Breast cancer prediction with supervised machine learning algorithm

My chosen dataset is the Breast Cancer Wisconsin Diagnostic dataset (https://www.kaggle.com/code/pratikkgandhi/predicting-breast-cancer-with-random-forest-95 accessed 21 Oct 2024) which includes 33 features, that are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. The mean, standard error and "worst" or largest (mean of the three largest values) of these features were computed for each image. It has 569 observations and no missing values.

The objective of this project is to compare different supervised machine learning models in terms of accurately predicting breast cancer. A high accuracy model would be beneficial as a backend platform to a business whose focus is on improving health with new technology.

First, the raw data is imported with the pandas package and inspected for size, weather there are any null values present, for the column types and then it goes through a series of data cleaning and visualisation techniques to ensure that the correct features are used for prediction. To train the ML model, the features that describe the size of the nuclei are chosen alongside the column that stores the values of the actual diagnosis. With the help of the correlation matrix from sklearn and visualisation from seaborn packages, the number of columns are narrowed down again (only highly correlated columns are taken forward).

In [16]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
from sklearn import metrics
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
```

In [17]:
```
data = pd.read_csv('data.csv')
```

In [18]:
```
data.shape
```

Out[18]:
```
(569, 33)
```

In [19]:
```
data.describe()
```

Out[19]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothnes
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0

8 rows × 32 columns

In [20]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

In [21]:
```
data.head(3).T
```

Out[21]:

	0	1	2
id	842302	842517	84300903
diagnosis	M	M	M
radius_mean	17.99	20.57	19.69
texture_mean	10.38	17.77	21.25
perimeter_mean	122.8	132.9	130.0
area_mean	1001.0	1326.0	1203.0
smoothness_mean	0.1184	0.08474	0.1096
compactness_mean	0.2776	0.07864	0.1599
concavity_mean	0.3001	0.0869	0.1974
concave points_mean	0.1471	0.07017	0.1279
symmetry_mean	0.2419	0.1812	0.2069
fractal_dimension_mean	0.07871	0.05667	0.05999
radius_se	1.095	0.5435	0.7456
texture_se	0.9053	0.7339	0.7869
perimeter_se	8.589	3.398	4.585
area_se	153.4	74.08	94.03
smoothness_se	0.006399	0.005225	0.00615
compactness_se	0.04904	0.01308	0.04006
concavity_se	0.05373	0.0186	0.03832
concave points_se	0.01587	0.0134	0.02058
symmetry_se	0.03003	0.01389	0.0225
fractal_dimension_se	0.006193	0.003532	0.004571
radius_worst	25.38	24.99	23.57
texture_worst	17.33	23.41	25.53
perimeter_worst	184.6	158.8	152.5
area_worst	2019.0	1956.0	1709.0
smoothness_worst	0.1622	0.1238	0.1444
compactness_worst	0.6656	0.1866	0.4245
concavity_worst	0.7119	0.2416	0.4504
concave points_worst	0.2654	0.186	0.243
symmetry_worst	0.4601	0.275	0.3613
fractal_dimension_worst	0.1189	0.08902	0.08758
