BIRMINGHAM CITY University

CMP6202

Artificial Intelligence & Machine Learning

2024–2025

Predicting the presence of diabetes mellitus using supervised learning models

Lewis Higgins - Student ID 22133848

Module Coordinator: Nouh Elmitwally

Word count excluding figures, tables and appendices: 4,379

Contents

Abstract

This report presents a comprehensive study on predicting the presence of diabetes mellitus using supervised learning models. The research integrates two datasets, the Pima Indian Diabetes Database and a Frankfurt diabetes dataset, resulting in a combined dataset of 2,768 samples. Extensive exploratory data analysis revealed key insights into feature relationships and data quality issues, which were addressed through preprocessing techniques including outlier handling, missing data imputation with KNN, and class balancing with SMOTE. Five classification algorithms were developed and evaluated: Random Forest, Support Vector Machine, Logistic Regression, Naïve Bayes, and K-Nearest Neighbours. The models had an initial iteration with reasonably good performance, which was further improved in a second iteration of each model using stratified cross-validation and hyperparameter tuning. K-Nearest Neighbours emerged as the top-performing model, achieving 97.8% accuracy, 96.2% recall, and an F1-score of 96.6 on the unseen test set, with this performance closely followed by Random Forest. The study demonstrates the potential of machine learning in the medical field, while also highlighting the importance of proper data preprocessing and model optimization techniques.

Introduction

Type 2 diabetes, or diabetes mellitus, accounts for 90% of the 4.4 million cases of diabetes in the UK, with estimations that there are 1.2 million undiagnosed cases of the condition across the country (Diabetes UK, 2024a). The condition's occurrence per 100,000 individuals is rapidly increasing, with M. A. B. Khan et al. (2020)'s analysis projecting that by 2030, the rate will reach 7,079 per 100,000. Many people with diabetes suffer immensely reduced quality of life, with approximately 50% of patients suffering from peripheral neuropathy (Dhanapalaratnam et al., 2024), an irreversible disability causing immense pain due to nerve damage from high blood sugar (NHS, 2022), which can occur when the patient was unaware they even had diabetes.

Therefore, it is imperative that systems are put in place to enable the swift diagnosis of diabetes mellitus. This can be accomplished by training machine learning models on existing clinical datasets to identify common trends in those with and without the condition. This report will document the planning, development and evaluation of five machine learning models to classify diabetes mellitus based on multiple clinical factors, specifically through the stages of:

- Dataset Identification
- Data Integration
- Data Preprocessing
- Exploratory Data Analysis (EDA)
- Model Development
- Model Evaluation
- Research Conclusions

1.1 Dataset Identification

Machine learning models require lots of training data, meaning a dataset must be identified consisting of many rows and features. Two datasets were identified which could be integrated into one larger dataset, the first of which being the Pima Indian Diabetes Database (UCI Machine Learning, 2024), downloaded from Kaggle. The data originates from the National Institute of Diabetes and Digestive and Kidney Diseases, who collected this data from Pima Indian[1] women aged 21 and over in hospitals in Phoenix, Arizona, USA. It has seen wide use across academic literature relating to machine learning where other researchers have also performed diabetes classification via supervised learning (AlZu'bi et al., 2023; Zou et al., 2024; Joshi and Dhakal, 2021; Hayashi and Yukita, 2016). This dataset contains 768 rows with 9 features.

The second dataset is also from Kaggle, and has been previously used in literature by Zou et al. (2024). This dataset (John DaSilva, 2024) is based on data from female patients in Frankfurt, Germany, and includes the same 9 features as the Pima Indian dataset, but with 2000 samples. By integrating these two datasets into one larger dataset of 2768 rows, it will be possible to give the machine learning models more data to train upon.

Table 1.1 details the 9 features seen in both datasets and their descriptions.

Feature	Description	Measurement
Pregnancies	The number of pregnancies the patient has had.	Ratio
Glucose	Plasma glucose concentration over 2 hours in an oral glucose tolerance test.	Ratio
BloodPressure	Diastolic blood pressure in mm/Hg.	Ratio
SkinThickness	Triceps skin fold thickness (mm)	Ratio
Insulin	2-hour serum insulin.	Ratio
BMI	Body Mass Index, calculated from the patient's weight and height.	Ratio
DiabetesPedigree-Function	The product of a function to ascertain the probability of diabetes based on family genetics. (Akmeşe, 2022)	Ratio
Age	The patient's age.	Ratio
Outcome	Whether the patient is likely to develop diabetes.	Nominal

Table 1.1: The features seen in both datasets.

[1] "Pima Indian" refers to a specific Native American ethnic group rather than people from India.

1.2 Supervised learning task identification

Because patients can have diabetes without knowing, it is paramount that swift and simple diagnosis methods are put in place, which can be achieved through the use of binary classification models. This requires the existence of the "ground truth": the label given to data that indicates its class (c3.ai, 2024). Within these datasets, the ground truth is present as the 'Outcome' feature, which was used as the target variable for the models.

Exploratory Data Analysis

2.1 Data Integration

The two datasets must first be merged into one to allow for an overall analysis to be performed. This was a simple process because they both contain the same 9 features.

```
pima_df = pd.read_csv("Data/pima.csv")
pima_df.shape
```
✓ 0.0s
```
(768, 9)
```

```
frankfurt_df = pd.read_csv("Data/frankfurt.csv")
frankfurt_df.shape
```
✓ 0.0s
```
(2000, 9)
```

```
df = pd.concat([pima_df, frankfurt_df], axis = 0, ignore_index = True)
df.shape
```
✓ 0.0s
```
(2768, 9)
```

Figure 2.1: Integrating the two separate datasets into one larger dataset.

2.2 Question identification and assumptions

The key factors involved in the diagnosis of diabetes are critical to understand, which can be solved through EDA on these datasets. It is possible to make various assumptions based on topical background research of each of the features in the dataset, detailed in Table 2.1

Feature	Research-based assumptions
Pregnancies	Approximately 13.4% of pregnant women develop a temporary condition known as Gestational Diabetes Mellitus (GDM), which typically subsides after birth (Adam et al., 2023). However, research by Dennison et al. (2021) indicates that 33% of women who develop GDM will go on to develop permanent diabetes mellitus within 15 years. Therefore, it is assumed that pregnancies will positively correlate with the diabetes outcome. It is also expected that pregnancies should naturally positively correlate with age.
Glucose	Glucose concentrations are an enormous factor in the diagnosis of diabetes mellitus, being one of the main metrics used to certify the condition, where results over 200mg/dL mean an absolute diagnosis[1] (Aftab et al., 2021). It is therefore assumed that the glucose concentrations will be one of the strongest influences of the outcome, and that it will also correlate heavily with insulin levels.
BloodPressure	Diastolic blood pressure (DBP) does influence the diagnosis of diabetes mellitus, as 56.2% of recently diagnosed patients presented with elevated DBP in Nelaj et al. (2023)'s limited study of 126 patients, but it is not a decisive factor by itself. Therefore, it is assumed that there will be some correlation between DBP and the outcome, but not as major as other factors like plasma glucose levels.
SkinThickness	It is a frequent assumption even non-academically that people who weigh more, and by consequence have higher skin thickness in certain areas such as the triceps, have a higher risk of developing conditions like type 2 diabetes. This is backed by a study by Ruiz-Alejos et al. (2020), which found strong associations between skin thickness and diabetes mellitus, as well as high blood pressure. Therefore, it is assumed that there will be a strong correlation between tricep skin thickness and the outcome, as well as an expectation of strong correlations between thickness, BMI and blood pressure.
Insulin	Diabetes mellitus is directly associated with insulin deficiency, and as such, it is assumed that this factor will be the strongest influence in the outcome. This is because 2-hour serum insulin tests, as used in this dataset, are frequently part of HOMA-IR[2] assessments.

[1]The other main metric is insulin deficiency, meaning that the patient could have glucose levels lower than 200mg/dL and still be diagnosed if they are instead insulin deficient. (Aftab et al., 2021).

[2]Homeostasis Model Assessment of Insulin Resistance, used to measure insulin resistance (Tahapary et al., 2022), which can be used in both type 1 and type 2 diabetes diagnosis (Khalili et al., 2023).

BMI	BMI is likely to be a significant factor in the outcome, which is backed by previous academic studies indicating that 71% of studied individuals showed increases in BMI prior to diagnosis (Donnelly et al., 2024). Additionally, BMI is used in insulin resistance measurement assessments, which are key assessments in diabetes diagnosis, meaning that it is a safe assumption that BMI will be a large factor in the outcome.
DiabetesPedigree-Function	People are more likely to develop diabetes mellitus if there is a family genetic history of the condition, though it is not directly caused by any one particular gene (Diabetes UK, 2024b). With the pedigree function aiming to quantify the inheritance probability, it can be assumed that it will likely correlate heavily with the outcome.
Age	Suastika et al. (2012) studied the effects of age as a risk factor for diabetes mellitus, finding that many natural associated factors of ageing including increases in body fat and decreases in lipid metabolism had considerable influence on the development of insulin resistance and diabetes mellitus by consequence. Therefore, it is likely that there will be a noticeable correlation between a patient's age and the outcome.

Table 2.1: Research-based assumptions prior to any EDA.

Based on these assumptions, the questions that this EDA process aims to answer are:

ID	Research-based assumptions
1	Are there any missing values or values that are not physically possible?
2	Are there any significant outliers?
3	Is the dataset evenly balanced in terms of the outcome? If not, what should be done?
4	Does the rate of diabetes positively correlate with the amount of pregnancies a woman has had?
5	Does the amount of pregnancies influence any of the other features?
6	What is the distribution of blood glucose levels in patients with and without diabetes?
7	Does BMI influence glucose levels?
8	Is diastolic blood pressure a worthwhile diagnosis method in this dataset?
9	Is the average skin thickness of those with diabetes actually higher than those without?
10	How does the relationship between insulin and glucose change between those with and without diabetes?

Table 2.2: The questions that this EDA process aims to answer.

2.3 Splitting the dataset

It is good practice to first split the data into training and testing sets before performing exploratory data analysis to avoid conceptual overfitting, also known as data leakage. Conceptual overfitting occurs when insights gained from the entire dataset influence model development decisions, which may eventually lead to actual overfitting. By excluding the training data from the analysis, it effectively simulates a real-world environment where the data being given to the model is not known, even to its developers.

Splitting the data is a mandatory process when developing supervised learning models, primarily for the prevention of overfitting. Overfitting is a significant challenge in machine learning, where models can perform exceptionally well on their original training data but are unable to generalize to unseen data, making them unsuitable for deployed use. The training set must consist of a large portion of the data so that the model has enough information to analyse and discover trends within, whereas the testing set is a smaller, unseen remainder of the data that the model's predictions can be evaluated against using various metrics. A key point of determination is the proportions of the dataset that should go in each set - there is no 'one-size-fits-all' percentage that can provide the best results for every possible dataset (Sivakumar et al., 2024), and factors such as the size of the dataset play a large part in this. Most commonly, splits are either 70:30 or 80:20 for training and testing sets respectively.

The integrated dataset for this project is 2,768 rows. This is considered to be a small dataset, and as such, it will be best to maximize the size of the training split, so a split of 80% training and 20% testing was used, visualized in Figure 2.2.

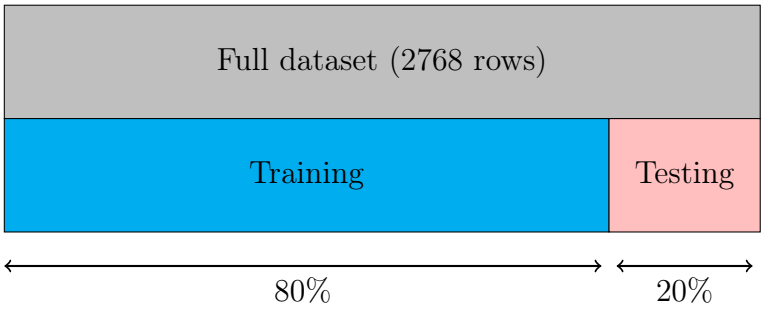


Figure 2.2: A visual representation of the train/test split.

To accomplish this, the data must be split into X and y tables, where X consists of the eight features, and y is the target variable. After the data is split to X and y, it can be split into training and testing sets through Scikit-Learn's "train_test_split" method, as depicted in Figure 2.3

```
X = df.drop(columns = "Outcome", axis = 1)
y = df["Outcome"]
✓ 0.0s

print(X.shape)
print(y.shape) # No columns because y is now a Series consisting only of the Outcome column.
✓ 0.0s
(2768, 8)
(2768,)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
✓ 0.0s

print(f"X_train: {X_train.shape}")
print(f"y_train: {y_train.shape}")
print(f"X_test: {X_test.shape}")
print(f"y_test: {y_test.shape}")
✓ 0.0s
X_train: (2214, 8)
y_train: (2214,)
X_test: (554, 8)
y_test: (554,)
```

Figure 2.3: Splitting the data at an 80:20 ratio.

By default, this method will shuffle all rows in the dataset before splitting it, which introduces an element of randomness, damaging reproducibility. To combat this, the "random_state" parameter was set to ensure that the same shuffle will occur every time.

When performing EDA, the Outcome column will be necessary to the analysis, so a deep copy[3] of X_train was made with y_train (the Outcome column) being added to it, merging the two back into a full training set, shown in Figure 2.4.

```
# It's important to make a deep copy, so that none of the data is tampered with.
fullTrainSet = X_train.copy(deep = True)
fullTrainSet["Outcome"] = y_train.values
✓ 0.0s
```

Figure 2.4: Duplicating X_train and adding the Outcome column for EDA.

[3]A complete copy rather than a pointer to the original data

2.4 EDA process and results

The full EDA process and results can be found in Appendix A. They were performed using the Seaborn Python library, which produces intepretable visualisations to answer each question posed in Table 2.2. The topics and issues identified during the process are solved in Section 3.3.

2.5 EDA conclusions

The extensive EDA conducted in Appendix A provided clear answers to each question posed in Table 2.2, which are detailed below in Table 2.3.

ID	EDA-based answer
1	There are missing values in the dataset that were not originally presented as such; they were instead 0s in columns where this would be physically impossible. After converting the impossible 0s to missing values with NumPy, it was established that there were 18 missing Glucose values, 125 missing BloodPressure values, 800 missing SkinThickness values, 1330 missing Insulin values, and 39 missing BMI values.
2	The analysis identified significant outliers across various features using box plots. Particularly extremely outliers were noted in the SkinThickness, BMI, and Insulin columns. These outliers could affect data scaling and model performance, and will need to be transformed in data cleaning.
3	The dataset is imbalanced, with 65.5% of rows in the training set having an outcome of 0, while the remaining 35.5% had an outcome of 1.
4	The EDA did show a somewhat positive correlation between the number of pregnancies and diabetes outcomes, although pregnancies alone are not decisive in predicting diabetes.
5	Pregnancies are found to correlate with age, as expected, but do not significantly influence other features.
6	There is a clear and significant distinction in glucose levels between diabetic and non-diabetic individuals, with higher levels being mostly present in those with diabetes, making it a strong indicator.
7	A slight positive correlation is observed between BMI and glucose levels, indicating that higher BMI could be associated with higher glucose levels.
8	Blood pressure shows some correlation with diabetes outcomes but is not a strong standalone diagnostic factor. However, like pregnancies, previously mentioned academic research specifies that it is still a useful feature to keep.
9	Individuals with diabetes tend to have higher skin thickness on average, although this feature contains significant high outliers in both individuals with and without diabetes.
10	The relationship between insulin and glucose varies between diabetic and non-diabetic individuals, with diabetic individuals showing higher glucose levels as a result of lower insulin levels, which is expected of type 2 diabetes.

Table 2.3: Answers to the questions after the completion of the EDA process.

Overall, the EDA process conducted in Appendix A has been highly beneficial for many reasons. Firstly, it identified critical data issues, such as the abundance of missing values and outliers, which are essential to address to improve model accuracy and reliability. By recognizing these issues early, the EDA ensures that data preprocessing steps can be effectively planned to mitigate their impact. Secondly, the EDA highlighted significant correlations between features and the diabetes outcome, such as the strong influence of glucose levels and BMI on diabetes diagnosis. Understanding these relationships helps in feature selection and engineering, which are crucial for building effective predictive models. Additionally, the EDA revealed class imbalances in the dataset, guiding the development of strategies to handle this imbalance during model training to prevent biased predictions.

