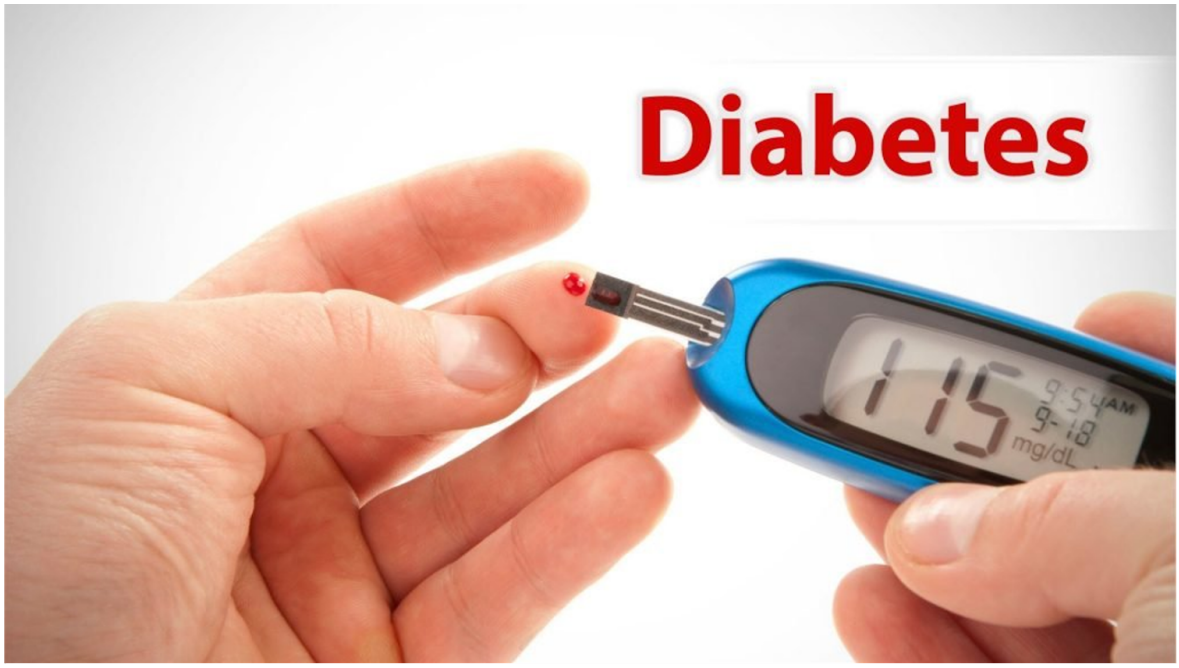


Detecting Diabetes Early

Daniel Kim

Problem Statement

Like any other disease, It is very important to detect the early symptoms of diabetes in order to be cured completely. Reportedly, there are two types of diabetes: type 1 and type 2.
The goal is to classify types of diabetes and their early symptoms so that people can cure diabetes, or prevent diabetes.

The Data

This data contains 17 attributes which include age, gender, and other symptoms.

1. **Target, y = 'Positive' in the column, 'class'**
2. **Other variables, Xs = all other features than y**
3. **Total number of rows = 521**
4. **Total number of columns = 17**

Data Wrangling

1. **Data Cleansing**

```
In [3]:
1 df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 520 entries, 0 to 519
Data columns (total 17 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Age                 520 non-null    int64
 1   Gender              520 non-null    object
 2   Polyuria            520 non-null    object
 3   Polydipsia          520 non-null    object
 4   sudden weight loss  520 non-null    object
 5   weakness            520 non-null    object
 6   Polyphagia          520 non-null    object
 7   Genital thrush      520 non-null    object
 8   visual blurring     520 non-null    object
 9   Itching             520 non-null    object
 10  Irritability        520 non-null    object
 11  delayed healing     520 non-null    object
 12  partial paresis     520 non-null    object
 13  muscle stiffness    520 non-null    object
 14  Alopecia            520 non-null    object
 15  Obesity             520 non-null    object
 16  class               520 non-null    object
dtypes: int64(1), object(16)
memory usage: 69.2+ KB
```

   a. **Null Value, NaN**

There are no Null Values.

