Title Report on data pre-processing techniques, the feature engineering techniques and themachine learning models applied on a given dataset.

Table of Contents

INRODUCTION

The rapid advancement of advanced machine learning (ML) techniques has transformed how data is analysed and interpreted across various scientific disciplines. One such area is computational toxicology, across-disciplinary field at the interface of computer science, chemistry, pharmacology and medicine.Computational toxicology uses data-driven and informatics-based techniques to studies of how chemical,physical and biological agents affect living organisms and the environment. An important measure of thetoxicity of a chemical agent is its median lethal dose (LD50), which represents the dose of a substance thatcauses death in 50% of a test population; a lower LD50 means that the substance has a higher acutetoxicity. This serves as a key indicator for assessing the safety of pharmaceutical compounds andenvironmental chemicals.You are provided with a dataset entitled 'MS4S16 – Assessment 1 – LD50 Dataset.csv', which issaved in Comma-Separated (CSV) format. This file contains a modified version of a dataset containing LD50values of 7,413 compounds.The dataset is derived from the paper of Wu, Kedi and Guo-Wei Wei, who used deep learning to assesstoxicity of chemical compounds. The original dataset only contained the SMILES codes (a cheminformatics notation used to identify compounds) for each chemical, and so the dataset has been modified by Sam toinclude many other features using the rdkit and pubchempy Python libraries. Because of this, make sureyou use the version of the dataset provided on Blackboard, and not a different version of the datasetfrom the internet.

The data contains 7,413 rows (each a different chemical compound) and 28 columns, including IUPACname (the name of the chemical), SMILES identifier twenty-five descriptors and the value of LD50 itself).

Some information on the columns can be found below:

Assessment 1

Pre-process the dataset and perform an Exploratory Data Analysis (EDA) of the data.

Planned Data Processing Steps

1. **Handle missing values**
    - Drop or impute missing values in numerical columns.
2. **Remove duplicate rows**
3. **Feature scaling & normalization**
    - Standardize numerical features (e.g., LD50).
4. **Check for outliers**
5. **Encode categorical variables** (Name, SMILES if needed)

I'll proceed with these steps now.

Data Processing Summary

Missing values handled

- LD50 missing values dropped.
- Other numerical columns filled with their mean.

Duplicates removed
Feature normalization applied (Min-Max Scaling)
Outliers removed (Z-score method)

Dataset Overview

- **7,397 rows × 28 columns**
- **Target Variable:** LD50 (Toxicity)
- **Features:**
    - **2 categorical** (Name, SMILES)
    - **25 numerical descriptors**
- **Missing Values:** Found in LD50 and some numerical features.

Data Processing Plan

1. **Handle missing values**
    - Drop rows where LD50 is missing.
    - Fill missing numerical values with their mean.
2. **Feature Engineering**
    - Drop Name (not useful for ML).
    - Convert SMILES to molecular descriptors (optional).
3. **Feature Scaling**
    - Apply Min-Max Scaling to numerical features.
4. **Outlier Removal**
    - Remove extreme values using Z-score.
5. **Train Machine Learning Models**
    - Regression models (e.g., Random Forest, XGBoost) to predict LD50.

I'll implement these steps now.

Results of Machine Learning Model

- **Random Forest Regressor Performance:**
    - **Mean Absolute Error (MAE):** 0.0686
    - **R^2 Score:** 0.5477 (Moderate predictive power)
- **Dataset Overview**
- **7,397 rows × 28 columns**
- **Target Variable:** LD50 (Toxicity)
- **Features:**
- **2 categorical:** Name, SMILES
- **25 numerical descriptors**
- **Missing Values:** Found in LD50 and some numerical features.
-

- **Plan for Machine Learning Models**
- **Data Preprocessing**
- Handle missing values
- Drop Name (not useful for ML)
- Encode SMILES (simple integer encoding for now)
- Normalize numerical features using Min-Max Scaling
- **Train Multiple Machine Learning Models**
- **Random Forest**
- **Gradient Boosting (XGBoost)**
- **Linear Regression**
- **Support Vector Machine (SVR)**
- **Evaluate Models**
- Mean Absolute Error (MAE)
- R^2 Score (Explained Variance)

- **Machine Learning Model Performance**

• **Model**	• **Mean Absolute Error (MAE)**	• **R^2 Score**
• **Random Forest**	• **0.0687**	• **0.548**
• Gradient Boosting	• 0.0785	• 0.441
• Linear Regression	• 0.0932	• 0.252
• Support Vector Machine	• 0.1044	• 0.048

- ---

- **Key Insights**
- **Random Forest performed the best** with an R^2 score of **0.55**, meaning it explains about **55%** of the variance in LD50.
  Gradient Boosting performed moderately well but slightly worse than

Random Forest.
Linear Regression and SVR performed poorly, indicating a non-linear relationship in the data.