The insights gained from the EDA therefore provide a solid foundation for informed decision-making in the experimental design and model development, ensuring that the models are trained on clean, balanced, and relevant data, ultimately enhancing their predictive performance. This is especially crucial given that the models would be used in the medical field, where incorrect predictions could have life-changing ramifications.

Experimental Design

3.1 Identification of chosen algorithms

Five classification algorithms will be leveraged on this dataset, these being Random Forests, Support Vector Machines, Logistic Regression, Naïve Bayes, and K-Nearest Neighbours. These were selected due to their previous use in literature based on these datasets, and also due to their high prevalence across many fields. Detailed descriptions of each algorithm, including their mathematical formulae, can be found in Appendix B.

3.2 Identification of appropriate evaluation techniques

When evaluating classification models, it is crucial to select appropriate metrics that provide a comprehensive understanding of the model's performance. For this project, three key evaluation metrics will be used: accuracy, recall, and F1 score. By using these three metrics in combination, we can gain a comprehensive understanding of our classification models' performance. Accuracy provides an overall measure of correctness, recall ensures positive cases are not missed, and the F1 score offers a balanced view that accounts for both precision and recall. This multi-metric approach will allow for a detailed evaluation of the five models. Detailed descriptions of each metric and how it is calculated can be found in Appendix C.

3.3 Data Cleaning and Pre-processing Transformations

3.3.1 Data encoding

Data encoding was not necessary in this dataset as all features were already numeric. However, a detailed description of typical encoding processes can be found in Appendix D.

3.3.2 Data cleaning

The EDA revealed the necessity for data cleaning in this dataset through the transformation of outliers and imputation of missing values.

Outlier handling

It was observed that all columns had outliers, with some having particularly extreme outliers that caused massive variance in the data and the visualisations produced from it. To address these, data was constrained to be within the 1st and 99th percentiles, and any data outside of these would be increased or decreased to one of these boundaries.

```python
# Transforming rows under the 1st percentile or over the 99th percentile.
for column in X_train.columns:
    lowerBound = X_train[column].quantile(0.01)
    upperBound = X_train[column].quantile(0.99)
    # Where data in the column is less than the 1st percentile or more than the 99th,
    # transform it to the percentile.
    X_train[column] = np.clip(X_train[column], lowerBound, upperBound)
    # np.clip will make values lower than the lower bound into the lower bound,
    # and values over the upper bound into the upper bound.
```

✓ 0.0s

Figure 3.1: Using np.clip to transform values less than or greater than the 1st and 99th percentiles.

Missing data imputation

After handling the outliers in the data, missing data can be imputed. The selected method for this was KNN imputation, which uses the KNN algorithm to aggregate data from the k nearest data points and imputes the average of these points in place of the missing value (TrainInData, 2024). This method can be very useful as it will not impute many rows of the same number as mean or median imputation would do, and instead imputes reasonable values based on similar rows, which is especially beneficial given the high correlations between some of the missing data (such as Insulin) and data that is mostly present (such as Glucose).

```python
# Creates a KNN Imputer that aggregates data from the 6 nearest neighbours to fill missing
# data across the previously identified columns.
imputer = KNNImputer(n_neighbors = 6)

# Fit the imputer on the training data and transform missing values.
X_train = imputer.fit_transform(X_train)
# This converts X_train to a NumPy array, thereby removing the column names.
# This is solved by converting it back to a DataFrame with the column names
# of the original X.
X_train = pd.DataFrame(X_train, columns = X.columns)

# The imputer is not fitted to the testing set, and instead only transforms
# missing rows within it.
X_test = imputer.transform(X_test)
# As with X_train, the column names are passed back by converting the
# NumPy array to a DataFrame with the column names of the original X.
X_test = pd.DataFrame(X_test, columns = X.columns)

print(X_train.isna().sum())
print(X_test.isna().sum())
```

✓ 0.0s

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
dtype: int64
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
```

Figure 3.2: Using KNN imputation to resolve missing data.

K was set to 6 in the imputer to increase the amount of data each imputed point is based on, but also to not average too many data points, as the data still varies despite containing many less outliers.

3.3.3 Data scaling

After imputing missing data, scaling can occur. Algorithms such as KNN and SVMs rely heavily on the distance between data points, meaning they can be heavily skewed by large distances between real numbers. Due to the high variance in the dataset, it was decided that standardisation will be the scaling procedure. Standardisation uses the formula in Equation 3.1 to scale data so that it has a mean of 0 and a standard deviation of 1.

$$Z = \frac{X - \mu}{\sigma} \tag{3.1}$$

Z is the standardized value (also known as "z-score")
X is the original value of the feature
μ is the mean of the feature
σ is the standard deviation of the feature

```
# Instantiate the scaler.
sc = StandardScaler()

# Fit the scaler to X_train and transform data in it.
X_train = sc.fit_transform(X_train)
# As with the imputer, the data can be made back into a
# DataFrame with column names.
X_train = pd.DataFrame(X_train, columns = X.columns)

# Only transform the testing data based on the fit from
# the training set.
X_test = sc.transform(X_test)
# As with the imputer, the data can be made back into a
# DataFrame with column names.
X_test = pd.DataFrame(X_test, columns = X.columns)


X_train.head()
```

✓ 0.0s Open 'X_train' in Data Wrangler

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	0.069906	-0.422956	-0.706207	1.624799	-0.532573	0.344790	1.379999	-0.619707
1	-0.842302	-0.127229	-1.212881	0.759820	-0.585500	0.124244	-0.667264	-0.876417
2	-0.538233	0.037064	-2.057339	0.327330	0.166058	1.418111	0.156092	-0.619707
3	-0.234164	0.628518	-0.115086	0.615657	-0.001543	-0.360956	0.922226	-0.534137
4	1.286183	-1.244421	0.138251	0.219208	-0.950693	0.859396	-0.044184	0.492705

Figure 3.3: Using a StandardScaler to standardise the data.

3.3.4 Data balancing

The dataset is imbalanced, containing more samples without diabetes. Therefore, the Synthetic Minority Oversampling Technique (SMOTE) was used to generate synthetic data to balance the classes. SMOTE also uses the KNN algorithm to generate data, but randomly selects nearest neighbours and generates data in-between the selected sample and the neighbour (TrainInData, 2023). This is repeated until the classes are balanced.

```
# Instantiate SMOTE, to be applied on the training set.
smote = SMOTE()

# View the original class imbalance.
print(y_train.value_counts())

# Fit SMOTE to the training set, and also generate data to balance it.
X_train, y_train = smote.fit_resample(X_train, y_train)

# We can now see that the classes are balanced.
y_train.value_counts()
```

✓ 0.0s

```
Outcome
0    1449
1     765
Name: count, dtype: int64

Outcome
1    1449
0    1449
Name: count, dtype: int64
```

Pre-SMOTE imbalance

Post-SMOTE

Figure 3.4: Using SMOTE to balance the training set.

SMOTE is not used to balance the testing set, as the testing set should be exclusively real data so that an accurate representation of the model's ability to predict real data can be gathered (Ozbun, 2021).

3.4 Limitations and Options

A key limitation in this design comes from the datasets themselves; they contain colossal amounts of missing data, with almost half of the Insulin values missing. With Insulin being such an integral part of type 2 diabetes diagnosis, the issue of having so much missing data cannot be overlooked. While it was addressed using KNN imputation, the synthetic data generated by this approach could possibly be of poor quality.

Furthermore, SMOTE was used to balance the classes. While SMOTE is a useful tool, it can introduce noise to a dataset. Additionally, SMOTE was performed after the KNN imputation, meaning that a substantial amount of the dataset was then synthetic data.

In future projects, it would be best to use more organised and clean datasets, where such substantial preprocessing measures would not be necessary.

Model Development

4.1 Predictive modelling process

After the extensive EDA and preprocessing of the data, the refined training set consisting of 80% of the data could then be used to train the five models. No features were removed, as all were deemed to be relevant in diabetes diagnosis from topical research and EDA.

```
# Creates 100 decision trees and aggregates their classifications, using the one
# found most commonly across them.
rf = RandomForestClassifier(n_estimators = 100, random_state = 42)

# Finds the optimal hyperplane to seperate classes by.
svc = SVC(random_state = 42)

# Computes a weighted sum of all features and maps it to between 0 and 1 using a
# sigmoid function. Rows over 0.5 are classified as 1, and rows under are classified as 0.
lr = LogisticRegression(random_state = 42)

# RF, SVC and LR have random elements to them. To ensure reproducibility, the random_state is
# set to 42.

# Calculates the probability of diabetes based on the combination of features
# and classifies based on the probability.
nb = GaussianNB()

# Classifies rows based on the 3 nearest data points.
knn = KNeighborsClassifier(n_neighbors = 3)

# Creating a list of the models so that they can be fitted in a for-loop.
models = [rf, svc, lr, nb, knn]

# Fits each model to the training dataset, where it will learn correlations and trends.
for model in models:
    model.fit(X_train, y_train)
✓ 0.3s
```

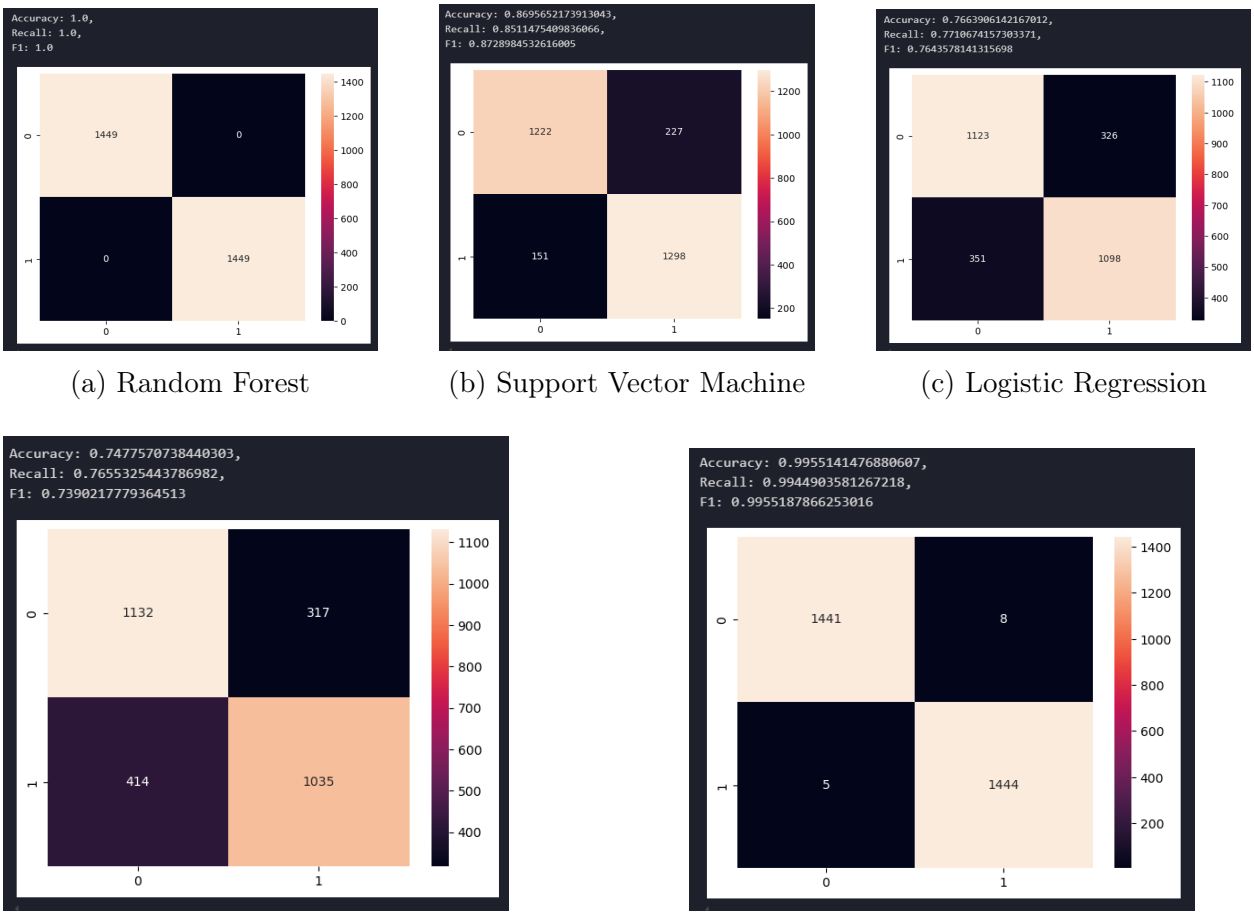
Figure 4.1: The code to fit each of the five models.

4.2 Results on seen data

To evaluate each model, the metrics previously discussed in Section 3.2 were logged alongside confusion matrices, which show the amounts of data each model correctly and incorrectly predicted, divided into true positives, false positives, false negatives and true negatives.

```
# Predict on the training set with all five models.
rf_seen_pred = rf.predict(X_train)
svc_seen_pred = svc.predict(X_train)
lr_seen_pred = lr.predict(X_train)
nb_seen_pred = nb.predict(X_train)
knn_seen_pred = knn.predict(X_train)

# Save a list of the predictions to iterate through.
predictions = [rf_seen_pred, svc_seen_pred, lr_seen_pred,
               nb_seen_pred, knn_seen_pred]

# Iterate through each model, outputting it's accuracy,
# recall and F1-Score, also producing a confusion matrix.
for pred in predictions:
    acc = accuracy_score(pred, y_train)
    recall = recall_score(pred, y_train)
    f1 = f1_score(pred, y_train)
    cm = confusion_matrix(y_train, pred) # Confusion matrix takes parameters in the inverse order.

    # Output the metrics.
    print(f"Accuracy: {acc},\nRecall: {recall},\nF1: {f1}")

    # Create the matrix.
    sns.heatmap(cm, annot = True, fmt = "g")
    # Each matrix needs to be individually printed. If this line
    # wasn't used, Seaborn would try to produce one overall matrix.
    plt.show()

✓ 0.7s
```

Figure 4.2: The code to predict with, and output the metrics and confusion matrices of each model on the training set.



(a) Random Forest (b) Support Vector Machine (c) Logistic Regression

(d) Naïve Bayes (e) KNN

Figure 4.3: The confusion matrices and evaluation metrics of each model on seen data.

Model	Accuracy	Recall	F1 Score
Random Forest	100%	100%	100
SVM	86.9%	85.1%	87.2
LR	76.6%	77.1%	76.4
NB	74.7%	76.5%	73.9
KNN	99.5%	99.4%	99.5

Table 4.1: The metrics of each algorithm on the seen training data.

4.2.1 Overall findings

Random Forest and KNN showed the best performance on the seen data, with perfect or nearly perfect scores across all metrics. SVM demonstrated good performance, while Logistic Regression and Naïve Bayes were the least effective. Performance on seen data doesn't necessarily translate to good performance on unseen data, and further evaluation using the unseen testing set was performed to assess the models' true predictive capabilities.

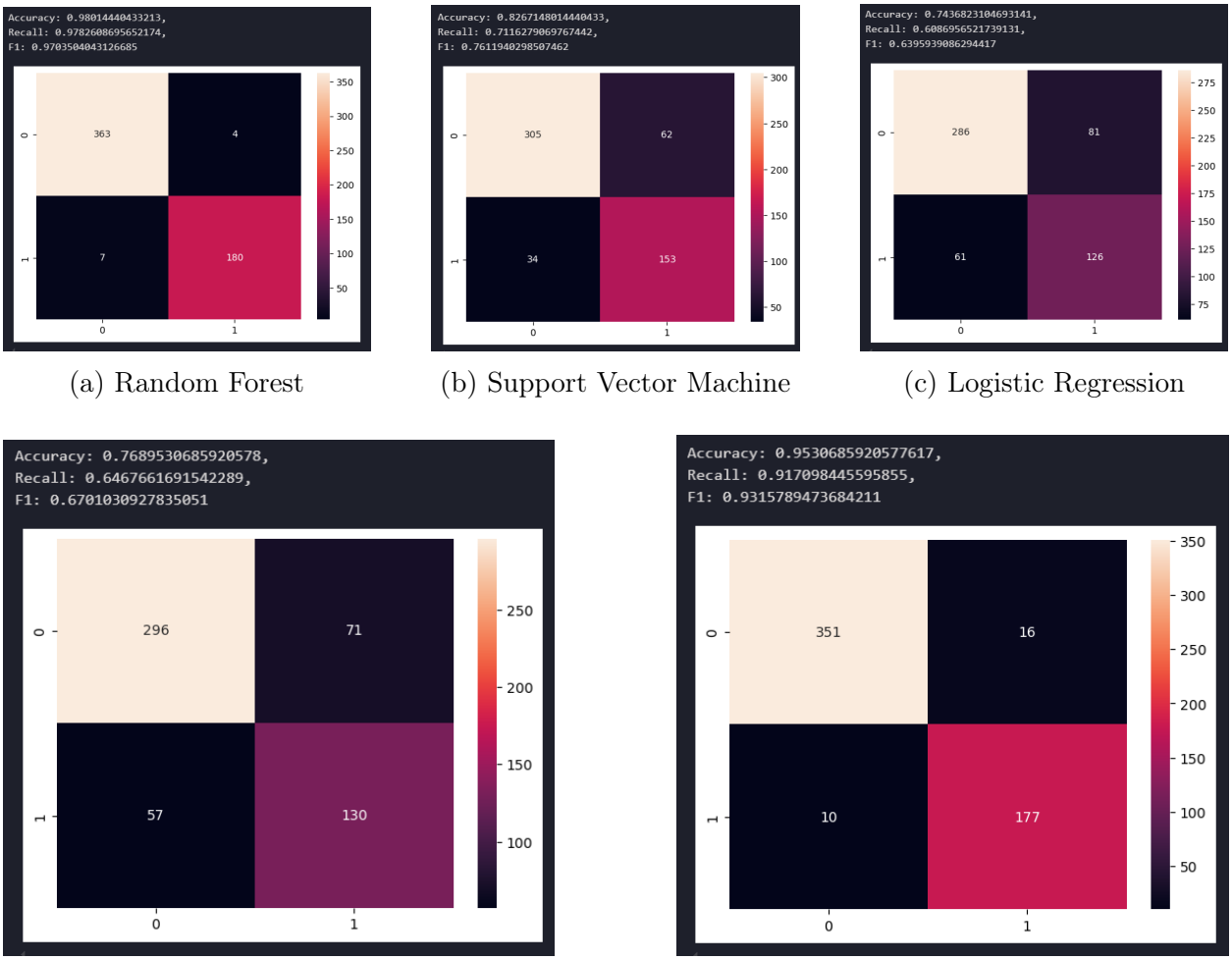
Evaluation and further improvements

5.1 Initial results on unseen data

The models were evaluated against the unseen training set, so that an analysis of how they would perform on real data could be gathered.