   b. **Head Samples**

```
In [4]:
1 df.head()
```

	Age	Gender	Polyuria	Polydipsia	sudden weight loss	weakness	Polyphagia	Genital thrush	visual blurring	Itching	Irritability	delayed healing	partial paresis	muscle stiffness	Alopecia	Obesity	class
0	40	Male	No	Yes	No	Yes	No	No	No	Yes	No	Yes	No	Yes	Yes	Yes	Positive
1	58	Male	No	No	No	Yes	No	No	Yes	No	No	No	Yes	No	Yes	No	Positive
2	41	Male	Yes	No	No	Yes	Yes	No	No	Yes	No	Yes	No	Yes	Yes	No	Positive
3	45	Male	No	No	Yes	Yes	Yes	Yes	No	Yes	No	Yes	No	No	No	No	Positive
4	60	Male	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Positive

As seen above, each column shows binary responses like ‘Male’ or ‘Female’, or ‘Yes’ or ‘No’, or ‘Positive’ or ‘Negative’ except for age. The dataset seems so organized that it can be moved to the next step, one-hot encoding.

c. One-Hot Encoding

```
df.head()
```

	Age	Gender_Female	Gender_Male	Polyuria_No	Polyuria_Yes	Polydipsia_No	Polydipsia_Yes	sudden weight loss_No	sudden weight loss_Yes	weakness_No	weakness_Yes	Polyphagia_No	Polyphagia_Yes	Genital thrush_No	Genital thrush_Yes	visu blurring_
0	40	0	1	1	0	0	1	1	0	0	1	1	0	1	0	1
1	58	0	1	1	0	1	0	1	0	0	1	1	0	1	0	0
2	41	0	1	0	1	1	0	1	0	0	1	0	1	1	0	1
3	45	0	1	1	0	1	0	0	1	0	1	0	1	0	1	1
4	60	0	1	0	1	0	1	0	1	0	1	0	1	1	0	0

Such features from Gender to Class are binary (Male or Female, Yes or No, or Positive or Negative). Thus let's remove such all columns ending with No or Negative.

d. Removing the other binary columns

```
for col in df.columns:
    if '_No' in col:
        del df[col]
df
```

