Final project- DA Machine Learning

Introduction

This project aims to assess the ability to use machine learning to predict whether a patient's histology sample is benign or malignant based on the analysis of the nuclei of cells from a fine needle aspiration. The significance of this project lies in the potential to improve aspects of cancer management, such as:

- Early detection and diagnosis as the accuracy and sensitivity of these models can provide reliable results.
- Clinical decision support and reducing subjectivity related to human interpretation, giving an objective approach.
- Research and discovery of potential relationships between diagnosis and cellular features that were not known previously.
- Streamline healthcare resources by reducing workload on healthcare workers.

The significance of developing such models can help with efficiency, accuracy, and speed of diagnosis, ultimately leading to improved patient outcomes and contributing to advancements in healthcare practices. The dataset chosen is publicly available, and contains features computed from digitised images of histology samples suspicious for malignancy.

Data

The data points are based on the nuclei of the cells obtained from fine needle aspirations of 569 patients. The measures are divided into 3 subsections, mean, standard error and worst. Each subsection contains 10 parameters: radius, texture, area, perimeter, compactness, concave points, fractal dimension, smoothness, concavity and symmetry. The target variable is included and categorised as either malignant(M) or benign(B). Having the diagnosis will enable us to train a model to classify samples based on the different features.

The dataset was obtained from kaggle and can be found here: https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data?resource=download

Importing

The data was imported through a flat CSV file using pandas.

```
df = pd.read_csv('./breast_cancer.csv')
```

Preparation

Initial overview of the df was conducted before EDA.

- Dimensions of the df
  - ```
    print("Total data points",df.shape[0]) #investigating the number of rows, i.e. samples
    print("Total number of features(as number of columns) are ", df.shape[1]) #investigating the number of columes i.e. features
    ```
- Contents of the columns
  - ```
    print(df.info())
    ```
- Checked for duplicates.
  - ```
    dup = df.duplicated()

    print(dup.unique()) #
    ```
- Summary stats
  - ```
    print(df.describe(include='all'))
    ```

From this, two unnecessary columns were identified: “ID” and “unnamed: 32”. These were removed from the df.

- ```
  df.drop(columns = ['id','Unnamed: 32'], inplace = True)
  ```

Here we see the different data types in the columns that have no missing values coded as ‘unknown’ string as they are floats. The null count does not report missing values either. If there were missing values, we could either impute new values to retain the sample sizes or remove them completely if there are few values.Using summary stats enables us to compare the ranges/scales of data and the need for standardisation. It was also noted that the minimum value in two columns related to concavity were zero. In this instance it is not likely that this is a missing value as cell contour can have a value of zero thus no removal/imputing is needed. There were no duplicate rows either, so the dataset was sufficient to proceed with EDA.

A custom function was created to speed up the creation of graphs during EDA.
  ```

```python
def make_jointplot(feature_1,feature_2):
  pearson_corr = df[feature_1].corr(df[feature_2])

  corr_var = sns.jointplot(x=feature_1, y=feature_2, data=df, kind='reg')

  corr_var.ax_joint.annotate(f"Pearson's r = {pearson_corr:.2f}", xy=(0.1, 0.9), xycoords='axes fraction', fontsize=12, color='red')
  corr_var.fig.suptitle(feature_1 + ' vs ' + feature_2)
  plt.show()
```

This allows us to quickly create several graphs without needing to duplicate code and the potential to use this function again in the future for similar analysis.

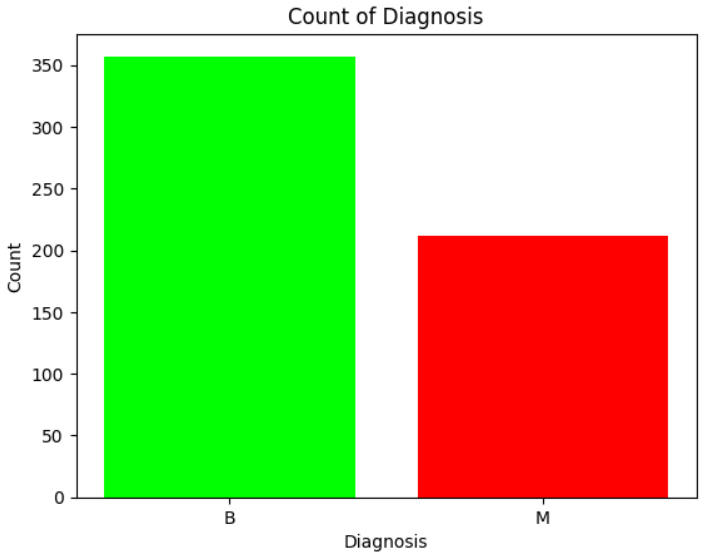
Data Visualisation

Green was used to match B and red M.

Matplotlib graphs:

- **Bar graph:**

```python
colours = ['#00FF00', '#FF0000'] #creating a custom colour pallete to show malignant as red i.e. bad and benign as green i.e. good.

counts = df['diagnosis'].value_counts() #storing the counts of the target variable
plt.bar(counts.index, counts.values, color=colours) #using matplotlib to create a bar graph of the counts

plt.xlabel('Diagnosis')
plt.ylabel('Count')
plt.title('Count of Diagnosis')


print('Benign samples total: ',counts[0])
print('Malignant samples total: ',counts[1], '\n')
```



It clear that there are nearly twice as many benign samples compared to malignant samples, indicating an unbalanced dataset. This is important to consider when choosing models.

1. **Histogram of the data spread of one feature to compare variance and identify outliers.**
   This block filters the df into two separate dfs based on the diagnosis label.