```python
# Predict on the testing set with all five models.
rf_unseen_pred = rf.predict(X_test)
svc_unseen_pred = svc.predict(X_test)
lr_unseen_pred = lr.predict(X_test)
nb_unseen_pred = nb.predict(X_test)
knn_unseen_pred = knn.predict(X_test)

# Save a list of the predictions to iterate through.
predictions = [rf_unseen_pred, svc_unseen_pred, lr_unseen_pred,
               nb_unseen_pred, knn_unseen_pred]

# Iterate through each model, outputting it's accuracy,
# recall and F1-Score, also producing a confusion matrix.
for pred in predictions:
    acc = accuracy_score(pred, y_test)
    recall = recall_score(pred, y_test)
    f1 = f1_score(pred, y_test)
    cm = confusion_matrix(y_test, pred) # Confusion matrix takes parameters in the inverse order.

    # Output the metrics.
    print(f"Accuracy: {acc},\nRecall: {recall},\nF1: {f1}")

    # Create the matrix.
    sns.heatmap(cm, annot = True, fmt = "g")
    # Each matrix needs to be individually printed. If this line
    # wasn't used, Seaborn would try to produce one overall matrix.
    plt.show()
```

Figure 5.1: The code to predict with, and output the metrics and confusion matrices of each model on the testing set.



(a) Random Forest (b) Support Vector Machine (c) Logistic Regression

(d) Naïve Bayes (e) KNN

Figure 5.2: The confusion matrices and evaluation metrics of each model on unseen data.

Model	Accuracy	Recall	F1 Score
Random Forest	98%	97.8%	97
SVM	82.6%	71.1%	76.1
LR	74.3%	60.8%	63.9
NB	76.8%	64.6%	67
KNN	95.3%	91.7%	93.1

Table 5.1: The metrics of each algorithm on the unseen testing data.

5.1.1 Overall findings

Random Forest and KNN were the best models, with Random Forest slightly edging out KNN. SVM showed moderate performance, while Logistic Regression and Naïve Bayes were the least effective models by a considerable margin.

5.2 Final model results

An additional iteration of each model was produced using stratified cross-validation and hyperparameter tuning to improve their performance. Detailed descriptions of these processes and how they were implemented can be found in Appendix E, with the results of these improved iterations shown in Figures 5.3 through 5.7.

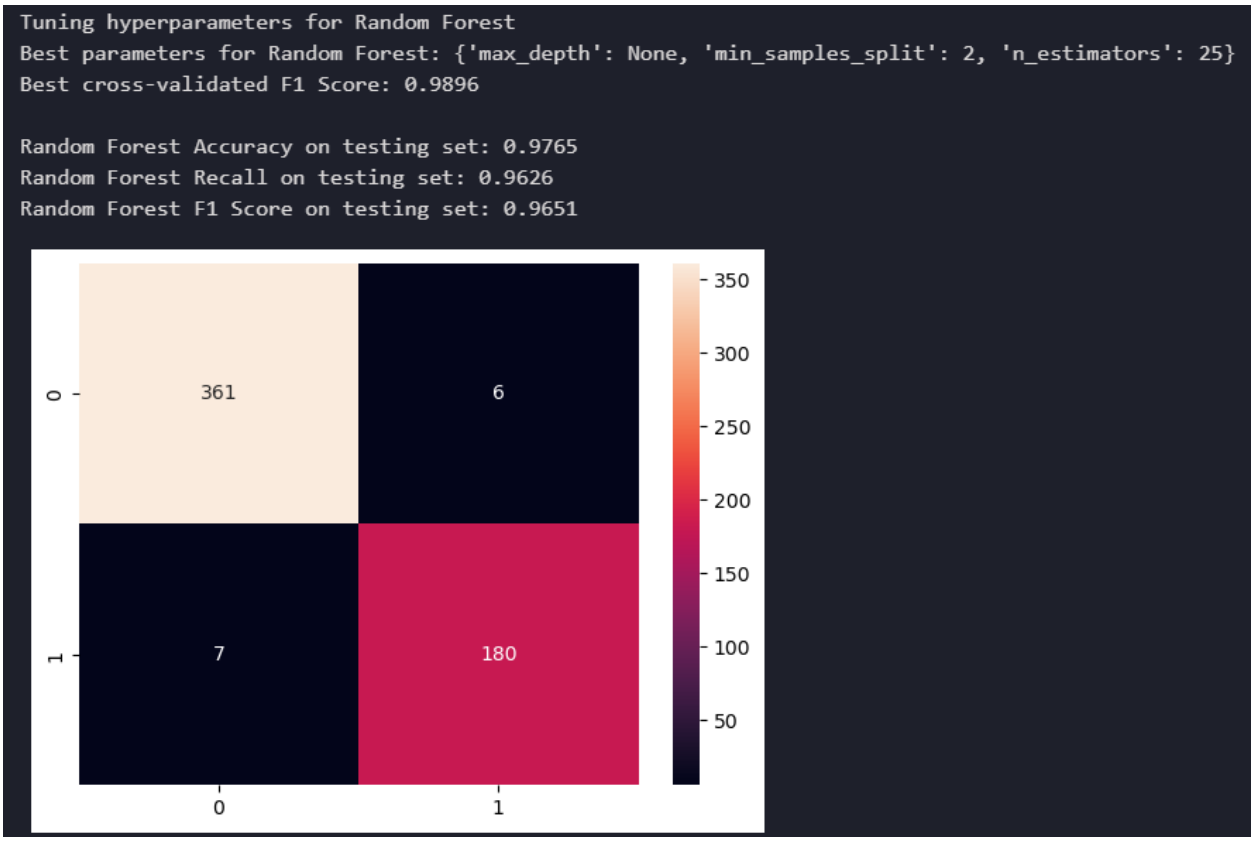


Figure 5.3: The optimal parameters, cross-validated F1 Score, evaluation metrics, and confusion matrix of the Random Forest model.

```
Tuning hyperparameters for Support Vector Classifier
Best parameters for Support Vector Classifier: {'C': 10, 'gamma': 'auto', 'kernel': 'rbf'}
Best cross-validated F1 Score: 0.9037

