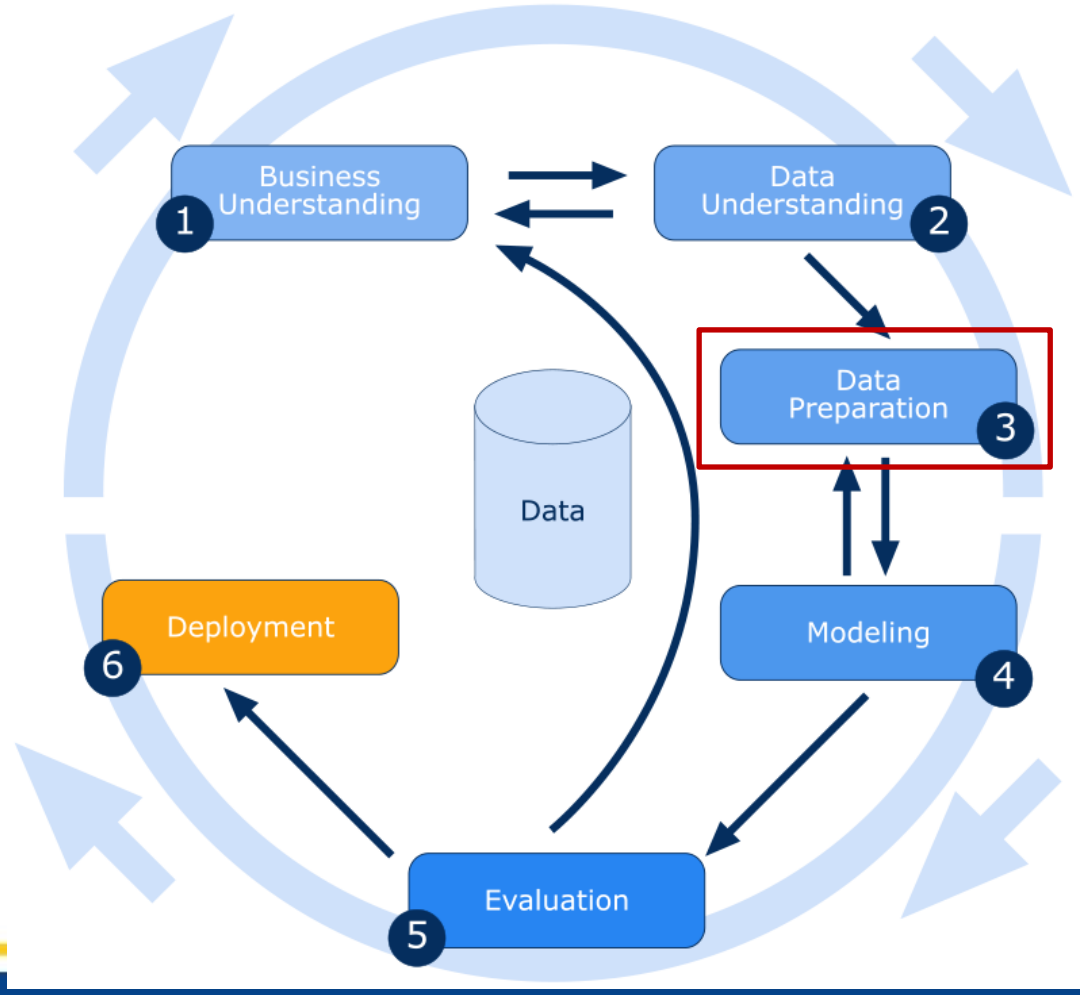
- Introduction to Data Preparation
- Data Discovery & Profiling
- Data Cleaning
- Data Transformation
- Feature Engineering
- Dimensionality Reduction

HORIZON
SCHOOL OF DIGITAL TECHNOLOGIES

1- Introduction to Data Preparation

What is Data Preparation ?

- Data preparation is the process of **cleaning**, **formatting**, **transforming**, and **integrating** raw data into a format suitable for modeling.
- Data preparation is a crucial step as it directly **impacts the performance of ML models**.
- Effective data preparation can help to **reduce errors**, **inconsistencies**, and **bias** in the data, and **improve the overall performance** of ML models.



Data Preparation Process

Data collection

- Defining required data
- Gathering and combining data from different sources

Data discovery and profiling

- Exploratory Data Analysis (EDA)

Data cleansing

- Formatting
- Cleaning
- Sampling

Data transformation and enrichment

- Scaling / Distribution transformation
- Aggregation
- Decomposition

Data validation and publishing

- Validating consistency, completeness and quality
- Storing in a data warehouse, data lake or another repository

Data Preparation Tasks

- **Data Discovery & Profiling:** Exploring the collected data to better understand what it contains and what needs to be done to prepare it for modeling.
- **Data Cleaning:** Identifying and correcting errors, missing values and outliers in the data.
- **Data Transformation:** Changing the scale (feature scaling) or distribution of variables (Distribution Transformation).
- **Feature Engineering:** Deriving new features from available ones.
- **Dimensionality Reduction:** Selecting more relevant features and creating compact projections with lower dimension of the data.

2- Data Discovery & Profiling

What are Data Discovery & Profiling ?

- **Data Discovery:** It is the process of **identifying, locating, and obtaining data from various sources**. It involves searching for relevant data, understanding its structure and format, and determining its relevance to the business.
- **Data Profiling:** It the process of **analyzing data to gain an understanding of its quality, structure, and content**. It involves examining data values, patterns, and relationships to identify any potential issues.
- Data discovery and profiling are used to ensure that the **data is accurate, consistent, and complete, and that it meets the requirements of the business**.

Load, Display, Get information & statistics

Load and display

```
import pandas as pd

# Load the data from a CSV file
data = pd.read_csv('filename.csv')

# Display the first few rows of the data
print(data.head())
```

Get basic information and statistics

```
# Display basic information of the data
print(data.info())

# Display basic statistics of the data
print(data.describe())
```

Get data types of the columns

```
# Display the data types of the columns
print(data.dtypes)
```

Check for Missing Values & Duplicates

Check for missing values

```
# Check if there are any missing values in the data
print(data.isnull().sum())
```

Check for duplicated rows

```
# Check for duplicate rows in the data
print(data.duplicated().sum())
```

Explore Categorical Features

Get unique values of a categorical column

```
# Display the unique values of a categorical column
print(data['column_name'].unique())
```

Group data by a column and get summary statistics

```
# Group the data by a column and get the mean of each group
grouped_data = data.groupby('column_name').mean()

# Display the summary statistics
print(grouped_data)
```

Data Profiling with Visualization (1/2)

