Lab Manual

Machine Learning

Course Code: AI-414

Submitted by:

Eman Tahir	**2023-BS-AI-015**
Semester no.	**4**

Submitted to:

Sir Muhammad Saeed

Bachelor of Science in Artificial Intelligence

Department of Computer Science

The University of Faisalabad

Contents

Project 1:

Student performance prediction using regression

Summary

This project focuses on building machine learning pipeline to predict student's performance outcomes using the dataset provided. The dataset includes information about secondary school students. The project begins with data exploration and cleaning where missing values and duplicates are removed and outliers are detected. The cleaned data is then transformed through normalization. The goal is to build a regression model to predict students grades with performance. The project follows a complete machine learning pipeline from pre-processing to prediction. The core objective is to train a regression model to predict grades of student.

Objective:

1. Analyze the data
2. Clean dataset
3. Applying preprocessing steps
4. Building a regression model
5. Evaluate the model performance

Abstract

The project aims to predict student performance using machine learning techniques applied to the dataset that contains information about students. The analysis begins with data cleaning that includes handling missing values, duplicates and outliers. Numerical features are scaled and categorical variables are encoded. Principal component analysis is considered for dimensionality reduction to improve model efficiency. The project demonstrates a complete machine learning pipeline and provides insights into the key factors influencing student success.

Steps involved

1. Importing libraries
2. Data reading
3. Data cleaning
4. Outlier detection and removal
5. Data transformation
6. One-hot encoding
7. Data reduction
8. Handling imbalanced data
9. Splitting data
10. Regression model
11. Evaluation metrices

Working

Importing libraries

- Matplotlib: For creating customizable charts and graphs.
- Seaborn: Enhances Matplotlib with stylish, statistical visualizations.
- NumPy: Handles complex numerical computations and array operations.
- Pandas: Manages structured data, making analysis easier.

```
[2]: import matplotlib.pyplot as plt
     import seaborn as sns
     color = sns.color_palette()
     import numpy as np
     import pandas as pd
```

Data reading

```
data = pd.read_csv('student-por.csv')
```

The head() function displays first few rows, you can also specify the rows that need to be displayed.

```
data.head()
```

	age	Medu	Fedu	traveltime	studytime	failures	famrel	freetime	goout	Dalc	...	activities_no	activities_yes	nursery_no	nursery_yes
0	1.031695	1.310216	1.540715	0.576718	0	-0.374305	0.072606	-0.171647	0.693785	-0.543555	...	0.971140	-0.971140	-0.495663	0.495663
1	0.210137	-1.336039	-1.188832	-0.760032	0	-0.374305	1.119748	-0.171647	-0.157380	-0.543555	...	0.971140	-0.971140	2.017502	-2.017502
2	-1.432980	-1.336039	-1.188832	-0.760032	0	-0.374305	0.072606	-0.171647	-1.008546	0.538553	...	0.971140	-0.971140	-0.495663	0.495663
3	-1.432980	1.310216	-0.278983	-0.760032	1	-0.374305	-0.974536	-1.123771	-1.008546	-0.543555	...	-1.029717	1.029717	-0.495663	0.495663
4	-0.611422	0.428131	0.630866	-0.760032	0	-0.374305	0.072606	-0.171647	-1.008546	-0.543555	...	0.971140	-0.971140	-0.495663	0.495663

5 rows × 59 columns

The tail() function displays last few rows, you can also specify the rows that need to be displayed.

```
[8]: data.tail()
```

[8]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3
644	MS	F	19	R	GT3	T	2	3	services	other	...	5	4	2	1	2	5	4	10	11	10
645	MS	F	18	U	LE3	T	3	1	teacher	services	...	4	3	4	1	1	1	4	15	15	16
646	MS	F	18	U	GT3	T	1	1	other	other	...	1	1	1	1	1	5	6	11	12	9
647	MS	M	17	U	LE3	T	3	1	services	services	...	2	4	5	3	4	2	6	10	10	10
648	MS	M	18	R	LE3	T	3	2	services	other	...	4	4	1	3	4	5	4	10	11	11

5 rows × 33 columns

.shape() helps you check the shape of the dataset i.e. rows and columns.

```
[10]: data.shape
[10]: (649, 33)
```

.sample() is used for checking the subsets of data.

```
[12]: data.sample()
```

[12]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3
424	MS	F	16	R	GT3	T	2	2	other	other	...	4	4	4	1	1	5	0	12	12	12

1 rows × 33 columns

.info() provides summary of the data frame i.e. column names.

```
[14]: data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 649 entries, 0 to 648
Data columns (total 33 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   school      649 non-null    object
 1   sex         649 non-null    object
 2   age         649 non-null    int64 
 3   address     649 non-null    object
 4   famsize     649 non-null    object
 5   Pstatus     649 non-null    object
 6   Medu        649 non-null    int64 
 7   Fedu        649 non-null    int64 
 8   Mjob        649 non-null    object
 9   Fjob        649 non-null    object
 10  reason      649 non-null    object
 11  guardian    649 non-null    object
 12  traveltime  649 non-null    int64 
 13  studytime   649 non-null    int64 
 14  failures    649 non-null    int64 
 15  schoolsup   649 non-null    object
 16  famsup      649 non-null    object
 17  paid        649 non-null    object
 18  activities  649 non-null    object
 19  nursery     649 non-null    object
 20  higher      649 non-null    object
 21  internet    649 non-null    object
 22  romantic    649 non-null    object
 23  famrel      649 non-null    int64 
 24  freetime    649 non-null    int64 
 25  goout       649 non-null    int64 
 26  Dalc        649 non-null    int64 
 27  Walc        649 non-null    int64 
 28  health      649 non-null    int64 
 29  absences    649 non-null    int64 
 30  G1          649 non-null    int64 
 31  G2          649 non-null    int64 
 32  G3          649 non-null    int64 
dtypes: int64(16), object(17)
memory usage: 167.4+ KB
```

.describe() provides descriptive statistics for numerical columns in data frame.

```
[16]: data.describe()
```

[16]:

	age	Medu	Fedu	traveltime	studytime	failures	famrel	freetime	goout	Dalc	Walc	health	absences
count	649.000000	649.000000	649.000000	649.000000	649.000000	649.000000	649.000000	649.000000	649.000000	649.000000	649.000000	649.000000	649.000000
mean	16.744222	2.514638	2.306626	1.568567	1.930663	0.221880	3.930663	3.180277	3.184900	1.502311	2.280431	3.536210	3.659476
std	1.218138	1.134552	1.099931	0.748660	0.829510	0.593235	0.955717	1.051093	1.175766	0.924834	1.284380	1.446259	4.640759
min	15.000000	0.000000	0.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000
25%	16.000000	2.000000	1.000000	1.000000	1.000000	0.000000	4.000000	3.000000	2.000000	1.000000	1.000000	2.000000	0.000000
50%	17.000000	2.000000	2.000000	1.000000	2.000000	0.000000	4.000000	3.000000	3.000000	1.000000	2.000000	4.000000	2.000000
75%	18.000000	4.000000	3.000000	2.000000	2.000000	0.000000	5.000000	4.000000	4.000000	2.000000	3.000000	5.000000	6.000000
max	22.000000	4.000000	4.000000	4.000000	4.000000	3.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	32.000000

