Lab Manual

Course Code	**AI-414**
Course Name	**Machine Learning**

Submitted by:

Name	Muhammad Tayyab Imran
Reg. No.	2023-BS-AI-019
Section	BSAI-4A

Submitted to:

Mr. Saeed Engr.

Department of Computer Science

Table of Contents

Regression

Classification

Regression

Project Summary: Gold Price Prediction

This project focuses on **predicting gold prices** using historical market data with a **Linear Regression model**. It involves **data preprocessing**, **outlier removal**, **data transformation**, **dimensionality reduction** using **PCA**, and finally, training and evaluating a **Linear Regression** model on the processed data

Dataset Description

The dataset **(Achilles_Data-Gold.csv) includes:**

Column	Description
open	Opening price of gold on a given day
high	Highest price of gold on that day
low	Lowest price of gold on that day
close	Closing price (target variable to predict)
ema	Exponential Moving Average - technical indicator
obv	On-Balance Volume - technical indicator
tick_volume	Volume of trades (or tick changes) observed

Objectives of the Project

- To predict the **closing price** of gold based on other market indicators.
- To apply **data preprocessing techniques** (handling outliers, scaling).
- To use **PCA** for dimensionality reduction before modeling.
- To train a **Linear Regression** model and evaluate its performance.

Step 1: Loading Libraries

These are the essential libraries used for **data manipulation**, **visualization**, and **machine learning**.

1: Loading Libraries

```
[2] import pandas as pd
    import numpy as np
    import sklearn
    import matplotlib.pyplot as plt
    import seaborn as sns

    from sklearn.preprocessing import StandardScaler
    from sklearn.decomposition import PCA
    from sklearn.model_selection import train_test_split
    from sklearn.linear_model import LinearRegression
    from sklearn.metrics import mean_absolute_error, mean_squared_error
```

Step 2: Reading and Exploring Data

This step gives you a **quick overview** of the dataset:

- .head() – Preview first rows.
- .info() – Check column types and missing values.
- .describe() – View basic statistics.
- .shape & .columns – Dataset structure.
- .nunique() – Unique values in each column.

2: Reading and Exploring Data

```
[3] data = pd.read_csv('/content/drive/MyDrive/Project/My Regression/Achilles_Data-Gold.csv')
```

```
[4] data.head()
```

	time	open	high	low	close	tick_volume	ema	obv
0	1672707720	1826.92	1827.65	1826.92	1827.24	47	2166.827022	47.0
1	1672707780	1827.26	1827.86	1827.18	1827.84	36	2166.827022	83.0
2	1672707840	1827.83	1828.70	1827.83	1828.66	47	2166.827022	130.0
3	1672707900	1828.65	1829.92	1828.62	1829.80	65	2166.827022	195.0
4	1672707960	1829.84	1830.94	1829.84	1830.63	87	2166.827022	282.0

```
[5] data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 708264 entries, 0 to 708263
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype  
---  ------       --------------   -----  
 0   time         708264 non-null  int64  
 1   open         708264 non-null  float64
 2   high         708264 non-null  float64
 3   low          708264 non-null  float64
 4   close        708264 non-null  float64
 5   tick_volume  708264 non-null  int64  
 6   ema          708264 non-null  float64
 7   obv          708264 non-null  float64
dtypes: float64(6), int64(2)
memory usage: 43.2 MB
```

```
[6] data.describe()
```

	time	open	high	low	close	tick_volume	ema	obv
count	7.082640e+05	708264.000000	708264.000000	708264.000000	708264.000000	708264.000000	708264.000000	708264.000000
mean	1.704141e+09	2166.827725	2167.117670	2166.537169	2166.828777	87.172062	2166.827022	111304.556951
std	1.819117e+07	276.537254	276.600773	276.472119	276.537495	67.389911	276.526264	137554.839012
min	1.672708e+09	1806.640000	1806.840000	1804.750000	1806.680000	1.000000	1808.018055	-17801.000000
25%	1.688442e+09	1945.370000	1945.560000	1945.170000	1945.370000	39.000000	1945.398445	3884.000000
50%	1.704253e+09	2028.970000	2029.200000	2028.720000	2028.970000	67.000000	2028.968310	63419.000000
75%	1.719968e+09	2381.780000	2382.170000	2381.420000	2381.780000	118.000000	2381.923933	153891.500000
max	1.735689e+09	2789.860000	2790.100000	2789.620000	2789.870000	547.000000	2789.087992	469474.000000

```
[7] data.shape
```

```
(708264, 8)
```

```
[8] data.columns
```

```
Index(['time', 'open', 'high', 'low', 'close', 'tick_volume', 'ema', 'obv'], dtype='object')
```

```
[9] data.nunique()
```

	0
time	708264
open	82314
high	82206
low	82325
close	82364
tick_volume	504
ema	708244
obv	242831

dtype: int64

Step 3: Data Cleaning

- Identifies any missing values in the dataset.
- Detects repeated rows that may bias the model.