Support Vector Classifier Accuracy on testing set: 0.8574
Support Vector Classifier Recall on testing set: 0.8503
Support Vector Classifier F1 Score on testing set: 0.8010
```

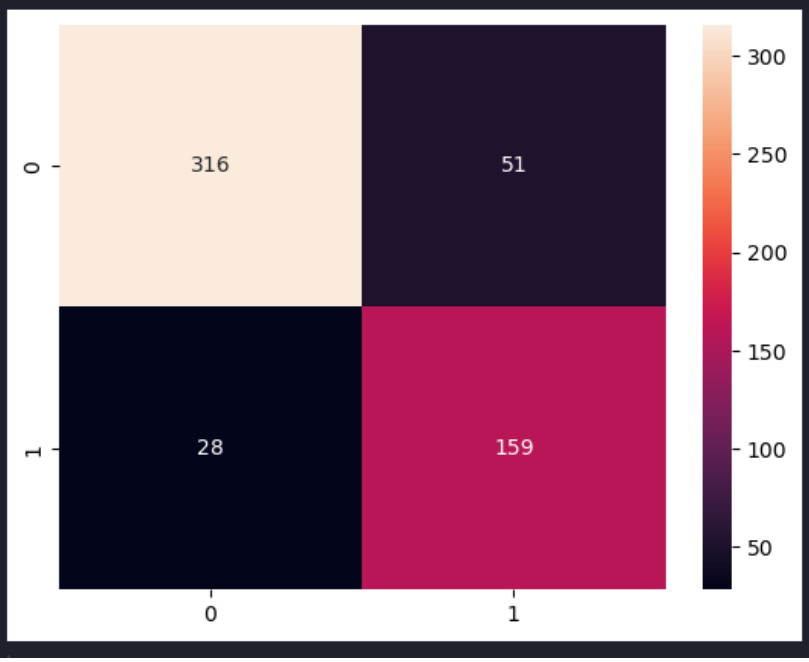


Figure 5.4: The optimal parameters, cross-validated F1 Score, evaluation metrics, and confusion matrix of the Support Vector Machine model.

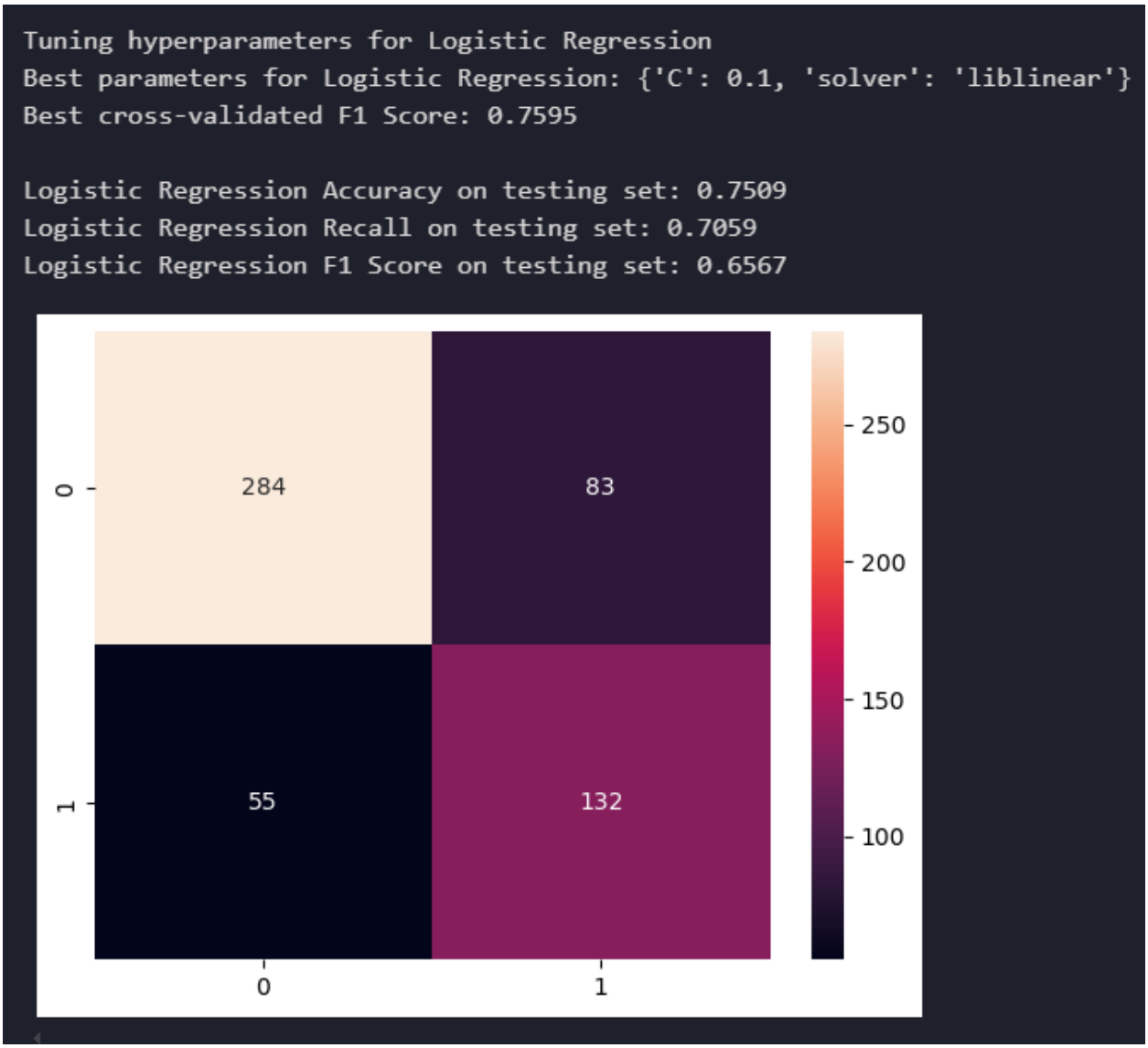


Figure 5.5: The optimal parameters, cross-validated F1 Score, evaluation metrics, and confusion matrix of the Logistic Regression model.

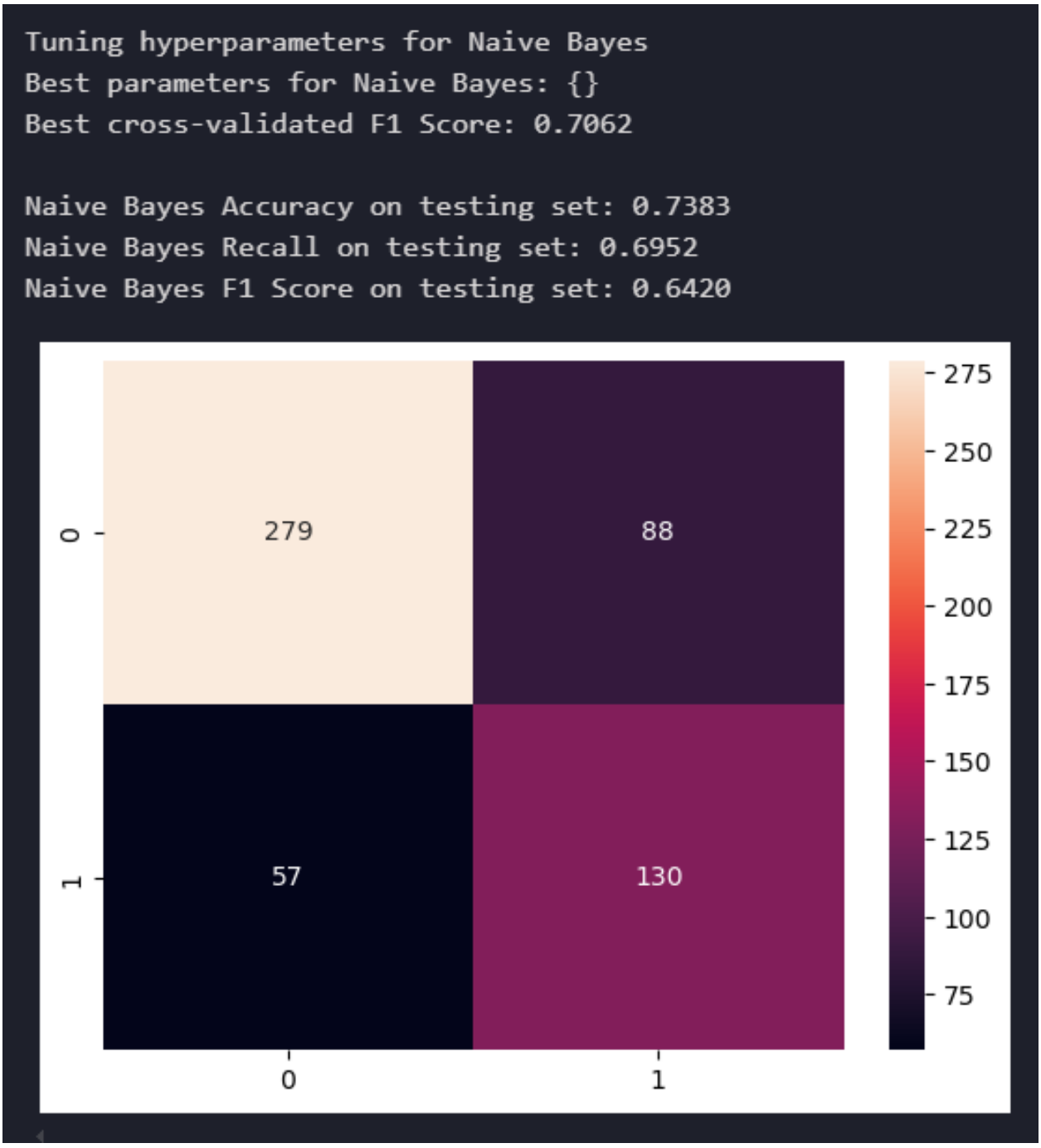


Figure 5.6: The cross-validated F1 Score, evaluation metrics, and confusion matrix of the Naïve Bayes model. There are no optimal parameters for this model as hyperparameter tuning was not performed on it. The reason for this is described in Appendix E.

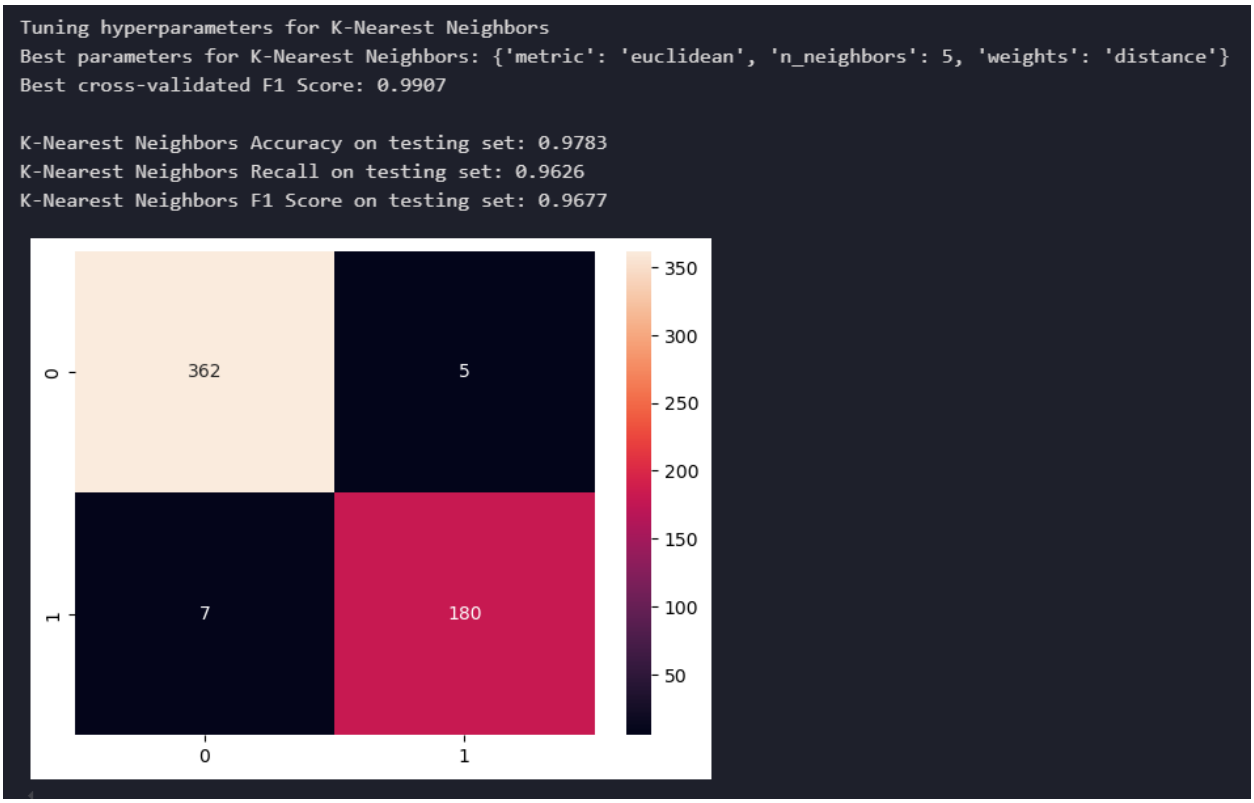


Figure 5.7: The optimal parameters, cross-validated F1 Score, evaluation metrics, and confusion matrix of the K-Nearest Neighbours model.

Conclusion

6.1 Summary of results

The introduction of stratified cross-validation and hyperparameter tuning[1] to the models resulted in significant improvements across most algorithms. Intriguingly, the Random Forest model was the only one to decrease in all three metrics, with a decrease in all three metrics on the testing data, leading to the KNN model slightly outperforming it (F1 score + 0.0026, Accuracy + 0.0017) due to one more false positive predicted by the Random Forest model. This suggests that the Random Forest's first iteration may have been overfitted. Despite this decrease, the model is still an excellent predictor.

Every model other than Random Forest and Naïve Bayes saw marked improvements in all three evaluation metrics, demonstrating the benefits of cross-validation and hyperparameter tuning.

Model	Accuracy	Recall	F1 Score
Random Forest	97.6%	96.2%	96.5
SVM	85.7%	85%	80.1
LR	75%	70.5%	65.6
NB	73.8%	69.5%	64.2
KNN	97.8%	96.2%	96.7

Table 6.1: The metrics of each second-iteration model on the unseen testing data.

Model	Accuracy	Recall	F1 Score
Random Forest	-0.3%	-1.6%	-0.5
SVM	+3.1%	+13.9%	+4
LR	+0.7%	+9.7%	+1.7
NB	-3%	+4.9%	-2.8
KNN	+2.5%	+4.5%	+3.6

Table 6.2: The differences in each evaluation metric in each second-iteration model on the unseen testing data compared to the first-iteration models.

[1]Except Naïve Bayes, which had no parameters to tune (See Appendix E.)

Overall, this project suceeded in developing and evaluating machine learning models for predicting the presence of diabetes mellitus using clinical data. Through extensive exploratory data analysis, data preprocessing, and model development, the following key results were achieved:

- Dataset Identification and Integration
  - The identification and integration of two datasets (Pima Indian and Frankfurt) resulted in a comprehensive dataset of 2,768 samples.
- Data Preprocessing and EDA
  - Assumptions and questions solved by exploratory data analysis revealed critical insights, including the presence of outliers, missing values, and class imbalance, which were addressed through constraining outliers to upper and lower quantiles, imputing missing data using KNN, and balancing classes with SMOTE.
- Model Development and Evaluation
  - KNN was the top-performing model after hyperparameter tuning, achieving 97.8% accuracy, 96.2% recall, and an F1-score of 96.6.
  - Random Forest performed exceptionally well, with 97.7% accuracy, 96.2% recall, and an F1-score of 96.5, only slightly behind KNN.
  - SVM showed significant improvement after tuning, reaching 85.7% accuracy, 85% recall, and an F1-score of 80.1.
  - Logistic Regression demonstrated moderate performance with 75% accuracy, 70.5% recall, and an F1-score of 65.6.
  - Naïve Bayes performed the least effectively, with 73.8% accuracy, 69.5% recall, and an F1-score of 64.2.
- Iterative improvements
  - Thorough research into machine learning practices revealed insights of the importance of cross-validation and hyperparameter tuning, which greatly improved 3 of the 5 models after implementation.

6.2 Reflection on Individual Learning

Over the course of this project, I have thoroughly enjoyed expanding my Python skills for data science and machine learning with industry standard packages like Pandas, NumPy and Scikit-Learn. The project was a massive challenge that seemed insurmountable at times, requiring extensive research into many academic papers and web resources to further my knowledge of the medical field and machine learning practices. The most challenging part of this project was the use of GridSearchCV on five different models, though I eventually reached the solution of an iterative for-loop that ran the search on all five in a clean block of code.

The topic of this project was one that is close to my heart - members of my family suffer from diabetes mellitus and other health complications caused by it, including heart attacks and strokes. Therefore, I am intensely interested in the applications of my passions in AI and Machine Learning to make a positive difference, and the knowledge I gained through this project will be a strong asset in doing so.

Despite the accomplishments of this project, it is not without flaws - KNN imputation was used on the testing set to address the missing data within it. While this did mean that more of the testing set could be used, this potentially comes at the expense of the data not being legitimate, as it was synthetically generated, and could have potentially caused overfitting.

I have always enjoyed writing code from as young as 11 years old, with Python being my first language. In University, I made my personal GitHub account, which I use for version control and iterative saving of my University work, including for this module in the CMP6202 repository, where every step taken in the production of this report and code is documented across many commits.

Appendix A - EDA Process and Results

Problems identified through the EDA conducted in this Appendix were resolved in Section 3.3.

A.1 Identification of missing or impossible values

An initial glance at the dataset would make it appear as though as there are not any missing values in the dataset, as shown in Figure A.1.

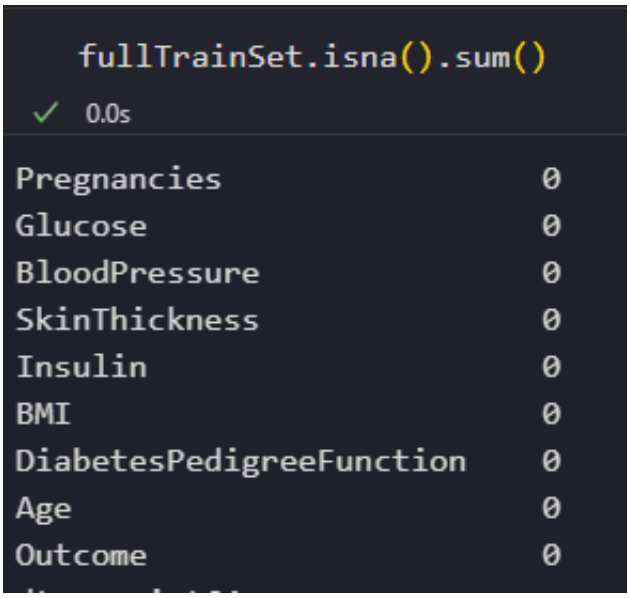


Figure A.1: An initial count of missing values before any further analysis.

This would be very good - if it were true. Instead of missing values, this dataset contains impossible values in five columns, highlighted in red in Figure A.2.[2]

```
fullTrainSet.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	2214.000000	2214.000000	2214.000000	2214.000000	2214.000000	2214.000000	2214.000000	2214.000000	2214.000000
mean	3.780036	121.210479	69.093496	20.599819	79.662150	32.058988	0.473864	33.284101	0.345528
std	3.321450	31.967589	19.319345	16.041849	111.192963	8.091686	0.328880	11.826703	0.475648
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	64.000000	0.000000	0.000000	27.300000	0.244000	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	40.000000	32.050000	0.375500	29.000000	0.000000
75%	6.000000	141.000000	80.000000	32.000000	129.750000	36.600000	0.629000	41.000000	1.000000
max	17.000000	199.000000	122.000000	110.000000	846.000000	80.600000	2.420000	81.000000	1.000000

Figure A.2: The overall dataset description. Physically impossible values are in red.

[2]The mean of the outcome is highlighted in blue, as the fact that it is under 0.5 indicates that there are more cases of outcome 0 (no diabetes) than 1 (diabetes). This will be further analysed in Section A.2.

It is not possible for a living person to have a 0 in any of the five highlighted columns, and these datasets do not contain information of the deceased. Therefore, it can be deduced that these values are erroneous, and should be treated as though they are missing. It is theorised by some in Kaggle discussions of these datasets that the zeroes in the Insulin column actually meant "imperceptible levels", which would have been useful data. However, a further review of literature around the datasets, especially Hayashi and Yukita (2016)'s paper, led to the discovery that these zeroes truly are missing values that were missing due to experimental invalidity at the time of their collection.

```
X_train[["Glucose","Insulin","BloodPressure","SkinThickness","BMI"]] = X_train[["Glucose","Insulin","BloodPressure","SkinThickness","BMI"]].replace(0, np.nan)
# These changes also need to be made to the full training set.
fullTrainSet[["Glucose","Insulin","BloodPressure","SkinThickness","BMI"]] = fullTrainSet[["Glucose","Insulin","BloodPressure","SkinThickness","BMI"]].replace(0, np.nan)
# Assuming that this issue is present in the testing set, it should also be fixed there.
X_test[["Glucose","Insulin","BloodPressure","SkinThickness","BMI"]] = X_test[["Glucose","Insulin","BloodPressure","SkinThickness","BMI"]].replace(0, np.nan)
✓ 0.0s
```

Figure A.3: Converting the impossible values to NaNs which are recognised by Pandas.

This conversion is applied to the actual X_train set for model training, as well as to the fullTrainSet copy made in Figure 2.4. Also, it is assumed that there will also be impossible values in the testing set, so they are also converted to NaNs there. Now that there are officially recognised missing values, they can be visualised using the Missingno package, which can produce a matrix of missing values by column, seen in Figure A.4.

Figure A.4: A Missingno matrix of the missing data per column.

This matrix shows that the Insulin and SkinThickness columns contain considerable amounts of missing data alongside many missing rows of BloodPressure, and some missing rows of BMI and Glucose, which will need to be fixed before any model can be trained. This is accomplished through imputation in Section 3.3.

A.2 Target univariate analysis

Binary classification models can develop bias if they do not have enough data of both classes to accurately predict what combinations of values should determine each class. Therefore, it is essential to view the balance between classes as in Figure A.6.

```
# We're only performing EDA on the training set. Y_train consists of the training set's Outcome column.
print(y_train.value_counts())
sns.countplot(X_train, x = y_train, color = "cyan")
```

```
Outcome
0    1449
1     765
```

Figure A.5: The code to produce Figure A.6.

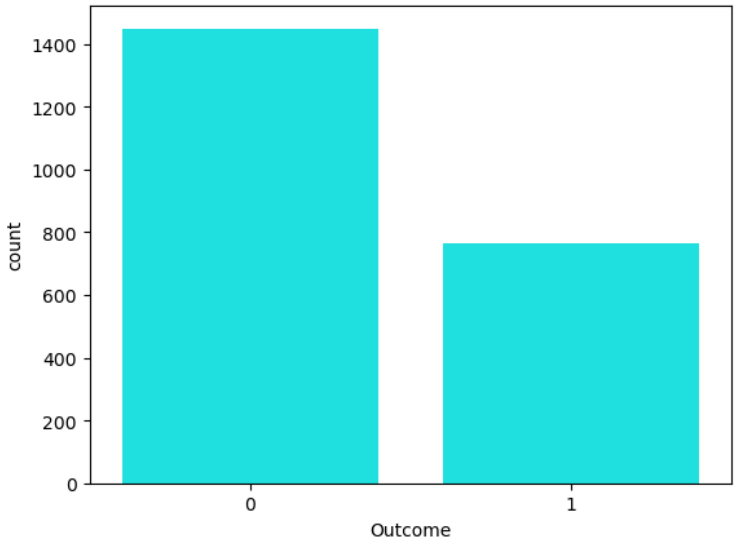


Figure A.6: The class distribution of the training set (0: 1449, 1: 765).

The dataset is imbalanced, favouring an outcome of 0. The exact percentages of each class can be calculated as seen below.

$$\text{Outcome } 0 = \frac{1449}{2214} = 0.655 \times 100\% = 65.5\% \tag{A.1}$$

$$\text{Outcome } 1 = \frac{765}{2214} = 0.345 \times 100\% = 34.5\% \tag{A.2}$$

The training set is 80% of the data, hence why it is 2214 rows rather than 2768. From these calculations, it can be determined that there is an imbalance present in the training set, which will be resolved in Section 3.3.

