Random-Forest-prediction-by-retzam-ai

May 9, 2024

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import RandomOverSampler
```

[2]:
```
df = pd.read_csv('diabetes_prediction_dataset.csv', header=0)
df.head()
```

[2]:
```
   gender   age  hypertension  heart_disease smoking_history    bmi  \
0  Female  80.0             0              1           never  25.19
1  Female  54.0             0              0         No Info  27.32
2    Male  28.0             0              0           never  27.32
3  Female  36.0             0              0         current  23.45
4    Male  76.0             1              1         current  20.14

   HbA1c_level  blood_glucose_level  diabetes
0          6.6                  140         0
1          6.6                   80         0
2          5.7                  158         0
3          5.0                  155         0
4          4.8                  155         0
```

[3]:
```
# Convert each column with nominal data to numbers from 0, 1, 2...
df["gender"], _ = pd.factorize(df["gender"])
df["smoking_history"], _ = pd.factorize(df["smoking_history"])

df.head()
```

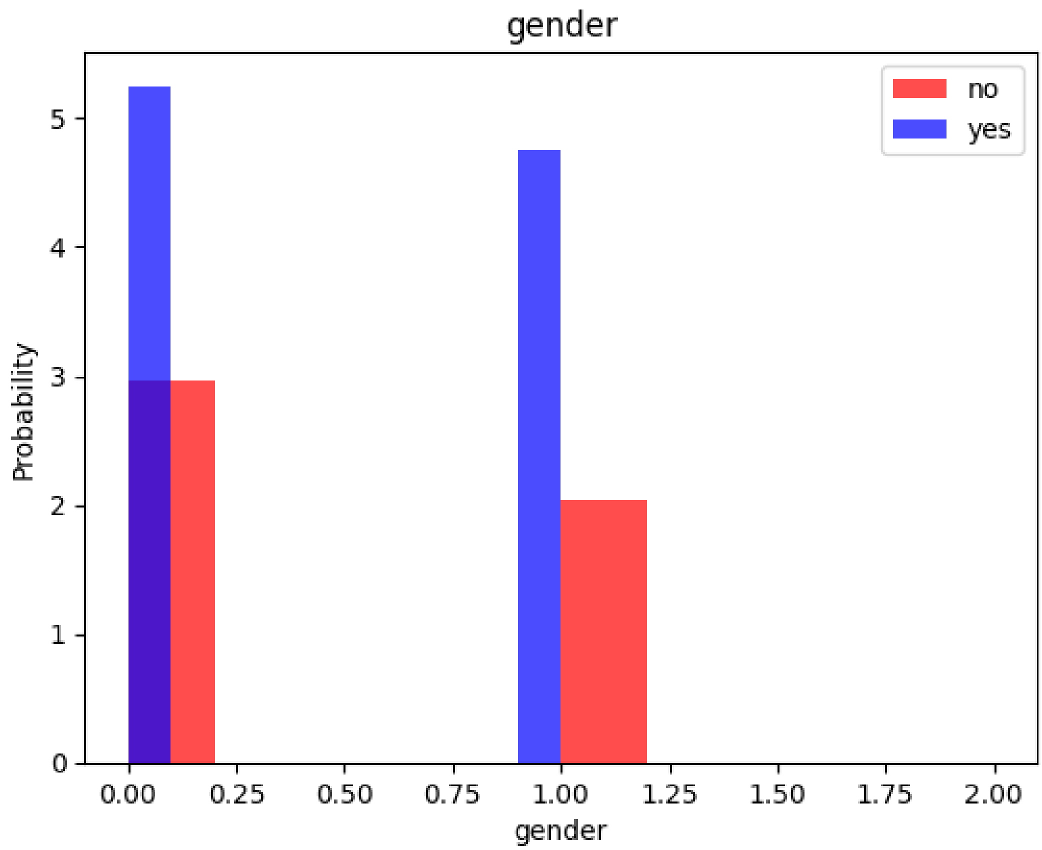
[3]:
```
   gender   age  hypertension  heart_disease  smoking_history    bmi  \
0       0  80.0             0              1                0  25.19
1       0  54.0             0              0                1  27.32
2       1  28.0             0              0                0  27.32
3       0  36.0             0              0                2  23.45
4       1  76.0             1              1                2  20.14