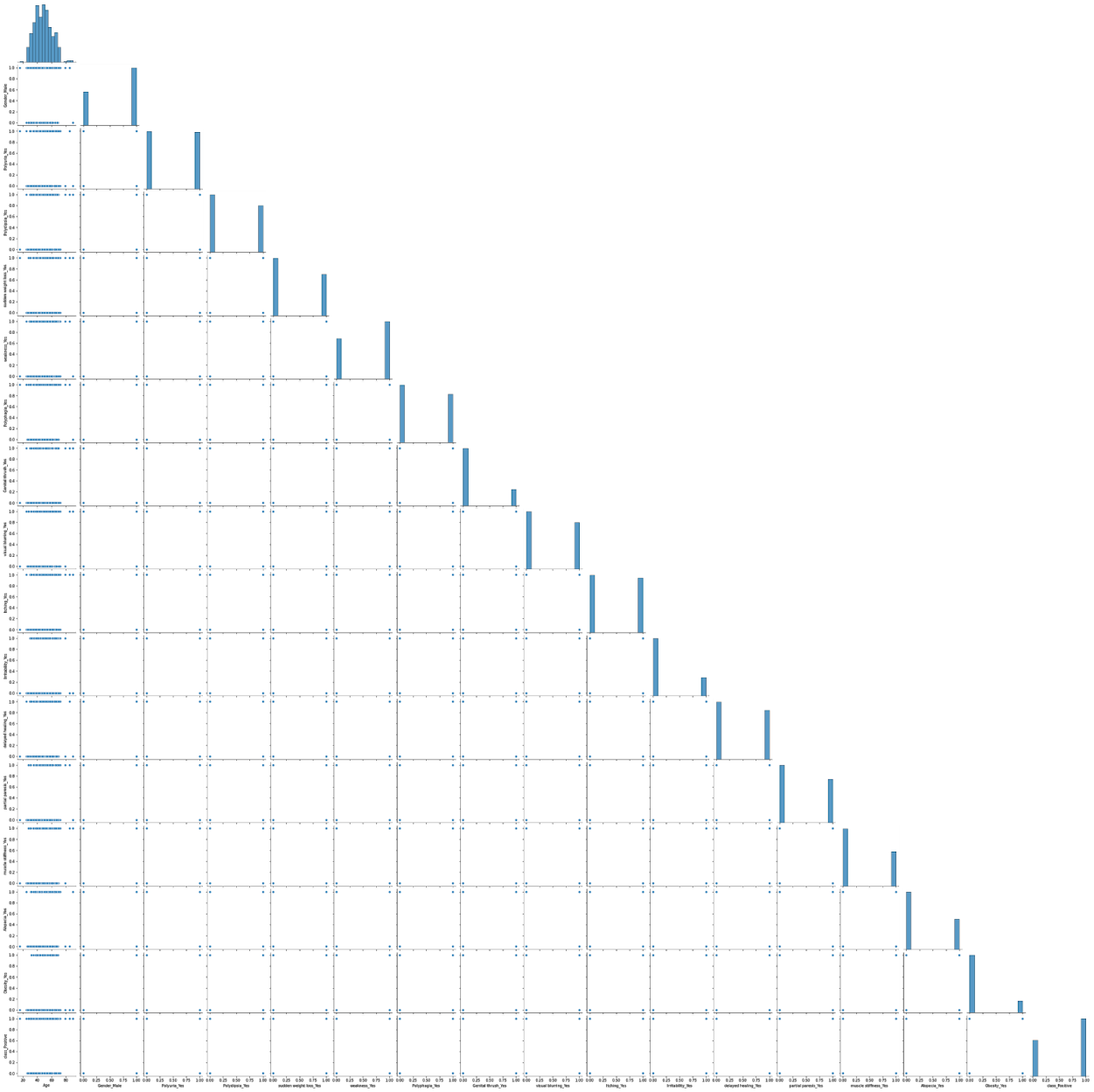
	Age	Gender_Female	Gender_Male	Polyuria_Yes	Polydipsia_Yes	sudden weight loss_Yes	weakness_Yes	Polyphagia_Yes	Genital thrush_Yes	visual blurring_Yes	Itching_Yes	Irritability_Yes	delayed healing_Yes	partial paresis_Yes	muscle stiffness_Yes	Alo
0	40	0	1	0	1	0	1	0	0	0	1	0	1	0	1	1
1	58	0	1	0	0	0	1	0	0	1	0	0	0	1	0	1
2	41	0	1	1	0	0	1	1	0	0	1	0	1	0	1	1
3	45	0	1	0	0	1	1	1	1	0	1	0	1	0	0	0
4	60	0	1	1	1	1	1	1	0	1	1	1	1	1	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
515	39	1	0	1	1	1	0	1	0	0	1	0	1	1	0	0
516	48	1	0	1	1	1	1	1	0	0	1	1	1	1	0	0
517	58	1	0	1	1	1	1	1	0	1	0	0	0	1	1	0
518	32	1	0	0	0	0	1	0	0	1	1	0	1	0	0	1
519	42	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