Unnamed: 32	NaN	NaN	NaN

Remove the unnecessary columns from the dataframe

In [22]:
```
data.drop('Unnamed: 32', axis = 1, inplace = True)
data.drop('id', axis = 1, inplace = True)
```

What are the values in the diagnosis column?

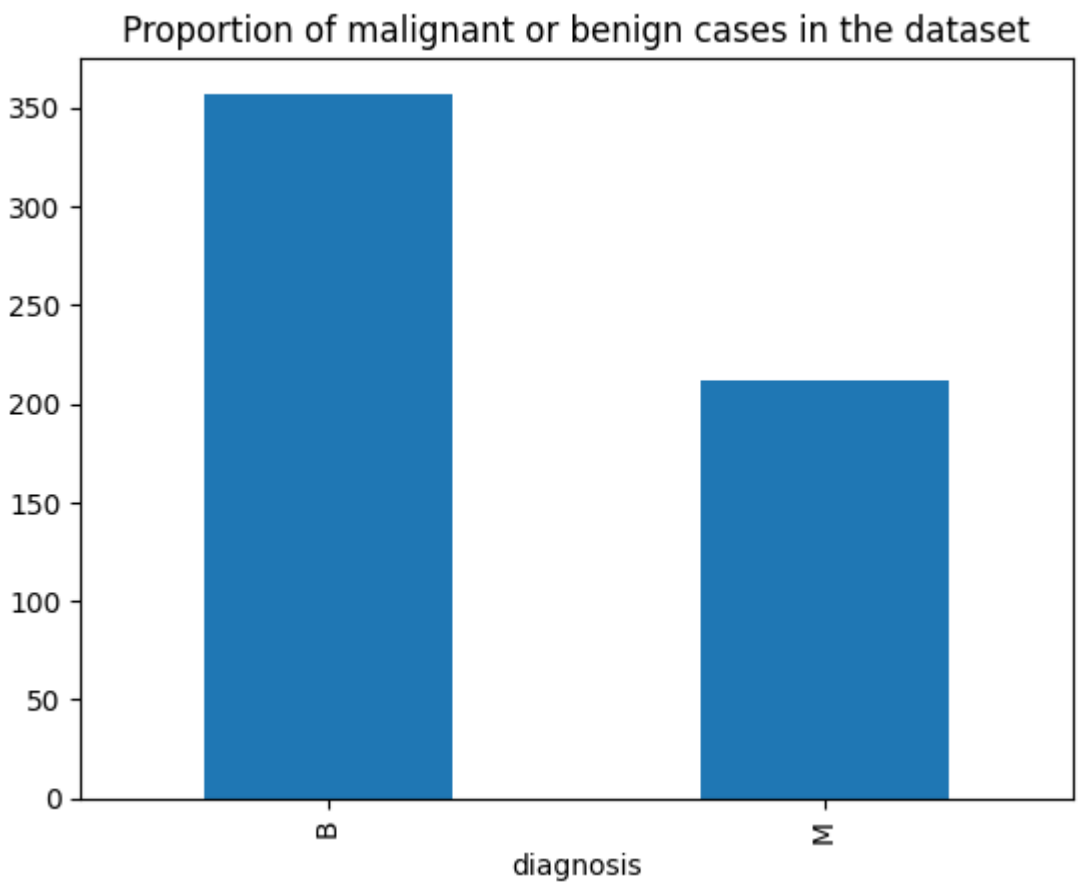
In [23]:
```
data['diagnosis'].unique()
```

Out[23]:
```
array(['M', 'B'], dtype=object)
```

In [24]:
```
ratio = data['diagnosis'].value_counts()
ratio
```

Out[24]:
```
diagnosis
B    357
M    212
Name: count, dtype: int64
```

In [25]:
```
ratio.plot(kind = 'bar')
plt.title('Proportion of malignant or benign cases in the dataset')
plt.show()
```



In [26]:
```
data['diagnosis'] = data['diagnosis'].replace({'B': 0, 'M': 1})
```

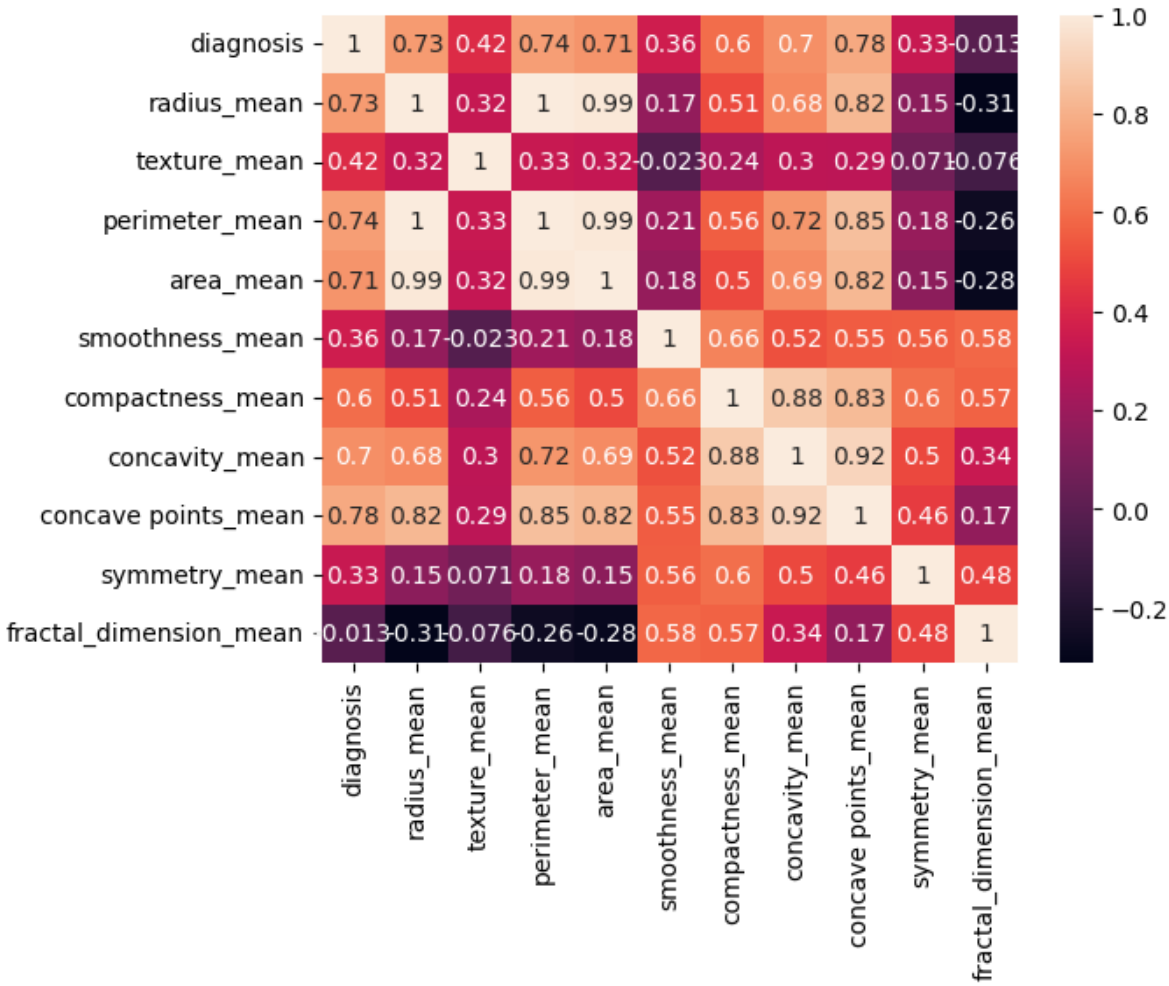
```
/var/folders/3c/5q4qt7wd2zbgbqznjhttqttm0000gn/T/ipykernel_20497/276317377
1.py:1: FutureWarning: Downcasting behavior in `replace` is deprecated and
will be removed in a future version. To retain the old behavior, explicitly
call `result.infer_objects(copy=False)`. To opt-in to the future behavior,
set `pd.set_option('future.no_silent_downcasting', True)`
  data['diagnosis'] = data['diagnosis'].replace({'B': 0, 'M': 1})
```

What is the influence of the feature on the diagnosis?

In [27]:
```
mean_data = data[['diagnosis','radius_mean','texture_mean','perimeter_mean',
```

In [28]:
```
heatmap = sns.heatmap(mean_data.corr(), annot=True)
heatmap
```

Out[28]: <Axes: >



In [29]:
```
features = data[['radius_mean', 'perimeter_mean','area_mean', 'compactness_m
target = data['diagnosis']