   HbA1c_level  blood_glucose_level  diabetes
0          6.6                  140         0
```

```
1          6.6                    80         0
2          5.7                   158         0
3          5.0                   155         0
4          4.8                   155         0
```

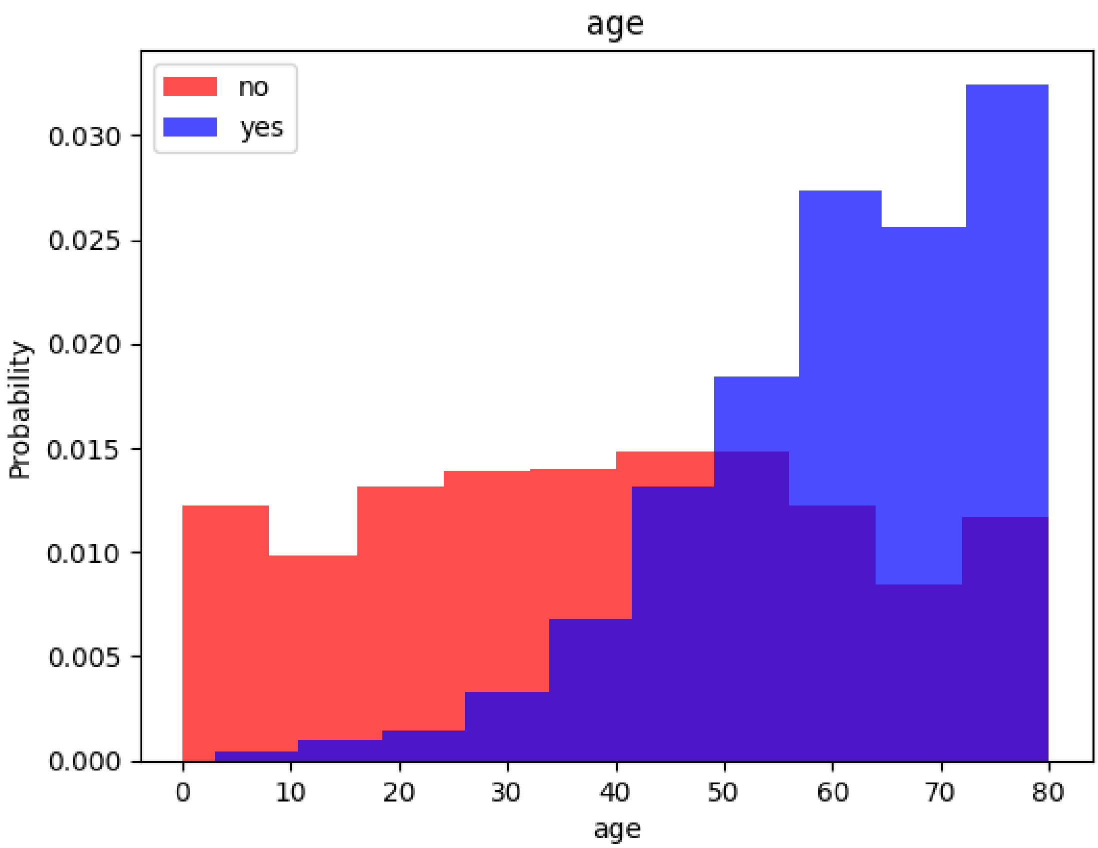
[4]:
```
header = df.columns
header
```

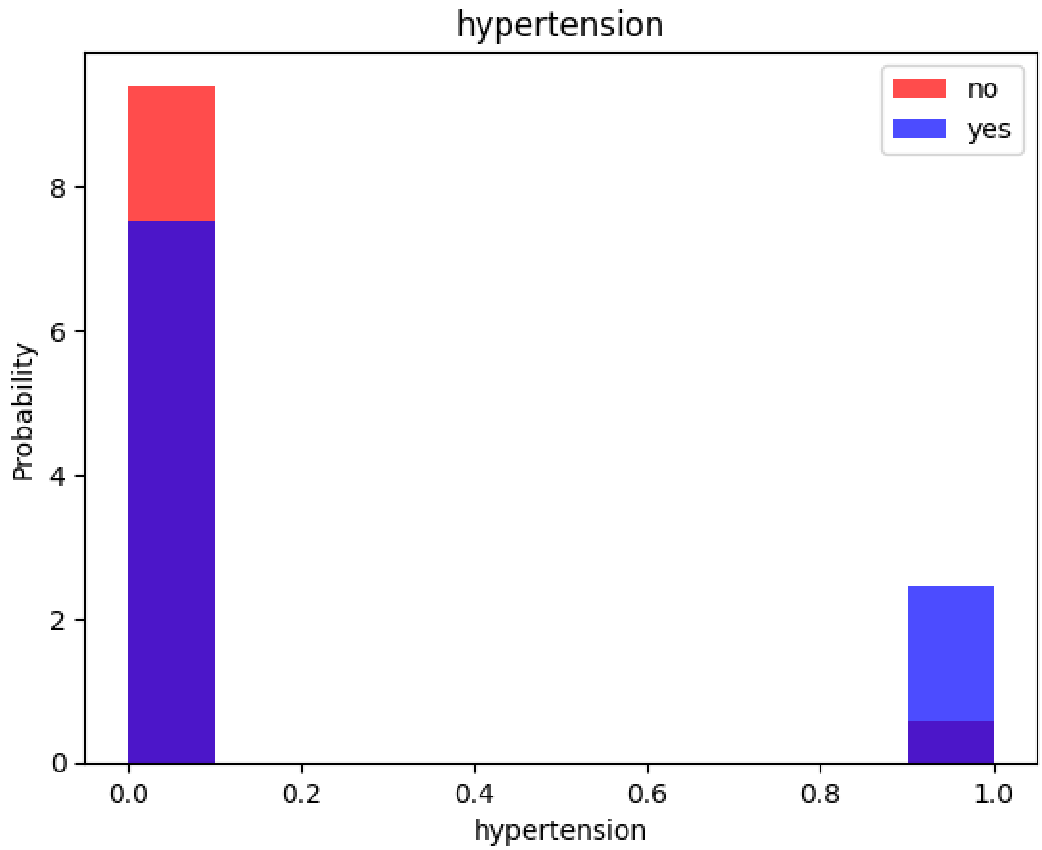