520 rows × 19 columns

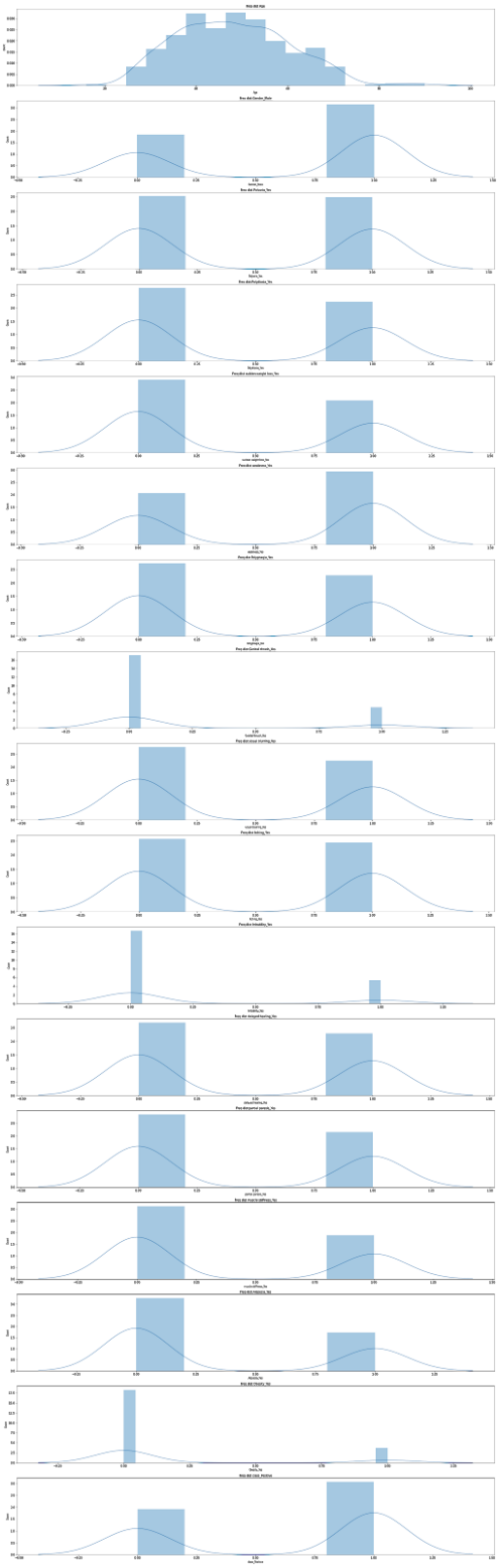
By using for loops as seen above, the other columns ending with '_Female', '_Negative', or '_No' were deleted not only for reducing data size but also for efficiency in running algorithms.

Exploratory Data Analysis

1. Pairplot

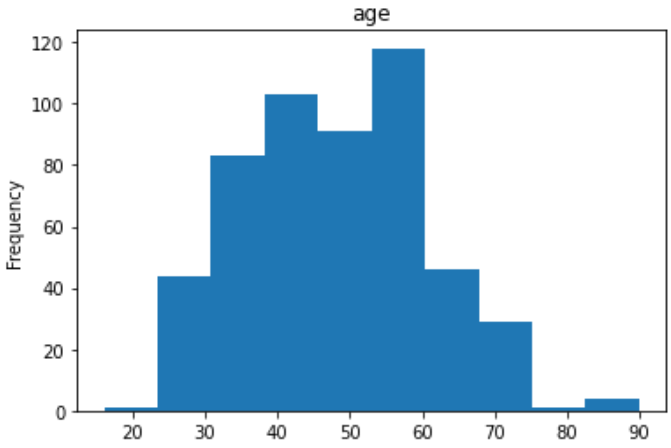


2. Histogram

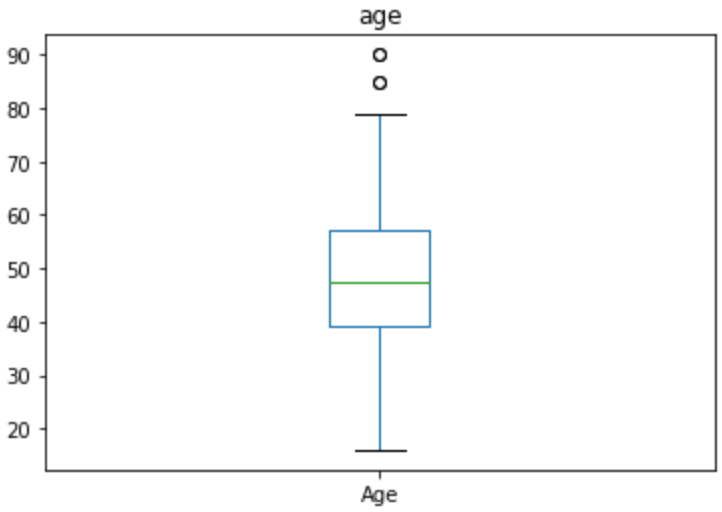


Except for the first feature, ‘age’, all the others do not show normal distribution because they are all binary-typed data.

