PROJECT REPORT

Hilal İlçin – Middlesex University, London, UK

At the end of the report, I shared the brief of the article that I read about project topic.

Project Topic: Benign and Malignant Tumor Classification using several data mining methods.

I used Breast Cancer Wisconsin (Diagnostic) Data Set from Kaggle to predict whether the cancer is benign or malignant. (https://www.kaggle.com/uciml/breast-cancer-wisconsin-data)

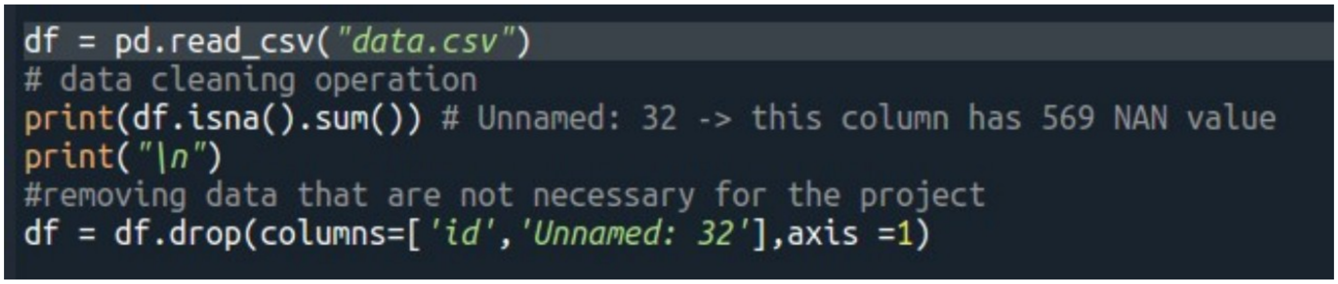
Dataset

Breast Cancer Wisconsin (Diagnostic) Data Set

Predict whether the cancer is benign or malignant

UCI Machine Learning • updated 5 years ago (Version 2)

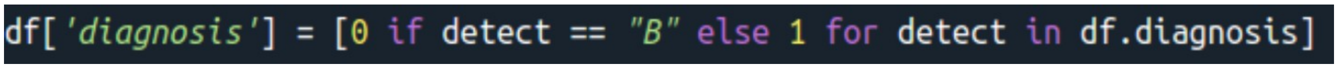
Data Cleaning:



```
df = pd.read_csv("data.csv")
# data cleaning operation
print(df.isna().sum()) # Unnamed: 32 -> this column has 569 NAN value
print("\n")
#removing data that are not necessary for the project
df = df.drop(columns=['id','Unnamed: 32'],axis =1)
```

I cleaned the data because some attributes are not necessary for the project. These attributes are ‘id’ and ‘Unnamed: 32’.

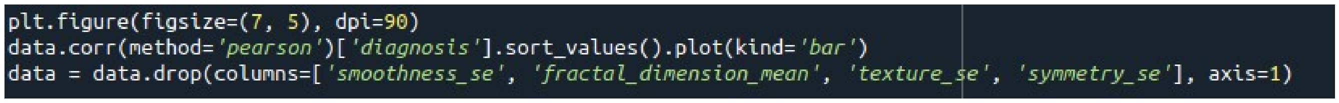
- After this operation I converted my target values from string to binary representation. ”diagnosis” column was represented by letters (M = malignant, B = benign). I changed it to binary values as Benign = 0 and Malignant = 1 with the help of following code part.



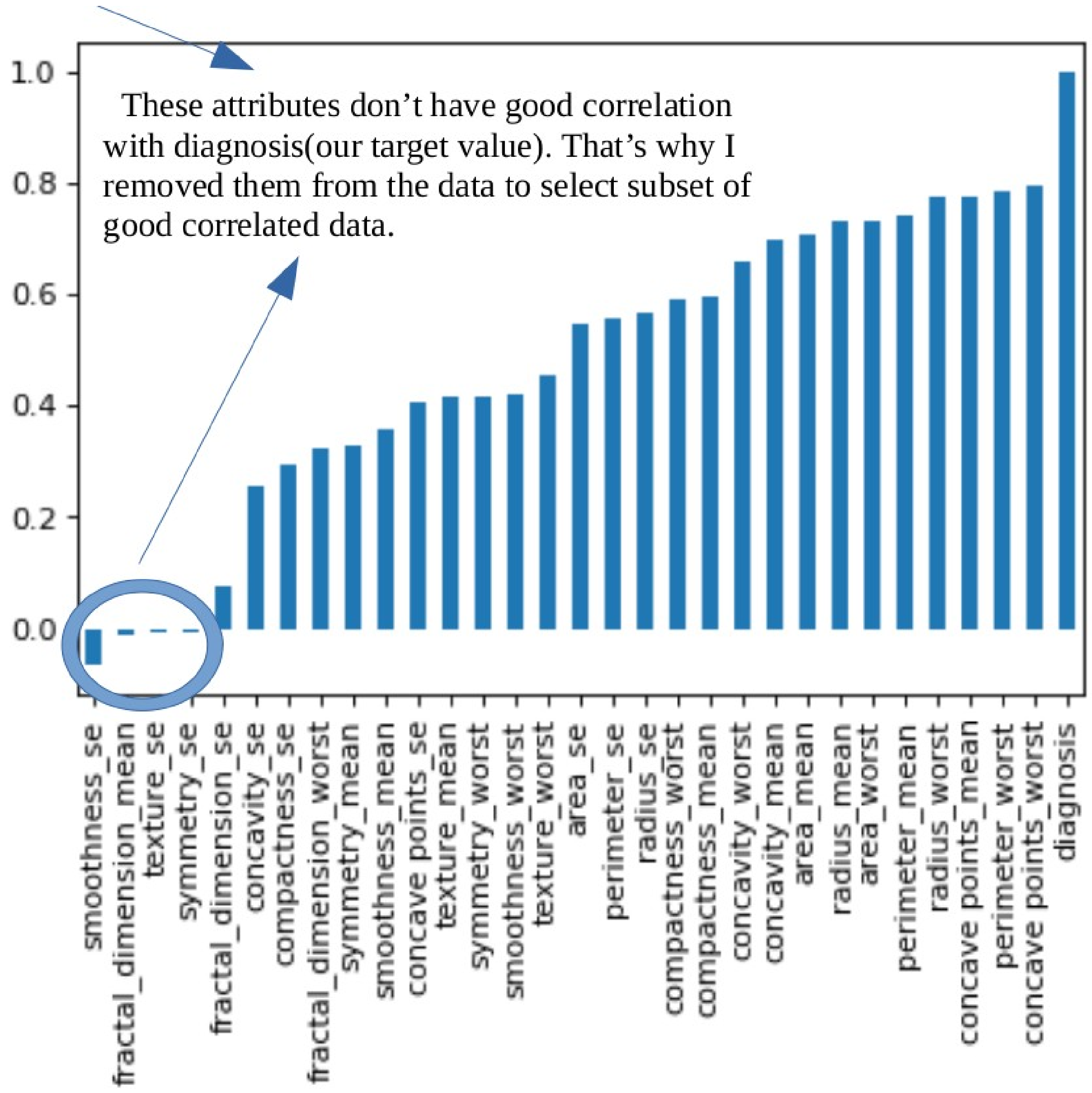
```
df['diagnosis'] = [0 if detect == "B" else 1 for detect in df.diagnosis]
```