3: Data Cleaning

Checking Null Values

```
[10] data.isnull().sum()
```

	0
time	0
open	0
high	0
low	0
close	0
tick_volume	0
ema	0
obv	0

dtype: int64

Checking Duplicates

```
[11] data.duplicated().sum()
```

```
np.int64(0)
```

Step 4: Outlier Detection and Removal

Filters out extreme values using the Interquartile Range (IQR).

4: Outlier Detection and Removal

```
[12] features = ["close", "open", "high", "low", "ema"]
     Q1 = data[features].quantile(0.25)
     Q3 = data[features].quantile(0.75)
     IQR = Q3 - Q1


     outlier_removed_data = data[~((data[features] < (Q1 - 1.5 * IQR)) | (data[features] > (Q3 + 1.5 * IQR))).any(axis=1)]
```

```
[13] outlier_removed_data.shape
```

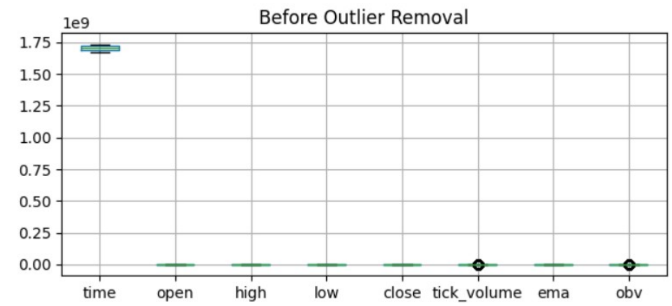
```
(708264, 8)
```

Outlier Visualization:

- **Boxplots** before and after removal to **compare the impact of outlier removal.**

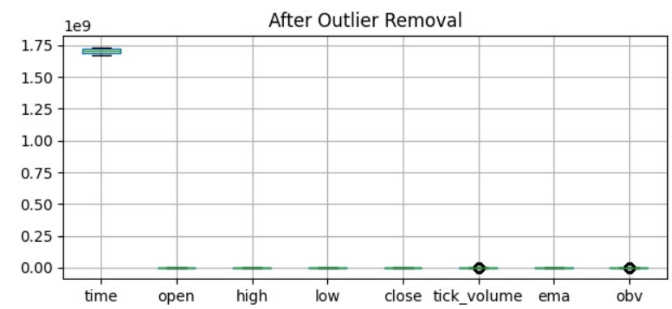
Box plot before removing outliers

```
[14] plt.figure(figsize=(12, 3))

     plt.subplot(1, 2, 1)
     data.boxplot()
     plt.title("Before Outlier Removal")

     plt.tight_layout()
     plt.show()
```



Box plot after removing outliers

```
[15] plt.figure(figsize=(12, 3))

     plt.subplot(1, 2, 2)
     outlier_removed_data.boxplot()
     plt.title("After Outlier Removal")

     plt.tight_layout()
     plt.show()
```



Step 5: Data Transformation (Standardization)

- Standardizes features to mean = 0 and standard deviation = 1.
- Necessary for PCA and regression models for better performance

5: Data Transformation

Standardization

```
[16] features_to_scale = ["open", "high", "low", "tick_volume", "ema", "obv"]
     x = outlier_removed_data[features_to_scale]
     y = outlier_removed_data['close']

     scaler = StandardScaler()
     x_scaled = scaler.fit_transform(x)
```

