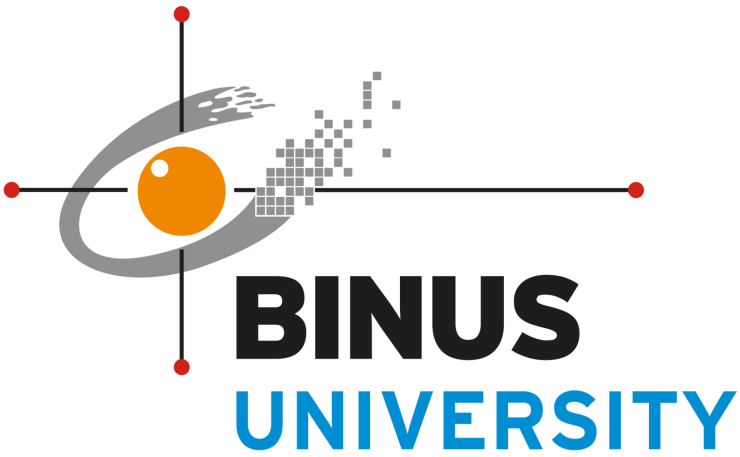
LAPORAN AOL MACHINE LEARNING



Disusun oleh:
Feta Kalih Wigati - 2502019451

Sekar Alisha Firdaus - 2501970531

Salsa Deswina Raihani - 2502069361

Kelompok 6

BAB I
PENDAHULUAN

1.1 Latar Belakang

Dataset yang kami gunakan berasal dari Institut Nasional Diabetes dan Penyakit Pencernaan dan Ginjal. Kami menggunakan *dataset* diabetes yang memiliki variabel-variabel yang dapat memprediksi seorang pasien menderita diabetes atau tidak. Dalam *dataset* diabetes ini semua data-data adalah hasil pencatatan data pada pasien perempuan berusia minimal 21 tahun keturunan India Pima.

1.2 Tujuan

- Memprediksi apakah seorang pasien menderita diabetes atau tidak berdasarkan pengukuran diagnostik
- Membangun model *machine learning* untuk mencari model terbaik yang dapat memprediksi penderita diabetes berdasarkan atribut-atribut yang ada, model terbaik akan diperoleh dari model yang memiliki nilai akurasi yang paling baik.

1.3 Deskripsi Data

Dataset kami memiliki 9 kolom, yaitu:

1. *Pregnancies*: Berapa kali pasien hamil.
2. *Glucose*: Konsentrasi glukosa plasma 2 jam dalam tes toleransi glukosa oral.
3. *BloodPressure*: Tekanan darah diastolik (mm Hg)
4. *SkinThickness* Ketebalan lipatan kulit trisep (mm)
5. *Insulin*: Insulin serum 2 jam (mu U/ml)
6. *BMI*: Indeks massa tubuh (berat dalam kg/(tinggi dalam m)^2)
7. *DiabetesPedigreeFunction*: Fungsi silsilah diabetes
8. *Age*: Usia (tahun)
9. *Outcome*: Hasil apakah pasien menderita diabetes atau tidak. Ditulis dalam 0 atau 1.

BAB II
LANDASAN TEORI

Kami menggunakan beberapa metode dalam melakukan pengamatan ini, berikut merupakan landasan teori dari metode-metode yang kami gunakan :

1. ***Random Forest***

   Pengertian *random forest*

   Random forest adalah salah satu teknik *machine learning* yang digunakan untuk memecahkan masalah regresi dan klasifikasi. *Random forest* menggunakan pembelajaran *ensemble*, yang merupakan teknik yang menggabungkan banyak pengklasifikasi untuk memberikan solusi untuk masalah yang kompleks.

   Algoritma dari *random forest* ini terdiri dari banyak pohon keputusan. '*Forest* 'yang dihasilkan oleh algoritma *random forest* dilatih melalui agregasi *bagging* atau *bootstrap*. *Bagging* adalah meta-algoritma ansambel yang meningkatkan akurasi algoritma *machine learning.*

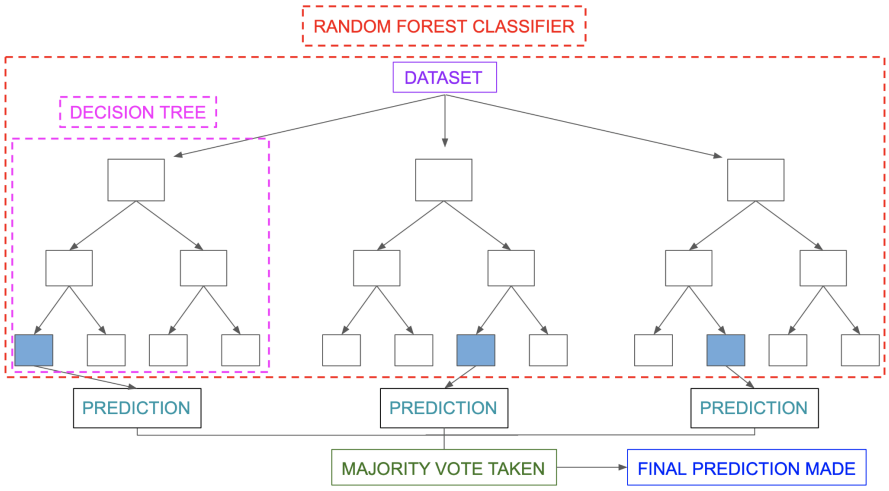
   Algoritma *random forest* menetapkan hasil berdasarkan prediksi dari *decision tree*. Ini memprediksi dengan mengambil rata-rata atau rata-rata output dari berbagai *tree*. Jika jumlah pohon ditingkatkan maka ketepatan hasil juga meningkat.

   Random forest menghilangkan batasan algoritma *decision tree*. Hal ini mengurangi *overfitting dataset* dan meningkatkan presisi. Ini menghasilkan prediksi tanpa memerlukan banyak konfigurasi dalam paket (seperti scikit-learn).

   Klasifikasi dalam *random forest*

   Klasifikasi dalam *random forest* menggunakan metodologi ansambel untuk mencapai hasil. Data pelatihan diumpankan untuk melatih berbagai *decision tree*. Dataset ini terdiri dari observasi dan fitur yang akan dipilih secara acak selama pemisahan node.

   Sistem *rain forest* bergantung pada berbagai *decision tree.* Setiap *decision tree* terdiri dari *decision nodes, leaf nodes*, dan *root node*. *Leaf node* dari setiap pohon adalah hasil akhir yang dihasilkan oleh *decision tree* tertentu. Pemilihan hasil akhir mengikuti sistem suara terbanyak. Dalam hal ini, keluaran yang dipilih oleh mayoritas *decision tree* menjadi keluaran akhir dari sistem *rain forest*. Diagram di bawah menunjukkan pengklasifikasi *random forest* sederhana.

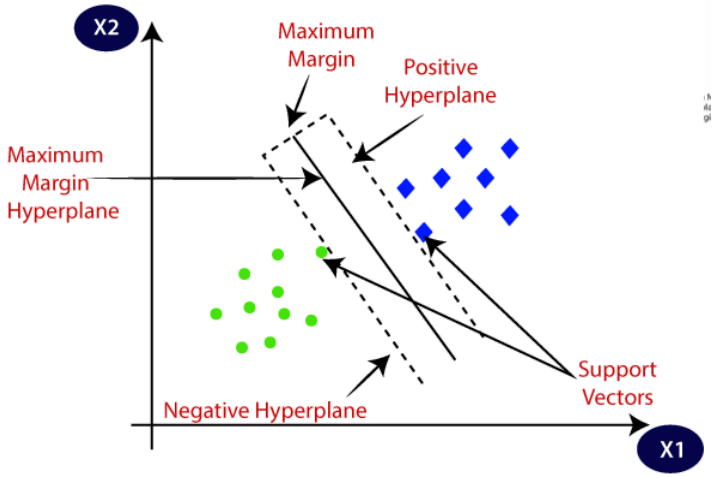


2. ***Support Vector Machine*** **(SVM)**

Support Vector Machine atau SVM adalah salah satu algoritma *supervised learning* yang paling populer, yang digunakan untuk klasifikasi.

Tujuan dari algoritma SVM adalah untuk membuat garis atau batas keputusan terbaik yang dapat memisahkan ruang n-dimensi ke dalam kelas sehingga kami dapat dengan mudah menempatkan titik data baru dalam kategori yang benar di masa depan. Batas keputusan terbaik ini disebut *hyperplane*.

SVM memilih titik/vektor ekstrem yang membantu menciptakan *hyperplane*. Kasus -kasus ekstrem ini disebut sebagai *support vectors*, dan karenanya algoritma disebut sebagai *support vector machine*. Pertimbangkan diagram di bawah ini di mana ada dua kategori berbeda yang diklasifikasikan menggunakan batas keputusan atau *hyperplane*:



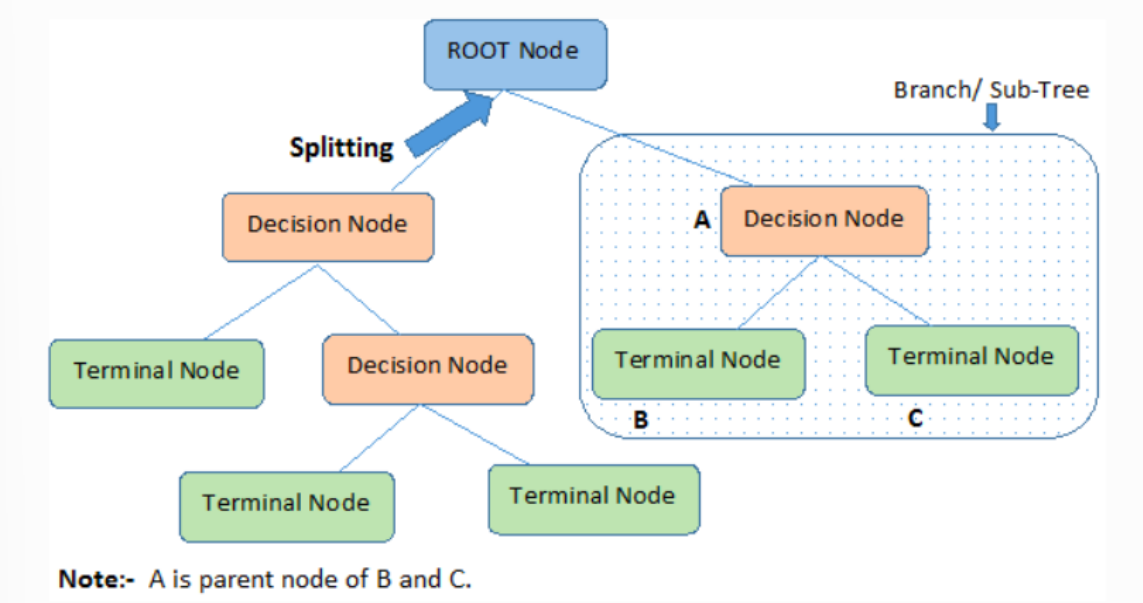
3. *Decision Tree*

Decision Tree adalah suatu algoritma *machine learning* yang dapat digunakan untuk mencari jawaban pada masalah dalam regresi dan klasifikasi dengan membagi data berdasarkan parameter tertentu dengan dua entitas, yaitu *decision nodes* dan *leaves*. *Leaves* yang menjadi hasil akhir dalam menentukan keputusan, sedangkan *decision nodes* merupakan tempat data.

Tujuan *decision tree* adalah memprediksi kelas atau variabel target dengan cara membuat model pelatihan yang akan memproses dan mempelajari aturan keputusan sederhana yang kemudian akan disimpulkan dari data sebelumnya.