X = features
y = target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, r
```

The dataset is split into training and test sets, in 70:30 ratio. Then Logistic Regression, a supervised machine learning model is applied and the accuracy of the predicted and true results are compared. Another model, Random Forest is used, which is an ensemble model. As the two models gave the same accuracy percentages, I used RandomSearchCV to optimise the parameters.

In [30]:
```
lr = LogisticRegression(random_state = 42)

clf = lr.fit(X_train, y_train)
y_pred_logreg = lr.predict(X_test)

cm_logreg = metrics.confusion_matrix(y_test, y_pred_logreg)
```

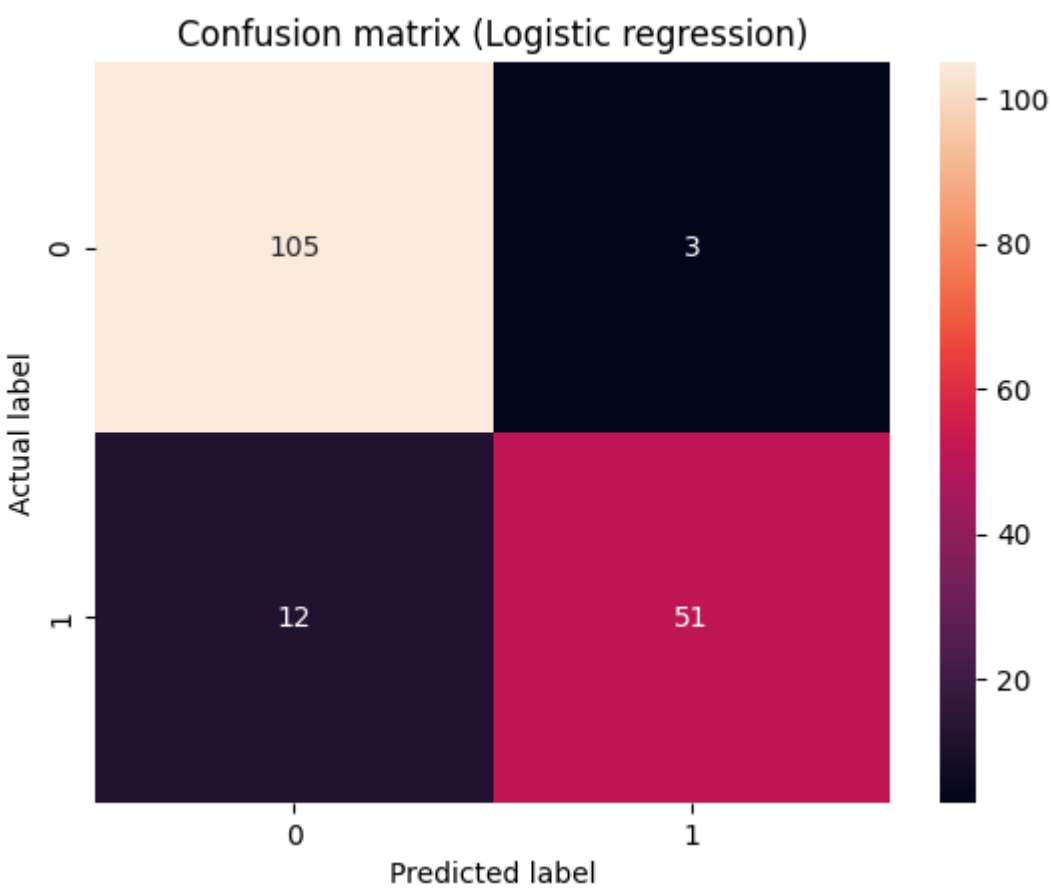
In [31]:
```
cm_logreg
```

Out[31]:
```
array([[105,   3],
       [ 12,  51]])
```

In [32]:
```
cm_logreg_heatmap = sns.heatmap(pd.DataFrame(cm_logreg), annot=True, fmt='g'

plt.title('Confusion matrix (Logistic regression)')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')

plt.show()
```



Training Random Forest Classifier

In [33]:
```
rf = RandomForestClassifier()

rfc = rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)