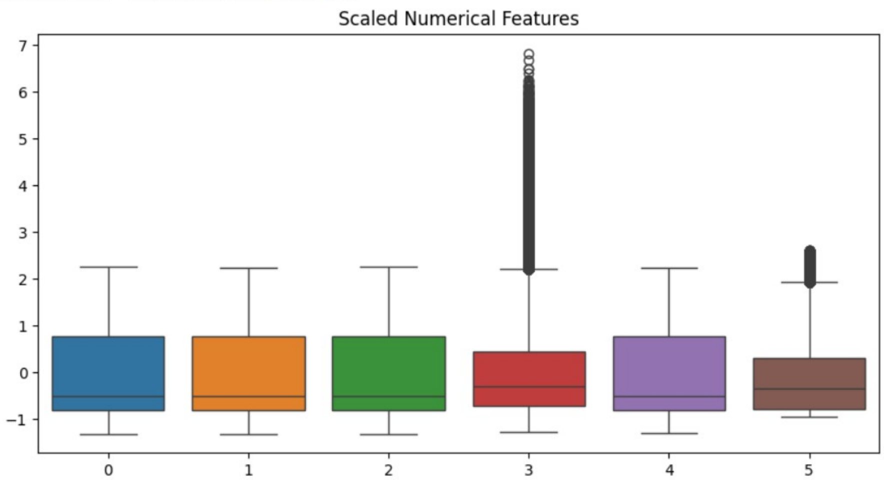
Standardized Data Visualization:

- **Boxplot Standardized Data** to ensures features are centered and standardized.

Plot the Standardized data

```
[17] plt.figure(figsize=(10, 5))
     sns.boxplot(data=x_scaled)
     plt.title('Scaled Numerical Features')
```

```
Text(0.5, 1.0, 'Scaled Numerical Features')
```



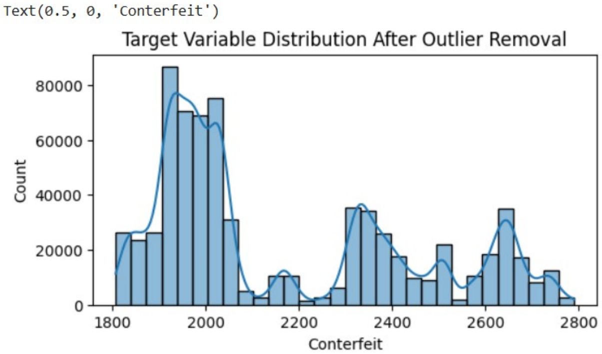
Target Variable Distribution Visualization:

- **Distribution Plot of** Target variable (Weather Type) is visualized to detect class imbalance.

Plot the Target Value Distribution

```
[18] plt.figure(figsize=(6, 3))
     sns.histplot(y, bins=30, kde=True)
     plt.title('Target Variable Distribution After Outlier Removal')
     plt.xlabel('Conterfeit')
```

```
Text(0.5, 0, 'Conterfeit')
```



Step 6: Principal Component Analysis (PCA)

- Reduces dimensionality while preserving variance.
- Helps in removing multicollinearity and speeding up model training.

6: Principal Component Analysis

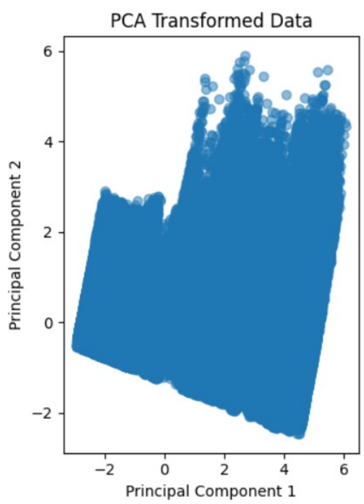
```
[19] pca = PCA(n_components=6)
     x_pca = pca.fit_transform(x_scaled)
```

PCA Visualization:

- **Scatter plot** of first two principal components shows **data distribution in reduced dimensions**.

Plot the PCA transformed data

```
[20] plt.subplot(1, 2, 2)
     plt.scatter(x_pca[:, 0], x_pca[:, 1], alpha=0.5)
     plt.title('PCA Transformed Data')
     plt.xlabel('Principal Component 1')
     plt.ylabel('Principal Component 2')

     plt.tight_layout()
     plt.show()
```



Step 7: Splitting Data

- Divides the dataset into:
    - **80% training** – used to fit the model
    - **20% testing** – used to evaluate performance.
- “random_state=42” ensures reproducibility.

7: Splitting Data

```
[21] x_train, x_test, y_train, y_test = train_test_split(x_pca, y, test_size=0.2, random_state=42)
```

Step 8: Linear Regression Modeling

Trains a **Linear Regression model** to learn the relationship between predictors and target.

8: Linear Regression

```
[22] model = LinearRegression()
     model.fit(x_train, y_train)
```

```
LinearRegression
LinearRegression()
```

Step 9: Evaluation Metrics

- **MAE (Mean Absolute Error):** Average absolute difference between predicted and actual.
- **MSE (Mean Squared Error):** Penalizes larger errors.
- **RMSE:** Root of MSE, interpretable in same units as target (close price).