Decision tree bekerja dengan cara mengajukan sebuah pertanyaan, selanjutnya membagi pohon menjadi subpohon berdasarkan jawaban ya atau tidak. Setiap *node* merupakan *test case* untuk beberapa atribut dan setiap *edge* yang turun sesuai dengan jawaban untuk *test case* tersebut. Adapun terminologinya yaitu :

1. *Root Node* : Mewakili seluruh populasi atau sampel.
2. *Splitting* : Proses membagi *node*.
3. *Desicion Node* : *Sub-node* yang terbagi menjadi lebih lanjut.
4. *Leaf / Terminal Node* : *Node* yang tidak terbelah
5. *Pruning* : Kebalikan dari proses *splitting*. Menghapus *sub-node* dari *decision node.*
6. *Branch / Sub-Tree* : Sub-bagian dari seluruh pohon.
7. *Parent and Child Node* : Sebuah *node* yang dibagi menjadi *sub-node*



Cara kerja algoritma *decision tree* sebagai berikut :

- Seluruh rangkaian pelatihan dianggap sebagai *root* atau akar.
- Nilai lebih baik bersifat kategoris. Temukan atribut terbaik dalam *dataset* dengan *Attribute Selection Measure (ASM)*. Jika nilai kontinu maka nilai tersebut didiskritisasi sebelum model dibangun.
- Berdasarkan nilai atribut yang telah dicatat, maka pencatatan akan didistribusikan secara rekursif.
- Dengan menggunakan beberapa pendekatan statistik, urutan penempatan internal *node* akan dikerjakan.

Logika algoritma dari *decision tree* lebih mudah dipahami dalam membuat model *machine learning* karena memiliki atau menampilkan struktur yang berbentuk seperti pohon. *Decision tree* meniru kemampuan cara berpikir manusia dalam membuat hasil akhir atau keputusan untuk lebih mudah dipahami.

4. Gaussian Naive Bayes

Naive bayes adalah suatu algoritma pembelajaran yang didasarkan pada teorema *bayes* untuk memecahkan suatu masalah klasifikasi. Adapun teorema *bayes* dikenal dengan hukum *bayes* yang digunakan untuk menentukan probabilitas suatu hipotesis. Rumusnya adalah :

$$P(A|B)= \frac{P(B|A)P(A)}{P(B)}$$

- P(A|B) Probabilitas *posterior* : Probabilitas hipotesis A pada peristiwa yang diamati oleh B.
- P(B|A) Probabilitas kemungkinan : Probabilitas bukti bahwa probabilitas suatu hipotesis bernilai benar.
- P(A) *Prior probability* : Probabilitas hipotesis sebelum mengamati bukti.
- P(B) Probabilitas *marginal* : Probabilitas bukti.

Keuntungan dari *naive bayes* ini adalah cepat dan mudah untuk memprediksi kelas dari suatu kumpulan data, dapat digunakan untuk klasifikasi biner dan multi-kelas, dapat bekerja dengan baik dalam memprediksi multi-kelas dibanding dengan model algoritma lainnya. Namun *naive bayes* juga memiliki kerugian, yaitu mengasumsikan semua fitur bersifat independen, tidak berhubungan, sehingga model algoritma ini tidak dapat mempelajari hubungan antar fitur-fitur yang ada satu sama lain. Pengklasifikasian *naive bayes* bisa digunakan untuk klasifikasi data-data medis.

Naive bayes dapat diperluas, perpanjangan *naive bayes* ini dapat disebut dengan *Gaussian naive bayes*, karena *naive bayes* paling sering diperluas ke atribut yang bernilai nyata dengan mengasumsikan distribusi *gaussian*. Model *gaussian* mengasumsikan fitur mengikuti distribusi normal, jika prediktor mengambil nilai kontinu dan bukan diskrit, maka model *naive bayes* akan mengasumsikan nilai ini diambil sampelnya dari distribusi *gaussian*. *Gaussian* atau distribusi normal paling mudah digunakan untuk memperkirakan distribusi data dibandingkan dengan fungsi lain, karena hanya perlu memperkirakan data dengan nilai rata-rata dan standar deviasi.

$$\text{mean}(x) = 1/n * \text{sum}(x)$$

Dimana n adalah jumlah *instance* dan x nilai untuk variabel *input*.
Sementara dalam menghitung standar deviasi menggunakan persamaan :

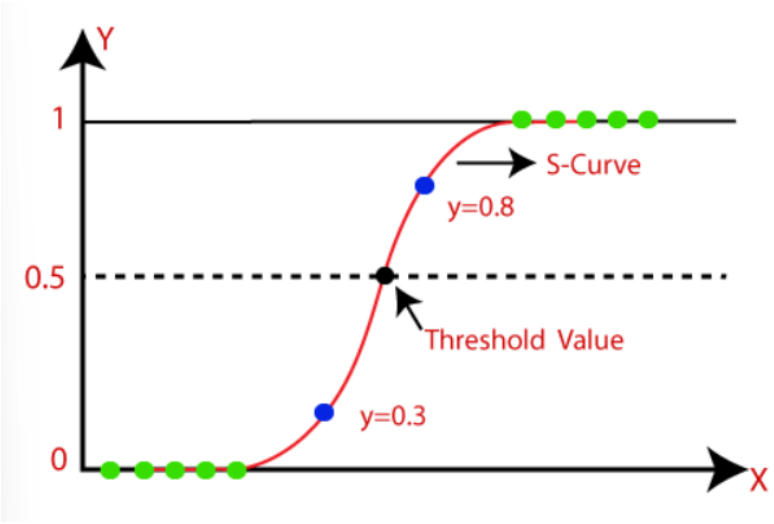
$$\text{standard deviation}(x) = \text{sqrt}(1/n * \text{sum}(xi-\text{mean}(x)^2))$$

Ini merupakan akar kuadrat dari selisih kuadrat rata-rata setiap nilai x dari nilai rata-rata x.

5. *Logistic Regression*

Logistic regression adalah salah satu algoritma *machine learning* yang berada dalam teknik *supervised learning* yang berguna untuk menyelesaikan masalah klasifikasi dengan memprediksi dari suatu variabel dependen yang bersifat kategoris dengan menggunakan seperangkat variabel yang masuk dalam variabel independen. Oleh sebab itu, hasilnya harus bernilai kategoris yang bisa berupa ya atau tidak, atau bisa juga 0 atau 1, dan lainnya. Logistic regression memiliki kemampuan untuk memgklasifikasi data baru menggunakan dataset kontinu dan diskrit serta memberikan probabilitas.

Logistic regression digunakan salah satunya adalah untuk mengklasifikasikan pengamatan dengan menentukan variabel mana yang paling efektif. Kurva dari fungsi logistik akan menunjukkan suatu kemungkinan seperti apakah pasien tersebut menderita diabetes atau tidak. Adapun asumsi untuk *logistic regression* diantaranya variabel independen tidak boleh multikolinearitas, dan variabel dependen harus bersifat kategoris.



Fungsi logistik biasa disebut juga dengan fungsi sigmoid. Fungsi logistik atau fungsi sigmoid merupakan suatu fungsi matematika yang digunakan untuk memetakan nilai prediksi ke probabilitas. Nilai suatu regresi logistik ini harus antara 0 dan 1, sehingga akan diperoleh bentuk kurva seperti huruf "S", oleh karena itu disebut dengan fungsi sigmoid.

Logistic regression memiliki beberapa jenis, diantaranya :

1. Binomial : Hanya ada dua kemungkinan jenis variabel dependen, contohnya ya atau tidak, 0 atau 1, positif atau negatif.
2. Multinomial : Mungkin ada tiga atau lebih jenis variabel dependen yang tidak terurut, contohnya "ayam", "bebek", atau "angsa".
3. Ordinal : Mungkin ada tiga atau lebih jenis variabel dependen yang mungkin diurutkan, contohnya "kurus", "normal", atau "gendut".

BAB III
METODE PENELITIAN

3.1 EDA

Exploratory Data Analysis adalah suatu proses uji investigasi awal yang bertujuan untuk mengidentifikasi pola, menemukan anomali, menguji hipotesis dan memeriksa asumsi. Dengan melakukan EDA, pengguna akan sangat terbantu dalam mendeteksi kesalahan dari awal, dapat mengidentifikasi *outlier*, mengetahui hubungan antar data serta dapat menggali faktor-faktor penting dari data. Proses EDA ini sangat bermanfaat dalam proses analisis statistik.

EDA yang kami lakukan pada *dataset* kami adalah dengan cara mencari *insight* dari *dataset* kami. Seperti mencari bentuknya, informasi pada masing-masing kolom, total data dari masing-masing kolom, rata-rata dari masing-masing kolom, standar deviasi dari masing-masing kolom, nilai minimal dari masing-masing kolom, nilai kuartil pertama dari masing-masing kolom, nilai kuartil kedua dari masing-masing kolom, nilai kuartil ketiga dari masing-masing kolom, dan nilai maksimal dari masing-masing kolom.

3.2 *Feature Engineering*

Feature engineering adalah sebuah teknik *machine learning* dalam membuat variabel baru yang tidak ada dalam *supervised learning* dengan memanfaatkan data mengikuti target yang akan dipelajari. Dengan *feature engineering* maka data yang diolah akan menjadi lebih sederhana, data menjadi lebih sempurna sehingga informasi yang didapatkan akan menjadi lebih baik dan cepat, serta performa model dan tingkat akurasi yang dihasilkan nantinya akan semakin baik jika dilakukan secara baik dan benar.

Ada beberapa teknik yang bisa dilakukan dalam melakukan *feature engineering* dalam mengolah data untuk *machine learning*, Berikut merupakan teknik-teknik yang kami gunakan diantaranya adalah :

1. Imputasi
   Dalam mengolah data nilai yang hilang menjadi salah satu masalah yang paling sering terjadi. Nilai hilang dapat menyebabkan proses pengolahan data menjadi tidak efisien. Untuk mengatasi masalah tersebut, perlu dilakukan imputasi data yaitu sebuah proses dalam menangani nilai-nilai yang hilang dengan menghapus catatan yang kehilangan nilai tertentu tersebut. Ada dua jenis imputasi :
   - Imputasi numerik
     Imputasi dengan cara mengisi semua nilai yang hilang dengan nilai 0 atau nilai -nilai numerik yang hilang akan diganti dengan rata-rata dari nilai yang sesuai di catatan lain.
   - Imputasi kategori

Imputasi dengan cara mengganti nilai-nilai yang hilang dengan nilai tertinggi atau nilai yang paling sering muncul pada kolom.

Kami melakukan imputasi secara numerik terhadap dataset dalam melakukan pengamatan ini, dengan cara mencari nilai 0 dan menggantinya dengan rata-rata dari nilai yang sesuai pada catatan lain.

2. Menghitung *Value*
   Menghitung *value* memiliki tujuan untuk memahami isi data secara lebih baik dan jelas. Dengan menghitung value kami bisa mendapatkan informasi tentang jumlah-jumlah data pada atribut-atribut yang tersedia dalam dataset. Dalam melakukan pengamatan pada *dataset* diabetes ini kami menghitung value terhadap jumlah data dari masing-masing data, serta melihat jumlah masing-masing pasien yang menderita diabetes dan yang tidak menderita diabetes.

3. Korelasi antar kolom
   Korelasi digunakan untuk mengetahui ada atau tidaknya hubungan linear antara dua variabel yang ditunjukkan oleh sebuah koefisien korelasi. Jika nilainya semakin mendekati 1 maka hubungan antara dua variabel tersebut akan semakin kuat, jika nilai semakin mendekati 0 maka hubungan antara dua variabel tersebut akan semakin rendah.

   Korelasi yang kami lakukan menggunakan korelasi pearson untuk menghitung korelasi antar dua variabel satu sama lain. Kami menggunakan grafik *heatmeap* dengan menggunakan *library Seaborn.*

4. *Outliers*
   Outlier merupakan nilai rendah atau tinggi yang dinilai tidak biasa dalam kumpulan data. Adanya suatu *outlier* bisa berdampak pada performa model nantinya, outlier bisa mempengaruhi nilai akurasi yang berdampak pada hasil prediksi. Untuk itu perlu dilakukannya pengecekan adanya outlier pada suatu atribut dan menanganinya secara baik. Adapun caranya dengan metode :
   - Penghapusan nilai
     Catatan yang mengandung *outlier* akan dihapus dari distribusi data. Namun jika ditemukan pada banyak variabel maka cara ini kurang efektik karena sebagian besar lembar data akan hilang.
   - Mengganti nilai
     Outlier akan ditangani sebagai nilai yang hilang dan diganti dengan menggunakan imputasi yang sesuai.
   - *Capping*

Mengganti dan membatasi nilai maksimum dan minimum dengan nilai arbitrer yaitu nilai dari distribusi variabel.