A.3 Multivariate analysis

To compare each variable to each other and the outcome, the full training set previously created in Figure 2.4 was used.

A.3.1 Data distributions by Outcome

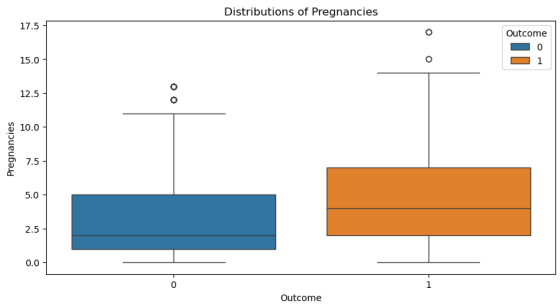
It is important to identify significant outliers in the dataset so that models do not become skewed by the inflated variance, standard deviation and mean that they cause. Outliers that are very high in comparison to the primary distribution of the data will cause the standard deviation and mean to increase, whereas low outliers will make them decrease. Their identification is important not only for data analysis, but also for model development, as outliers can cause some algorithms to perform poorly or overfit, most notably distance-based algorithms such as K-Nearest Neighbours (KNN) and Support Vector Machines (SVM), which is further discussed in Section 3.1.

One of the simplest ways to visualise data distributions in the dataset is through box plots, which present the spread and skewness of numerical data in a simple visual format by showing the inter-quartile range (IQR). This has the added benefit of identifying outliers, as rows outside of the IQR. The boxplots of each column in the training set are shown below in Figure A.8. Each of the plots contains two boxplots, where one is for individuals without diabetes, and one is for those with diabetes.

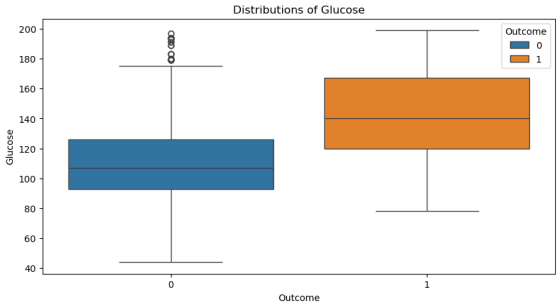
In the figure, it can be seen that all columns contain some outliers. However, most of these outliers are still relevant statistical data that would be possible, such as the outliers in Age being elderly people. Some of these outliers are less feasible, however; most notably the extreme high outliers in SkinThickness, BMI, and Insulin. Leaving these outliers in would affect the scaling of the data, and because it is generally ill-advised to remove rows of data especially with smaller datasets, they could instead be transformed to the upper bound to give a more even distribution. This will be accomplished in Section 3.3.

```
# Iterates through each column, producing a boxplot titled "Distributions of [column name]"
# Only iterates through X_train so that it doesn't produce one for the Outcome column itself.
for col in X_train:
    plt.figure(figsize = (10, 5))
    sns.boxplot(data = fullTrainSet, y = col, x = "Outcome", hue = "Outcome")
    plt.title(f'Distributions of {col}')
    plt.show()
✓ 0.7s
```

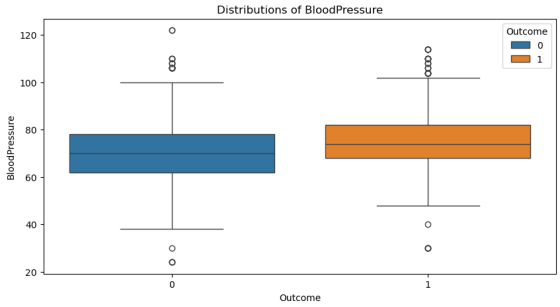
Figure A.7: The code used to produce Figure A.8.



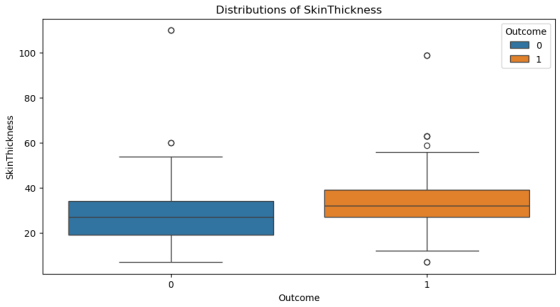
(a) Pregnancies



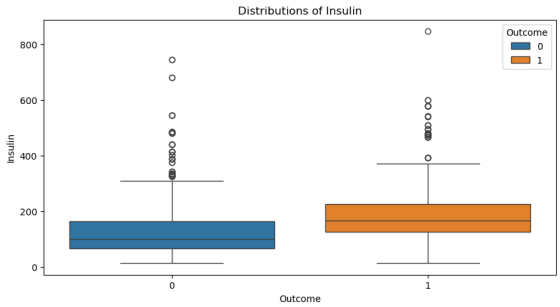
(b) Glucose



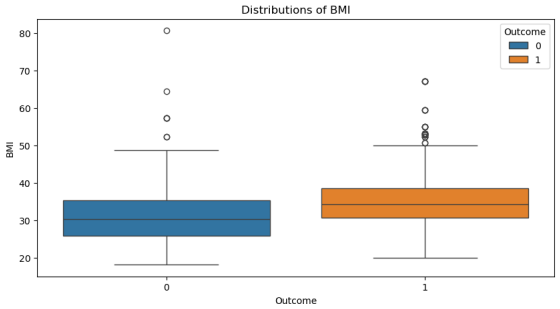
(c) BloodPressure



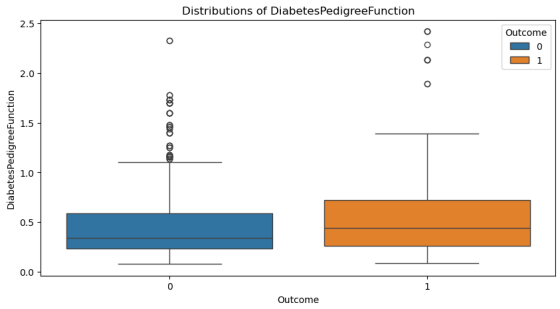
(d) SkinThickness



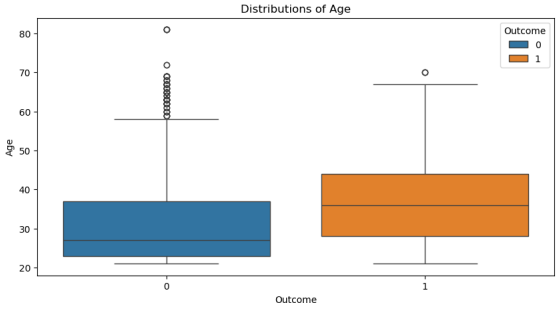
(e) Insulin



(f) BMI



(g) DiabetesPedigreeFunction



(h) Age

Figure A.8: Boxplots of all features by the outcome (Blue: No diabetes, Orange: Diabetes).

Pregnancies

It was assumed in Table 2.1 that the amount of pregnancies a woman has had may bear a strong correlation on whether they develop type 2 diabetes. This box plot does support the original assumption that the amount of pregnancies can positively correlate with the diabetes outcome, as the median and upper quartiles of the distributions of those with diabetes are higher than those without, though it also confirms that pregnancies are not a decisive factor on their own, as some patients had many pregnancies without having diabetes.

Glucose

This feature clearly shows a direct and strong correlation with the outcome; individuals with diabetes have much higher glucose values, with a clear upward shift in both the median and interquartile range (IQR). The lower whisker of the boxplots begins much higher with patients with diabetes, which implies that low glucose is a storng indicator that a patient does not have diabetes.

BloodPressure

Blood pressure values are fairly similar between the two groups, though individuals with diabetes show a slightly higher median. Both groups have similar variability and range, indicating that blood pressure is not as strong a distinguishing factor. However, this slight correlation is still useful information, so this column should be kept.

SkinThickness

This column contains significant outliers that will cause issues with data scaling. Aside from these outliers, it does appear that skin thickness positively correlates with the outcome.

Insulin

This column contains many outliers, with some of these being extremely high. Aside from these, the main IQRs of the plots appear to indicate that higher insulin values are indicative of diabetes, showing a positive correlation.

BMI

Individuals with diabetes tend to have higher BMI values on average. The median BMI for the diabetic group is higher, with a slightly larger IQR, showing that BMI is an important factor in distinguishing diabetes outcomes.

DiabetesPedigreeFunction

The diabetes pedigree function has a slightly higher median for individuals with diabetes, but the difference is not substantial. Both groups share significant overlap, suggesting that this feature alone may not strongly differentiate between outcomes, which differs from the original assumption.

Age

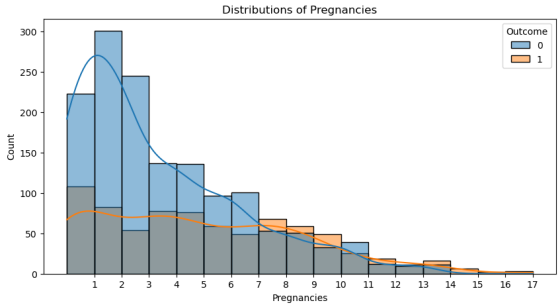
Individuals with diabetes tend to be older on average, as indicated by a higher median age for the diabetic group, and the fact that the IQR for the diabetic group is shifted upwards. However, many outliers without diabetes exist.

Another way to view data distributions is through histograms, which group data into "bins" before counting the occurrence of data in each bin and plotting this similarly to a bar plot. These are very useful for viewing the distribution curves of data. An additional feature of Seaborn is the Kernel Density Estimate (KDE), which will plot a line showing the distribution in the histogram.

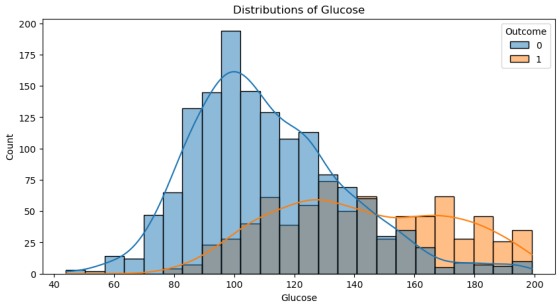
```
# Iterates through each column, producing a histogram titled "Distributions of [column name]"
# Only iterates through X_train so that it doesn't produce one for the Outcome column itself.
# KDE stands for Kernel Density Estimate. Setting it to true plots a line of best fit of the distribution.
for col in X_train:
    plt.figure(figsize = (10, 5))
    if col == "Pregnancies":
        # By default with Pregnancies, Seaborn will use decimal values in the X axis, creating a bad histogram.
        # Instead, set the bin width to 1, and the X axis to increment by 1, from 1 to 18.
        sns.histplot(data = fullTrainSet, x = col, kde = True, hue = "Outcome", binwidth = 1)
        plt.xticks(range(1, 18))
    else:
        # If it's any other column, use Seaborn's default X axis scale.
        sns.histplot(data = fullTrainSet, x = col, kde = True, hue = "Outcome")
    plt.title(f'Distributions of {col}')
    plt.show()
✓ 1.3s
```

Figure A.9: The code used to produce Figure A.10.

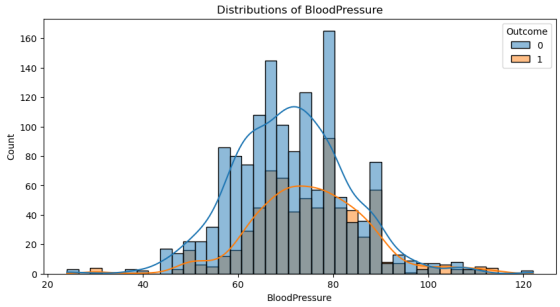
Figure A.10 on the following page shows the distributions of each feature. There are a variety of distributions, with many features having left-skewed distribution curves, though others such as BloodPressure and Glucose have bell curves, also known as normal distributions. Interestingly, Age follows a power law distribution, with the vast majority of data inhabiting the lower values, which would correlate with the knowledge of the dataset - women aged 21 or over had data collected, which reasonably explains why many of the values are from women aged between 20 and 30.



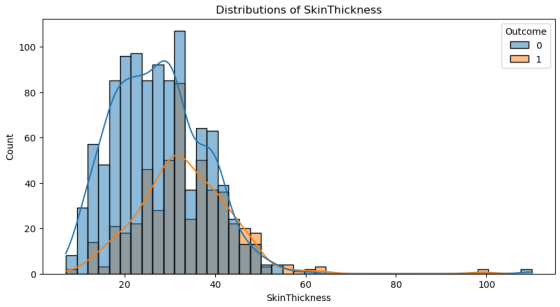
(a) Pregnancies



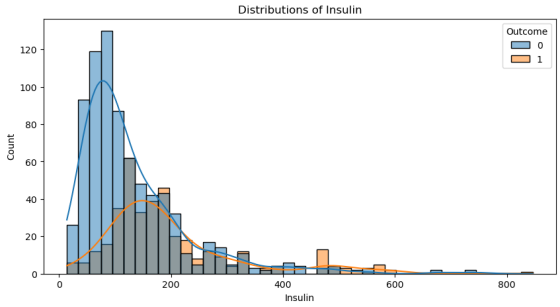
(b) Glucose



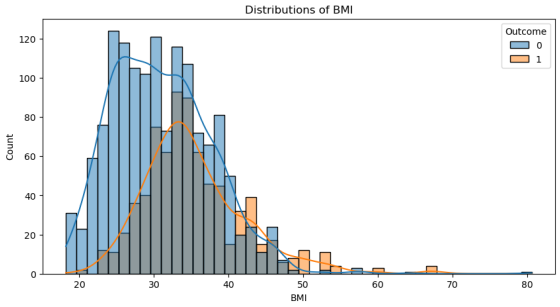
(c) BloodPressure



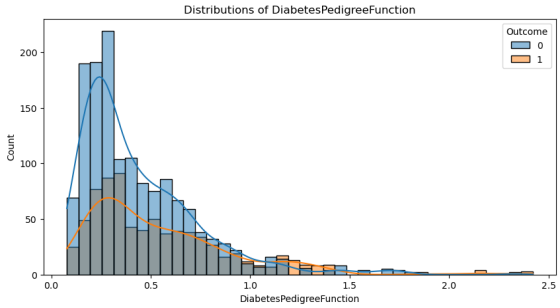
(d) SkinThickness



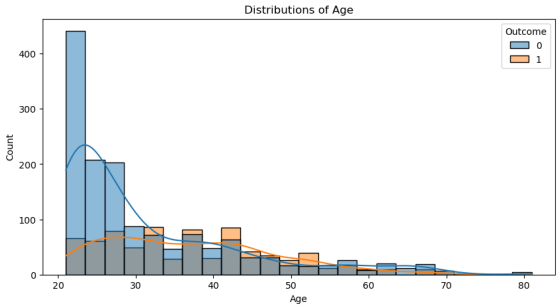
(e) Insulin



(f) BMI



(g) DiabetesPedigreeFunction



(h) Age

Figure A.10: Histograms of all features by the outcome (Blue: No diabetes, Orange: Diabetes).

A.3.2 Pair plot

To view every column in the dataset plotted against each other, a pair plot can be created using Seaborn, pictured in Figure A.12.

```
# Produces a pair plot of all columns, colouring each point by the outcome.
sns.pairplot(fullTrainSet, hue = "Outcome")
✓ 12.7s
```

Figure A.11: The code to produce Figure A.12.