‘Age’ seems normally distributed.



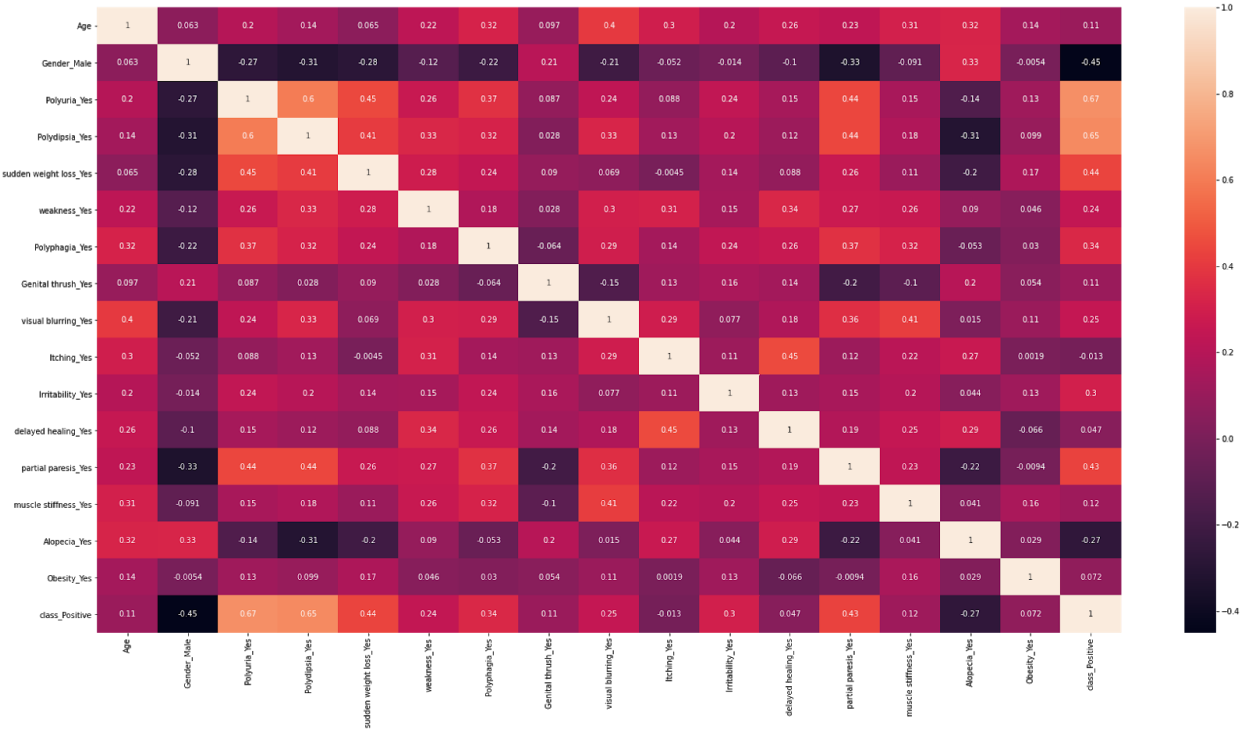
3. Boxplot for Age



The boxplot of ‘Age’ shows that there are two outliers.

Surely, the outliers can be detected as outliers when the outlier detection algorithm is applied. So the next step will be the outlier detection by Isolation Forest.

4. Heatmap



As seen above, too many features seem correlated to each other, so it is needed to simplify their correlation by feature importance of several machine learning algorithms.

