ML Assignment: Logistic Regression, SVM and KNN

Submitted by:
Arun Rimal - N264S646

Submitted To:
Shruti Kshirsagar
CS770 Machine Learning

10/30/2024

Contents

List of Tables

List of Figures

1. Introduction

Classification is one of the most widely used approaches in machine learning. It is widely applicable in the field of healthcare, finance, and social science. In the given industries categorical problems are very common and classifying data into different categories hold valuable insight. This project involves classification problem where we aim to predict BMI (Index) based on several characteristics: gender, height. and weight. Predicting such indices is important in classifying people.Groups related to health or other categorical outcomes that reflects physiology or behavior metric This projects uses three machine learning models for problem solving - Logistic Regression, Support Vector Machine (SVM), and K-Nearest Neighbors (KNN). Problem is to predict the BMI of people based on the features like gender, height and weight. So to solve this classification different model offers different strengths in handling classification problem, making them suitable for binary and multiclass problem.

- Logistic Regression: It is a common approach to solving classification tasks and is effective in interpreting the relationship between input features and class probabilities. This model can effectively capture the linear relationship among features to predict the classification problem. In our project, Logistic Regression tries to identify the linear relationship between gender, height and weight to predict the Index (or BMI).
- Support Vector Machine: SVM is another useful classification technique which uses boundary lines to separate the categories. It can also separate high-dimensional data using hyperplanes. SVM is effective in handling non-linear data using kernel trick, which allows for the transformation of data into higher dimensions. This makes SVM a suitable model for this classification task which holds the ability to separate the complex features. Also, SVM has the ability to address complex data efficiently using the kernel tree.
- K-Nearest Neighbors (KNN): KNN algorithm classifies instances based on its proximity with other instances in the cluster. This model is simple and non-parametric compared to another classification model. The KNN model performance is highly dependent on the feature scaling and the value of k (number of neighbors). In this particular problem height and weight are continuous attributes, and KNN is suitable in identifying class patters based on the physical proximity in feature space.

2. Methods

2.1. Datasets

The dataset used for the classification problem is the Body mass index dataset.
Dataset details are as follows:
Number of Instances: 500
Number of Attributes: 4 numeric/categorical data.
Attribute Information (in order):

Gender: Gender Male/Female.

Height: Height of the person.

Weight: Weight of the person.

Index: Body mass index of the person.

2.2. Methods for Exploratory Data Analysis

Exploratory Data Analysis(EDA) is used to understand data by summarizing and analyzing the important statistics of the dataset. It is done using both methods graphical and non-graphical techniques by visualizing the patterns. Some of the methods used are as follows

- **Statistics**: Mean, median, variance, percentiles. Using describe() method.

```
# Checking the statistics of dataset
df_bmi.describe()
```

- **Visualizations**: Using histograms, scatter plots, box plots, heatmaps, and other plots.

```
# List of features to plot
features = ['Height', 'Weight']
fig, axs = plt.subplots(1, 2, figsize=(8, 4))
axs = axs.flatten()
# Loop through each feature and plot its distribution
for i, feature in enumerate(features):
    sns.histplot(df[feature], bins=20, kde=True, color='
       orangered', ax=axs[i])
    axs[i].set_title(f'{feature} Distribution')
# Show plot
```

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

fig, axs = plt.subplots(1, 1, figsize=(5, 3))
# Distribution of the target variable
sns.countplot(x='Index', color= 'dodgerblue',data=df)
plt.title("BMI Distribution")
plt.xlabel("BMI (Index)")
plt.show()

# List of features
features = ['Height', 'Weight']
fig, axs = plt.subplots(1, 2, figsize=(8, 4))
axs = axs.flatten()
# Iterate over features to create a boxplot for each
   feature
for i, feature in enumerate(features):
    # Create a Seaborn boxplot
    sns.boxplot(x='Gender', y=feature, data=df, ax=axs[i])
    # Set titles and labels
    axs[i].set_title(f'{feature} by Gender')
    axs[i].set_xlabel('Gender')
    axs[i].set_ylabel(feature)
# Show plot
plt.tight_layout()
plt.show()
```

– **Missing Data**: Identifying and visualizing missing or null values. Using isnull(), sum() and other methods.

```
# To see the info of data
print(df.info())

# To check for nulls
print(df.isnull().sum())

# To check data types
print(df.dtypes)
```

– **Outlier Detection**: Identifying and visualizing outliers. Using box plots, Z-scores, Interquartile Range (IQR) and other methods.

```
# List of features
features = ['Height', 'Weight']
fig, axs = plt.subplots(2, 1, figsize=(12, 14))
axs = axs.flatten()
# Iterate over features to create a boxplot for each
for i, feature in enumerate(features):
    # Create a Seaborn boxplot
    # sns.boxplot(x='Gender', y=feature, data=df, ax=axs[i
       ])
    sns.boxplot(x='Index', y=feature, hue='Gender', data=df
       , ax=axs[i])
    # Set titles and labels
    axs[i].set_title(f'{feature} Distribution by Index and
       Gender')
    axs[i].set_xlabel('BMI Distribution (Index)')
    axs[i].set_ylabel(feature)
# Show plot
plt.tight_layout()
plt.show()
```

– **Correlation**: Analyzing the relation between features and target. Using correlation matrices, head map, scatter plots and other methods.

```
# Show Correlation between features
correlation_matrix = df.drop(columns=['Gender']).corr()
correlation_matrix