Filter Based Feature Selection:

- I analyzed the correlation between diagnosis and other features. I used “Pearson correlation” method for this operation. The test statistic Pearson's correlation coefficient assesses the statistical link, or association, between two continuous variables. Because it is based on the method of covariance, it is known as the best method for quantifying the relationship between variables of interest. It provides information on the magnitude and direction of the relationship's link, or correlation.



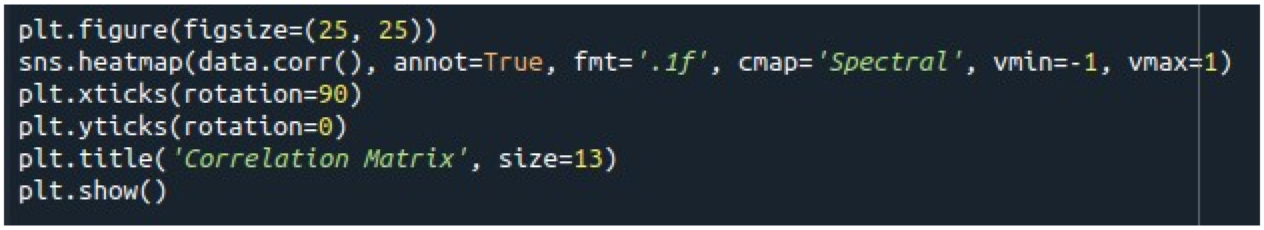
```
plt.figure(figsize=(7, 5), dpi=90)
data.corr(method='pearson')['diagnosis'].sort_values().plot(kind='bar')
data = data.drop(columns=['smoothness_se', 'fractal_dimension_mean', 'texture_se', 'symmetry_se'], axis=1)
```



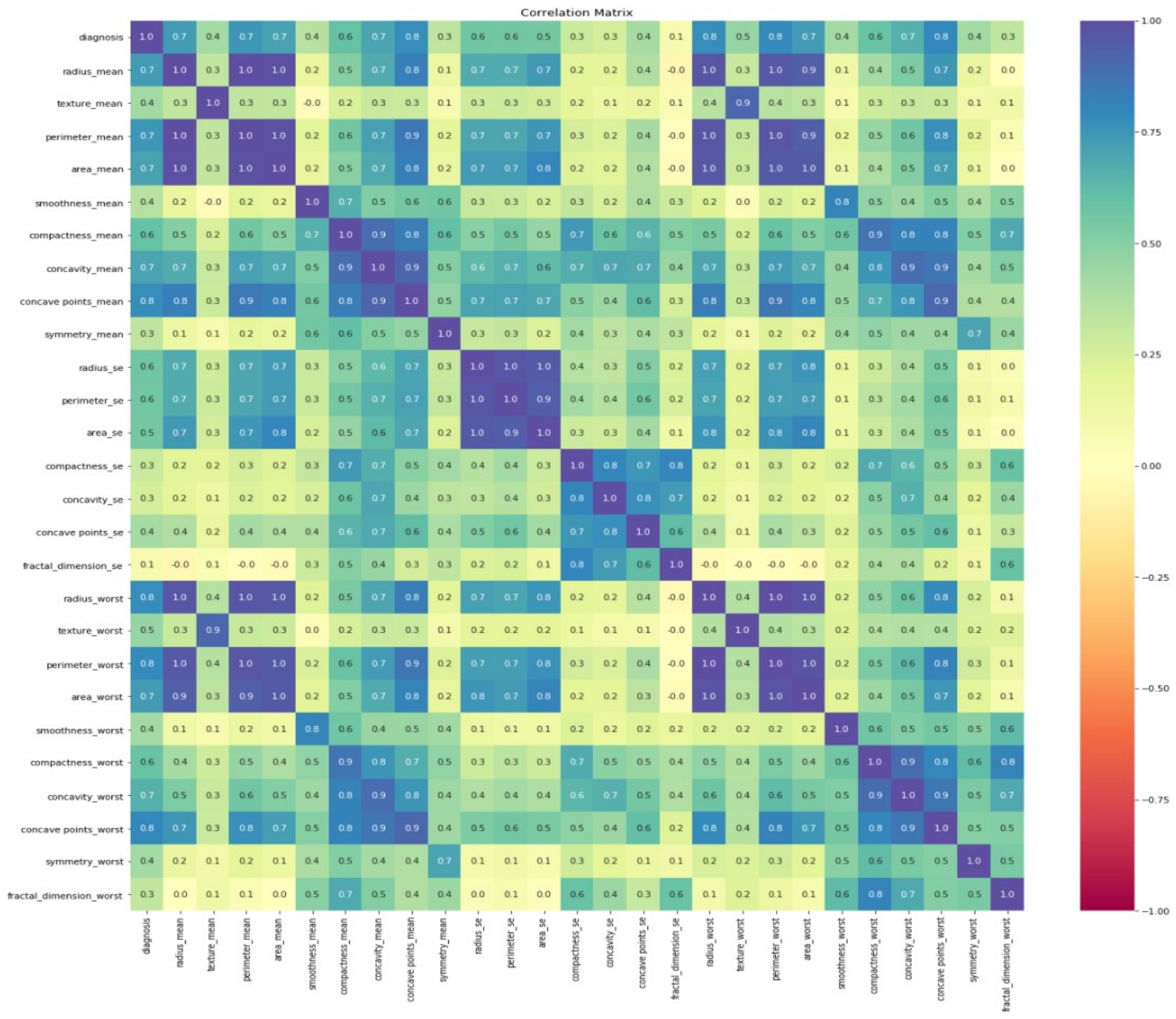
These attributes don’t have good correlation with diagnosis(our target value). That’s why I removed them from the data to select subset of good correlated data.

Correlation values with the help of Correlation Matrix:

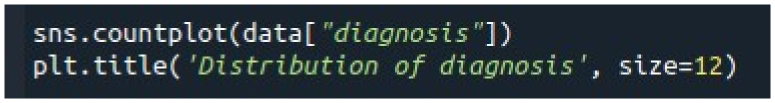
- I wanted to see correlation values of all of the attributes and I used correlation matrix for this operation.



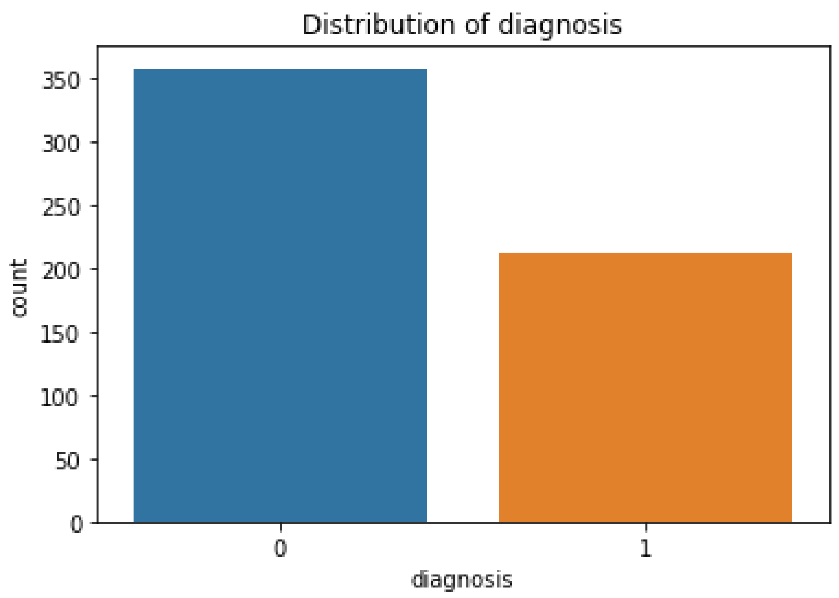
```
plt.figure(figsize=(25, 25))
sns.heatmap(data.corr(), annot=True, fmt='.1f', cmap='Spectral', vmin=-1, vmax=1)
plt.xticks(rotation=90)
plt.yticks(rotation=0)
plt.title('Correlation Matrix', size=13)
plt.show()
```



After this operation. I wanted to see distribution of the target values(diagnosis) and I wrote the following code segment for doing this.



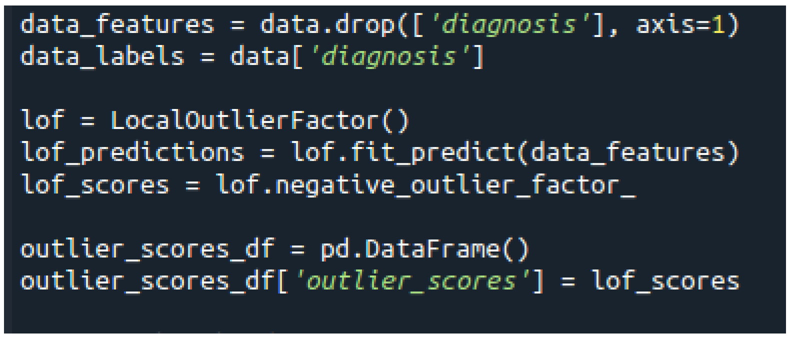
```
sns.countplot(data["diagnosis"])
plt.title('Distribution of diagnosis', size=12)
```



- Diagnosis values are in balanced level. There is no huge balance difference between the two binary values. We can get good results during classification operations.

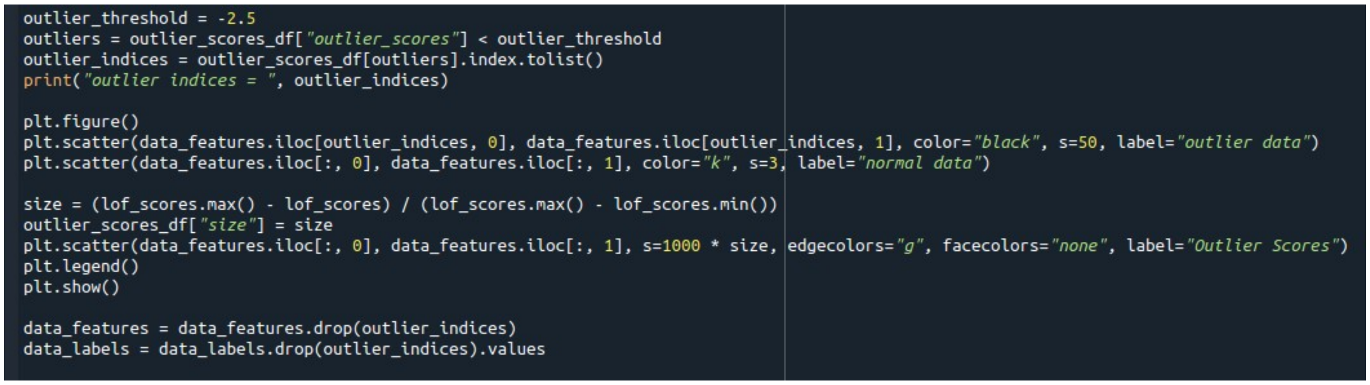
Outlier Detection with LOF(Local Outlier Factor):

I used LocalOutlierFactor method for outlier detection. First I separate the data as with diagnosis and without diagnosis. In the following steps I applied the data without diagnosis for prediction fitting operation of LOF method and got LOF scores at the end of the process. I saved these LOF scores to used them for outlier detection in the following parts.

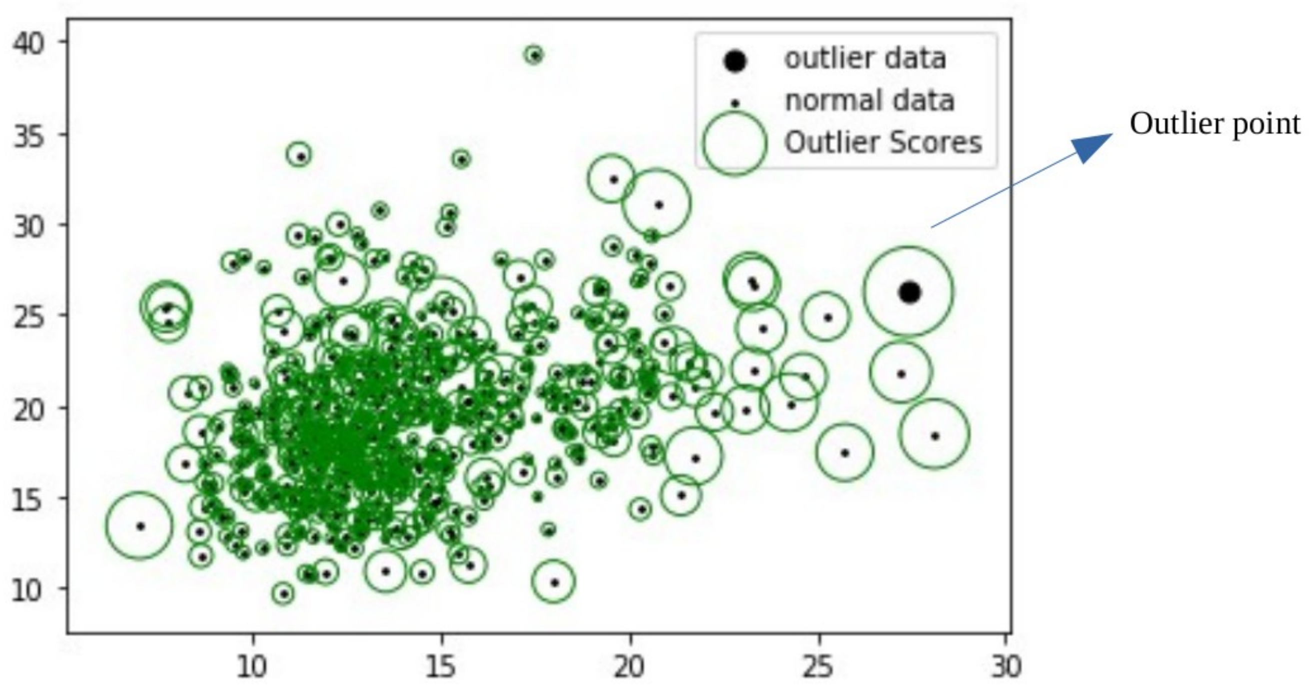


```
data_features = data.drop(['diagnosis'], axis=1)
data_labels = data['diagnosis']

lof = LocalOutlierFactor()
lof_predictions = lof.fit_predict(data_features)
lof_scores = lof.negative_outlier_factor_

outlier_scores_df = pd.DataFrame()
outlier_scores_df['outlier_scores'] = lof_scores
```

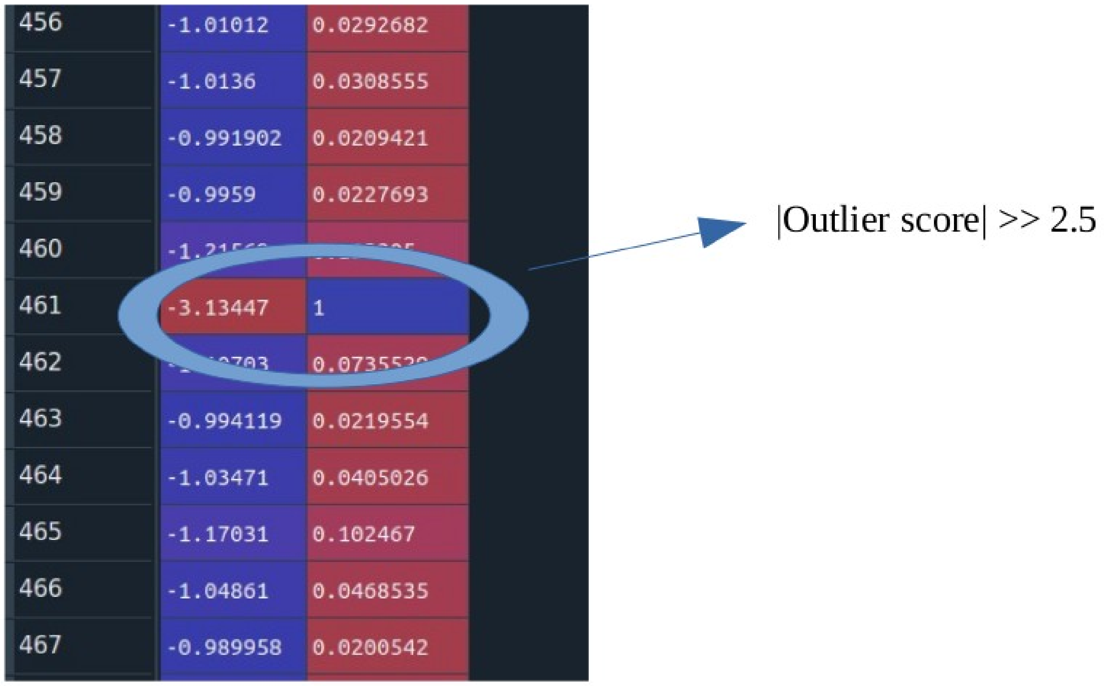
I set an outlier detection limit for actual outlier scores. I chose it as -2.5 and it means that it shows outliers values above 2.5.



```
outlier_threshold = -2.5
outliers = outlier_scores_df["outlier_scores"] < outlier_threshold
outlier_indices = outlier_scores_df[outliers].index.tolist()
print("outlier indices = ", outlier_indices)

plt.figure()
plt.scatter(data_features.iloc[outlier_indices, 0], data_features.iloc[outlier_indices, 1], color="black", s=50, label="outlier data")
plt.scatter(data_features.iloc[:, 0], data_features.iloc[:, 1], color="k", s=3, label="normal data")

size = (lof_scores.max() - lof_scores) / (lof_scores.max() - lof_scores.min())
outlier_scores_df["size"] = size
plt.scatter(data_features.iloc[:, 0], data_features.iloc[:, 1], s=1000 * size, edgecolors="g", facecolors="none", label="Outlier Scores")
plt.legend()
plt.show()

data_features = data_features.drop(outlier_indices)
data_labels = data_labels.drop(outlier_indices).values
```



Outlier index is 461 in my data and if we check the value of 461. index then we see that it has value which is greater than 2.5 as you see from following section.



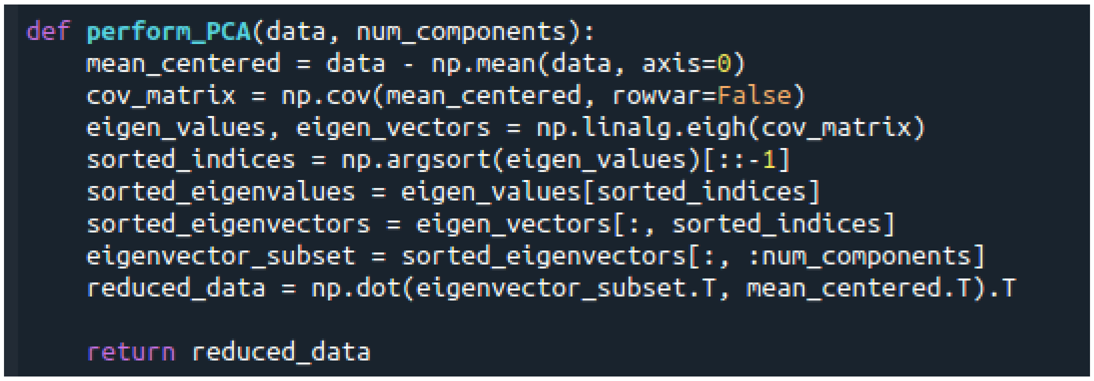
456	-1.01012	0.0292682
457	-1.0136	0.0308555
458	-0.991902	0.0209421
459	-0.9959	0.0227693
460	-1.215[illegible]	[illegible]
461	-3.13447	1
462	[illegible]	0.073552[illegible]
463	-0.994119	0.0219554
464	-1.03471	0.0405026
465	-1.17031	0.102467
466	-1.04861	0.0468535
467	-0.989958	0.0200542

Feature Extraction using PCA (Principal Component Analysis):

In data science, Principle Component Analysis (PCA) is a typical feature extraction method. PCA works by finding the eigenvectors of a covariance matrix with the highest eigenvalues and then projecting the data onto a new subspace with the same or less dimensions. In my data, I reduced number of dimensions from 32 to 2 with the help of PCA method. I implemented the PCA method by myself without using any library with the help of following steps.

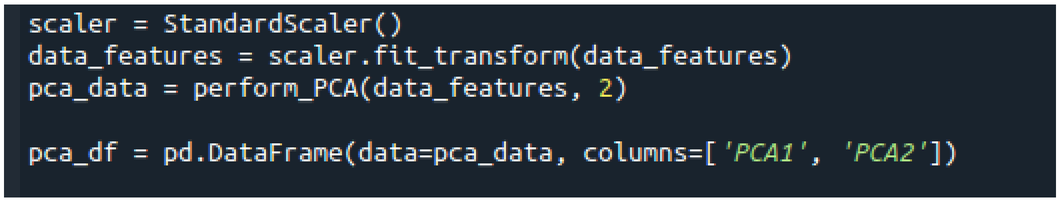
Step 1: Standardize the dataset.
Step 2: Calculate the covariance matrix for the features in the dataset.
Step 3: Calculate the eigenvalues and eigenvectors for the covariance matrix.
Step 4: Sort eigenvalues and their corresponding eigenvectors.
Step 5: Pick k eigenvalues and form a matrix of eigenvectors.
Step 6: Transform the original matrix.

PCA implementation:



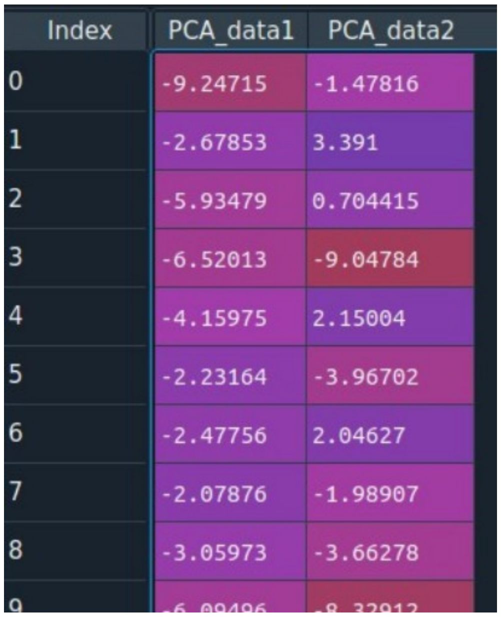
```python
def perform_PCA(data, num_components):
    mean_centered = data - np.mean(data, axis=0)
    cov_matrix = np.cov(mean_centered, rowvar=False)
    eigen_values, eigen_vectors = np.linalg.eigh(cov_matrix)
    sorted_indices = np.argsort(eigen_values)[::-1]
    sorted_eigenvalues = eigen_values[sorted_indices]
    sorted_eigenvectors = eigen_vectors[:, sorted_indices]
    eigenvector_subset = sorted_eigenvectors[:, :num_components]
    reduced_data = np.dot(eigenvector_subset.T, mean_centered.T).T

    return reduced_data
```

Using of PCA:



```python
scaler = StandardScaler()
data_features = scaler.fit_transform(data_features)
pca_data = perform_PCA(data_features, 2)

pca_df = pd.DataFrame(data=pca_data, columns=['PCA1', 'PCA2'])
```

PCA dataframe with 2 columns (instead of 32):



Index	PCA_data1	PCA_data2
0	-9.24715	-1.47816
1	-2.67853	3.391
2	-5.93479	0.704415
3	-6.52013	-9.04784
4	-4.15975	2.15004
5	-2.23164	-3.96702
6	-2.47756	2.04627
7	-2.07876	-1.98907
8	-3.05973	-3.66278
9	-6.09496	-8.32912

CLASSIFICATION:

I used RandomForest, GradientBoosting, AdaBoost methods for classification and compared results of them. I wanted to make classification with algorithms that we didn't cover much in the lessons. I wanted to research new topics and learn new things in addition to what I learned in the course.

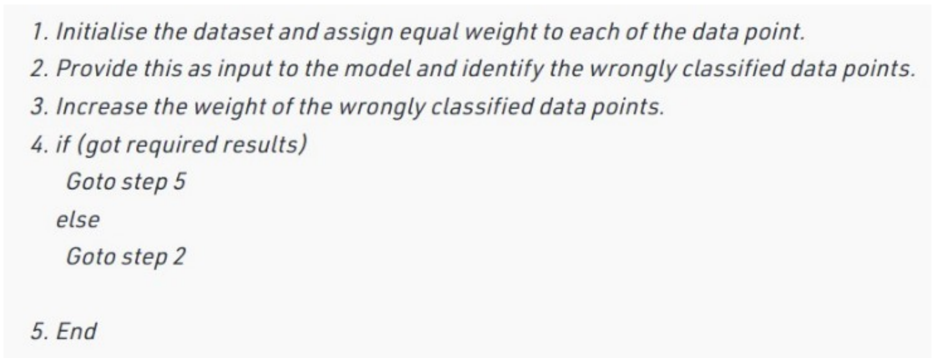
Random Forest: It is a classification algorithm consisting of many decisions trees. It uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree.

Gradient Boosting: It is a machine learning technique for regression and classification problems, which
produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.

AdaBoost: It was the first really successful boosting algorithm developed for the purpose of binary classification. AdaBoost is short for Adaptive Boosting and is a very popular boosting technique that combines multiple "weak classifiers" into a single "strong classifier".

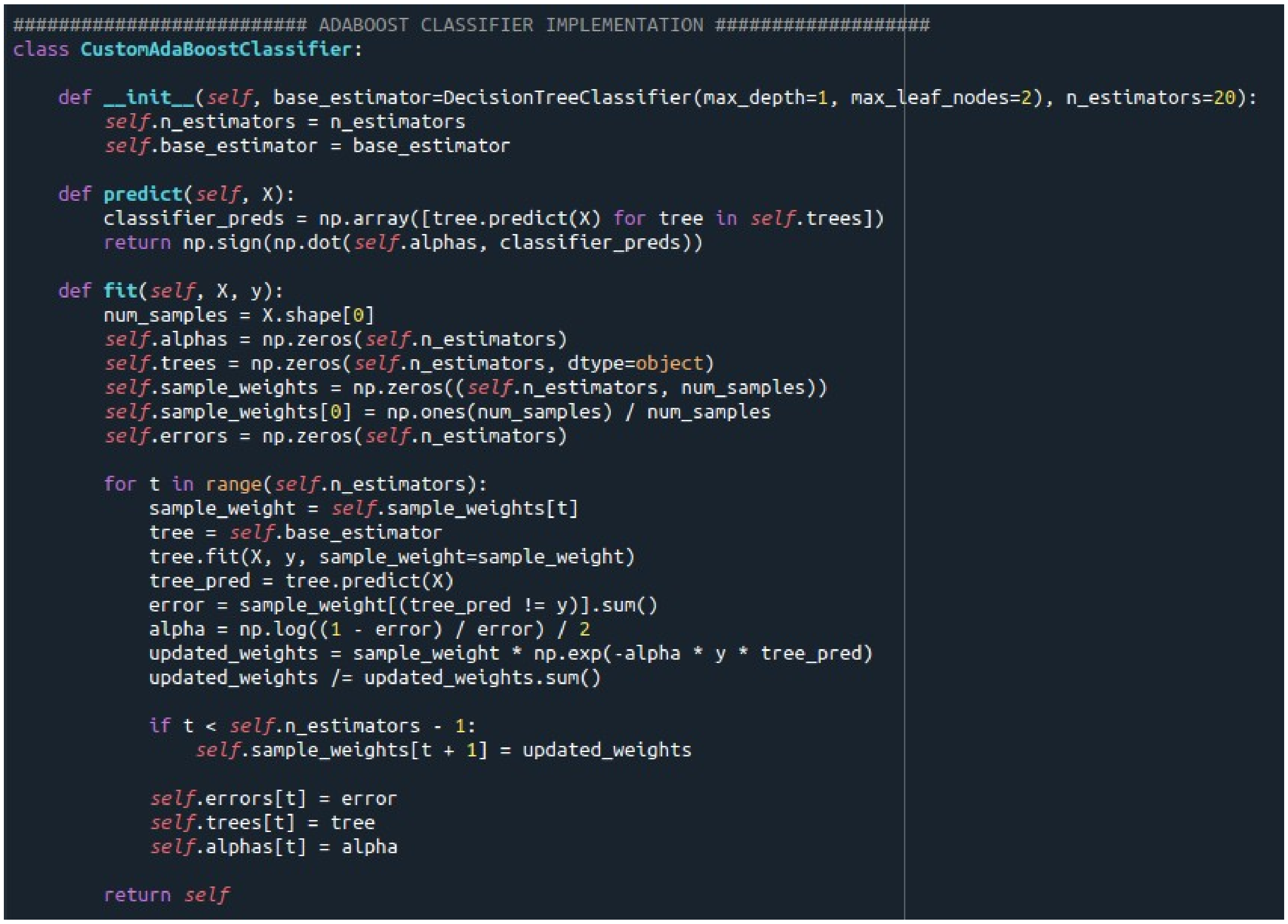
I used RandomForest and GradientBoosting from sklearn library but I implemented the AdaBoost algorithm by myself.

Algorithm of the AdaBoost:

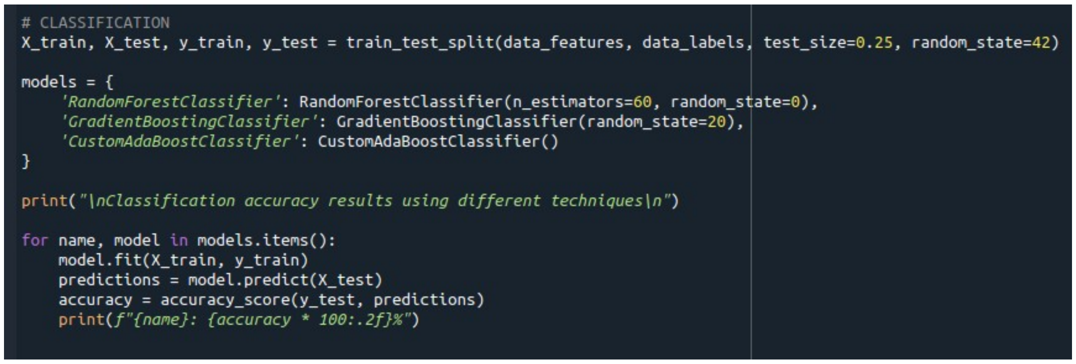


```
1. Initialise the dataset and assign equal weight to each of the data point.
2. Provide this as input to the model and identify the wrongly classified data points.
3. Increase the weight of the wrongly classified data points.
4. if (got required results)
     Goto step 5
   else
     Goto step 2

5. End
```

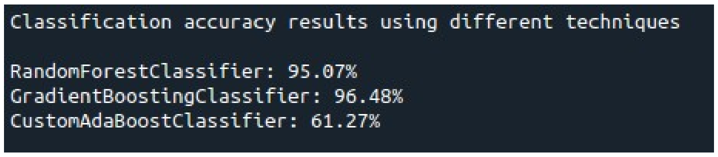
AdaBoost Implementation:



```python
########################### ADABOOST CLASSIFIER IMPLEMENTATION ###################
class CustomAdaBoostClassifier:

    def __init__(self, base_estimator=DecisionTreeClassifier(max_depth=1, max_leaf_nodes=2), n_estimators=20):
        self.n_estimators = n_estimators
        self.base_estimator = base_estimator

    def predict(self, X):
        classifier_preds = np.array([tree.predict(X) for tree in self.trees])
        return np.sign(np.dot(self.alphas, classifier_preds))

    def fit(self, X, y):
        num_samples = X.shape[0]
        self.alphas = np.zeros(self.n_estimators)
        self.trees = np.zeros(self.n_estimators, dtype=object)
        self.sample_weights = np.zeros((self.n_estimators, num_samples))
        self.sample_weights[0] = np.ones(num_samples) / num_samples
        self.errors = np.zeros(self.n_estimators)

        for t in range(self.n_estimators):
            sample_weight = self.sample_weights[t]
            tree = self.base_estimator
            tree.fit(X, y, sample_weight=sample_weight)
            tree_pred = tree.predict(X)
            error = sample_weight[(tree_pred != y)].sum()
            alpha = np.log((1 - error) / error) / 2
            updated_weights = sample_weight * np.exp(-alpha * y * tree_pred)
            updated_weights /= updated_weights.sum()

            if t < self.n_estimators - 1:
                self.sample_weights[t + 1] = updated_weights

            self.errors[t] = error
            self.trees[t] = tree
            self.alphas[t] = alpha

        return self
```



```python
# CLASSIFICATION
X_train, X_test, y_train, y_test = train_test_split(data_features, data_labels, test_size=0.25, random_state=42)

models = {
    'RandomForestClassifier': RandomForestClassifier(n_estimators=60, random_state=0),
    'GradientBoostingClassifier': GradientBoostingClassifier(random_state=20),
    'CustomAdaBoostClassifier': CustomAdaBoostClassifier()
}

print("\nClassification accuracy results using different techniques\n")

for name, model in models.items():
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    accuracy = accuracy_score(y_test, predictions)
    print(f"{name}: {accuracy * 100:.2f}%")
```



```
Classification accuracy results using different techniques

RandomForestClassifier: 95.07%
GradientBoostingClassifier: 96.48%
CustomAdaBoostClassifier: 61.27%
```

ARTICLE SUMMARY:

Breast Cancer Survivability via AdaBoost Algorithms

Jaree Thongkam, Guandong Xu, Yanchun Zhang and Fuchun Huang
School of Computer Science and Mathematics
Victoria University, Melbourne, Australia
{jaree,xu,yzhang,fuchun}@csm.vu.edu.au

The application of data mining techniques in medical fields is continuously growing. This is mostly due to improvements in the effectiveness of these classification and prediction algorithms, particularly in terms of assisting medical practitioners in their decision-making. This form of study is becoming increasingly significant as a means of improving patient outcomes, lowering medical costs, and advancing clinical investigations.

They propose a data mining pro-processing method and AdaBoost to predict breast cancer survivability in a hospital's databases. They investigate the performance of Adaboost algorithms to gain a better understanding of the relative importance of their features in terms of accuracy,sensitivity and specificity. Moreover, data mining tasks including pre-processing, data transformation, RELIEF attributes selection,confusion matrix and the stratified 10-fold cross validation, are used to prepare the breast cancer input data set.

Adaboost is the most popular ensemble method and has been shown to significantly enhance the prediction accuracy of the base learner. It is a learning algorithm used to generate multiple classifiers and to utilise them to build the best classifier. The advantage of this algorithm is that it requires less input parameters and needs little prior knowledge about the weak learner. Moreover, it has high flexibility suited for combining with other methods for finding weak hypotheses. AdaBoost algorithm is not only used for predicting in classification tasks, but also for presenting self-rated confidence scores which estimate the reliability of their predictions. This algorithm requires user less knowledge of computing in order to improve accuracy of models over datasets. With this method medical practitioners are able to focus on finding weak learning algorithms that only should be better than the original algorithm (weak learner).

Feature selection is an important step in building a classification model. It is advantageous to limit the number of inputs attributes in a classifier in order to have good predictive, and less computationally intensive models.

Confusion matrix is a visualization tool which is commonly used to present the accuracy of the classifiers in classification. It is used to show the relationship between outcomes and predicted classes. The level of efectiveness of the classification model is calculated with the number of correct and incorrect classifications in each possible value of the variables being classified in the confusion matrix.