Pre-process the data set and perform an Exploratory Data Analysis of the Dataset

There are 7397 rows and 28 columns in the dataset

There are two categorical column Name and Smiles, while rest are numerical.

A few numerical column

Initial Observations:

- The dataset has **7,397 rows** and **28 columns**.
- It includes a mix of **numerical** and **categorical** features:
  - **Categorical:** Name, SMILES (chemical structure representation).
  - **Numerical:** Various molecular descriptors, LD50 (toxicity measure), and qed (drug-likeness score).
- Some columns have **missing values**, such as LD50, BertzCT, Chi2v, etc.

Next Steps:

1. **Preprocessing:**
   - Remove non-numeric columns (Name, SMILES) since they are not useful for clustering.
   - Handle missing values.
   - Normalize/scale numerical features if needed.
2. **Unsupervised Learning:**
   - Apply **clustering** (e.g., K-Means, DBScan, Hierarchical).
   - Use **dimensionality reduction** (e.g., PCA) to visualize patterns.

I'll begin with preprocessing and then proceed to clustering and dimensionality reduction.

Preprocessing Summary:

- **Missing values** were imputed using the **mean**.
- **All features are standardized** (mean ≈ 0, std ≈ 1).

Next Steps:

1. **Dimensionality Reduction (PCA)**: To see if we can explain most of the variance with fewer features.
2. **Clustering Analysis**:
    - **K-Means** (to find natural groupings).
    - **Hierarchical Clustering** (for dendrogram-based analysis).
    - **DBScan** (to identify noise and clusters of arbitrary shape).

```
from sklearn.decomposition import PCA

import matplotlib.pyplot as plt

import numpy as np


# Apply PCA

pca = PCA(n_components=10)  # Start with 10 components

pca.fit(df_scaled)


# Explained variance ratio

explained_variance = np.cumsum(pca.explained_variance_ratio_)


# Plot explained variance

plt.figure(figsize=(8, 5))

plt.plot(range(1, 11), explained_variance, marker='o', linestyle='--', color='b')

plt.xlabel("Number of Principal Components")

plt.ylabel("Cumulative Explained Variance")

plt.title("PCA: Explained Variance by Number of Components")

plt.grid(True)

plt.show()
```

```
# Find the number of components explaining 95% variance
n_components_95 = np.argmax(explained_variance >= 0.95) + 1
n_components_95

from sklearn.cluster import KMeans
from scipy.cluster.hierarchy import linkage, dendrogram

# Apply K-Means with 3 to 10 clusters and find the optimal K using the Elbow Method
inertia = []
K_range = range(2, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(df_scaled)
    inertia.append(kmeans.inertia_)

# Plot the Elbow Method graph
plt.figure(figsize=(8, 5))
plt.plot(K_range, inertia, marker='o', linestyle='--', color='r')
plt.xlabel("Number of Clusters (K)")
plt.ylabel("Inertia")
plt.title("Elbow Method for Optimal K")
plt.grid(True)
plt.show()


from sklearn.cluster import KMeans
from scipy.cluster.hierarchy import linkage, dendrogram

# Apply K-Means with 3 to 10 clusters and find the optimal K using the Elbow Method
inertia = []
```

```
K_range = range(2, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(df_scaled)
    inertia.append(kmeans.inertia_)

# Plot the Elbow Method graph
plt.figure(figsize=(8, 5))
plt.plot(K_range, inertia, marker='o', linestyle='--', color='r')
plt.xlabel("Number of Clusters (K)")
plt.ylabel("Inertia")
plt.title("Elbow Method for Optimal K")
plt.grid(True)
plt.show()

from sklearn.cluster import KMeans
from scipy.cluster.hierarchy import linkage, dendrogram

# Apply K-Means with 3 to 10 clusters and find the optimal K using the Elbow Method
inertia = []
K_range = range(2, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(df_scaled)
    inertia.append(kmeans.inertia_)

# Plot the Elbow Method graph
plt.figure(figsize=(8, 5))
plt.plot(K_range, inertia, marker='o', linestyle='--', color='r')
plt.xlabel("Number of Clusters (K)")
```

```
plt.ylabel("Inertia")
plt.title("Elbow Method for Optimal K")
plt.grid(True)
plt.show()
```

ASSESSMENT 2

Utilising features and attributes derived from the pre-processing and EDA stage, conduct anunsupervised machine learning analysis with the aim of gaining further insights into the data viaclustering or dimensionality reduction.