# Show the correlation plot heatmap
fig, ax = plt.subplots(figsize = (6,3))
sns.heatmap(correlation_matrix, annot=True, annot_kws = {'
   size': 12} )
```

– **Dimensionality Reduction**: Detecting similar or highly multi-collinear features and reducing them to a single feature. Using Techniques like Principal component analysis (PCA).

Correlation Matrix:

	Height	Weight	Index
Height	1.000000	0.000446	-0.422223
Weight	0.000446	1.000000	0.804569
Index	-0.422223	0.804569	1.000000

Table 2.1: Correlation Matrix of BMI Datasets

Barplot for BMI distribution is shown below:

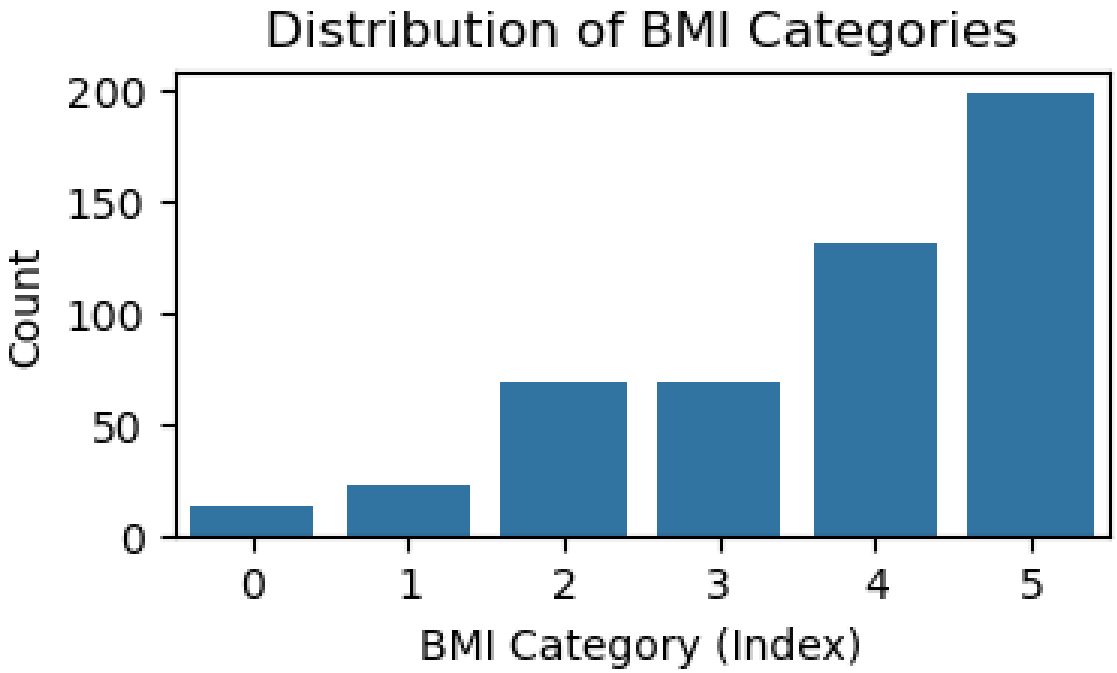

Figure 2.1: BMI Distribution

Correlation between features using heat map is shown below:

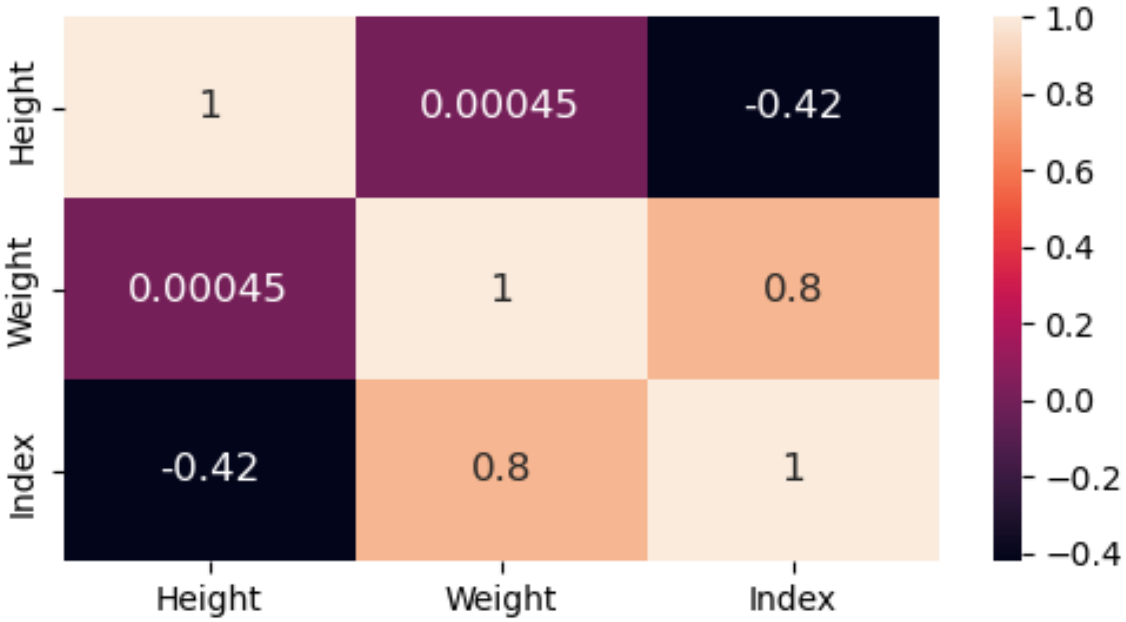

Figure 2.2: Correlation between Features and Target

Histogram for Height and Weight distribution of the BMI data set is shown below:

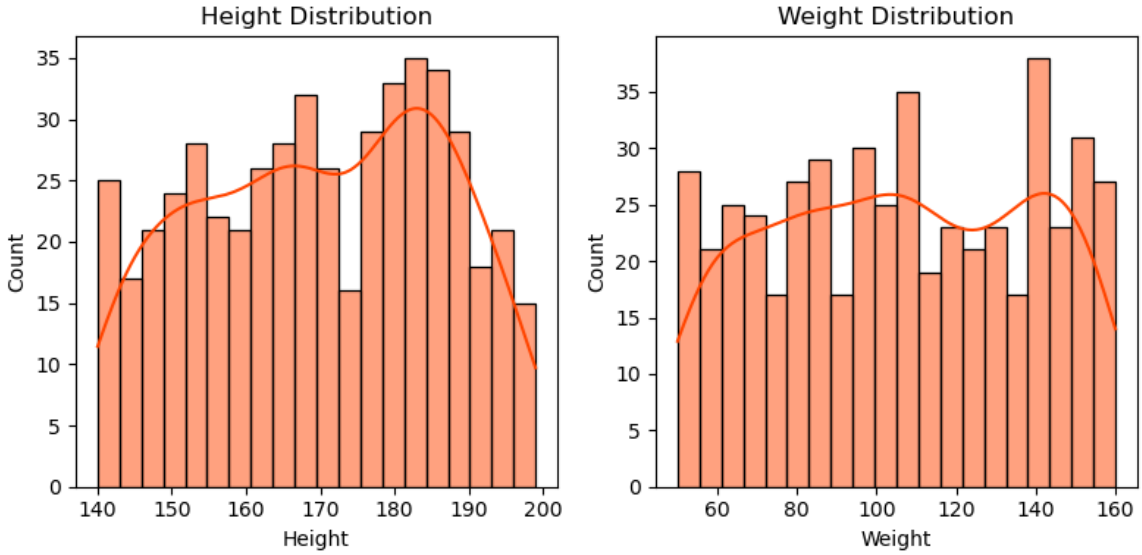

Figure 2.3: Height and Weight Distribution