- Diskritisasi
  Mengubah variabel kontinu, model, dan fungsi menjadi variabel diskrit.

3.3 *Train Machine Learning Models*

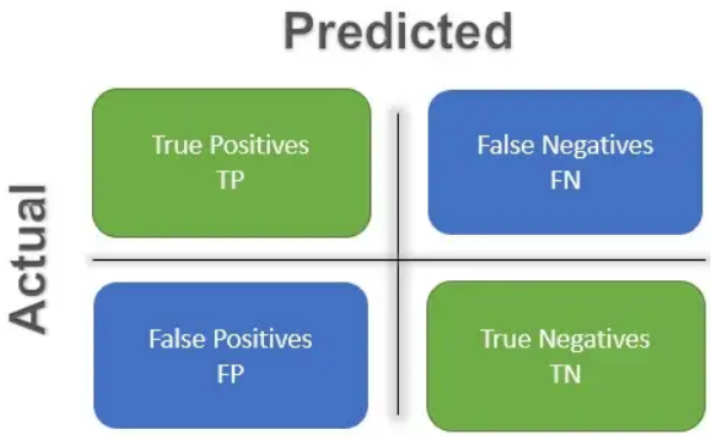
Training model adalah teknik untuk pembelajaran mendalam yang termasuk pemisahan *dataset*. *Training model* dilakukan dengan cara membagi *dataset* atau *splitting dataset,* data yang dikumpulkan untuk *training* biasanya dibagi menjadi dua atau tiga bagian yang berbeda, yaitu : *training*, validasi, dan *test*. Tipe pemisahan *training* dan testing biasanya menggunakan 70% data untuk pelatihan/*training* dan 30% untuk data pengujian/*testing*. Kita tidak dapat menggunakan set *testing* untuk evaluasi saat kita membangun model untuk menemukan parameter terbaik dalam mencegah kemungkinan *overfitting* ke set *testing*, untuk itu dibuatlah subset ketika yang disebut dengan validasi yang biasanya menggunakan 60% data untuk *training* berguna untuk menetapkan bias yang masuk ke dalam model, 20% untuk validasi berguna untuk mengevaluasi akurasi awal model melihat validasi tetapi tidak menggunakannya untuk menetapkan bobot bias, dan 20% untuk *testing* berguna sebagai evaluasi terakhir.

Pada dataset diabetes ini, kami membagi data dengan rasio 80% untuk *training set* dan 20% untuk *test set*, kemudian kami mengevaluasi model dan membangun model *machine learning*.

3.4 *Evaluate Machine Learning Models*

Mengevaluasi model berguna untuk mengetahui seberapa baik kinerja model yang dibuat untuk mendapatkan dan menetapkan hasil yang model yang terbaik. Setelah melakukan *splitting data,* langkah selanjutnya adalah membuat model dan mengevaluasi model. Saat melakukan prediksi suatu klasifikasi, maka akan ada empat jenis hasil. Sebuah *confusion matrix* yang menggambarkan kinerja model pengklasifikasi :

- *True positives* : ketika prediksi pengamatan milik kelas dan itu benar-benar milik kelas itu. Diprediksi positif dan sebenarnya positif.
- *True negatives* : ketika prediksi pengamatan bukan milik kelas dan sebenarnya bukan milik kelas itu. Diprediksi positif dan sebenarnya negatif.
- *False positives* : ketika prediksi pengamatan milik kelas padahal sebenarnya tidak. Prediksi negatif dan sebenarnya negatif.
- *False negatives* : ketika prediksi pengamatan bukan miliki kelas padahal sebenarnya milik kelas itu. Prediksi negatif dan sebenarnya positif.



Adapun tiga metrik utama yang digunakan dalam mengevaluasi model :

1. Akurasi
   Presentasi prediksi yang benar untuk data uji. Akurasi dilakukan dengan cara membagi jumlah prediksi yang benar dengan jumlah total prediksi.

$$\frac{TP + TN}{TP + FP + TN + FN}$$

2. Presisi
   Fraksi dari contoh yang relevan diantara semua contoh yang diperkirakan.

$$\frac{TP}{TP + FP}$$

3. Recall
   Fraksi contoh yang diprediksi menjadi milik kelas sehubugan dengan semua contoh yang benar-benar termasuk dalam kelas.

$$\frac{TP}{TP + FN}$$

Mengevaluasi model pada setiap model yang telah kami buat :

- *Random forest*

```
[[40  4]
 [10 13]]
              precision    recall  f1-score   support

           0       0.80      0.91      0.85        44
           1       0.76      0.57      0.65        23

    accuracy                           0.79        67
   macro avg       0.78      0.74      0.75        67
weighted avg       0.79      0.79      0.78        67
```

Confusion matrix :

n = 67	*Actual : no*	*Actual : yes*
Predicted : no	TN : 40	FP : 4
Predicted : yes	FN : 10	TP : 13

Akurasi :
13 + 40 / 40 + 10 + 4 + 13 = 0.79

- *Support Vector Machine (SVM)*

```
[[43  1]
 [17  6]]
              precision    recall  f1-score   support

           0       0.72      0.98      0.83        44
           1       0.86      0.26      0.40        23

    accuracy                           0.73        67
   macro avg       0.79      0.62      0.61        67
weighted avg       0.76      0.73      0.68        67
```

Confusion matrix :

n = 67	*Actual : no*	*Actual : yes*
Predicted : no	TN : 43	FP : 1
Predicted : yes	FN : 17	TP : 6

Akurasi :
6 + 43 / 43 + 17 + 1 + 6 = 0.73

- *Decision tree*

```
[[32 12]
 [10 13]]
              precision    recall  f1-score   support

           0       0.76      0.73      0.74        44
           1       0.52      0.57      0.54        23

    accuracy                           0.67        67
   macro avg       0.64      0.65      0.64        67
weighted avg       0.68      0.67      0.67        67
```

Confusion matrix :

n = 67	*Actual : no*	*Actual : yes*

Predicted : no	TN : 32	FP : 12
Predicted : yes	FN : 10	TP : 13

Akurasi :
13 + 32 / 32 + 10 + 12 + 13 =

- *Gaussian Naive Bayes*

```
[[39  5]
 [ 8 15]]
              precision    recall  f1-score   support

           0       0.83      0.89      0.86        44
           1       0.75      0.65      0.70        23

    accuracy                           0.81        67
   macro avg       0.79      0.77      0.78        67
weighted avg       0.80      0.81      0.80        67
```

Confusion matrix :

n = 67	*Actual : no*	*Actual : yes*
Predicted : no	TN : 39	FP : 5
Predicted : yes	FN : 8	TP : 15

Akurasi :
15 + 39 / 39 + 8 + 5 + 15 = 0.81

- *Logistic Regression*

```
[[43  1]
 [11 12]]
              precision    recall  f1-score   support

           0       0.80      0.98      0.88        44
           1       0.92      0.52      0.67        23

    accuracy                           0.82        67
   macro avg       0.86      0.75      0.77        67
weighted avg       0.84      0.82      0.81        67
```

Confusion matrix :

n = 67	*Actual : no*	*Actual : yes*
Predicted : no	TN : 43	FP : 1
Predicted : yes	FN : 11	TP : 12

Akurasi :

$12 + 43 / 43 + 11 + 1 + 12 = 0.82$

BAB IV
PEMBAHASAN

4.1 EDA

```
In [2]: diabet = pd.read_csv('diabetes.csv')
        diabet.head()
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Disini kita membaca *dataset* kita dan memasukkannya ke variabel diabet. Kemudian kita lihat bagian atas *dataset*nya dan kita mendapatkan sedikit gambaran tentang *dataset* kita. *Dataset* ini memiliki 9 kolom.

```
In [3]: diabet.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

Menggunakan *function* di atas, kita akan mendapatkan informasi-informasi mengenai *dataset* ini. *Dataset* ini memiliki 9 kolom yaitu *Pregnancies*, *Glucose*, *BloodPressure*, *SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, Age, dan Outcome*. *Dataset* ini semua datanya bersifat *non-null* atau tidak ada nilai null. Tipe datanya ada *integer* dan *float*. Untuk tipe data *integer* ada 7 kolom dan tipe data *float* ada 2 kolom.

```
In [4]: diabet.describe()
```

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

Menggunakan *function* di atas, kita akan mendapatkan informasi-infrormasi tentang jumlah data, rata-rata, standar deviasi, nilai minimal, kuartil pertama, kuartil kedua, kuartil ketiga, dan nilai maksimal dari masing-masing kolom.

```
In [5]: diabet = diabet.drop_duplicates()
```

Menggunakan *function* di atas, kita akan membuang duplikat-duplikat.

4.2 *Feature Engineering*

1. Imputasi

```
In [6]: diabet.isnull().sum()
Out[6]: Pregnancies                 0
        Glucose                     0
        BloodPressure               0
        SkinThickness               0
        Insulin                     0
        BMI                         0
        DiabetesPedigreeFunction    0
        Age                         0
        Outcome                     0
        dtype: int64
```

Menggunakan *function* di atas kita akan mendapatkan apakah ada nilai null di dataset ini

```
In [7]: diabetcopy = diabet.copy(deep = True)
        diabetcopy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = diabetcopy[['Glucose','BloodPressure','SkinThickness','
```

```
In [7]: ue)
        re','SkinThickness','Insulin','BMI']] = diabetcopy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']].replace(0,np.NaN)
```

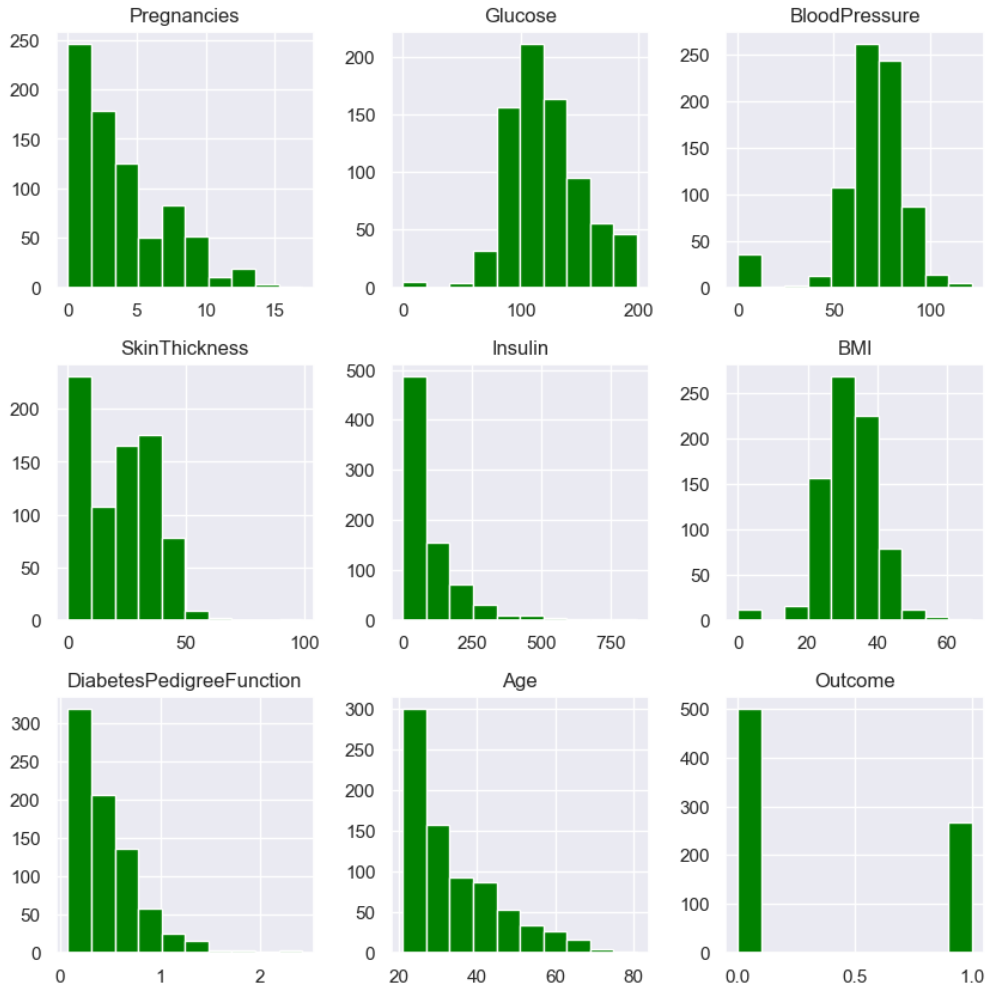
Disini pertama kita akan menyalin data diabet ke diabetcopy, kemudian kita mengganti semua nilai 0 menjadi NaN.

```
In [8]: print(diabetcopy.isnull().sum())
```

```
Pregnancies                 0
Glucose                     5
BloodPressure              35
SkinThickness             227
Insulin                   374
BMI                        11
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Kita mengecek apakah ada nilai null di diabetcopy dan ternyata ada di kolom *Glucose, BloodPressure, SkinThickness, Insulin, dan BMI.*