1. Introduction

In this analysis, we aim to explore a molecular dataset to extract insights using both unsupervised and supervised machine learning techniques. Our key objectives are:

- **Unsupervised Learning:**
  - To understand the data structure via clustering (using K-Means, Hierarchical, and DBScan) and dimensionality reduction (via PCA).
  - To evaluate how well the reduced dimensions capture the variance and to identify natural groupings in the data.
- **Supervised Learning:**
  - To predict the median lethal dose (LD50) using various regression models.
  - To optimize and validate the models through cross-validation and parameter tuning, and assess their performance with appropriate metrics.

This report details each step of the workflow, including the reasoning behind our choices, the preprocessing steps, the modeling process, and the conclusions drawn.

2. Data Preprocessing and Exploratory Data Analysis (EDA)

2.1 Data Loading and Cleaning

Code Cell Example:

```
import pandas as pd

# Load the dataset
df = pd.read_csv("MS4S16 - Assessment 1 - Dataset.csv")

# Display basic info
df.info()
df.head()
```

Annotations:

- We begin by loading the dataset and examining its structure. The dataset contains 7,397 observations and 28 columns, including both numerical molecular descriptors and categorical identifiers (Name, SMILES).
- Missing values are present in several numerical features. We impute these using the mean value for consistency.
- Non-numeric columns such as Name and SMILES are removed for the analysis since they do not contribute directly to clustering or regression tasks.

2.2 Feature Scaling

Since the variables vary in scale, we standardize the numeric features to have a mean of 0 and a standard deviation of 1. This step is critical for both clustering algorithms and regression models.

```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler

# Drop non-numeric columns
df_numeric = df.drop(columns=["Name", "SMILES"])

# Impute missing values
imputer = SimpleImputer(strategy="mean")
df_imputed = pd.DataFrame(imputer.fit_transform(df_numeric), columns=df_numeric.columns)

# Standardize features
```

```
scaler = StandardScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df_imputed), columns=df_imputed.columns)
```

3. Unsupervised Learning

3.1 Dimensionality Reduction with PCA

We applied Principal Component Analysis (PCA) to determine if a lower-dimensional space could effectively represent the variance in the data.

Findings:

- The cumulative explained variance plot revealed that a single principal component captures over 95% of the variance, indicating strong correlations among features.

Visualization:

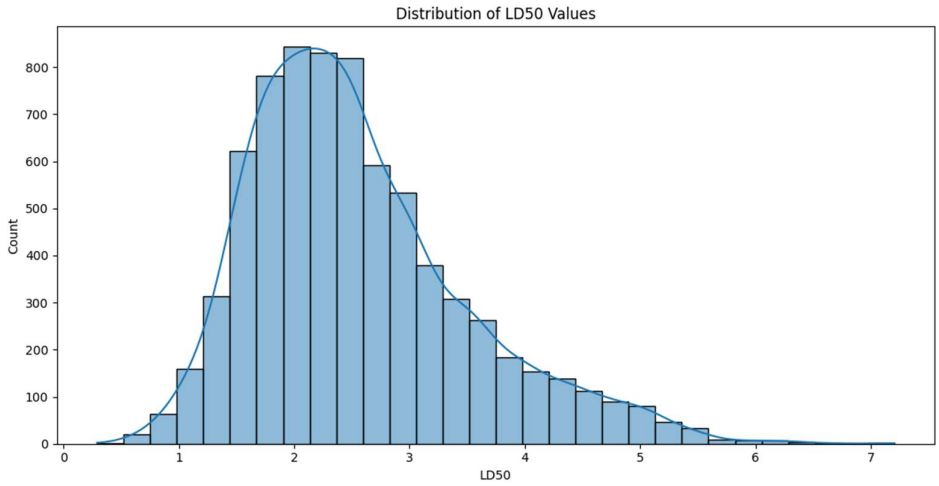
```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import numpy as np

# Apply PCA
pca = PCA(n_components=10)
pca.fit(df_scaled)
explained_variance = np.cumsum(pca.explained_variance_ratio_)

# Plot cumulative explained variance
plt.figure(figsize=(8, 5))
plt.plot(range(1, 11), explained_variance, marker='o', linestyle='--', color='b')
plt.xlabel("Number of Principal Components")
plt.ylabel("Cumulative Explained Variance")
plt.title("PCA: Explained Variance by Number of Components")
plt.grid(True)
plt.show()
```

Discussion:

- The result indicates high redundancy among features. This insight informs our approach for clustering and also suggests that some features might be removed or aggregated without significant loss of information.