```
features_M=features_2[features_2['diagnosis'] =='M'] #dividing data based on label coding, i.e. df of malignant only
features_B=features_2[features_2['diagnosis'] =='B'] #df of benign only
```

- This block selects the feature of interest from the Malignant dataset and extracts the values.

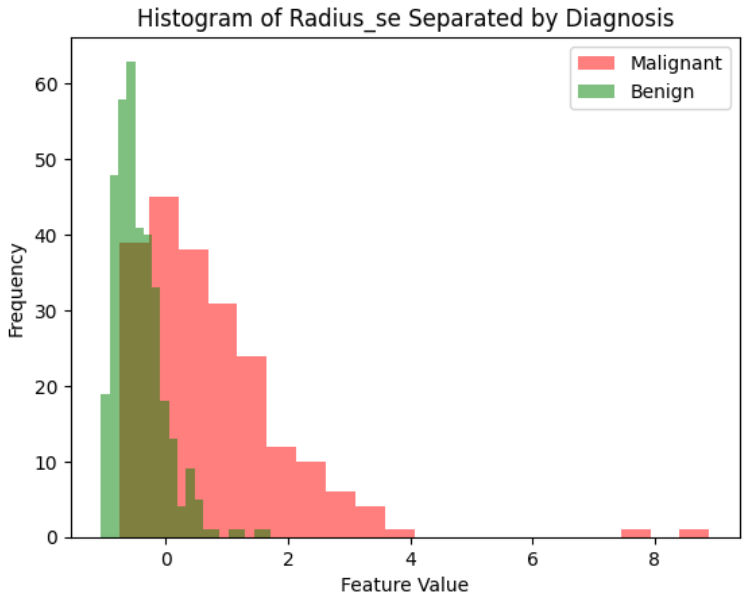
```
selected_rows_M = features_M[features_M['features'] == 'radius_se']
selected_values_M =selected_rows_M['value'] #filtering based on the desired featurea and then extracting only the associated values
```

- This is repeated here for the benign samples

```
selected_rows_B = features_B[features_B['features'] == 'radius_se']
selected_values_B =selected_rows_B['value']
```

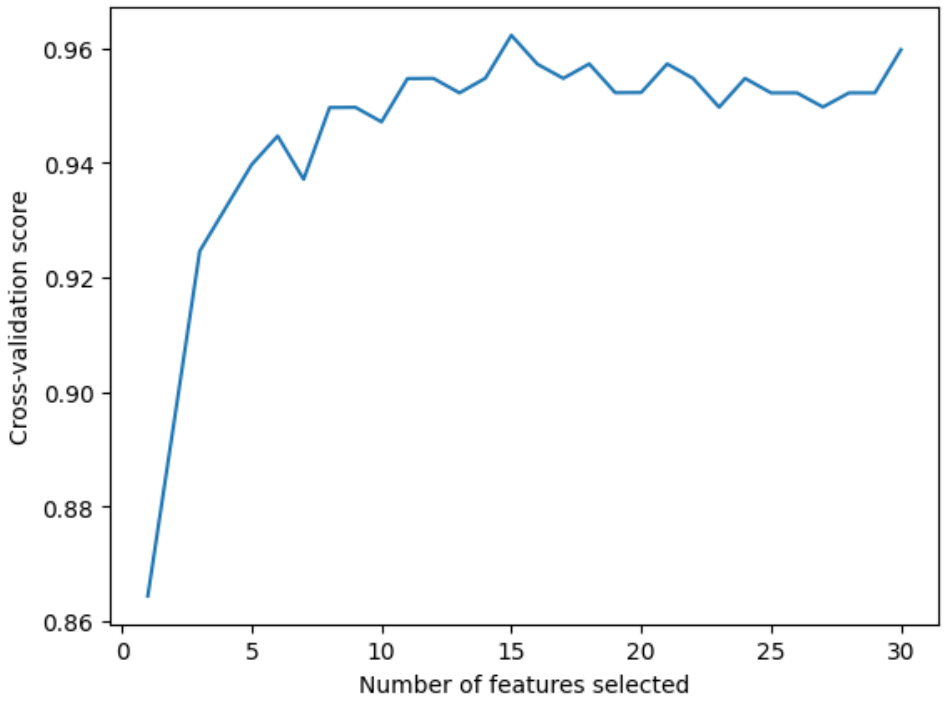
- This plots and superimposes the histograms of the values of the chosen feature.

```
plt.hist(selected_values_M, bins=20, alpha = 0.5, label = 'Malignant', color = 'red')
plt.hist(selected_values_B, bins=20, alpha = 0.5, label = 'Benign', color = 'green')


plt.xlabel('Feature Value')
plt.ylabel('Frequency')
plt.title('Histogram of Radius_se Separated by Diagnosis')
plt.legend()
```



2. **Line graph, to visually assess the optimal number of features chosen.**