```
In [9]: # Mengecek apakah ada nilai 0 lewat visualization
        p = diabet.hist(color='green',figsize = (10,10))
```

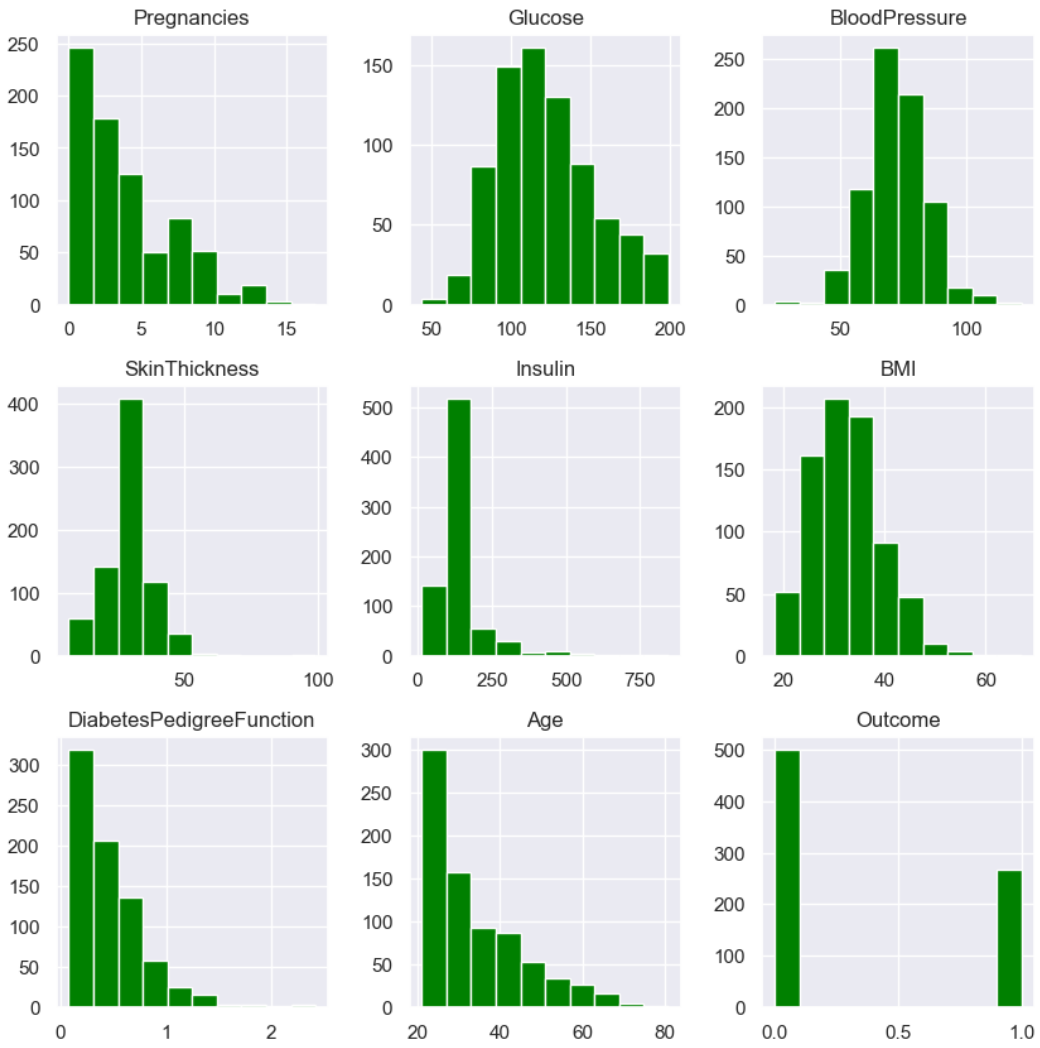


Untuk membuktikan apakah benar ada nilai *null*, kita coba lihat dengan visualisasi dan ternyata ada nilai *null*.

```
In [10]: diabetcopy['Glucose'].fillna(diabetcopy['Glucose'].mean(), inplace = True)
         diabetcopy['BloodPressure'].fillna(diabetcopy['BloodPressure'].mean(), inplace = True)
         diabetcopy['SkinThickness'].fillna(diabetcopy['SkinThickness'].median(), inplace = True)
         diabetcopy['Insulin'].fillna(diabetcopy['Insulin'].median(), inplace = True)
         diabetcopy['BMI'].fillna(diabetcopy['BMI'].median(), inplace = True)
```

Mengganti nilai null dengan rata-rata dan median.

```
In [11]: p = diabetcopy.hist(color='green',figsize = (10,10))
```



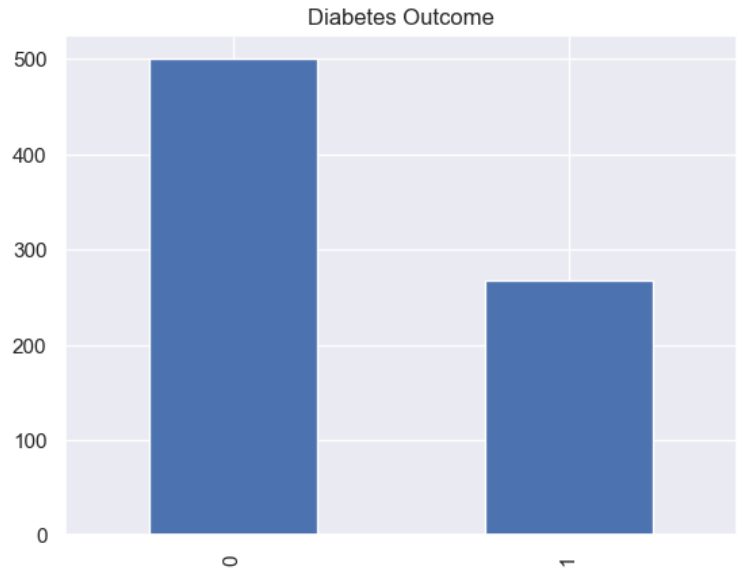
Setelah mengganti nilai null, kita lihat lagi dengan visualisasi dan hasilnya sudah tidak ada nilai null.

2. Menghitung *Value*

```
In [12]: diabetcopy.count()

Out[12]: Pregnancies                 768
         Glucose                     768
         BloodPressure               768
         SkinThickness               768
         Insulin                     768
         BMI                         768
         DiabetesPedigreeFunction    768
         Age                         768
         Outcome                     768
         dtype: int64
```

Menggunakan *function* di atas, kita mendapatkan jumlah data dari masing-masing kolom. Hasilnya sama seperti jumlah baris, yaitu 768.

```
In [13]: diabet.Outcome.value_counts()
         diabet['Outcome'].value_counts().plot(kind='bar').set_title('Diabetes Outcome')

Out[13]: Text(0.5, 1.0, 'Diabetes Outcome')
```



Menggunakan *function* di atas, kita dapat melihat visualisasi dari kolom *Outcome*. Hasil dari visualisasi di atas adalah ada pasien yang tidak menderita diabetes ditandai dengan nilai 0 dan

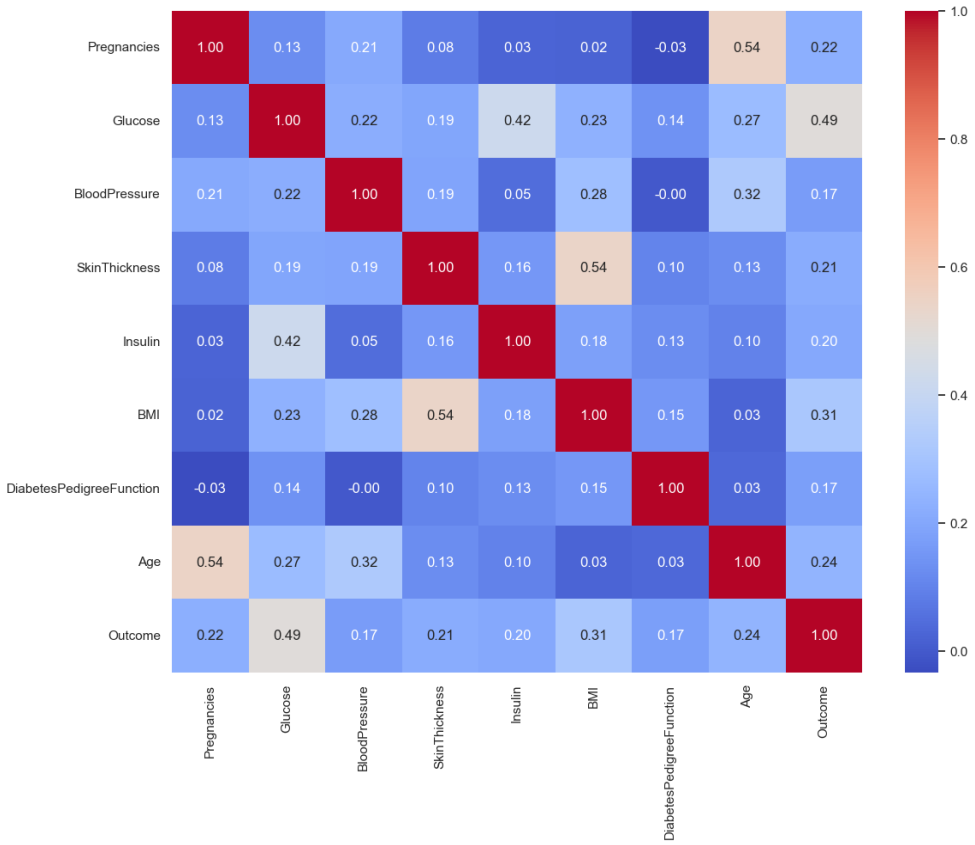
pasien yang menderita diabetes yang ditandai dengan nilai 1. Kita bisa melihat bahwa lebih banyak pasien yang tidak menderita diabetes daripada yang menderita.

```
In [14]: diabet.Outcome.value_counts()

Out[14]: 0    500
         1    268
         Name: Outcome, dtype: int64
```

Pasien yang tidak menderita diabetes tepatnya ada 500 orang dan yang menderita diabetes ada 268 orang.

3. Korelasi antar kolom

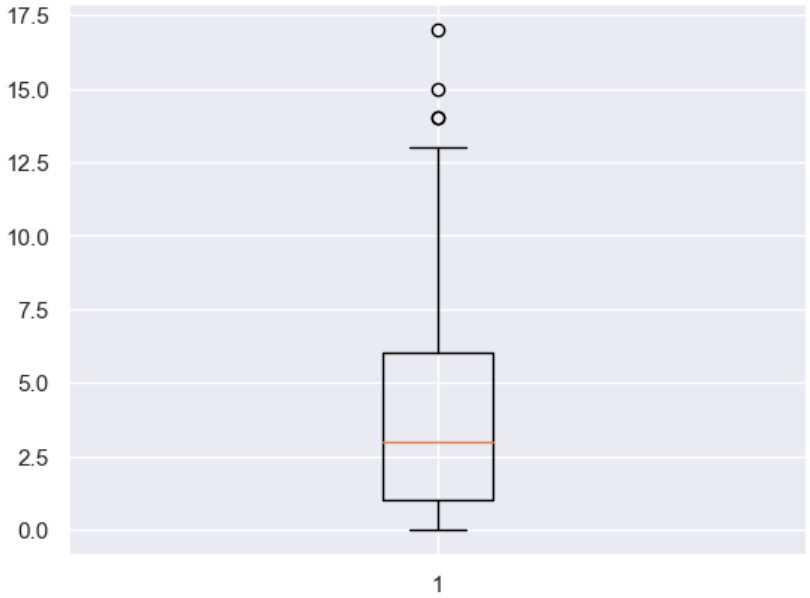
```
In [15]: plt.figure(figsize=(13,10))
         sns.heatmap(diabetcopy.corr(),annot=True, fmt = ".2f", cmap = "coolwarm")

Out[15]: <AxesSubplot:>
```



Menggunakan *function* di atas, kita mendapatkan visualisasi dari korelasi antar kolom. Korelasi tertinggi ada pada korelasi antara *SkinThickness* dan *BMI* dan korelasi antara *Pregnancies* dan *Age* yaitu 0.54.

4. *Outliers*
    - *Pregnancies outliers*

```
In [16]: plt.boxplot(diabetcopy['Pregnancies'])
```



Dari *boxplot* di atas kita dapat melihat ada beberapa *outlier* di atas nilai 12.5.

