BREAST CANCER DATA ANALYSIS OUTLINE

Chenyue Yang | Github: anny19951126anny

DATASET DESCRIPTION:

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. n the 3-dimensional space is that described in: [K. P. Bennett and O. L. Mangasarian: "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets", Optimization Methods and Software 1, 1992, 23-34].

Attribute Information:

1) ID number 2) Diagnosis (M = malignant, B = benign) 3-32)

Ten real-valued features are computed for each cell nucleus:

a) radius (mean of distances from center to points on the perimeter) b) texture (standard deviation of gray-scale values) c) perimeter d) area e) smoothness (local variation in radius lengths) f) compactness (perimeter^2 / area - 1.0) g) concavity (severity of concave portions of the contour) h) concave points (# of concave portions of the contour) i) symmetry j) fractal dimension ("coastline approximation" - 1)

The **mean**, **standard** error and "**worst**" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For instance, field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius.

Class distribution: 357 benign, 212 malignant

DATA PREPROCESSING:

1. **Import dataset, reshape into desirable format, and inspect dataset**:

   Dataset inspection:

```
# Load data:
data = pd.read_csv('breast_cancer_wisconsin-data.csv')
data.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	ra
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	

5 rows × 32 columns

Dataset descriptive statistics:

```
# Look at distributions of data:
data.describe()
```

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200

8 rows × 31 columns

- The ranges of values for each variable are very different, may need standardization later.

2. Define feature and outcome variables:

```
# Feature matrix:
X = data[data.columns[1:]]
X.head()
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dim
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	

5 rows × 30 columns

```
# Outcome variable: cancerous or not
y = data['diagnosis']
y.head()
```

```
0    M
1    M
2    M
3    M
4    M
Name: diagnosis, dtype: object
```

- Total of 30 features (dropped id); diagnosis is used as a binary outcome (Benign vs. Malignant)

3. Split dataset into train and test:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)
y_train_binary = y_train
y_test_binary = y_test
```

```
y_train_binary = y_train_binary.replace(['M'],1)
y_train_binary = y_train_binary.replace(['B'],0)
y_test_binary = y_test_binary.replace(['M'],1)
y_test_binary = y_test_binary.replace(['B'],0)
y_test_binary.head()
```

```
86     1
19     0
214    1
229    1
20     0
Name: diagnosis, dtype: int64
```

```
print(np.shape(X_train))
print(np.shape(X_test))
print(len(y_train))
print(len(y_test))
```

```
(398, 30)
(171, 30)
398
171
```

4. Assess (and impute) missing values, and remove any bad columns:

```
# Look at missing values for each variable:
print(data.isnull().sum())
```

```
diagnosis                0
radius_mean              0
texture_mean             0
perimeter_mean           0
area_mean                0
```

- *There's no missing values in dataset, do not need missing imputations*

5. Standardize dataset, train, and test data:

Standardize whole dataset:

Range of each variable varies a lot, could be a problem for distance/variance based methods.

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
data_scaled = data.iloc[:,1:] # exclude diagnosis
scaled_values = scaler.fit_transform(data_scaled)
data_scaled = pd.DataFrame(scaled_values, columns = feature_names[1:])
data_scaled['diagnosis'] = data.iloc[:, 0]
data_scaled.head()
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dim
0	1.097064	-2.073335	1.269934	0.984375	1.568466	3.283515	2.652874	2.532475	2.217515	
1	1.829821	-0.353632	1.685955	1.908708	-0.826962	-0.487072	-0.023846	0.548144	0.001392	
2	1.579888	0.456187	1.566503	1.558884	0.942210	1.052926	1.363478	2.037231	0.939685	
3	-0.768909	0.253732	-0.592687	-0.764464	3.283553	3.402909	1.915897	1.451707	2.867383	
4	1.750297	-1.151816	1.776573	1.826229	0.280372	0.539340	1.371011	1.428493	-0.009560	

5 rows × 31 columns

Standardize train and test:

```
scaler = StandardScaler()
X_train_scaled_values = scaler.fit_transform(X_train)
X_test_scaled_values = scaler.transform(X_test)
```

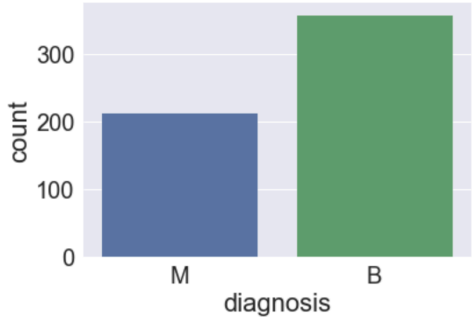
```
X_train_scaled = pd.DataFrame(X_train_scaled_values, columns = feature_names[1:])
X_test_scaled = pd.DataFrame(X_test_scaled_values, columns = feature_names[1:])
```

EXPLORATORY DATA ANALYSIS & FEATURE SELECTION:

1. Visualize data:

Count plot is used to inspect distribution of benign vs. malignant:

```
# Counts for malignant and benign:
ax = sns.countplot(x = 'diagnosis', data=data)
```

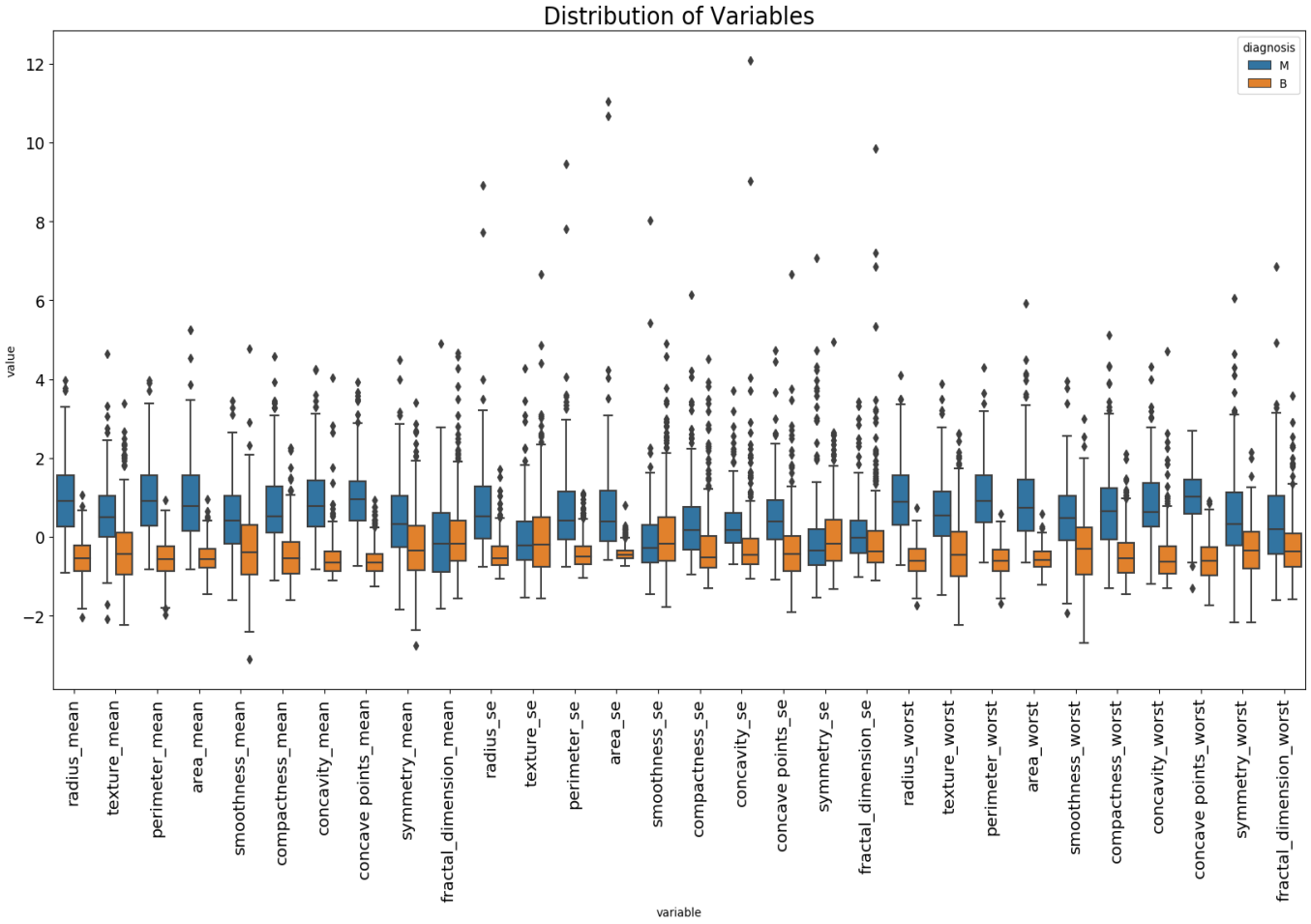


Boxplot showing distribution of each predictor variable:

```
# Transform dataset for boxplot:
data_scaled_melt = data_scaled.melt(id_vars = ['diagnosis'])
data_scaled_melt.head()
```

	diagnosis	variable	value
0	M	radius_mean	1.097064
1	M	radius_mean	1.829821
2	M	radius_mean	1.579888
3	M	radius_mean	-0.768909
4	M	radius_mean	1.750297

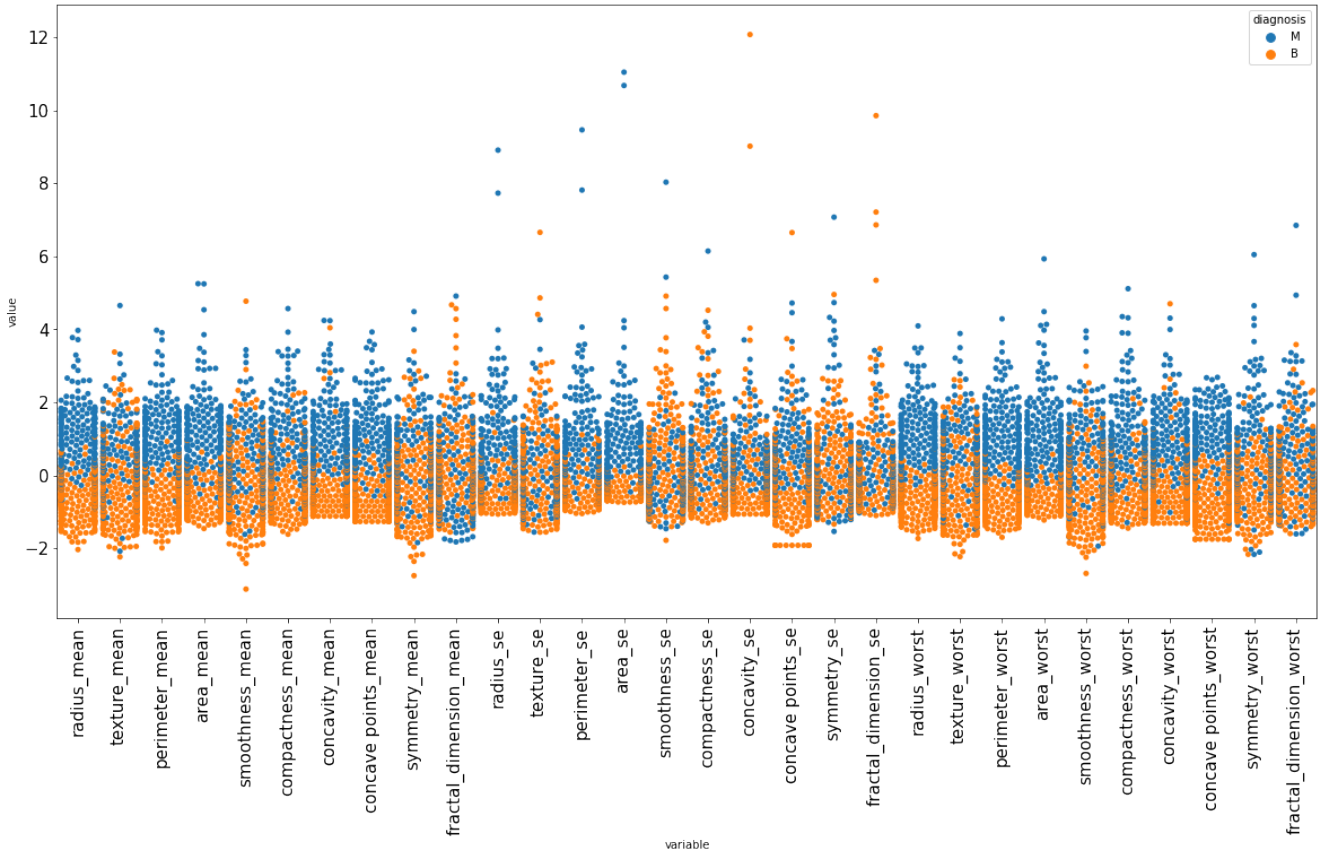
```
# Boxplot to look at distribution of each variable:
plt.figure(figsize = (20,10))
ax = sns.boxplot(x = 'variable', y = 'value', hue = 'diagnosis', data = data_scaled_melt)
plt.title('Distribution of Variables', fontsize = 20)
plt.xticks(rotation=90)
ax.tick_params(labelsize = 14)
```



- Lots of outliers for each variable.
- By inspection, mean value of variables for benign samples are lower than for malignant samples.

Swarmplot is used to get a better idea of how much each variable is separated by benign vs. malignant. Variables that seem to be more associated with the separation could be included in classification model:

```
# Swarmplot to look at separation:
plt.figure(figsize = (20,10))
ax = sns.swarmplot(x = 'variable', y = 'value', hue = 'diagnosis', data = data_scaled_melt)
plt.xticks(rotation=90)
ax.tick_params(labelsize = 15)
```



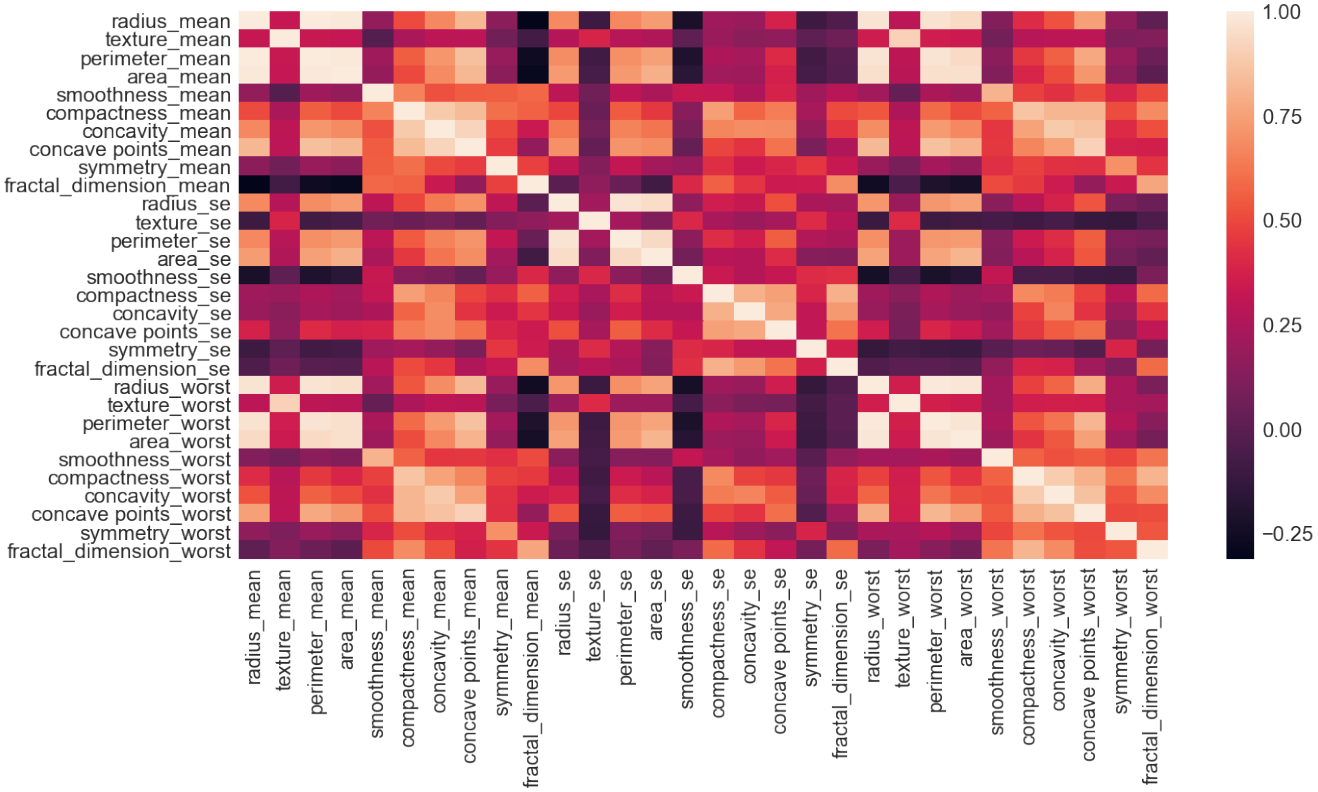
- Variables where B vs. M are relatively well-separated visually:
  {radius_mean, perimeter_mean, area_mean, compactness_mean, concavity_mean, concave points_mean, radius_se, perimeter_se, area_se, radius_worst, perimeter_worst, area_worst, concave points_worst}
- Could consider including these variables as features for classification model

2. Assess and visualize multicollinearity between each variable through correlation matrix and heat map:

Correlation matrix:

```
# Calculate correlation between each variable:
corr = data_scaled.corr()
corr
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	s
radius_mean	1.000000	0.323782	0.997855	0.987357	0.170581	0.506124	0.676764	0.822529	
texture_mean	0.323782	1.000000	0.329533	0.321086	-0.023389	0.236702	0.302418	0.293464	
perimeter_mean	0.997855	0.329533	1.000000	0.986507	0.207278	0.556936	0.716136	0.850977	
area_mean	0.987357	0.321086	0.986507	1.000000	0.177028	0.498502	0.685983	0.823269	
smoothness_mean	0.170581	-0.023389	0.207278	0.177028	1.000000	0.659123	0.521984	0.553695	
compactness_mean	0.506124	0.236702	0.556936	0.498502	0.659123	1.000000	0.883121	0.831135	
concavity_mean	0.676764	0.302418	0.716136	0.685983	0.521984	0.883121	1.000000	0.921391	
concave points_mean	0.822529	0.293464	0.850977	0.823269	0.553695	0.831135	0.921391	1.000000	



Highly correlated features groups:

- *{radius_mean, perimeter_mean, area_mean, radius_worst, perimeter_worst, area_worst, concave points_mean, concave points_worst} / {texture_mean, texture_worst} / {smoothness_mean, smoothness_worst} / {compactness_mean, concavity_mean, compactness_worst, concavity_worst, perimeter_worst, radius_worst } / {area_se, area_worst}*
- Lots of multicollinearity exist in dataset, could use various feature selection methods and PCA to address such problem.

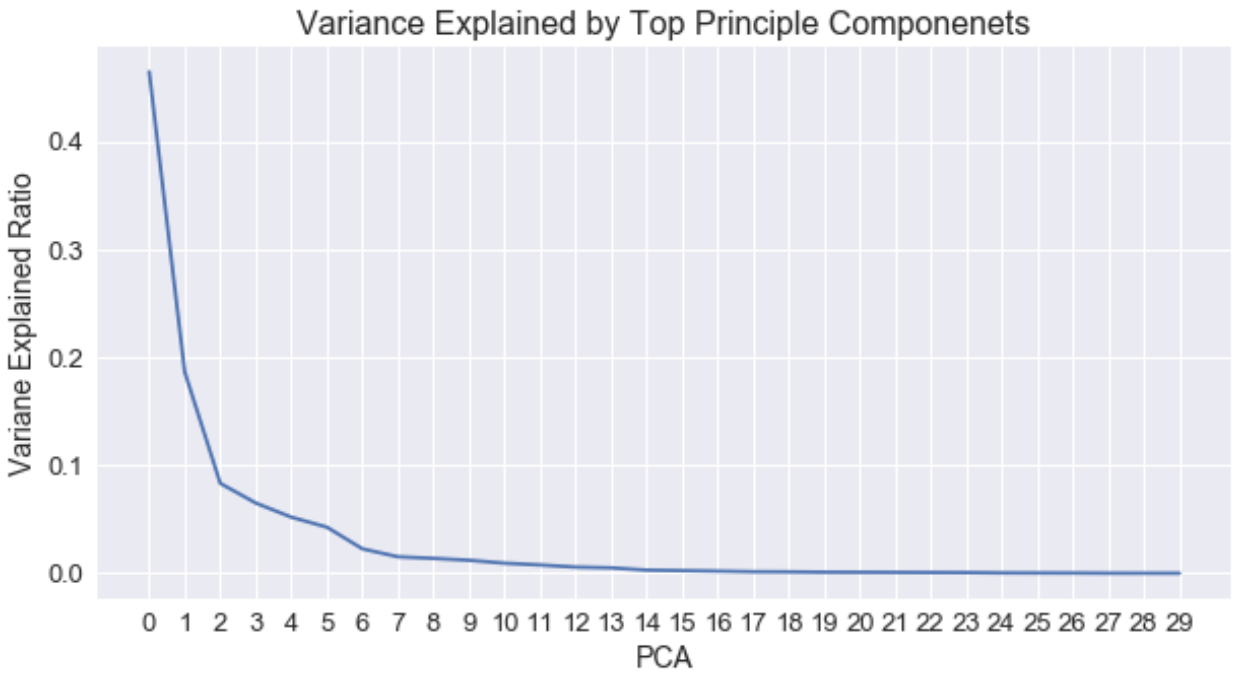
3. **PCA to reduce dimensionality and multicollinearity and get a benchmark for expected performance of predictive models:**

Use percentage of variance explained to determine optimal number of PCs. This gives an idea of how many features to include in classification models. Look at which variables weight more in each PC that have most explained variance.

```
# Apply PCA on training data:
pca = PCA()
pca.fit_transform(X_train_scaled)
pca.explained_variance_ratio_
```

```
array([4.64322848e-01, 1.86481934e-01, 8.32744230e-02, 6.50201310e-02,
       5.19164578e-02, 4.27008587e-02, 2.26193624e-02, 1.52432565e-02,
       1.37535560e-02, 1.20381585e-02, 9.29799403e-03, 7.69104319e-03,
       5.70072886e-03, 5.01878581e-03, 2.89160339e-03, 2.43384644e-03,
       1.97096217e-03, 1.40590288e-03, 1.28841255e-03, 9.75016361e-04,
       9.05993726e-04, 8.22212933e-04, 6.96247936e-04, 6.33813693e-04,
       3.38037000e-04, 2.62923525e-04, 2.22026065e-04, 4.82961388e-05,
       2.06268856e-05, 4.53974495e-06])
```

```
# Plot percentage of variance explained by each PC:
plt.figure(figsize = (10,5))
pcs = np.arange(pca.n_components_)
plt.plot(pcs, pca.explained_variance_ratio_)
plt.xticks(pcs, fontsize = 13)
plt.yticks(fontsize = 13)
plt.xlabel('PCA', fontsize = 14)
plt.ylabel('Variane Explained Ratio', fontsize = 14)
plt.title('Variance Explained by Top Principle Componenets', fontsize = 16)
```



- Significant decrease of variance explained ratio at PC = 2, PC = 6, (and PC = 14?).
- Could use the number of PCs as a reference of how many features to include in classification model.

Look at which original variable weight more in each PC:

```
# Look at each PC's components as the original variables:
varaince_original = pd.DataFrame(pca.components_, columns = feature_names[1:])

# Look at components of first 6 PCs:
varaince_original.T.iloc[:, :6]
```

	0	1	2	3	4	5
radius_mean	0.212611	-0.234236	-0.027340	-0.053168	0.022213	0.030838
texture_mean	0.110031	-0.082318	0.080673	0.581345	-0.036211	-0.058904
perimeter_mean	0.221448	-0.214748	-0.026328	-0.054067	0.023984	0.029956
area_mean	0.214746	-0.233301	0.010870	-0.061555	0.007325	0.001906
smoothness_mean	0.144936	0.197338	-0.083209	-0.175531	-0.343793	-0.300110
compactness_mean	0.235223	0.158619	-0.068974	-0.025977	0.019092	-0.012501
concavity_mean	0.255338	0.040808	-0.007717	-0.027330	0.058538	0.007276
concave points_mean	0.253547	-0.030277	-0.023606	-0.084466	-0.055764	-0.045386
symmetry_mean	0.142611	0.190705	-0.019018	-0.055765	-0.330915	0.322569
fractal_dimension_mean	0.066869	0.367378	-0.030096	-0.020253	-0.030806	-0.182456
radius_se	0.204555	-0.114653	0.268577	-0.090422	-0.119716	-0.057150
texture_se	0.016043	0.067991	0.414523	0.354923	-0.153939	-0.100894
perimeter_se	0.213197	-0.089549	0.270789	-0.085056	-0.056663	-0.018511
area_se	0.201866	-0.157780	0.211156	-0.091449	-0.082599	-0.063707
smoothness_se	0.025456	0.215892	0.363508	-0.101277	-0.164637	-0.302903
compactness_se	0.172420	0.222011	0.151282	0.031535	0.318439	0.123519
concavity_se	0.184200	0.162257	0.144052	-0.012408	0.340380	0.131925
concave points_se	0.185979	0.102100	0.201190	-0.147481	0.174735	0.034174
symmetry_se	0.040819	0.186473	0.305752	-0.007902	-0.270671	0.490778
fractal_dimension_se	0.110938	0.278855	0.171441	0.007094	0.292893	-0.074249
radius_worst	0.222308	-0.223088	-0.053068	-0.018846	-0.026962	-0.011016
texture_worst	0.113202	-0.062689	-0.013309	0.617009	-0.091744	-0.091500
perimeter_worst	0.231370	-0.202099	-0.050294	-0.018636	-0.006599	0.002272
area_worst	0.218970	-0.224973	-0.011074	-0.024834	-0.039224	-0.044958
smoothness_worst	0.142049	0.179282	-0.223890	-0.035534	-0.317441	-0.365125
compactness_worst	0.208787	0.151996	-0.223940	0.119799	0.141015	0.055600
concavity_worst	0.230417	0.082357	-0.180035	0.078291	0.165911	0.045461
concave points_worst	0.245959	-0.005760	-0.172318	-0.016429	0.014797	-0.012639
symmetry_worst	0.127223	0.150792	-0.231004	0.086468	-0.326320	0.463062
fractal_dimension_worst	0.135927	0.271983	-0.236584	0.122840	0.119991	-0.107790

Look at which variable(s) have more impact in each PC. Could consider including these variables in the model:

- PC1: concavity_mean, concavity points_mean
- PC2: fractal_dimension_mean, fractal_dimension_worst
- PC3: texture_se, smoothness_se
- PC4: texture_mean, texture_worst
- PC5: smoothness_mean, concavity_se
- PC6: symmetry_se, symmetry_worst

Dimensionally reduce data to 2D to:

- *Visualize data points*
- *To identify any sample subgroups*
- *To determine if clinical outcome associated with any sample subgroups (i.e. colors separated by each subgroup if any)*

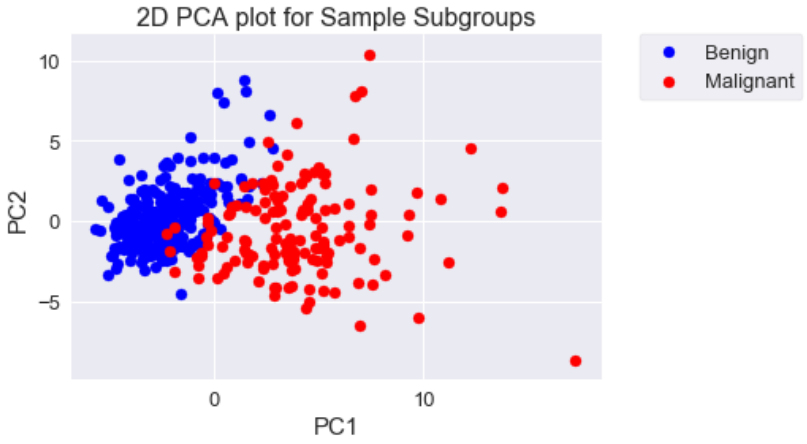
```
pc_2D = PCA(n_components = 2)
X_train_2D = pc_2D.fit_transform(X_train_scaled)
```

```
df_train_2D = pd.DataFrame(X_train_2D, columns = ['PC1', 'PC2'], index = X_train.index)
df_train_2D['diagnosis'] = y_train
df_train_2D.head()
```

	PC1	PC2	diagnosis
40	-2.159600	-1.916635	M
409	-1.921507	-0.140895	B
113	-1.145023	3.761724	B
74	-2.617820	-0.601161	B
369	7.542934	-3.928613	M

```
plt.scatter(df_train_2D[df_train_2D['diagnosis'] == 'B']['PC1'], df_train_2D[df_train_2D['diagnosis'] == 'B']['PC2'], c
plt.scatter(df_train_2D[df_train_2D['diagnosis'] == 'M']['PC1'], df_train_2D[df_train_2D['diagnosis'] == 'M']['PC2'], c

plt.legend(frameon=True, bbox_to_anchor=(1.4, 1.03), fontsize = 13)
plt.xticks(fontsize = 13)
plt.yticks(fontsize = 13)
plt.xlabel('PC1', fontsize = 15)
plt.ylabel('PC2', fontsize = 15)
plt.title('2D PCA plot for Sample Subgroups', fontsize = 16)
```



- Sample subgroups relatively well-determined by diagnosis status.

4. **Filter Method for feature selection based on Chi-square test:**
   - Initial selection of features. Select features with significant t-test scores
   - However, filter method only select features based on statistical score of feature by outcome, it does not handle multicollinearity and includes all correlated features if they are important

```
print ('Features Selected by Filter Method: p-value \n')

filtered_features = list()
for i in feature_names[1:]:
    test = stats.ttest_ind(data_scaled_B[i], data_scaled_M[i])
    if test.pvalue < 0.05:
        print(i, ':', test.pvalue)
        filtered_features.append(i)
```

```
Features Selected by Filter Method: p-value

radius_mean : 8.465940572263146e-96
texture_mean : 4.0586360478981613e-25
perimeter_mean : 8.436251036172569e-101
area_mean : 4.734564310307614e-88
smoothness_mean : 1.0518503592032693e-18
compactness_mean : 3.938263105886996e-56
concavity_mean : 9.966555755072963e-84
concave points_mean : 7.101150161057016e-116
symmetry_mean : 5.733384028466786e-16
radius_se : 9.73894865646109e-50
perimeter_se : 1.651905175849735e-47
area_se : 5.8955213926058635e-46
compactness_se : 9.975994654074837e-13
concavity_se : 8.260176167970112e-10
concave points_se : 3.07230876881808 74e-24
radius_worst : 8.482291921683931e-116
texture_worst : 1.0780574879493623e-30
perimeter_worst : 5.771397139668948e-119
area_worst : 2.828847704286774e-97
smoothness_worst : 6.57514363398425e-26
compactness_worst : 7.069816352538678e-55
concavity_worst : 2.464663956782782e-72
concave points_worst : 1.9690997072164375e-124
symmetry_worst : 2.9511205771538056e-25
fractal_dimension_worst : 2.316432449982804e-15
```

   - A total of 25 features selected by filter methods.

5. **Combine observation from boxplot, swarmplot, multicollinearity, PCA, and filter selection. Hand-select the following features considering:**
   - Association with outcome variable
   - Variance explained
   - Multicollinearity

```
selected_features = ['radius_mean', 'texture_worst', 'smoothness_worst', 'concave points_mean', 'area_worst', 'symmetry_
X_train_selected = X_train_scaled[X_train_scaled.columns & selected_features]
X_test_selected = X_test_scaled[X_test_scaled.columns & selected_features]
print('hand-selected features: \n', list(X_train_selected.columns))
```

```
hand-selected features:
 ['radius_mean', 'concave points_mean', 'texture_worst', 'area_worst', 'smoothness_worst', 'symmetry_worst']
```

   - Features selected: *{radius_mean, concave_points_mean, texture_worst, area_worst, smoothness_worst, symmetry_worst}*

CLASSIFICATION MODELS:

1. **Model 1 - Logistic Regression**:
   - From EDA, we know multicollinearity exist between features selected above. Use regularization on logistic regression to reduce multicollinearity and overfitting. Use cross-validation to search for best regularization parameter c.
   - *Performance no tuning original (30 features):*

```
Logistic Regression without any tunning or feature selection (30 features):
               Train      Test
Accuracy    0.994975  0.959064
Precision   1.000000  0.968254
Recall      0.986301  0.924242
f1          0.993103  0.945736
AUC         0.997880  0.994228
```

   - *Performance with cross validation & grid search original (30 features):*

```
Logistic Regression with cross-validation (30 features):
               Train      Test
Accuracy    0.989950  0.964912
Precision   0.993056  0.983871
Recall      0.979452  0.924242
f1          0.986207  0.953125
AUC         0.996331  0.994661
```

   - *Performance with cross validation & grid search filtered (25 features):*

```
Logistic Regression with cross-validation and feature filter (25 features):
               Train      Test
Accuracy    0.989950  0.959064
Precision   0.993056  0.983607
Recall      0.979452  0.909091
f1          0.986207  0.944882
AUC         0.996630  0.994661
```

   - *Performance with cross validation & grid search hand selected (6 features):*

```
Logistic Regression with cross-validation and selected features (6 features):
               Train      Test
Accuracy    0.984925  0.947368
Precision   1.000000  0.938462
Recall      0.958904  0.924242
f1          0.979021  0.931298
AUC         0.990134  0.992641
```

Feature Selection for Logistic Regression with SelectFromModel():

```
# Feature selection for regression:
logistic_clf = LogisticRegression()
logistic_clf.fit(X_train_scaled, y_train_binary)
logistic_model = SelectFromModel(logistic_clf, prefit = True)
X_train_selected_logistic = logistic_model.transform(X_train_scaled)
X_test_selected_logistic = logistic_model.transform(X_test_scaled)

# Get feature names from final logistic regression model:
selected_features_logistic = list(X_train_scaled.T.iloc[logistic_model.get_support()].index)
print('features selected for logistic regression: \n', selected_features_logistic)
```

```
features selected for logistic regression:
 ['compactness_mean', 'concavity_mean', 'concave points_mean', 'radius_se', 'perimeter_se', 'area_se', 'compactness_s
e', 'radius_worst', 'texture_worst', 'perimeter_worst', 'area_worst', 'smoothness_worst', 'concavity_worst', 'concave
points_worst', 'symmetry_worst']
```

- *Performance with cross validation & grid search model selected (15 features):*

```
Logistic Regression with cross-validationa and feature selection from model (15 features):
              Train      Test
Accuracy   0.987437  0.953216
Precision  1.000000  0.983333
Recall     0.965753  0.893939
f1         0.982578  0.936508
AUC        0.997092  0.995094
```

- Pros vs. Cons of logistic regression:

  (-) prone to overfitting, (-) prone to outliers, (+) feature importance interpretable (standardized).

- It can be seen from the above results, logistic regression had very good performance on both normalized training and testing datasets. Tuning regularization parameter C slightly reduced overfitting on training data. However, further feature selections did not improve model.

2. **Model 2 - Random Forest**:

- *Performance no tuning original (30 features):*

```
Random Forest without any tunning or feature selection (30 features):
           Train      Test
Accuracy     1.0  0.959064
Precision    1.0  0.968254
Recall       1.0  0.924242
f1           1.0  0.945736
AUC          1.0  0.983189
```

- *Performance with cross validation & grid search original (30 features):*

```
Random Forest with cross-validation (30 features):
              Train      Test
Accuracy   0.992462  0.941520
Precision  0.993103  0.937500
Recall     0.986301  0.909091
f1         0.989691  0.923077
AUC        0.999946  0.988745
```

- *Performance with cross validation & grid search filtered (25 features):*

```
Random Forest with cross-validation and filtered features (25 features):
              Train      Test
Accuracy   0.992462  0.947368
Precision  0.993103  0.925373
Recall     0.986301  0.939394
f1         0.989691  0.932331
AUC        0.999837  0.988889
```

- *Performance with cross validation & grid search hand selected (6 features):*

```
Random Forest with cross-validation and filtered features (6 features):
              Train      Test
Accuracy   0.992462  0.964912
Precision  1.000000  0.968750
Recall     0.979452  0.939394
f1         0.989619  0.953846
AUC        0.999891  0.992641
```

- *Performance with cross validation & grid search model selected (14 features):*

Recursive Feature Elimination to select features:

```
forest_clf = RandomForestClassifier()
score_type = make_scorer(roc_auc_score)
rfecv = RFECV(estimator = forest_clf, cv = 5, scoring = score_type)
rfecv.fit(X_train_scaled, y_train_binary)
selected_features_forest = list(X_train_scaled.T.iloc[rfecv.support_].index)

X_train_selected_forest = rfecv.transform(X_train_scaled)
X_test_selected_forest = rfecv.transform(X_test_scaled)

print('number of features selected: ', rfecv.n_features_)
print('features selected: \n', selected_features_forest)
```

```
number of features selected:  14
features selected:
 ['texture_mean', 'perimeter_mean', 'smoothness_mean', 'compactness_mean', 'concave points_mean', 'area_se', 'compact
ness_se', 'radius_worst', 'texture_worst', 'perimeter_worst', 'area_worst', 'concavity_worst', 'concave points_wors
t', 'fractal_dimension_worst']
```

```
Random Forest with cross-validation and selected features (14 features):
              Train      Test
Accuracy   0.992462  0.947368
Precision  1.000000  0.938462
Recall     0.979452  0.924242
f1         0.989619  0.931298
AUC        0.999918  0.988528
```

- Pros vs. Cons of random forest:

  (+) not prone to multicollinearity, (+) no linearity assumptions, (+) robust to high-dimension data, (+) can directly output feature importance, can visualize feature importance, (-) slow training, may still over-fit

- Tuning did not improve model much. But reducing features hand-selected 6 features improved accuracy on test data.

3. **Gradient Boosting Tree**:

- *Performance no tuning original (30 features):*

```
Gradient Boosting without any tunning or feature selection (30 features):
           Train      Test
Accuracy     1.0  0.935673
Precision    1.0  0.923077
Recall       1.0  0.909091
f1           1.0  0.916031
AUC          1.0  0.984848
```

- *Performance with cross validation & grid search original (30 features):*

```
Gradient Boost with cross-validation (30 features):
           Train      Test
Accuracy     1.0  0.918129
Precision    1.0  0.882353
Recall       1.0  0.909091
f1           1.0  0.895522
AUC          1.0  0.984993
```

- *Performance with cross validation & grid search filtered (25 features):*

```
Gradient Boost with cross-validation (25 features):
           Train      Test
Accuracy     1.0  0.918129
Precision    1.0  0.893939
Recall       1.0  0.893939
f1           1.0  0.893939
AUC          1.0  0.983261
```

- *Performance with cross validation & grid search hand selected (6 features):*

```
Gradient Boost with hand selected features (6 features):
           Train      Test
Accuracy     1.0  0.959064
Precision    1.0  0.968254
Recall       1.0  0.924242
f1           1.0  0.945736
AUC          1.0  0.991342
```

- *Performance with cross validation & grid search model selected (14 features):*

Recursive Feature Elimination to Select Features:

```
gradient_clf = GradientBoostingClassifier()
score_type = make_scorer(roc_auc_score)
rfecv = RFECV(estimator = gradient_clf, cv = 5, scoring = score_type)
rfecv.fit(X_train_scaled, y_train_binary)
selected_features_gradient = list(X_train_scaled.T.iloc[rfecv.support_].index)

X_train_selected_gradient = rfecv.transform(X_train_scaled)
X_test_selected_gradient = rfecv.transform(X_test_scaled)

print('number of features selected: ', rfecv.n_features_)
print('features selected: \n', selected_features_gradient)
```

```
number of features selected:  30
features selected:
 ['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mea
n', 'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean', 'radius_se', 'texture_se', 'perimeter_se', 'are
a_se', 'smoothness_se', 'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se', 'fractal_dimension_se',
'radius_worst', 'texture_worst', 'perimeter_worst', 'area_worst', 'smoothness_worst', 'compactness_worst', 'concavity
_worst', 'concave points_worst', 'symmetry_worst', 'fractal_dimension_worst']
```

```
Gradient Boost with cross-validation selected features (9 features):
           Train      Test
Accuracy     1.0  0.935673
Precision    1.0  0.936508
Recall       1.0  0.893939
f1           1.0  0.914729
AUC          1.0  0.984993
```

- Pros vs. Cons of gradient boosting tree:

  (+) no linearity assumptions, (+) easy tuning, (-) performance influenced by base learner, (-) performance influenced by outliers

- Very good performance on training set, but overfitting problem present. Performance on testing data improve after reducing features to 6.

4. **KNN**: Cross-validation to find optimal k (selected features)
   - *Performance no tuning original (30 features):*

```
KNN without tunning or feature selection:
              Train      Test
Accuracy   0.979899  0.941520
Precision  0.992857  0.966667
Recall     0.952055  0.878788
f1         0.972028  0.920635
AUC        0.998410  0.976479
```

   - *Performance with cross validation & grid search original (30 features):*

```
KNN with cross-validation (30 features):
               Train      Test
Accuracy   0.979899  0.941520
Precision  0.992857  0.966667
Recall     0.952055  0.878788
f1         0.972028  0.920635
AUC        0.998410  0.976479
```

   - *Performance with cross validation & grid search filtered (25 features):*

```
KNN with cross-validation and feature filter (25 features):
              Train      Test
Accuracy   0.984925  0.947368
Precision  0.992958  0.952381
Recall     0.965753  0.909091
f1         0.979167  0.930233
AUC        0.998736  0.969553
```

   - *Performance with cross validation & grid search hand selected (6 features):*

```
KNN with cross-validation and feature filter (6 features):
              Train      Test
Accuracy   0.982412  0.947368
Precision  1.000000  0.952381
Recall     0.952055  0.909091
f1         0.975439  0.930233
AUC        0.999198  0.971429
```

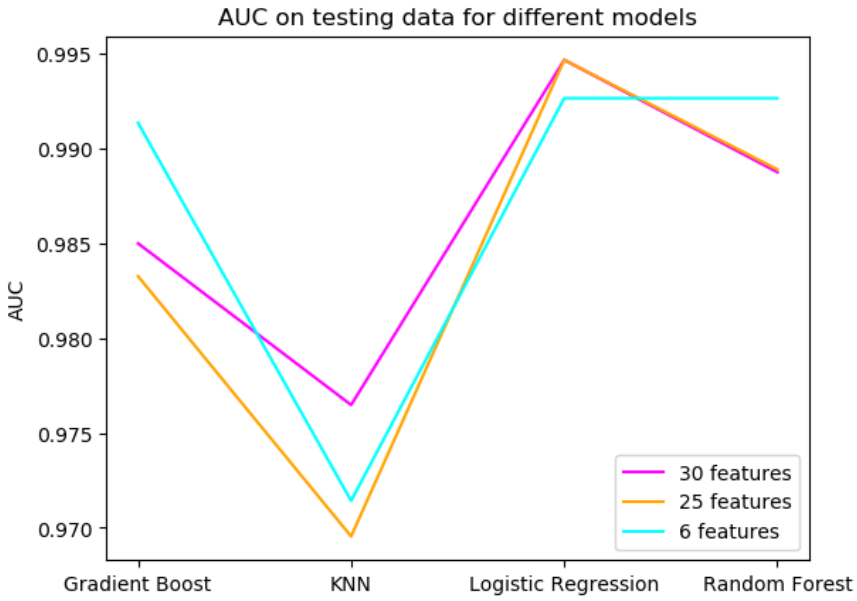
   - Pros vs. Cons of KNN: (+) no linearity assumptions, (+) fast train, (-) curse of dimensionality
   - Very similar performance with/without tuning and feature selection.

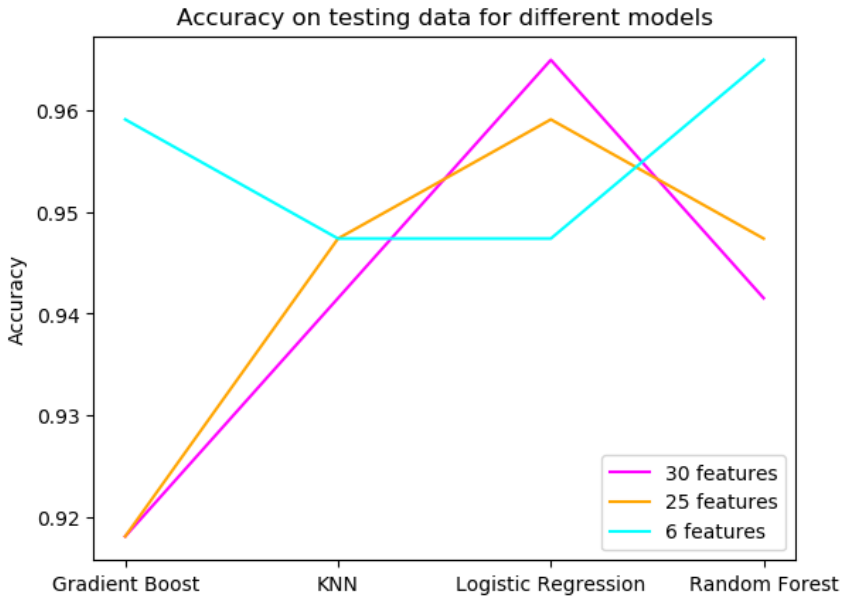
COMPARE MODEL PERFORMANCE:

Compare model performance on sets of features selected through:

1. Filter methods: features with top chi-sq scores with outcome variables (do not for sure reduce multicollinearity)
2. Embedded methods: regularization (reduce multicollinearity in logistic regression)
3. Wrapper methods: recursive feature elimination to determine optimal number of features (reduce multicollinearity)

- Plot AUC and Accuracy for each model for each set of features to visualize model performance:





- No model is universally better on this dataset.
- Generally, regularized logistic regression have better performance with more features. Tree-based methods have better performance with less features. KNN has worst AUC for all feature sets.
- Note the ranges of y axis on above plots are small, meaning all model have similar performance. - Nevertheless, all models achieved >95% accuracy and >0.90 AUC on predicting breast cancer for this dataset after tuning and feature selection.