Data cleaning

.isnull().sum() helps to check missing values.

```
[18]: data.isnull().sum()

[18]: school        0
      sex           0
      age           0
      address       0
      famsize       0
      Pstatus       0
      Medu          0
      Fedu          0
      Mjob          0
      Fjob          0
      reason        0
      guardian      0
      traveltime    0
      studytime     0
      failures      0
      schoolsup     0
      famsup        0
      paid          0
      activities    0
      nursery       0
      higher        0
      internet      0
      romantic      0
      famrel        0
      freetime      0
      goout         0
      Dalc          0
      Walc          0
      health        0
      absences      0
      G1            0
      G2            0
      G3            0
      dtype: int64
```

Removing missing values and checking remaining values.

```
[24]: data.dropna(inplace=True)
      missing_values = data.isnull().sum()
      print(missing_values)

      age           0
      Medu          0
      Fedu          0
      traveltime    0
      studytime     0
      failures      0
      famrel        0
      freetime      0
      goout         0
      Dalc          0
      Walc          0
      health        0
      absences      0
      G1            0
      G2            0
      G3            0
      school        0
      sex           0
      address       0
      famsize       0
      Pstatus       0
      Mjob          0
      Fjob          0
      reason        0
      guardian      0
      schoolsup     0
      famsup        0
      paid          0
      activities    0
      nursery       0
      higher        0
      internet      0
      romantic      0
      dtype: int64
```

Removing duplicated rows and checking updated data frame.

```
[26]: data.drop_duplicates(inplace=True)
      data.shape

[26]: (649, 33)
```

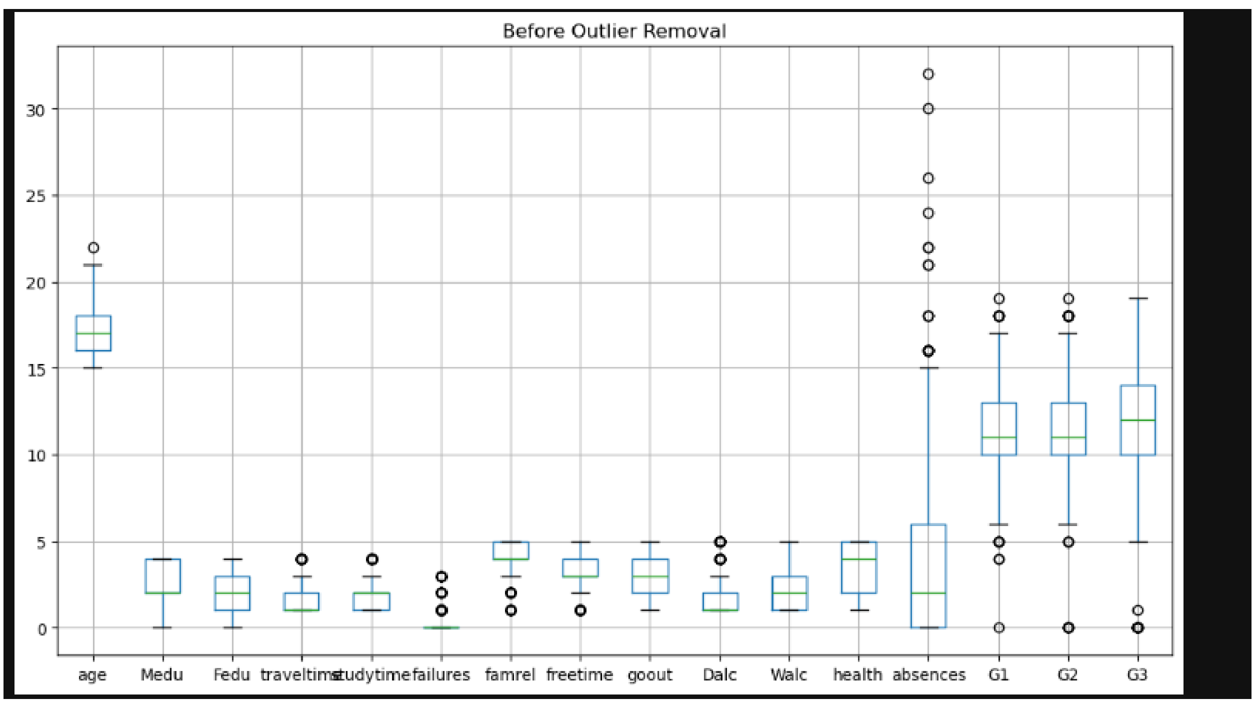
Outlier detection and removal

```
[28]: 0.25-1.5*0.5

[28]: -0.5

[30]: 0.75 + 1.5 * 0.5

[30]: 1.5
```

Before Outlier Removal

Selecting the numerical data and identifying outlier using interquartile range. Outliers are values that are usually high or low compared to the rest of the data.

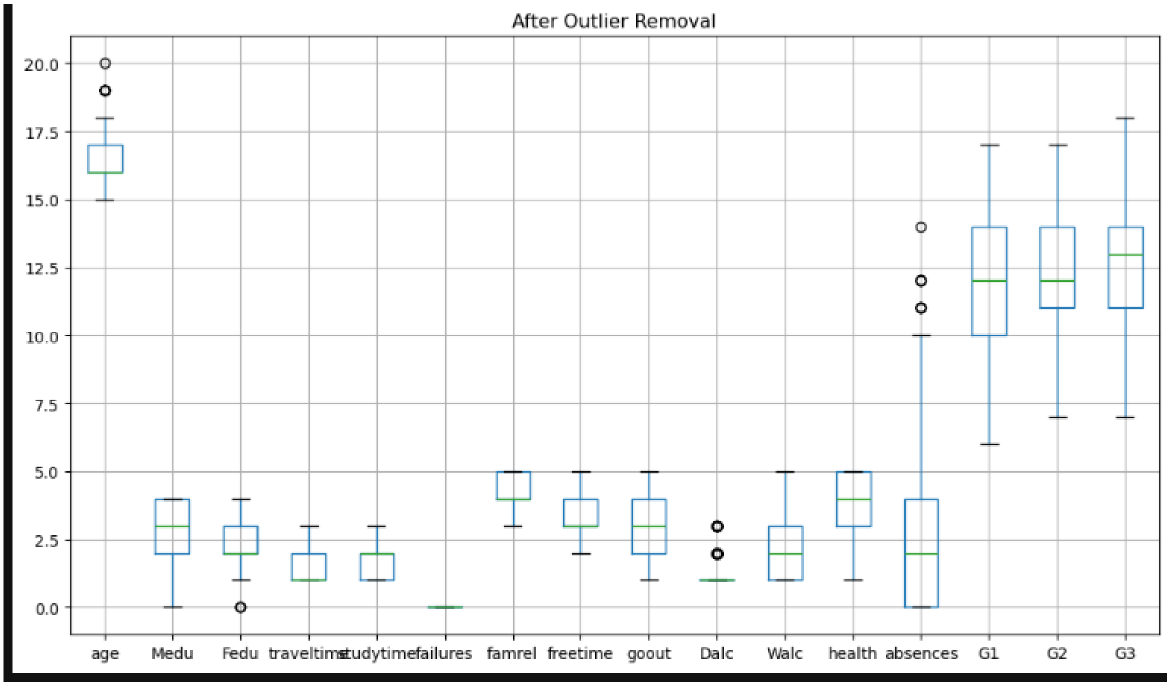
```
[32]: numeric_cols = data.select_dtypes(include=[np.number])

      Q1 = numeric_cols.quantile(0.25)
      Q3 = numeric_cols.quantile(0.75)
      IQR = Q3 - Q1

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


      plt.figure(figsize=(20, 6))

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

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



After Outlier Removal

Creating a box to visualize data after removing outliers. It helps understanding the distribution of numbers by showing key values like minimum, maximum, medium etc. it uses matplot and pandas library to select only the numeric data.