```
In [17]: q1 =diabetcopy['Pregnancies'].quantile(0.25)
         q3 = diabetcopy['Pregnancies'].quantile(0.75)
         iqr = q3-q1

         lower = q1-1.5*iqr
         upper = q3+1.5*iqr

         outlier_low = (diabetcopy['Pregnancies']<lower)
         outlier_upp = (diabetcopy['Pregnancies']>upper)
         diabetcopy['Pregnancies'][(outlier_low|outlier_upp)]
         diabetcopy['Pregnancies'][~(outlier_low|outlier_upp)]
         diabetcopy = diabetcopy[~(outlier_low|outlier_upp)]
         diabetcopy
```

Out[17]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35.0	125.0	33.6	0.627	50	1
1	1	85.0	66.0	29.0	125.0	26.6	0.351	31	0
2	8	183.0	64.0	29.0	125.0	23.3	0.672	32	1
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101.0	76.0	48.0	180.0	32.9	0.171	63	0
764	2	122.0	70.0	27.0	125.0	36.8	0.340	27	0
765	5	121.0	72.0	23.0	112.0	26.2	0.245	30	0
766	1	126.0	60.0	29.0	125.0	30.1	0.349	47	1
767	1	93.0	70.0	31.0	125.0	30.4	0.315	23	0

764 rows × 9 columns

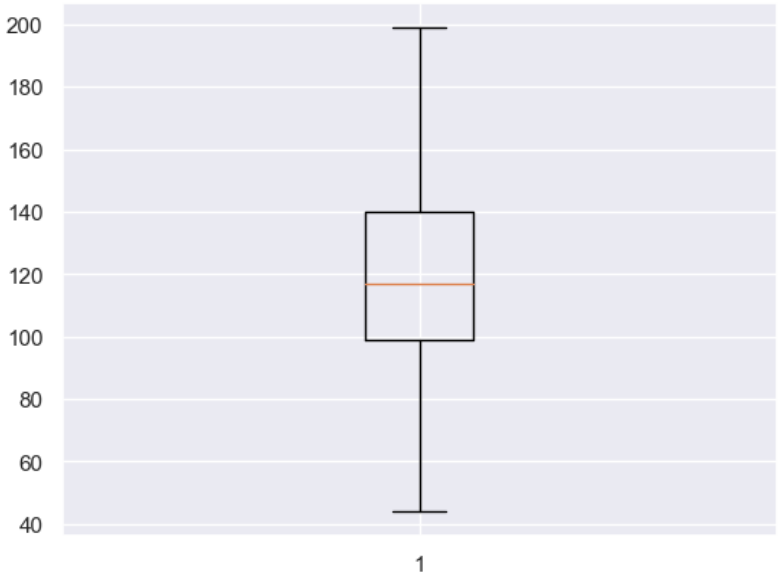
Menggunakan *function-function* di atas kita akan menangani *outlier-outlier* yang ada di kolom *Pregnancies*. Pertama kita masukkan nilai kuartil satu dan kuartil tiga ke variabel q1 dan q3. Kemudian kita mendapatkan nilai jarak interkuartil atau iqr dengan mengurangi nilai q3 dan q1 nya. Setelah itu kita tentukan batas atas dan bawah. Rumus batas atas adalah Q3 + 1.5 * IQR. dan rumus batas bawah adalah Q1–1.5 * IQR**.** Kemudian kita tentukan juga nilai outlier_low yang nilainya di bawah nilai batas bawah dan nilai outlier_upp yang nilainya di atas nilai batas atas. Kita menggunakan aturan Tukey dimana data-data diantara nilai batas bawah dan batas atas adalah nilai yang harus kita simpan, sedangkan nilai selain itu kita buang. Setelah kita buang nilai-nilainya, kita *update* data yang ada di diabetcopy dan sekarang baris yang tersisa adalah 764 baris.

```
In [18]: plt.boxplot(diabetcopy['Pregnancies'])
```

Setelah menghapus nilai *outlier*, kita buat lagi *boxplot* dari kolom *Pregnancies* dan hasilnya sudah tidak ada *outlier*.

- *Glucose outliers*

```
In [19]: plt.boxplot(diabetcopy['Glucose'])
```



Dari *boxplot* di atas, kita bisa melihat bahwa tidak ada nilai *outlier* pada kolom *Glucose*.

- *BloodPressure outliers*

```
In [20]: plt.boxplot(diabetcopy['BloodPressure'])
```

Dari *boxplot* di atas kita dapat melihat bahwa ada *outlier* di atas nilai 100 dan di bawah nilai 40.

In [21]:
```
q1 =diabetcopy['BloodPressure'].quantile(0.25)
q3 = diabetcopy['BloodPressure'].quantile(0.75)
iqr = q3-q1

lower = q1-1.5*iqr
upper = q3+1.5*iqr

outlier_low = (diabetcopy['BloodPressure']<lower)
outlier_upp = (diabetcopy['BloodPressure']>upper)
```

In [22]:
```
diabetcopy['BloodPressure'][(outlier_low|outlier_upp)]
```

In [23]:
```
diabetcopy['BloodPressure'][~(outlier_low|outlier_upp)]
```

Out[23]:
```
0      72.0
1      66.0
2      64.0
3      66.0
4      40.0
       ...
763    76.0
764    70.0
765    72.0
766    60.0
767    70.0
Name: BloodPressure, Length: 750, dtype: float64
```

In [24]:
```
diabetcopy = diabetcopy[~(outlier_low|outlier_upp)]
diabetcopy
```

Out[24]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35.0	125.0	33.6	0.627	50	1
1	1	85.0	66.0	29.0	125.0	26.6	0.351	31	0
2	8	183.0	64.0	29.0	125.0	23.3	0.672	32	1
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101.0	76.0	48.0	180.0	32.9	0.171	63	0
764	2	122.0	70.0	27.0	125.0	36.8	0.340	27	0
765	5	121.0	72.0	23.0	112.0	26.2	0.245	30	0
766	1	126.0	60.0	29.0	125.0	30.1	0.349	47	1
767	1	93.0	70.0	31.0	125.0	30.4	0.315	23	0

750 rows × 9 columns

Menggunakan *function-function* di atas kita akan menangani *outlier-outlier* yang ada di kolom *BloodPressure* . Pertama kita masukkan nilai kuartil satu dan kuartil tiga ke variabel q1 dan q3.

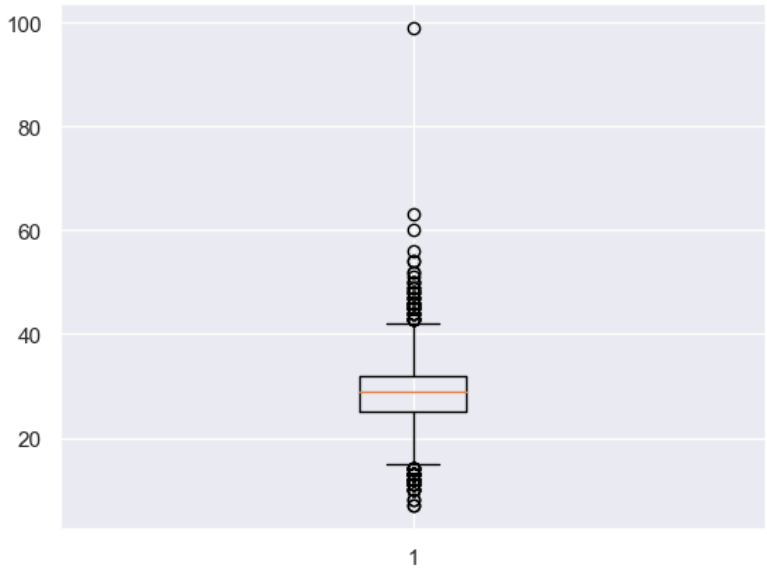
Kemudian kita mendapatkan nilai jarak interkuartil atau iqr dengan mengurangi nilai q3 dan q1 nya. Setelah itu kita tentukan batas atas dan bawah. Rumus batas atas adalah Q3 + 1.5 * IQR. dan rumus batas bawah adalah Q1–1.5 * IQR**.** Kemudian kita tentukan juga nilai outlier_low yang nilainya di bawah nilai batas bawah dan nilai outlier_upp yang nilainya di atas nilai batas atas. Kita menggunakan aturan Tukey dimana data-data diantara nilai batas bawah dan batas atas adalah nilai yang harus kita simpan, sedangkan nilai selain itu kita buang. Setelah kita buang nilai-nilainya, kita *update* data yang ada di diabetcopy dan sekarang baris yang tersisa adalah 750 baris.

```
In [25]: plt.boxplot(diabetcopy['BloodPressure'])
```

Setelah menghapus nilai *outlier*, kita buat lagi boxplot dari kolom *BloodPressure* dan hasilnya sudah tidak ada *outlier*.

- *SkinThickness outliers*

```
In [26]: plt.boxplot(diabetcopy['SkinThickness'])
```



Dari *boxplot* di atas kita dapat melihat bahwa ada *outlier* di atas nilai 40 dan di bawah nilai 20.

In [27]:
```
q1 =diabetcopy['SkinThickness'].quantile(0.25)
q3 = diabetcopy['SkinThickness'].quantile(0.75)
iqr = q3-q1

lower = q1-1.5*iqr
upper = q3+1.5*iqr

outlier_low = (diabetcopy['SkinThickness']<lower)
outlier_upp = (diabetcopy['SkinThickness']>upper)
```

In [28]:
```
diabetcopy['SkinThickness'][(outlier_low|outlier_upp)]
```

```
In [29]: diabetcopy['SkinThickness'][~(outlier_low|outlier_upp)]
```

```
Out[29]: 0      35.0
         1      29.0
         2      29.0
         3      23.0
         4      35.0
                ...
         762    29.0
         764    27.0
         765    23.0
         766    29.0
         767    31.0
         Name: SkinThickness, Length: 665, dtype: float64
```