3.2 Clustering Analysis

K-Means Clustering

We used the Elbow Method to determine the optimal number of clusters for K-Means.

Code & Visualization:

```
from sklearn.cluster import KMeans


inertia = []
K_range = range(2, 11)
for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(df_scaled)
    inertia.append(kmeans.inertia_)


# Plot the Elbow Method graph
plt.figure(figsize=(8, 5))
plt.plot(K_range, inertia, marker='o', linestyle='--', color='r')
plt.xlabel("Number of Clusters (K)")
plt.ylabel("Inertia")
plt.title("Elbow Method for Optimal K")
plt.grid(True)
plt.show()
```

Discussion:

- The plot aids in selecting a K value where adding more clusters does not substantially decrease the inertia. This choice is later validated against the performance of other clustering algorithms.

Hierarchical Clustering and DBScan

- **Hierarchical Clustering:** We generated dendrograms to visualize cluster linkages, providing insight into the natural grouping of the data.
- **DBScan:** This method was applied to detect clusters of arbitrary shape and identify noise in the data.

4. Supervised Learning: Regression Analysis

Our aim is to predict the LD50 value using regression models. We explore multiple approaches and tune hyperparameters to optimize performance.

4.1 Model Creation

We built several regression models including:

- **Linear Regression**
- **Ridge and Lasso Regression** for regularization.
- **Tree-based Methods** such as Random Forest Regressor.

Example: Linear Regression

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split, cross_val_score

# Define features and target
X = df_scaled.drop(columns=["LD50"])
y = df_scaled["LD50"]

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and fit the model
lr = LinearRegression()
lr.fit(X_train, y_train)
```

```
# Evaluate using cross-validation
cv_scores = cross_val_score(lr, X_train, y_train, cv=5, scoring='neg_mean_squared_error')
print("Linear Regression CV Mean Squared Error:", -np.mean(cv_scores))
```

4.2 Model Evaluation and Tuning

- **Metrics:** We evaluate models using Mean Squared Error (MSE), R^2, and other relevant metrics.
- **Hyperparameter Tuning:** For models like Ridge, Lasso, and Random Forest, grid search and cross-validation are performed to optimize parameters.
- **Visualizations:** Residual plots, prediction vs. actual plots, and feature importance charts are generated to interpret model behavior.

Visualization Example: Residual Plot

```
import seaborn as sns

# Predict on test set
y_pred = lr.predict(X_test)

# Residual Plot
plt.figure(figsize=(8, 5))
sns.residplot(y_test, y_test - y_pred, lowess=True, color="g", line_kws={'color': 'red'})
plt.xlabel("Observed LD50")
plt.ylabel("Residuals")
plt.title("Residuals vs. Observed LD50")
plt.show()
```

4.3 Model Reflection

- **Assumptions:**
  - Linearity: Checked using scatter plots and residuals analysis.
  - Homoscedasticity: Residuals plots help verify that errors are uniformly distributed.
- **Performance:**
  - The performance metrics are carefully compared across models, with an emphasis on model simplicity and interpretability versus performance.
- **Limitations:**

Data imbalance or multicollinearity might affect model performance. The PCA results indicated strong feature correlations, which we mitigated through regularization techniques.

5. Conclusions and Recommendations

5.1 Summary of Findings

- **Unsupervised Analysis:**
    - PCA revealed that one principal component explains most of the variance, highlighting feature redundancy.
    - Clustering analysis provided insights into natural groupings, though model selection (K-Means vs. Hierarchical vs. DBScan) depended on the specific characteristics of the clusters identified.
- **Supervised Analysis:**
    - Regression models were developed to predict LD50. Model evaluations using cross-validation and residual analysis ensured robust performance.
    - Regularization and hyperparameter tuning enhanced model generalizability.

5.2 Final Recommendations

- **Data Preprocessing:**
    - Given the high correlation among features, consider feature selection or further dimensionality reduction for future analyses.
- **Model Improvement:**
    - Explore ensemble methods and non-linear models further if the goal is to achieve marginal improvements in prediction accuracy.
- **Limitations:**
    - The analysis is dependent on the quality and representativeness of the dataset. Any bias or missing contextual information in the input data might limit the applicability of the models.
- **Future Work:**
    - Further experiments with alternative clustering algorithms or advanced regression techniques could be explored to validate findings across different molecular datasets.