9: Evaluation Metrics

```
[23] y_pred = model.predict(x_test)
     mae = mean_absolute_error(y_test, y_pred)
     mse = mean_squared_error(y_test, y_pred)
     rmse = np.sqrt(mse)

     print("\nEvaluation Metrics:")
     print(f"MAE:  {mae:.2f}")
     print(f"MSE:  {mse:.2f}")
     print(f"RMSE: {rmse:.2f}")
```

```
Evaluation Metrics:
MAE:  0.14
MSE:  0.04
RMSE: 0.21
```

Step 10: Sample Predictions

Displays a comparison of **real vs predicted** values.

10: Sample Prediction

```
[24] results = pd.DataFrame({"Actual": y_test.values, "Predicted": y_pred})
     print("\nSample Predictions:\n", results.head())
```

```
Sample Predictions:
     Actual    Predicted
0  2331.28  2331.350590
1  1927.66  1927.647141
2  1950.58  1950.719322
3  1907.60  1907.736168
4  2522.43  2522.224296
```

Final Summary:

Step	Purpose
Loading Libraries	To import required Python tools
Data Reading & Exploration	Understand dataset structure and contents
Data Cleaning	Remove noise and errors (nulls, duplicates)
Outlier Removal	Eliminate extreme values that skew the model
Standardization	Normalize features for fair comparison
PCA	Reduce complexity while preserving variance
Train-Test Split	Evaluate model generalizability
Linear Regression	Learn and predict relationships
Metrics	Assess model accuracy and performance
Predictions	Visually compare predictions against actual values

Classification

Project Summary: Weather Classification

This machine learning project involves classifying weather types (like Sunny, Rainy, Cloudy, etc.) based on various meteorological factors using a Random Forest Classifier. The process includes data cleaning, handling outliers, one-hot encoding, standardization, training a classifier, and evaluating its performance.

Dataset Explanation

The dataset (weather_classification_data.csv) includes:

Column Name	Description
Temperature	Measured in °C
Humidity	Relative humidity (%)
Wind Speed	Wind speed (possibly in km/h or m/s)
Precipitation (%)	Likelihood or intensity of precipitation
Atmospheric Pressure	Pressure in hPa or similar
UV Index	UV radiation index
Visibility (km)	Distance visible
Cloud Cover	Categorical (e.g. Clear, Partial, Overcast)
Season	Categorical (e.g. Winter, Summer)
Location	Categorical (e.g. Lahore, Karachi)
Weather Type	Target variable (e.g. Sunny, Rainy, Snowy)

Objectives of the Project

- To build a model that predicts the type of weather based on environmental data.
- To perform data preprocessing, including outlier removal, feature encoding, and scaling.
- To train a classification model (Random Forest) and evaluate its performance.

Step 1: Loading Libraries

These are the essential libraries used for **data manipulation**, **visualization**, and **machine learning**.

1: Loading Libraries

```
[2] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns

    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import StandardScaler
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.metrics import classification_report, confusion_matrix
```

Step 2: Reading and Exploring Data

This step gives you a **quick overview** of the dataset:

- .head() – Preview first rows.
- .info() – Check column types and missing values.
- .describe() – View basic statistics.
- .shape & .columns – Dataset structure.
- .nunique() – Unique values in each column.

2: Reading and Exploring Data

```
[3] data = pd.read_csv('/content/drive/MyDrive/Project/My Classification/weather_classification_data.csv')
```

```
[4] data.head()
```

	Temperature	Humidity	Wind Speed	Precipitation (%)	Cloud Cover	Atmospheric Pressure	UV Index	Season	Visibility (km)	Location	Weather Type
0	14.0	73	9.5	82.0	partly cloudy	1010.82	2	Winter	3.5	inland	Rainy
1	39.0	96	8.5	71.0	partly cloudy	1011.43	7	Spring	10.0	inland	Cloudy
2	30.0	64	7.0	16.0	clear	1018.72	5	Spring	5.5	mountain	Sunny
3	38.0	83	1.5	82.0	clear	1026.25	7	Spring	1.0	coastal	Sunny
4	27.0	74	17.0	66.0	overcast	990.67	1	Winter	2.5	mountain	Rainy