[4]:
```
Index(['gender', 'age', 'hypertension', 'heart_disease', 'smoking_history',
       'bmi', 'HbA1c_level', 'blood_glucose_level', 'diabetes'],
      dtype='object')
```

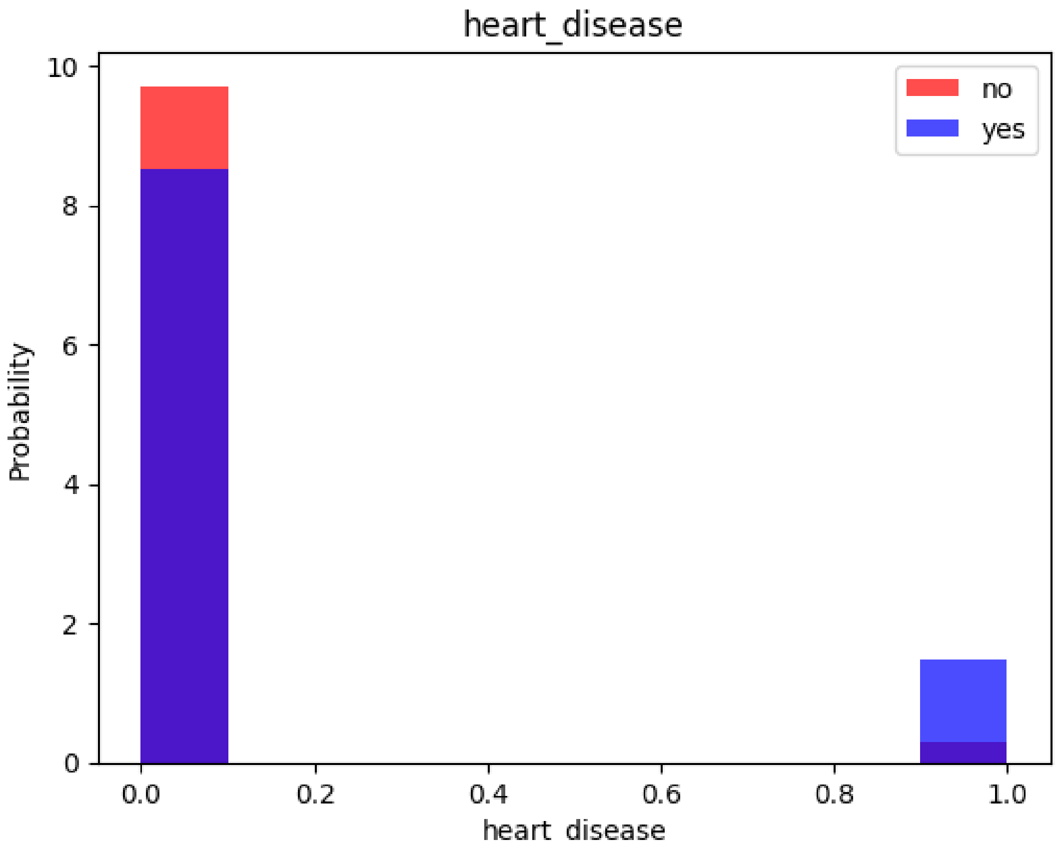
[7]:
```
# We plot a histogram to check which features affect the outcome the most or␣
 ↪the least
# This helps us determine, which features to use in training our model and the␣
 ↪ones to discard