cm_rf = metrics.confusion_matrix(y_test, y_pred_rf)
```

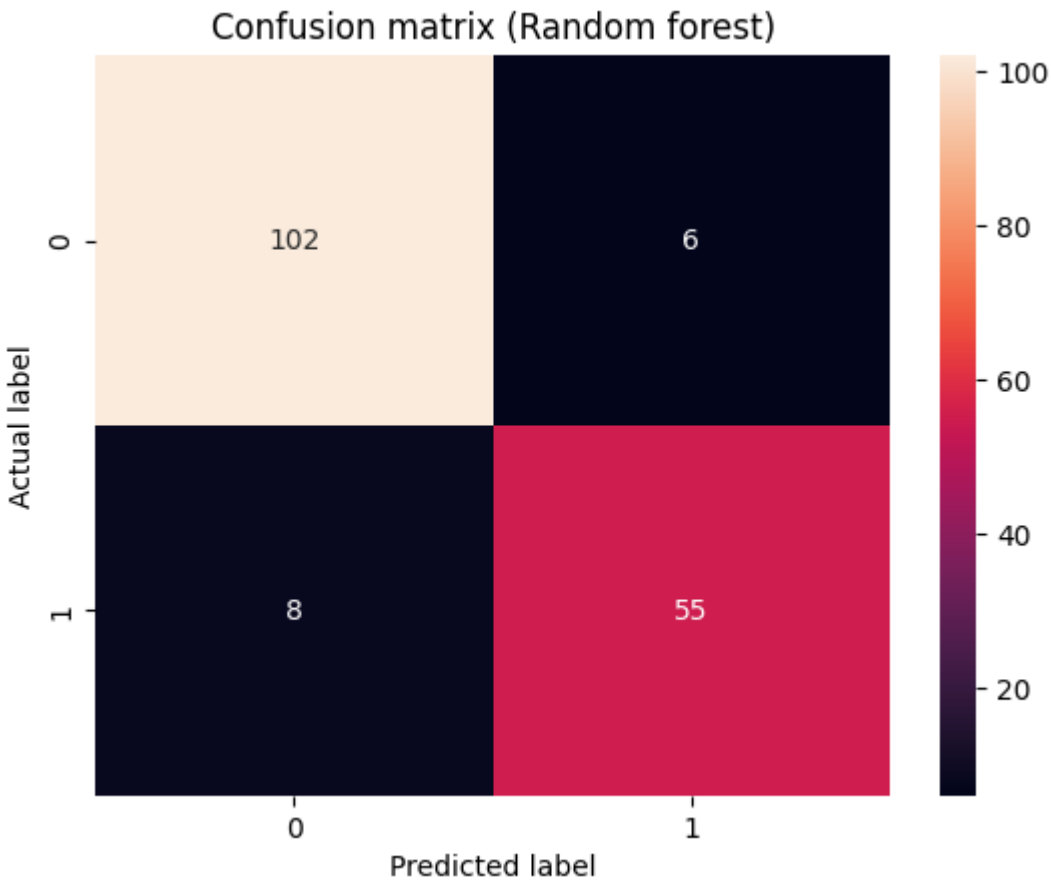
In [34]:
```
cm_rf
```

Out[34]:
```
array([[102,   6],
       [  8,  55]])
```

In [35]:
```
cm_rf_heatmap = sns.heatmap(pd.DataFrame(cm_rf), annot=True, fmt='g')

plt.title('Confusion matrix (Random forest)')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')

plt.show()
```



In [36]:
```
from sklearn.metrics import accuracy_score

acc_logreg = accuracy_score(y_test, y_pred_logreg)
acc_rf = accuracy_score(y_test, y_pred_rf)
```

```
print("Accuracy of Logistic Regression model:", acc_logreg)
print("Accuracy of Random Forest model:", acc_rf)
```

```
Accuracy of Logistic Regression model: 0.9122807017543859
Accuracy of Random Forest model: 0.9181286549707602
```

As the more complex model did not result in better prediction, let's perform hyperparameter tuning on the Random Forest Classifier

In [37]:
```
#Hyperparameter tuning using RandomizedSearchCV

param_dist = {'n_estimators': (10,20,30,40,50), 'max_depth': (1,5,10,15,20)}

rand_search = RandomizedSearchCV(rf, param_distributions = param_dist, n_ite

rand_search.fit(X_train, y_train)
```

Out[37]:
```
▸ RandomizedSearchCV
▸ estimator: RandomForestClassifier
  ▸ RandomForestClassifier
```

In [38]:
```
rand_search.best_params_
```

Out[38]:
```
{'n_estimators': 40, 'max_depth': 5}
```

In [39]:
```
#Optimised parameters for Random Forest

rf_best= RandomForestClassifier(n_estimators=50, max_depth=5)

rfc_best = rf_best.fit(X_train, y_train)
y_pred_rf_best= rf_best.predict(X_test)