Boxplot for height weight distribution by gender is shown below:

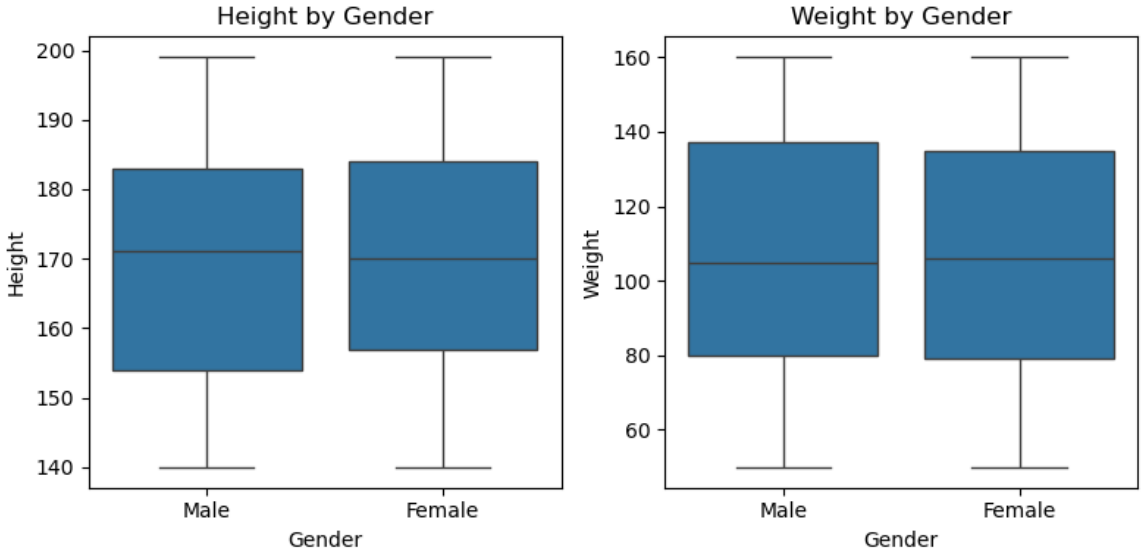

Figure 2.4: Height and Weight Distribution by Gender

Boxplot for Height and Weight distribution by BMI(Index) is shown below:

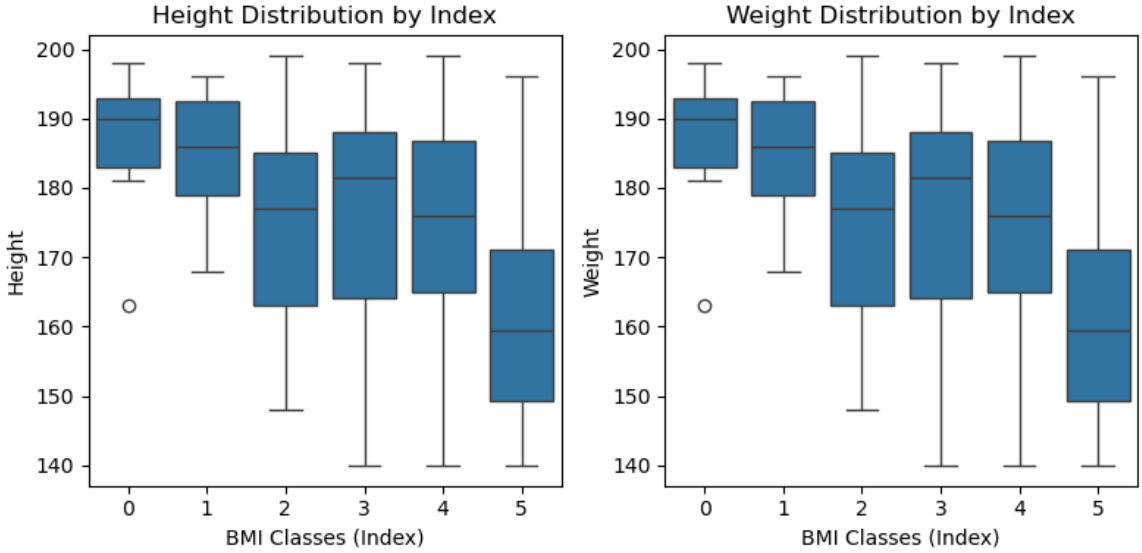

Figure 2.5: Height and Weight Distribution by BMI (Index)

Boxplot for Height and Weight distribution by BMI(Index) and Gender is shown below:

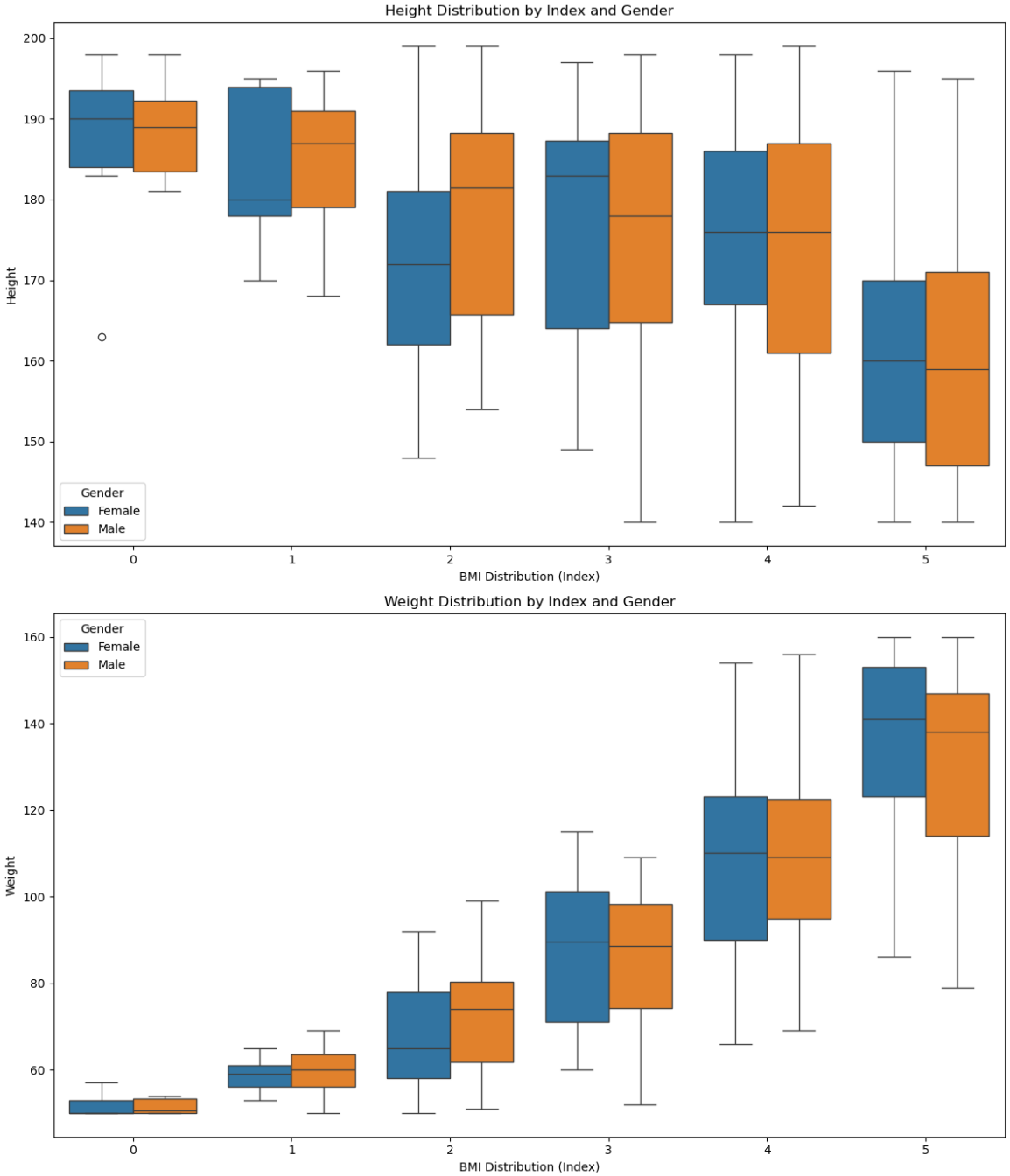

Figure 2.6: Height and Weight Distribution by BMI(Index) and Gender

2.3. Methods for Classification Problems Solution

Classification problems involve predicting a discrete label (or class) for an input sample. Steps involved in classification problem analysis are:

2.3.1. Preparation of Data

The first step involve is the preparation of data suitable for the model building. This includes data cleaning, feature scaling, feature engineering etc.

- The Loaded data from the csv file and convert it into dataframe