[5] data.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13200 entries, 0 to 13199
Data columns (total 11 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Temperature           13200 non-null  float64
 1   Humidity              13200 non-null  int64  
 2   Wind Speed            13200 non-null  float64
 3   Precipitation (%)     13200 non-null  float64
 4   Cloud Cover           13200 non-null  object 
 5   Atmospheric Pressure  13200 non-null  float64
 6   UV Index              13200 non-null  int64  
 7   Season                13200 non-null  object 
 8   Visibility (km)       13200 non-null  float64
 9   Location              13200 non-null  object 
 10  Weather Type          13200 non-null  object 
dtypes: float64(5), int64(2), object(4)
memory usage: 1.1+ MB
```

[6] data.describe()

	Temperature	Humidity	Wind Speed	Precipitation (%)	Atmospheric Pressure	UV Index	Visibility (km)
count	13200.000000	13200.000000	13200.000000	13200.000000	13200.000000	13200.000000	13200.000000
mean	19.127576	68.710833	9.832197	53.644394	1005.827896	4.005758	5.462917
std	17.386327	20.194248	6.908704	31.946541	37.199589	3.856600	3.371499
min	-25.000000	20.000000	0.000000	0.000000	800.120000	0.000000	0.000000
25%	4.000000	57.000000	5.000000	19.000000	994.800000	1.000000	3.000000
50%	21.000000	70.000000	9.000000	58.000000	1007.650000	3.000000	5.000000
75%	31.000000	84.000000	13.500000	82.000000	1016.772500	7.000000	7.500000
max	109.000000	109.000000	48.500000	109.000000	1199.210000	14.000000	20.000000

[7] data.shape

```
(13200, 11)
```

[8] data.columns

```
Index(['Temperature', 'Humidity', 'Wind Speed', 'Precipitation (%)',
       'Cloud Cover', 'Atmospheric Pressure', 'UV Index', 'Season',
       'Visibility (km)', 'Location', 'Weather Type'],
      dtype='object')
```

[9] data.nunique()

	0
Temperature	126
Humidity	90
Wind Speed	97
Precipitation (%)	110
Cloud Cover	4
Atmospheric Pressure	5456
UV Index	15
Season	4
Visibility (km)	41
Location	3
Weather Type	4

dtype: int64

Step 3: Data Cleaning

- Identifies any missing values in the dataset.
- Detects repeated rows that may bias the model.

3: Data Cleaning

Checking Null Values

```
[10] data.isnull().sum()
```

	0
Temperature	0
Humidity	0
Wind Speed	0
Precipitation (%)	0
Cloud Cover	0
Atmospheric Pressure	0
UV Index	0
Season	0
Visibility (km)	0
Location	0
Weather Type	0

dtype: int64

Checking Duplicates

```
[11] data.duplicated().sum()
```

```
np.int64(0)
```

Step 4: Outlier Detection and Removal

Filters out extreme values using the Interquartile Range (IQR).

4: Outlier Detection and Removal

```
[12] numerical_cols = ['Temperature', 'Humidity', 'Wind Speed', 'Precipitation (%)', 'Atmospheric Pressure', 'UV Index', 'Visibility (km)']

     Q1 = data[numerical_cols].quantile(0.25)
     Q3 = data[numerical_cols].quantile(0.75)
     IQR = Q3 - Q1

     outlier_removed_data = data[~((data[numerical_cols] < (Q1 - 1.5 * IQR)) | (data[numerical_cols] > (Q3 + 1.5 * IQR))).any(axis=1)]
```