for label in header[:-1]:
  plt.hist(df[df['diabetes'] == 0][label], color = 'red', label='no', alpha=0.
 ↪7, density=True)
  plt.hist(df[df['diabetes'] == 1][label], color = 'blue', label='yes', alpha=0.
 ↪7, density=True)
  plt.title(label)
  plt.ylabel('Probability')
  plt.xlabel(label)
  plt.legend()
  plt.show()
```

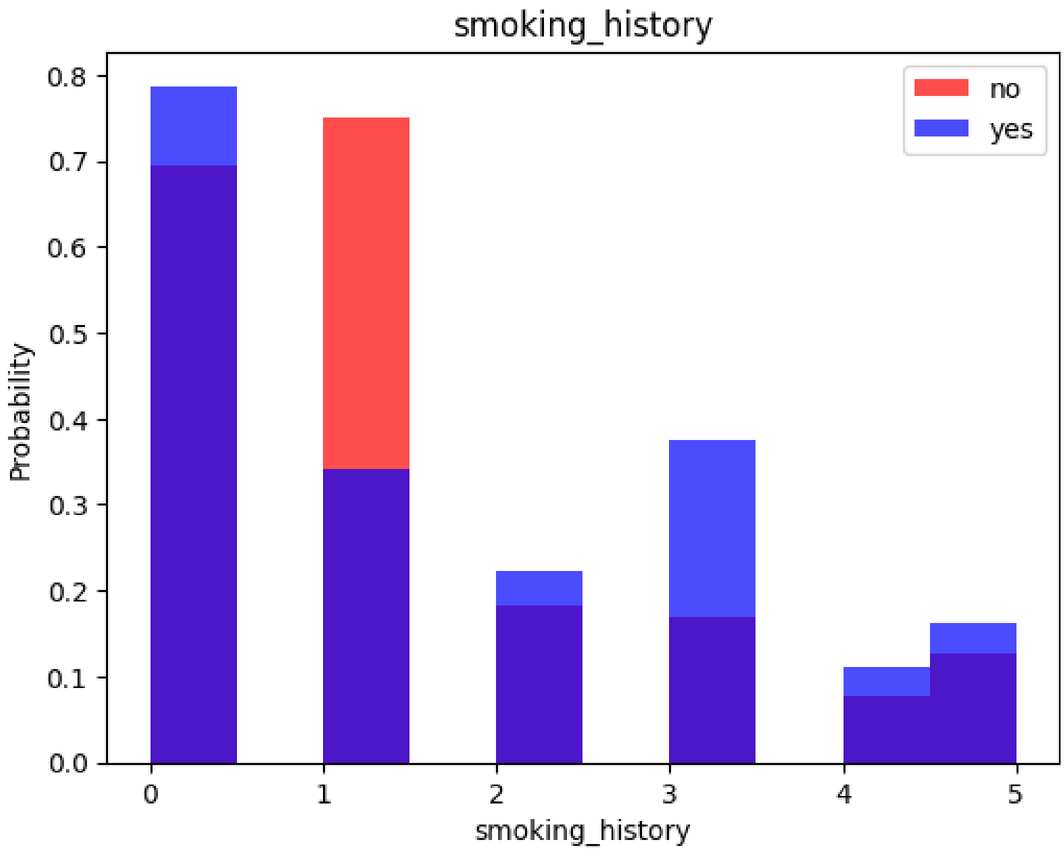


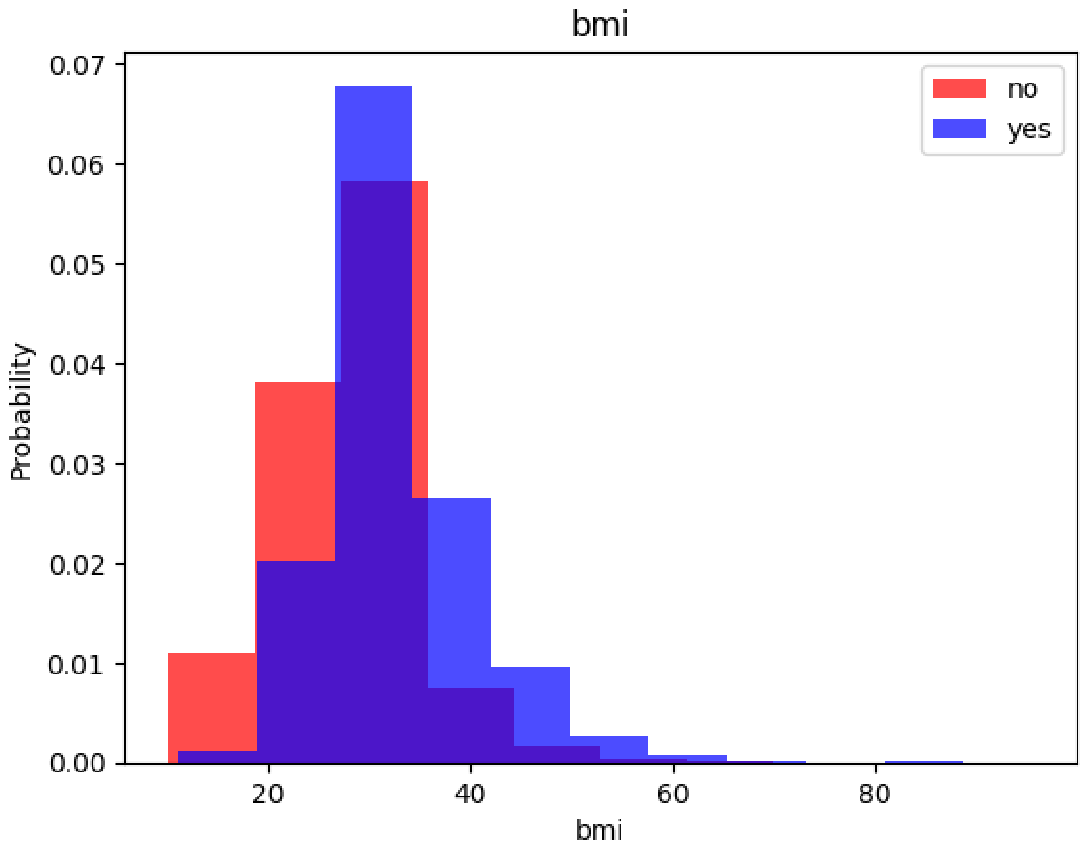
gender
no
yes
Probability
0
1
2
3
4
5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