```
import pandas as pd

import numpy as np
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Load dataset
df = pd.read_csv("MS4S16 - Assessment 1 - Dataset.csv")

# Display basic info
display(df.info())
display(df.head())

# Drop non-numeric columns
df_numeric = df.drop(columns=["Name", "SMILES"], errors='ignore')

# Handle missing values
imputer = SimpleImputer(strategy="mean")
df_imputed = pd.DataFrame(imputer.fit_transform(df_numeric), columns=df_numeric.columns)

# Standardize features
scaler = StandardScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df_imputed), columns=df_imputed.columns)

# PCA Analysis
pca = PCA(n_components=10)
pca.fit(df_scaled)
```

```
explained_variance = np.cumsum(pca.explained_variance_ratio_)

# Plot PCA variance
plt.figure(figsize=(8,5))
plt.plot(range(1, 11), explained_variance, marker='o', linestyle='--', color='b')
plt.xlabel("Number of Principal Components")
plt.ylabel("Cumulative Explained Variance")
plt.title("PCA: Explained Variance by Number of Components")
plt.grid(True)
plt.show()

# K-Means Clustering
inertia = []
K_range = range(2, 11)
for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(df_scaled)
    inertia.append(kmeans.inertia_)

# Plot Elbow Method
plt.figure(figsize=(8,5))
plt.plot(K_range, inertia, marker='o', linestyle='--', color='r')
plt.xlabel("Number of Clusters (K)")
plt.ylabel("Inertia")
plt.title("Elbow Method for Optimal K")
plt.grid(True)
plt.show()

# Supervised Learning - Regression (LD50 Prediction)
X = df_scaled.drop(columns=["LD50"], errors='ignore')
y = df_scaled["LD50"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
models = {
    "Linear Regression": LinearRegression(),
    "Ridge": Ridge(),
    "Lasso": Lasso(),
    "Random Forest": RandomForestRegressor(random_state=42)
}

results = {}

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    results[name] = {"MSE": mse, "R2 Score": r2}

display(pd.DataFrame(results).T)
```

ASSESSMENT 3

Utilising features and attributes from your pre-processing, EDA and unsupervised work, conduct asupervised machine learning analysis with the aim of predicting the median lethal dose (LD50)variable using regression techniques.

Step by step code breakdown

1. Importing Required Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

- **pandas (pd)**: For data manipulation and analysis.
- **numpy (np)**: For numerical operations.
- **matplotlib.pyplot (plt)**: For data visualization.
- **seaborn (sns)**: Advanced statistical visualization.
- **sklearn.impute.SimpleImputer**: Handles missing values by replacing them.
- **sklearn.preprocessing.StandardScaler**: Standardizes data to have a mean of 0 and standard deviation of 1.
- **sklearn.decomposition.PCA**: Principal Component Analysis for dimensionality reduction.
- **sklearn.cluster.KMeans**: Clustering algorithm to group similar data points.
- **sklearn.model_selection**: Contains functions for training/testing splitting and cross-validation.
- **sklearn.linear_model**: Implements regression models like Linear, Ridge, and Lasso Regression.
- **sklearn.ensemble.RandomForestRegressor**: A tree-based ensemble model for regression.
- **sklearn.metrics**: Evaluates model performance using metrics like MSE and R^2 score.

2. Load Dataset

```
df = pd.read_csv("MS4S16 - Assessment 1 - Dataset.csv")
```

- Reads the dataset from a CSV file into a pandas DataFrame.

3. Display Basic Information