```
plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross-validation score")
plt.plot(range(1, len(rfecv.cv_results_['mean_test_score']) + 1), rfecv.cv_results_['mean_test_score'])
plt.show()
```



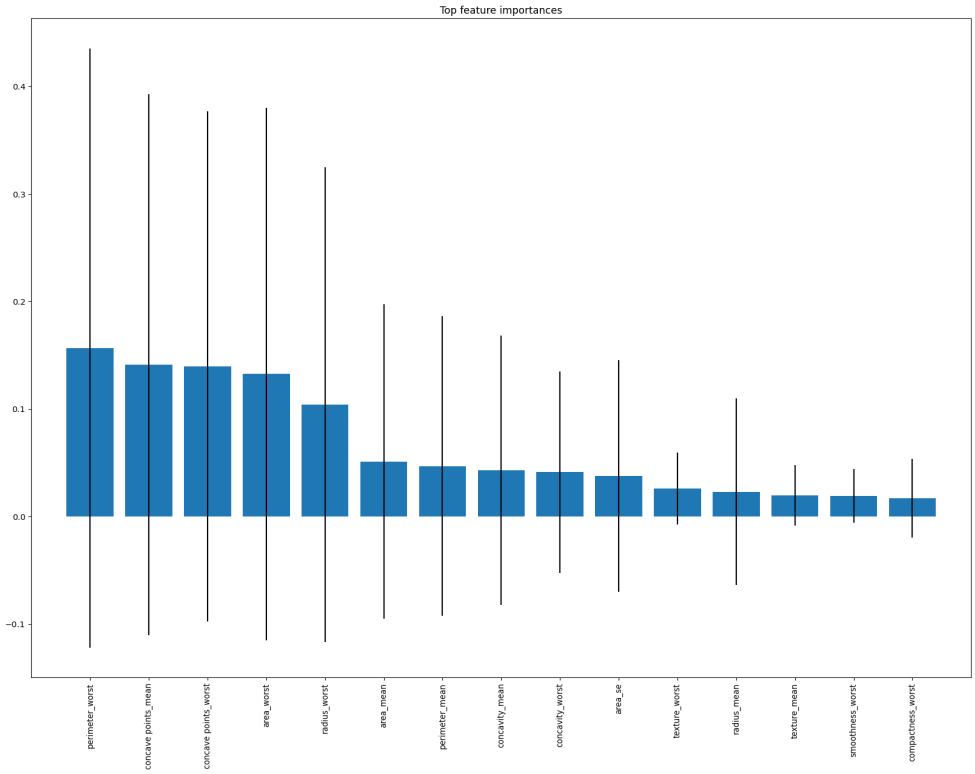
The optimal model chosen after gridsearch and cross validation (in code) was a random forest model. This is a representation of how the model determined that 15 features generated the highest accuracy.

3. **A bar and whisker graph displaying the importance of different features in classifying samples.**

Plotting which features contributed most to the model when classifying the samples as bar graphs with the black line representing standard deviation. The broad standard deviations indicated that there is a big variance in the importance of the feature in the different trees. Incorporating more features could potentially help with the accuracy of the model and capture more complex relationships.

```
rf_classifier
rf_bf = rf_classifier.fit(x_train_bf,y_train_bf)
importances = rf_bf.feature_importances_
std = np.std([tree.feature_importances_ for tree in rf_bf.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]

# Print the feature ranking
print("Feature ranking:")

for f in range(x_train_bf.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))

# Plot the feature importances of the forest

plt.figure(1, figsize=(20, 15))
plt.title("Top feature importances")
plt.bar(range(x_train_bf.shape[1]), importances[indices],
        yerr=std[indices], align="center")
plt.xticks(range(x_train_bf.shape[1]), x_train_bf.columns[indices],rotation=90)
plt.xlim([-1, x_train_bf.shape[1]])
plt.show()
```

-

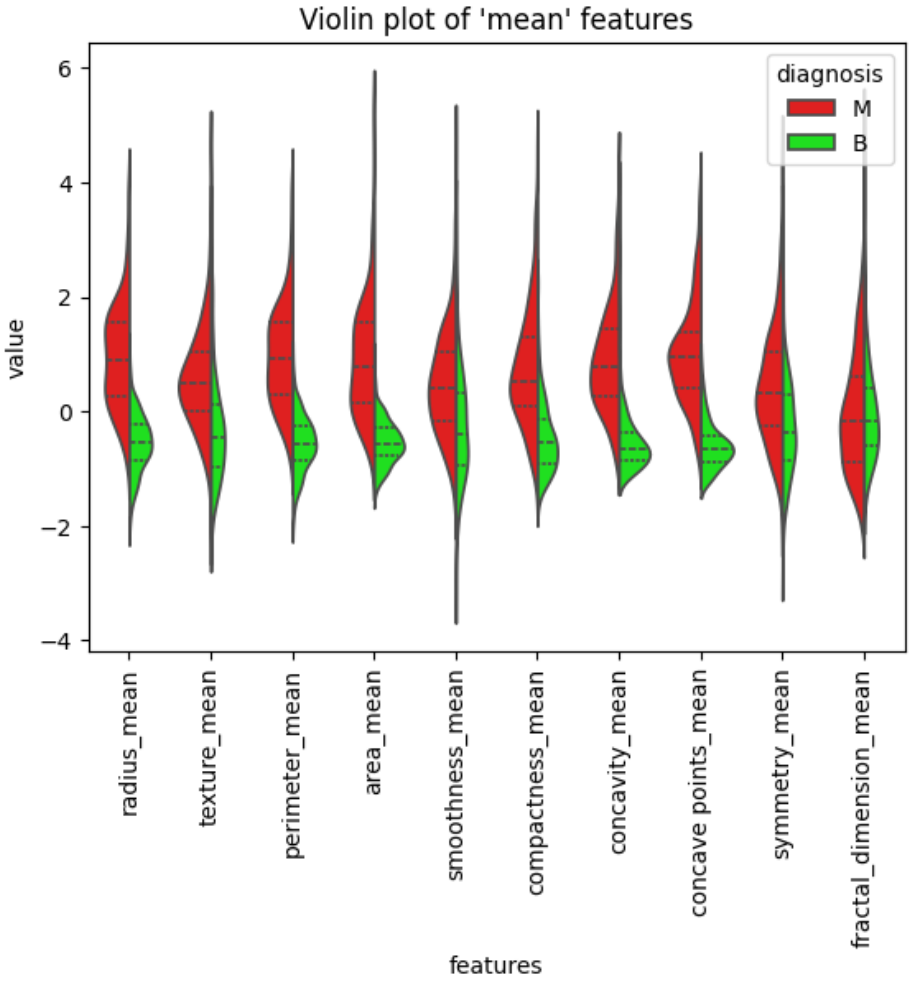


-

Seaborn graphs:

1. **Violin graphs of the standardised data for comparison.**