```
[34]: plt.figure(figsize=(20, 6))

plt.subplot(1, 2, 2)
data_cleaned.select_dtypes(include=[np.number]).boxplot()
plt.title("After Outlier Removal")

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



Displaying the cleaned data:

```
[36]: data_cleaned.shape

[36]: (393, 33)
```

```
[38]: data_cleaned.head()
```

[38]:

	age	Medu	Fedu	traveltime	studytime	failures	famrel	freetime	goout	Dalc	...	reason	guardian	schoolsup	famsup	paid	activities	nursery	higher	int
1	17	1	1	1	2	0	5	3	3	1	...	course	father	no	yes	no	no	no	yes	
2	15	1	1	1	2	0	4	3	2	2	...	other	mother	yes	no	no	no	yes	yes	
3	15	4	2	1	3	0	3	2	2	1	...	home	mother	no	yes	no	yes	yes	yes	
4	16	3	3	1	2	0	4	3	2	1	...	home	father	no	yes	no	no	yes	yes	
5	16	4	3	1	2	0	5	4	2	1	...	reputation	mother	no	yes	no	yes	yes	yes	

5 rows × 33 columns

Data Transformation

Data normalization

The process of scaling numerical values to a specific range. It helps improve model performance by preventing certain variables from dominating due to differences in scale.

1.data normalization

[42]:
```python
from sklearn.preprocessing import MinMaxScaler
numeric_cols = data.select_dtypes(include=[np.number])
non_numeric_cols = data.select_dtypes(exclude=[np.number])

scaler = MinMaxScaler()
scaled_numeric_data = scaler.fit_transform(numeric_cols)

scaled_numeric_df = pd.DataFrame(scaled_numeric_data, columns=numeric_cols.columns)

scaled_data = pd.concat([scaled_numeric_df, non_numeric_cols.reset_index(drop=True)], axis=1)

print(scaled_data.shape)
print()
print('*' * 60)
scaled_data.head()
```

```
(649, 33)

************************************************************
```

[42]:

	age	Medu	Fedu	traveltime	studytime	failures	famrel	freetime	goout	Dalc	...	reason	guardian	schoolsup	famsup	paid	activities	nursery	higher	i
0	0.428571	1.00	1.00	0.333333	0.333333	0.0	0.75	0.50	0.75	0.00	...	course	mother	yes	no	no	no	yes	yes	
1	0.285714	0.25	0.25	0.000000	0.333333	0.0	1.00	0.50	0.50	0.00	...	course	father	no	yes	no	no	no	yes	
2	0.000000	0.25	0.25	0.000000	0.333333	0.0	0.75	0.50	0.25	0.25	...	other	mother	yes	no	no	no	yes	yes	
3	0.000000	1.00	0.50	0.000000	0.666667	0.0	0.50	0.25	0.25	0.00	...	home	mother	no	yes	no	yes	yes	yes	
4	0.142857	0.75	0.75	0.000000	0.333333	0.0	0.75	0.50	0.25	0.00	...	home	father	no	yes	no	no	yes	yes	

5 rows × 33 columns

Data standardization

It transforms numerical values to have mean of 0 and a standard deviation of 1. It helps models handle differences in magnitude without biasing predictions towards larger values.

2.data standardization

[44]:
```python
from sklearn.preprocessing import StandardScaler
numeric_cols = data.select_dtypes(include=[np.number])
non_numeric_cols = data.select_dtypes(exclude=[np.number])

scaler = StandardScaler()
scaled_numeric_data = scaler.fit_transform(numeric_cols)

scaled_numeric_df = pd.DataFrame(scaled_numeric_data, columns=numeric_cols.columns)

scaled_data = pd.concat([scaled_numeric_df, non_numeric_cols.reset_index(drop=True)], axis=1)

print(scaled_data.shape)
print()
print('*' * 60)
scaled_data.head()
```

```
(649, 33)

************************************************************
```

[44]:

	age	Medu	Fedu	traveltime	studytime	failures	famrel	freetime	goout	Dalc	...	reason	guardian	schoolsup	famsup	paid	activiti
0	1.031695	1.310216	1.540715	0.576718	0.083653	-0.374305	0.072606	-0.171647	0.693785	-0.543555	...	course	mother	yes	no	no	r
1	0.210137	-1.336039	-1.188832	-0.760032	0.083653	-0.374305	1.119748	-0.171647	-0.157380	-0.543555	...	course	father	no	yes	no	r
2	-1.432980	-1.336039	-1.188832	-0.760032	0.083653	-0.374305	0.072606	-0.171647	-1.008546	0.538553	...	other	mother	yes	no	no	r
3	-1.432980	1.310216	-0.278983	-0.760032	1.290114	-0.374305	-0.974536	-1.123771	-1.008546	-0.543555	...	home	mother	no	yes	no	y
4	-0.611422	0.428131	0.630866	-0.760032	0.083653	-0.374305	0.072606	-0.171647	-1.008546	-0.543555	...	home	father	no	yes	no	r

5 rows × 33 columns

One-hot encoding

.unique() is used to find unique values in column of array.

```
[46]: data["studytime"].unique()

[46]: array([2, 3, 1, 4], dtype=int64)
```

The following code processes categorical data into in a dataset. It identifies the text values and convert them into numerical format using a method called "dummy encoding". It helps the data suitable for machine learning models. cat_features refers to categorical features.

```
[50]: cat_features = [feature for feature in data.columns if data[feature].dtype == 'O']

      data1 = pd.get_dummies(cat_features)
      data1
```

[50]:

	Fjob	Mjob	Pstatus	activities	address	famsize	famsup	guardian	higher	internet	nursery	paid	reason	romantic	school	schoolsup	sex
0	False	False	False	False	False	False	False	False	False	False	False	False	False	False	True	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	True
2	False	False	False	False	True	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	True	False	False	False	False	False	False	False	False	False	False	False
4	False	False	True	False	False	False	False	False	False	False	False	False	False	False	False	False	False
5	False	True	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
6	True	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
7	False	False	False	False	False	False	False	False	False	False	False	False	True	False	False	False	False
8	False	False	False	False	False	False	False	True	False	False	False	False	False	False	False	False	False
9	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	True	False
10	False	False	False	False	False	False	True	False	False	False	False	False	False	False	False	False	False
11	False	False	False	False	False	False	False	False	False	False	False	True	False	False	False	False	False
12	False	False	False	True	False	False	False	False	False	False	False	False	False	False	False	False	False
13	False	False	False	False	False	False	False	False	False	False	True	False	False	False	False	False	False
14	False	False	False	False	False	False	False	False	True	False	False	False	False	False	False	False	False
15	False	False	False	False	False	False	False	False	False	True	False	False	False	False	False	False	False
16	False	False	False	False	False	False	False	False	False	False	False	False	False	True	False	False	False

```
[52]: cat_features = [feature for feature in data.columns if data[feature].dtype == 'O']

      data1 = pd.get_dummies(data, columns=cat_features)

      scaled_data = pd.concat([data, data1], axis=1)

      print(scaled_data.shape)
      print()
      print('*' * 70)

      scaled_data.head()

(649, 92)

**********************************************************************
```