```
[13] outlier_removed_data.shape
```

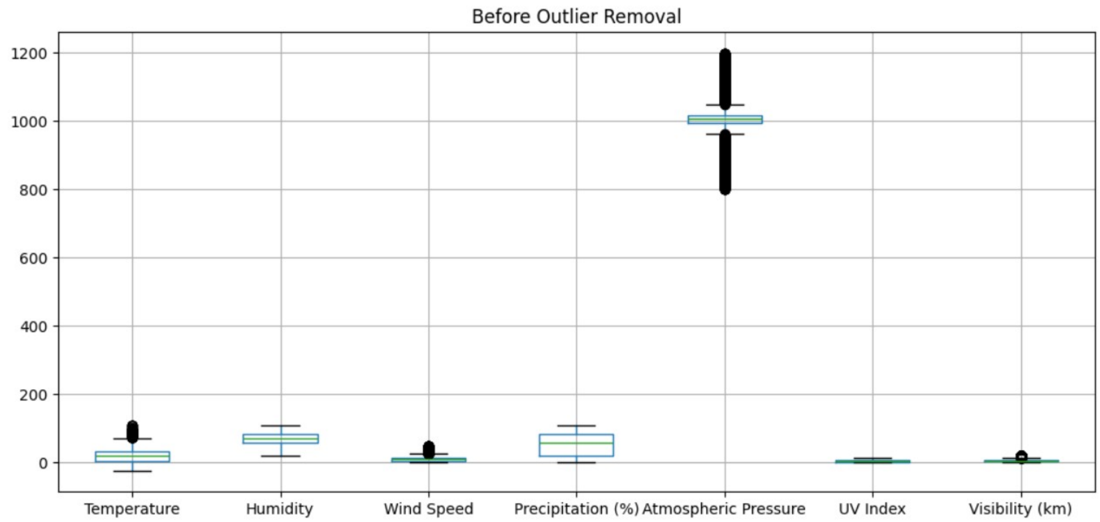
```
(11689, 11)
```

Outlier Visualization:

- Boxplots before and after removal to compare the impact of outlier removal.

Box plot before removing outliers

```
[14] plt.figure(figsize=(20, 5))

     plt.subplot(1, 2, 1)
     data.boxplot()
     plt.title("Before Outlier Removal")

     plt.tight_layout()
     plt.show()
```



Box plot after removing outliers

```
plt.figure(figsize=(20, 5))

plt.subplot(1, 2, 2)
outlier_removed_data.boxplot()
plt.title("After Outlier Removal")

plt.tight_layout()
plt.show()
```

Step 5: One-Hot Encoding (for Categorical Variables)

Converts categorical variables into binary (0/1) format required for machine learning models.

5: One Hot Encoding

```
[16] categorical_cols = ['Cloud Cover', 'Season', 'Location']

     encoded_data = pd.get_dummies(outlier_removed_data, columns=categorical_cols)

     encoded_data.head()
```

	Temperature	Humidity	Wind Speed	Precipitation (%)	Atmospheric Pressure	UV Index	Visibility (km)	Weather Type	Cloud Cover_clear	Cloud Cover_cloudy	Cloud Cover_overcast	Cloud Cover_partly cloudy
0	14.0	73	9.5	82.0	1010.82	2	3.5	Rainy	False	False	False	True
1	39.0	96	8.5	71.0	1011.43	7	10.0	Cloudy	False	False	False	True
2	30.0	64	7.0	16.0	1018.72	5	5.5	Sunny	True	False	False	False
3	38.0	83	1.5	82.0	1026.25	7	1.0	Sunny	True	False	False	False
4	27.0	74	17.0	66.0	990.67	1	2.5	Rainy	False	False	True	False

Step 6: Data Transformation (Standardization)

- Standardizes features to mean = 0 and standard deviation = 1.
- Necessary for PCA and regression models for better performance.

6: Data Transformation

Standardization

```
[17] x = encoded_data.drop('Weather Type', axis=1)
     y = encoded_data['Weather Type']

     scaler = StandardScaler()
     x_scaled = scaler.fit_transform(x)

     feature_names = x.columns
     x_scaled = pd.DataFrame(x_scaled, columns=feature_names, index=x.index)

     x_scaled.head()
```

	Temperature	Humidity	Wind Speed	Precipitation (%)	Atmospheric Pressure	UV Index	Visibility (km)	Cloud Cover_clear	Cloud Cover_cloudy	Cloud Cover_overcast	Cloud Cover_partly cloudy	Se
0	-0.319956	0.180018	0.050410	0.935683	0.352182	-0.448083	-0.616594	-0.465801	-0.070002	-0.938817	1.367691	
1	1.250401	1.364687	-0.127689	0.592525	0.399333	0.940475	1.878101	-0.465801	-0.070002	-0.938817	1.367691	
2	0.685072	-0.283548	-0.394836	-1.123266	0.962832	0.385052	0.151005	2.146841	-0.070002	-0.938817	-0.731159	
3	1.187586	0.695092	-1.374378	0.935683	1.544883	0.940475	-1.576092	2.146841	-0.070002	-0.938817	-0.731159	
4	0.496630	0.231525	1.386148	0.436544	-1.205363	-0.725794	-1.000393	-0.465801	-0.070002	1.065170	-0.731159	

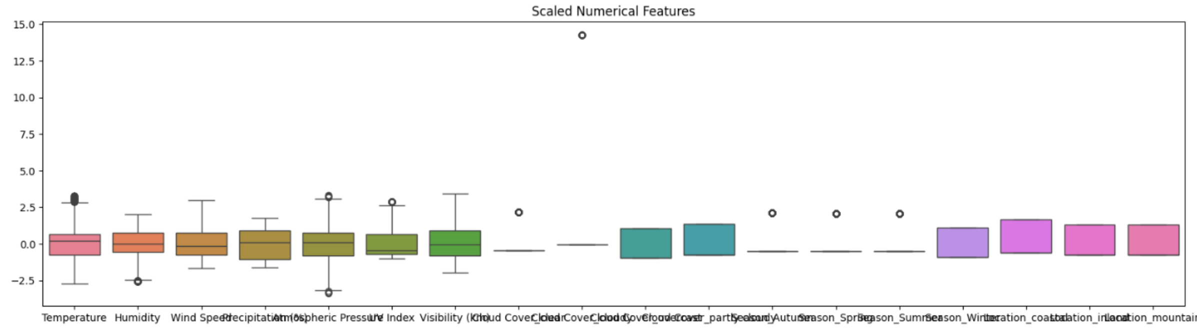
Standardized Data Visualization:

- **Boxplot Standardized Data** to ensures features are centered and standardized.

⌄ Plot the Standardized data