```
features_1 = pd.concat([label, noramlised_features.iloc[:,0:10]],axis=1) #isolating the feautres associated with 'mean'
features_1 = pd.melt(features_1, id_vars= 'diagnosis', var_name = 'features', value_name = 'value')

sns.violinplot(x='features', y = 'value', hue = 'diagnosis', data = features_1, split =True, inner = 'quart', palette = custom_colours)
plt.title("Violin plot of 'mean' features")
plt.xticks(rotation = 90); #Rotating the labels on the x axis for easier read
```

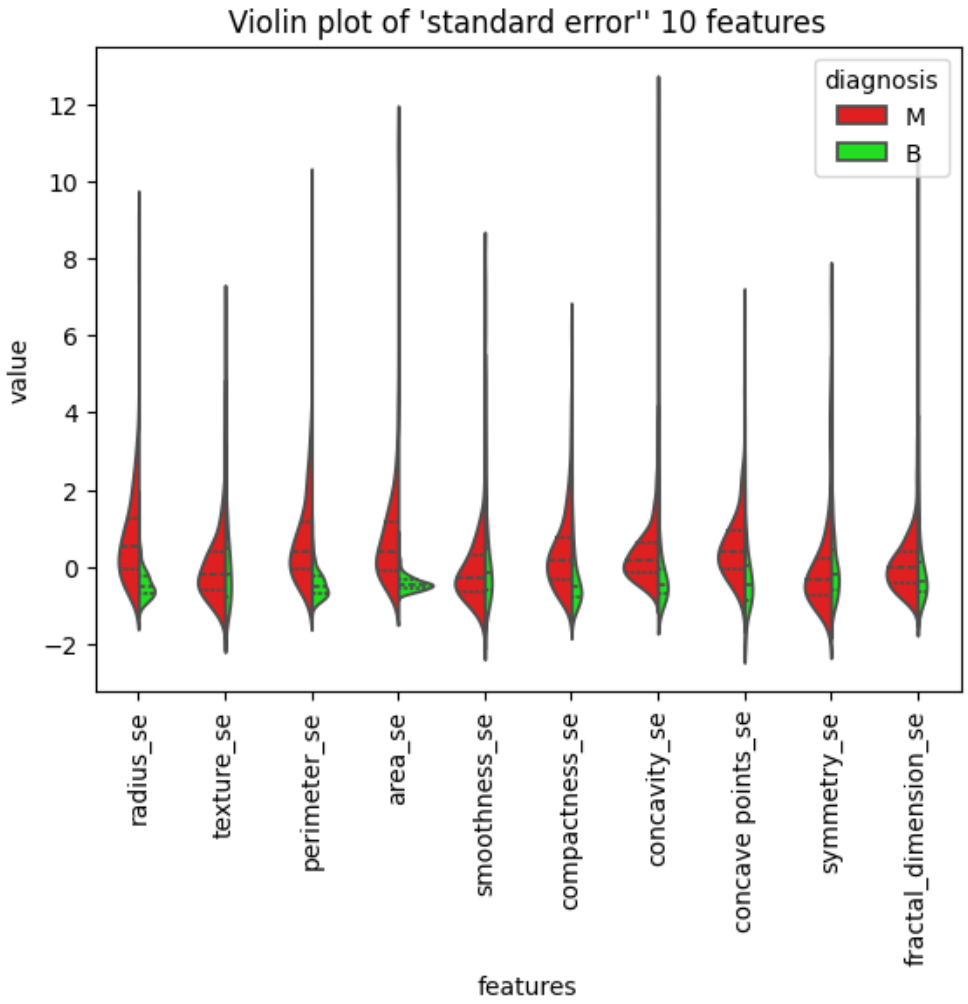


In order to understand if we could classify between the two groups we want to find features that have distinct differences.

We can see in many of the features here that the median of the benign samples is much lower than that of the malignant samples, i.e. radius_mean, permiter_mean, area_mean, compactness_mean or concavity_mean. Thus these features could potentially be used for the M/L model to distinguish between the diagnoses. It is also worth noting that features such as radius, permiter_mean and area_mean may be dependent on each other, due to their already established relationship, so every variable may not be necessary.

On the other hand features such as smoothness_mean or fractal_dimension_mean have very similar values between the groups thus would not make ideal candidates for classification.

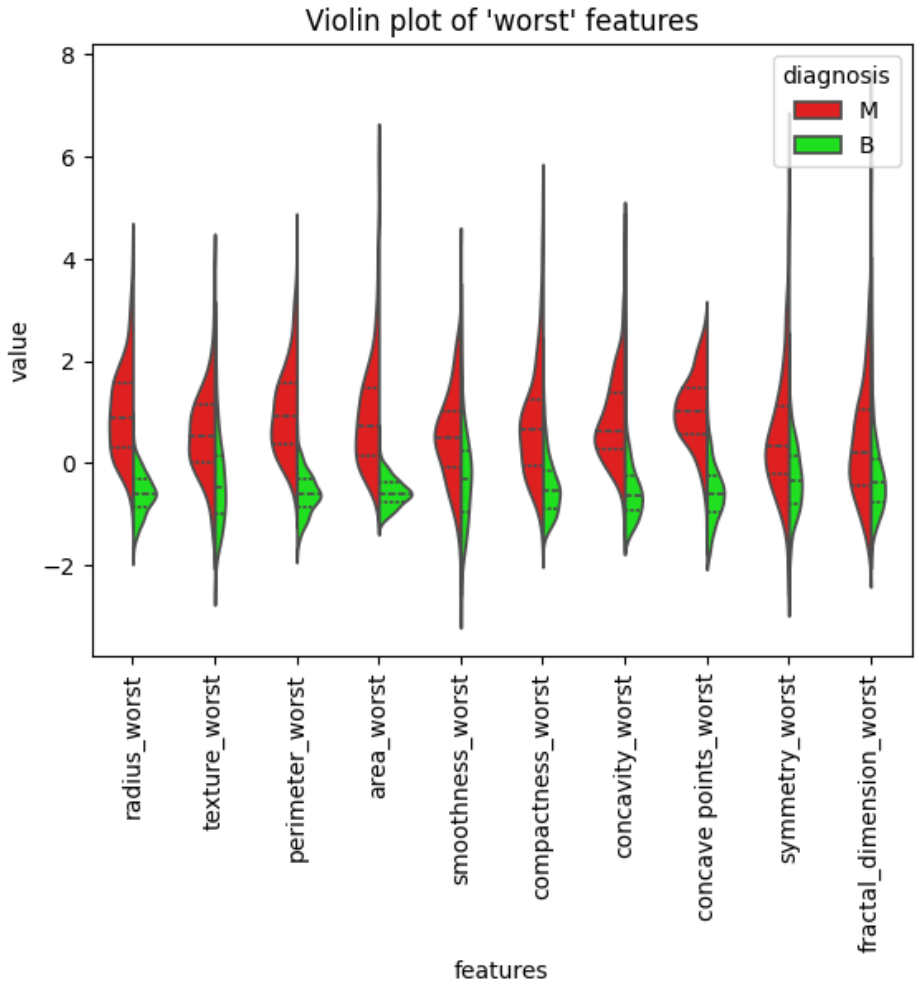
```
features_2 = pd.concat([label, noramlised_features.iloc[:,10:20]],axis=1) # creating the next group of 'standard error' feautres
features_2 = pd.melt(features_2, id_vars= 'diagnosis', var_name = 'features', value_name = 'value')