[52]:

	age	Medu	Fedu	traveltime	studytime	failures	famrel	freetime	goout	Dalc	...	activities_no	activities_yes	nursery_no	nursery_yes	higher_no	higher_yes	ir
0	18	4	4	2	2	0	4	3	4	1	...	True	False	False	True	False	True	
1	17	1	1	1	2	0	5	3	3	1	...	True	False	True	False	False	True	
2	15	1	1	1	2	0	4	3	2	2	...	True	False	False	True	False	True	
3	15	4	2	1	3	0	3	2	2	1	...	False	True	False	True	False	True	
4	16	3	3	1	2	0	4	3	2	1	...	True	False	False	True	False	True	

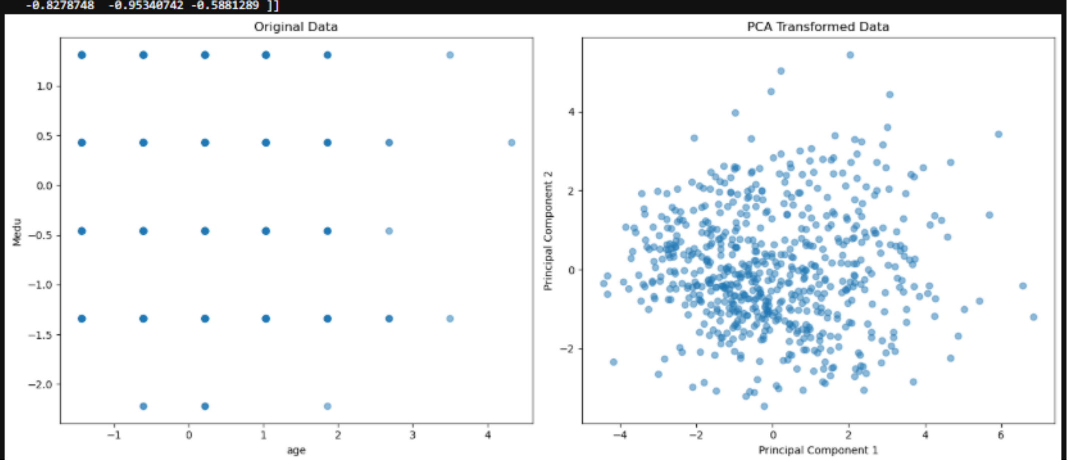
5 rows × 92 columns

Data reduction

The following code applies Principal component analysis to the dataset. It is a method used to reduce the number of features in a dataset while keeping as much useful information as possible. The code first fills missing values, encode categorical features into numeric values and scales numeric data for better processing. Then it applies PCA to transform the data and visualize the data suing scatter plots.

```
[54]: from sklearn.decomposition import PCA
data.fillna(data.mean(numeric_only=True), inplace=True)

cat_features = [feature for feature in data.columns if data[feature].dtype == 'O']
numeric_features = [feature for feature in data.columns if data[feature].dtype != 'O']

data = pd.get_dummies(data, columns=cat_features)

scaler = StandardScaler()
data[numeric_features] = scaler.fit_transform(data[numeric_features].values)

pca = PCA(n_components=15)
data_pca = pca.fit_transform(data)

print(data_pca.shape)
print(data_pca[:5])

plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
plt.scatter(data[numeric_features[0]], data[numeric_features[1]], alpha=0.5)
plt.title('Original Data')
plt.xlabel(numeric_features[0])
plt.ylabel(numeric_features[1])

pca = PCA(n_components=15)
data_pca = pca.fit_transform(data)

plt.subplot(1, 2, 2)
plt.scatter(data_pca[:, 0], data_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()
```

```
(649, 15)
[[ 1.24359735 -0.04607092  2.29412904  0.02160444  0.66754215  1.38800611
  -0.31697371 -1.20210798  0.5785898  -0.24840881 -1.86383449 -0.18984744
   0.34926162 -1.08081886  0.53148961]
 [ 0.44164659 -1.81425203 -0.08264077 -0.60818698  0.68359366 -0.42632998
   0.93183767  0.64078366  0.3674831  -1.12344162 -0.02326333  0.50582875
  -0.94526021 -0.53567439 -0.33745732]
 [-0.18795645 -0.78651758 -0.7517497   0.33103408 -1.24772806 -0.74453131
   1.39442798  1.41137353  0.35418199 -0.56069185  0.93464452  0.08569945
   0.07004942 -0.75035016  0.48735703]
 [-2.72543199 -0.75755663  1.36820591  0.2073205  -0.65117644 -0.31674116
  -0.97585709  1.3778649   0.09356464  0.42232039  0.27145735  0.1973103
   0.22284217  1.31035328 -0.62905355]
 [-1.28624073 -0.46395006  1.13075033 -0.55446889 -0.56698153 -0.91930784
  -0.24887859  0.50175092 -0.17670652  0.11062277 -0.26170716  0.38802489
  -0.8278748  -0.95340742 -0.5881289 ]]
```



```
[56]: type(data_pca)
      data_pca.ndim
      data_pca.shape

[56]: (649, 15)
```

Handling Imbalanced Data

The data is balanced using SMOTE by creating synthetic samples for the minority class. This help ensure fair training for machine learning models improving prediction accuracy and performance on imbalanced data.

```
[58]: from sklearn.preprocessing import StandardScaler, LabelEncoder
      from sklearn.decomposition import PCA
      from imblearn.over_sampling import SMOTE

      data.fillna(data.mean(numeric_only=True), inplace=True)

      cat_features = [feature for feature in data.columns if data[feature].dtype == 'O']
      numeric_features = [feature for feature in data.columns if data[feature].dtype != 'O']

      data = pd.get_dummies(data, columns=cat_features)

      scaler = StandardScaler()
      data[numeric_features] = scaler.fit_transform(data[numeric_features].values)

      if data['studytime'].dtype != 'int64' and data['studytime'].dtype != 'bool':
          data['studytime'] = (data['studytime'] > 0.5).astype(int)

      # Separate features and target
      X = data.drop(columns=['studytime'])
      y = data['studytime']
      if y.dtype == 'O':
          le = LabelEncoder()
          y = le.fit_transform(y)

      print(X.shape, y.shape)

      smote = SMOTE(random_state=42)
      X_resampled, y_resampled = smote.fit_resample(X, y)

      data_resampled = pd.concat([pd.DataFrame(X_resampled, columns=X.columns), pd.DataFrame(y_resampled, columns=['studytime'])], axis=1)
      data_resampled.head()

      (649, 58) (649,)
```

[58]:

	age	Medu	Fedu	traveltime	failures	famrel	freetime	goout	Dalc	Walc	...	activities_yes	nursery_no	nursery_yes	higher_no	hi
0	1.031695	1.310216	1.540715	0.576718	-0.374305	0.072606	-0.171647	0.693785	-0.543555	-0.997695	...	-0.971140	-0.495663	0.495663	-0.344914	
1	0.210137	-1.336039	-1.188832	-0.760032	-0.374305	1.119748	-0.171647	-0.157380	-0.543555	-0.997695	...	-0.971140	2.017502	-2.017502	-0.344914	
2	-1.432980	-1.336039	-1.188832	-0.760032	-0.374305	0.072606	-0.171647	-1.008546	0.538553	0.560678	...	-0.971140	-0.495663	0.495663	-0.344914	
3	-1.432980	1.310216	-0.278983	-0.760032	-0.374305	-0.974536	-1.123771	-1.008546	-0.543555	-0.997695	...	1.029717	-0.495663	0.495663	-0.344914	
4	-0.611422	0.428131	0.630866	-0.760032	-0.374305	0.072606	-0.171647	-1.008546	-0.543555	-0.218508	...	-0.971140	-0.495663	0.495663	-0.344914	