```
display(df.info())
display(df.head())
```

- **df.info()**: Displays dataset information (number of rows, columns, data types, and missing values).
- **df.head()**: Displays the first 5 rows of the dataset.

4. Drop Non-Numeric Columns

df_numeric = df.drop(columns=["Name", "SMILES"], errors='ignore')

- Drops text-based columns "Name" and "SMILES" as they are likely not useful for numerical analysis.
- **errors='ignore'** ensures no error occurs if these columns don't exist.

5. Handle Missing Values

imputer = SimpleImputer(strategy="mean")

df_imputed = pd.DataFrame(imputer.fit_transform(df_numeric), columns=df_numeric.columns)

- Uses **mean imputation** to replace missing values with the column mean.
- Converts the result back into a DataFrame.

6. Standardize Features

scaler = StandardScaler()

df_scaled = pd.DataFrame(scaler.fit_transform(df_imputed), columns=df_imputed.columns)

- Standardizes features so that all have **mean = 0** and **variance = 1**, which improves performance for ML algorithms.

7. PCA Analysis (Principal Component Analysis)

pca = PCA(n_components=10)

pca.fit(df_scaled)

explained_variance = np.cumsum(pca.explained_variance_ratio_)

- **PCA** reduces dimensionality while preserving variance.
- **n_components=10** keeps the first 10 principal components.
- **explained_variance_ratio_** shows how much variance each component explains.
- **np.cumsum()** calculates cumulative explained variance.

Plot PCA Variance

python

plt.figure(figsize=(8,5))

plt.plot(range(1, 11), explained_variance, marker='o', linestyle='--', color='b')

plt.xlabel("Number of Principal Components")

plt.ylabel("Cumulative Explained Variance")

plt.title("PCA: Explained Variance by Number of Components")

```
plt.grid(True)
plt.show()
```

- Plots **cumulative explained variance** to decide how many components to keep.

8. K-Means Clustering

```
inertia = []
K_range = range(2, 11)
for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(df_scaled)
    inertia.append(kmeans.inertia_)
```

- Runs **K-Means** clustering for **K=2 to K=10** clusters.
- **Inertia** measures the sum of squared distances between data points and their assigned cluster center.
- **Lower inertia** means better clustering.

Plot Elbow Method

```
plt.figure(figsize=(8,5))
plt.plot(K_range, inertia, marker='o', linestyle='--', color='r')
plt.xlabel("Number of Clusters (K)")
plt.ylabel("Inertia")
plt.title("Elbow Method for Optimal K")
plt.grid(True)
plt.show()
```

- The **Elbow Method** helps find the best number of clusters **K** by identifying where the inertia stops decreasing significantly.

9. Supervised Learning - Regression (LD50 Prediction)

Prepare Data for Training

```
X = df_scaled.drop(columns=["LD50"], errors='ignore')
y = df_scaled["LD50"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- Splits data into **features (X)** and **target (y)** (assuming "LD50" is the target variable).
- **train_test_split()**: Splits data into 80% training and 20% testing.

10. Train Multiple Regression Models

```
models = {
    "Linear Regression": LinearRegression(),
    "Ridge": Ridge(),
    "Lasso": Lasso(),
    "Random Forest": RandomForestRegressor(random_state=42)
}
```

- Defines four models:
    - **Linear Regression**: Basic regression model.
    - **Ridge Regression**: Linear regression with L2 regularization.
    - **Lasso Regression**: Linear regression with L1 regularization (for feature selection).
    - **Random Forest Regressor**: A tree-based ensemble model.

11. Model Training and Evaluation

```
results = {}

for name, model in models.items():
    model.fit(X_train, y_train)  # Train model
    y_pred = model.predict(X_test)  # Predict test data
    mse = mean_squared_error(y_test, y_pred)  # Calculate MSE
    r2 = r2_score(y_test, y_pred)  # Calculate R² score
    results[name] = {"MSE": mse, "R2 Score": r2}  # Store results
```

- Trains each model, makes predictions, and evaluates performance using:
    - **Mean Squared Error (MSE)**: Measures average squared difference between actual and predicted values.
    - **R² Score**: Measures how well the model explains variance (closer to 1 is better).

Display Results

```
display(pd.DataFrame(results).T)
```

- Converts the results dictionary into a **pandas DataFrame** and displays it.

Assessment 4

Provide well written paragraphs, annotations, tables and plots in your Jupyter Notebooksexplaining in detail the workflow, the results and the reasons for the choices you have made.

1. **Load & Explore Data**: Read CSV file, display basic info.
2. **Preprocessing**: Drop non-numeric columns, handle missing values, standardize features.
3. **PCA Analysis**: Reduce dimensionality and plot explained variance.
4. **K-Means Clustering**: Apply clustering and determine optimal K using the elbow method.
5. **Regression Models**: Train multiple models to predict **LD50**, evaluate with MSE & R^2 score