sns.violinplot(x='features', y = 'value', hue = 'diagnosis', data = features_2, split =True, inner = 'quart', palette = custom_colours)
plt.title("Violin plot of 'standard error'' 10 features")
plt.xticks(rotation = 90);
```



-

- In the above plot, radius_se, perimeter_se and area_se for the benign cells have values with medians lower than the malignant cells. Some of the benign cells do have the same values as the malignant cells indicating the complexity of accurate diagnosis based on such features.
- Again the features relating to smoothness and fractal dimension are quite similar so are unlikely to be useful in the model.

```
features_3 = pd.concat([label, noramlised_features.iloc[:,20:31]],axis=1) #creating the next group of feautres for 'worst'
features_3 = pd.melt(features_3, id_vars= 'diagnosis', var_name = 'features', value_name = 'value')

sns.violinplot(x='features', y = 'value', hue = 'diagnosis', data = features_3, split =True, inner = 'quart', palette = custom_colours)
plt.title("Violin plot of 'worst' features")
plt.xticks(rotation = 90);
```



-

Similar to the other features it is those related to radius, perimeter, area and concavity that appear to have a good separation between the groups that can be informative for the model.

Further symmetry and fractal dimension related features don't appear to provide much information to distinguish the two groups.

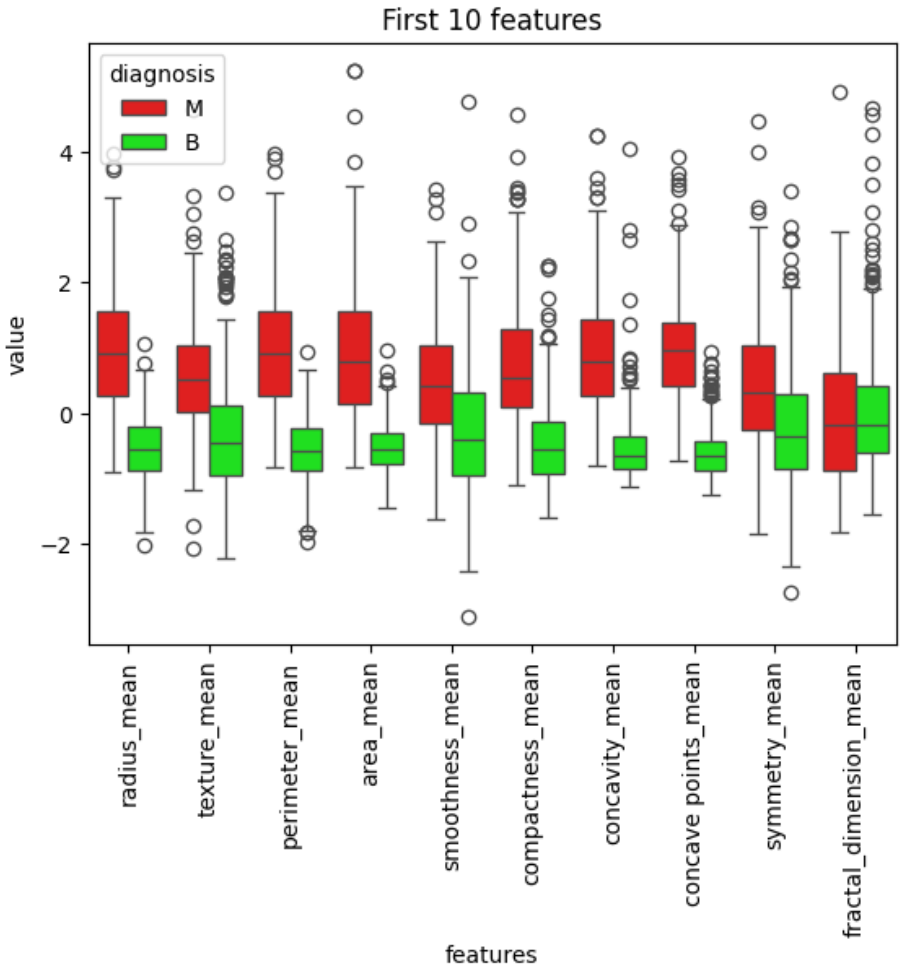
Outliers are also important to consider. These are seen in the tails of the violin plots. These can also be observed by generating box and whisker plots, which has been completed below. However, it is important to determine if the 'outlier' is due to data mis-entry, natural variation or whether these are in fact the true values. As I am not an expert in this area it is hard to know what is truly noise in the data. These outliers could represent an aggressive form of cancer or a misclassification of that sample due to human error.

For the purpose of this study the outliers will be left, but we may return to this point if the accuracy of the model is not sufficient.

The first 10 features have been plotted to demonstrate the visual representation.

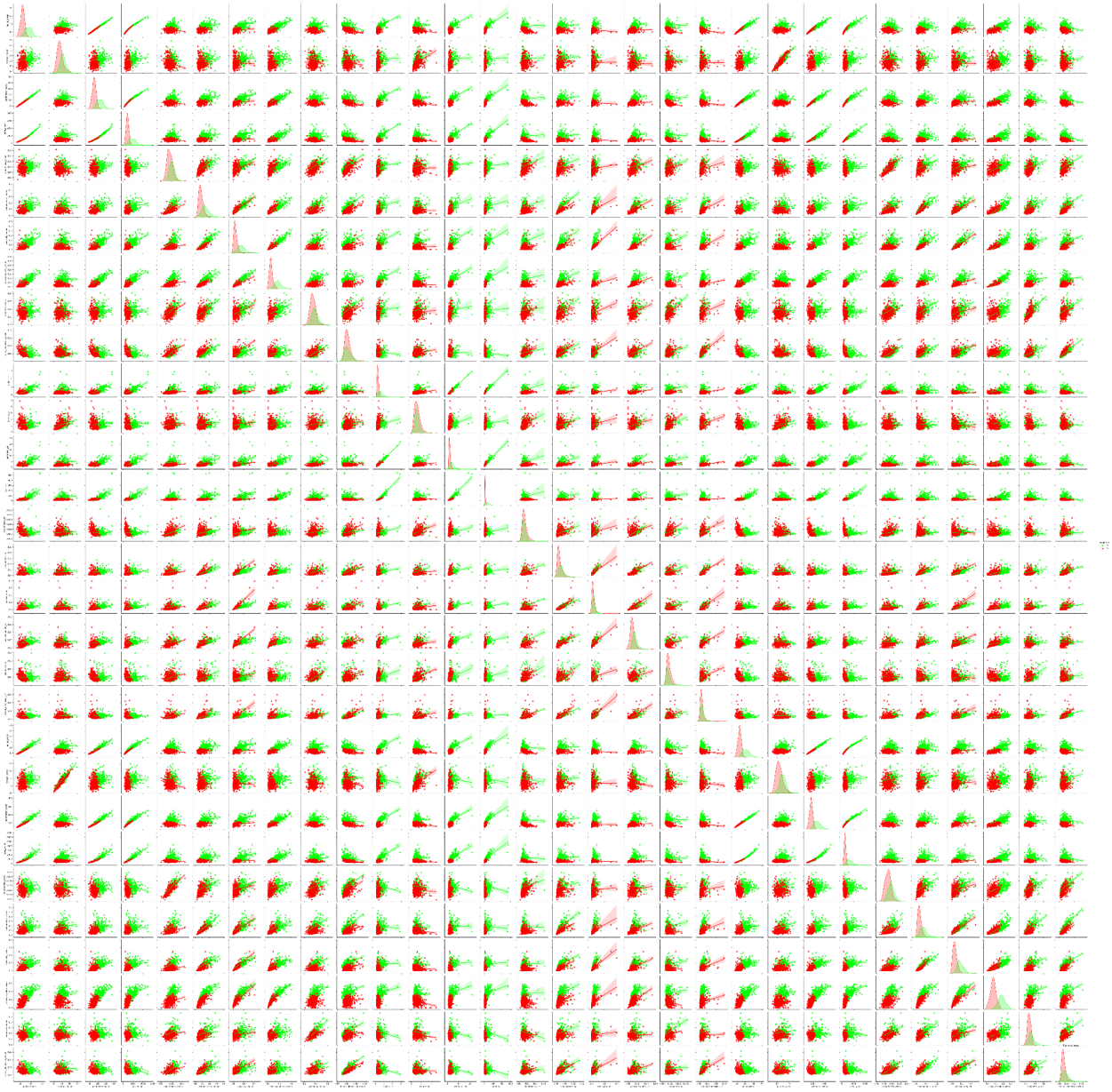
2. **Box and whisker plots to identify outliers**

```
sns.boxplot(x='features', y= 'value', hue = 'diagnosis', palette = custom_colours, data = features_1)
plt.title('First 10 features')
plt.xticks(rotation=90);
```



3. **Pairplot**