5 rows × 59 columns

The data set is resampled using random under sampler technique. It reduces the majority class to create the balanced dataset.

```
[60]: from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.decomposition import PCA
from imblearn.over_sampling import SMOTE


data.fillna(data.mean(numeric_only=True), inplace=True)

cat_features = [feature for feature in data.columns if data[feature].dtype == 'O']
numeric_features = [feature for feature in data.columns if data[feature].dtype != 'O']

data = pd.get_dummies(data, columns=cat_features)

scaler = StandardScaler()
data[numeric_features] = scaler.fit_transform(data[numeric_features].values)

if data['studytime'].dtype != 'int64' and data['studytime'].dtype != 'bool':
    data['studytime'] = (data['studytime'] > 0.5).astype(int)

X = data.drop(columns=['studytime'])
y = data['studytime']
if y.dtype == 'O':
    le = LabelEncoder()
    y = le.fit_transform(y)

print(X.shape, y.shape)

from imblearn.under_sampling import RandomUnderSampler

rus = RandomUnderSampler()
X_resampled, y_resampled = rus.fit_resample(X, y)

data_resampled = pd.concat([pd.DataFrame(X_resampled, columns=X.columns), pd.DataFrame(y_resampled, columns=['studytime'])], axis=1)
data_resampled.head()
```

```
(649, 58) (649,)
```

[60]:

	age	Medu	Fedu	traveltime	failures	famrel	freetime	goout	Dalc	Walc	...	activities_yes	nursery_no	nursery_yes	higher_no
261	0.210137	-0.453954	-1.188832	1.913468	-0.374305	-2.021678	-2.075896	-1.859711	-0.543555	-0.997695	...	1.029717	-0.495663	0.495663	-0.344914
534	-0.611422	1.310216	1.540715	-0.760032	-0.374305	0.072606	-2.075896	-1.008546	0.538553	2.119051	...	-0.971140	-0.495663	0.495663	-0.344914
185	-0.611422	-1.336039	-2.098682	0.576718	-0.374305	0.072606	-0.171647	-1.008546	-0.543555	-0.997695	...	1.029717	-0.495663	0.495663	-0.344914
34	-0.611422	0.428131	-0.278983	-0.760032	-0.374305	1.119748	0.780478	-0.157380	-0.543555	-0.997695	...	-0.971140	2.017502	-2.017502	-0.344914
264	0.210137	-0.453954	-0.278983	-0.760032	1.312667	-0.974536	-2.075896	-1.008546	-0.543555	-0.997695	...	-0.971140	2.017502	-2.017502	2.899275

5 rows × 59 columns

Splitting data

The data is splitter into training and testing sets. The code ensures 70% of dataset is used for training and 30%for testing setting a fixed random_state for reproducibility.

```
[64]: from sklearn.model_selection import train_test_split


X = data.drop('studytime', axis=1)
y = data['studytime']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
[66]: X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

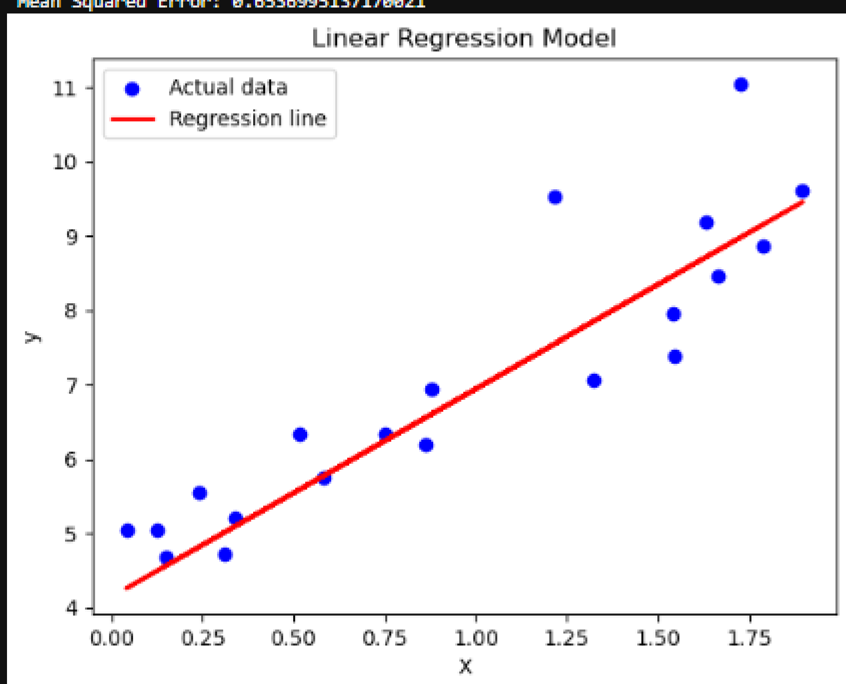
```
[66]: ((454, 58), (195, 58), (454,), (195,))
```

Regression model

The implementation of linear regression is done here. A linear regression model is trained used training data and predictions are made for test set. The code calculated mean squared error to measure how accurate the predictions are. Finally, a scatter plot is created to visualize the actual data points and fitted regression line.

```
[11]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

np.random.seed(42)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')

plt.scatter(X_test, y_test, color='blue', label='Actual data')
plt.plot(X_test, y_pred, color='red', linewidth=2, label='Regression line')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.title('Linear Regression Model')
plt.show()
```

```
Mean Squared Error: 0.6536995137170021
```



Evaluation metrices

Performance metrics like MSE, MAE, RMSE and R^2 score is calculated to evaluate how model fits the data.

```
[14]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

np.random.seed(42)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error (MSE): {mse}')
print(f'Mean Absolute Error (MAE): {mae}')
print(f'Root Mean Squared Error (RMSE): {rmse}')
print(f'R² Score: {r2}')
```