```
In [30]: diabetcopy = diabetcopy[~(outlier_low|outlier_upp)]
         diabetcopy
```

Out[30]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35.0	125.0	33.6	0.627	50	1
1	1	85.0	66.0	29.0	125.0	26.6	0.351	31	0
2	8	183.0	64.0	29.0	125.0	23.3	0.672	32	1
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
762	9	89.0	62.0	29.0	125.0	22.5	0.142	33	0
764	2	122.0	70.0	27.0	125.0	36.8	0.340	27	0
765	5	121.0	72.0	23.0	112.0	26.2	0.245	30	0
766	1	126.0	60.0	29.0	125.0	30.1	0.349	47	1
767	1	93.0	70.0	31.0	125.0	30.4	0.315	23	0

665 rows × 9 columns

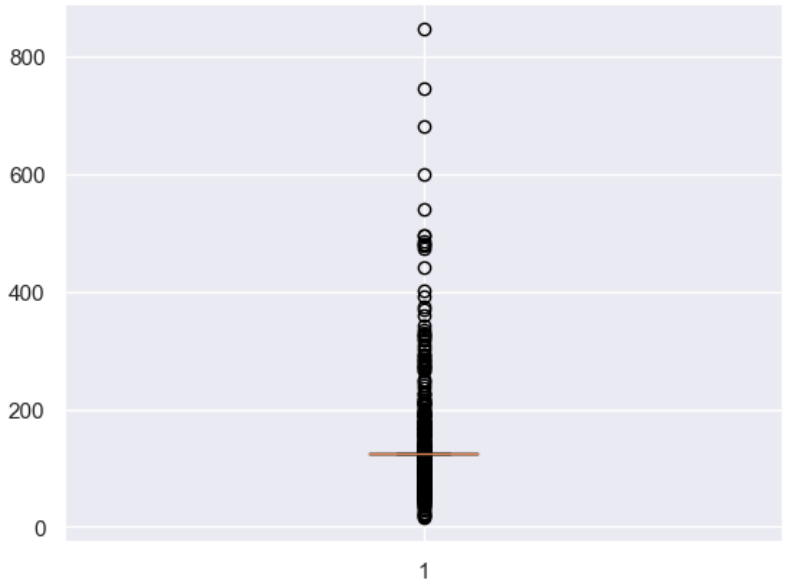
Menggunakan *function-function* di atas kita akan menangani *outlier-outlier* yang ada di kolom *SkinThickness* . Pertama kita masukkan nilai kuartil satu dan kuartil tiga ke variabel q1 dan q3. Kemudian kita mendapatkan nilai jarak interkuartil atau iqr dengan mengurangi nilai q3 dan q1 nya. Setelah itu kita tentukan batas atas dan bawah. Rumus batas atas adalah Q3 + 1.5 * IQR. dan rumus batas bawah adalah Q1–1.5 * IQR**.** Kemudian kita tentukan juga nilai outlier_low yang nilainya di bawah nilai batas bawah dan nilai outlier_upp yang nilainya di atas nilai batas atas. Kita menggunakan aturan Tukey dimana data-data diantara nilai batas bawah dan batas atas adalah nilai yang harus kita simpan, sedangkan nilai selain itu kita buang. Setelah kita buang nilai-nilainya, kita *update* data yang ada di diabetcopy dan sekarang baris yang tersisa adalah 665 baris.

```
In [31]: plt.boxplot(diabetcopy['SkinThickness'])
```

Setelah menghapus beberapa nilai *outlier*, kita buat lagi *boxplot* dari kolom *SkinThickness* dan hasilnya *outlier* yang ada tidak sebanyak sebelumnya.

- *Insulin outliers*

```
In [32]: plt.boxplot(diabetcopy['Insulin'])
```



Dari *boxplot* di atas kita dapat melihat bahwa ada *outlier* di atas nilai 200 dan di bawah nilai 200.

In [33]:
```
q1 =diabetcopy['Insulin'].quantile(0.25)
q3 = diabetcopy['Insulin'].quantile(0.75)
iqr = q3-q1

lower = q1-1.5*iqr
upper = q3+1.5*iqr

outlier_low = (diabetcopy['Insulin']<lower)
outlier_upp = (diabetcopy['Insulin']>upper)
```

In [34]:
```
diabetcopy['Insulin'][(outlier_low|outlier_upp)]
```

In [35]:
```
diabetcopy['Insulin'][~(outlier_low|outlier_upp)]
```

Out[35]:
```
0      125.0
1      125.0
2      125.0
5      125.0
7      125.0
       ...
761    125.0
762    125.0
764    125.0
766    125.0
767    125.0
Name: Insulin, Length: 358, dtype: float64
```

In [36]:
```
diabetcopy = diabetcopy[~(outlier_low|outlier_upp)]
diabetcopy
```

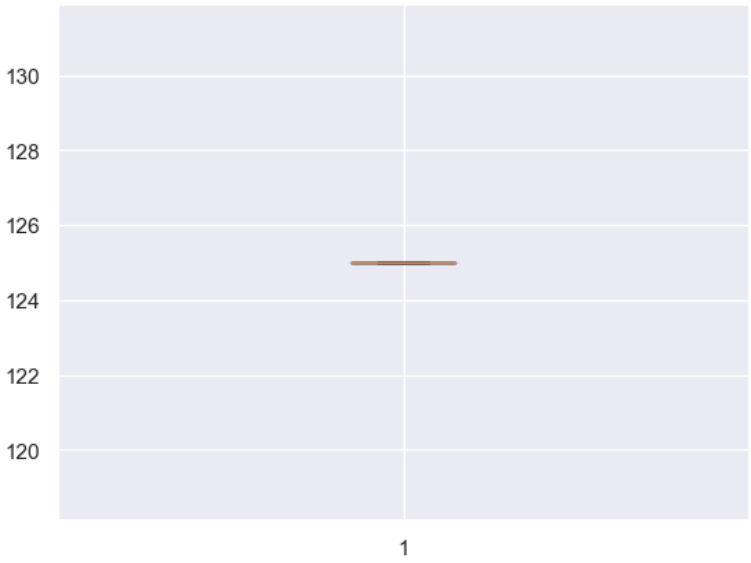
Out[36]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.000000	35.0	125.0	33.6	0.627	50	1
1	1	85.0	66.000000	29.0	125.0	26.6	0.351	31	0
2	8	183.0	64.000000	29.0	125.0	23.3	0.672	32	1
5	5	116.0	74.000000	29.0	125.0	25.6	0.201	30	0
7	10	115.0	72.405184	29.0	125.0	35.3	0.134	29	0
...	...	...	...	...	...	...	...	...	...
761	9	170.0	74.000000	31.0	125.0	44.0	0.403	43	1
762	9	89.0	62.000000	29.0	125.0	22.5	0.142	33	0
764	2	122.0	70.000000	27.0	125.0	36.8	0.340	27	0
766	1	126.0	60.000000	29.0	125.0	30.1	0.349	47	1
767	1	93.0	70.000000	31.0	125.0	30.4	0.315	23	0

358 rows × 9 columns

Menggunakan *function-function* di atas kita akan menangani *outlier-outlier* yang ada di kolom Insulin. Pertama kita masukkan nilai kuartil satu dan kuartil tiga ke variabel q1 dan q3. Kemudian kita mendapatkan nilai jarak interkuartil atau iqr dengan mengurangi nilai q3 dan q1 nya. Setelah itu kita tentukan batas atas dan bawah. Rumus batas atas adalah Q3 + 1.5 * IQR. dan rumus batas bawah adalah Q1–1.5 * IQR**.** Kemudian kita tentukan juga nilai outlier_low yang nilainya di bawah nilai batas bawah dan nilai outlier_upp yang nilainya di atas nilai batas atas. Kita menggunakan aturan Tukey dimana data-data diantara nilai batas bawah dan batas atas adalah nilai yang harus kita simpan, sedangkan nilai selain itu kita buang. Setelah kita buang nilai-nilainya, kita *update* data yang ada di diabetcopy dan sekarang baris yang tersisa adalah 358 baris.

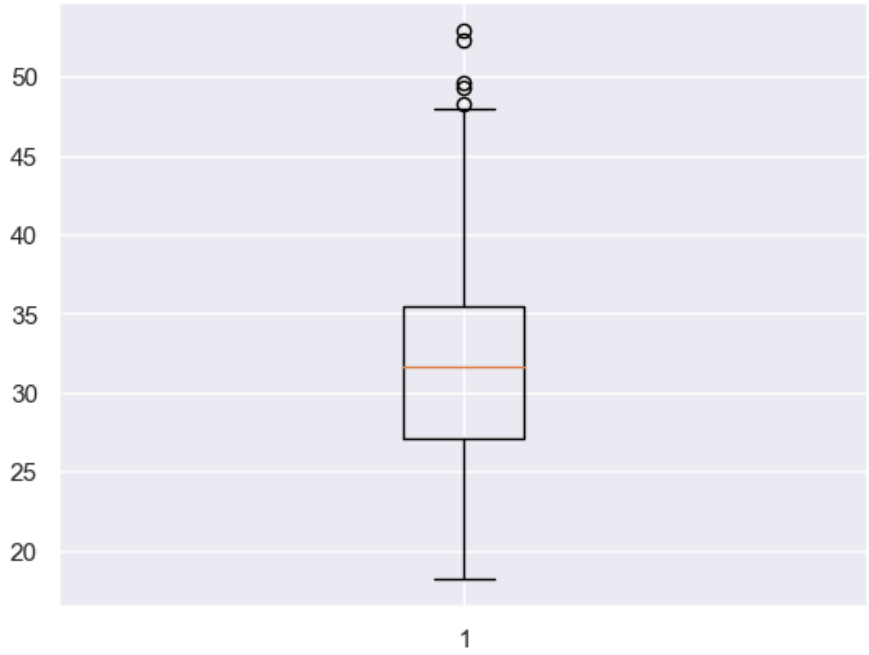
```
In [37]: plt.boxplot(diabetcopy['Insulin'])
```



Setelah menghapus nilai *outlier*, kita buat lagi boxplot dari kolom Insulin dan hasilnya sudah tidak ada *outlier*.

- *BMI outliers*

```
In [38]: plt.boxplot(diabetcopy['BMI'])
```



Dari *boxplot* di atas kita dapat melihat bahwa ada *outlier* di atas nilai 45.

In [39]:
```
q1 =diabetcopy['BMI'].quantile(0.25)
q3 = diabetcopy['BMI'].quantile(0.75)
iqr = q3-q1

lower = q1-1.5*iqr
upper = q3+1.5*iqr

outlier_low = (diabetcopy['BMI']<lower)
outlier_upp = (diabetcopy['BMI']>upper)
diabetcopy['BMI'][(outlier_low|outlier_upp)]
diabetcopy['BMI'][~(outlier_low|outlier_upp)]
diabetcopy = diabetcopy[~(outlier_low|outlier_upp)]
diabetcopy
```

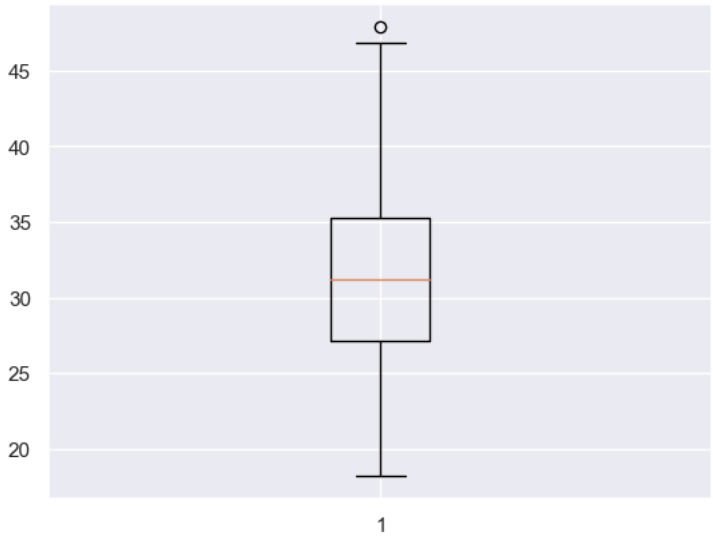
Out[39]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.000000	35.0	125.0	33.6	0.627	50	1
1	1	85.0	66.000000	29.0	125.0	26.6	0.351	31	0
2	8	183.0	64.000000	29.0	125.0	23.3	0.672	32	1
5	5	116.0	74.000000	29.0	125.0	25.6	0.201	30	0
7	10	115.0	72.405184	29.0	125.0	35.3	0.134	29	0
...	...	...	...	...	...	...	...	...	...
761	9	170.0	74.000000	31.0	125.0	44.0	0.403	43	1
762	9	89.0	62.000000	29.0	125.0	22.5	0.142	33	0
764	2	122.0	70.000000	27.0	125.0	36.8	0.340	27	0
766	1	126.0	60.000000	29.0	125.0	30.1	0.349	47	1
767	1	93.0	70.000000	31.0	125.0	30.4	0.315	23	0

353 rows × 9 columns

Menggunakan *function-function* di atas kita akan menangani *outlier-outlier* yang ada di kolom BMI. Pertama kita masukkan nilai kuartil satu dan kuartil tiga ke variabel q1 dan q3. Kemudian kita mendapatkan nilai jarak interkuartil atau iqr dengan mengurangi nilai q3 dan q1 nya. Setelah itu kita tentukan batas atas dan bawah. Rumus batas atas adalah Q3 + 1.5 * IQR. dan rumus batas bawah adalah Q1–1.5 * IQR**.** Kemudian kita tentukan juga nilai outlier_low yang nilainya di bawah nilai batas bawah dan nilai outlier_upp yang nilainya di atas nilai batas atas. Kita menggunakan aturan Tukey dimana data-data diantara nilai batas bawah dan batas atas adalah nilai yang harus kita simpan, sedangkan nilai selain itu kita buang. Setelah kita buang nilai-nilainya, kita *update* data yang ada di diabetcopy dan sekarang baris yang tersisa adalah 353 baris.

In [40]:
```
plt.boxplot(diabetcopy['BMI'])
```

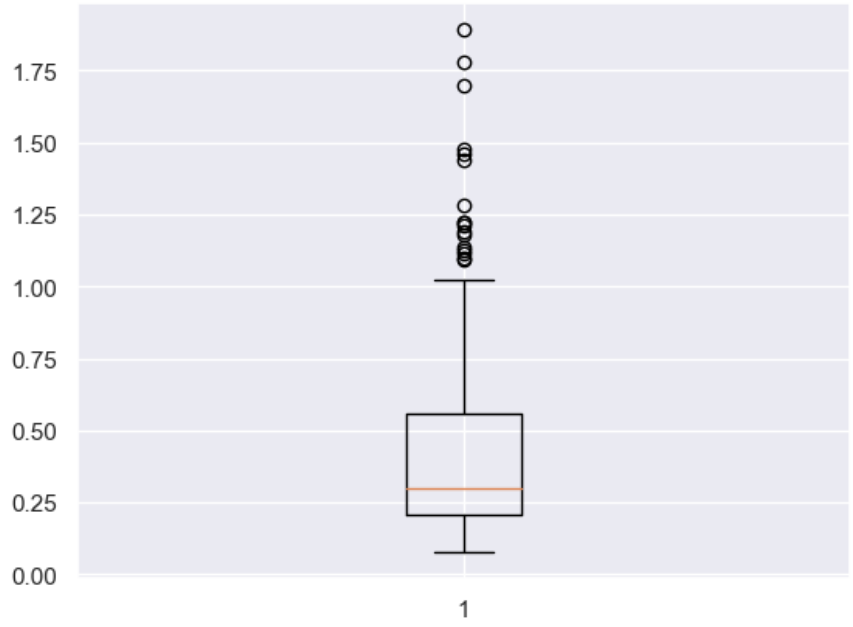


Setelah menghapus beberapa nilai *outlier*, kita buat lagi *boxplot* dari kolom BMI dan hasilnya *outlier* yang ada hanya satu.

.

- *DiabetesPedigreeFunction Outliers*