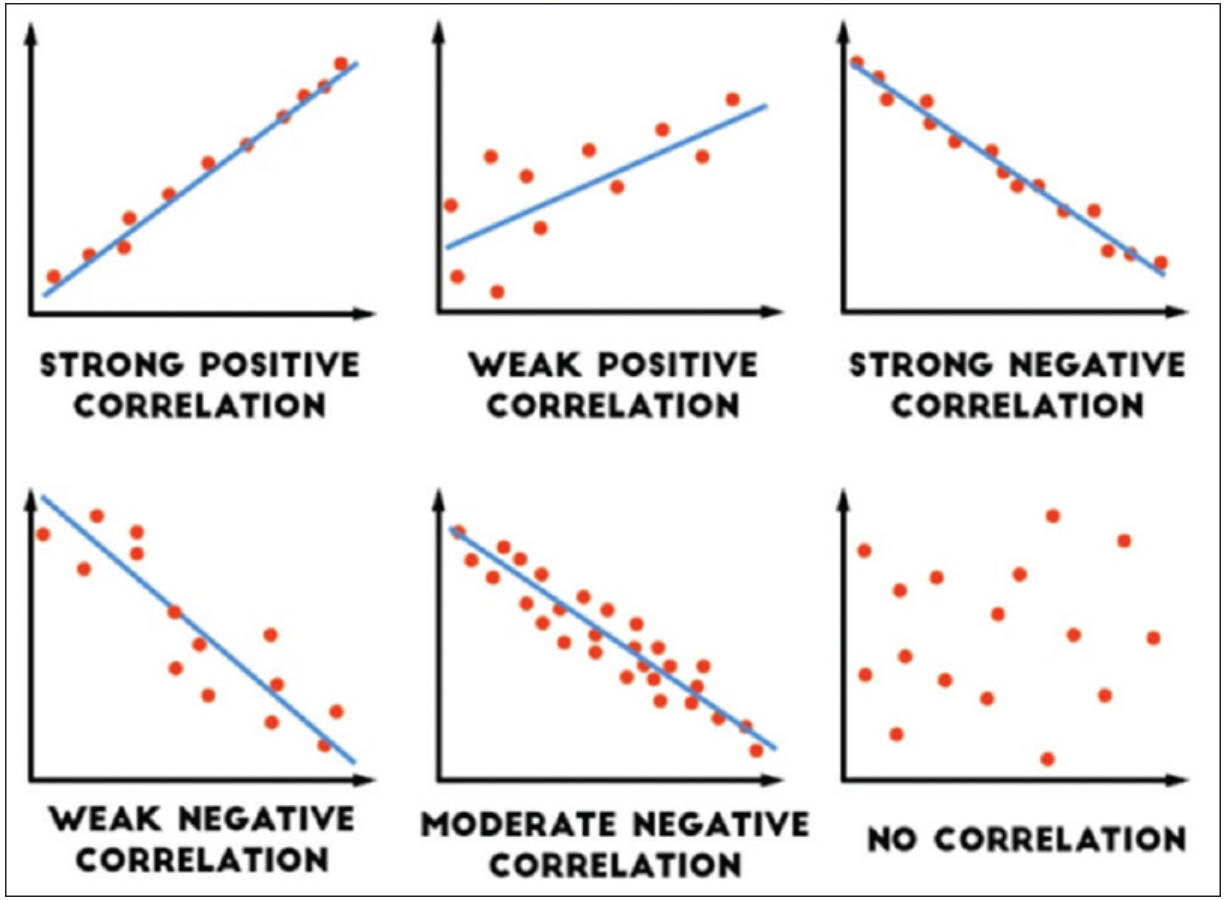
Scatter Plots: Exploring the relationship between two variables

```
data.plot.scatter(x='column_name_1', y='column_name_2')
plt.title('Scatter plot of column_name_1 vs column_name_2')
plt.xlabel('column_name_1')
plt.ylabel('column_name_2')
plt.show()
```

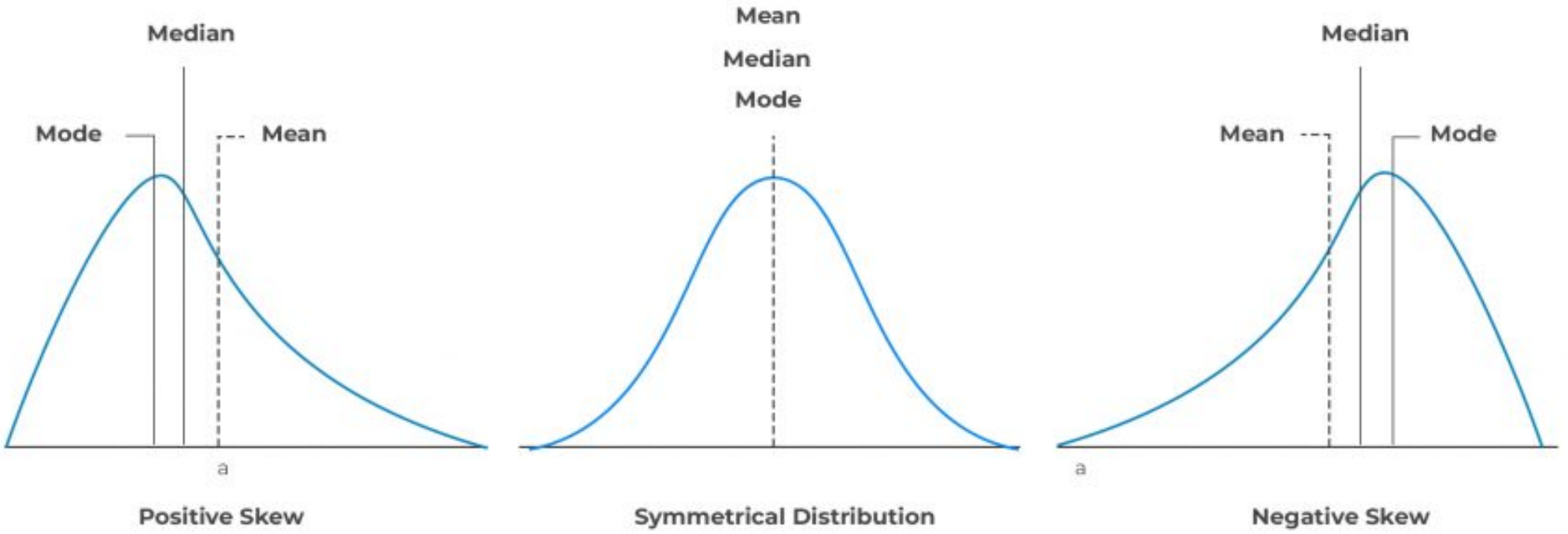
Histograms: Visualizing the distribution of a variable by grouping the data into bins and plotting the frequency of each bin.

```
data['column_name'].plot.hist(bins=10)
plt.title('Histogram of column_name')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.show()
```

Scatter plot: Correlation Between variables



Histogram plot: Asymmetry of a Distribution



Data Profiling with Visualization (2/2)

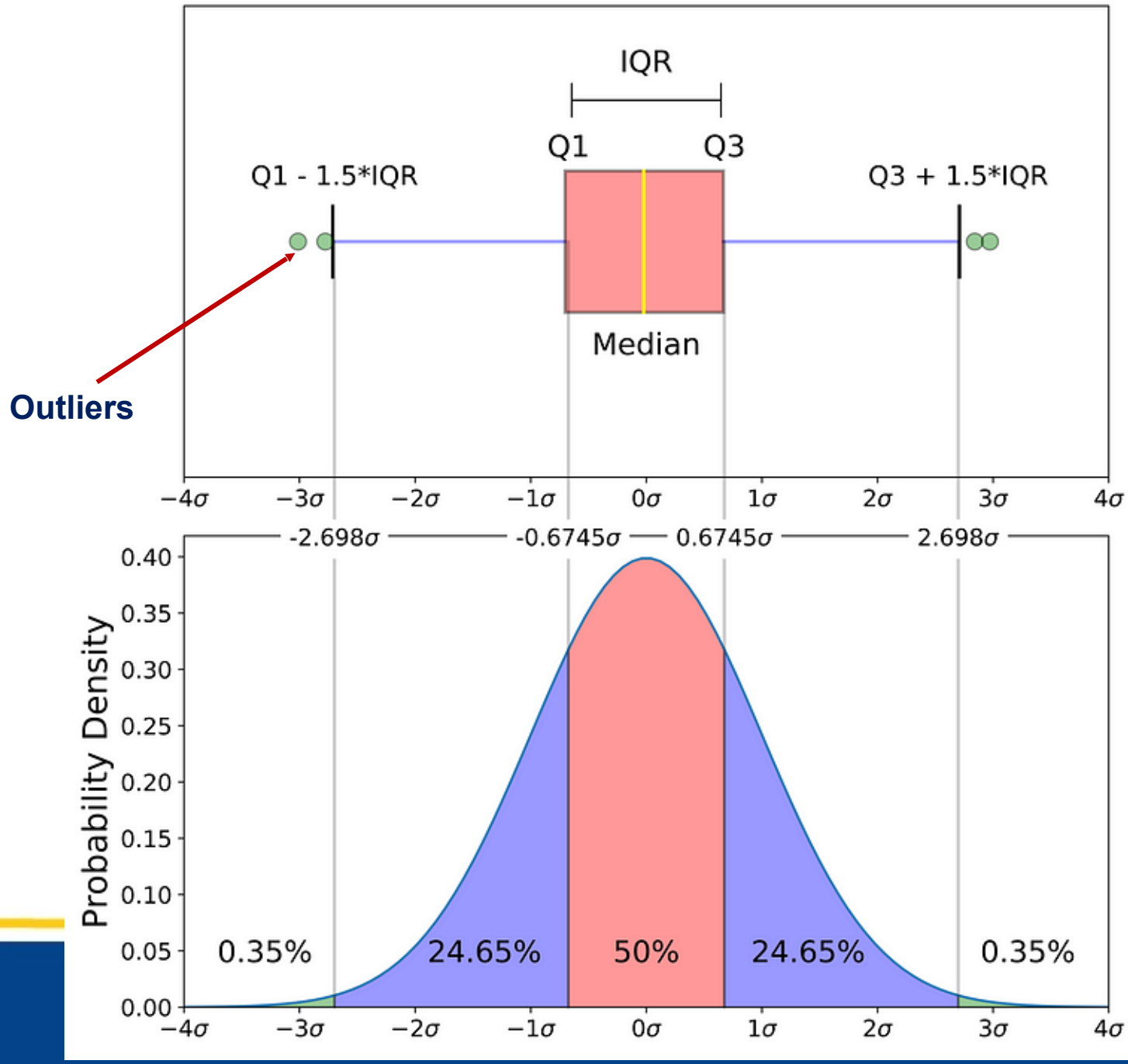
Box Plots: Showing the range of a variable, **including the median, quartiles, and outliers.**

```
# create a box plot of the 'Values' column
df.boxplot(column=['Values'])
```