```
Mean Squared Error (MSE): 0.6536995137170021
Mean Absolute Error (MAE): 0.5913425779189777
Root Mean Squared Error (RMSE): 0.8085168605026132
R² Score: 0.8072059636181392
```

Conclusion

This project uses a dataset about secondary school students. The dataset includes 649 records and 33 columns. It contains details such as student age, gender, parental education, academic performance (grades G1, G2, G3), and other personal and lifestyle factors (like study time, alcohol consumption, and internet access). The project follows several key steps to clean and prepare this data and then applies a regression model to make predictions.

1. **Reading Data**: The first step is to load the data into the program using Python libraries like Pandas. Commands like head(), tail(), and info() help understand the basic structure of the data.
2. **Data Cleaning**: This step checks for missing or duplicate values. In this dataset, there were no missing values, but duplicates were removed to avoid repeating data that could affect the model.
3. **Outlier Detection and Removal:** Outliers are values that are too high or too low compared to the rest of the data. They can distort the model's accuracy. These were identified and removed to improve model performance.
4. **Data Transformation:** The data was scaled, which means all numerical values were adjusted to be in the same range. This helps the model understand the data better.
5. **One-Hot Encoding:** Since the dataset has categorical data like gender or school type, these were converted into numeric format using one-hot encoding. This allows machine learning algorithms to process these features properly.

6. **Data Reduction:** PCA was used to reduce the number of features while keeping important information. This step makes the model faster and can also improve accuracy.
7. **Handling Imbalanced Data:** If some categories have very few examples, the model can be biased. This step adjusts the data so that the model learns equally from all types of outcomes.
8. **Splitting the Data:** The data was divided into training and testing parts. The model learns from the training data and is tested on the test data to check how well it performs.
9. **Regression Model and Evaluation:** Regression is useful for predicting continuous values, like exam scores. After training the model, an evaluation metric was applied to check the model's performance. Common metrics include Mean Squared Error (MSE) or R^2 score, which show how accurate the predictions are.

Project 2

Obesity Classification using Machine Learning

Summary

The project is basically building of a machine learning model for obesity classification. The dataset provides information about individuals including their age, gender, height, weight etc. it classifies them into different weight categories i.e. underweight, overweight etc. machine learning models like randomforest etc are applied to automate obesity classification making it a tool for health research, medical assessment etc. Firstly, there is loading of dataset, then exploring it to understand its structure. The data is later split into two parts; one for training and the other for testing. After training the model makes predictions and performance is measure. Lastly a chart is created to show how many people are in each obesity category.

Objective

1. Analyze the data
2. Preprocess and clean the dataset
3. Converting categorical features
4. Selection of important features
5. Splitting the dataset
6. Building machine learning model
7. Training the model
8. Evaluating models performance
9. Visualizing the distribution
10. Demonstrating use of machine learning

Abstract

The project focuses on predicting obesity level in individuals using machine learning techniques. The dataset includes personal health data. The model achieves reliable results in classifying different obesity categories. The project demonstrates how machine learning can assist in early detection and classification of obesity which Is important for promoting better health outcomes. This kind of prediction analysis can help in early detection and better management of obesity which is major global health concern.

Steps involved

1. Importing libraries
2. Loading the dataset
3. Exploring the dataset
4. Encode categorical columns
5. Prepare features and target
6. Splitting into training and testing sets
7. Train the random forest classifier
8. Make predictions and evaluate the model

9. Visualize class distribution
10. Plot confusion matrix
11. Feature importance
12. Trying another classifier
13. ANN
14. Cross validation

Working

Importing libraries

- **pandas**: Helps manage and analyze data in tables, like an Excel spreadsheet.
- **Train test split**: Splits data into **training** (to learn) and **testing** (to check accuracy).
- **LabelEncoder**: Turns words (like "Male" and "Female") into numbers for machine learning models.
- **RandomForestClassifier**: A smart model that makes predictions based on many small decision trees.
- **classification_report & accuracy_score**: Shows how good your model is at making correct predictions.

```
[63]: import pandas as pd
      from sklearn.model_selection import train_test_split
      from sklearn.preprocessing import LabelEncoder
      from sklearn.ensemble import RandomForestClassifier
      from sklearn.metrics import classification_report, accuracy_score
```

Loading the dataset

```
[66]: df = pd.read_csv("Obesity Classification.csv")
```

Exploring the dataset

The head() function displays first few rows, you can also specify the rows that need to be displayed.

```
[26]: df.head()
```

[26]:

	ID	Age	Gender	Height	Weight	BMI	Label
0	1	25	Male	175	80	25.3	Normal Weight
1	2	30	Female	160	60	22.5	Normal Weight
2	3	35	Male	180	90	27.3	Overweight
3	4	40	Female	150	50	20.0	Underweight
4	5	45	Male	190	100	31.2	Obese

.info() provides summary of the data frame i.e. column names.

```
[28]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 108 entries, 0 to 107
Data columns (total 7 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   ID      108 non-null    int64
 1   Age     108 non-null    int64
 2   Gender  108 non-null    object
 3   Height  108 non-null    int64
 4   Weight  108 non-null    int64
 5   BMI     108 non-null    float64
 6   Label   108 non-null    object
dtypes: float64(1), int64(4), object(2)
memory usage: 6.0+ KB
```

.describe() provides descriptive statistics for numerical columns in data frame.

```
[30]: df.describe()
```

[30]:

	ID	Age	Height	Weight	BMI
count	108.000000	108.000000	108.000000	108.000000	108.000000
mean	56.046296	46.555556	166.574074	59.490741	20.549074
std	31.917939	24.720620	27.873615	28.856233	7.583818
min	1.000000	11.000000	120.000000	10.000000	3.900000
25%	28.750000	27.000000	140.000000	35.000000	16.700000
50%	56.500000	42.500000	175.000000	55.000000	21.200000
75%	83.250000	59.250000	190.000000	85.000000	26.100000
max	110.000000	112.000000	210.000000	120.000000	37.200000

Encode categorical columns

This code categorical data is represented as numbers allowing models to process it effectively while maintain relationship between different categories.

```
[32]: le_gender = LabelEncoder()
      df['Gender'] = le_gender.fit_transform(df['Gender'])

[34]: le_label = LabelEncoder()
      df['Label'] = le_label.fit_transform(df['Label'])
```

Prepare features and target

The code selects specific columns from a dataset an assign them to variables for further processing.

```
[36]: X = df[['Age', 'Gender', 'Height', 'Weight', 'BMI']]
      y = df['Label']
```

Splitting into training and testing sets

The data is splitter into training and testing sets.

```
[38]: X_train, X_test, y_train, y_test = train_test_split(
          X, y, test_size=0.2, random_state=42)
```

Train the random forest classifier

Random forest classifier builds multiple decision trees and combine their outputs to make final classification decision.

```
[40]: clf = RandomForestClassifier(n_estimators=100, random_state=42)
      clf.fit(X_train, y_train)