```
In [41]: plt.boxplot(diabetcopy['DiabetesPedigreeFunction'])
```



Dari *boxplot* di atas kita dapat melihat bahwa ada *outlier* di atas nilai 1.00.

In [42]:
```
q1 =diabetcopy['DiabetesPedigreeFunction'].quantile(0.25)
q3 = diabetcopy['DiabetesPedigreeFunction'].quantile(0.75)
iqr = q3-q1

lower = q1-1.5*iqr
upper = q3+1.5*iqr

outlier_low = (diabetcopy['DiabetesPedigreeFunction']<lower)
outlier_upp = (diabetcopy['DiabetesPedigreeFunction']>upper)
diabetcopy['DiabetesPedigreeFunction'][(outlier_low|outlier_upp)]
diabetcopy['DiabetesPedigreeFunction'][~(outlier_low|outlier_upp)]
diabetcopy = diabetcopy[~(outlier_low|outlier_upp)]
diabetcopy
```

Out[42]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.000000	35.0	125.0	33.6	0.627	50	1
1	1	85.0	66.000000	29.0	125.0	26.6	0.351	31	0
2	8	183.0	64.000000	29.0	125.0	23.3	0.672	32	1
5	5	116.0	74.000000	29.0	125.0	25.6	0.201	30	0
7	10	115.0	72.405184	29.0	125.0	35.3	0.134	29	0
...	...	...	...	...	...	...	...	...	...
761	9	170.0	74.000000	31.0	125.0	44.0	0.403	43	1
762	9	89.0	62.000000	29.0	125.0	22.5	0.142	33	0
764	2	122.0	70.000000	27.0	125.0	36.8	0.340	27	0
766	1	126.0	60.000000	29.0	125.0	30.1	0.349	47	1
767	1	93.0	70.000000	31.0	125.0	30.4	0.315	23	0

336 rows × 9 columns

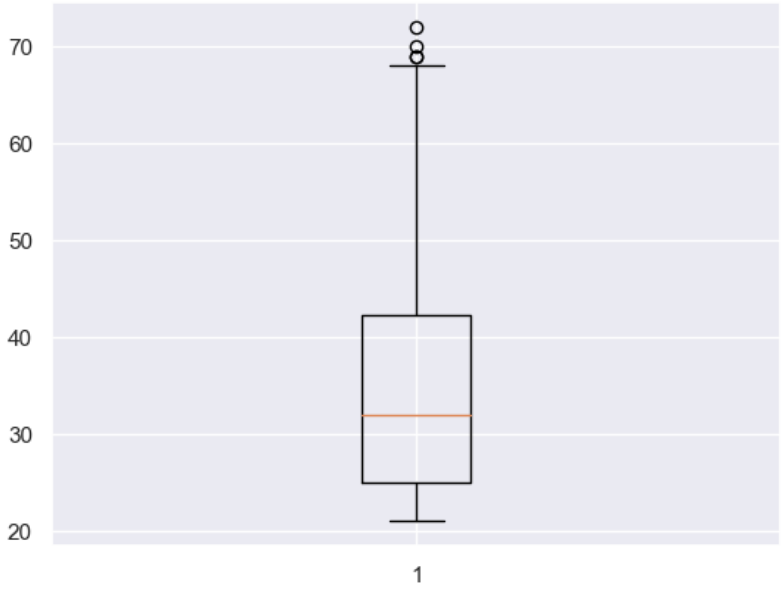
Menggunakan *function-function* di atas kita akan menangani *outlier-outlier* yang ada di kolom *DiabetesPedigreeFunction* . Pertama kita masukkan nilai kuartil satu dan kuartil tiga ke variabel q1 dan q3. Kemudian kita mendapatkan nilai jarak interkuartil atau iqr dengan mengurangi nilai q3 dan q1 nya. Setelah itu kita tentukan batas atas dan bawah. Rumus batas atas adalah Q3 + 1.5 * IQR. dan rumus batas bawah adalah Q1–1.5 * IQR**.** Kemudian kita tentukan juga nilai outlier_low yang nilainya di bawah nilai batas bawah dan nilai outlier_upp yang nilainya di atas nilai batas atas. Kita menggunakan aturan Tukey dimana data-data diantara nilai batas bawah dan batas atas adalah nilai yang harus kita simpan, sedangkan nilai selain itu kita buang. Setelah kita buang nilai-nilainya, kita *update* data yang ada di diabetcopy dan sekarang baris yang tersisa adalah 336 baris.

In [43]:
```
plt.boxplot(diabetcopy['DiabetesPedigreeFunction'])
```

Setelah menghapus beberapa nilai *outlier*, kita buat lagi *boxplot* dari kolom *DiabetesPedigreeFunction* dan hasilnya outlier yang ada hanya dua.

- *Age outliers*

```
In [44]:  plt.boxplot(diabetcopy['Age'])
```



In [45]:
```
q1 =diabetcopy['Age'].quantile(0.25)
q3 = diabetcopy['Age'].quantile(0.75)
iqr = q3-q1

lower = q1-1.5*iqr
upper = q3+1.5*iqr

outlier_low = (diabetcopy['Age']<lower)
outlier_upp = (diabetcopy['Age']>upper)
diabetcopy['Age'][(outlier_low|outlier_upp)]
diabetcopy['Age'][~(outlier_low|outlier_upp)]
diabetcopy = diabetcopy[~(outlier_low|outlier_upp)]
diabetcopy
```

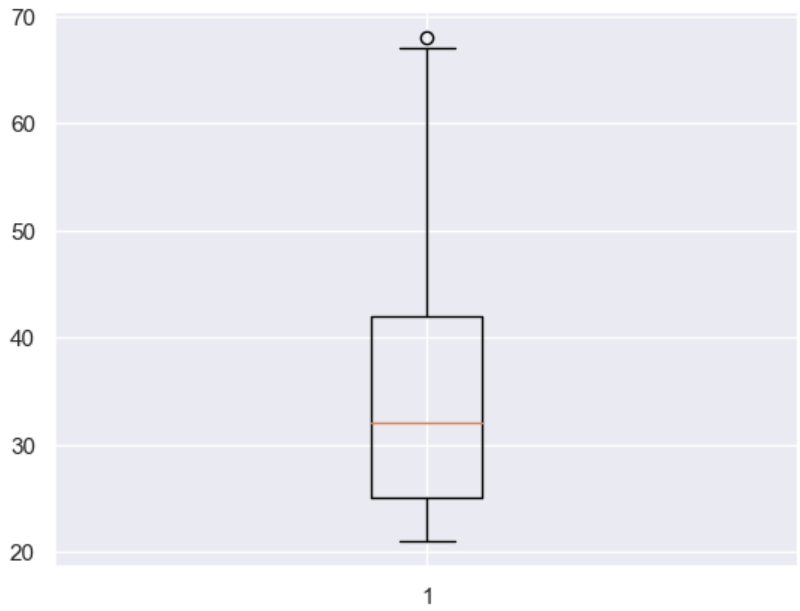
Out[45]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.000000	35.0	125.0	33.6	0.627	50	1
1	1	85.0	66.000000	29.0	125.0	26.6	0.351	31	0
2	8	183.0	64.000000	29.0	125.0	23.3	0.672	32	1
5	5	116.0	74.000000	29.0	125.0	25.6	0.201	30	0
7	10	115.0	72.405184	29.0	125.0	35.3	0.134	29	0
...	...	...	...	...	...	...	...	...	...
761	9	170.0	74.000000	31.0	125.0	44.0	0.403	43	1
762	9	89.0	62.000000	29.0	125.0	22.5	0.142	33	0
764	2	122.0	70.000000	27.0	125.0	36.8	0.340	27	0
766	1	126.0	60.000000	29.0	125.0	30.1	0.349	47	1
767	1	93.0	70.000000	31.0	125.0	30.4	0.315	23	0

332 rows × 9 columns

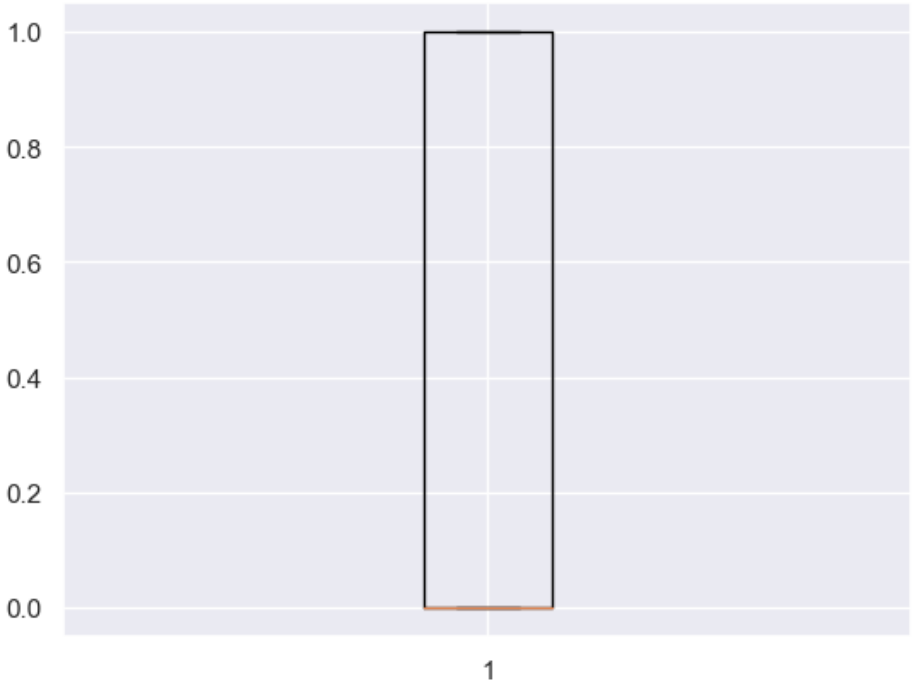
Menggunakan *function-function* di atas kita akan menangani *outlier-outlier* yang ada di kolom *Age*. Pertama kita masukkan nilai kuartil satu dan kuartil tiga ke variabel q1 dan q3. Kemudian kita mendapatkan nilai jarak interkuartil atau iqr dengan mengurangi nilai q3 dan q1 nya. Setelah itu kita tentukan batas atas dan bawah. Rumus batas atas adalah Q3 + 1.5 * IQR. dan rumus batas bawah adalah Q1–1.5 * IQR**.** Kemudian kita tentukan juga nilai outlier_low yang nilainya di bawah nilai batas bawah dan nilai outlier_upp yang nilainya di atas nilai batas atas. Kita menggunakan aturan Tukey dimana data-data diantara nilai batas bawah dan batas atas adalah nilai yang harus kita simpan, sedangkan nilai selain itu kita buang. Setelah kita buang nilai-nilainya, kita *update* data yang ada di diabetcopy dan sekarang baris yang tersisa adalah 332 baris.

In [46]:
```
plt.boxplot(diabetcopy['Age'])
```



- *Outcome outliers*