```
# Import Dataset
df = pd.read_csv('bmi.csv')
```

- Perform one-hot-encoding

```
# Convert 'Gender' to numerical format using one-hot
    encoding
df_encoded = pd.get_dummies(df, columns=['Gender']).astype
    (int)
```

- Split the data frame into feature and target variables.

```
# Split features (X) and target (y)
X = df_encoded[['Height', 'Weight', 'Gender_Female', '
    Gender_Male']]
y = df_encoded['Index']
```

- X and Y are split into train and test data.

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

- **Feature Scaling** is performed on the X train and test data.

```
# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

2.3.2. Sampling Technique SMOTE

Sampling Technique SMOTE is performed to address unbalanced data. The target variable in data is highly unbalanced so for good model building data must be balanced. So the following steps were performed.

```
from imblearn.over_sampling import SMOTE
smote = SMOTE()
X_resampled, y_resampled = smote.fit_resample(X, y)
```

Once the SMOTE sampling is performed on the data the target variable i.e. BMI is balanced across the classes as shown in the figure below.

Barplot of BMI Distribution Before and After SMOTE:

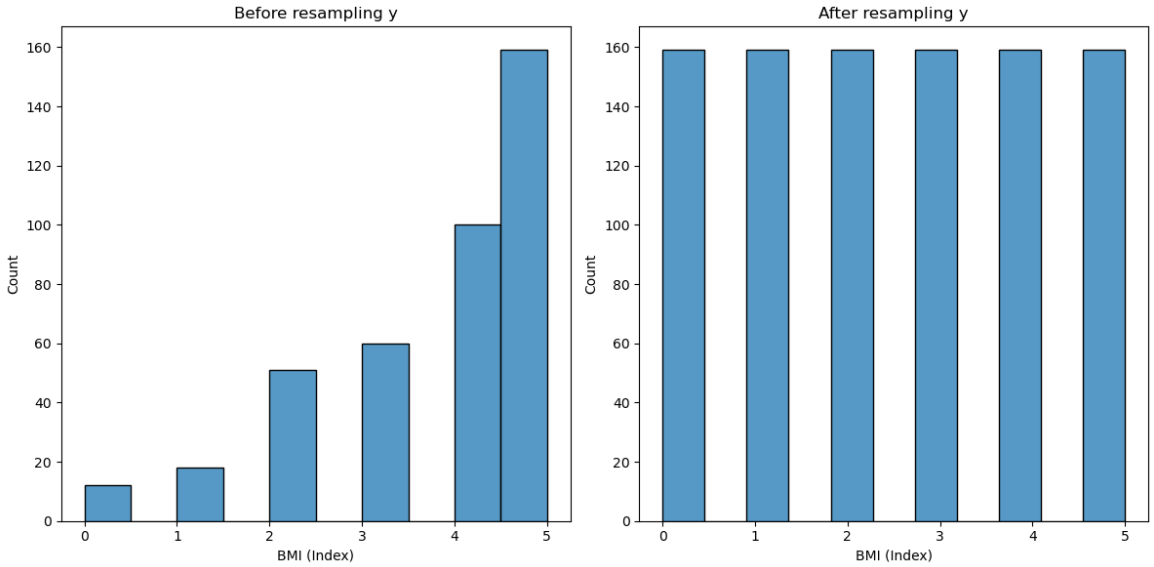

Figure 2.7: BMI Distribution Before and After SMOTE

Checking the gender level data balance, it is found that after SMOTE sampling data at the gender level were highly imbalanced. In general, SMOTE was unable to populate the Gender_Female and Gender_Male columns with the respective value of either 1 or 0 or vice versa. The pie chart below shows the percentages of missing gender assessments during SMOTE.

Piechart of Missing Gender Encoding(Assessment) After SMOTE:

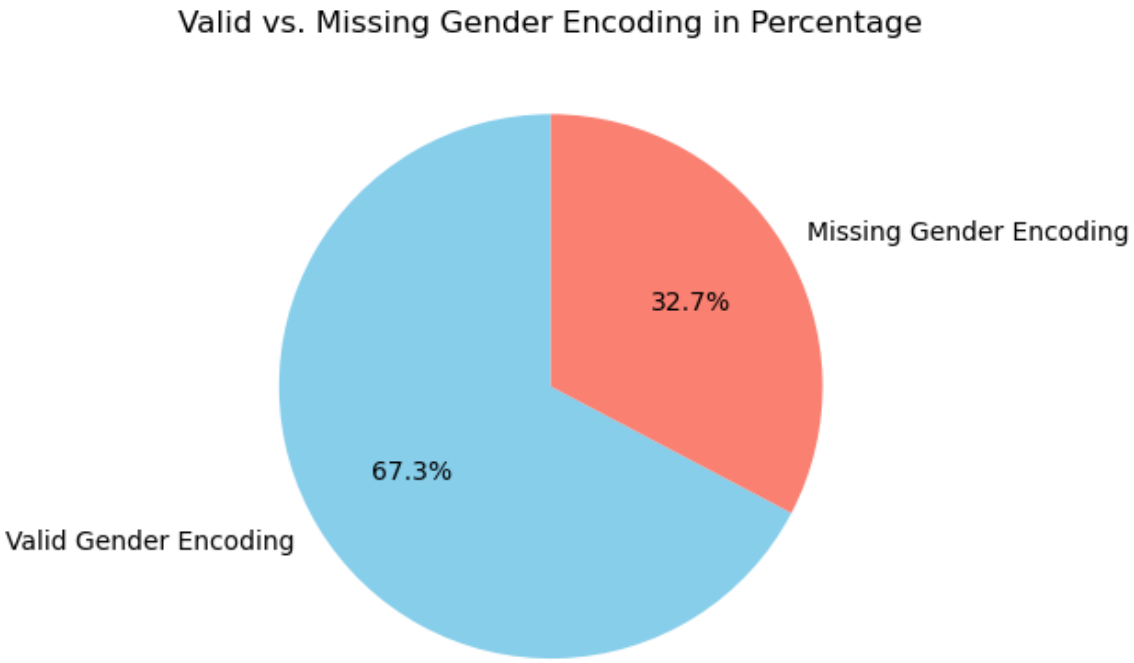

Figure 2.8: Gender Imbalance After SMOTE

The missing encoded values were divided equally into two for male and female gender. Then the missing values are updated to 1 for females (0 in the respective male column) and 1 for males (0 in the respective female column). The balance dataset is achieved on the gender level as well as shown in the figure below.

Barchart of Gender Distribution After Fixing:

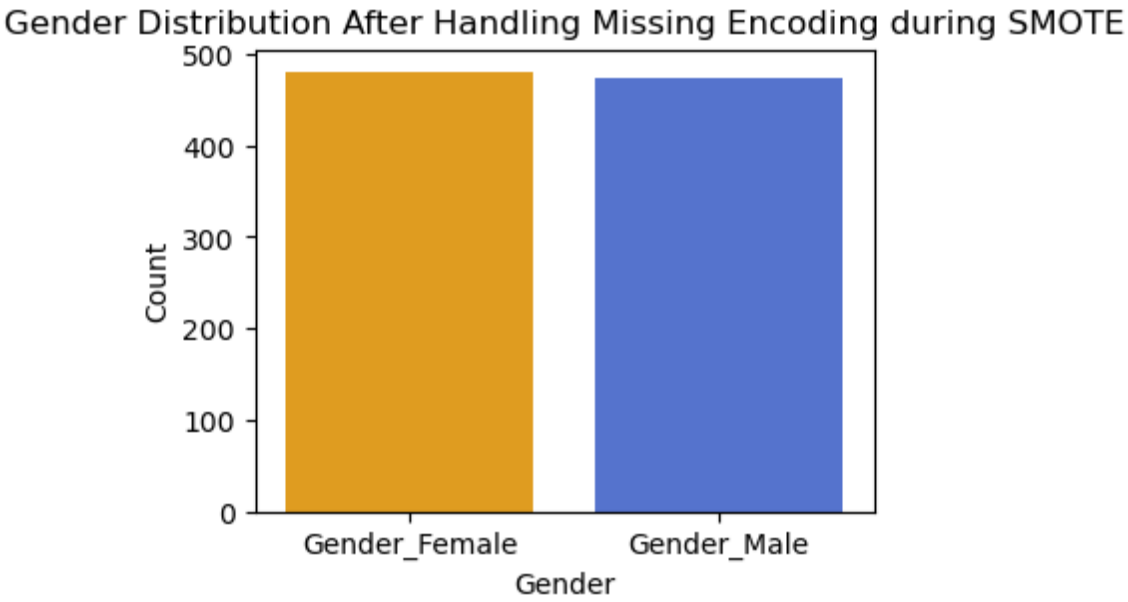

Figure 2.9: Gender Distribution After Fixing

2.3.3. Model building

There are several methods to handle classification problems. Each can be chosen based on data size, complexity, and class distribution. After data preparation model building is performed using various methods. During model building, Hyperparameter tuning is performed using the GridSearchCV library.

Logistic Regression is a method to build a model used to predict the probability of a category target variable based on other independent variables. Generally, it is popular to predict problems having binary outcomes (like yes or no). But it can also handle more categories by using techniques like One-vs-Rest (OvR) or Multinomial, making logistic regression used for multiple classes. Model building in the logistic regression is as follows:

```
# Initializing lr model
model = LogisticRegression()
param_grid_notBalanced = [
    param_grid = {
    'C': [1, 10, 100],
    'penalty': ['l1', 'l2'],
```

```
    'solver': ['lbfgs', 'saga'],
    'max_iter': [100, 200, 300]
}
]
grid_search = GridSearchCV(model, param_grid_notBalanced, cv=5,
    scoring='accuracy', verbose=1)
# Using optimized model from GridSearchCV to fit the training
   data
grid_search.fit(X_train_g, y_train_g)
# Evaluate the best model found by GridSearchCV on the test set
best_model = grid_search_notBalanced.best_estimator_
predictions = best_model.predict(X_test_g)
```

Support Vector Machine SVM is a model that use line to separate the categories among data by finding the best boundary (called a hyperplane) between them. It and handle two classes ans well as multiple classes by using different techniques to divide the data. It's useful for classification tasks and is known for working well even when data is not perfectly separated.

```
# Support Vector Machine (SVM)
param_grid_svm = {
'C': [0.1, 1, 10, 100],
'kernel': ['linear', 'rbf'],
'gamma': ['scale'] # 'scale' since it's usually effective with
   SVM
}


# Initializing Support Vector Machine (SVM) model
svm = SVC()
grid_search_svm = GridSearchCV(svm, param_grid_svm, cv=5,
   scoring='accuracy', verbose=1)
# Using optimized model from GridSearchCV to fit the training
   data
grid_search_svm.fit(X_train_resampled_s, y_train_resampled_s)
# Predict on the testing set
svm_pred = svm.predict(X_test)
```

K Nearest Neighbour KNN classify data based on the similarity to other points. It does find the boundary instead looking for the closest k neighbour of a new data point and assigning it to the category where most of its neighbors belong to. KNN works

well for both binary and multiclass classification and can handle complex, nonlinear boundaries based on how the data is arranged.

```
# K-Nearest Neighbors (KNN)
param_grid = {
    'n_neighbors': range(5, 10),
    'metric': ['euclidean', 'manhattan', 'minkowski'],
    'weights': ['uniform', 'distance']
}
knn = KNeighborsClassifier()
grid_search_knn = GridSearchCV(knn, param_grid, cv=5, scoring='
    accuracy')
# Using optimized model from GridSearchCV to fit the training
    data
grid_search_knn.fit(X_train_g, y_train_g)
# Evaluate the best model found by GridSearchCV on the test set
best_model = grid_search_knn.best_estimator_
y_pred_knn = best_model.predict(X_test_g)
```

2.3.4. Model Evaluation

Model evaluation is performed using evaluation matrices

```
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
# Output the accuracy
print(f"Accuracy: {accuracy}")
# Calculate precision, recall, and F1 score
precision = precision_score(y_test, y_pred, average='micro')
recall = recall_score(y_test, y_pred, average='micro')
f1 = f1_score(y_test, y_pred, average='micro')
# Output the precision, recall, and F1 score
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
# Evaluate the model performance
print(classification_report(y_test, y_pred))
```

3. Result and Discussion

3.1. Performance Evaluation Using Evaluation Matrices

When solving classification problems, it is important to understand a model's strengths and weaknesses and identify areas for improvement. Among several evaluation matrices are Accuracy, Precision, Recall, and F1 Score. These metrics, which provide insights into a model's ability to correctly classify instances, are as follows.

Accuracy is the ratio of correctly predicted instances to the total number of instances. It is a straightforward measure of overall performance. It is easy to interpret and provides a general overview of model performance. It is suitable for a balanced dataset and does not provide a good overview for imbalanced datasets. In an imbalanced dataset, accuracy is misleading because it generally provides the accuracy of the highest repeated class or the class which are in the majority.

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Instances}}$$

Precision is the ratio of correctly predicted positive observations to the total predicted positives. It measures the model's accuracy by identifying true positive instances.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

Recall, or sensitivity, is the ratio of correctly predicted positive observations to all actual positives. It measures the model's ability to capture all true positive instances.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

F1 Score is the harmonic mean of Precision and Recall. It balances the two metrics, providing a single metric that accounts for both false positives and false negatives.

$$\text{Recall} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{False Negatives} \times \text{False Negatives}}$$

3.2. Performance evaluation using Accuracy, Precision, Revall and F1 Score

3.2.1. Logistic Regression without SMOTE

Metric	Class Weight: None			Class Weight: Balanced		
	Accuracy	Precision (Micro)	Recall (Micro)	Accuracy	Precision (Micro)	Recall (Micro)
Value	0.92	0.92	0.92	0.86	0.86	0.86
Class/Instances	**Precision**	**Recall**	**F1 Score**	**Precision**	**Recall**	**F1 Score**
0 (1)	1.00	1.00	1.00	0.33	1.00	0.50
1 (4)	0.80	1.00	0.89	0.60	0.75	0.67
2 (18)	0.94	0.94	0.94	1.00	0.78	0.88
3 (8)	0.70	0.88	0.78	0.53	1.00	0.70
4 (30)	0.93	0.87	0.90	0.92	0.77	0.84
5 (39)	0.97	0.95	0.96	0.97	0.95	0.96
Overall Accuracy		0.92			0.86	
Macro Avg	0.89	0.94	0.91	0.73	0.87	0.76
Weighted Avg	0.93	0.92	0.92	0.86	0.86	0.86

Table 3.1: Logistic Regression Comparison without SMOTE

Accuracy: Logistic Regression model without balanced class weight has achieved an accuracy of 0.92. However, the model with balanced class weight drops to 0.86, indicating that the first model performs better overall.

Precision, Recall, F1-Score: LR with the same class weight to all the classes perform better. However, the second model with balanced class weight performs lower in most classes. The balanced model did not represent the minority class better than the model without balanced class weight.

Macro Average: The first model shows a higher macro average (0.89) compared to the balanced model's 0.73. This indicates a better balance across classes without applying weights.

Weighted Average: The first model performs slightly better with a weighted average of 0.93 compared to the balanced model of 0.91. First model reflects more consistent accuracy across all samples.

Summary: The Logistic Regression model without class balancing outperforms the balanced LR model. Particularly for minority classes, unbalanced model performs better by achieving higher precision and recall. So this shows that balancing weight is not required for this dataset.

Recommendation: The LR model without class balancing is performing better across all classes, especially in the minority classes as well. So first model without class balancing is better as per the figures.

3.2.2. Logistic Regression using SMOTE

Metric	Model (Class weight None)			Model(Class Weight Balanced)		
	Accuracy	Precision (Micro)	Recall (Micro)	Accuracy	Precision (Micro)	Recall (Micro)
Value	0.85	0.85	0.85	0.87	0.87	0.87
Class/Instances	**Precision**	**Recall**	**F1 Score**	**Precision**	**Recall**	**F1 Score**
0 (1)	0.50	1.00	0.67	0.50	1.00	0.67
1 (4)	0.80	1.00	0.89	0.80	1.00	0.89
2 (18)	0.89	0.89	0.89	0.94	0.89	0.91
3 (8)	0.43	0.75	0.55	0.47	0.88	0.61
4 (30)	0.91	0.70	0.79	0.92	0.73	0.81
5 (39)	0.97	0.95	0.96	1.00	0.95	0.97
Overall Accuracy		0.85			0.87	
Macro Avg	0.75	0.88	0.79	0.77	0.91	0.81
Weighted Avg	0.88	0.85	0.86	0.91	0.87	0.88

Table 3.2: Logistic Regression Comparison with SMOTE

Accuracy: The model with same class weightage performs with an accuracy of 0.85 and the model with balanced class weightage improves slightly to 0.87. This indicates stable performance across both models, with slight improvements on the second model.

Precision, Recall, F1-Score: Both models perform similarly across all classes but on class 3 (one of the minority class) model with a balanced weight class performs slightly better with better precision, recall and f1-score.

Macro Average: The macro average on the first model with the same class weightage is lower across all classes compared to the second model. This shows that the second model is slightly better to represent classes.

Weighted Average: weighted scores across all classes in the second model are slightly better than the first model.

Summary: Both the Logistic Regression models perform well, with slight improvements in precision and recall on the second model, particularly for Classes 3 and 4.

Recommendation: Both the models are similar in accuracy but the second model performs slightly better to represent at class level. So model with a balanced weight is better than the model without a balanced weight.

3.2.3. SVM without SMOTE vs with SMOTE

Metric	SVM Model Without SMOTE			SVM Model With SMOTE		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall
Value	0.93			0.92		
Class/Instances	**Precision**	**Recall**	**F1 Score**	**Precision**	**Recall**	**F1 Score**
0 (1)	1.00	1.00	1.00	1.00	1.00	1.00
1 (4)	0.80	1.00	0.89	1.00	1.00	1.00
2 (18)	0.94	0.94	0.94	0.95	1.00	0.97
3 (8)	0.88	0.88	0.88	0.64	0.88	0.74
4 (30)	0.96	0.87	0.91	0.93	0.83	0.88
5 (39)	0.93	0.97	0.95	0.97	0.95	0.96
Overall Accuracy		0.93			0.92	
Macro Avg	0.92	0.94	0.93	0.91	0.94	0.92
Weighted Avg	0.93	0.93	0.93	0.93	0.92	0.92

Table 3.3: SVM Model Comparison with and without SMOTE

Accuracy: The SVM model without SMOTE achieves an accuracy of 0.93 and the model with SMOTE has a slightly lower accuracy of 0.92. This difference indicates that the model without SMOTE performs slightly better overall.

Precision, Recall, F1-Score: Both models are competitive by giving perfect scores at the class level. The model without a smote has better performance in classes 3 and 4 where whereas a model with a smote shows perfect scores in classes 1 and class 2. The recall score for class 4 and class 5 is higher for models without smote sampling.

Macro Average: The macro average across classes for the model without smote is comparatively higher than the model with SMOTE sampling. This indicated that first model which is without SMOTE performs balanced across all classes.

Weighted Average: The weighted averages for precision, recall, and F1-score are quite sim-

ilar in both models, with the model without SMOTE achieving a weighted F1-score of 0.93 compared to 0.92 for the SMOTE model.

Summary: The SVM model without SMOTE performs better in most of the metrics compared to the model with SMOTE. However, for the minority classes, the model with SMOTE gives the perfect scores on the precision, recall and f-1 score. So overall second model performs better.

Recommendation: Although the SVM model without SMOTE maintains higher overall accuracy without resampling, the SVM model with SMOTE performs balanced performance across classes. So a model with SMOTE is recommended.

3.2.4. KNN without SMOTE vs with SMOTE

Metric	KNN Model Without SMOTE			KNN Model With SMOTE		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall
Value	0.90			0.93		
Class/Instances	**Precision**	**Recall**	**F1 Score**	**Precision**	**Recall**	**F1 Score**
0 (1)	1.00	1.00	1.00	1.00	1.00	1.00
1 (4)	0.80	1.00	0.89	1.00	1.00	1.00
2 (18)	0.94	0.89	0.91	0.95	1.00	0.97
3 (8)	0.73	1.00	0.84	0.88	0.88	0.88
4 (30)	0.92	0.80	0.86	0.93	0.87	0.90
5 (39)	0.93	0.95	0.94	0.93	0.95	0.94
Overall Accuracy		0.90			0.93	
Macro Avg	0.89	0.94	0.91	0.95	0.95	0.95
Weighted Avg	0.91	0.90	0.90	0.93	0.93	0.93

Table 3.4: KNN Model Comparison with and without SMOTE

Accuracy: The second model that is KNN with SMOTE has higher accuracy (0.93) compared to the first model KNN without SMOTE (0.90). This suggests that SMOTE enabled model performs better.

Precision, Recall, F1-Score: Both models has high scores on all the metrics of class but the model without SMOTE lags behind the model with SMOTE. A model with SMOTE has

perfect scores for many classes for recall and precision. Also, maintain higher F1 scores across classes.

Macro Average: The second model with SMOTE has a higher F1 score (0.95) compared to the first model without SMOTE (0.91). This indicates more balance and consistent scores across all classes.

Weighted Average: The second model with SMOTE weighted averages is also higher with 0.93 score than those of the non-SMOTE model with 0.90. This shows that SMOTE contributed positively to the model.

Summary: KNN model with SMOTE outperforms the model without SMOTE in accuracy, macro average, and weighted average scores. The performance across classes is better in the model with SMOTE sampling for this dataset.

Recommendation: The KNN model with SMOTE is preferable for this dataset. It provides higher accuracy and more balanced performance across classes, making it more suitable for datasets with class imbalance.

3.2.5. Gender Specific Model Training

Male gender-specific SVM model evaluation:

Metric	SVM Model Without SMOTE			SVM Model With SMOTE		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall
Value	0.98			0.98		
Class/Instances	**Precision**	**Recall**	**F1 Score**	**Precision**	**Recall**	**F1 Score**
0 (1)	0.50	1.00	0.67	0.50	1.00	0.67
1 (5)	1.00	0.80	0.89	1.00	0.80	0.89
2 (7)	1.00	1.00	1.00	1.00	1.00	1.00
3 (9)	1.00	1.00	1.00	1.00	1.00	1.00
4 (10)	1.00	1.00	1.00	1.00	1.00	1.00
5 (17)	1.00	1.00	1.00	1.00	1.00	1.00
Overall Accuracy		0.98			0.98	
Macro Avg	0.92	0.97	0.93	0.92	0.97	0.93
Weighted Avg	0.99	0.98	0.98	0.99	0.98	0.98

Table 3.5: SVM Model Comparison with and without SMOTE

Accuracy: Both models perform well for the male gender dataset. Both SVM have identical accuracy of approximately 0.98.

Precision, Recall, F1-Score: Most of the classes have perfect scores in both models. The scores are very identical on both models. So SMOTE did not have any impact on the performance metrics of the second model.

Macro Average and Weighted Average: Both the models have the same average score. This indicates that model performance is identical.

Summary: Both SVM models perform equally in terms of accuracy and class-specific metrics. SMOTE has no noticeable effect on performance.

Recommendation: Any model can be chosen but to reduce extra computational load model without SMOTE is recommended.

Male gender-specific KNN model evaluation:

Metric	KNN Model Without SMOTE			KNN Model With SMOTE		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall
Value	0.94			0.90		
Class/Instances	**Precision**	**Recall**	**F1 Score**	**Precision**	**Recall**	**F1 Score**
0 (1)	0.50	1.00	0.67	0.50	1.00	0.67
1 (5)	1.00	0.80	0.89	1.00	0.80	0.89
2 (7)	1.00	1.00	1.00	0.88	1.00	0.93
3 (9)	1.00	0.89	0.94	0.88	0.78	0.82
4 (10)	0.90	0.90	0.90	0.82	0.90	0.86
5 (17)	0.94	1.00	0.97	1.00	0.94	0.97
Overall Accuracy		0.94			0.90	
Macro Avg	0.89	0.93	0.89	0.84	0.90	0.86
Weighted Avg	0.95	0.94	0.94	0.91	0.90	0.90

Table 3.6: KNN Model Comparison with and without SMOTE

Accuracy: KNN without SMOTE has higher accuracy compared to KNN with SMOTE 0.94 and 0.90 respectively. This indicates that the non-SMOTE model performs better overall in terms of accuracy and correctly classifying instances in the dataset.

Precision, Recall, F1-Score: KNN without SMOTE has perfect scores across many classes and shows strong precision and recall across classes. KNN with SMOTE also performs well but less compared to the previous model.

Macro Average and Weighted Average: KNN without SMOTE has a better average and weighted score across all classes. This means the non-SMOTE model performs better across classes. achieving higher averages.

Summary: KNN without SMOTE outperforms the model with SMOTE in both accuracy and class-specific metrics.

Recommendation: Given the superior performance of the KNN model without SMOTE, it is the recommended approach for this dataset.

Female gender-specific SVM model evaluation:

Metric	SVM Model Without SMOTE			SVM Model With SMOTE		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall
Value	0.90			0.92		
Class/Instances	**Precision**	**Recall**	**F1 Score**	**Precision**	**Recall**	**F1 Score**
1 (2)	0.00	0.00	0.00	1.00	1.00	1.00
2 (6)	0.56	0.83	0.67	0.62	0.83	0.71
3 (10)	1.00	0.80	0.89	1.00	0.70	0.82
4 (12)	0.92	1.00	0.96	0.92	1.00	0.96
5 (21)	1.00	1.00	1.00	1.00	1.00	1.00
Overall Accuracy		0.90			0.92	
Macro Avg	0.70	0.73	0.70	0.91	0.91	0.90
Weighted Avg	0.89	0.90	0.89	0.94	0.92	0.92

Table 3.7: SVM Model Comparison with and without SMOTE

Accuracy: The SVM model with SMOTE has an accuracy of 0.92 and the model without SMOTE has an accuracy of 0.90. This suggests that the SMOTE model offers minor improvements in overall performance.

Precision, Recall, F1-Score: The model with SMOTE performs better when recognizing the instances of classes. Some of the classes have perfect scores under the SMOTE model. Whereas the non-SMOTE model struggles to recognize the variability in classes.

Macro Average and Weighted Average: The non-SMOTE model has macro and weighted averages of 0.70 and 0.89 respectively. SMOTE raises these averages to 0.91 and 0.92. This indicates enhanced performance across all classes.

Summary: The SVM model with SMOTE outperforms SVM without SMOTE. SMOTE model has higher accuracy, precision and recall. SMOTE makes the model more balanced and effective in handling imbalanced data.

Recommendation: For this dataset, the SVM model with SMOTE is recommended as it improves recognition of minority classes without sacrificing accuracy, making it more reliable overall.

Female gender-specific KNN model evaluation:

Metric	KNN Model Without SMOTE			KNN Model With SMOTE		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall
Value	0.84			0.90		
Class/Instances	**Precision**	**Recall**	**F1 Score**	**Precision**	**Recall**	**F1 Score**
1 (2)	0.00	0.00	0.00	1.00	1.00	1.00
2 (6)	0.50	0.83	0.62	0.71	0.83	0.77
3 (10)	1.00	0.60	0.75	1.00	0.70	0.82
4 (12)	0.80	1.00	0.89	0.80	1.00	0.89
5 (21)	1.00	0.95	0.98	1.00	0.95	0.98
Overall Accuracy		0.84			0.90	
Macro Avg	0.66	0.68	0.65	0.90	0.90	0.89
Weighted Avg	0.85	0.84	0.83	0.92	0.90	0.90

Table 3.8: KNN Model Comparison with and without SMOTE

Accuracy: The KNN model with SMOT achieves an accuracy of 0.90 whereas the non-SMOTE model is only 0.84. This suggests that SMOTE enables the model to learn and improve the classification accuracy.

Precision, Recall, F1-Score: The SMOTE model has a perfect score on class 0 and a high score on other minority classes. Whereas the non-SMOTE model has zero precision, recall and F1-score which indicates that the model has failed to identify the instances of minority classes.

Macro Average and Weighted Average: While calculating the average and weighted average of the classes SMOTE enabled model has outsmarted the non-SMOTE model. Using SMOTE sampling has improved the performance across all classes.

Summary: The KNN model with SMOTE provides better support for minority classes.

Recommendation: The model with SMOTE is preferred over the non-SMOTE model because the SMOTE model has a better ability to handle minority classes with high accuracy.

4. Conclusion

Several models are trained on the given dataset. Models like Logistic Regression, Support Vector Machine, and K-Nearest Neighbors are used to search the optimum classification model. Since the classes are highly imbalanced in this dataset, SMOTE was done in order to balance the class distributions. It was an important step in developing a more uniform dataset for training. The hyperparameters tuning were performed to optimized the models, hence giving better scores of their performance. Overall results using SMOTE were mixed; at times greatly benefiting the performance of the models on the evaluation metrics, while at times similar or even better performances of the models were shown without SMOTE. It is crucial to know whether the data is balanced or not after the SMOTE sampling. In this dataset SMOTE was not able to correctly assign the gender values in the respective gender field which created a noisy data in the dataset. In such scenario it is always better to address such noisy data. The result indicated that:

- The Logistic Regression model demonstrates commendable performance without the need for SMOTE by achieving high accuracy. SMOTE do not reduces the performance but neither increases it significantly, it is advisable to refrain from using it if computational efficiency is a priority. However, to reduce the skewness of the dataset balancing the dataset is recommended to reduce the biasness in the accuracy. SMOTE is recommended.

- The SVM model without SMOTE demonstrates superior performance across most metrics; however, for minority classes, the model with SMOTE achieves perfect precision, recall, and F1 scores. Therefore, the SVM with SMOTE is recommended for balanced performance across classes however the non-SMOTE model maintains a slightly higher overall accuracy. In some comparisons, both SVM models, with and without SMOTE, exhibit equal performance in terms of accuracy and class-specific metrics. This indicates that SMOTE has no significant effect. So, either model is suitable for the given dataset. If the performance is the same then the model selection can be done based on omputational load to the resources.

- K-Nearest Neighbors (KNN) Models: The KNN model with SMOTE shows improvised results in terms of accuracy, macro and weighted average scores. It also offered balanced performance across classes. Therefore, the KNN model with SMOTE is recommended for this dataset. But in some instances KNN without SMOTE also shows better performance. So It is recommended to choose the model as per the requirement criteria.

- When comparing the general model and gender-specific model, it can be seen that the performance metrics in gender-specific model are better compared to the general model. This is because in gender-specific model there is feature reduction and model has to fit less dimensional metrix. This reduces the complexity of model and becomes better fit on the data.