[40]:  RandomForestClassifier
       RandomForestClassifier(random_state=42)
```

Make predictions and evaluate the model

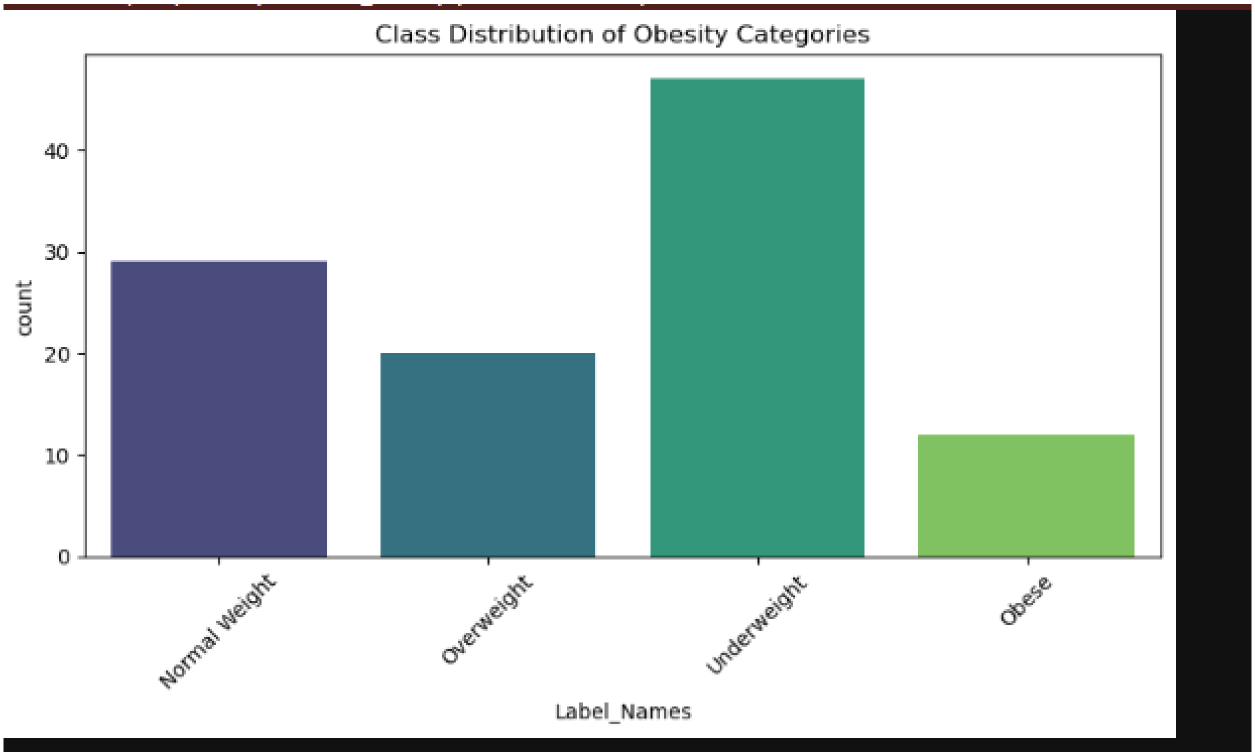
This code evaluates a model by predicting the test data and calculates its accuracy.

```
[42]: y_pred = clf.predict(X_test)
      print("Accuracy:", accuracy_score(y_test, y_pred))
      print("\nClassification Report:\n", classification_report(y_test, y_pred))

      Accuracy: 1.0

      Classification Report:
                     precision    recall  f1-score   support

                 0       1.00      1.00      1.00         6
                 1       1.00      1.00      1.00         4
                 2       1.00      1.00      1.00         4
                 3       1.00      1.00      1.00         8

          accuracy                           1.00        22
         macro avg       1.00      1.00      1.00        22
      weighted avg       1.00      1.00      1.00        22
```

Visualize class distribution

This code creates a count plot to visualize the distribution of obesity categories on a dataset using seaborn and matplotlib.

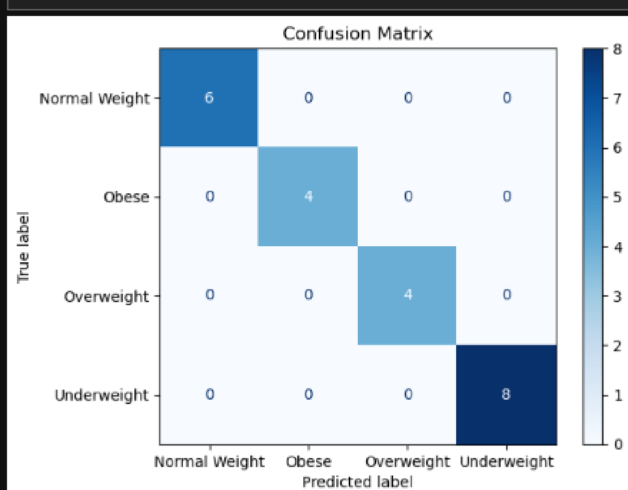
```
[44]: import seaborn as sns
      import matplotlib.pyplot as plt

      # Reverse encoding for better readability
      df['Label_Names'] = le_label.inverse_transform(df['Label'])

      # Countplot of obesity categories
      plt.figure(figsize=(8, 5))
      sns.countplot(data=df, x='Label_Names', palette='viridis')
      plt.title('Class Distribution of Obesity Categories')
      plt.xticks(rotation=45)
      plt.tight_layout()
      plt.show()
```



Plot confusion matrix

It creates a confusion matrix to evaluate a machine learning models performance. The confusion matrix helps analyze how well the model classifies different categories showing correct and incorrect predictions in a structured format.

```
[46]: from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=le_label.classes_)
disp.plot(cmap='Blues')
plt.title('Confusion Matrix')
plt.show()
```

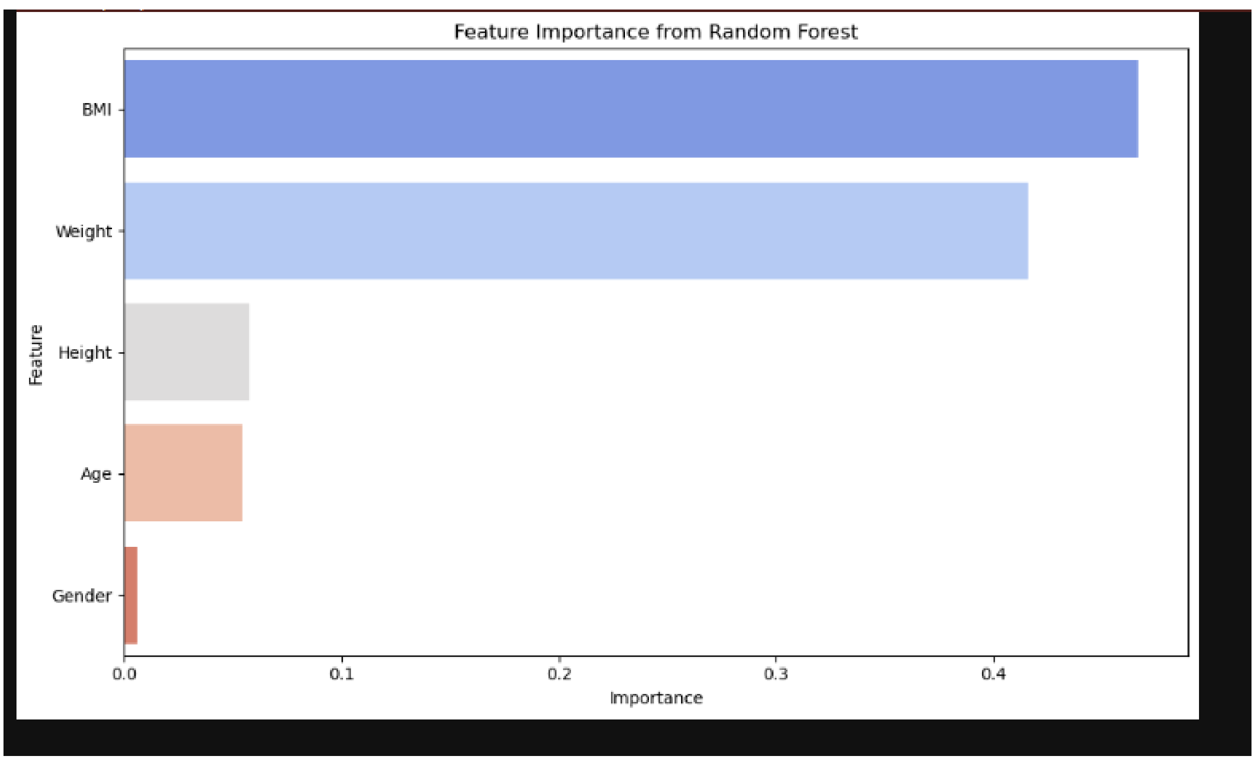


Feature importance

The following code extracts feature importance form a trained machine learning model and c=visualize the results using a bar plot. It helps understand which attributes have the most impact on model predictions.