```
[18] plt.figure(figsize=(20, 5))
     sns.boxplot(data=x_scaled)
     plt.title('Scaled Numerical Features')
```

```
Text(0.5, 1.0, 'Scaled Numerical Features')
```



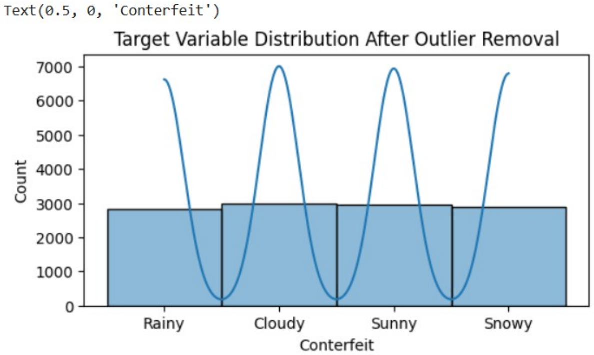
Target Variable Distribution Visualization:

- **Distribution Plot of** Target variable (Weather Type) is visualized to detect class imbalance.

⌄ Plot the Target Value Distribution

```
[19] plt.figure(figsize=(6, 3))
     sns.histplot(y, bins=30, kde=True)
     plt.title('Target Variable Distribution After Outlier Removal')
     plt.xlabel('Conterfeit')
```

```
Text(0.5, 0, 'Conterfeit')
```



Step 7: Splitting Data

- Divides the dataset into:
  - **80% training** – used to fit the model
  - **20% testing** – used to evaluate performance.
- “random_state=42” ensures reproducibility.

7: Splitting Data

```
[20] x_train, x_test, y_train, y_test = train_test_split(x_scaled, y, test_size=0.2, random_state=42)
```

Step 8: Classification (Random Forest Classifier)

A robust, non-linear classifier that uses an ensemble of decision trees to make predictions.

8: Classification