```
sns.pairplot(df, hue='diagnosis', kind='reg', palette = custom_colours);
```

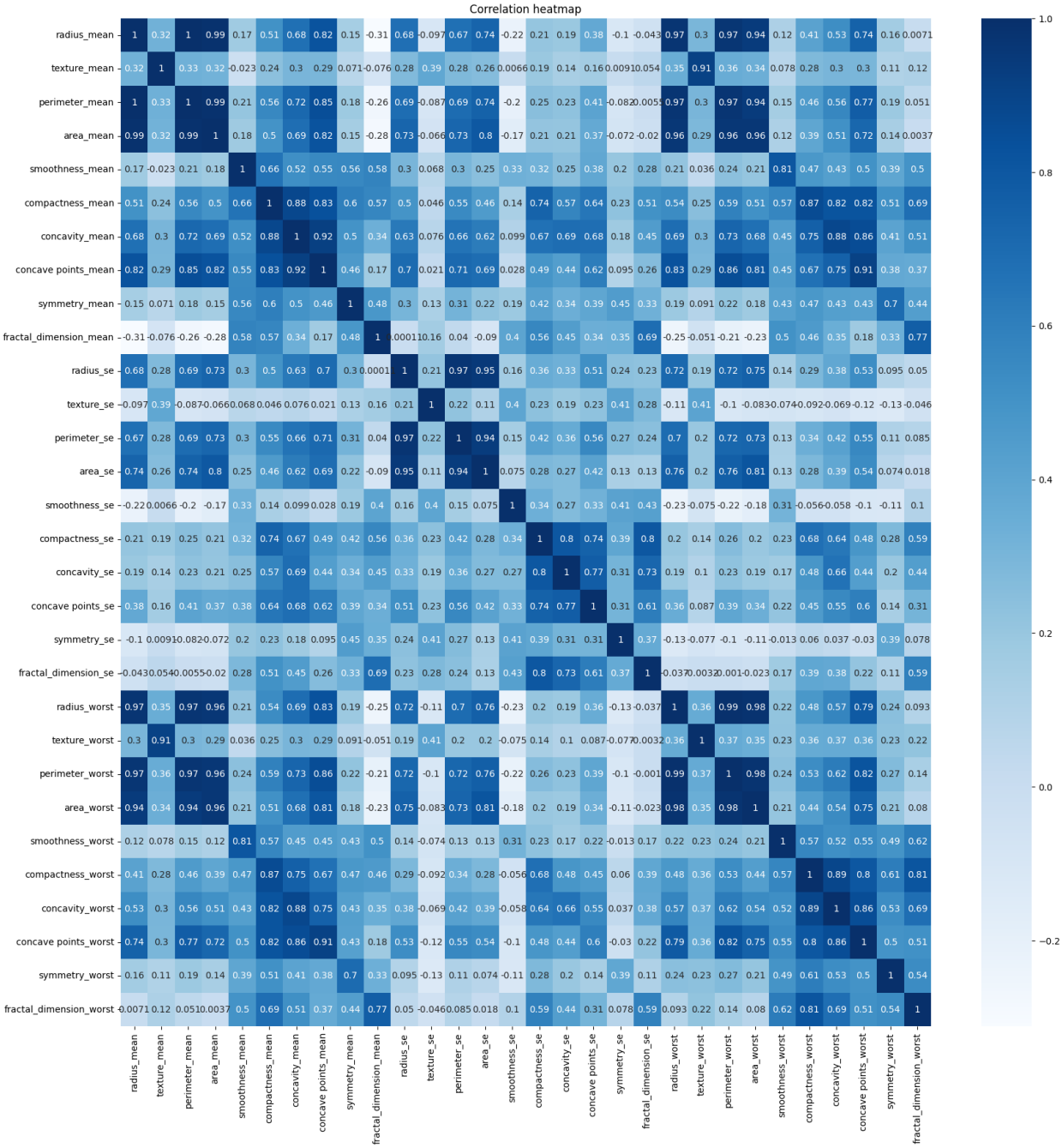


As mentioned, many of these features may be related: i.e. area, perimeter and mean. We can visualise this by creating scatter plots of the relationships between two variables as seen above. As we can see it is very visually and labour intensive to assess which features are correlated. Zooming in on the plot to go through the individual plots would take some time.

Instead we can generate a correlation matrix, using Pearson's R as a measure of correlation, and display it as a heatmap. This is more concise and will be easier to pick out the highly correlated features based on colour.

4. **Heatmap of correlation matrix**

```
plt.subplots(figsize=(20,20))
sns.heatmap(feature_df.corr(), annot=True, cmap='Blues');
plt.title('Correlation heatmap');
```

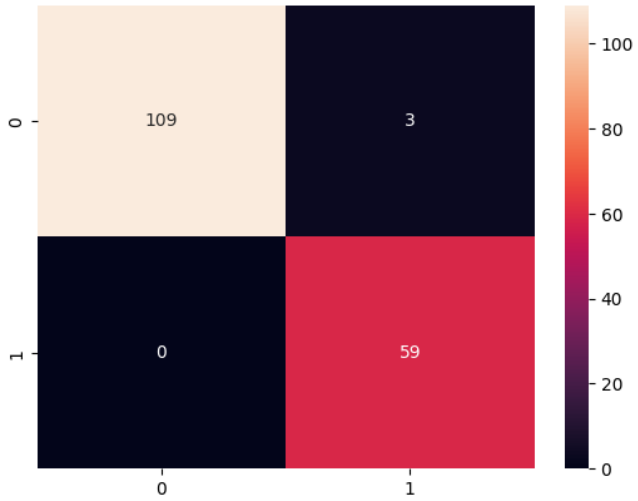


-

While this is visually easier to identify the related features and see the Pearson's variable quite easily an interactive heat map has been created below to easily identify the two features.

5. **Heatmap of confusion matrix.**

-
```
con_mat_rf_1 = confusion_matrix(y_test_bf,y_pred_bf)
sns.heatmap(con_mat_rf_1, annot=True, fmt = 'd')
```



This model built on 15 variables gives us an accuracy of 98% meaning that only 2 in 100 people would get an incorrect classification. However, the dataset is rather unbalanced so we can look at the other values of precision and recall. With a value of 1 for recall the model does not miss any positive diagnosis but it does incorrectly label some as positive as the precision is 95%. In the context of diagnosing cancer this is likely acceptable that no positive cases are missed but some negative cases could be sent for further testing.

Bokeh graph

- **Please refer to the code submission for the bokeh graph**.