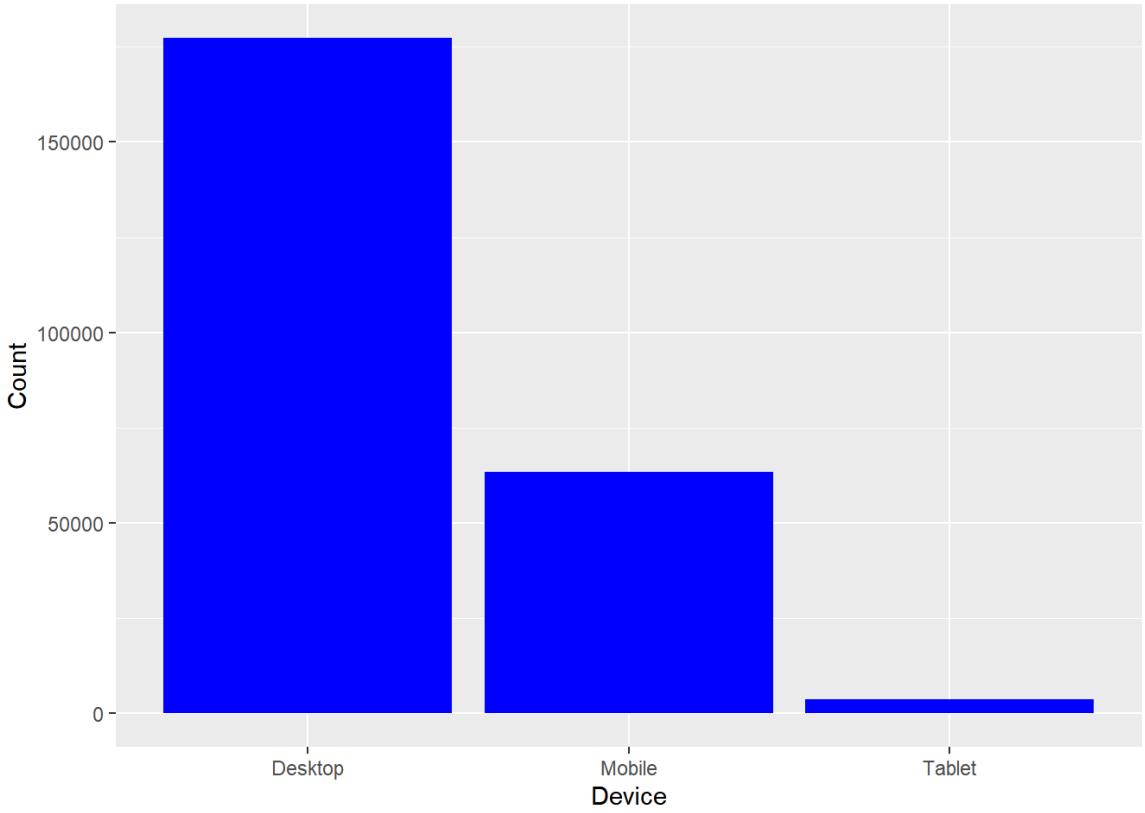
Bar Plots: Showing the frequency of values in a categorical column.

```
# get frequency count of each category
category_counts = data['category'].value_counts()

# create bar plot
category_counts.plot.bar()
```

Box Plot: Check for Outliers

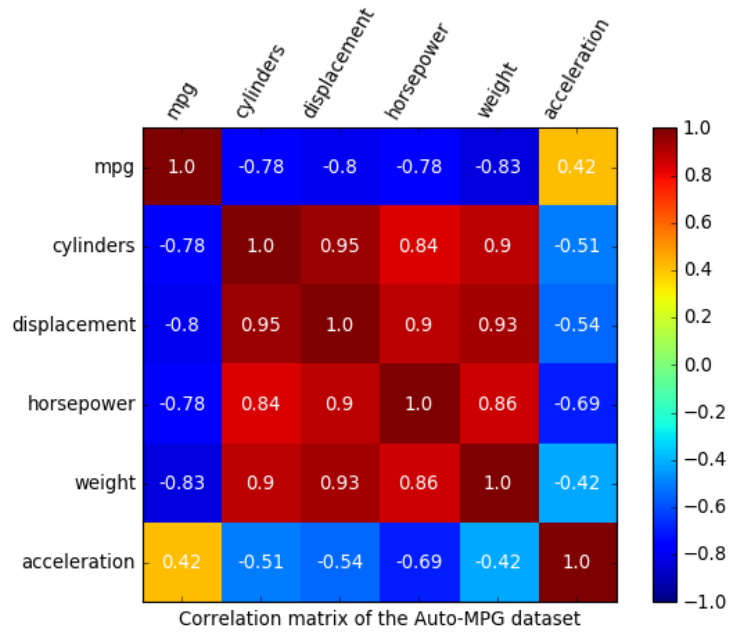


Bar Plots: Frequency of Values of a Categorical Column



Explore Correlation Between Variables

```
correlation_matrix = data.corr()
plt.figure(figsize=(10,10))
plt.title('Heat map of the correlation matrix')
plt.xlabel('Columns')
plt.ylabel('Columns')
plt.imshow(correlation_matrix, cmap='coolwarm', interpolation='nearest')
plt.colorbar()
plt.show()
```



Correlation matrix of the Auto-MPG dataset

3- Data Cleaning

Data Cleaning Steps

Remove irrelevant observations	• Delete duplicated rows. • Delete columns that contain single values or have small stds.
Handle missing values	• Delete or impute missing values. • In some cases, keep missing if they contain relevant information.
Correct errors	• Correct errors e.g., values, mislabeled classes, names of features, units etc.
Filter outliers	• Detect and filter outliers which are not fitting in datasets.

Remove Irrelevant Observations

Remove duplicated rows

```
# remove the duplicated rows based on all columns
df = df.drop_duplicates()
```

Remove columns with constant values

```
# remove the columns with constant values
df = df.drop(df.columns[np.where(df.std() == 0)], axis=1)
```

Remove columns with high number of missing values

```
# Drop columns with less than 75% non-null values
threshold = int(len(df) * 0.75) # set the threshold to 75%
df = df.dropna(axis=1, thresh=threshold)
```

Remove columns with small stds

```
# remove columns with std below a threshold value
threshold = 0.5
stds = df.std()
df = df.loc[:, stds >= threshold]
```

Remove Highly Correlated Features

```python
import pandas as pd
import numpy as np

# create a sample DataFrame
df = pd.DataFrame({
    'col1': [1, 2, 3, 4, 5],
    'col2': [2, 4, 6, 8, 10],
    'col3': [3, 6, 9, 12, 15],
    'col4': [4, 8, 12, 16, 20]
})

# calculate the correlation matrix
corr_matrix = df.corr().abs()

# define threshod
corr_th = 0.95

# get the upper triangle of the correlation matrix
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))

# get the indices of the highly correlated columns
high_corr_cols = [col for col in upper.columns if any(upper[col]> corr_th)]

print(high_corr_cols)

# drop the highly correlated columns from the DataFrame
df = df.drop(df[high_corr_cols], axis=1)

print(df)
```

```
['col2', 'col3', 'col4']
   col1
0     1
1     2
2     3
3     4
4     5
```

Handle Missing Values

- **Remove missing values:**

```
# drop rows with missing values
df_clean = df.dropna()
```