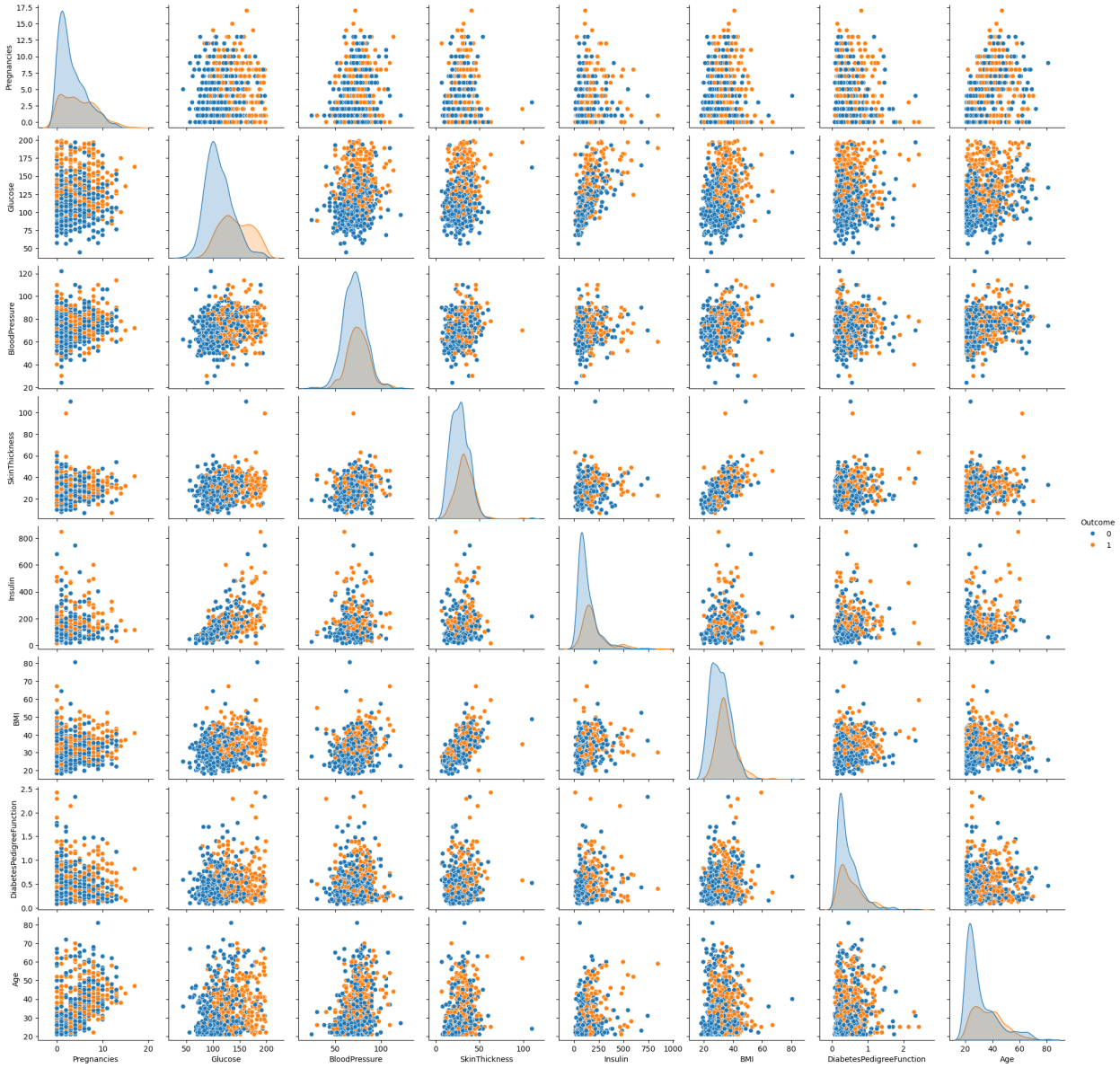


Figure A.12: The pair plot of the full training set (Blue: No diabetes, Orange: Diabetes).

There are multiple observations on the relationships between features to be made from this pair plot, such as:

- Glucose, BMI and Outcome
  - Glucose and BMI appear to be the two strongest predictors, with there being more cases of diabetes as each of these features increase, shown by the higher clustering of points.
  - Glucose also appears to slightly increase as BMI increases, suggesting a slight positive correlation between the two features.
- Pregnancies and Age
  - As expected, pregnancies do typically increase with age.
- Pregnancies and Outcome
  - There is no clear separation in the outcome based on pregnancies alone, suggesting it may not be a decisive factor.
- Age
  - While a person's age appears to not be directly relevant to them having diabetes, it instead appears that older individuals instead typically have higher glucose levels and BMI, which are decisive factors.
- Insulin
  - Appears to be a good predictor, but not quite as much as glucose. Most rows where Insulin is between 400 and 600 are diabetic.
- SkinThickness and BMI
  - As expected, these two features are closely interlinked as seen by the mostly tight clustering between them in their associated plot.

Relation between Insulin and Glucose by Outcome

A particular relation of note given the subject matter is the relation between insulin and glucose dependent on whether the individual has diabetes or not. Type 2 diabetes is characterised by insulin resistance and underproduction, meaning that lower insulin values are expected as glucose rises. Figure A.14 details this relation in patients with and without diabetes.

```
# Scatter plot using Linear Regression to produce a line of best fit.
# "scatter_kws" refers to keywords for the scatter points. "s: 5" makes the points smaller.
sns.lmplot(x = "Glucose", y = "Insulin", data = fullTrainSet, hue = "Outcome", scatter_kws={"s": 8})
✓ 0.2s
```

Figure A.13: The code to produce Figure A.14.

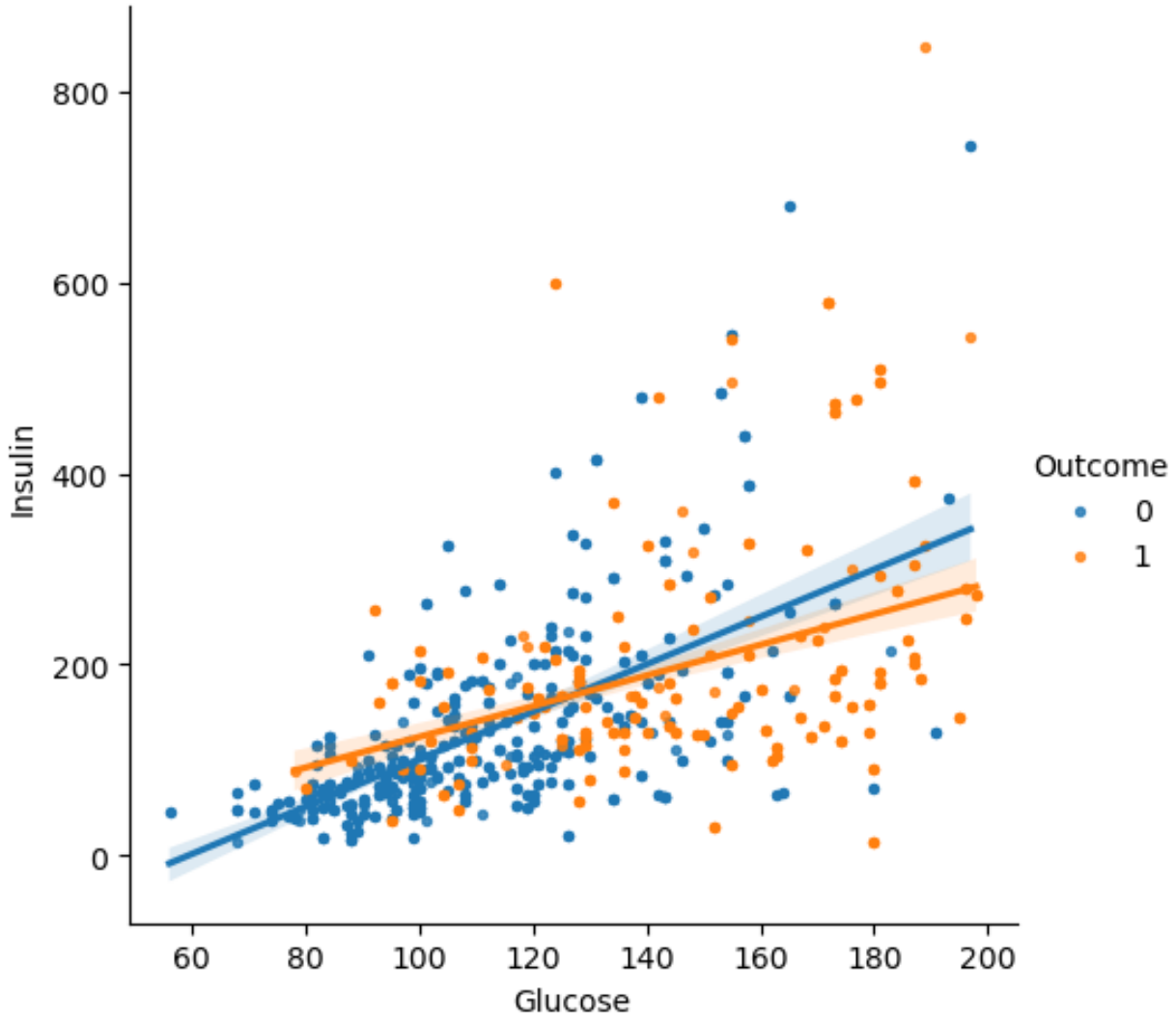


Figure A.14: A scatter plot with lines of best fit to show the trends in Insulin and Glucose.

The expectation is matched, as glucose rises more than insulin in patients with diabetes, and slower than insulin in patients without diabetes.

A.3.3 Correlation Matrix

An alternative and simpler way to visualise the correlations between features in a dataset is through a correlation matrix as pictured in Figure A.16. The matrix uses the full training set created earlier to show the correlations between the dataset's features in an annotated heatmap format, quantifying each correlation as a number. Correlation matrices only work with numerical features, though all features are numeric in this dataset which enables the matrix to show all features.

```
sns.heatmap(fullTrainSet.corr(), annot = True)
✓ 0.2s
```

Figure A.15: The code to produce Figure A.16.

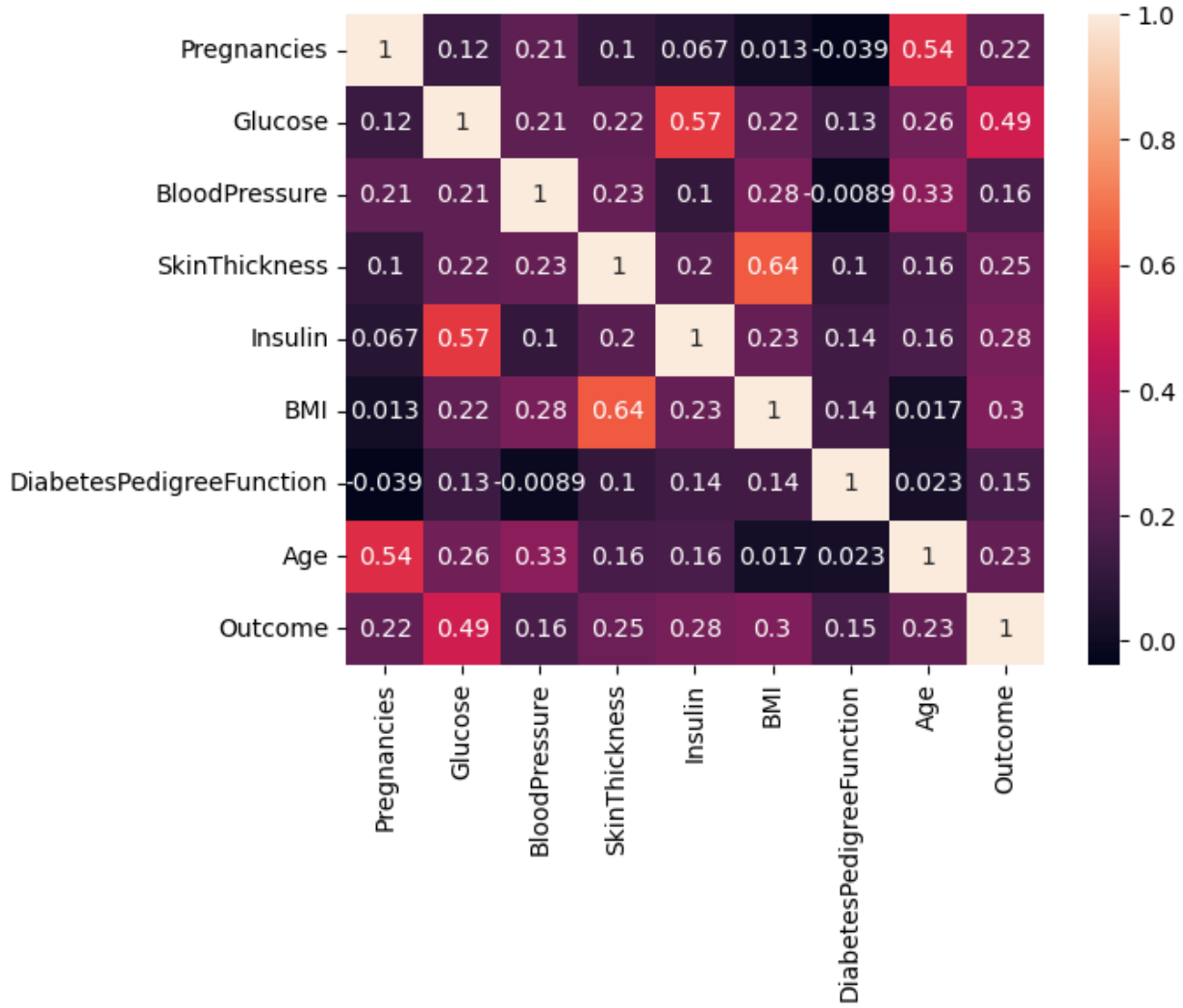


Figure A.16: The correlation matrix of the full training set.

The correlation matrix reflects many of the observations noted in the pair plot and data distributions. It shows the relationships between all features, including many expected relations like those of Glucose and Outcome, Pregnancies and Age, and BMI and SkinThickness. However, the matrix also reveals insights that subvert the previous assumptions about the dataset, such as the pedigree function actually having the weakest correlation with the outcome. This can be explained through other insights in the data as well as academic research.

Mambiya et al. (2019) argue that diabetes mellitus can be caused by genetic predisposition, which is represented by the pedigree function in this dataset, though lifestyle factors such as weight gain and poor dietary variance can be a more significant factor in the cause of the disease in both those with and without genetic predispositions. Their findings are also echoed within this dataset, with high BMIs representing overweightiness/obesity and high Glucose values representing high sugar intake, which are the two most significant factors in the outcome according to the correlation matrix.

Appendix B - Classification Algorithms

B.1 Random Forest

Random forests are ensemble models, meaning that they leverage multiple other models and average their findings to give a single result. The other models used by a random forest are decision trees, which are flowchart-like structures where data is split at each node of the tree based on feature values to arrive at a classification. Random forests create many decision trees based on randomly sampling the dataset, and these trees then classify rows based on what they learn. Then, the predictions for each row by all decision trees are aggregated, and the prediction reached by the most trees is used. The amount of decision trees used is a parameter that can be set, and the computational requirements of the algorithm also increase alongside this amount.

Random forests are reputed as one of the best classification algorithms, and are used frequently in many fields including diabetes diagnosis as seen in works from Chang et al. (2023) and AlZu'bi et al. (2023), demonstrating their suitability for this classification task.

B.2 Support Vector Machine

Support vector machines aim to find the optimal hyperplane[3] which separates classes with the highest possible margin, and are commonly used in classification problems (IBM, 2023). Data points on either side of the margin are known as support vectors. Figure B.1 shows how a perfect SVM would look in a two-dimensional dataset for demonstrative purposes.

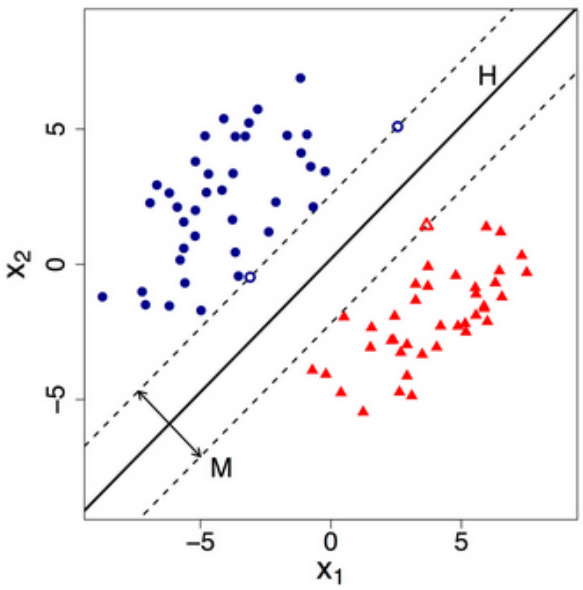


Figure B.1: An optimal SVM that demonstrates the algorithm's key features. Rows on the dotted line are the support vectors, the solid line is the hyperplane, and the space between the hyperplane and support vectors is the margin. (Kirchner and Signorino, 2018)

[3]A decision boundary that separates the classes. In a two-dimensional graph, this would be a line of best fit, but in a multidimensional dataset, it would be a hyperplane.

Kirchner and Signorino (2018) state that SVMs are computationally intensive, requiring lots of memory and processing power to run optimally. Additionally, Atla et al. (2011) mention that SVMs are particularly sensitive to noise in datasets, meaning their quality can be poor when there are many outliers. Despite these drawbacks, however, they can still be useful in diabetes classification as observed in the work of Zou et al. (2024).

B.3 Logistic Regression

Logistic regression computes a weighted sum of all input features, where the appropriate weights are learned by the algorithm as it trains. A sigmoid function, shown in Equation B.3, is then used to transform the sum to a value between 0 and 1, which is then used to classify the row as 0 if it is below 0.5, or classify it as 1 if it is above 0.5.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{B.3}$$

Logistic regression is a very common algorithm in binary classification tasks, as it is simple to implement and interpret, and can perform well even with limited amounts of training data (Kavya et al., 2024). It has also been widely used in previous analyses of the Pima Indian dataset. (AlZu'bi et al. (2023), Joshi and Dhakal (2021), Zou et al. (2024))

B.4 Naïve Bayes

Naïve Bayes is an example of a probabilistic classification algorithm (IBM, 2021), as it is based on Bayes' theorem, which calculates the probability of an event given that another event has occurred, shown in Equation B.4. In binary classification, it is the probability of the target given the other features. Naïve Bayes gets its name from its naïve assumption that features are all independent of each other.

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \tag{B.4}$$

This theorem calculates that the probability of an event occurring is equal to the probability of the event given prior knowledge multiplied by the prior probability of the event. This can be contextualised to this specific dataset, shown in Equation B.5.

$$P(Diabetes|Otherfeatures) = \frac{P(Otherfeatures|Diabetes) \cdot P(Diabetes)}{P(Otherfeatures)} \tag{B.5}$$

Gaussian Naïve Bayes is used in this project, as it is best suited to continuous data as seen in this dataset's features. Interestingly, Naïve Bayes typically has lower accuracy than other models such as Random Forests (A. Khan et al., 2023), though it has been used in previous diabetes classification tasks (Aftab et al., 2021; Chang et al., 2023; Zou et al., 2024).

B.5 K-Nearest Neighbours (KNN)

The K-Nearest Neighbours (KNN) algorithm is a fundamental machine learning classification method that operates by identifying the k closest training examples to a data point across all features and assigns the most common class among these neighbours as the prediction. K is a parameter that can be set within the algorithm, and has a heavy influence over its accuracy. KNN doesn't make any assumptions about the distribution of the data, making it useful for complex, real-world scenarios such as healthcare (Thomas and Rajabi, 2021), and has also been used on the Pima Indian and Frankfurt datasets to high success (AlZu'bi et al., 2023; Zou et al., 2024).

Appendix D - Descriptions of metrics

C.1 Accuracy

Accuracy is the most straightforward metric, measuring the overall correctness of the model's predictions. It is calculated as the ratio of correct predictions (both true positives [TP] and true negatives [TN]) to the total number of predictions, including false positives [FP] and negatives [FN] (Google, 2024), shown in Equation C.1.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{C.1}$$

While accuracy is intuitive and easy to interpret, it can be misleading in cases of imbalanced datasets, which is a concern in this dataset that will need to be addressed in preprocessing.

C.2 Recall

Recall, also known as sensitivity or true positive rate, is particularly important in medical diagnosis scenarios like diabetes detection. It measures the model's ability to correctly identify all positive cases. In this context, high recall ensures that we minimize the number of diabetic patients who are incorrectly classified as non-diabetic (false negatives). This is crucial because missing a diabetes diagnosis could have serious health implications for the patient such as neuropathy, vision problems and other health complications such as heart disease or strokes (NHS, 2024).

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{C.2}$$

C.3 F1 Score

The F1 score provides a balanced measure of the model's performance by combining precision[4] and recall into a single metric, and is particularly useful with imbalanced datasets such as the one used in this project. The F1 score is calculated as the harmonic mean[5] of precision and recall, giving equal weight to both metrics, which is useful because in a medical context, false positives and negatives are extremely important and can bear significant consequences.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{C.3}$$

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{C.4}$$

[4]The rate of correctly classified positives. Often has an inverse relationship with Recall (Google, 2024).

[5]The reciprocal of the mean of the reciprocals of the features. A reciprocal is 1 divided by the number, such as 4's reciprocal being 1/4.

Appendix D - What is data encoding?

Machine learning models are unable to directly interpret string data. This is a significant issue in many datasets, as some features may contain strings for categorical data, such as "Yes" and "No", for example. To solve this issue, data is **encoded** to a numerical representation. Many strategies exist to do so, for both nominal and ordinal data.

D.1 Nominal data

Nominal data is defined by Oxford Brookes University (2024) as "unordered, categorical and mutually exclusive", providing examples of country names or "Yes" or "No" questions[6]. Mutually exclusive means that data cannot fit into multiple of these categories, as a question cannot be answered with both "Yes" and "No", for example. To encode nominal data, processes such as one-hot encoding are used.

D.1.1 One-hot encoding

One-hot encoding is a technique to encode nominal data by creating new features representing each possibility of the data. For example, if a "Countries" feature included "England", "America", and "France", one-hot encoding of that feature would replace the original feature with "Countries_England", "Countries_America", and "Countries_France", where the data in each of these would either be a 1 or 0 based on what the row's value originally was. With one-hot encoding, it is especially important to ensure data does all fit in the expected categories, as errors like spelling mistakes would cause the creation of an entirely new and irrelevant feature.

Because a new feature is created for every possibility, one-hot encoding drastically increases the dimensionality of datasets it is leveraged against. This can massively increase storage and processing requirements, especially in datasets with many rows.

D.2 Ordinal data

Oxford Brookes University (2024) defines ordinal data as "ordered, categorical and mutually exclusive." The examples they provide are BMI categories (underweight, normal, overweight, obese) and questionnaire answers (strongly disagree, disagree, etc.). Ordinal data can be encoded using processes such as label encoding.

D.2.1 Label encoding

Label encoding encodes ordinal data by assigning integer values to each possibility. Reusing the questionnaire example, "strongly disagree" could correlate to 1, with "disagree" correlating to 2, and so on[7]. Encoding data in this way maintains its ordinal nature, as machine learning algorithms can interpret "larger number = better", which allows them to use the data for classification or regression tasks.

If label encoding is used on non-ordinal data, however, it can mislead machine learning algorithms into believing there is an inherent order in data where there is not, and interpret these as relationships that do not exist. Reusing the countries example, if data such as

[6]Data with only two possibilities is known as dichotomous data (Oxford Brookes University, 2024).
[7]This particular example is known as a "Likert scale". (Oxford Brookes University, 2024)

"England", "America" and "France" was encoded to 0, 1 and 2 respectively, a machine learning algorithm may interpret France as being the best country, despite them all being equal.

Appendix E - Additional techniques

To enhance the performance of the models, a second iteration of each model was created. These models implemented stratified cross-validation and hyperparameter tuning to improve the evaluation metrics, and therefore predictive capabilities, of each model.

E.1 Cross-Validation

E.1.1 Definition

Cross-validation is a technique used in machine learning to evaluate a model's performance on unseen data. It involves partitioning the dataset into subsets, using some subsets for training and others for validation, and repeating this process multiple times with different partitions. This helps assess how well the model generalizes to new data and can detect issues like overfitting. This is typically done using KFold.

Figure E.1: A visual representation of KFold cross-validation (Rosaen, 2024).

However, KFold does not account for class imbalance, which means that the sets it chooses may have differing levels of balance across the target class. Even in balanced datasets, this can still occur if the data has been shuffled as the set chosen may still contain fewer rows of one class than the other. This is solved by stratified cross-validation.

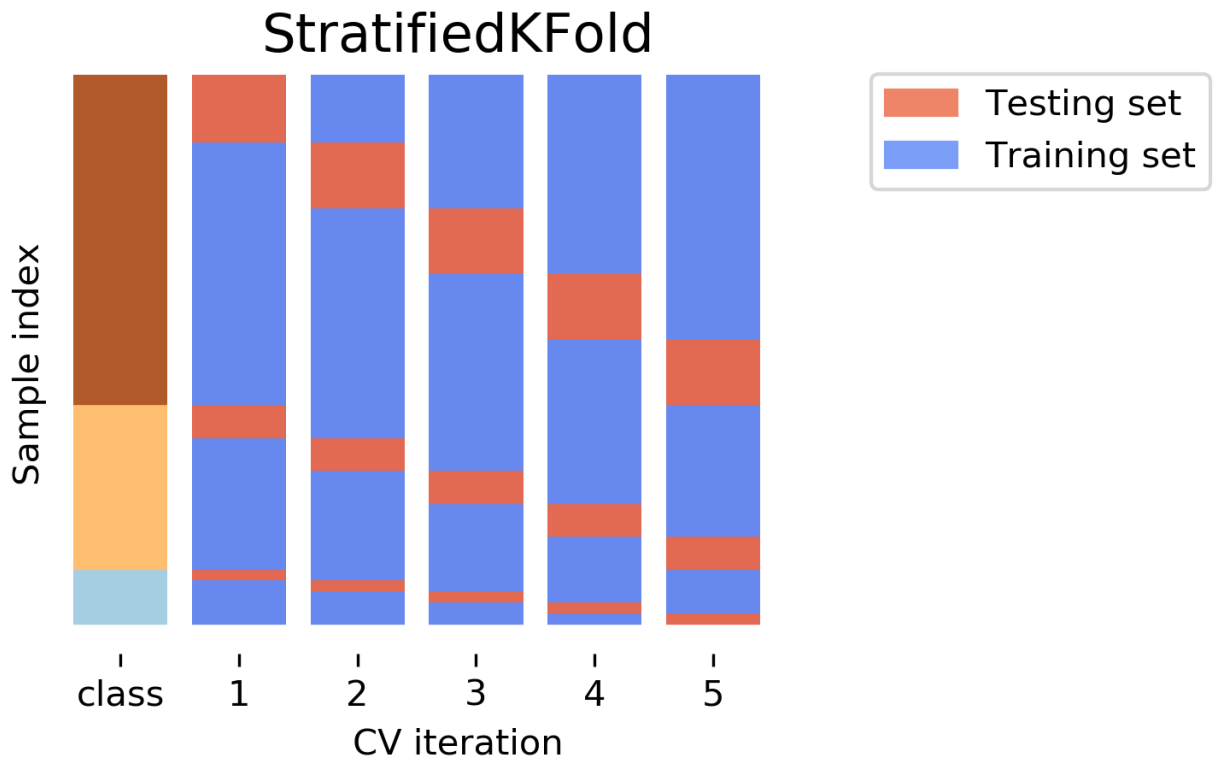


Figure E.2: A visual representation of StratifiedKFold cross-validation (Müller, 2024).

Stratified cross-validation is a variant that ensures each fold maintains the same proportion of samples for each target class. This is particularly useful for imbalanced datasets, as it helps maintain the class distribution across all folds, providing a more reliable estimate of the model's performance across various subsets of data, though it still is helpful even in balanced datasets. This dataset was balanced using SMOTE, though a stratified approach can still be beneficial for it.

Another key topic with KFold and StratifiedKFold is the amount of folds to use. As with train/test splits, there does not appear to be a definitive answer to the amount of folds to use in every dataset. However, previous literature on model development with this datasets such as that of Zou et al. (2024)'s used ten folds and achieved strong results, showcasing that this number of folds can work well with this data. Therefore, this project will also use ten folds.

E.1.2 Application

The new iterations use 10 fold stratified cross-validation through StratifiedKFold as shown in Figure E.3.

```
# Initialise the StratifiedKFold with 10 randomly picked splits.
# To ensure reproducibility, the random_state is set to 42.
skf = StratifiedKFold(n_splits = 10, shuffle = True, random_state = 42)

# Set the scoring parameters, as StratifiedKFold normally only uses Accuracy,
# but it was established that Recall and F1 will be used in evaluating each model.
# By doing this, each model's metrics can be retrieved, as they normally would not
# be measured.
metrics = {
    'accuracy': make_scorer(accuracy_score),
    'recall': make_scorer(recall_score),
    'f1': make_scorer(f1_score)
}
```

Figure E.3: The code used to instantiate a 10-fold StratifiedKFold.

The method used to perform hyperparameter tuning will take the SKF object as a parameter, so its implementation is displayed in Section E.2.2.

E.2 Hyperparameter tuning

E.2.1 Definition

Hyperparameter tuning refers to the configuration of parameters passed to a machine learning algorithm to alter its functionality. It is performed to maximise the predictive ability of a model, as models can have drastically different results depending on the parameters they were trained with. Some examples of parameters include the amount of decision trees (n_estimators) in a Random Forest, or the amount of neighbours (n_neighbors) in KNN.

Hyperparameter tuning can be performed manually by changing the parameters with each individual run of the model after evaluating its metrics, or automatically using Scikit-Learn functionality such as GridSearchCV. Automatic methods like GridSearchCV can be much faster than manual hyperparameter tuning.

GridSearchCV is supplied with a dictionary of parameters, known as the parameter grid. It will then exhaustively evaluate a model using every combination of parameters assigned in the parameter grid. This requires the fitting and training of a new model for every single combination, which is extremely computationally intensive in terms of processing power and memory usage. After this is completed, the GridSearchCV object will return the best model based on the metrics it achieved.

E.2.2 Application

GridSearchCV requires a parameter grid to be set up for each model containing the parameters to test in each combination.

```
# Initialises the parameter grids for each model. Each parameter listed here and
# each combination of them will be tested and evaluated, with the best one being returned.

# The tweakable parameters were identified from the Scikit-Learn documentation of each model.
param_grids = {
    'Random Forest': {
        'n_estimators': [1, 2, 3, 5, 10, 25, 50, 100, 200],
        'max_depth': [None, 1, 2, 3, 5, 10, 25, 50],
        'min_samples_split': [1, 2, 3, 5, 10]
    },
    'Support Vector Classifier': {
        'C': [0.1, 1, 3, 5, 10],
        # "Precomputed" is also a kernel option, but one has not been precomputed.
        'kernel': ['linear', 'rbf', 'poly', 'sigmoid'],
        'gamma': ['scale', 'auto']
    },
    'Logistic Regression': {
        'C': [0.1, 1, 3, 5, 10],
        'solver': ['liblinear', 'lbfgs', 'newton-cholesky', 'newton-cg', 'sag', 'saga']
    },

    # GaussianNB doesn't have any tweakable hyperparameters that will benefit this analysis,
    # but it still has an entry here to prevent an error with an upcoming for-loop. The only parameters
    # it has are "priors" for pre-existing probabilities and "var_smoothing" for calculation
    # stability, though neither of these will be useful in this classification.
    "Naive Bayes": {},

    'K-Nearest Neighbors': {
        'n_neighbors': [3, 5, 10, 25, 50, 100, 200],
        'weights': ['uniform', 'distance'],
        'metric': ['euclidean', 'manhattan']
    }
}

# The GridSearchCV object isn't instantiated here, because a new one
# will need to be made for each model. This will be done in a for-loop later.
```

Figure E.4: The parameter grid that will be used for GridSearchCV. As mentioned in the code, GaussianNB does not have any relevant parameters to tune.

The parameters of each model were found on their respective Scikit-Learn documentation page. A range of values was set for numerical parameters, and predefined strings were used for categorical parameters, such as the Support Vector Classifier's kernels. Following the creation of the parameter grid, a dictionary of models was created in the format "Model Name : Model object" to allow each model to be iterated through when using GridSearchCV.

```python
# Initialising new models. This is set up as a dictionary for use with the param_grid.
models = {
    'Random Forest': RandomForestClassifier(random_state = 42),
    'Support Vector Classifier': SVC(random_state = 42),
    'Logistic Regression': LogisticRegression(random_state = 42),
    'Naive Bayes': GaussianNB(),
    'K-Nearest Neighbors': KNeighborsClassifier()
}
# Note that no parameters other than the random_state of models that use randomness
# are set here, as GridSearchCV will do this later.
```

Figure E.5: The created dictionary of models. RF, SVC and LR had their random_state parameters set ahead of time for reproducibility.

This dictionary of models was then iterated through, using GridSearchCV on each model to identify the best combination of parameters from those listed in the parameter grid. The StratifiedKFold produced in Figure E.3 is used as the cross-validator.

```python
# Creating a list to hold each of the best models produced during
# this process.
bestModels = []

# For each model in the previously set models dictionary, run GridSearchCV to find the optimal parameter based on
# the parameter grid for the model of that name. Then, test it on th
for model_name, model in models.items():
    print(f"Tuning hyperparameters for {model_name}")

    grid_search = GridSearchCV(
        estimator = model, # The current model in the loop. Starts with Random Forest, then SVC, etc.
        param_grid = param_grids[model_name], # Using the parameter grid previously set.
        cv = skf, # Using the StratifiedKFold for cross-validation
        scoring = metrics, # Scoring the model on Accuracy, Recall and F1.
        refit = 'f1', # F1 score is one of the most important metrics, as discussed in Section 3.2.
        # As such, GridSearchCV will now aim to maximise the F1 score.
        n_jobs = -1 # Use all available CPU cores.
    )

    # Fit the model with the best parameters to the training set.
    grid_search.fit(X_train, y_train)

    # Output the best parameters for the model based on those used in the best model.
    print(f"Best parameters for {model_name}: {grid_search.best_params_}")
    print(f"Best cross-validated F1 Score: {grid_search.best_score_:.4f}\n") # Newline at the end to space the metrics.
    # :.4f means the metric will be trimmed to 4 decimal places.

    # Selects the model with the highest cross-validated F1 score on the training set.
    best_model = grid_search.best_estimator_

    # Make predictions on the testing set with the best model.
    y_pred = best_model.predict(X_test)

    # Get the metrics of the predictions.
    acc = accuracy_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred,)
    f1 = f1_score(y_test, y_pred)

    # Output the metrics.
    print(f"{model_name} Accuracy on testing set: {acc:.4f}") # Rounding it to 4 d.p.
    print(f"{model_name} Recall on testing set: {recall:.4f}") # Rounding it to 4 d.p.
    print(f"{model_name} F1 Score on testing set: {f1:.4f}") # Rounding it to 4 d.p.

    # The models were cross-validated on the TRAINING SET, and the testing set
    # is unseen data. This is why the best cross-validated F1 Score will differ
    # from the F1 score on the testing set.

    # Also produce a confusion matrix of its predictions.
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot = True, fmt = "g") # fmt = "g" avoids scientific notation.

    # Each matrix needs to be individually printed. If this line
    # wasn't used, Seaborn would try to produce one overall matrix.
    plt.show()

    # Add the model to the list of models so that it can be referenced later.
    bestModels.append(best_model)
```

✓ 44.4s

Figure E.6: Using GridSearchCV to iterate through the dictionary of models and cross-validate them on the training set. After this, the best model identified by GridSearchCV based on the maximum F1 score on the training set is saved and used to predict the unseen testing set, outputting its evaluation metrics and confusion matrices.

The evaluation metrics and confusion matrices produced are visible in Section 5.2.

Bibliography

Adam, Sumaiya, Harold David McIntyre, Kit Ying Tsoi, Anil Kapur, Ronald C. Ma, Stephanie Dias, Pius Okong, Moshe Hod, Liona C. Poon, Graeme N. Smith, Lina Bergman, Esraa Algurjia, Patrick O'Brien, Virna P. Medina, Cynthia V. Maxwell, Lesley Regan, Mary L. Rosser, Bo Jacobsson, Mark A. Hanson, Sharleen L. O'Reilly, Fionnuala M. McAuliffe, and the FIGO Committee on the Impact of Pregnancy on Long-term Health and the FIGO Division of Maternal and Newborn Health (2023). "Pregnancy as an opportunity to prevent type 2 diabetes mellitus: FIGO Best Practice Advice". In: *International Journal of Gynecology & Obstetrics* 160 (S1), pp. 56–67. ISSN: 1879-3479. DOI: 10.1002/ijgo.14537.

Aftab, Shabib, Saad Alanazi, Munir Ahmad, Muhammad Adnan Khan, Areej Fatima, and Nouh Sabri Elmitwally (2021). "Cloud-Based Diabetes Decision Support System Using Machine Learning Fusion". In: *Computers, Materials & Continua* 68 (1), pp. 1341–1357. ISSN: 1546-2226. DOI: 10.32604/cmc.2021.016814.

Akmeşe, Ömer Faruk (Mar. 30, 2022). "Diagnosing Diabetes with Machine Learning Techiques". In: *Hittite Journal of Science and Engineering* 9 (1), pp. 9–18. ISSN: 2148-4171. DOI: 10.17350/HJSE19030000250.

AlZu'bi, Shadi, Mohammad Elbes, Ala Mughaid, Noor Bdair, Laith Abualigah, Agostino Forestiero, and Raed Abu Zitar (Feb. 2023). "Diabetes Monitoring System in Smart Health Cities Based on Big Data Intelligence". In: *Future Internet* 15 (2). Number: 2 Publisher: Multidisciplinary Digital Publishing Institute, p. 85. ISSN: 1999-5903. DOI: 10.3390/fi15020085.

Atla, Abhinav, Rahul Tada, Victor Sheng, and Naveen Singireddy (May 1, 2011). "Sensitivity of different machine learning algorithms to noise". In: *J. Comput. Sci. Coll.* 26 (5), pp. 96–103. ISSN: 1937-4771.

c3.ai (2024). *What is Ground Truth? | Machine Learning Glossary Definition*. C3 AI. URL: https://c3.ai/glossary/machine-learning/ground-truth/ (visited on 12/09/2024).

Chang, Victor, Jozeene Bailey, Qianwen Ariel Xu, and Zhili Sun (Aug. 2023). "Pima Indians diabetes mellitus classification based on machine learning (ML) algorithms". In: *Neural Computing and Applications* 35 (22), pp. 16157–16173. ISSN: 0941-0643, 1433-3058. DOI: 10.1007/s00521-022-07049-z.

Dennison, Rebecca A., Eileen S. Chen, Madeline E. Green, Chloe Legard, Deeya Kotecha, George Farmer, Stephen J. Sharp, Rebecca J. Ward, Juliet A. Usher-Smith, and Simon J. Griffin (Jan. 2021). "The absolute and relative risk of type 2 diabetes after gestational diabetes: A systematic review and meta-analysis of 129 studies". In: *Diabetes Research and Clinical Practice* 171, p. 108625. ISSN: 01688227. DOI: 10.1016/j.diabres.2020.108625.

Dhanapalaratnam, Roshan, Tushar Issar, Leiao Leon Wang, Darren Tran, Ann M. Poynten, Kerry-Lee Milner, Natalie C.G. Kwai, and Arun V. Krishnan (Aug. 21, 2024). "Effect of Metformin on Peripheral Nerve Morphology in Type 2 Diabetes: A Cross-Sectional Observational Study". In: *Diabetes* 73 (11), pp. 1875–1882. ISSN: 0012-1797. DOI: 10.2337/db24-0365.

Diabetes UK (2024a). *How many people in the UK have diabetes?* Diabetes UK. URL: https://www.diabetes.org.uk/about-us/about-the-charity/our-strategy/statistics (visited on 11/27/2024).

Diabetes UK (2024b). *What causes type 2 diabetes?* Diabetes UK. URL: https://www.diabetes.org.uk/about-diabetes/type-2-diabetes/causes (visited on 12/14/2024).

Donnelly, Louise A., Rory J. McCrimmon, and Ewan R. Pearson (2024). "Trajectories of BMI before and after diagnosis of type 2 diabetes in a real-world population". In: *Diabetologia* 67 (10), pp. 2236–2245. ISSN: 0012-186X. DOI: 10.1007/s00125-024-06217-1.

Google (2024). *Classification: Accuracy, recall, precision, and related metrics | Machine Learning*. Google for Developers. URL: https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall (visited on 12/17/2024).

Hayashi, Yoichi and Shonosuke Yukita (Jan. 1, 2016). "Rule extraction using Recursive-Rule extraction algorithm with J48graft combined with sampling selection techniques for the diagnosis of type 2 diabetes mellitus in the Pima Indian dataset". In: *Informatics in Medicine Unlocked* 2, pp. 92–104. ISSN: 2352-9148. DOI: 10.1016/j.imu.2016.02.001.

IBM (Oct. 6, 2021). *What Are Naïve Bayes Classifiers? | IBM*. URL: https://www.ibm.com/topics/naive-bayes (visited on 12/17/2024).

IBM (Dec. 12, 2023). *What are SVMs?* URL: https://www.ibm.com/topics/support-vector-machine (visited on 12/17/2024).

John DaSilva (2024). *Frankfurt Diabetes Dataset*. diabetes. URL: https://www.kaggle.com/datasets/johndasilva/diabetes (visited on 11/25/2024).

Joshi, Ram D. and Chandra K. Dhakal (July 9, 2021). "Predicting Type 2 Diabetes Using Logistic Regression and Machine Learning Approaches". In: *International Journal of Environmental Research and Public Health* 18 (14), p. 7346. ISSN: 1661-7827. DOI: 10.3390/ijerph18147346.

Kavya, Mangala Shetty, Spoorthi B. Shetty, and Ihsan Mohammed Iqbal (Apr. 2024). "Applications of Thyroid Disease Detection Using Machine Learning". In: *2024 Third International Conference on Distributed Computing and Electrical Circuits and Electronics (ICDCECE)*. 2024 Third International Conference on Distributed Computing and Electrical Circuits and Electronics (ICDCECE), pp. 1–6. DOI: 10.1109/ICDCECE60827.2024.10548749.

Khalili, Davood, Marjan Khayamzadeh, Karim Kohansal, Noushin Sadat Ahanchi, Mitra Hasheminia, Farzad Hadaegh, Maryam Tohidi, Fereidoun Azizi, and Ali Siamak Habibi-Moeini (Feb. 14, 2023). "Are HOMA-IR and HOMA-B good predictors for diabetes and pre-diabetes subtypes?" In: *BMC Endocrine Disorders* 23, p. 39. ISSN: 1472-6823. DOI: 10.1186/s12902-023-01291-9.

Khan, Arsalan, Moiz Qureshi, Muhammad Daniyal, and Kassim Tawiah (2023). "A Novel Study on Machine Learning Algorithm-Based Cardiovascular Disease Prediction". In: *Health & Social Care in the Community* 2023 (1), p. 1406060. ISSN: 1365-2524. DOI: 10.1155/2023/1406060.

Khan, Moien Abdul Basith, Muhammad Jawad Hashim, Jeffrey Kwan King, Romona Devi Govender, Halla Mustafa, and Juma Al Kaabi (Mar. 2020). "Epidemiology of Type 2 Diabetes – Global Burden of Disease and Forecasted Trends". In: *Journal of Epidemiology and Global Health* 10 (1), pp. 107–111. ISSN: 2210-6006. DOI: 10.2991/jegh.k.191028.001.

Kirchner, Antje and Curtis S. Signorino (Jan. 2, 2018). "Using Support Vector Machines for Survey Research". In: *Survey Practice* 11 (1). DOI: 10.29115/SP-2018-0001.

Mambiya, Michael, Mengke Shang, Yue Wang, Qian Li, Shan Liu, Luping Yang, Qian Zhang, Kaili Zhang, Mengwei Liu, Fangfang Nie, Fanxin Zeng, and Wanyang Liu (Nov. 19, 2019).

"The Play of Genes and Non-genetic Factors on Type 2 Diabetes". In: *Frontiers in Public Health* 7. Publisher: Frontiers. ISSN: 2296-2565. DOI: 10.3389/fpubh.2019.00349.

Müller, Andreas C. (2024). *Data Splitting Strategies — Applied Machine Learning in Python*. URL: https://amueller.github.io/aml/04-model-evaluation/1-data-splitting-strategies.html (visited on 12/19/2024).

Nelaj, Ergita, Margarita Gjata, Irida Kecaj, Ilir Gjermeni, and Mihal Tase (June 2023). "HIGH BLOOD PRESSURE IN THE NEWLY DIAGNOSED TYPE 2 DIABETES PATIENTS". In: *Journal of Hypertension* 41 (Suppl 3), e172. ISSN: 0263-6352. DOI: 10.1097/01.hjh.0000940640.80128.7a.

NHS (Oct. 16, 2022). *Peripheral neuropathy - Causes*. nhs.uk. Section: conditions. URL: https://www.nhs.uk/conditions/peripheral-neuropathy/causes/ (visited on 12/04/2024).

NHS (2024). *Type 2 diabetes symptoms and treatments*. NHS inform. URL: https://www.nhsinform.scot/illnesses-and-conditions/diabetes/type-2-diabetes/ (visited on 12/17/2024).

Oxford Brookes University (2024). *Types of data*. Oxford Brookes University. URL: https://www.brookes.ac.uk/students/academic-development/maths-and-stats/statistics/types-of-data (visited on 12/18/2024).

Ozbun, Andrew (July 6, 2021). *Properly Using SMOTE*. Nerd For Tech. URL: https://medium.com/nerd-for-tech/properly-using-smote-930924e81ab5 (visited on 12/17/2024).

Rosaen, Karl (2024). *scikit-learn Pipeline gotchas, k-fold cross-validation, hyperparameter tuning and improving my score on Kaggle's Forest Cover Type Competition | ML Learning Log*. URL: http://karlrosaen.com/ml/learning-log/2016-06-20/ (visited on 12/19/2024).

Ruiz-Alejos, Andrea, Rodrigo M Carrillo-Larco, J Jaime Miranda, Robert H Gilman, Liam Smeeth, and Antonio Bernabé-Ortiz (Jan. 2020). "Skinfold thickness and the incidence of type 2 diabetes mellitus and hypertension: an analysis of the PERU MIGRANT study". In: *Public Health Nutrition* 23 (1), pp. 63–71. ISSN: 1368-9800. DOI: 10.1017/S1368980019001307.

Sivakumar, Muthuramalingam, Sudhaman Parthasarathy, and Thiyagarajan Padmapriya (Sept. 6, 2024). "Trade-off between training and testing ratio in machine learning for medical image processing". In: *PeerJ Computer Science* 10, e2245. ISSN: 2376-5992. DOI: 10.7717/peerj-cs.2245.

Suastika, Ketut, Pande Dwipayana, Made Siswadi, and R.A. Tuty (Dec. 12, 2012). "Age is an Important Risk Factor for Type 2 Diabetes Mellitus and Cardiovascular Diseases". In: *Glucose Tolerance*. Ed. by Sureka Chackrewarthy. InTech. ISBN: 978-953-51-0891-7. DOI: 10.5772/52397.

Tahapary, Dicky Levenus, Livy Bonita Pratisthita, Nissha Audina Fitri, Cicilia Marcella, Syahidatul Wafa, Farid Kurniawan, Aulia Rizka, Tri Juli Edi Tarigan, Dante Saksono Harbuwono, Dyah Purnamasari, and Pradana Soewondo (Aug. 1, 2022). "Challenges in the diagnosis of insulin resistance: Focusing on the role of HOMA-IR and Tryglyceride/glucose index". In: *Diabetes & Metabolic Syndrome: Clinical Research & Reviews* 16 (8), p. 102581. ISSN: 1871-4021. DOI: 10.1016/j.dsx.2022.102581.

Thomas, Tressy and Enayat Rajabi (2021). "Addressing Missing Data in a Healthcare Dataset Using an Improved kNN Algorithm". In: *Computational Science – ICCS 2021*. Ed. by Maciej Paszynski, Dieter Kranzlmüller, Valeria V. Krzhizhanovskaya, Jack J. Dongarra, and Peter M.A. Sloot. Cham: Springer International Publishing, pp. 223–230. ISBN: 978-3-030-77977-1. DOI: 10.1007/978-3-030-77977-1_17.

TrainInData (Mar. 28, 2023). *Overcoming Class Imbalance with SMOTE: How to Tackle Imbalanced Datasets in Machine Learning*. Train in Data's Blog. URL: https://www.blog.trainindata.com/overcoming-class-imbalance-with-smote/ (visited on 12/17/2024).

TrainInData (July 24, 2024). *KNN imputation of missing values in machine learning*. Train in Data's Blog. URL: https://www.blog.trainindata.com/knn-imputation-of-missing-values-in-machine-learning/ (visited on 12/17/2024).

UCI Machine Learning (2024). *Pima Indians Diabetes Database*. Pima Indians Diabetes Database. URL: https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database (visited on 11/25/2024).

Zou, Qiong, Yang Zhang, and Chang Sheng Chen (Feb. 13, 2024). "Construction and Application of a Machine Learning Prediction Model Based on Unbalanced Diabetes Data Fusion". In: *Proceedings of the 2023 International Joint Conference on Robotics and Artificial Intelligence*. JCRAI '23. New York, NY, USA: Association for Computing Machinery, pp. 114–123. ISBN: 979-8-4007-0770-4. DOI: 10.1145/3632971.3633348.