```
In [47]:    plt.boxplot(diabetcopy['Outcome'])
```



```
In [48]: diabetcopy.shape
Out[48]: (332, 9)
```

Setelah menghapus banyak *outliers*, baris yang ada sekarang ada 332 baris.

5. *Scaling*

```
In [49]: diabetcopy.head()
```

Out[49]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.000000	35.0	125.0	33.6	0.627	50	1
1	1	85.0	66.000000	29.0	125.0	26.6	0.351	31	0
2	8	183.0	64.000000	29.0	125.0	23.3	0.672	32	1
5	5	116.0	74.000000	29.0	125.0	25.6	0.201	30	0
7	10	115.0	72.405184	29.0	125.0	35.3	0.134	29	0

Menggunakan *function* di atas, kita akan mendapatkan gambaran bagian atas dari dataset kita

```
In [50]: sc_X = StandardScaler()
X =  pd.DataFrame(sc_X.fit_transform(diabetcopy.drop(["Outcome"],axis = 1),), columns=['Pregnancies',
'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age'])
X.head()
```

Out[50]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	0.490124	1.001053	-0.157383	1.439678	0.0	0.366217	1.204203	1.186805
1	-1.005476	-1.160124	-0.745165	0.015726	0.0	-0.790311	-0.077565	-0.355186
2	1.088364	2.201707	-0.941092	0.015726	0.0	-1.335532	1.413186	-0.274028
3	0.191004	-0.096687	0.038545	0.015726	0.0	-0.955530	-0.774177	-0.436343
4	1.686604	-0.130992	-0.117689	0.015726	0.0	0.647089	-1.085331	-0.517501

Setelah proses *scaling*, hasil yang didapatkan seperti ini.

4.3 *Model Building*

Kami membagi data dengan rasio 80% untuk *training set* dan 20% untuk *test set.*

```
In [51]: X = diabetcopy.drop('Outcome', axis=1)
y = diabetcopy['Outcome']

# split data to 80:20 ratio for train/test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.2, random_state=SEED, stratify=y)
```

a. *Random Forest*

In [52]:
```
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n_estimators=200)
rfc.fit(X_train, y_train)
```

Out[52]:
```
RandomForestClassifier(n_estimators=200)
```

In [53]:
```
#Check the accuracy score for random forest
from sklearn import metrics

predictions = rfc.predict(X_test)
print("Accuracy_Score =", format(metrics.accuracy_score(y_test, predictions)))
```

```
Accuracy_Score = 0.7910447761194029
```

In [54]:
```
from sklearn.metrics import classification_report, confusion_matrix

print(confusion_matrix(y_test, predictions))
print(classification_report(y_test,predictions))
```

```
[[40  4]
 [10 13]]
              precision    recall  f1-score   support

           0       0.80      0.91      0.85        44
           1       0.76      0.57      0.65        23

    accuracy                           0.79        67
   macro avg       0.78      0.74      0.75        67
weighted avg       0.79      0.79      0.78        67
```

Pertama kita membuat model menggunakan *function* RandomForestClassifier, kemudian kita cek akurasinya dan kita mendapat hasilnya 0.79. Setelah itu kita buat *classification report* dan *confusion matrix*nya.

b. *Support Vector Machine (SVM)*

```
In [55]: from sklearn.svm import SVC

         svc_model = SVC()
         svc_model.fit(X_train, y_train)
```

```
Out[55]: SVC()
```

```
In [56]: svc_pred = svc_model.predict(X_test)
```

```
In [57]: from sklearn import metrics

         print("Accuracy Score =", format(metrics.accuracy_score(y_test, svc_pred)))
```

```
Accuracy Score = 0.7313432835820896
```

```
In [58]: from sklearn.metrics import classification_report, confusion_matrix

         print(confusion_matrix(y_test, svc_pred))
         print(classification_report(y_test,svc_pred))
```

```
[[43  1]
 [17  6]]
              precision    recall  f1-score   support

           0       0.72      0.98      0.83        44
           1       0.86      0.26      0.40        23

    accuracy                           0.73        67
   macro avg       0.79      0.62      0.61        67
weighted avg       0.76      0.73      0.68        67
```

Pertama kita membuat model menggunakan *function* SVC, kemudian kita cek akurasinya dan kita mendapat hasilnya 0.73. Setelah itu kita buat *classification report* dan *confusion matrix*nya.

c. *Decision Tree*

```
In [59]: from sklearn.tree import DecisionTreeClassifier

         dtree = DecisionTreeClassifier()
         dtree.fit(X_train, y_train)
```

```
Out[59]: DecisionTreeClassifier()
```

```
In [60]: from sklearn import metrics

         predictions = dtree.predict(X_test)
         print("Accuracy Score =", format(metrics.accuracy_score(y_test,predictions)))
```

```
Accuracy Score = 0.6716417910447762
```

```
In [61]: from sklearn.metrics import classification_report, confusion_matrix

         print(confusion_matrix(y_test, predictions))
         print(classification_report(y_test,predictions))
```

```
[[32 12]
 [10 13]]
              precision    recall  f1-score   support

           0       0.76      0.73      0.74        44
           1       0.52      0.57      0.54        23

    accuracy                           0.67        67
   macro avg       0.64      0.65      0.64        67
weighted avg       0.68      0.67      0.67        67
```

Pertama kita membuat model menggunakan *function* DecisionTreeClassifier, kemudian kita cek akurasinya dan kita mendapat hasilnya 0.67. Setelah itu kita buat *classification report* dan *confusion matrix*nya.

d. *Gaussian Naive Bayes*

```
In [62]: from sklearn.naive_bayes import GaussianNB
         gnb = GaussianNB()
         gnb.fit(X_train,y_train)

Out[62]: GaussianNB()

In [63]: from sklearn import metrics

         predictions = gnb.predict(X_test)
         print("Accuracy Score =", format(metrics.accuracy_score(y_test,predictions)))

         Accuracy Score = 0.8059701492537313

In [64]: from sklearn.metrics import confusion_matrix

         print(confusion_matrix(y_test, predictions))
         print(classification_report(y_test,predictions))
```

```
[[39  5]
 [ 8 15]]
              precision    recall  f1-score   support

           0       0.83      0.89      0.86        44
           1       0.75      0.65      0.70        23

    accuracy                           0.81        67
   macro avg       0.79      0.77      0.78        67
weighted avg       0.80      0.81      0.80        67
```

Pertama kita membuat model menggunakan *function* GaussianNB, kemudian kita cek akurasinya dan kita mendapat hasilnya 0.81. Setelah itu kita buat *classification report* dan *confusion matrix*nya.

e. *Logistic Regression*

```
In [65]: from sklearn.linear_model import LogisticRegression
         logreg = LogisticRegression()
         logreg.fit(X_train, y_train)

Out[65]: LogisticRegression()

In [66]: from sklearn import metrics

         predictions = logreg.predict(X_test)
         print("Accuracy Score =", format(metrics.accuracy_score(y_test,predictions)))

         Accuracy Score = 0.8208955223880597

In [67]: from sklearn.metrics import confusion_matrix

         print(confusion_matrix(y_test, predictions))
         print(classification_report(y_test,predictions))

[[43  1]
 [11 12]]
              precision    recall  f1-score   support

           0       0.80      0.98      0.88        44
           1       0.92      0.52      0.67        23

    accuracy                           0.82        67
   macro avg       0.86      0.75      0.77        67
weighted avg       0.84      0.82      0.81        67
```

Pertama kita membuat model menggunakan *function* LogisticRegression, kemudian kita cek akurasinya dan kita mendapat hasilnya 0.82. Setelah itu kita buat *classification report* dan *confusion matrix*nya.

Kami telah membuat lima model. Dari kelima model di atas kita bisa menarik kesimpulan bahwa model *logistic regression* adalah yang terbaik karena memilki akurasi tertinggi yaitu 0.82.

4.4 Menyimpan model *logistic regression*

```
In [68]: import pickle
         # save the model using pickle
         savedmodel = pickle.dumps(logreg)
```

```
In [69]: # loading that saved model
         logreg_from_pickle = pickle.loads(savedmodel)
```

```
In [70]: # make predictions
         logreg_from_pickle.predict(X_test)
```

```
Out[70]: array([0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1,
                0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
                0], dtype=int64)
```

Kita akan menyimpan model *logistic regression* menggunakan pickle. Kemudian membuat *predictions*.

```
In [71]: diabet.head()
```

Out[71]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
In [72]: diabet.tail()
```

Out[72]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

```
In [73]: # make predictions based on patient no 4
         logreg.predict([[0,137,40,35,168,43.1,2.228,33]])
```

```
Out[73]: array([1], dtype=int64)
```

Patient is predicted to have diabetes.

```
In [74]: # make predictions based on patient no 763
         logreg.predict([[10,101,76,48,180,32.9,0.171,63]])
```

```
Out[74]: array([0], dtype=int64)
```

Patient is predicted not to have diabetes.

Setelah menyimpan modelnya, kita coba apakah model kita cukup baik untuk membuat prediksi. Kita ambil data dari bagian atas, data nomor empat dan data dari bagian bawah, data nomor 763. Hasilnya sesuai dengan prediksi yang artinya model kita memiliki tingkat prediksi yang baik.

BAB V
KESIMPULAN

Dalam melakukan prediksi untuk menentukan pasien penderita diabetes, kami membangun beberapa model dengan metode-metode yang berbeda untuk mendapatkan hasil yang terbaik, dengan beberapa analisis data dan *feature engineering* yang dilakukan sebelum membangun model-model tersebut. Pada dataset diabetes dengan semua catatan pasien yang ada pada atribut-atribut ini, kami membagi data dengan rasio 80% untuk training set dan 20% untuk test set, kemudian kami mengevaluasi model dan membangun model *machine learning* untuk memprediksi apakah pasien dalam kumpulan data tersebut menderita diabetes atau tidak, kami menggunakan lima model, diantaranya *Random Forest, Support Vector Machine (SVM)*, *Decision Tree, Gaussian Naive Bayes, dan Logictic Regression.* Melalui 5 metode yang telah dilakukan kami dapatkan model *Logistic Regression* merupakan model terbaik dengan nilai akurasi tertinggi yaitu 0.82, karena *logistic regression* mengklasifikasi catatan-catatan yang tidak diketahui, *logistic regression* juga dapat bekerja dengan baik ketika kumpulan data dapat dipisahkan secara linear, sehingga dapat menafsirkan koefisien model sebagai indikator pentingnya suatu fitur serta *logistic regression* diterapkan untuk memprediksi variabel dependen yang bersifat kategorikal, misalnya ya atau tidak, 0 atau 1. Dalam melakukan pengamatan ini, kami juga dapat menarik beberapa wawasan dari data melalui analisis dan visualisasi data.