```
[48]: import pandas as pd

      # Get feature names and their importances
      feature_names = X_train.columns
      feature_importances = clf.feature_importances_

      # Combine and sort
      importance_df = pd.DataFrame({
          'Feature': feature_names,
          'Importance': feature_importances
      }).sort_values(by='Importance', ascending=False)
```

```
[50]: import seaborn as sns
      import matplotlib.pyplot as plt

      plt.figure(figsize=(10, 6))
      sns.barplot(
          x='Importance',
          y='Feature',
          data=importance_df,
          palette='coolwarm'
      )
      plt.title("Feature Importance from Random Forest")
      plt.xlabel("Importance")
      plt.ylabel("Feature")
      plt.tight_layout()
      plt.show()
```



ANN

This code builds and trains a neural network model using tensor flow and scikit-learn for classification tasks. This approach is useful for making predictions on multi-class classification problems

```
[18]: import tensorflow as tf
      from tensorflow.keras.models import Sequential
      from tensorflow.keras.layers import Dense
      from sklearn.model_selection import train_test_split
      from sklearn.preprocessing import LabelEncoder, StandardScaler
      from tensorflow.keras.utils import to_categorical

      le = LabelEncoder()
      y_encoded = le.fit_transform(y)
      y_categorical = to_categorical(y_encoded)

      X_scaled = scaler.fit_transform(X)

      X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_categorical, test_size=0.2, random_state=42)

      model = Sequential()
      model.add(Dense(64, activation='relu', input_shape=(X_train.shape[1],)))
      model.add(Dense(32, activation='relu'))
      model.add(Dense(y_categorical.shape[1], activation='softmax'))

      model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
      model.fit(X_train, y_train, epochs=20, batch_size=32, validation_split=0.2)
```

```
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 159ms/step - accuracy: 0.3185 - loss: 1.3869 - val_accuracy: 0.5000 - val_loss: 1.3188
Epoch 2/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step - accuracy: 0.3987 - loss: 1.3111 - val_accuracy: 0.5000 - val_loss: 1.2723
Epoch 3/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step - accuracy: 0.4320 - loss: 1.2839 - val_accuracy: 0.5000 - val_loss: 1.2345
Epoch 4/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step - accuracy: 0.5416 - loss: 1.2432 - val_accuracy: 0.5556 - val_loss: 1.2027
Epoch 5/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.5880 - loss: 1.1787 - val_accuracy: 0.5556 - val_loss: 1.1717
Epoch 6/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step - accuracy: 0.5949 - loss: 1.1547 - val_accuracy: 0.5556 - val_loss: 1.1435
Epoch 7/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step - accuracy: 0.6135 - loss: 1.1234 - val_accuracy: 0.5556 - val_loss: 1.1177
Epoch 8/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step - accuracy: 0.6438 - loss: 1.0829 - val_accuracy: 0.5556 - val_loss: 1.0925
Epoch 9/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - accuracy: 0.6556 - loss: 1.0503 - val_accuracy: 0.5556 - val_loss: 1.0661
Epoch 10/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - accuracy: 0.6673 - loss: 1.0130 - val_accuracy: 0.5556 - val_loss: 1.0389
Epoch 11/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.6438 - loss: 1.0022 - val_accuracy: 0.5556 - val_loss: 1.0112
Epoch 12/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step - accuracy: 0.6282 - loss: 0.9770 - val_accuracy: 0.5556 - val_loss: 0.9854
Epoch 13/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.6477 - loss: 0.9426 - val_accuracy: 0.5556 - val_loss: 0.9631
Epoch 14/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step - accuracy: 0.6477 - loss: 0.9028 - val_accuracy: 0.5556 - val_loss: 0.9442
Epoch 15/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - accuracy: 0.6629 - loss: 0.8731 - val_accuracy: 0.6111 - val_loss: 0.9263
Epoch 16/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step - accuracy: 0.6326 - loss: 0.8569 - val_accuracy: 0.6111 - val_loss: 0.9093
Epoch 17/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step - accuracy: 0.6477 - loss: 0.8252 - val_accuracy: 0.6111 - val_loss: 0.8938
Epoch 18/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - accuracy: 0.6512 - loss: 0.8242 - val_accuracy: 0.6111 - val_loss: 0.8768
Epoch 19/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - accuracy: 0.6664 - loss: 0.8031 - val_accuracy: 0.6667 - val_loss: 0.8608
Epoch 20/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.6889 - loss: 0.7651 - val_accuracy: 0.6667 - val_loss: 0.8476
[18]: <keras.src.callbacks.history.History at 0x219c030a450>
```

This code evaluates performance of an ANN model using accuracy, confusion matrix and classification report. The results provide insights into model's precisions, recall and effectiveness.

```
[22]: from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

      # Predict probabilities and convert to binary class labels (0 or 1)
      y_pred_prob = model.predict(X_test)
      y_pred_ann = (y_pred_prob > 0.5).astype(int)

      # Flatten predictions and true labels to 1D arrays if necessary
      y_pred_ann = y_pred_ann.ravel()
      y_test = y_test.ravel()

      # Evaluation
      print("ANN Evaluation:")
      print("Accuracy:", accuracy_score(y_test, y_pred_ann))
      print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_ann))
      print("Classification Report:\n", classification_report(y_test, y_pred_ann))

      2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
      ANN Evaluation:
      Accuracy: 0.9407894736842105
      Confusion Matrix:
       [[110   4]
       [  5  33]]
      Classification Report:
                     precision    recall  f1-score   support

                 0       0.96      0.96      0.96       114
                 1       0.89      0.87      0.88        38

          accuracy                           0.94       152
         macro avg       0.92      0.92      0.92       152
      weighted avg       0.94      0.94      0.94       152
```

Trying another classifier

It demonstrates how to standardize feature data and train a logistic regression model.

```
]: from sklearn.preprocessing import StandardScaler
   from sklearn.linear_model import LogisticRegression

   scaler = StandardScaler()
   X_train_scaled = scaler.fit_transform(X_train)
   X_test_scaled = scaler.transform(X_test)

   lr = LogisticRegression(max_iter=2000)
   lr.fit(X_train_scaled, y_train)

]:   LogisticRegression
   LogisticRegression(max_iter=2000)
```

Cross validation

The code applies cross validation to evaluate random forest model. The result shows high mean accuracy of 0.97 indicating strong model performance.

```
[54]: from sklearn.model_selection import cross_val_score

      scores = cross_val_score(clf, X, y, cv=5)
      print("Random Forest Cross-Validation Accuracy Scores:", scores)
      print("Mean Accuracy:", scores.mean())

      Random Forest Cross-Validation Accuracy Scores: [0.86363636 1.         1.         1.         1.        ]
      Mean Accuracy: 0.9727272727272727
```

Conclusion

This project successfully demonstrates how machine learning ca be used to classify obesity levels on basic health information such as age, gender, height weight etc. by using a random forest classifier the model was able to achieve accurate predictions showing its effectiveness in handling classification problems in healthcare data. The visual analysis of obesity categories also provides helpful insights into distribution of health conditions in the dataset. Overall, this project proves that machine learning can play a valuable role in early detection and monitoring of obesity which can lead to a better health awareness and preventive care strategies. This approach no only aids in predicting obesity but also demonstrates how data driven tools can support healthcare decisions.