- **Impute missing values:**

```
# replace missing values with the mean
df_mean = df.fillna(df.mean())
```

```
# replace missing values with the median
df_median = df.fillna(df.median())
```

Factor to Consider	Delete Missing Values	Impute Missing Values
Percentage of Missing Values	High (>5-10%)	Low (<5-10%)
Importance of Missing Values	Not Critical	Critical
Patterns of Missing Values	Random	Systematic
Size of the Dataset	Large	Small
Type of Analysis	Descriptive	Inferential

```
# replace missing values with the previous value
df_ffill = df.fillna(method='ffill')

# replace missing values with the next value
df_bfill = df.fillna(method='bfill')
```

Correct Errors

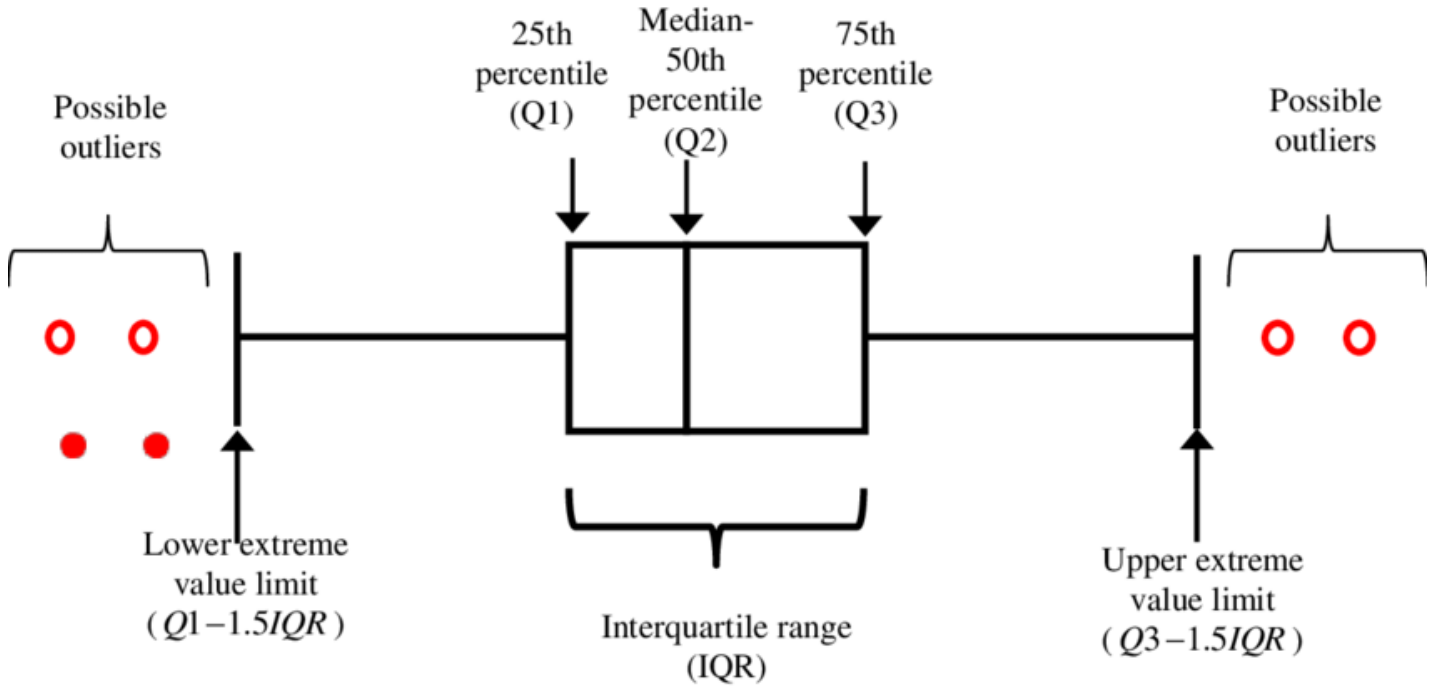
- **Correct erroneous values:**

```
df['Age'] = df['Age'].replace({-5: 35})
```

- **Drop erroneous values:**

```
df = df.drop(index=df[df['Age'] < 0].index)
```

Filter Outliers: Interquartile Range IQR




```
Q1 = df['Age'].quantile(0.25)
Q3 = df['Age'].quantile(0.75)
IQR = Q3 - Q1

df = df[~((df['Age'] < Q1 - 1.5 * IQR) | (df['Age'] > Q3 + 1.5 * IQR))]
```

4- Data Transformation

Split Data: Training & Testing Sets

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris

# load the iris dataset
iris = load_iris()

# create a DataFrame of the features and target variable
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = pd.DataFrame(iris.target, columns=['target'])

# split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# print the shape of the training and testing sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

Feature Scaling

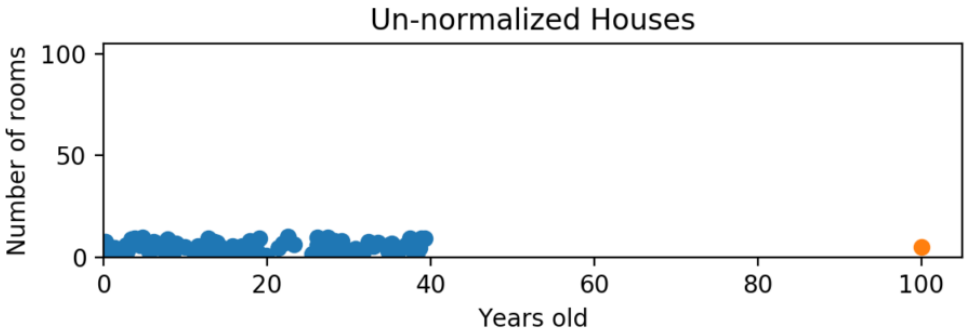
Normalization	Standardization
Feature scaling method to bring the data into common range such as [0, 1], [-1, 1], etc.	Feature scaling method bring the data with mean 0 and unit variance
Scikit-learn provides MinMaxScaler, MaxAbsScaler and RobustScaler methods for normalization	Scikit-learn provides StandardScaler for standardization
MinMaxScaler and MaxAbsScaler are sensitive to outliers whereas RobustScaler is more robust to outliers	Standardization is less sensitive to outliers compared to MinMaxScaler and MaxAbsScaler
Useful when we don't know about the distribution of features and there are no or little outliers - MinMaxScaler: if features don't follow normal distriubtion and if there are no or less outliers - MaxAbsScaler: if the data is sparse - RobustScaler: if the data contains outliers	Useful when we know features are normally distributed (Gaussian distribution)