Removing Outlier by Machine Learning Algorithm: Isolation Forest

Just for confirming the outlier detection, the Isolation Forest algorithm was applied.

```
%%time
from sklearn.ensemble import IsolationForest
clf=IsolationForest(n_estimators=50, max_samples=50, contamination=float(0.004),
                    max_features=1.0, bootstrap=False, n_jobs=-1,
                    random_state=42, verbose=0,behaviour="new")
clf.fit(df)
pred = clf.predict(df)
df['anomaly']=pred
outliers=df.loc[df['anomaly']==-1]
outlier_index=list(outliers.index)
print(df['anomaly'].value_counts())
```

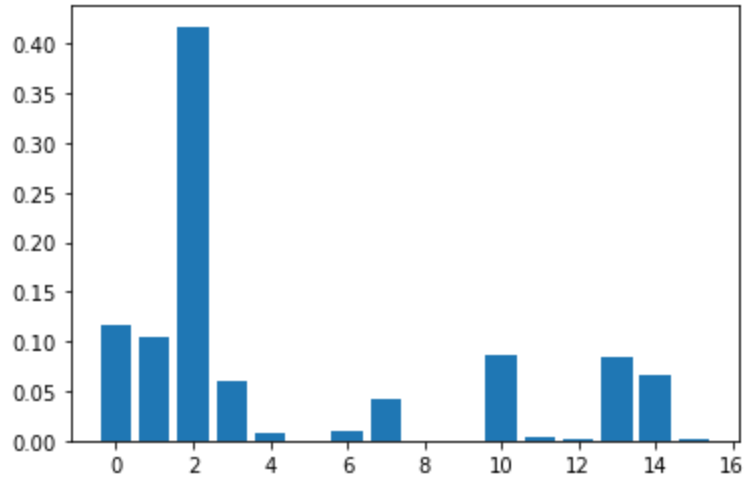
```
 1    518
-1      2
Name: anomaly, dtype: int64
Wall time: 885 ms

C:\Users\Danie\Anaconda\lib\site-packages\sklearn\ensemble\_iforest.py:255: FutureWarning: 'behaviour' is
deprecated in 0.22 and will be removed in 0.24. You should not pass or set this parameter.
  FutureWarning
```

As seen above, only two outliers (“-1”) were detected.

Comparison of Feature Importance:

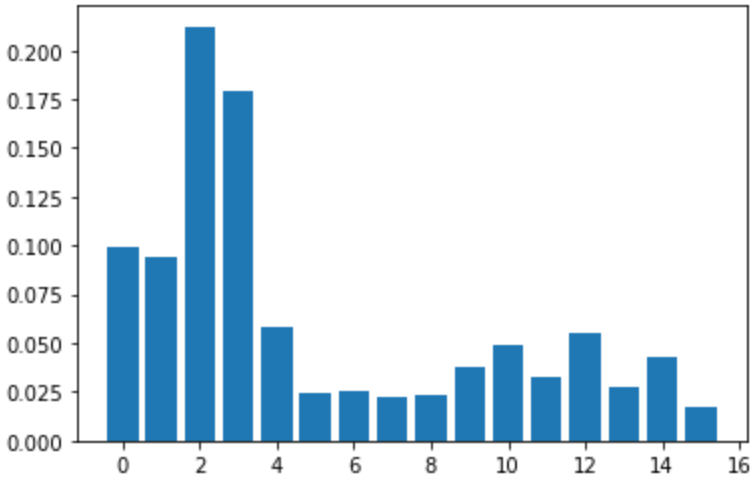
Decision Tree Classifier vs. Random Forest Classifier vs. Gradient Boost Classifier

The below compares each algorithms' feature importances whose importance has more than 5% only.



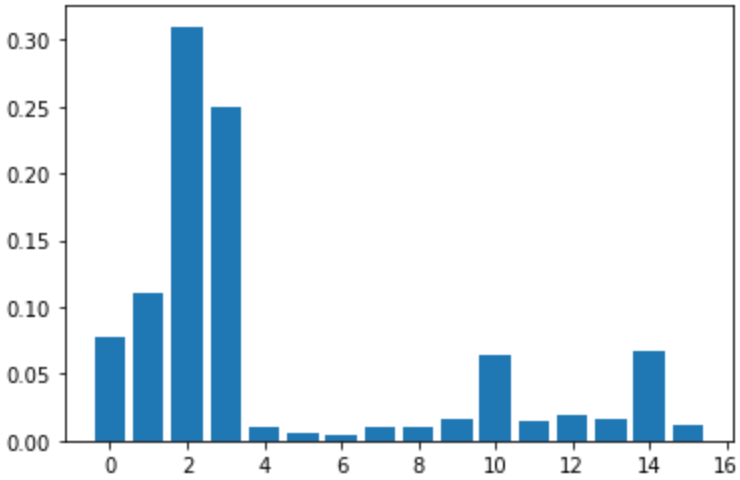
a. Decision Tree Classifier (x>=5% only)