gender

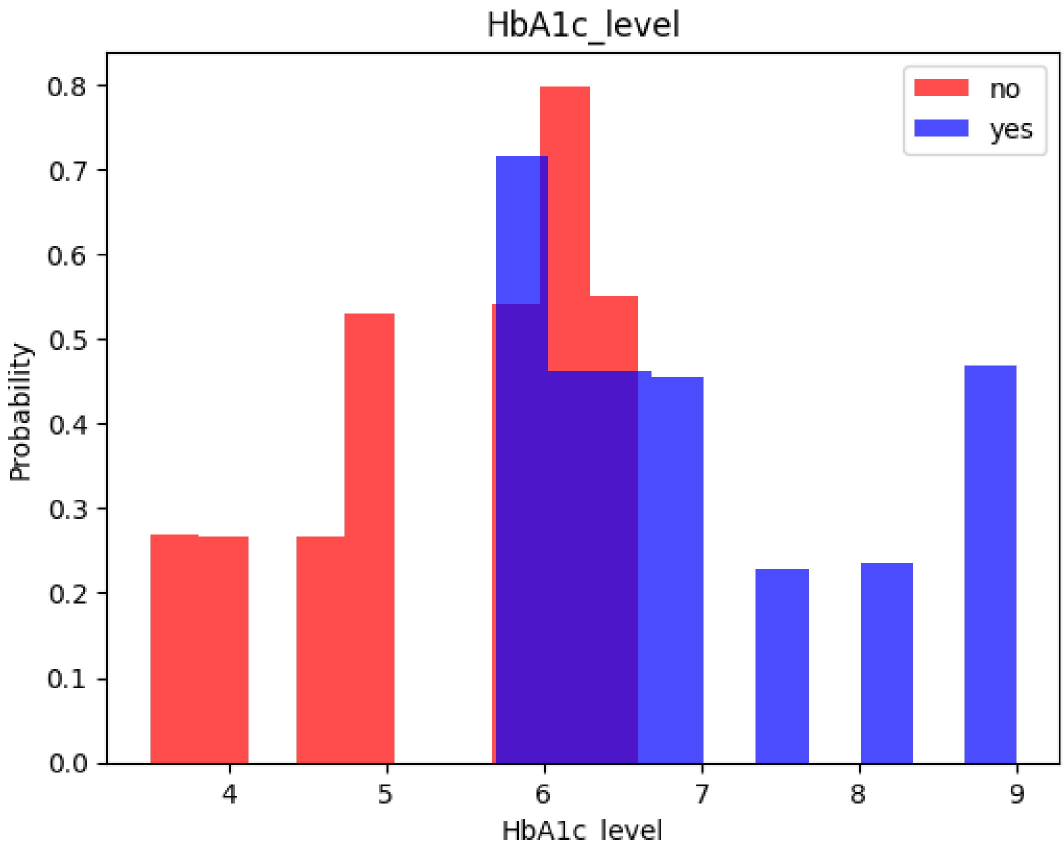

age
no
yes
0.030
0.025
0.020
0.015
0.010
0.005
0.000
Probability
0
10
20
30
40
50
60
70
80
age

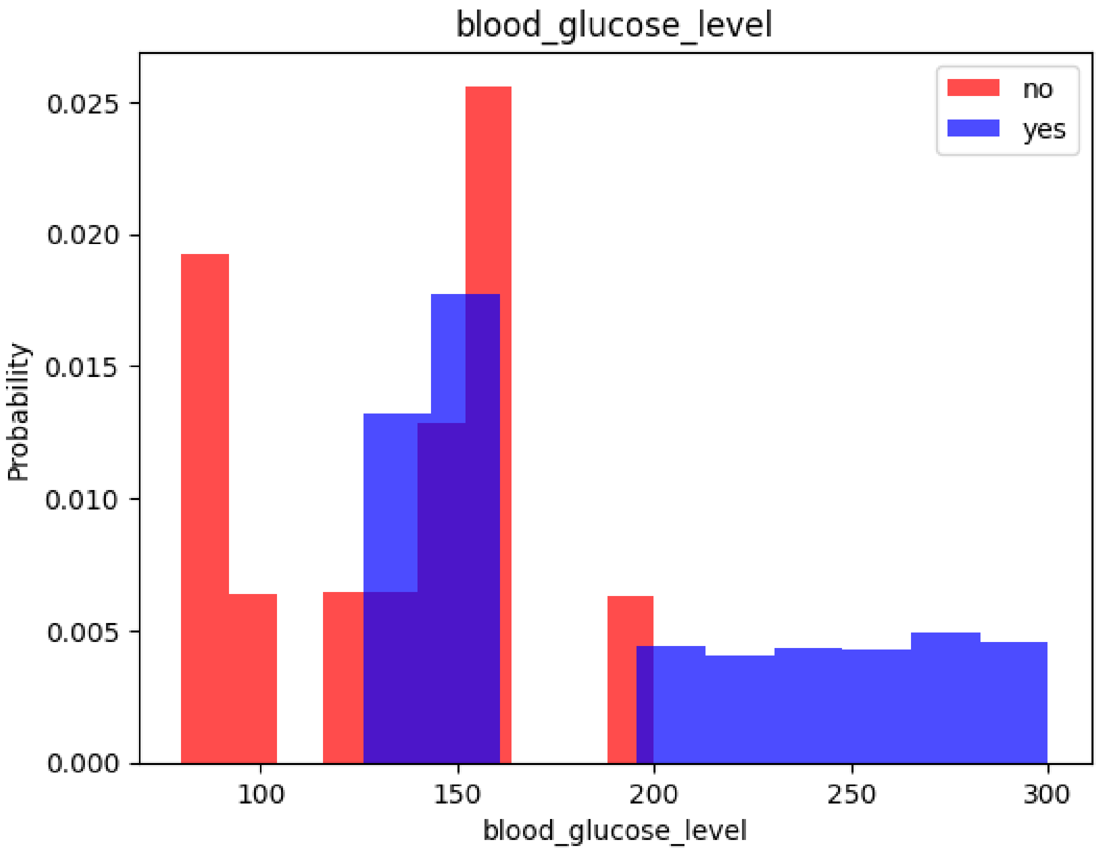

hypertension
no
yes
Probability
0
2
4
6
8
0.0
0.2
0.4
0.6
0.8
1.0
hypertension


heart_disease
no
yes
Probability
heart_disease
0
2
4
6
8
10
0.0
0.2
0.4
0.6
0.8
1.0


smoking_history
no
yes
Probability
smoking_history
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
1
2
3
4
5


bmi
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
Probability
no
yes
20
40
60
80
bmi


HbA1c_level
Probability
HbA1c_level
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
4
5
6
7
8
9
no
yes



[8]:
```
train, test = np.split(df.sample(frac=1), [int(0.8 *len(df))])
```

[5]:
```
# Scale dataset so better prediction can be made.
def scale_dataset(dataframe, oversample=False):
  # This selects all columns in the DataFrame except the last one as the␣
↪features.
  X = dataframe[dataframe.columns[:-1]].values