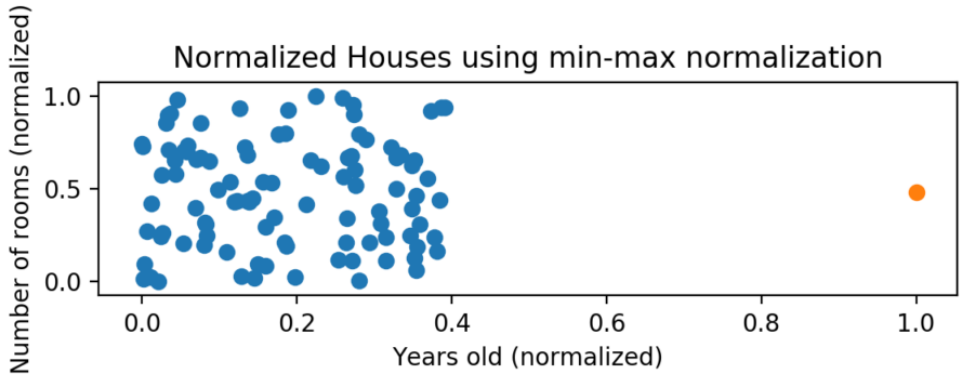
Min-Max Normalization

$$X_{new} = \frac{X_i - \min(X)}{\max(x) - \min(X)}$$

```
from sklearn.preprocessing import MinMaxScaler
```

```
# create a scaler object and fit it to the training data
scaler = MinMaxScaler()
scaler.fit(X_train)

# transform the training and testing data using the scaler object
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```



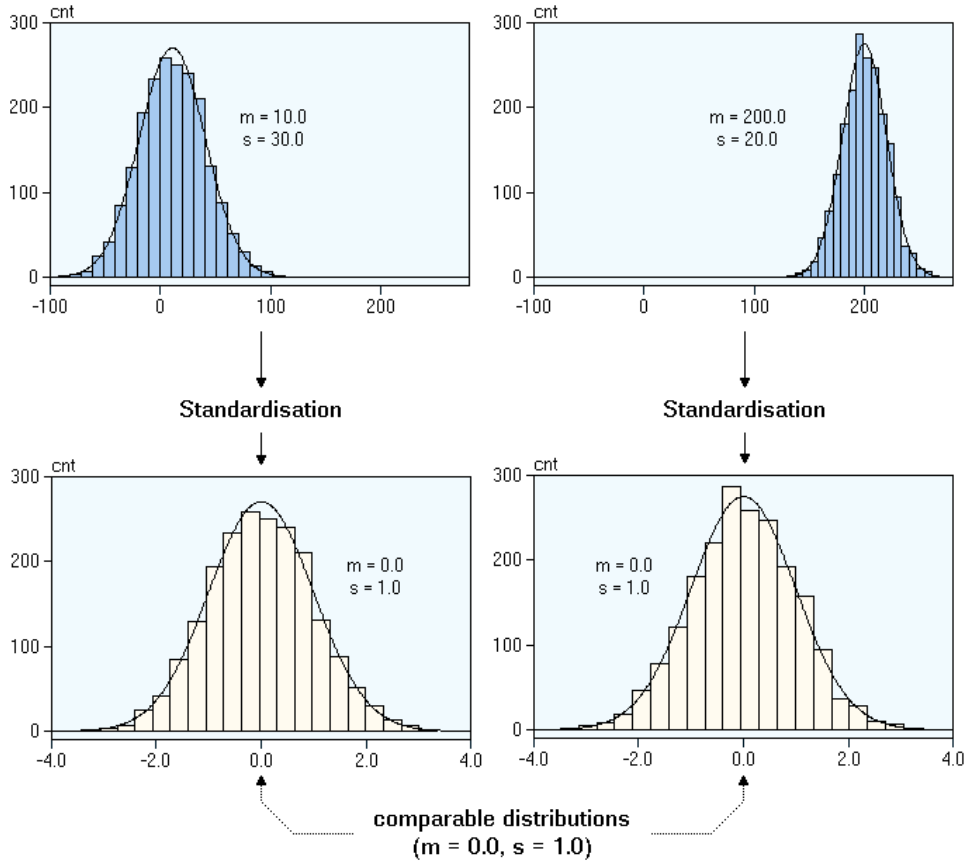


Standardization

$$x_{new} = \frac{x - \mu}{\sigma}$$

```
from sklearn.preprocessing import StandardScaler
```

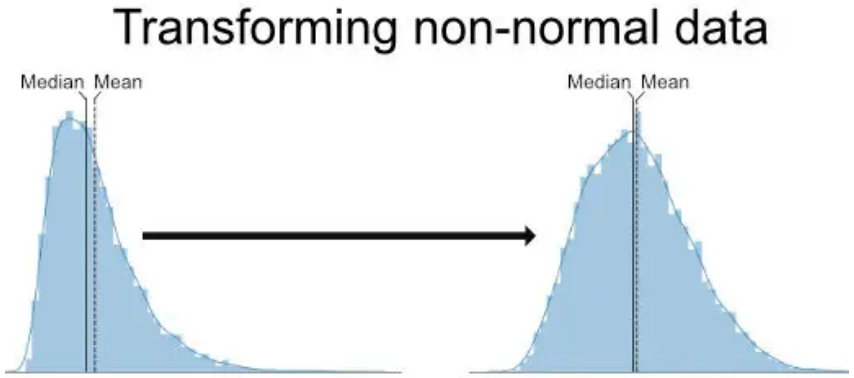
```
# create a scaler object and fit it to the training data
scaler = StandardScaler()
scaler.fit(X_train)

# transform the training and testing data using the scaler object
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```



Distribution Transformation

```
from sklearn.preprocessing import PowerTransformer
```

```
# create a transformer object and fit it to the training data
transformer = PowerTransformer(method='yeo-johnson')
transformer.fit(X_train)

# transform the training and testing data using the transformer object
X_train_transformed = transformer.transform(X_train)
X_test_transformed = transformer.transform(X_test)
```



5- Feature Engineering

Encoding Categorical Features

Encoding Method	Description	Pros	Cons
One-Hot Encoding	Creates a new binary column for each category in the feature	Easy to understand, handles nominal data well	Increases the dimensionality of the data
Label Encoding	Assigns each category a numerical value	Preserves order of ordinal data, reduces dimensionality	May introduce arbitrary values or imply an order that does not exist
Binary Encoding	Converts each category to a binary string	Reduces dimensionality, handles nominal data well	Can be difficult to interpret
Frequency Encoding	Replaces each category with its frequency in the data	Preserves information about the frequency of each category, reduces dimensionality	May not work well for rare categories or categories with equal frequency
Target Encoding	Replaces each category with the mean target value for that category	Preserves information about the relationship between the feature and target, handles nominal and ordinal data well	May overfit or introduce bias if the target variable is correlated with the encoding

One-Hot Encoding for Nominal Features

```
Before One-Hot Encoding
    color
0     red
1   green
2    blue
3     red
4     red
5    blue
```

```
After One-Hot Encoding
    color_blue  color_green  color_red
0            0            0          1
1            0            1          0
2            1            0          0
3            0            0          1
4            0            0          1
5            1            0          0
```

```python
import pandas as pd

# create a sample dataframe with a nominal categorical feature 'color'
df = pd.DataFrame({
    'color': ['red', 'green', 'blue', 'red', 'red', 'blue']
})

print("Before One-Hot Encoding\n", df)

# use get_dummies() to perform one-hot encoding
one_hot_encoded = pd.get_dummies(df['color'], prefix='color')

# concatenate the one-hot encoded dataframe with the original dataframe
df = pd.concat([df, one_hot_encoded], axis=1)

# drop the original 'color' column since it is no longer needed
df = df.drop('color', axis=1)

print("Afer One-Hot Encoding\n", df)
```

Label Encoding for Ordinal Features

```
After Label Encoding
   grade  age  grade_encoded
0      A   20              0
1      B   25              1
2      C   30              2
3      D   35              3
4      E   40              4
5      F   45              5
```

```python
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Create a sample DataFrame
df = pd.DataFrame({
    'grade': ['A', 'B', 'C', 'D', 'E', 'F'],
    'age': [20, 25, 30, 35, 40, 45]
})

# Create a label encoder object
le = LabelEncoder()

# Encode the "grade" column
df['grade_encoded'] = le.fit_transform(df['grade'])

# Print the encoded DataFrame
print("After Label Encoding\n", df)
```

Binning of Numerical Features

	What is it?	Why use it?	Methods
Binning	Grouping a continuous variable into a smaller number of categories or bins	1. Simplification of data; 2. Capturing non-linear relationships; 3. Handling outliers	Equal width, equal frequency, k-means

```python
import pandas as pd
import numpy as np

# create a DataFrame with a continuous variable
df = pd.DataFrame({'age': np.random.randint(18, 99, size=100)})

# create 5 bins with equal width
df['age_bin'] = pd.cut(df['age'], bins=5)

# print the first 10 rows
print(df.head(10))
```

	age	age_bin
0	45	(36.6, 50.4]
1	23	(18.0, 31.8]
2	27	(18.0, 31.8]
3	68	(64.2, 78.0]
4	51	(50.4, 64.2]
5	78	(64.2, 78.0]
6	24	(18.0, 31.8]
7	61	(50.4, 64.2]
8	73	(64.2, 78.0]
9	31	(18.0, 31.8]

Aggregation

- Aggregation is the process of creating new features by **computing summary statistics** on existing features.
- Aggregation can **help capture important patterns and relationships in the data**.
- By computing summary statistics on groups of data, aggregation can **highlight differences and similarities between groups**, and can help to **identify important clusters within the data**.
- Aggregation help to extract valuable insights from complex datasets and improve the accuracy of machine learning models.