```
# Calculate the correlation matrix
corr_matrix = feature_df.corr();

# Create a list of column names for x-axis and y-axis
columns = list(corr_matrix.columns)
rows = list(corr_matrix.index)

# Convert the correlation matrix to a list of tuples (x, y, value)
data = [(columns[i], rows[j], corr_matrix.iloc[j, i]) for i in range(len(columns)) for j in range(len(rows))]

correlation_df = pd.DataFrame(data)
correlation_df.rename(columns={0: "x", 1: "y", 2: "correlation"}, inplace=True)

source = ColumnDataSource(correlation_df)
color_mapper = LinearColorMapper(palette="Viridis256", low=correlation_df['correlation'].min(), high=correlation_df['correlation'].max())

# Create the Bokeh figure
p = figure(title="Correlation Heatmap", x_range=columns, y_range=list(reversed(rows)),
           toolbar_location=None, tools="hover",
           tooltips=[("Variables", "@y, @x"), ("correlation", "@correlation")])

# Add rectangles representing the correlation values
r = p.rect(x="x", y="y", width=1, height=1, source=source,
           fill_color={'field': 'correlation', 'transform': color_mapper}, line_color=None)

# Add hover tool
hover = HoverTool()
hover.tooltips = [("Variables", "@y, @x"), ("correlation", "@correlation")]
p.add_layout(r.construct_color_bar(
    major_label_text_font_size="7px",
    ticker=BasicTicker(),
    formatter=PrintfTickFormatter(format="%d%%"),
    label_standoff=6,
    border_line_color=None,
    padding=5
), 'right')

p.xaxis.major_label_orientation = 1.5
output_notebook()
show(p)
```

Creating this interactive graph allows us to easily identify which two variables were the high correlation. In this graph the brighter the colour the stronger the relationship.

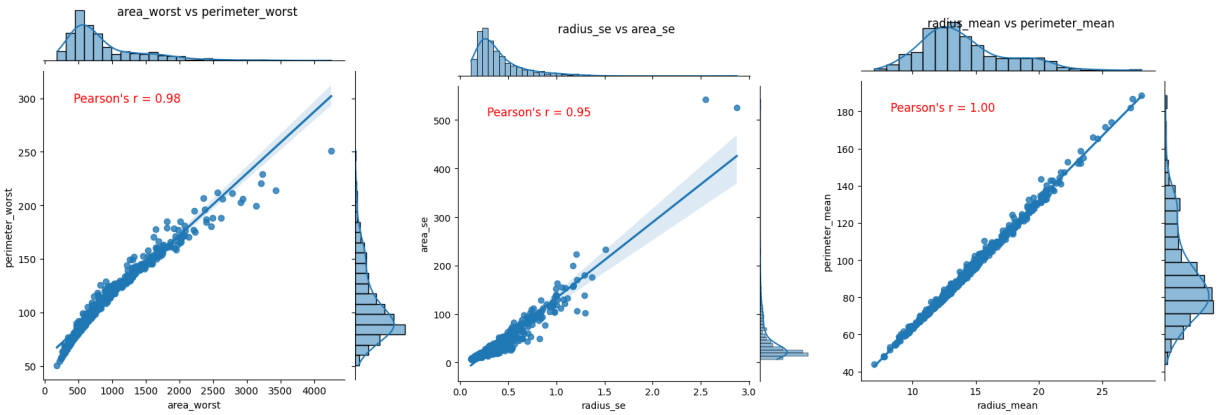
The highly correlated features are as predicted. There are strong correlations between radius, perimeter, and area related variables. This is likely due to the intrinsic relationship between these values and the corresponding mathematical calculations.

Including correlated variables in the model can lead to redundancy as they are both giving us the same information and do not provide additional predictive power and can increase computational load.

To visualise the correlation between two variables at a closer level than the pairplot one will be plotted separately for easier visualisation.

6. Joint plot of chosen features to investigate potential correlation.

```
make_jointplot('area_worst','perimeter_worst')
make_jointplot('radius_se', 'area_se')
make_jointplot('radius_mean', 'perimeter_mean')
```



As we can see these two variables are almost identical in all three graphs. They all have a positive correlation, i.e. as one increases so does the other. The majority of points also cluster tightly around the line and this is captured by the high Pearson's R correlation coefficient.

Insights:

1. There is an imbalance in the number of labels, as seen in the bar graph and the summary stats, that needs to be considered when choosing the machine learning model.
2. Biology is highly complex, this is seen in the similarities of values between certain features in the violin plots, which might increase the complexity of the model as a larger number of features may be needed to build an accurate model.
3. It is difficult to determine if outliers in the data are due to true noise, as this would require an expert opinion. This may impact the accuracy of the model. If satisfactory accuracy cannot be reached, we may need to remove these values. These can be seen in the tails of the violin plots and the box plots.
4. We need to consider carefully which features we want to include in our model as there is a high degree of correlation between features that would lead to redundancy and increased computational load in the final model.
5. Out of three different classification models, (support vector machine, random forest and logistic regression), random forest was selected as the most accurate based on the hyperparameters that were tested using grid search cross validation.
   a.

	model	best_score	best_params
0	svm	0.615601	{'C': 0.1, 'gamma': 'scale', 'kernel': 'sigmoid'}
1	random_forest	0.962247	{'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
2	logistic_regression	0.957215	{'C': 10, 'penalty': 'l1'}

6. 15 features from the df were identified as the optimal number of features to use in the model as this provided the highest cross validation score. This reduces computational complexity of the model.
7. The top 3 features contributing to the model are perimeter worst, concave points worst and mean. This information has the potential to contribute to our biological understanding of how cells deviate from normal activities to malignancy.
8. We can accurately diagnose 98% of patients using this model based on the confusion matrix generated and the corresponding precision and recall values.

Machine Learning:

Based on this data alone theoretically we would be able to diagnose patients based on computer generated reports of histology slides.

This model also has potential to be expanded to enable additional predictions should we could return to the same patients and collect additional data such as survival outcomes or demographics. Integrating this data could facilitate prognosis predictions based on survival data and identification of risk factors from demographic data. Together, this information can guide physicians in clinical decision making and tailoring treatments decisions.

For diagnosis, classification models are more suitable as they can categorise patients into distinct groups such as malignant or benign. In terms of risk predictions, a model could generate a probability score but to relay this information to patients a more interpretable format such as risk groups may be more understandable than a percentage.

In summary, utilising machine learning techniques on the histology data used in this project can accurately diagnose patients.

Conclusion

Accurate classification models can be built on patient data and be improved and adapted as more information becomes available. These models hold great potential for integration into healthcare systems to speed up and improve diagnosis for patients.