cm_rf_best= metrics.confusion_matrix(y_test, y_pred_rf_best)
```

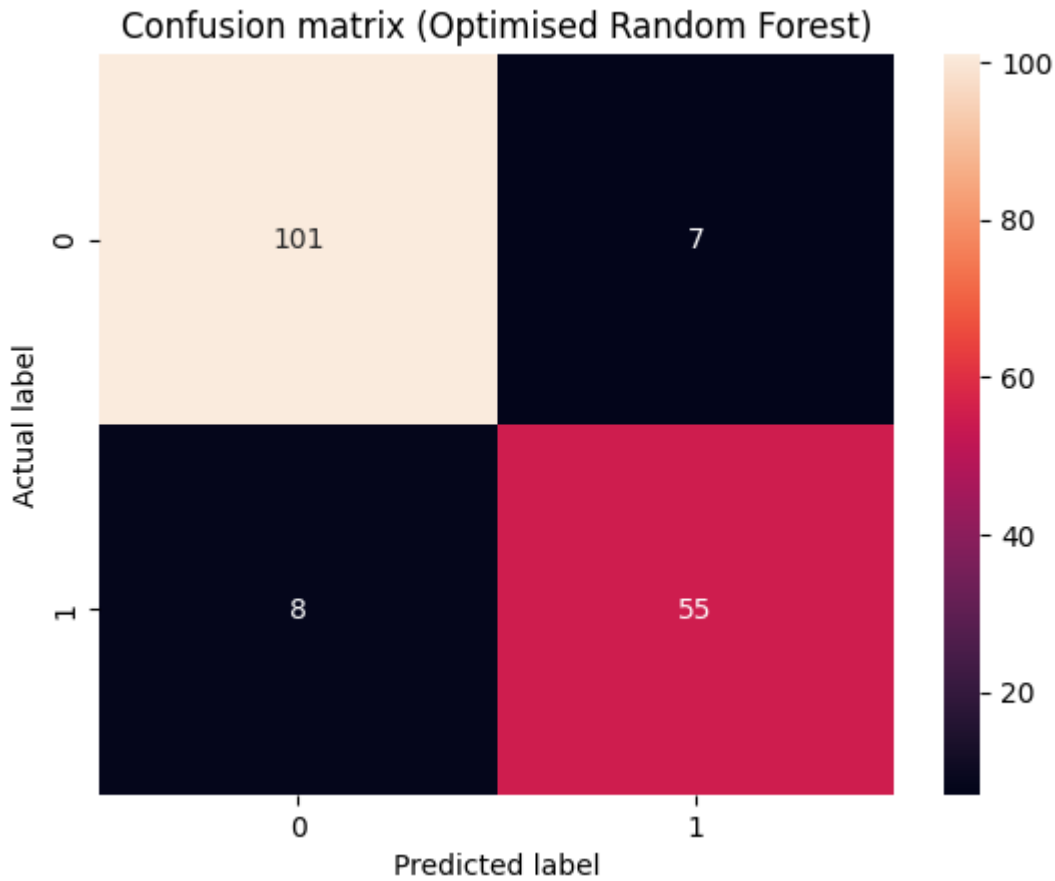
In [40]:
```
cm_rf_best
```

Out[40]:
```
array([[101,   7],
       [  8,  55]])
```

In [41]:
```
cm_rf_best_heatmap = sns.heatmap(pd.DataFrame(cm_rf_best), annot=True, fmt='

plt.title('Confusion matrix (Optimised Random Forest)')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')

plt.show()
```



Final results:

In [42]:
```
acc_rf_best = accuracy_score(y_test, y_pred_rf_best)

print("Accuracy of Logistic Regression model:", acc_logreg)
print("Accuracy of Random Forest model:", acc_rf)
print("Accuracy of Optimised Random Forest model:", acc_rf_best)
```

```
Accuracy of Logistic Regression model: 0.9122807017543859
Accuracy of Random Forest model: 0.9181286549707602
Accuracy of Optimised Random Forest model: 0.9122807017543859
```

Conclusion

All three machine learning models gave a very promising score predicting breast cancer, achieving 91% accuracy. This could be explained by the nature of the dataset, where the features are well separated. Logistic Regression, as a linear model, works well with these linear datasets. Random Forest Classifier, on the other hand, may have overfitted, as it is a non-linear ensemble model. Key insights from this analysis revealed that some features highly influenced whether the tumor was malignant or benign. Applying RandomSearchCV hyperparameter tuning improved the computation time but at the same time did not compromise accuracy. One way to improve this metric would be to evaluate a bigger dataset. Also other feature engineering techniques, such as PCA dimensionality reduction could be beneficial. Furthermore, other machine learning

models such as Gradient Boosting might provide improvements. This project confirms, that with limited data, Logistic Regression model is as accurate as a more complex Random Forest Classifier.