Aggregation: Example

	age				income		
	mean	median	min	max	sum	count	mean
gender							
F	32.6	37.0	19	42	180000	3	60000
M	28.3	28.0	25	32	185000	3	61666.666667

```python
import pandas as pd
import numpy as np

# create a DataFrame with some data
data = {
    'gender': ['M', 'M', 'F', 'F', 'F', 'M'],
    'age': [25, 32, 19, 42, 37, 28],
    'income': [50000, 75000, 20000, 100000, 80000, 60000]
}
df = pd.DataFrame(data)

# group the data by gender and compute summary statistics
agg_df = df.groupby('gender').agg({
    'age': ['mean', 'median', 'min', 'max'],
    'income': ['sum', 'count', 'mean']
})

# print the aggregated data
print(agg_df)
```

Decomposition

- Decomposition is the process of **breaking down a feature into multiple new features**.
- Decomposition **is useful when a feature contains complex patterns or relationships** that may be difficult for a machine learning model to capture.
- Decomposition can help to **extract more useful information and improve the accuracy of the model**.

```python
import pandas as pd

# create a DataFrame with a date-time feature and a corresponding
data = {
    'date': ['2022-01-01 01:00:00', '2022-01-01 02:00:00'],
    'value': [10, 20]
}
df = pd.DataFrame(data)

# convert the date-time feature to a pandas datetime object
df['date'] = pd.to_datetime(df['date'])

# extract the year, month, day, and hour features
df['year'] = df['date'].dt.year
df['month'] = df['date'].dt.month
df['day'] = df['date'].dt.day
df['hour'] = df['date'].dt.hour

# print the resulting DataFrame
print(df)
```

```
                 date  value  year  month  day  hour
0 2022-01-01 01:00:00     10  2022      1    1     1
1 2022-01-01 02:00:00     20  2022      1    1     2
```

Polynomial Features

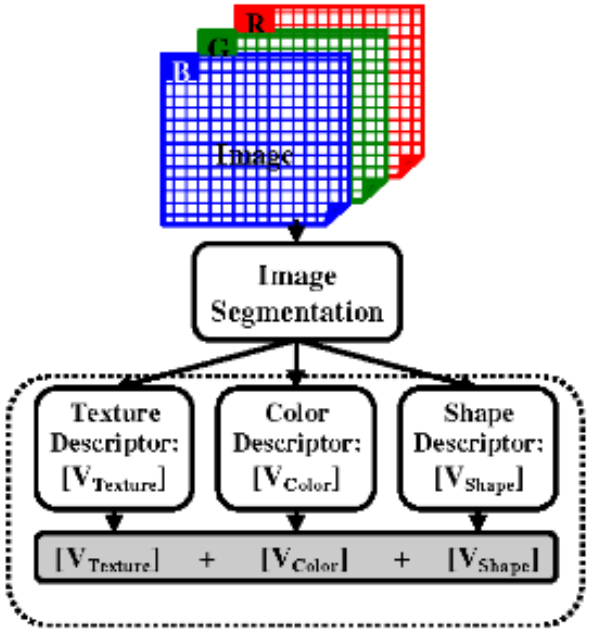
- Polynomial features are derived from the original features of a dataset by **raising them to different powers.**
- This can help to **capture non-linear relationships between features and the target variable**.

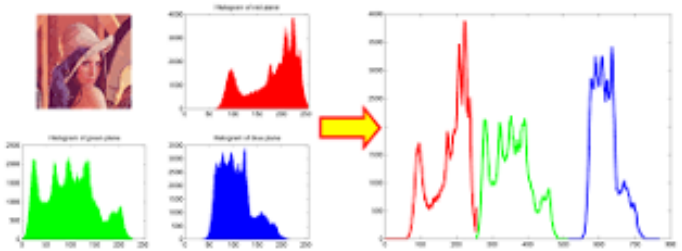
```python
from sklearn.preprocessing import PolynomialFeatures
import pandas as pd

# create a sample dataset
data = {
    'feature1': [1, 2, 3],
    'feature2': [4, 5, 6],
    'target': [10, 20, 30]
}
df = pd.DataFrame(data)

# extract polynomial features
poly = PolynomialFeatures(degree=2, include_bias=False)
poly_features = poly.fit_transform(df[['feature1', 'feature2']])

# create a new dataframe with the polynomial features and the target variable
poly_df = pd.DataFrame(poly_features, columns=poly.get_feature_names_out(['feature1', 'feature2']))
poly_df['target'] = df['target']

# print the resulting dataframe
print(poly_df)
```

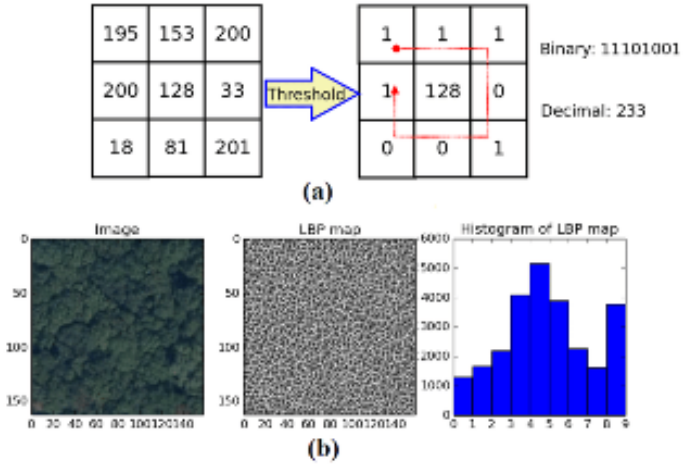
```
   feature1  feature2  feature1^2  feature1 feature2  feature2^2  target
0       1.0       4.0         1.0                4.0        16.0      10
1       2.0       5.0         4.0               10.0        25.0      20
2       3.0       6.0         9.0               18.0        36.0      30
```

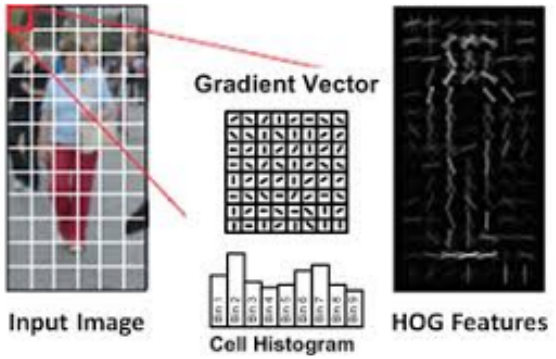
Feature Extraction for Computer Vision

- A RGB image is represented by **3 matrices {R,G,B}** of numeric pixel values ranging between 0 and 255.
- The **scikit-image** and **OpenCV** are excellent python libraries containing several algorithms for image processing and feature extraction.
- **Color, Texture and Shape features** can be extracted **globally** from the whole image or **locally** from patches or Regions of Interest (ROI).





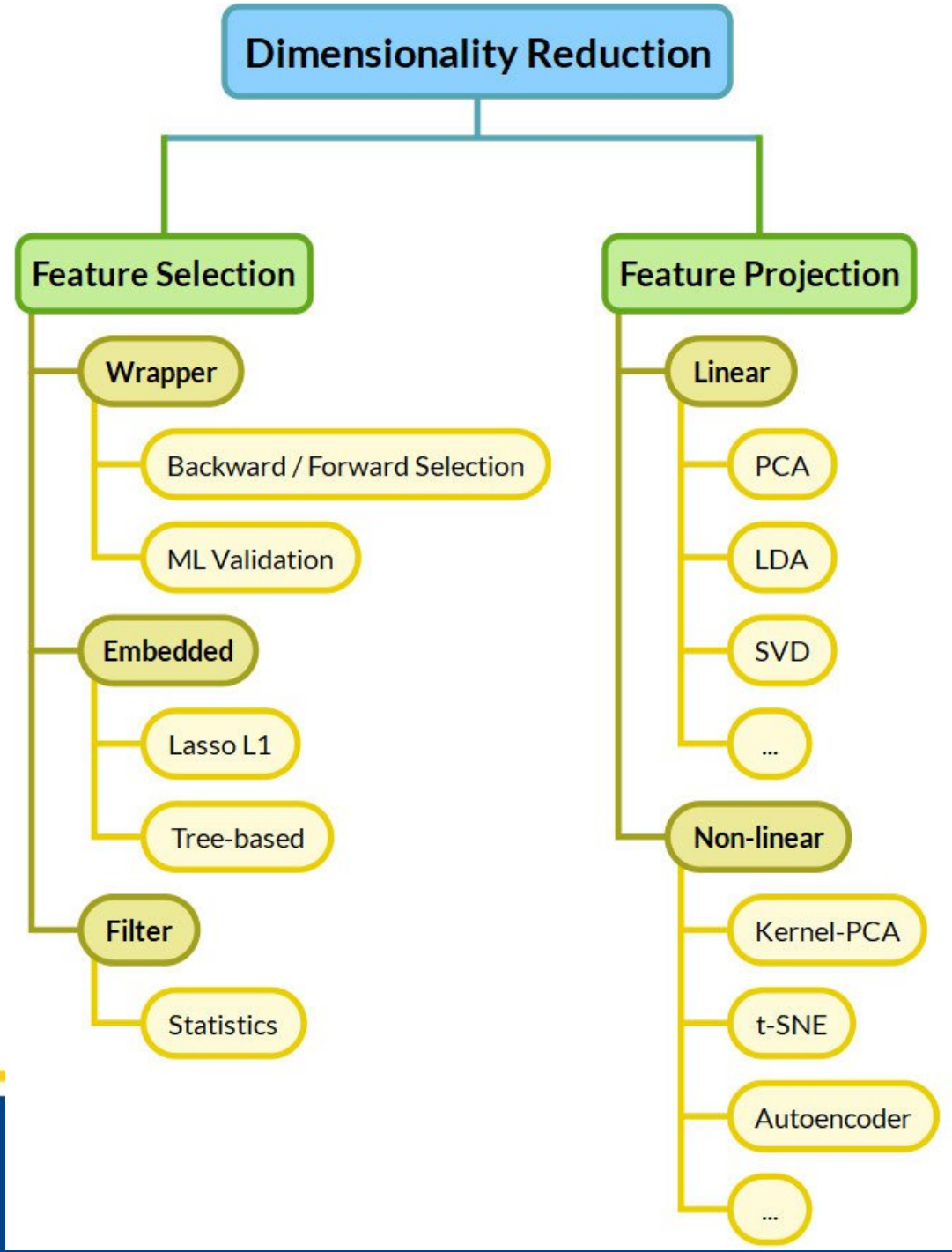




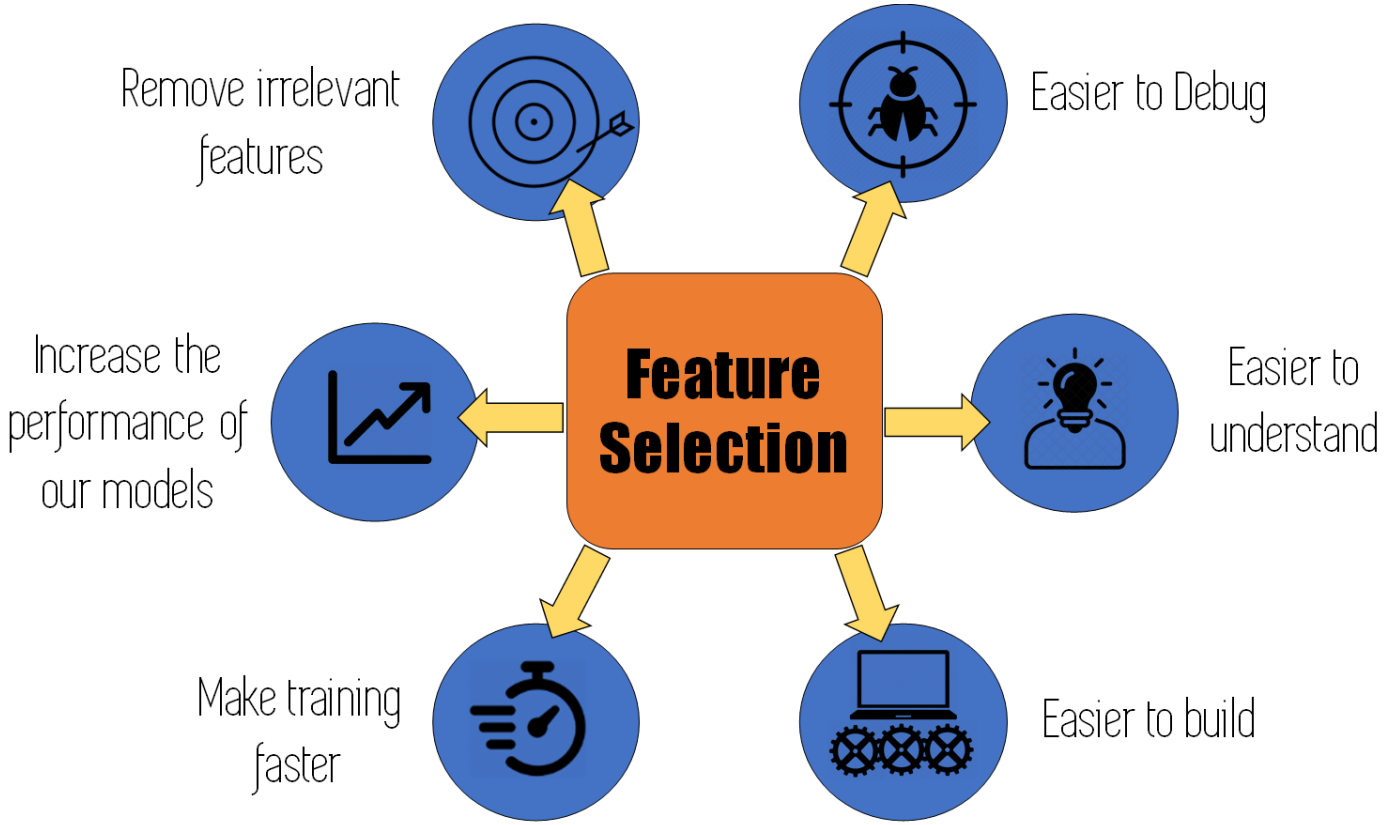
6- Dimensionality Reduction

Dimensionality Reduction

1. **Feature Selection**
2. **Feature Projection**



Why Is Feature Selection Useful ?

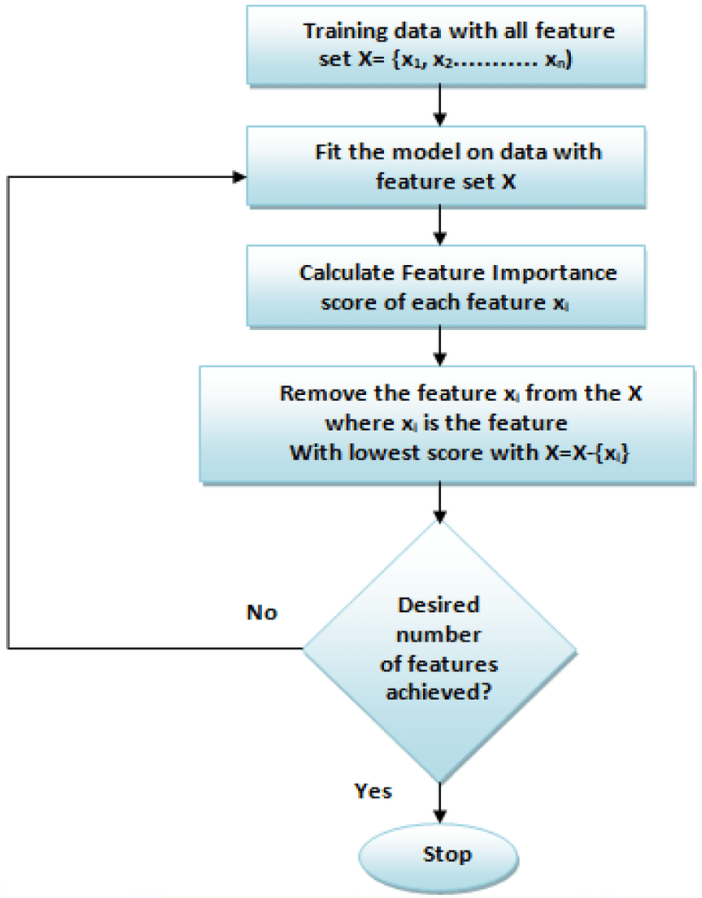


Filter Method: Mutual Information

A mutual information-based feature selection considers a feature effective if it has maximum MI with its class label (**maximum relevance**) and minimum MI with rest of the features (**minimum redundancy**).

```
from sklearn.feature_selection import SelectKBest, mutual_info_classif
from sklearn.datasets import load_iris

# load the iris dataset
iris = load_iris()
X, y = iris.data, iris.target

# select the k best features based on mutual information
k = 2
selector = SelectKBest(mutual_info_classif, k=k)
X_new = selector.fit_transform(X, y)
```

Wrapper Method: Recursive Feature Elimination




```python
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris

# load the iris dataset
iris = load_iris()
X, y = iris.data, iris.target

# create a logistic regression model
model = LogisticRegression()

# create the RFE object and set the number of features to select
rfe = RFE(model, n_features_to_select=2)

# fit the RFE object to the data
rfe.fit(X, y)
```

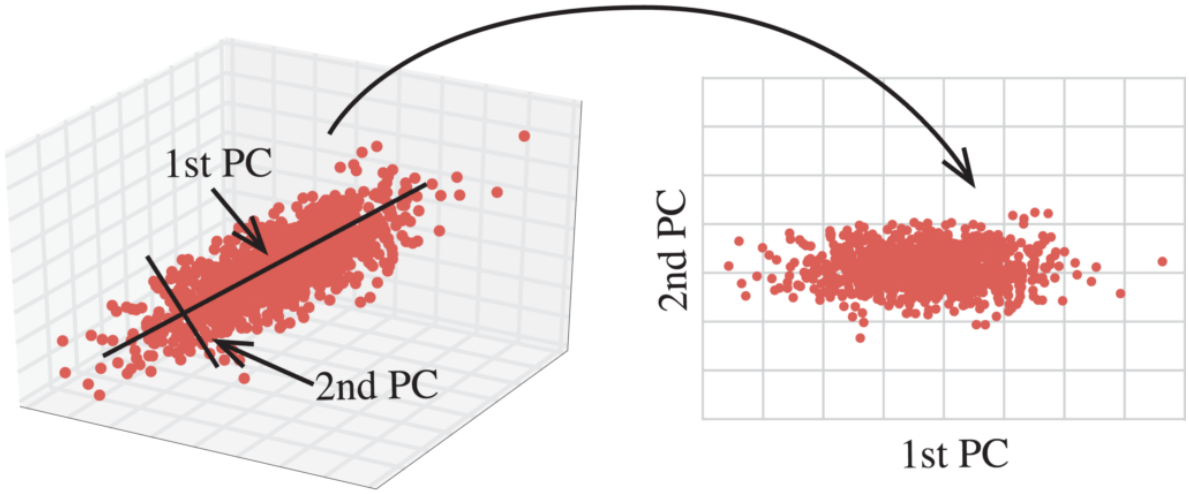
Embedding Method: Decision Tree

- The Decision Tree algorithm can automatically **select the most informative features** for a given task by splitting the data based on the features that provide the most **information gain**.
- Feature selection is based on feature importance.

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris

# load the iris dataset
iris = load_iris()
X, y = iris.data, iris.target

# create a decision tree classifier
clf = DecisionTreeClassifier()

# fit the decision tree classifier to the data
clf.fit(X, y)

# print the feature importances
print(clf.feature_importances_)
```

Why Is Feature Projection Useful ?

- Reduce computation time and model complexity.
- Decorrelate features.
- Simpler models are more robust on small datasets.
- Data visualization: more interpretable clusters, patterns and outliers if plotted in 2D/3D.



Feature Projection: Principal Component Analysis

(1) Standardize the data $\hat{\boldsymbol{\mu}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_i$

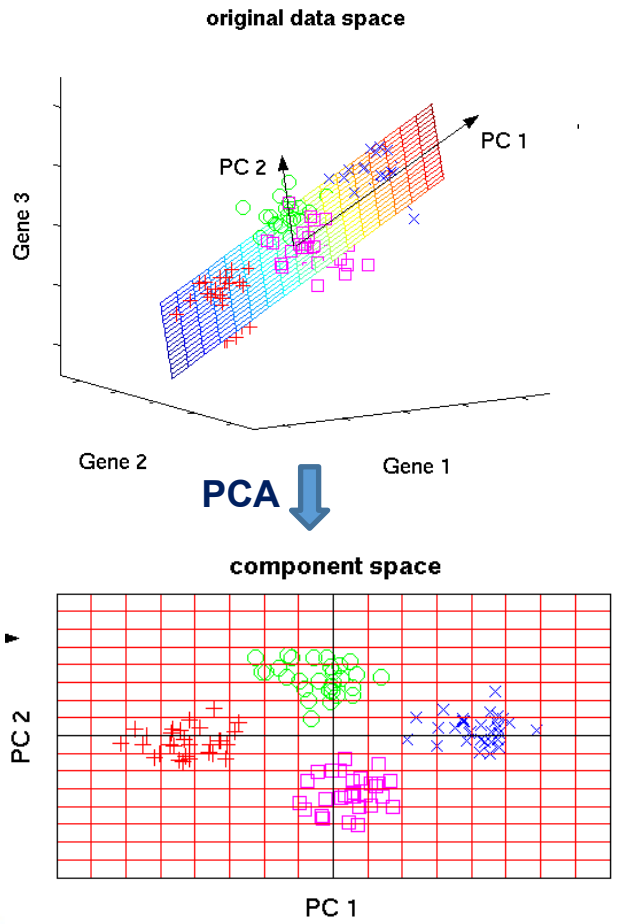
(2) Compute covariance matrix C. $\hat{\Sigma} = \frac{1}{N}\sum_{i=1}^{N} (\mathbf{x}_i - \hat{\boldsymbol{\mu}})(\mathbf{x}_i - \hat{\boldsymbol{\mu}})^{\mathsf{T}}$

(3) Compute the M eigenvectors $\{u_1, u_2, \ldots, u_D, u_{D+1}, \ldots, u_M\}$ with associated eigenvalues $\{\lambda_1 > \lambda_2 > \ldots > \lambda_D > \lambda_{D+1} > \ldots > \lambda_M\}$ such that $Cu_j = \lambda_j u_j,\ j = 1, \ldots,\ M$.

(4) Choose D such that $\frac{\sum_{i=1}^{D} \lambda_i}{\sum_{i=1}^{M} \lambda_i} > Threshold$ (e.g., 0.98).

(5) Pick-up the D first eigenvectors to form the D principal components.

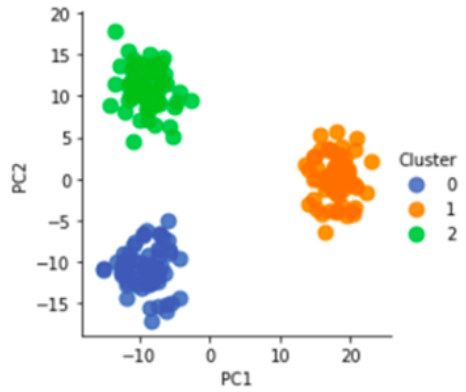
(6) Apply the transformation $G = [u_1, u_2, \ldots, u_D]$ from $\mathbb{R}^M \rightarrow \mathbb{R}^D$ with $D \ll M$: For each point $\boldsymbol{x} \in \mathbb{R}^M \rightarrow G^T \boldsymbol{x} \in \mathbb{R}^D$



```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
from sklearn.datasets import make_blobs
from sklearn import decomposition
%matplotlib inline
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
X, Y = make_blobs(n_features=10,n_samples=200, centers=3, random_state=4, cluster_std=2.5) #generate 3 clusters
print(X.shape)
pca = decomposition.PCA(n_components=4) # compute pca
pc = pca.fit_transform(X)
df = pd.DataFrame(data = pc , columns = ['PC1', 'PC2','PC3','PC4'])
df['Cluster'] = Y
print(df.head())
sns.lmplot( x="PC1", y="PC2",data=df, height=3.5, fit_reg=False, hue='Cluster', legend=True, scatter_kws={"s": 80})
```

```
(200, 10)
         PC1        PC2       PC3       PC4  Cluster
0 -11.595147  -9.036705 -3.760851  2.933824        0
1  17.991221  -0.578620  0.204946 -2.514116        1
2  15.999807  -4.228419  3.000733  2.414236        1
3  -6.374576 -11.515598  1.248614  1.176755        0
4  -8.304501  10.414224  0.858316 -1.007353        2
```

```
<seaborn.axisgrid.FacetGrid at 0x7f59637b9a50>
```



HORIZON
SCHOOL OF DIGITAL TECHNOLOGIES

EXPAND YOUR HORIZON

WWW.HORIZON-TECH.TN

HORIZON SCHOOL OF DIGITAL TECHNOLOGIES