Data Pre-Processing

Prepared by: *Ted Lee*

Prepared on: *2025-04-09*

Data Pre-Processing (1-2)

Name	Gender	Age
Ted	Male	18
Ted	M	18
Mandy	F	-

1. **Data Cleaning**
   - Handling missing values (imputation, removal)
     - ❖ `from sklearn.impute import SimpleImputer`
     - ❖ `imputer = SimpleImputer(strategy='mean')` `# strategy='most_frequent'`
   - Removing duplicates
   - Correcting inconsistencies (e.g., typos in categorical variables)

2. **Data Transformation**
   - Normalization
     - Scaling numerical data to a specific range [0;1] using X' = (X – Xmin) / (Xmax – Xmin)
     - `from sklearn.preprocessing import MinMaxScaler`
     - `scaler = MinMaxScaler()`
     - `normalized_data = scaler.fit_transform(data)`
   - Standardization
     - Scaling data to have a mean of 0 and a standard deviation of 1
     - [-3;+3] X' = (X - μ) / σ
     - ❖ `from sklearn.preprocessing import StandardScaler`
     - ❖ `scaler = StandardScaler()`
     - ❖ `standardized_data = scaler.fit_transform(data)`
   - Log Transformation (for skewed numerical data).
     - ❖ `log_data = np.log(data)`

← Use this in all situations

Data Pre-Processing (3-4)

3. Encoding Categorical Variables

- One-Hot Encoding
  - Converting independent categorical variables into binary vectors
  - Creating dummy variables
  - ❖ `from sklearn.preprocessing import OneHotEncoder`
  - ❖ `onehot_encoder = OneHotEncoder(sparse=False)`
  - ❖ `onehot_encoded = onehot_encoder.fit_transform(categories)`

City	Class
HK	A
Macau	B
PH	C

→

City	Macau	PH	Class
1	0	0	0
0	1	0	1
0	0	1	2

- Label Encoding
  - Converting dependent categories into numerical values
  - ❖ `from sklearn.preprocessing import LabelEncoder`
  - ❖ `label_encoder = LabelEncoder()`
  - ❖ `encoded_labels = label_encoder.fit_transform(categories)`

4. Data Reduction – (A) Feature selection

Selecting a subset of relevant features to reduce overfitting, decrease training time, and improve model interpretability.

I. Filter Methods

- Calculate the **correlation coefficient** between **each feature** and the **target variable**
- E.g. Pearson correlation to predict house prices
- ❖ `correlation = diff[['Location', 'Size', 'Age', 'House_Price']].corr()`

Data Pre-Processing (4)

4. Data Reduction – (A) Feature selection

II. Wrapper Methods
- Example - **Recursive Feature Elimination** (RFE):
  - Use Case: Using a linear regression model, you start with all features, remove the one with the lowest absolute **coefficient**, re-fit the model, and repeat the process.

```
from sklearn.feature_selection import RFE
rfe = RFE(estimator=LinearRegression(), n_features_to_select=3)
rfe.fit(X_train, y_train)
selected_features = X.columns[rfe.support_]
```

→ **It returns 'Location', 'Size' and 'Age'.**

III. Embedded Methods
- Example - **Lasso Regression**:
  - Use Case: In a dataset with many features, applying Lasso regression could reveal which features are most significant by setting less important feature coefficients to zero.

```
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error, r2_score
lasso = Lasso(alpha=0.1)
lasso.fit(X_train, y_train)
y_pred = lasso.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
coefficients = pd.Series(lasso.coef_, index=X.columns)
```

Data Pre-Processing (4)

4. Data Reduction – (A) Feature selection

IV. Tree-Based Methods

- Example - **Feature Importance** from Decision Trees:
  - Use Case: Fit a Random Forest to a dataset and examine the **feature importance scores**, then select the top features based on those scores.

```
from sklearn.ensemble import RandomForestRegressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
importances = rf_model.feature_importances_
importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': importances})
importance_df = importance_df.sort_values(by='Importance', ascending=False)
```

V. Univariate Selection

- Example - **Chi-Squared Test**:
  - Use Case: In a dataset with several categorical features, you could apply the Chi-squared test to determine which features are significantly associated with the target class.

```
from scipy import stats
data = {'Gender': ['Male', 'Male', 'Female', 'Female', 'Male', 'Female', 'Male', 'Female'],
'Preference': ['Like', 'Dislike', 'Like', 'Dislike', 'Like', 'Like', 'Dislike', 'Dislike']}
df = pd.DataFrame(data)
contingency_table = pd.crosstab(df['Gender'], df['Preference'])
chi2_stats, p_val, deg_of_freedom, expected_freq = stats.chi2_contingency(contingency_table)
if p_val < alpha:
    print("Reject the null hypothesis: Significant association between Gender and Preference.")
```

Data Pre-Processing (4)

4. **Data Reduction** – (B) Dimensionality reduction techniques

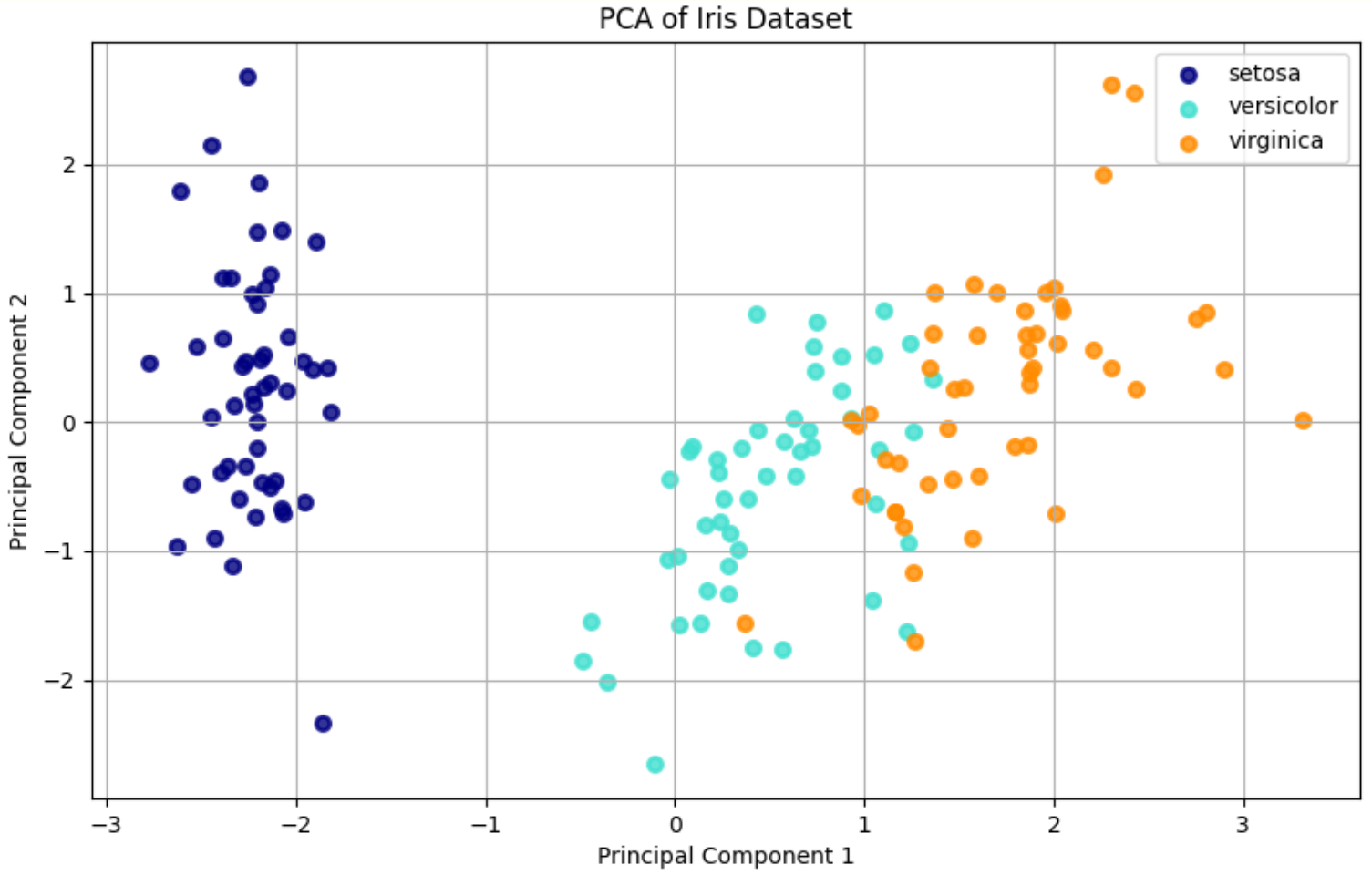
   i. **Principal Component Analysis (PCA)**
   - **Overview**: PCA is a linear dimensionality reduction technique that transforms the original features into a new set of uncorrelated features called principal components. These components capture the maximum variance in the data.
   - **How it Works**:
     - PCA computes the eigenvectors and eigenvalues of the covariance matrix of the data.
     - The eigenvectors (principal components) are ordered by their corresponding eigenvalues (variance captured).
     - The top k eigenvectors are selected to form a new feature space, where k is less than the original number of features.
   - **Applications**: PCA is widely used in data preprocessing, for noise reduction, and to visualize high-dimensional data in two or three dimensions.

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Standardize the datascaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA and reduce to 2 dimensions
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Create a DataFrame for easy plotting
df_pca = pd.DataFrame(data=X_pca, columns=['Principal Component 1', 'Principal Component 2’])
df_pca['Target'] = y
```

Data Pre-Processing (4)

4. **Data Reduction** – (B) Dimensionality reduction techniques

   i. **Principal Component Analysis (PCA)**



Data Pre-Processing (4)

4. **Data Reduction** – (B) Dimensionality reduction techniques

   ii. **t-Distributed Stochastic Neighbor Embedding (t-SNE)**

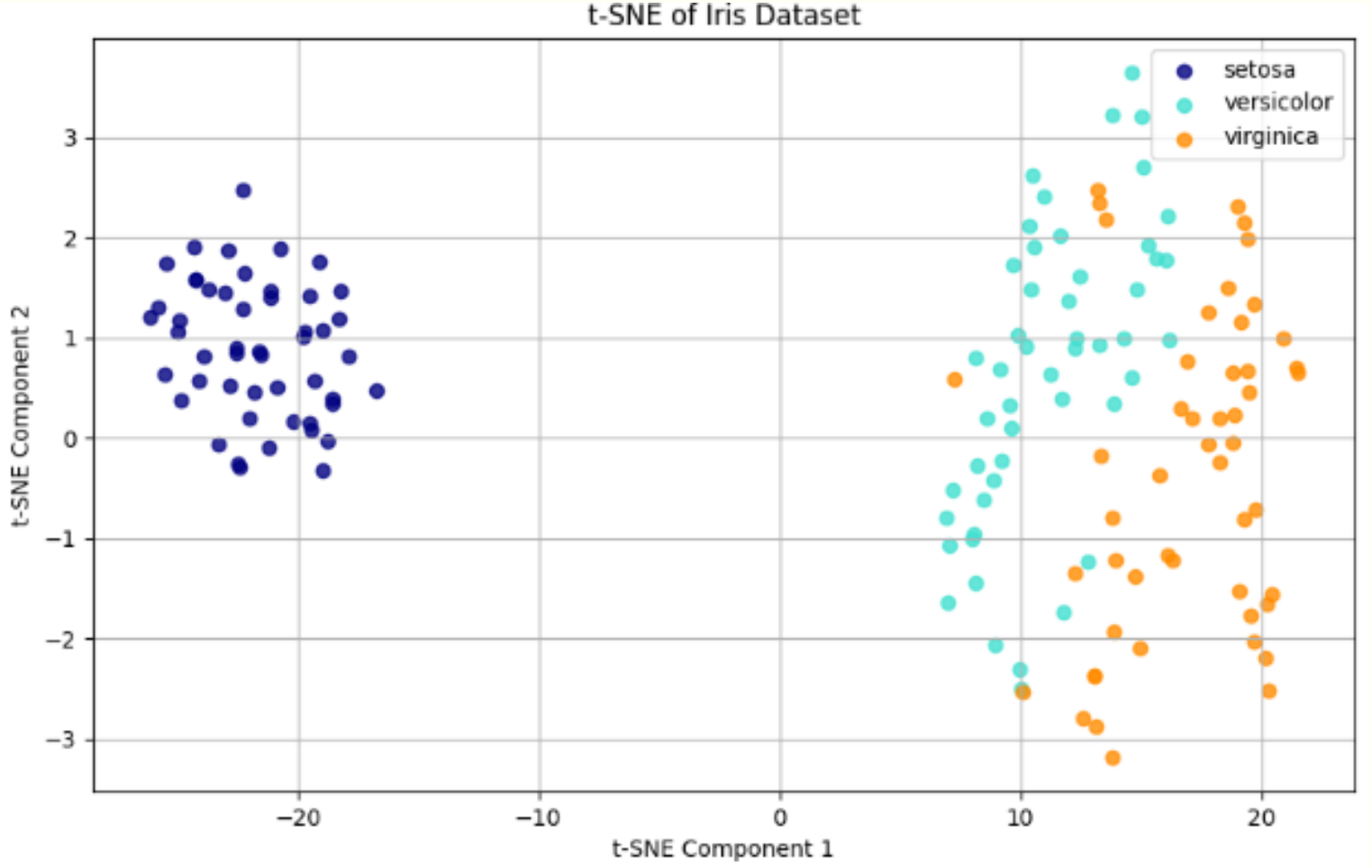
   - **Overview**: t-SNE is a non-linear dimensionality reduction technique particularly well-suited for visualization of high-dimensional data in low-dimensional spaces (usually 2D or 3D).
   - **How it Works**:
     - t-SNE converts similarities (distances) between points in a high-dimensional space into probabilities, creating a probability distribution for pairwise similarities.
     - It then finds a lower-dimensional representation that reflects these probabilities as closely as possible.
     - The technique emphasizes preserving local structures, meaning that points that are close in high-dimensional space remain close in low-dimensional representation.
   - **Applications**: t-SNE is commonly used for visualizing complex datasets, such as word embeddings, image data, or clustering results.

```
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply t-SNE
tsne = TSNE(n_components=2, random_state=42)
X_tsne = tsne.fit_transform(X_scaled)

tsne_df = pd.DataFrame(data=X_tsne, columns=['t-SNE Component 1', 't-SNE Component 2'])
tsne_df['Target'] = y
```

Data Pre-Processing (4)

4. Data Reduction – (B) Dimensionality reduction techniques

ii. t-Distributed Stochastic Neighbor Embedding (t-SNE)



Data Pre-Processing (4)

4. **Data Reduction** – (B) Dimensionality reduction techniques

   iii. **Linear Discriminant Analysis (LDA)**

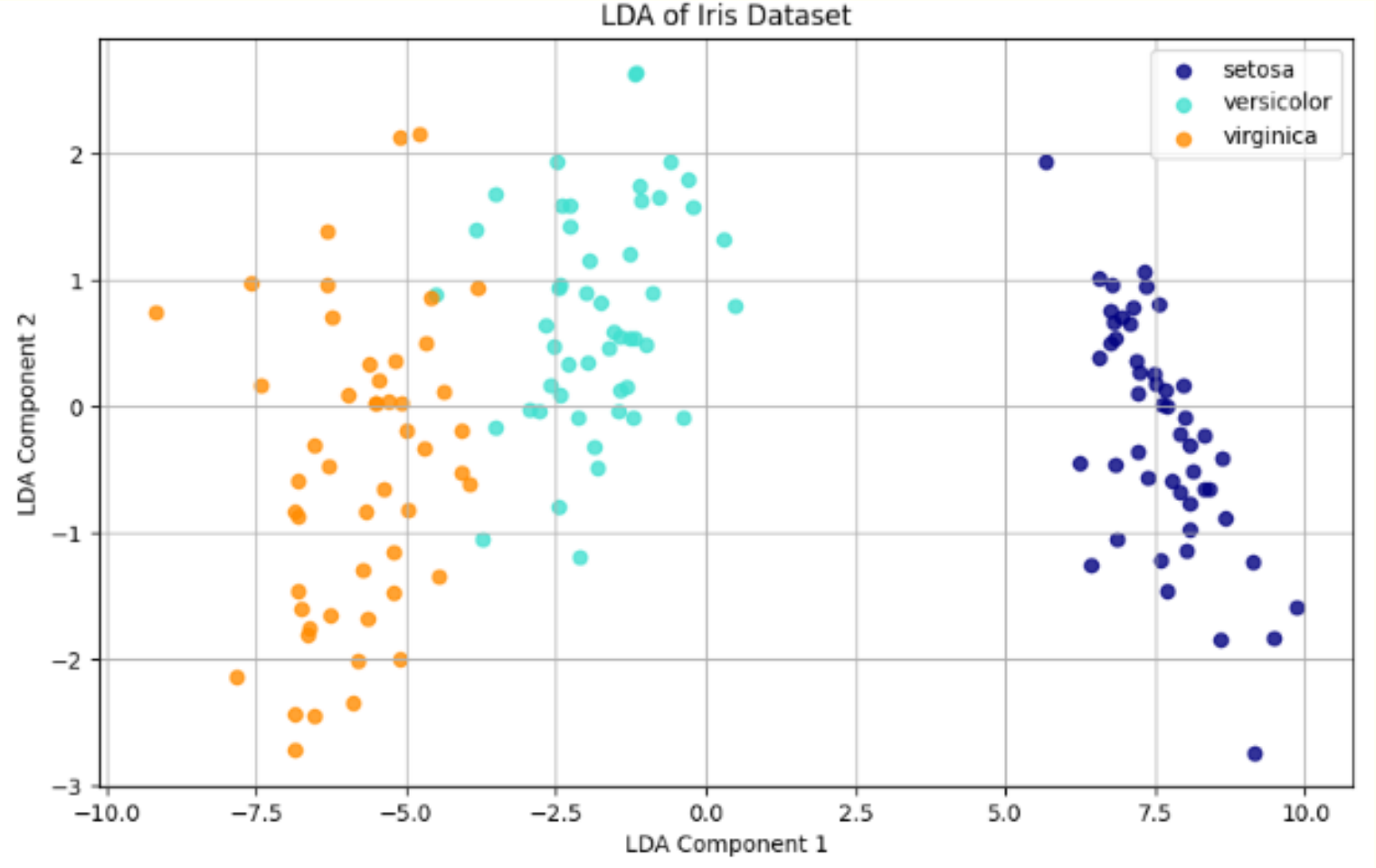
   - **Overview**: LDA is a supervised dimensionality reduction technique used mainly for classification. It aims to maximize the separation between multiple classes.
   - **How it Works**:
     - LDA computes the mean vectors of each class and the overall mean.
     - The within-class and between-class scatter matrices are calculated to find the linear combinations of features that best separate the classes.
     - The resulting linear discriminants can be used for both dimensionality reduction and as classifiers.
   - **Applications**: LDA is mainly used in scenarios where class labels are available and is effective in reducing the dimensionality of data for classification purposes.

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply LDA
lda = LinearDiscriminantAnalysis(n_components=2)
X_lda = lda.fit_transform(X_scaled, y)

lda_df = pd.DataFrame(data=X_lda, columns=['LDA Component 1', 'LDA Component 2'])
lda_df['Target'] = y
```

Data Pre-Processing (4)

4. **Data Reduction** – (B) Dimensionality reduction techniques

   iii. **Linear Discriminant Analysis (LDA)**



Data Pre-Processing (4)

4. **Data Reduction** – (B) Dimensionality reduction techniques

   iv. **Autoencoders**
   - **Overview**: Autoencoders are a type of neural network used for unsupervised learning. They learn a compressed representation (encoding) of the input data.
   - **How it Works**:
     - An autoencoder consists of two parts: an encoder that compresses the input into a lower-dimensional representation, and a decoder that reconstructs the input from this representation.
     - The network is trained to minimize the reconstruction error (the difference between the input and output).
   - **Applications**: Autoencoders are used in various tasks, including noise reduction, anomaly detection, and extracting meaningful features from data.

A bit completed

Data Pre-Processing (4)

4. **Data Reduction** – (B) Dimensionality reduction techniques

   v. **Isomap**
   - **Overview**: Isomap is a non-linear dimensionality reduction technique that preserves the global geometric structure of the data.
   - **How it Works**:
     - Isomap computes the geodesic distances between data points on a manifold (using nearest neighbors).
     - It then applies classical multidimensional scaling (MDS) to embed this distance matrix into a lower-dimensional space.
   - **Applications**: Isomap is effective when the data lies on a lower-dimensional manifold and is useful in various fields, including pattern recognition and image processing.

```
from sklearn.manifold import Isomap
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply Isomap
isomap = Isomap(n_components=2)
X_isomap = isomap.fit_transform(X_scaled)

isomap_df = pd.DataFrame(data=X_isomap, columns=['Isomap Component 1', 'Isomap Component 2'])
isomap_df['Target'] = y
```

Data Pre-Processing (4-6)

4. **Data Reduction** – (C) Sampling (reducing the size of the dataset)

5. **Data Integration**:
   - Combining data from different sources to create a unified dataset.

```
online_sales = pd.read_csv('online_sales.csv')
physical_sales = pd.read_csv('physical_sales.csv')
integrated_sales = pd.concat([online_sales, physical_sales], ignore_index=True)
```

6. **Data Discretization**:
   - Converting continuous variables into categorical ones.

```
data = { 'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva', 'Frank', 'Grace', 'Hannah'], 'Age': [4, 15, 23, 45, 64, 12, 34, 89] }
df = pd.DataFrame(data)

# Define the age bins and corresponding labels
bins = [0, 12, 19, 64, 100]
labels = ['Child', 'Teenager', 'Adult', 'Senior']

# Use pd.cut to discretize the 'Age' column
df['AgeGroup'] = pd.cut(df['Age'], bins=bins, labels=labels, right=True)
```

Data Pre-Processing (7-9)

7. **Feature Engineering**:
   - Creating new features based on existing data to improve model performance.

8. **Data Formatting**:
   - Converting data into a suitable format for analysis (e.g., converting date formats).

9. **Outlier Detection and Treatment**:
   - Identifying outliers and deciding whether to remove, replace, or keep them.

   Statistical Tests

   i. Z-Score Method: Identifies outliers based on standard deviations from the mean. A common threshold is a z-score of ±3.

```
from scipy import stats
z_scores = np.abs(stats.zscore(df['Values']))
outliers_z = df[z_scores > 3]
```

   ii. IQR Method: Uses the interquartile range (IQR). Data points that fall below Q1 - 1.5 * IQR or above Q3 + 1.5 * IQR are considered outliers.

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

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers_iqr = df[(df['Values'] < lower_bound) | (df['Values'] > upper_bound)]
```

Data Pre-Processing (10)

10. Text Pre-processing (for text data):

```
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize

text_data = ["Hello, world! This is an example text.", "Text pre-processing is essential for NLP tasks."]

# Initialize the stemmer and stopwords
stemmer = PorterStemmer()
stop_words = set(stopwords.words('english'))

# Function for text pre-processing
def preprocess(text):
    text = text.lower()                                          # 1. Lowercasing
    text = text.translate(str.maketrans("", "", string.punctuation))   # 2. Removing punctuation
    tokens = word_tokenize(text)                                 # 3. Tokenization → Split the sentence into words
    tokens = [word for word in tokens if word not in stop_words] # 4. Removing stop words → E.g. 'the', 'is'
    tokens = [stemmer.stem(word) for word in tokens]             # 5. Stemming → Change "Running" to "Run"
    return ' '.join(tokens)                                      # Rejoining tokens

cleaned_data = [preprocess(text) for text in text_data]          # Apply pre-processing to the text data

for original, cleaned in zip(text_data, cleaned_data):
    print(f"Original: {original}\nCleaned: {cleaned}\n")         # Display the cleaned text
```

~ *END* ~