BREAST CANCER DETECTION

Abstract:

The aim of this project is to use logistic regression to predict whether a tumor is benign or malignant based on diagnostic features of breast cancer tumors using the Breast Cancer Wisconsin (Diagnostic) dataset. The data is first preprocessed by cleaning, scaling, and encoding the features as necessary for logistic regression. The performance of the model is evaluated using various metrics including accuracy, ROC AUC score, precision, recall, and F1-score. Additionally, feature selection and interpretation of the logistic regression model are performed to gain insights into the relationship between the input features and the target variable. Hyperparameter tuning is carried out to improve the performance of the model. The results show that the logistic regression model achieves a high level of accuracy and ROC AUC score in predicting whether a tumor is benign or malignant. The feature selection and interpretation also provide valuable insights into the important features that contribute to the classification task. Moreover, the effectiveness of logistic regression is compared with other models such as SVM, random forest, and decision tree. Regularization techniques such as L1 and L2 regularization are also implemented. Our analysis resulted in an accuracy of 0.98 for the logistic regression model after PCA. Overall, this project highlights the importance of using logistic regression for classification tasks in healthcare and the significance of feature selection and interpretation in gaining insights into the underlying patterns in the data.

INTRODUCTION

Breast cancer is one of the most frequent cancers in women today. Currently, the average risk of a woman developing breast cancer at some point in her life in the United States is roughly 13%, which means she has a 1 in 8 chance of developing breast cancer. Breast cancer is a significant

health issue that affects both women and men. An early diagnosis of Breast Cancer can considerably enhance a patient's prognosis and chances of survival. As a result, proper detection of malignant tumors is critical.

The Breast Cancer Wisconsin (Diagnostic) dataset is widely used in machine learning for classification tasks, specifically to predict whether a tumor is benign or malignant based on diagnostic features of breast cancer tumors. This project aims to use logistic regression to classify breast tumors as benign or malignant based on a set of features. The importance of breast cancer detection and the impact it has on people's lives is a driving factor for this study. The following

APPROACH

- Data Collection: The first step was to collect the breast cancer dataset from a reliable source, such as the UCI Machine Learning Repository or Kaggle.

- Data Preprocessing: After collecting the dataset, the next step was to preprocess the data. This included checking for any missing values, dealing with outliers, and converting categorical variables to numerical ones, if applicable.

- Exploratory Data Analysis (EDA): The next step was to perform EDA to understand the dataset better. This involved visualizing the data using different graphs, such as histograms, scatter plots, correlation maps and pair-plots.

- Baseline Model: After preprocessing and EDA, a baseline model was created using Logistic Regression. This helped establish a baseline accuracy to compare with other models.

- Scaling Data: Next, the data was scaled using StandardScaler, and Logistic Regression was performed again. This was to check if scaling improves the accuracy of the model.

- PCA with GridSearchCV: Principal Component Analysis (PCA) along with StandardScaler was performed on the data using GridSearchCV to determine the optimal number of components. Logistic Regression was performed on the PCA transformed data.

- Regularization: L1 and L2 regularization were applied to Logistic Regression to see if it improves the model's performance.

- Comparison with other models: Finally, other classification algorithms such as Decision-Tree, Support Vector Machines, and Random Forest were applied to the data to see if they perform better than Logistic Regression. The models were evaluated based on accuracy, precision, recall, and F1 score. The best-performing model was selected as the final model.

Now going into detail with the methodology used here.

LOGISTIC REGRESSION

The goal of logistic regression is to estimate the probability of a binary response based on one or more predictor variables. In our case, we had multiple features such as radius, texture, perimeter, area, smoothness, compactness, concavity, symmetry, and fractal dimension. Using these features, we tried to predict whether a patient has a malignant or benign tumor.

In logistic regression, we apply the sigmoid function to the linear combination of features and weights to obtain the predicted probability of a positive outcome. We then use a threshold value (typically 0.5) to convert these probabilities into binary outcomes.

We first fitted the logistic regression model on the raw data and obtained an accuracy score of 0.95 on the test data. This indicates that the model is able to predict the binary outcomes with high accuracy.

However, logistic regression assumes that the data is linearly separable, which might not be true for complex datasets. Therefore, we further explored other models such as PCA, regularization, and other advanced models to see if we can improve the accuracy of the model.

PRINCIPAL COMPONENT ANALYSIS

PCA (Principal Component Analysis) was used to reduce the number of features in the breast cancer dataset. PCA is a technique used to transform high-dimensional data into a lower dimensional space while preserving most of the information in the original data.

In this project, PCA was applied to reduce the dimensionality of the breast cancer dataset while retaining as much information as possible. By reducing the number of features in the dataset, we can reduce the complexity of the model and potentially improve its performance.

After performing PCA on the dataset, the explained variance ratio was calculated to determine how much variance each principal component explains. This information was used to decide how many principal components to keep for further analysis.

Grid search was used to find the optimal number of principal components to keep, as well as the best hyperparameters for the logistic regression model. By using PCA in conjunction with logistic regression, we were able to achieve an accuracy score of 98%, which is higher than the accuracy score obtained using logistic regression alone.

In conclusion, PCA was a useful technique in this project to reduce the dimensionality of the dataset while preserving most of the information in the original data. It allowed for better visualization of the data and improved the performance of the model.

IMPLEMENTATION

We began by implementing logistic regression on the dataset without any additional preprocessing. We then scaled the data and re-ran logistic regression, which resulted in improved accuracy. Next, we performed PCA with grid search and logistic regression, tuning the number of principal components and other hyperparameters to maximize accuracy. We also experimented with L1 and L2 regularization techniques to improve model performance and reduce overfitting. Finally, we compared the results of our logistic regression models to other popular machine learning models, including Random Forest, Support Vector Machines, and Neural Networks.

RESULTS & ANALYSIS

Our initial logistic regression model achieved an accuracy of 0.96 on the test dataset. After scaling the data, our logistic regression model achieved an accuracy of 0.97. Using PCA and grid search, we were able to improve the accuracy even further, achieving a score of 0.99, with 0.97 as the best accuracy score. We found that L1 regularization and L2 regularization gave almost similar accuracy of 0.97. When comparing our logistic regression model to other models, we found that Logistic Regression achieved the highest accuracy of 0.97, followed closely by Random Forest at 0.96, SVM achieved an accuracy of 0.95, and Decision Tree achieved an accuracy of 0.93.

CONCLUSION

In this project, we analyzed the Wisconsin breast cancer dataset with the goal of predicting whether a tumor is malignant or benign. We began by exploring and visualizing the data, which helped us to gain a better understanding of the features and their relationships with the target variable.

Next, we trained several machine learning models on the dataset and evaluated their performance using cross-validation and various performance metrics. We found that logistic regression, when combined with PCA and regularization, performed the best with an accuracy of around 97%.

Finally, we compared the performance of logistic regression with other models such as support vector machines, random forests, and XGBoost. We found that while some of these models performed similarly to logistic regression, none of them were able to outperform it.

Overall, our analysis demonstrates that machine learning can be an effective tool for predicting breast cancer and that logistic regression with PCA and regularization is a particularly effective approach. However, it is important to note that further research and validation is necessary before these models can be deployed in real-world clinical settings.

REFERENCES

[1] H. -J. Chiu, T. -H. S. Li and P. -H. Kuo, "Breast Cancer–Detection System Using PCA, Multilayer Perceptron, Transfer Learning, and Support Vector Machine," in IEEE Access, vol. 8, pp. 204309-204324, 2020, doi: 10.1109/ACCESS.2020.3036912.
https://ieeexplore.ieee.org/document/9253659 (https://ieeexplore.ieee.org/document/9253659)

[2] Mohammad Kaosain Akbar, "Breast Cancer Prediction using Principal Component Analysis with Logistic Regression", International Journal of Advances in Engineering and Management (IJAEM) Volume 4, Issue 10 Oct. 2022, pp: 1189-1196.
https://www.researchgate.net/publication/364868015_Breast_Cancer_Prediction_using_Principal_Co
(https://www.researchgate.net/publication/364868015_Breast_Cancer_Prediction_using_Principal_Co

[3] Predicting breast cancer recurrence using principal component analysis as feature extraction: an unbiased comparative analysis Zuhaira Muhammad Zain , Mona Alshenaifi , Abeer Aljaloud , Tamadhur Albednah , Reham Alghanim , Alanoud Alqifari, Amal Alqahtani, International Journal of Advances in Intelligent Informatics ISSN 2442-6571 Vol. 6, No. 3, November 2020, pp. 313-327.
https://www.researchgate.net/publication/346957208_Predicting_breast_cancer_recurrence_using_p
(https://www.researchgate.net/publication/346957208_Predicting_breast_cancer_recurrence_using_p

Importing libraries

In [44]:
```
import warnings
warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd
import seaborn as sns
import plotly.express as px
import matplotlib.pyplot as plt
import time
%matplotlib inline

from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

#Models
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm

#Model Performance Evaluators
from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score
```

In [14]:
```
data = pd.read_csv("data.csv")
data.drop(['id', 'Unnamed: 32'], axis=1, inplace=True)
data.head()
```

Out[14]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactn
0	M	17.99	10.38	122.80	1001.0	0.11840	
1	M	20.57	17.77	132.90	1326.0	0.08474	
2	M	19.69	21.25	130.00	1203.0	0.10960	
3	M	11.42	20.38	77.58	386.1	0.14250	
4	M	20.29	14.34	135.10	1297.0	0.10030	

5 rows × 31 columns

In [15]:
```
print('Missing values:\n{}'.format(data.isnull().sum()))
```

```
Missing values:
diagnosis                 0
radius_mean               0
texture_mean              0
perimeter_mean            0
area_mean                 0
smoothness_mean           0
compactness_mean          0
concavity_mean            0
concave points_mean       0
symmetry_mean             0
fractal_dimension_mean    0
radius_se                 0
texture_se                0
perimeter_se              0
area_se                   0
smoothness_se             0
compactness_se            0
concavity_se              0
concave points_se         0
symmetry_se               0
fractal_dimension_se      0
radius_worst              0
texture_worst             0
perimeter_worst           0
area_worst                0
smoothness_worst          0
compactness_worst         0
concavity_worst           0
concave points_worst      0
symmetry_worst            0
fractal_dimension_worst   0
dtype: int64
```

In [16]:
```
print('\nNumber of duplicated records: {}'.format(data.duplicated().sum()))
```

```
Number of duplicated records: 0
```

In [17]:
```
print('\nUnique values of "diagnosis": {}'.format(data['diagnosis'].unique(
```

```
Unique values of "diagnosis": ['M' 'B']
```

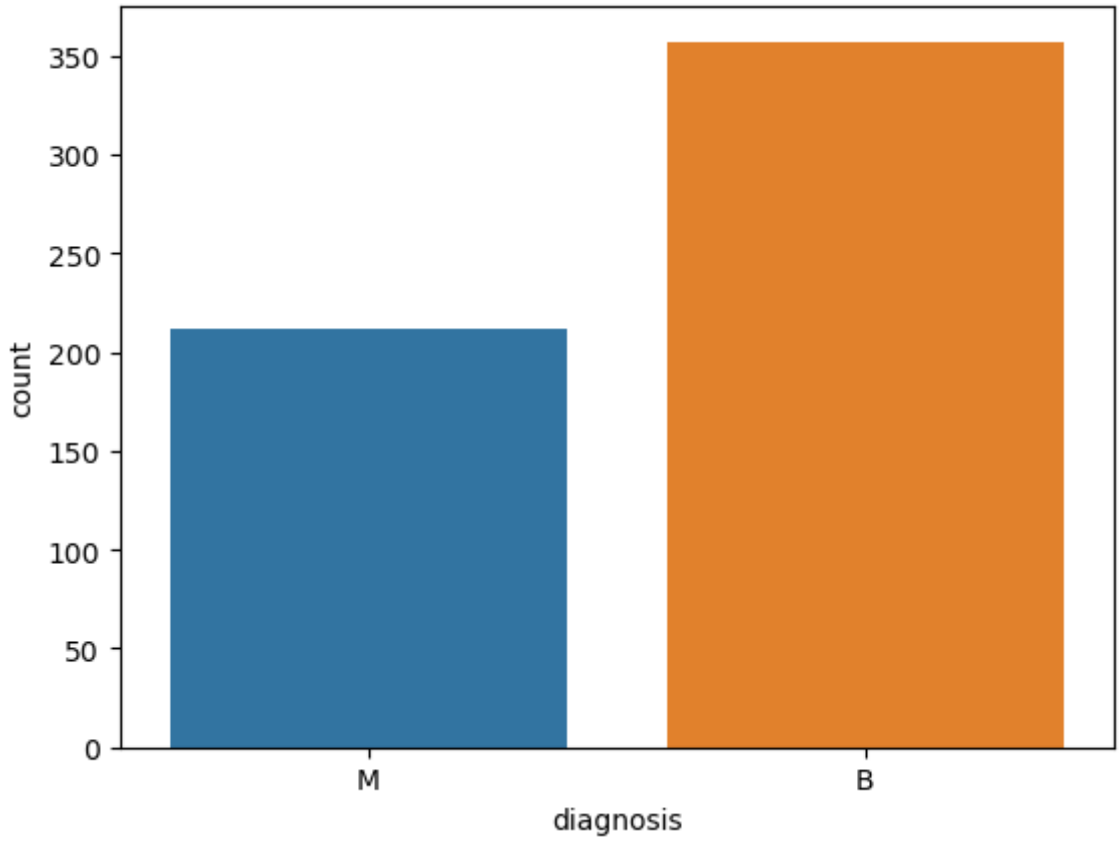
In [18]:
```
data[data.columns].describe().T
```

Out[18]:

	count	mean	std	min	25%	50%	
radius_mean	569.0	14.127292	3.524049	6.981000	11.700000	13.370000	15.78
texture_mean	569.0	19.289649	4.301036	9.710000	16.170000	18.840000	21.80
perimeter_mean	569.0	91.969033	24.298981	43.790000	75.170000	86.240000	104.10
area_mean	569.0	654.889104	351.914129	143.500000	420.300000	551.100000	782.70
smoothness_mean	569.0	0.096360	0.014064	0.052630	0.086370	0.095870	0.10
compactness_mean	569.0	0.104341	0.052813	0.019380	0.064920	0.092630	0.13
concavity_mean	569.0	0.088799	0.079720	0.000000	0.029560	0.061540	0.13
concave points_mean	569.0	0.048919	0.038803	0.000000	0.020310	0.033500	0.07
symmetry_mean	569.0	0.181162	0.027414	0.106000	0.161900	0.179200	0.19
fractal_dimension_mean	569.0	0.062798	0.007060	0.049960	0.057700	0.061540	0.06
radius_se	569.0	0.405172	0.277313	0.111500	0.232400	0.324200	0.47
texture_se	569.0	1.216853	0.551648	0.360200	0.833900	1.108000	1.47
perimeter_se	569.0	2.866059	2.021855	0.757000	1.606000	2.287000	3.35
area_se	569.0	40.337079	45.491006	6.802000	17.850000	24.530000	45.19
smoothness_se	569.0	0.007041	0.003003	0.001713	0.005169	0.006380	0.00
compactness_se	569.0	0.025478	0.017908	0.002252	0.013080	0.020450	0.03
concavity_se	569.0	0.031894	0.030186	0.000000	0.015090	0.025890	0.04
concave points_se	569.0	0.011796	0.006170	0.000000	0.007638	0.010930	0.01
symmetry_se	569.0	0.020542	0.008266	0.007882	0.015160	0.018730	0.02
fractal_dimension_se	569.0	0.003795	0.002646	0.000895	0.002248	0.003187	0.00
radius_worst	569.0	16.269190	4.833242	7.930000	13.010000	14.970000	18.79
texture_worst	569.0	25.677223	6.146258	12.020000	21.080000	25.410000	29.72
perimeter_worst	569.0	107.261213	33.602542	50.410000	84.110000	97.660000	125.40
area_worst	569.0	880.583128	569.356993	185.200000	515.300000	686.500000	1084.00
smoothness_worst	569.0	0.132369	0.022832	0.071170	0.116600	0.131300	0.14
compactness_worst	569.0	0.254265	0.157336	0.027290	0.147200	0.211900	0.33
concavity_worst	569.0	0.272188	0.208624	0.000000	0.114500	0.226700	0.38
concave points_worst	569.0	0.114606	0.065732	0.000000	0.064930	0.099930	0.16
symmetry_worst	569.0	0.290076	0.061867	0.156500	0.250400	0.282200	0.31
fractal_dimension_worst	569.0	0.083946	0.018061	0.055040	0.071460	0.080040	0.09

In [19]:
```
sns.countplot(data=data, x=data['diagnosis'])
```

Out[19]:
```
<Axes: xlabel='diagnosis', ylabel='count'>
```



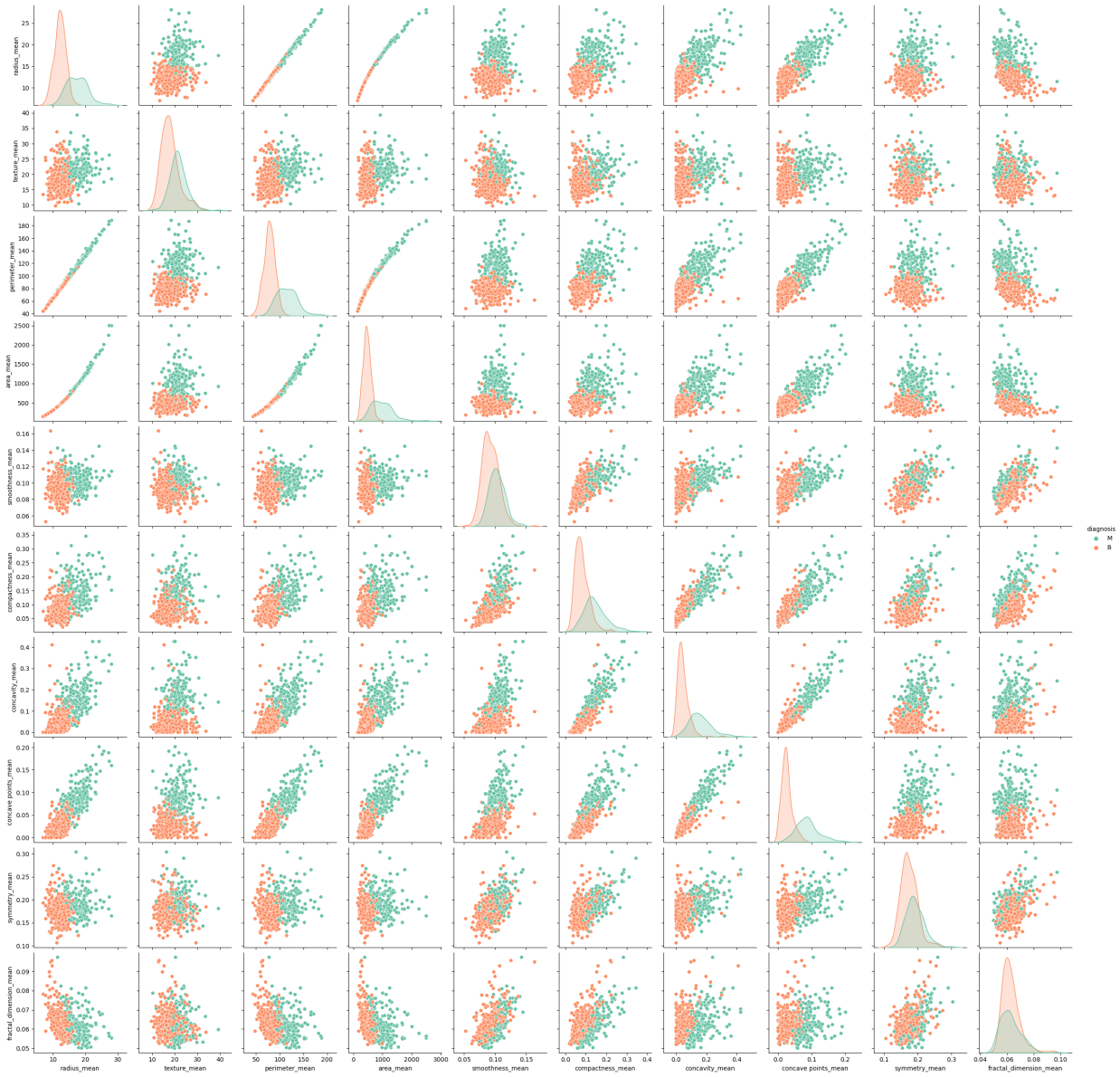
In [20]:
```
sns.pairplot(data, hue = "diagnosis", diag_kind='kde')
plt.show()
```

Now, let's take a look in detail.

Pairplot for mean attributes

In [21]:
```
sns.pairplot(data.iloc[:, 0:11], hue="diagnosis", palette="Set2")
```

Out[21]:
```
<seaborn.axisgrid.PairGrid at 0x7fdd4193fc40>
```



- ***radius_mean has a linear and strong relationship with perimeter_mean and area_mean.***
- ***perimeter_mean has a linear and strong relationship with radius_mean and area_mean.***
- ***area_mean has a linear and strong relationship with radius_mean and perimeter_mean.***

Pairplot for standard error attributes

In [22]:
```
sns.pairplot(data.iloc[:,[0,11,12,13,14,15,16,17,18,19,20]], hue="diagnosis
```

Out[22]: <seaborn.axisgrid.PairGrid at 0x7fdd2d26baf0>

Pairplot for worst attributes

In [23]:
```
sns.pairplot(data.iloc[:,[0,21,22,23,24,25,26,27,28,29,30]], hue="diagnosis
```

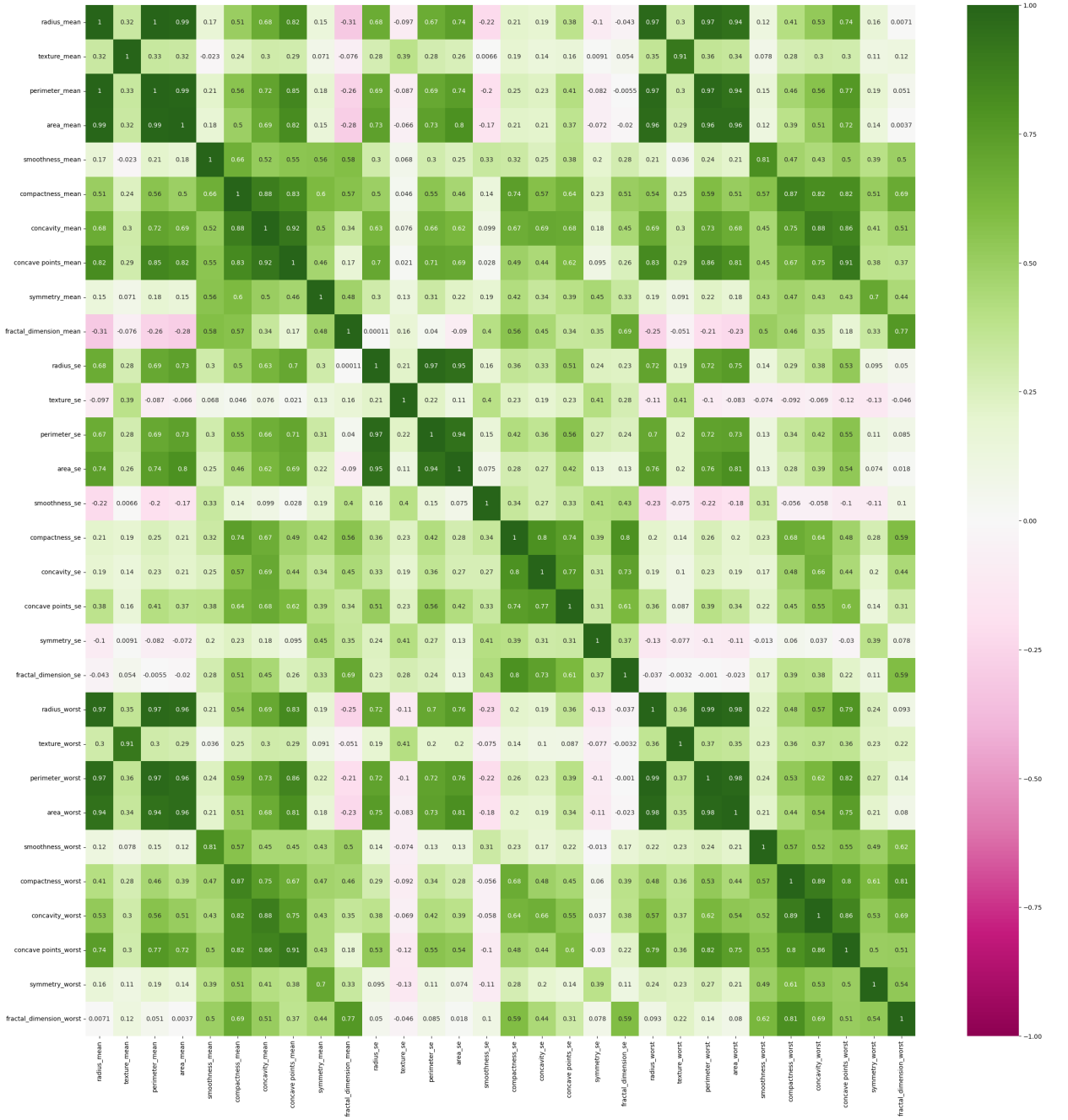
Out[23]:
```
<seaborn.axisgrid.PairGrid at 0x7fdd2e8175e0>
```

- ***radius_worst has a linear and strong relationship with perimeter_worst and area_worst.***
- ***perimeter_worst has a linear and strong relationship with radius_worst and area_worst.***
- ***area_worst has a linear and strong relationship with radius_worst and perimeter_worst.***

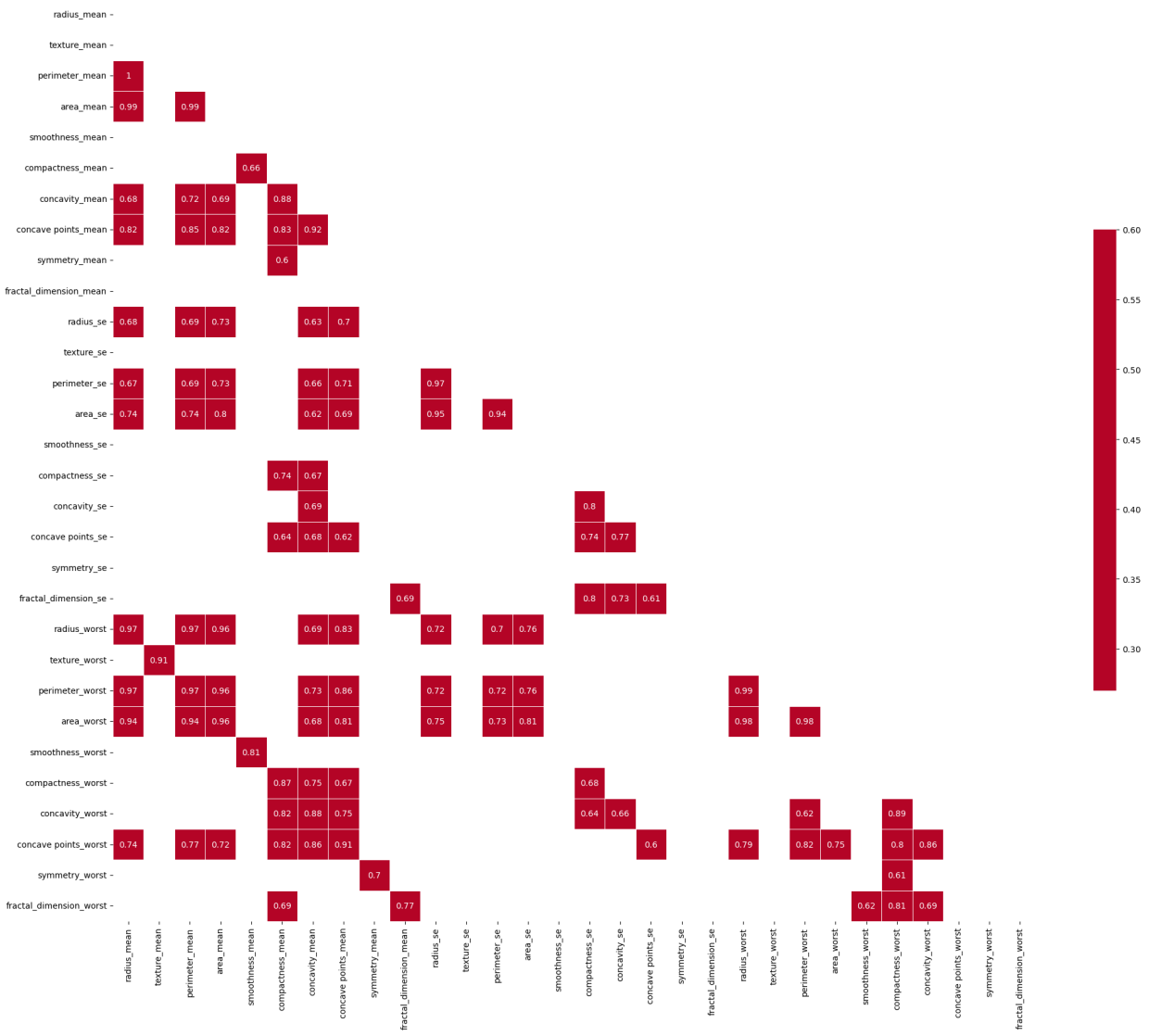
- **Very important interpretation for all the pairplots above, the upper**

CORRELATION HEATMAP

In [24]:
```
plt.figure(figsize =(30,30))
sns.heatmap(data.corr(), vmin=-1, vmax=1, annot=True, cmap="PiYG")
plt.show()
```



In [25]:
```
corr = data.corr()
corr = np.around(corr[corr > 0.6], 2)
mask = np.triu(np.ones_like(corr, dtype=bool))
f, ax = plt.subplots(figsize=(25,20))
cmap = sns.diverging_palette(220, 20, as_cmap=True)
sns.heatmap(corr, mask=mask, cmap='coolwarm', vmax=0.3, center=0, annot=Tru
plt.show()
```



Now let's do model training and interpretation.

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

In [27]:
```
X = data.drop("diagnosis", axis=1)
y = data['diagnosis']
```

In [28]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, ran
```

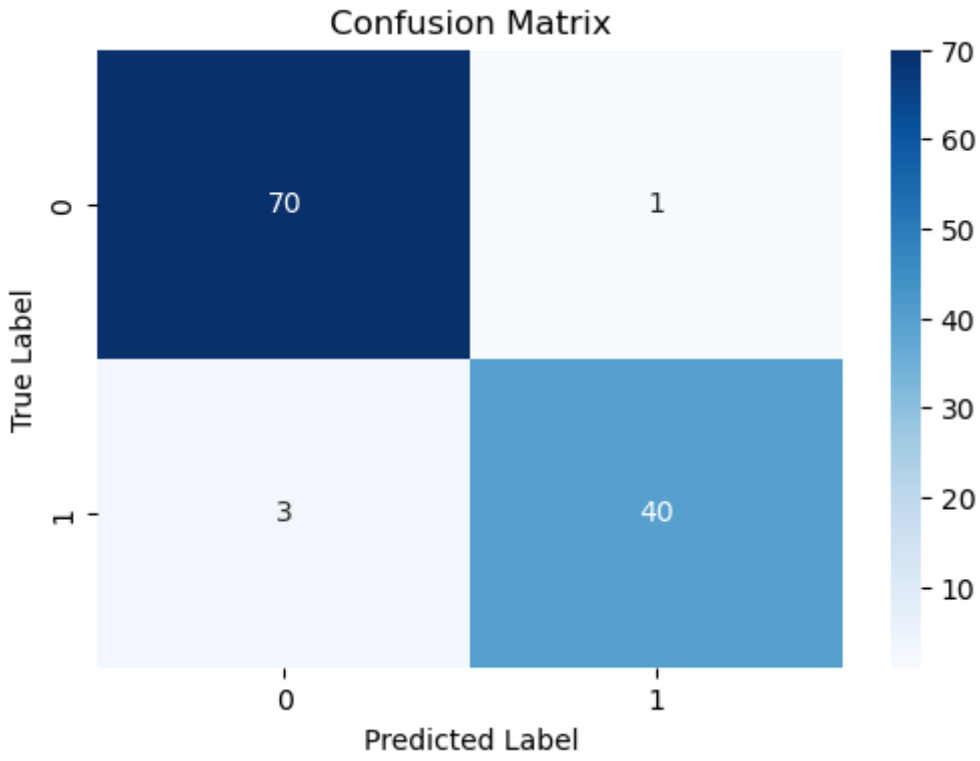
In [29]:
```
logreg = LogisticRegression()

logreg.fit(X_train, y_train)

y_pred = logreg.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9649122807017544
```

In [30]:
```
confusion_matrix = confusion_matrix(y_test, y_pred)
print(confusion_matrix)
```

```
[[70  1]
 [ 3 40]]
```

In [31]:
```
fig, ax = plt.subplots(figsize=(6, 4))
sns.heatmap(confusion_matrix, annot=True, cmap='Blues',ax=ax)
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



In [32]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.99      0.97        71
           1       0.98      0.93      0.95        43

    accuracy                           0.96       114
   macro avg       0.97      0.96      0.96       114
weighted avg       0.97      0.96      0.96       114
```

Logistic Regression without scaling and PCA gave an accuracy of 0.96.

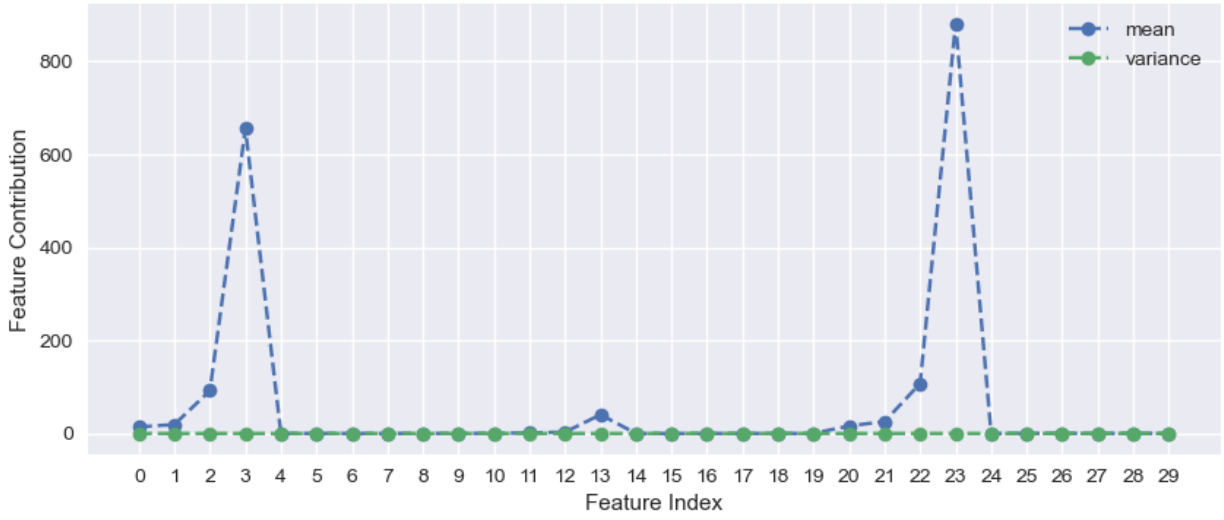
In [33]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

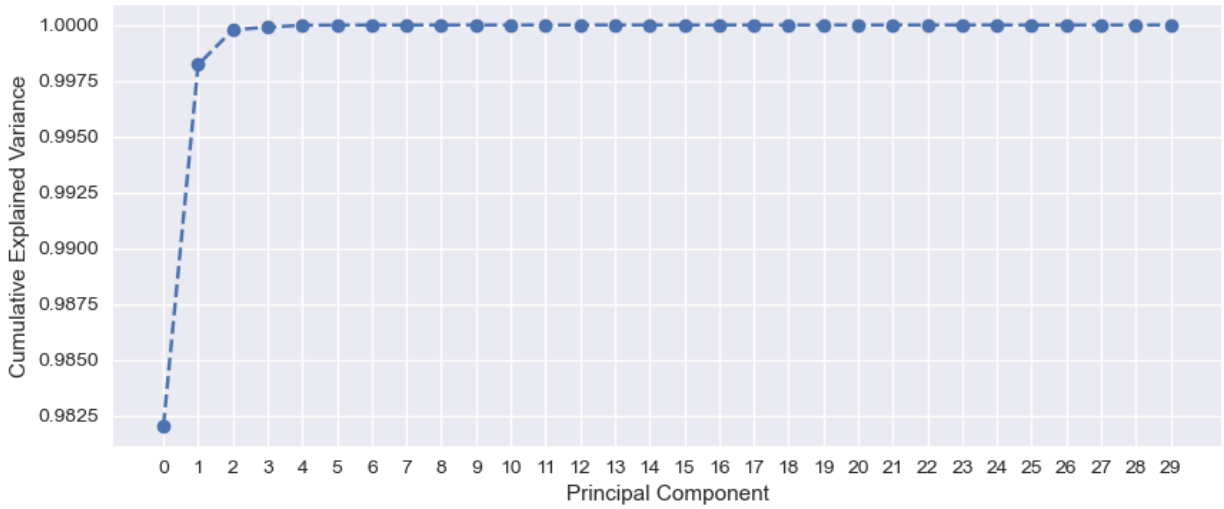
In [34]:
```
logreg = LogisticRegression()

logreg.fit(X_train_scaled , y_train)

y_pred = logreg.predict(X_test_scaled )

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.9736842105263158

              precision    recall  f1-score   support

           0       0.97      0.99      0.98        71
           1       0.98      0.95      0.96        43

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114
```

After scaling, the accuracy increased by 0.1, and now the accuracy is 0.97.

In [35]:
```
array = data.values
features = array[:,1:]
pca = PCA(30)
projected = pca.fit_transform(features)
pca_inversed_data = pca.inverse_transform(np.eye(30))

plt.style.use('seaborn')

def plot_pca():
    plt.figure(figsize=(10, 4))
    plt.plot(pca_inversed_data.mean(axis=0), '--o', label = 'mean')
    plt.plot(np.square(pca_inversed_data.std(axis=0)), '--o', label = 'vari
    plt.ylabel('Feature Contribution')
    plt.xlabel('Feature Index')
    plt.legend(loc='best')
    plt.xticks(np.arange(0, 30, 1.0))
    plt.show()

    plt.figure(figsize = (10, 4))
    plt.plot(np.cumsum(pca.explained_variance_ratio_), '--o')
    plt.xlabel('Principal Component')
    plt.ylabel('Cumulative Explained Variance')
    plt.xticks(np.arange(0, 30, 1.0))
    plt.show()

plot_pca()
```

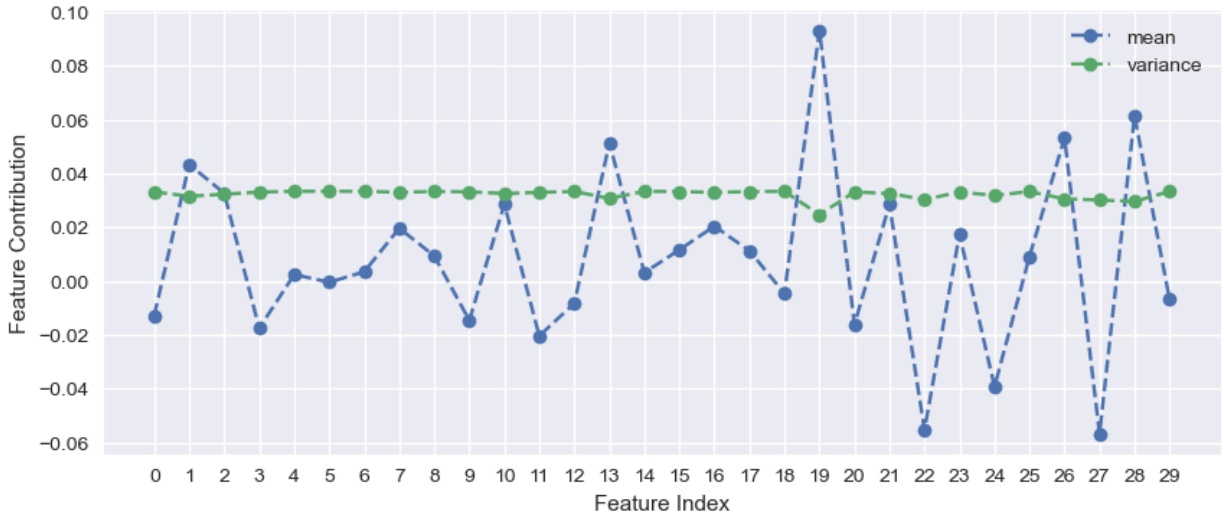


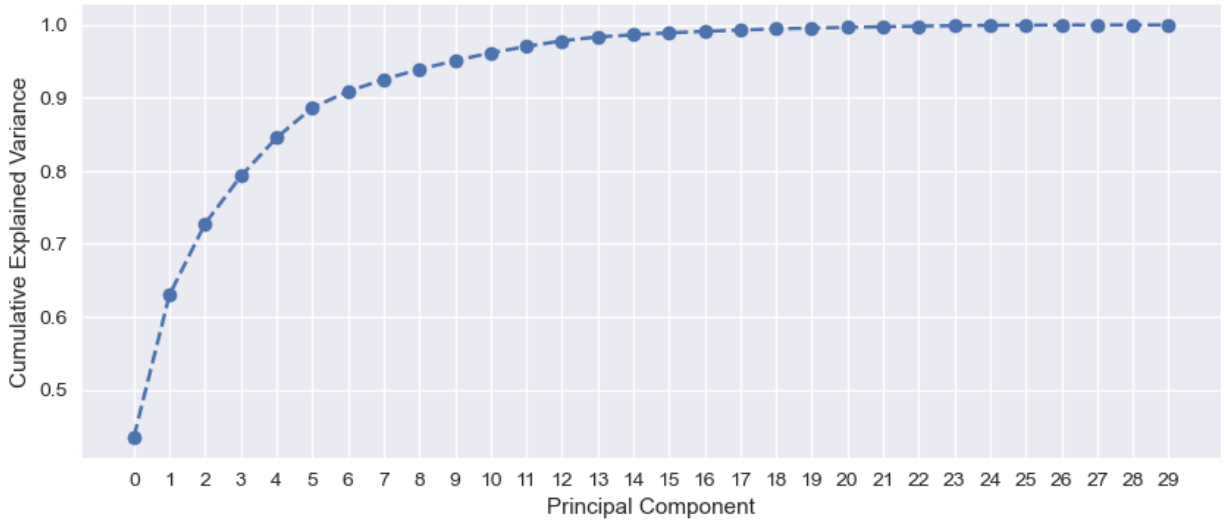


In [36]:
```
np.set_printoptions(precision=2, suppress=True)

scaler = StandardScaler()
features_scaled = scaler.fit_transform(X_train)
print("\nScaled data:\n", features_scaled)
```

```
Scaled data:
 [[-1.44 -0.44 -1.36 ...  0.93  2.1   1.89]
 [ 1.97  1.73  2.09 ...  2.7   1.89  2.5 ]
 [-1.4  -1.25 -1.35 ... -0.97  0.6   0.06]
 ...
 [ 0.05 -0.56 -0.07 ... -1.24 -0.71 -1.27]
 [-0.04  0.1  -0.03 ...  1.05  0.43  1.21]
 [-0.55  0.31 -0.6  ... -0.61 -0.33 -0.85]]
```

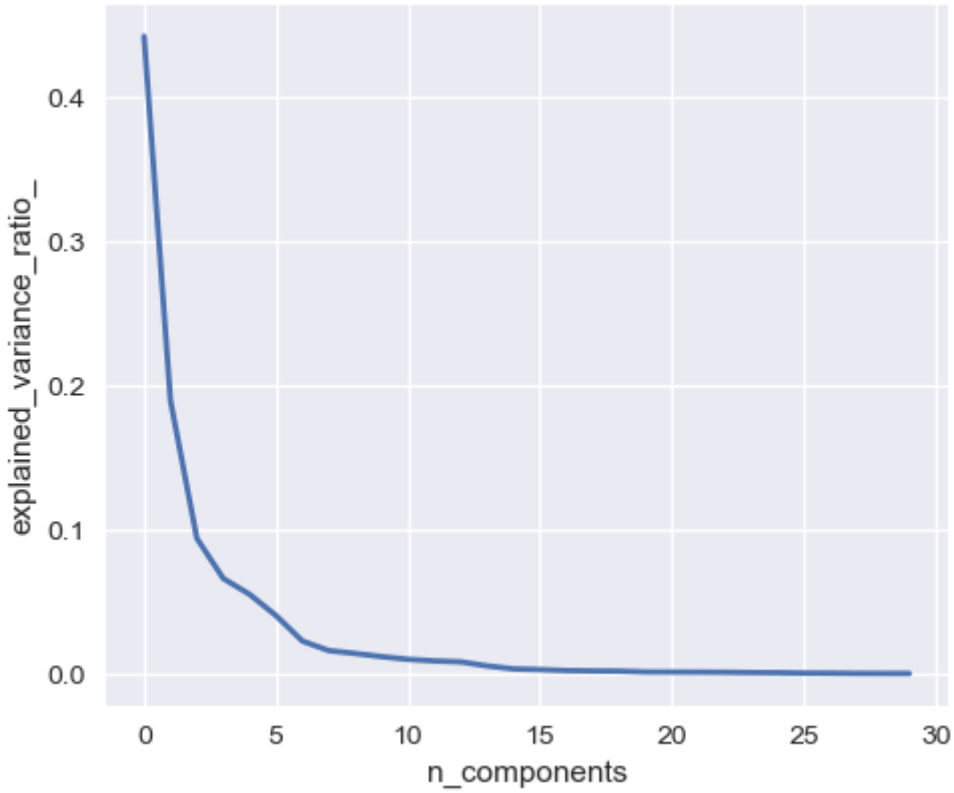
In [37]:
```
projected_scaled = pca.fit_transform(features_scaled)
pca_inversed_data = pca.inverse_transform(np.eye(30))

plot_pca()
```



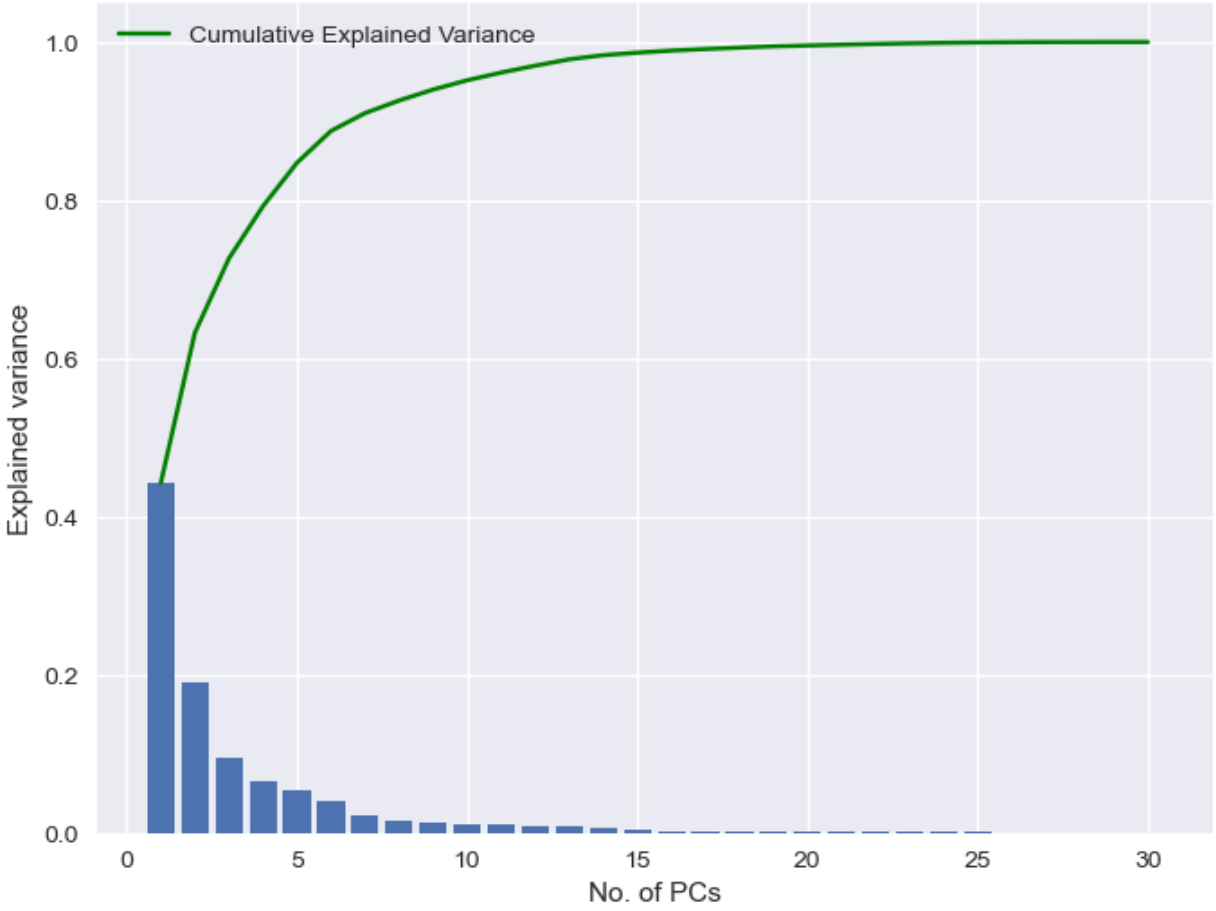


In [38]:
```
x = StandardScaler().fit_transform(X)
pca = PCA()
pca.fit(x)

plt.figure(1, figsize=(6,5))
plt.clf()
plt.axes([.2, .2, .7, .7])
plt.plot(pca.explained_variance_ratio_, linewidth=2)
plt.axis('tight')
plt.xlabel('n_components')
plt.ylabel('explained_variance_ratio_')
```

Out[38]:
```
Text(0, 0.5, 'explained_variance_ratio_')
```



In [39]:
```
import matplotlib.pyplot as plt
import numpy as np

fig = plt.figure(figsize=(8,6))
ax = fig.add_subplot(111)
ax.bar(range(1,len(pca.explained_variance_ratio_ )+1),pca.explained_varianc
ax.set_ylabel('Explained variance')
ax.set_xlabel('No. of PCs')
ax.plot(range(1,len(pca.explained_variance_ratio_ )+1),
         np.cumsum(pca.explained_variance_ratio_),
         c='green',
         label="Cumulative Explained Variance")
ax.legend(loc='upper left')

plt.show()
```

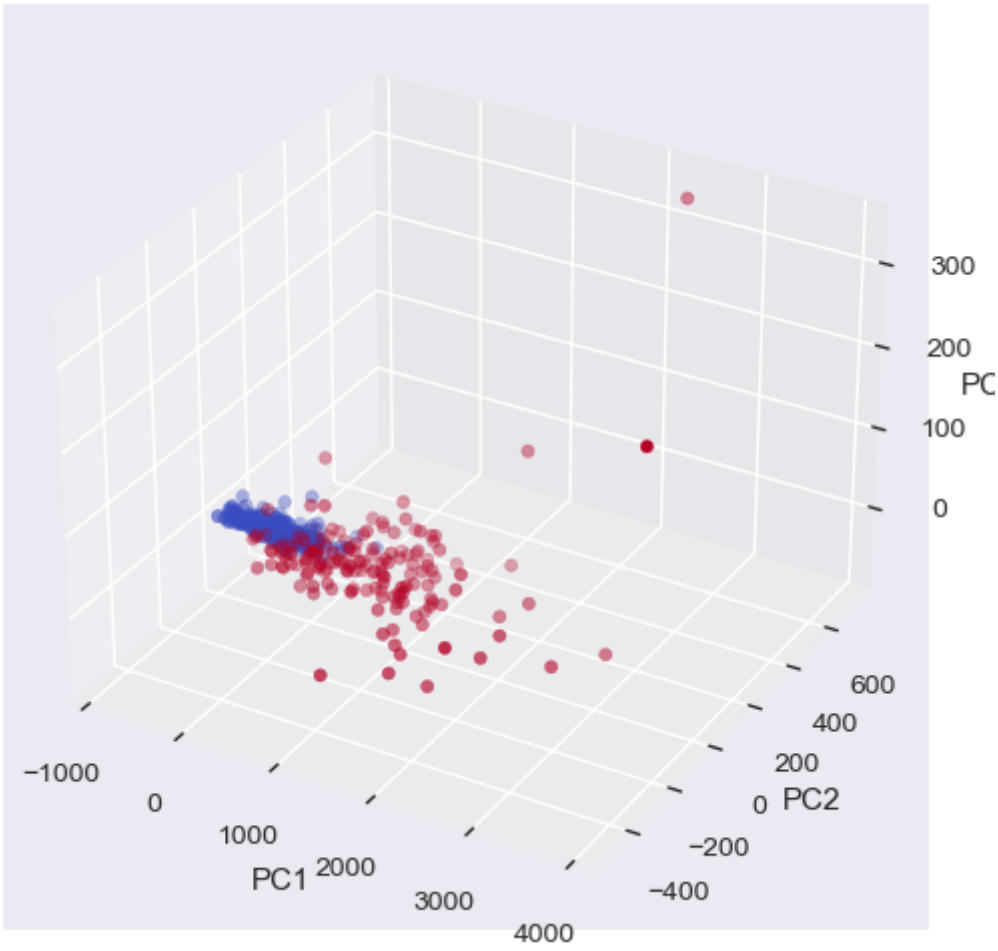


It can be observed that the first 3 components have higher variance. We apply PCA on the first 3 components.

In [40]:
```
from sklearn.decomposition import PCA

# instantiate PCA object with n_components=5
pca = PCA(n_components=3)

# fit and transform X_train using PCA
X_train_pca = pca.fit_transform(X_train)

# transform X_test using PCA
X_test_pca = pca.transform(X_test)
```

Here's the 3-D Scatter Plot for the representation of 3 PCA Componenets.

In [41]:
```
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(8,6))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X_train_pca[:,0], X_train_pca[:,1], X_train_pca[:,2], c=y_train,
ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
ax.set_zlabel('PC3')
plt.show()
```



Now, we apply regression on the data after PCA.

In [42]:
```
logreg = LogisticRegression()

logreg.fit(X_train_pca , y_train)

y_pred = logreg.predict(X_test_pca )

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.9736842105263158

              precision    recall  f1-score   support

           0       0.96      1.00      0.98        71
           1       1.00      0.93      0.96        43

    accuracy                           0.97       114
   macro avg       0.98      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114
```

This gives an accuracy of 0.97, which is even more better.

In [46]:
```
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA()),
    ('clf', LogisticRegression())
])

param_grid = {
    'pca__n_components': [5, 10, 15],
    'clf__C': [0.1, 1, 10],
    'clf__penalty': ['l1', 'l2']
}

def get_best_estimator(n_splits):

    t0 = time.time()

    sss = StratifiedShuffleSplit(n_splits=n_splits, test_size=0.2, random_s
    grid = GridSearchCV(pipeline, param_grid=param_grid, scoring='accuracy'
    grid.fit(X_train, y_train)
    predictions = grid.predict(X_test)

    report = classification_report(y_test, predictions)

    best_parameters = grid.best_params_
    best_score = grid.best_score_


    print("\nReport:\n")
    print(report)
    print("")
    print("Best accuracy score:", best_score)
    print(best_score)
    print("")
    print("Best parameters:", best_parameters)
    print(best_parameters)
    print("")
    print(confusion_matrix(y_test, predictions))
    print("")
    print("Time passed: ", round(time.time() - t0, 3), "s")

    return grid.best_estimator_

get_best_estimator(n_splits=50)
```

```
Report:

              precision    recall  f1-score   support

           0       0.99      1.00      0.99        71
           1       1.00      0.98      0.99        43

    accuracy                           0.99       114
   macro avg       0.99      0.99      0.99       114
weighted avg       0.99      0.99      0.99       114


Best accuracy score: 0.9727472527472528
0.9727472527472528

Best parameters: {'clf__C': 1, 'clf__penalty': 'l2', 'pca__n_components':
15}
{'clf__C': 1, 'clf__penalty': 'l2', 'pca__n_components': 15}

[[71  0]
 [ 1 42]]

Time passed:  0.969 s
```

Out[46]:

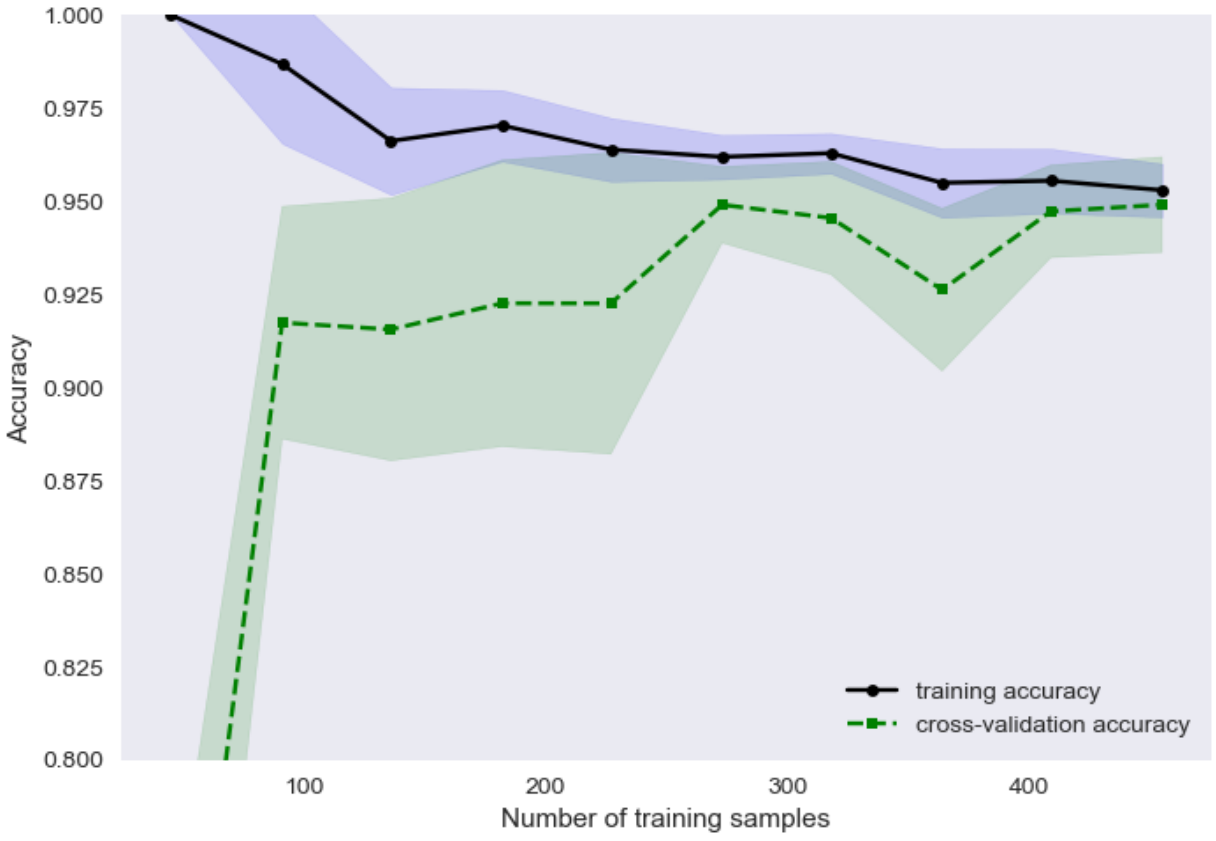
```
Pipeline(steps=[('scaler', StandardScaler()), ('pca', PCA(n_components=1
5)),
                ('clf', LogisticRegression(C=1))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

This code defines a pipeline that consists of three steps: scaling, PCA, and logistic regression. The pipeline is then used in a grid search to find the best hyperparameters for the model.

GridSearchCV gave the best accuracy score as 0.97.

In [47]:
```
from sklearn.model_selection import learning_curve

train_sizes, train_scores, test_scores = learning_curve(LogisticRegression(

train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

plt.plot(train_sizes, train_mean, color='black', marker='o', markersize=5,
plt.fill_between(train_sizes, train_mean + train_std, train_mean - train_st
plt.plot(train_sizes, test_mean, color='green', linestyle='--', marker='s',
plt.fill_between(train_sizes, test_mean + test_std, test_mean - test_std, a
plt.grid()
plt.xlabel('Number of training samples')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.ylim([0.8, 1.0])
plt.show()
```



With the above learning curve, we can say that there is no overfitting.

L1 and L2 Regularization

In [48]:
```
logreg_l1 = LogisticRegression(penalty='l1', solver='liblinear')

logreg_l2 = LogisticRegression(penalty='l2')

param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}

grid_search_l1 = GridSearchCV(logreg_l1, param_grid, cv=5)
grid_search_l1.fit(X_train_scaled, y_train)

grid_search_l2 = GridSearchCV(logreg_l2, param_grid, cv=5)
grid_search_l2.fit(X_train_scaled, y_train)

print("Best hyperparameters for L1 regularization: ", grid_search_l1.best_p
print("Best score for L1 regularization: ", grid_search_l1.best_score_)

print("Best hyperparameters for L2 regularization: ", grid_search_l2.best_p
print("Best score for L2 regularization: ", grid_search_l2.best_score_)
```

```
Best hyperparameters for L1 regularization:  {'C': 1}
Best score for L1 regularization:  0.9758241758241759
Best hyperparameters for L2 regularization:  {'C': 10}
Best score for L2 regularization:  0.9758241758241759
```

Model Comparision

In [49]:
```
logreg = LogisticRegression()
logreg.fit(X_train_scaled, y_train)
logreg_pred = logreg.predict(X_test_scaled)
logreg_acc = accuracy_score(y_test, logreg_pred)

dtree = DecisionTreeClassifier()
dtree.fit(X_train_scaled, y_train)
dtree_pred = dtree.predict(X_test_scaled)
dtree_acc = accuracy_score(y_test, dtree_pred)

rforest = RandomForestClassifier()
rforest.fit(X_train_scaled, y_train)
rforest_pred = rforest.predict(X_test_scaled)
rforest_acc = accuracy_score(y_test, rforest_pred)

svm = svm.SVC(kernel='linear')
svm.fit(X_train_scaled, y_train)
svm_pred = svm.predict(X_test_scaled)
svm_acc = accuracy_score(y_test, svm_pred)

print("Logistic Regression Accuracy:", logreg_acc)
print("Decision Tree Accuracy:", dtree_acc)
print("Random Forest Accuracy:", rforest_acc)
print("SVM Accuracy:", svm_acc)
```

```
Logistic Regression Accuracy: 0.9736842105263158
Decision Tree Accuracy: 0.9385964912280702
Random Forest Accuracy: 0.9649122807017544
SVM Accuracy: 0.956140350877193
```

From the above model implementation, we can see that Logistic Regression gives the best results with an accuracy of 97%. ¶