Feature: Age, Score: 12%
Feature: Gender_Male, Score: 10%
Feature: Polyuria, Score: 42%
Feature: Polydipsia, Score: 6%
Feature: Irritability, Score: 9%
Feature: muscle stiffness, Score: 8%
Feature: Alopecia, Score: 7%



b. Random Forest Classifier (x>=5% only)

Feature: Age, Score: 10%
Feature: Gender_Male, Score: 9%
Feature: Polyuria, Score: 21%
Feature: Polydipsia, Score: 18%
Feature: sudden weight loss, Score: 6%
Feature: partial paresis, Score: 5%



c. Gradient Boost Classifier (x>=5% only)

Feature: Age, Score: 8%
Feature: Gender_Male, Score: 11%
Feature: Polyuria, Score: 31%
Feature: Polydipsia, Score: 25%
Feature: Irritability, Score: 6%
Feature: Alopecia, Score: 7%

By comparing each feature importance from three different machine learning algorithms, ‘Polyuria’ showed the strongest feature when it comes to identifying ‘class_positive’’. So we can conclude that ‘Polyuria’ plays a key role in detect diabetes.

Besides ‘Polyuria’, there are the following common strong features to detect diabetes even if their importance rates vary: Age, Gender_Male, and Polydipsia.

Last but not least, except for the above-mentioned features, all the other features with over 5% of importance rates should be watched for detecting diabetes: Irritability, muscle stiffness, sudden weight loss, partial paresis, and Alopecia.

In a nutshell, combinations of the above-mentioned features can give you optimal diabetes procedures. For example, a test of ‘Polyuria’ should be an independent variable for detecting diabetes. Such following features should be dependant variables with higher weights than others: ‘Age’, ‘Gender_Male’, and ‘Polydipsia’. And then, the other else features should be optional test variables with lower weights.

Comparison of Accuracy

Algorithms	**Decision Tree Classifier**	Random Forest Classifier	Gradient Boost Classifier
Accuracy (R-Squared)	**97%**	93%	95%
Randomized Search CV Hyper Parameter Run Time	**0 seconds**	0 seconds	0 seconds
Best Parameters by Randomized Search CV	**'splitter': 'random', 'random_state': 42, 'min_samples_split' : 2, 'min_samples_leaf': 1, 'max_features': 'auto', 'max_depth': 30, 'criterion': 'entropy', 'class_weight': 'balanced'**	'n_estimators': 1200, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': None, 'max_depth': 5, 'criterion': 'gini', 'class_weight': 'balanced'	'n_estimators': 500, 'min_samples_split': 15, 'min_samples_leaf': 1, 'max_features': 'sqrt', 'max_depth': 5, 'loss': 'exponential', 'criterion': 'mae'

Given the above test results, the Decision Tree Classifier showed the best score (97%). Other algorithms showed quite remarkable accuracy scores, but their scores are still lower than the Decision Tree Classifier. The reason for the difference seems due to ‘max_depth’ per each algorithm; the Decision Tree Classifier had more depths than others: 30 vs. 5.

Recommendation & Solution

In order to identify diabetes, you may conduct either of the following testing methods on a trade-off basis for both efficiency and effectiveness.

On one hand, combinations of the above-mentioned features can give you optimal diabetes procedures. For example, a test of ‘Polyuria’ should be an independent variable for detecting diabetes. Such following features should be dependant variables with higher weights than others: ‘Age’, ‘Gender_Male’, and ‘Polydipsia’. And then, the other else features should be optional test variables with lower weights.

On the other hand, you can just conduct a diabetes test by the Decision Tree Classifier method.