```
[21] model = RandomForestClassifier()
     model.fit(x_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier()
```

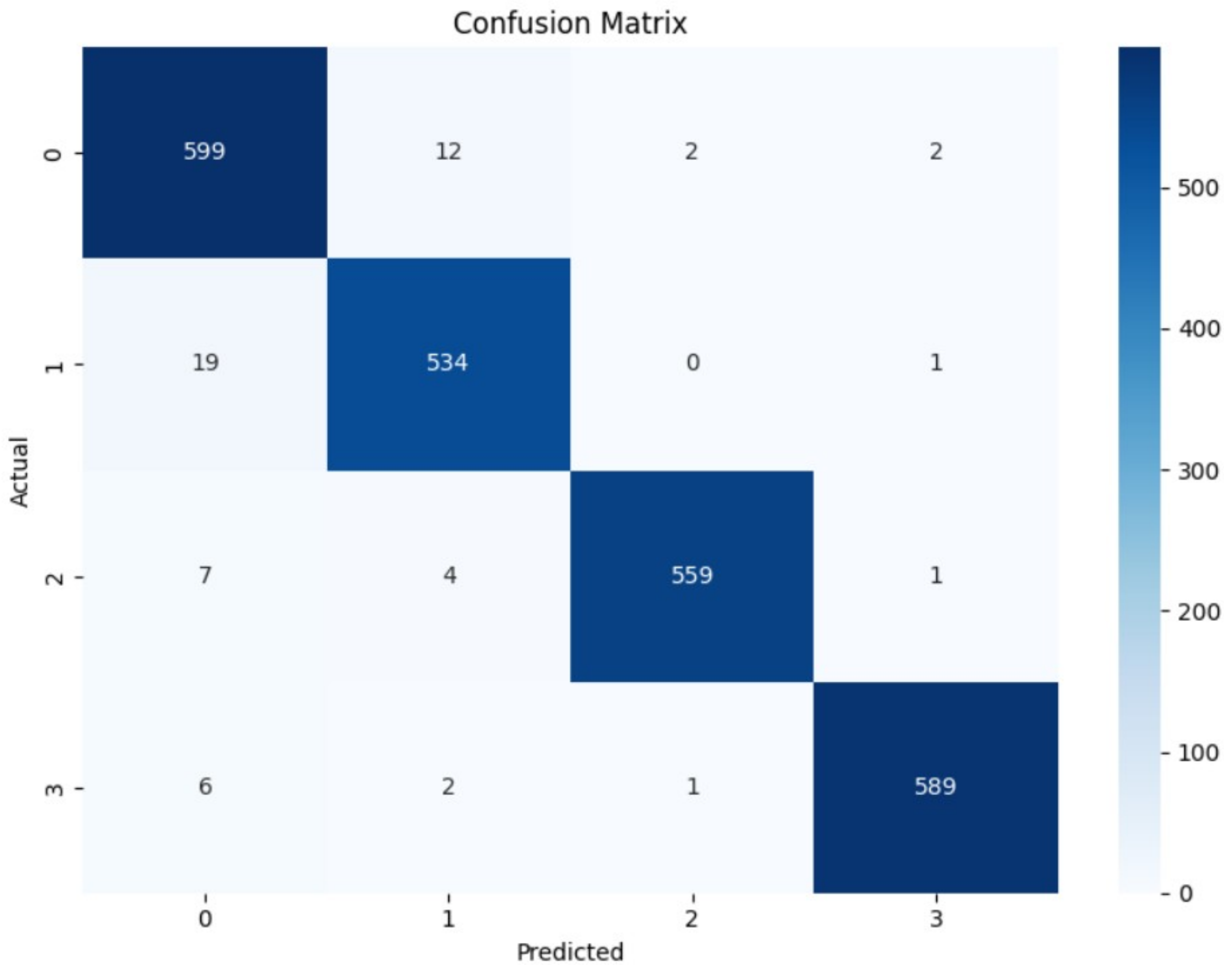
Step 9: Evaluation Metrics

- **Classification Report:** Shows precision, recall, f1-score, and support for each weather type class.
- **Confusion Matrix:** Displays the performance of the classifier across different classes in a visual format

9: Evaluation Metrics

```
[22] y_pred = model.predict(x_test)

     # Print classification metrics
     print("\nClassification Report:\n")
     print(classification_report(y_test, y_pred))

     # Confusion matrix plot
     conf_matrix = confusion_matrix(y_test, y_pred)
     plt.figure(figsize=(8, 6))
     sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
     plt.title("Confusion Matrix")
     plt.xlabel("Predicted")
     plt.ylabel("Actual")
     plt.tight_layout()
     plt.show()
```

```
Classification Report:

              precision    recall  f1-score   support

      Cloudy       0.95      0.97      0.96       615
       Rainy       0.97      0.96      0.97       554
       Snowy       0.99      0.98      0.99       571
       Sunny       0.99      0.98      0.99       598

    accuracy                           0.98      2338
   macro avg       0.98      0.98      0.98      2338
weighted avg       0.98      0.98      0.98      2338
```



Confusion Matrix
599
12
2
2
19
534
0
1
7
4
559
1
6
2
1
589
Actual
Predicted
0
1
2
3
500
400
300
200
100
0

Final Summary:

Step	Purpose
Libraries	Set up necessary Python tools
Data Exploration	Understand dataset shape and structure
Cleaning	Remove errors (nulls, duplicates)
Outlier Removal	Eliminate values that distort learning
One-Hot Encoding	Convert categories into machine-readable format
Standardization	Normalize feature scales
Train-Test Split	Prepare data for model training and testing
Random Forest Classifier	Train a model to predict weather types
Evaluation Metrics	Assess performance (accuracy, precision, recall)