  # This selects the last column in the DataFrame as the target.
  y = dataframe[dataframe.columns[-1]].values

  # This removes the mean and scaling to unit variance
  # Known as standardization. Basically removes outliers.
  scaler = StandardScaler()
  X = scaler.fit_transform(X)

  """
    Make both x and y sets equal sets as appropriate.
```

```
    RandomOverSampler is important in cases where there is alot more features␣
↪vector of a
    specific output.

    Example if you have a dataset with 100 rows with output as "Yes" and 20
    rows with "No".
    You can see that our datasets would be biased towards the output with "Yes".
    To solve this, RandomOverSampler strategically duplicates rows with "No" so␣
↪the dataset ends up
    having 100 rows with "Yes" and 100 with "No" outputs.

    This is called over-sampling.
  """
  if oversample:
    ros = RandomOverSampler()
    X, y = ros.fit_resample(X, y)

  # Stack horizontally
  # Reshape y and concatenate it with X
  # This simply means attaching each feature vector with the appropriate output.
  data = np.hstack((X, np.reshape(y, (-1, 1))))

  return data, X, y
```

[9]:
```
train, X_train, y_train = scale_dataset(train, oversample=True)

# test sets are not oversampled because they
# are used to test new data
test, X_test, y_test = scale_dataset(test, oversample=False)
```

[10]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
```

[11]:
```
rf_model = RandomForestClassifier()
rf_model = rf_model.fit(X_train, y_train)
```

[12]:
```
y_pred = rf_model.predict(X_test)
y_pred
```

[12]:
```
array([0, 1, 0, …, 0, 0, 0])
```

[13]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98     18226
           1       0.88      0.70      0.78      1774
```

```
    accuracy                           0.96     20000
   macro avg       0.93      0.84      0.88     20000
weighted avg       0.96      0.96      0.96     20000
```

[ ]: