1. Loading libraries and cleaning data

```
import numpy as np
import pandas as pd

import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
from sklearn.preprocessing import RobustScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split, GridSearchCV,
StratifiedKFold
from sklearn.feature_selection import RFECV
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, plot_confusion_matrix,
roc_curve, precision_recall_curve, auc
from plotly.subplots import make_subplots
import itertools
# Run the following two lines of code for Uncaught Error: Script error
for plotly
from plotly.offline import plot, iplot, init_notebook_mode
init_notebook_mode(connected=True)

df = pd.read_csv('../input/breast-cancer-wisconsin-data/data.csv')
df.head()
```

```
         id diagnosis  radius_mean  texture_mean  perimeter_mean
area_mean  \
0    842302         M        17.99         10.38          122.80
1001.0
1    842517         M        20.57         17.77          132.90
1326.0
2  84300903         M        19.69         21.25          130.00
1203.0
3  84348301         M        11.42         20.38           77.58
386.1
4  84358402         M        20.29         14.34          135.10
1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave
points_mean  \
0          0.11840           0.27760          0.3001
0.14710
1          0.08474           0.07864          0.0869
0.07017
```

```
2          0.10960          0.15990          0.1974
0.12790
3          0.14250          0.28390          0.2414
0.10520
4          0.10030          0.13280          0.1980
0.10430

   ...  texture_worst  perimeter_worst  area_worst
smoothness_worst  \
0  ...          17.33           184.60      2019.0            0.1622

1  ...          23.41           158.80      1956.0            0.1238

2  ...          25.53           152.50      1709.0            0.1444

3  ...          26.50            98.87       567.7            0.2098

4  ...          16.67           152.20      1575.0            0.1374


   compactness_worst  concavity_worst  concave points_worst
symmetry_worst  \
0             0.6656           0.7119                0.2654
0.4601
1             0.1866           0.2416                0.1860
0.2750
2             0.4245           0.4504                0.2430
0.3613
3             0.8663           0.6869                0.2575
0.6638
4             0.2050           0.4000                0.1625
0.2364

   fractal_dimension_worst  Unnamed: 32
0                  0.11890          NaN
1                  0.08902          NaN
2                  0.08758          NaN
3                  0.17300          NaN
4                  0.07678          NaN

[5 rows x 33 columns]
```

```
missing_values_count = df.isnull().sum()
missing_values_count
```

```
id                             0
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
Unnamed: 32              569
dtype: int64
```

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

Data analysis

```
df.shape
```

```
(569, 31)
```

```
df.diagnosis.unique()
```

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

```
df.describe()
```

```
       radius_mean  texture_mean  perimeter_mean    area_mean  \
count   569.000000    569.000000      569.000000   569.000000
mean     14.127292     19.289649       91.969033   654.889104
std       3.524049      4.301036       24.298981   351.914129
min       6.981000      9.710000       43.790000   143.500000
25%      11.700000     16.170000       75.170000   420.300000
50%      13.370000     18.840000       86.240000   551.100000
75%      15.780000     21.800000      104.100000   782.700000
```

max	28.110000	39.280000	188.500000	2501.000000

	smoothness_mean	compactness_mean	concavity_mean	concave points_mean
count	569.000000	569.000000	569.000000	569.000000
mean	0.096360	0.104341	0.088799	0.048919
std	0.014064	0.052813	0.079720	0.038803
min	0.052630	0.019380	0.000000	0.000000
25%	0.086370	0.064920	0.029560	0.020310
50%	0.095870	0.092630	0.061540	0.033500
75%	0.105300	0.130400	0.130700	0.074000
max	0.163400	0.345400	0.426800	0.201200

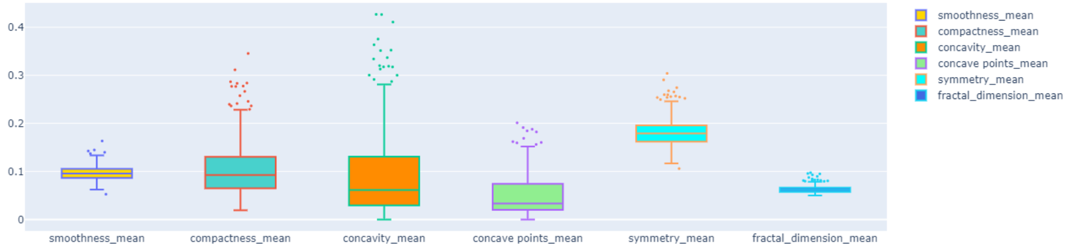
	symmetry_mean	fractal_dimension_mean	...	radius_worst
count	569.000000	569.000000	...	569.000000
mean	0.181162	0.062798	...	16.269190
std	0.027414	0.007060	...	4.833242
min	0.106000	0.049960	...	7.930000
25%	0.161900	0.057700	...	13.010000
50%	0.179200	0.061540	...	14.970000
75%	0.195700	0.066120	...	18.790000
max	0.304000	0.097440	...	36.040000

	texture_worst	perimeter_worst	area_worst	smoothness_worst
count	569.000000	569.000000	569.000000	569.000000
mean	25.677223	107.261213	880.583128	0.132369
std	6.146258	33.602542	569.356993	0.022832
min	12.020000	50.410000	185.200000	0.071170
25%	21.080000	84.110000	515.300000	0.116600
50%	25.410000	97.660000	686.500000	0.131300
75%	29.720000	125.400000	1084.000000	0.146000
max	49.540000	251.200000	4254.000000	0.222600

```
       compactness_worst  concavity_worst  concave points_worst  \
count         569.000000       569.000000            569.000000
mean            0.254265         0.272188              0.114606
std             0.157336         0.208624              0.065732
min             0.027290         0.000000              0.000000
25%             0.147200         0.114500              0.064930
50%             0.211900         0.226700              0.099930
75%             0.339100         0.382900              0.161400
max             1.058000         1.252000              0.291000

       symmetry_worst  fractal_dimension_worst
count      569.000000               569.000000
mean         0.290076                 0.083946
std          0.061867                 0.018061
min          0.156500                 0.055040
25%          0.250400                 0.071460
50%          0.282200                 0.080040
75%          0.317900                 0.092080
max          0.663800                 0.207500

[8 rows x 30 columns]
```

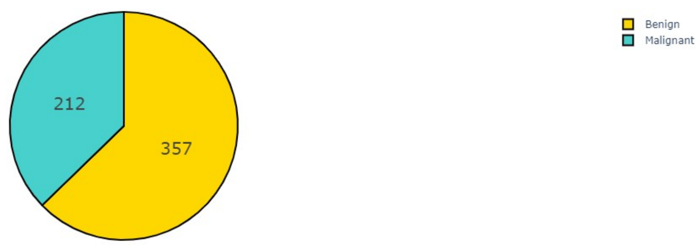
Outlier detection

```
names = df.columns[5:11]
# convert DataFrame to list
values=[]
for column in df.iloc[:,5:11].columns:
    li = df[column].tolist()
    values.append(li)
colors = ['gold', 'mediumturquoise', 'darkorange',
'lightgreen','cyan', 'royalblue']

fig = go.Figure()
for xd, yd, cls in zip(names, values, colors):
        fig.add_trace(go.Box(
            y=yd,
            name=xd,
            boxpoints='outliers',
            jitter=0.5,
            whiskerwidth=0.2,
            fillcolor=cls,
            marker_size=3,
            line_width=2)
        )
fig.show()
```



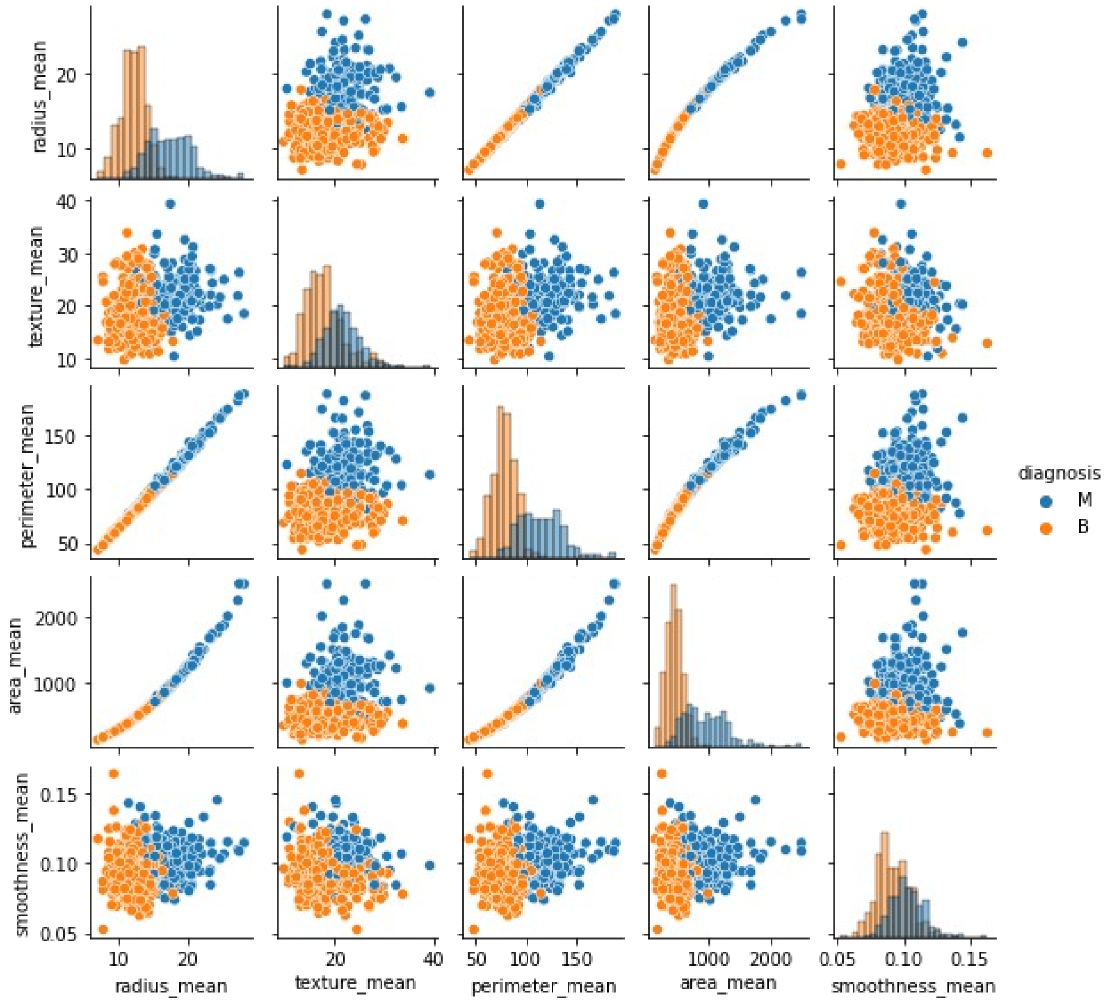
Target distribution

```
fig = go.Figure(data=[go.Pie(labels=['Benign','Malignant'],
values=df['diagnosis'].value_counts(), textinfo='label+percent')])
fig.update_traces(hoverinfo='label+percent', textinfo='value',
textfont_size=20,
                  marker=dict(colors=['gold', 'mediumturquoise'],
line=dict(color='#000000', width=2)))
fig.show()
```



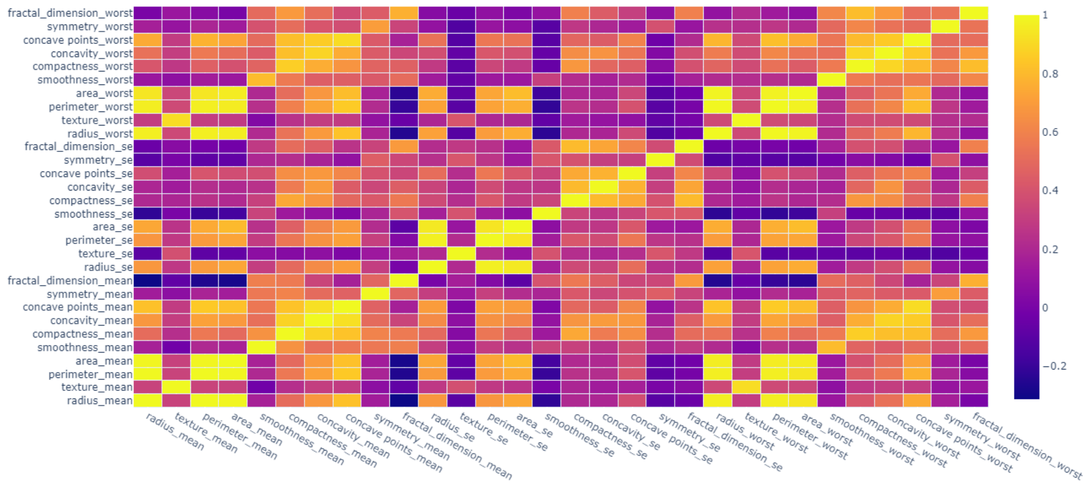

```
sns.pairplot(df.iloc[:,:6],hue='diagnosis',
diag_kind='hist',height=1.6)
```

```
<seaborn.axisgrid.PairGrid at 0x7f60f9d36910>
```



2. 5. Correlation matrix

```
corr = df.iloc[:,1:].corr()
fig =
go.Figure(data=go.Heatmap(z=np.array(corr),x=corr.columns.tolist(),y=c
orr.columns.tolist(),xgap = 1,ygap = 1))
fig.update_layout(margin = dict(t=25,r=0,b=200,l=200),width = 1000,
height = 700)
fig.show()
```



Data preprocessing and Feature Engineering

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['diagnosis'] = le.fit_transform(df['diagnosis']) # M:1, B:0
df['diagnosis'].value_counts()
```

```
0    357
1    212
Name: diagnosis, dtype: int64
```

Splitting the data

```
random_state = 42
X_train, X_test, y_train, y_test = train_test_split(df.iloc[:,1:],
df['diagnosis'], test_size = 0.2, random_state = random_state)
```

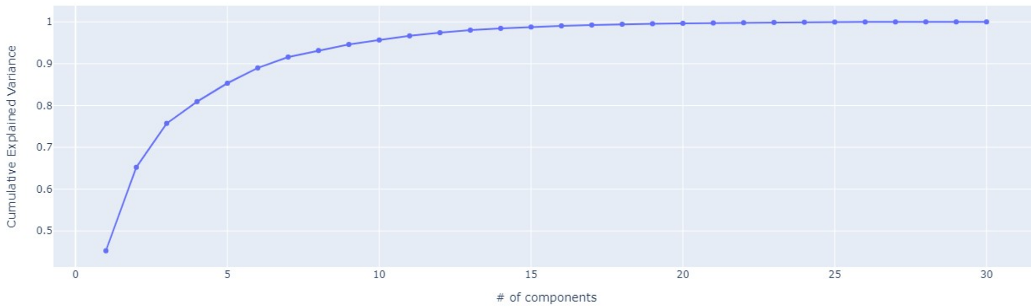
```
# Scaling the data using robust scaler because of its robustness
against outliers
scale = RobustScaler()
X_train = scale.fit_transform(X_train)
X_test = scale.transform(X_test)
```

Reducing Dimensionality

- **Principal component analysis (PCA)**

```
pca = PCA()
pca.fit(X_train)
exp_var_cumul = np.cumsum(pca.explained_variance_ratio_)

fig = px.line(x=np.arange(1,exp_var_cumul.shape[0]+1),
y=exp_var_cumul, markers=True, labels={'x':'# of components',
'y':'Cumulative Explained Variance'})
```

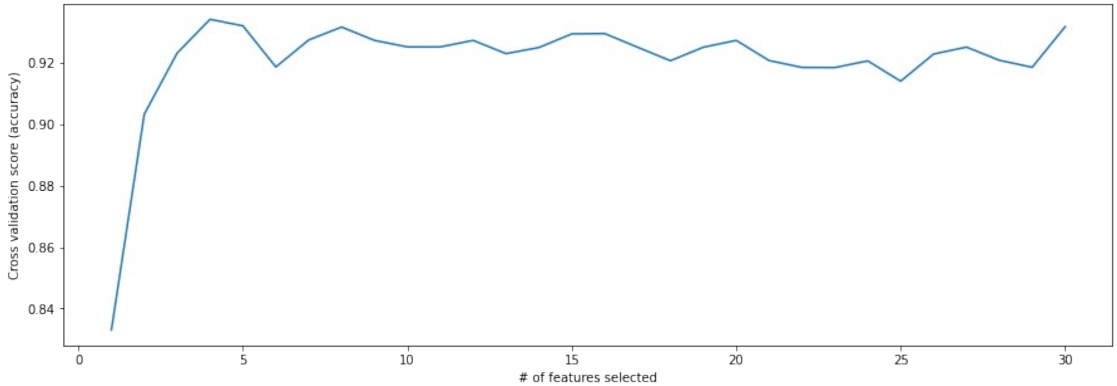
```
#fig.add_shape(type='line', line=dict(dash='dash'),x0=0, x1=30,
y0=0.95, y1=0.95)

fig.show()
```



There is an elbow after the seventh component, and 91% of the total variance is explained by the firrst seventh components. If keeping the first 10 or 17 principal components, we can preserve about 95 % or even more than 99% of the total variance.

- **Recursive features elimination(RFE)**

```
# Fit the RFE model to identify the optimum number of features .
rfecv = RFECV(cv=StratifiedKFold(n_splits=10,
random_state=random_state, shuffle=True),
      estimator=DecisionTreeClassifier(), scoring='accuracy')

rfecv.fit(X_train, y_train)

print("Optimal number of features : %d" % rfecv.n_features_)

# Plot number of features VS. cross-validation scores
plt.figure(figsize=(15,5))
plt.xlabel("# of features selected")
plt.ylabel("Cross validation score (accuracy)")
plt.plot(
    range(1, len(rfecv.cv_results_['mean_test_score']) + 1),
    rfecv.cv_results_['mean_test_score'],
)
plt.show()

Optimal number of features : 4
```




```
# Identifying the features RFE selected
df_features = pd.DataFrame(columns = ['feature', 'support',
'ranking'])

for i in range(X_train.shape[1]):
    row = {'feature': i, 'support': rfecv.support_[i], 'ranking':
rfecv.ranking_[i]}
    df_features = df_features.append(row, ignore_index=True)

df_features.sort_values(by='ranking').head(10)
#df_features[df_features['support']==True]
```

	feature	support	ranking
22	22	True	1
21	21	True	1
20	20	True	1
7	7	True	1
27	27	False	2
19	19	False	3
17	17	False	4
13	13	False	5
24	24	False	6
14	14	False	7

```
# Identifying the features' name RFE selected
df.columns[1:][rfecv.get_support()]
```

```
Index(['concave points_mean', 'radius_worst', 'texture_worst',
       'perimeter_worst'],
      dtype='object')
```

Grid Search Cross validation

```
def modelselection(classifier, parameters, scoring, X_train):
    clf = GridSearchCV(estimator=classifier,
                       param_grid=parameters,
                       scoring= scoring,
                       cv=5,
                       n_jobs=-1)# n_jobs refers to the number of CPU's
```

```
that you want to use for excution, -1 means that use all available
computing power.
    clf.fit(X_train, y_train)
    cv_results = clf.cv_results_
    best_parameters = clf.best_params_
    best_result = clf.best_score_
    print('The best parameters for classifier is', best_parameters)
    print('The best training score is %.3f:'% best_result)
#    print(sorted(cv_results.keys()))
    return cv_results, best_parameters, best_result
```

No of Components in PCA versus Model Accuracy/Training Time

```
def PCA_curves(PCA_cv_score, PCA_test_score, training_time):
    fig = make_subplots(
    rows=1, cols=2,
    specs=[[{"type": "scatter"}, {"type": "scatter"}]],
    subplot_titles=('# of Components in PCA versus Model Accuracy','#
of Components in PCA versus Training Time')
    )

    fig.add_trace(go.Scatter(x=n,y=PCA_cv_score,
                             line=dict(color='rgb(231,107,243)',
width=2), name='CV score'),
                  row=1, col=1)
    fig.add_trace(go.Scatter(x=n,y=PCA_test_score,
                             line=dict(color='rgb(0,176,246)',
width=2), name='Test score'),
                  row=1, col=1)
    fig.add_trace(go.Scatter(x=n,y=training_time,
                             line=dict(color='rgb(0,100,80)',
width=2), name='Training time'),
                  row=1, col=2)
    fig.update_xaxes(title_text='# of components')
    fig.update_yaxes(title_text='Accuracy', row=1, col=1)
    # fig.update_xaxes(title_text="Recall", row=1, col=2)
    fig.update_yaxes(title_text='Training time', row=1, col=2)
    fig.show()
```

Model Measures

Confusion Matrices & Metrics

A confusion matrix is a table that categorizes predictions according to whether they match the actual value

- True Positive (TP): Malignant tumour correctly classified as Malignant
- True Negative (TN): Benign tumour correctly classified as benign
- False Positive (FP): Benign tumour incorrectly classified as malignant
- False Negative (FN): Malignant tumour incorrectly classified as benign

Metrics

- Accuracy, Sensitivity, Precision, Recall, F-measure etcc

```
def metrics(X,CV_clf):
    y_pred = CV_clf.predict(X)
    cm = confusion_matrix(y_test, y_pred)
    tn = cm[0,0]
    fp = cm[0,1]
    fn = cm[1,0]
    tp = cm[1,1]
    Accuracy=(tp+tn)/(tp+tn+fp+fn)
    Sensitivity=tp/(tp+fn)
    Specificity=tn/(tn+fp)
    Precision=tp/(tp+fp)
    F_measure=2*tp/(2*tp+fp+fn)
    print('Accuracy=%.3f'%Accuracy)
    print('Sensitivity=%.3f'%Sensitivity) # as the same as recall
    print('Specificity=%.3f'%Specificity)
    print('Precision=%.3f'%Precision)
    print('F-measure=%.3f'%F_measure)
    return Accuracy, Sensitivity, Specificity, Precision, F_measure
 #   plot_confusion_matrix(CV_clf, X_test, y_test)
```

ROC and PRC

```
def plot_roc_prc():
    fpr, tpr, thresholds = roc_curve(y_test, y_score)
    precision, recall, thresholds = precision_recall_curve(y_test, 
y_score)
    fig = make_subplots(
        rows=1, cols=2,
        specs=[[{"type": "scatter"}, {"type": "scatter"}]],
        subplot_titles=(f'ROC Curve (AUC={auc(fpr, 
tpr):.4f})',f'Precision-Recall Curve (AUC={auc(fpr, tpr):.4f})')
    )
    fig.add_trace(go.Scatter(x=fpr, y=tpr),row=1, col=1)
    fig.add_shape(type='line', line=dict(dash='dash'),x0=0, x1=1, 
y0=0, y1=1,row=1, col=1)
    fig.add_trace(go.Scatter(x=recall, y=precision),row=1, col=2)
    fig.add_shape(type='line', line=dict(dash='dash'),x0=0, x1=1, 
y0=0.5, y1=0.5,row=1, col=2)
    # Update axis properties
    fig.update_xaxes(title_text="False Positive Rate / 1-Specificity", 
row=1, col=1)
    fig.update_yaxes(title_text="True Positive Rate / Recall", row=1, 
col=1)
    fig.update_xaxes(title_text="Recall", row=1, col=2)
    fig.update_yaxes(title_text="Precision", row=1, col=2)
    fig.show()
```

logisitic Regression

```
classifier_log =
LogisticRegression(random_state=random_state,solver='lbfgs',
max_iter=1000)
parameters_log = {
            'penalty' : ['l2'],
            'C' : [0.01, 0.1, 1, 10, 100]
}
scoring='accuracy'
# Find the best hyperparameters
cv_results, best_param, best_result =
modelselection(classifier_log,parameters_log, scoring, X_train)
```

```
The best parameters for classifier is {'C': 10, 'penalty': 'l2'}
The best training score is 0.978:
```

```
# Classifier with the best hyperparameters
logReg_clf = LogisticRegression(penalty = best_param['penalty'],
                            C = best_param['C'],
                            random_state=random_state)
logReg_clf.fit(X_train, y_train)

# Metrics
logReg_metrics = metrics(X_test,logReg_clf)
```

```
Accuracy=0.974
Sensitivity=0.977
Specificity=0.972
Precision=0.955
F-measure=0.966
```

logistic Regression with PCA

```
def compare_pca(n_components):
    cv_score, test_score, cv_training_time = [], [], []
    for n in n_components:
        print("The number of components in PCA is:%d "% n)
        pca = PCA(n_components=n,
svd_solver="full",random_state=random_state)
        X_PCA_train = pca.fit_transform(X_train)
        X_PCA_test = pca.transform(X_test)
        # Model Selection
        cv_results, best_param, best_result =
modelselection(classifier_log,parameters_log, scoring, X_PCA_train)
        training_time = np.mean(np.array(cv_results['mean_fit_time'])
+np.array(cv_results['mean_score_time']))
        cv_score.append(best_result)
        cv_training_time.append(training_time)
        CV_clf = LogisticRegression(penalty = best_param['penalty'],
                                    C = best_param['C'],
                                    random_state=random_state)
        CV_clf.fit(X_PCA_train, y_train)
```

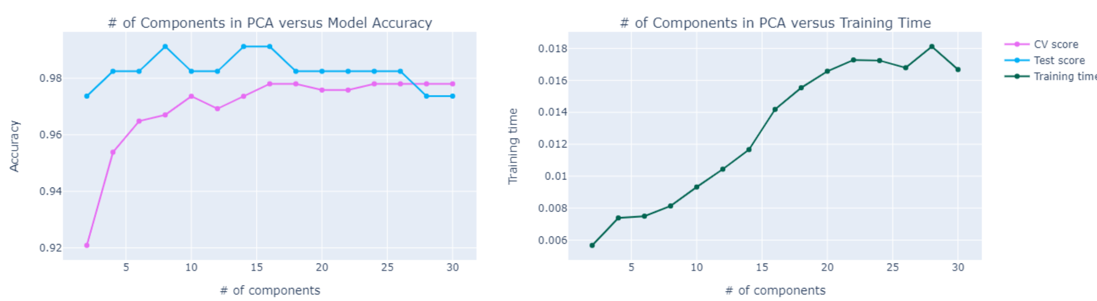
```python
        score = CV_clf.score(X_PCA_test, y_test)
        test_score.append(score)
    print(cv_score, test_score, cv_training_time)
    return cv_score, test_score, cv_training_time

n_features = X_train.shape[1]
n = np.arange(2, n_features+2, 2)

PCA_cv_score, PCA_test_score, PCA_cv_training_time=
compare_pca(n_components = n)
```

```
The number of components in PCA is:2
The best parameters for classifier is {'C': 100, 'penalty': 'l2'}
The best training score is 0.921:
The number of components in PCA is:4
The best parameters for classifier is {'C': 1, 'penalty': 'l2'}
The best training score is 0.954:
The number of components in PCA is:6
The best parameters for classifier is {'C': 10, 'penalty': 'l2'}
The best training score is 0.965:
The number of components in PCA is:8
The best parameters for classifier is {'C': 0.1, 'penalty': 'l2'}
The best training score is 0.967:
The number of components in PCA is:10
The best parameters for classifier is {'C': 1, 'penalty': 'l2'}
The best training score is 0.974:
The number of components in PCA is:12
The best parameters for classifier is {'C': 1, 'penalty': 'l2'}
The best training score is 0.969:
The number of components in PCA is:14
The best parameters for classifier is {'C': 1, 'penalty': 'l2'}
The best training score is 0.974:
The number of components in PCA is:16
The best parameters for classifier is {'C': 1, 'penalty': 'l2'}
The best training score is 0.978:
The number of components in PCA is:18
The best parameters for classifier is {'C': 10, 'penalty': 'l2'}
The best training score is 0.978:
The number of components in PCA is:20
The best parameters for classifier is {'C': 10, 'penalty': 'l2'}
The best training score is 0.976:
The number of components in PCA is:22
The best parameters for classifier is {'C': 10, 'penalty': 'l2'}
The best training score is 0.976:
The number of components in PCA is:24
The best parameters for classifier is {'C': 10, 'penalty': 'l2'}
The best training score is 0.978:
The number of components in PCA is:26
The best parameters for classifier is {'C': 10, 'penalty': 'l2'}
The best training score is 0.978:
```

```
The number of components in PCA is:28
The best parameters for classifier is {'C': 10, 'penalty': 'l2'}
The best training score is 0.978:
The number of components in PCA is:30
The best parameters for classifier is {'C': 10, 'penalty': 'l2'}
The best training score is 0.978:
[0.9208791208791209, 0.9538461538461538, 0.9648351648351647,
0.9670329670329672, 0.9736263736263737, 0.9692307692307691,
0.9736263736263737, 0.9780219780219781, 0.9780219780219781,
0.9758241758241759, 0.9758241758241759, 0.9780219780219781,
0.9780219780219781, 0.9780219780219781, 0.9780219780219781]
[0.9736842105263158, 0.9824561403508771, 0.9824561403508771,
0.9912280701754386, 0.9824561403508771, 0.9824561403508771,
0.9912280701754386, 0.9912280701754386, 0.9824561403508771,
0.9824561403508771, 0.9824561403508771, 0.9824561403508771,
0.9824561403508771, 0.9736842105263158, 0.9736842105263158]
[0.005665903091430663, 0.007385101318359376, 0.007496204376220704,
0.008134241104125976, 0.009320449829101561, 0.010440435409545899,
0.011664266586303712, 0.014184713363647461, 0.015540065765380858,
0.016575126647949218, 0.01728216171264648, 0.017242679595947264,
0.01679497718811035, 0.018123645782470704, 0.016681098937988283]
```

No of Components in PCA versus Model Accuracy/Training Time

```
PCA_curves(PCA_cv_score,PCA_test_score,PCA_cv_training_time)
```



logistic Regression with PCA (8 components)

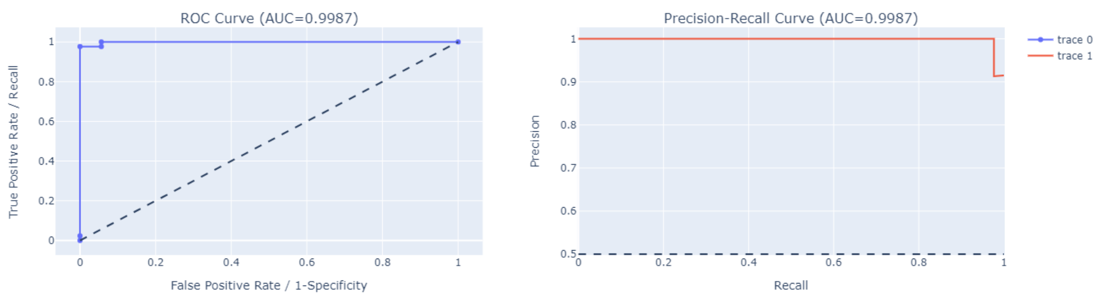
```
i =PCA_test_score.index(max(PCA_test_score))
print('The best accuracy of logistic regression classifier is: %.3f'%
max(PCA_test_score)+', where the total number of components in PCA is
{:.0f}'.format((i+1)*2))
```

```
The best accuracy of logistic regression classifier is: 0.991, where
the total number of components in PCA is 8
```

```
pca = PCA(n_components=(i+1)*2,
svd_solver="full",random_state=random_state)
X_PCA_train = pca.fit_transform(X_train)
X_PCA_test = pca.transform(X_test)
```

```python
# Model Selection
cv_results, best_param, best_result =
modelselection(classifier_log,parameters_log, scoring, X_PCA_train)

# Classifier with the best hyperparameters
logReg_PCA = LogisticRegression(penalty = best_param['penalty'],
                                C = best_param['C'],
                                random_state=random_state)
logReg_PCA.fit(X_PCA_train, y_train)

# Metrics
logReg_PCA_metrics = metrics(X_PCA_test,logReg_PCA)

# ROC Curve & Precision-Recall Curves
y_score = logReg_PCA.predict_proba(X_PCA_test)[:, 1] # predict
probabilities
plot_roc_prc()
```

```
The best parameters for classifier is {'C': 0.1, 'penalty': 'l2'}
The best training score is 0.967:
Accuracy=0.991
Sensitivity=0.977
Specificity=1.000
Precision=1.000
F-measure=0.988
```

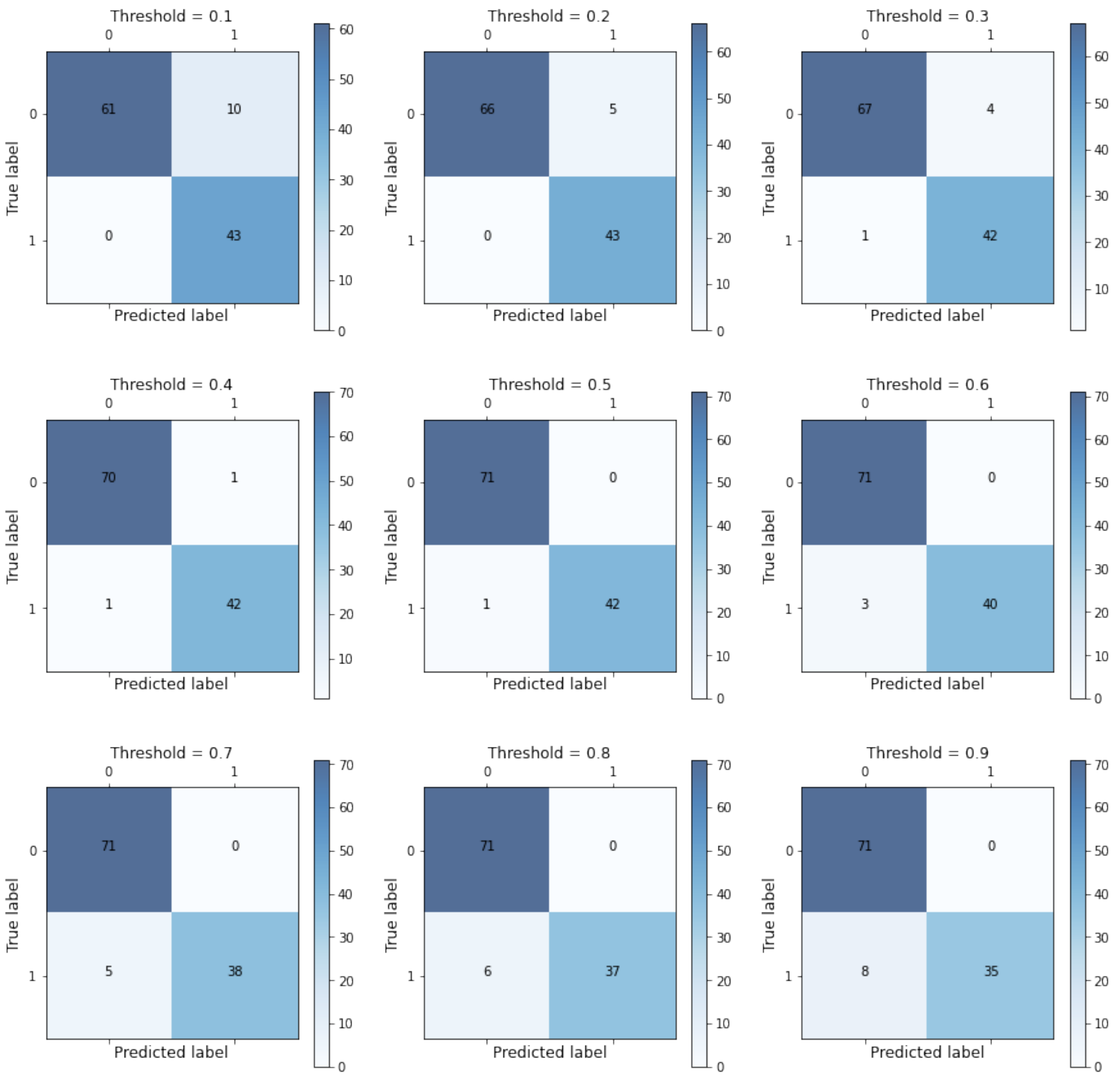


Specifying thresholds for metrics

```python
thresholds = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(15, 15))

for n, ax in zip(thresholds,axs.ravel()):
    y_score = logReg_PCA.predict_proba(X_PCA_test)[:,1] > n

    cm = confusion_matrix(y_test, y_score)

    tp = cm[1,1]
    fn = cm[1,0]
    fp = cm[0,1]
```

```
    tn = cm[0,0]

    print('threshold = %s :'%n,
          'Accuracy={:.3f}'.format((tp+tn)/(tp+tn+fp+fn)),
          'Sensitivity={:.3f}'.format(tp/(tp+fn)),
          'Specificity={:.3f}'.format(tn/(tn+fp)),
          'Precision={:.3f}'.format(tp/(tp+fp)))

    im=ax.matshow(cm, cmap='Blues', alpha=0.7)

    for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])) :
        ax.text(j, i, cm[i, j], horizontalalignment = 'center')

    ax.set_ylabel('True label',fontsize=12)
    ax.set_xlabel('Predicted label',fontsize=12)
    ax.set_title('Threshold = %s'%n, fontsize=12)
    fig.colorbar(im, ax=ax,orientation='vertical');
plt.show()
```

```
threshold = 0.1 : Accuracy=0.912 Sensitivity=1.000 Specificity=0.859
Precision=0.811
threshold = 0.2 : Accuracy=0.956 Sensitivity=1.000 Specificity=0.930
Precision=0.896
threshold = 0.3 : Accuracy=0.956 Sensitivity=0.977 Specificity=0.944
Precision=0.913
threshold = 0.4 : Accuracy=0.982 Sensitivity=0.977 Specificity=0.986
Precision=0.977
threshold = 0.5 : Accuracy=0.991 Sensitivity=0.977 Specificity=1.000
Precision=1.000
threshold = 0.6 : Accuracy=0.974 Sensitivity=0.930 Specificity=1.000
Precision=1.000
threshold = 0.7 : Accuracy=0.956 Sensitivity=0.884 Specificity=1.000
Precision=1.000
threshold = 0.8 : Accuracy=0.947 Sensitivity=0.860 Specificity=1.000
Precision=1.000
threshold = 0.9 : Accuracy=0.930 Sensitivity=0.814 Specificity=1.000
Precision=1.000
```



6. 3. Random Forest

```
parameters_rf = {
    'n_estimators': [20, 50, 100, 150, 200],
    'criterion': ['gini', 'entropy'],
    'bootstrap': [True, False],
}
# We can test values for other parameters, such as max_features,
max_depth, max_leaf_nodes, to see if the accuracy further impoves or
not
scoring_rf = 'accuracy'
"""
    scoring parameters:
https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-
parameter
"""
classifier_rf = RandomForestClassifier(random_state=random_state)
```

```python
# Find the best hyperparameters
cv_results, best_param, best_result =
modelselection(classifier_rf,parameters_rf, scoring_rf, X_train)

# Classifier with the best hyperparameters
rf_clf = RandomForestClassifier(n_estimators =
best_param['n_estimators'],
                                criterion = best_param['criterion'],
                                bootstrap = best_param['bootstrap'],
                                random_state=random_state)
rf_clf.fit(X_train, y_train)

# Metrics
rf_metrics = metrics(X_test,rf_clf)
```

```
The best parameters for classifier is {'bootstrap': True, 'criterion':
'entropy', 'n_estimators': 100}
The best training score is 0.967:
Accuracy=0.965
Sensitivity=0.930
Specificity=0.986
Precision=0.976
F-measure=0.952
```

6. 4. Random Forest with PCA

```python
def compare_pca(n_components):
    cv_score, test_score, cv_training_time = [], [], []
    for n in n_components:
        print("The number of components in PCA is:%d "% n)
        pca = PCA(n_components=n,
svd_solver="full",random_state=random_state)
        X_PCA_train = pca.fit_transform(X_train)
        X_PCA_test = pca.transform(X_test)
        # Model Selection
        cv_results, best_param, best_result =
modelselection(classifier_rf,parameters_rf, scoring, X_PCA_train)
        training_time = np.mean(np.array(cv_results['mean_fit_time'])
+np.array(cv_results['mean_score_time']))
        cv_score.append(best_result)
        cv_training_time.append(training_time)
        CV_clf = RandomForestClassifier(n_estimators =
best_param['n_estimators'],
                                        criterion =
best_param['criterion'],
                                        bootstrap =
best_param['bootstrap'],
                                        random_state=random_state)
        CV_clf.fit(X_PCA_train, y_train)
        score = CV_clf.score(X_PCA_test, y_test)
        test_score.append(score)
```

```
    print(cv_score, test_score, cv_training_time)
    return cv_score, test_score, cv_training_time
```

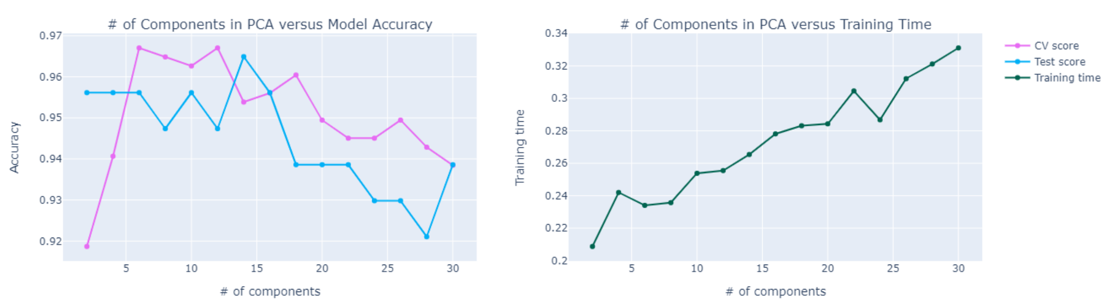
```
n_features = X_train.shape[1]
n = np.arange(2, n_features+2, 2)
PCA_cv_score, PCA_test_score, PCA_cv_training_time=
compare_pca(n_components = n)
```

```
The number of components in PCA is:2
The best parameters for classifier is {'bootstrap': True, 'criterion':
'entropy', 'n_estimators': 150}
The best training score is 0.919:
The number of components in PCA is:4
The best parameters for classifier is {'bootstrap': True, 'criterion':
'gini', 'n_estimators': 50}
The best training score is 0.941:
The number of components in PCA is:6
The best parameters for classifier is {'bootstrap': False,
'criterion': 'gini', 'n_estimators': 200}
The best training score is 0.967:
The number of components in PCA is:8
The best parameters for classifier is {'bootstrap': False,
'criterion': 'gini', 'n_estimators': 200}
The best training score is 0.965:
The number of components in PCA is:10
The best parameters for classifier is {'bootstrap': True, 'criterion':
'entropy', 'n_estimators': 200}
The best training score is 0.963:
The number of components in PCA is:12
The best parameters for classifier is {'bootstrap': False,
'criterion': 'entropy', 'n_estimators': 100}
The best training score is 0.967:
The number of components in PCA is:14
The best parameters for classifier is {'bootstrap': True, 'criterion':
'entropy', 'n_estimators': 20}
The best training score is 0.954:
The number of components in PCA is:16
The best parameters for classifier is {'bootstrap': False,
'criterion': 'entropy', 'n_estimators': 100}
The best training score is 0.956:
The number of components in PCA is:18
The best parameters for classifier is {'bootstrap': False,
'criterion': 'entropy', 'n_estimators': 20}
The best training score is 0.960:
The number of components in PCA is:20
The best parameters for classifier is {'bootstrap': False,
'criterion': 'entropy', 'n_estimators': 200}
The best training score is 0.949:
The number of components in PCA is:22
The best parameters for classifier is {'bootstrap': False,
'criterion': 'entropy', 'n_estimators': 100}
```

```
The best training score is 0.945:
The number of components in PCA is:24
The best parameters for classifier is {'bootstrap': False,
'criterion': 'gini', 'n_estimators': 20}
The best training score is 0.945:
The number of components in PCA is:26
The best parameters for classifier is {'bootstrap': False,
'criterion': 'gini', 'n_estimators': 20}
The best training score is 0.949:
The number of components in PCA is:28
The best parameters for classifier is {'bootstrap': True, 'criterion':
'entropy', 'n_estimators': 20}
The best training score is 0.943:
The number of components in PCA is:30
The best parameters for classifier is {'bootstrap': True, 'criterion':
'entropy', 'n_estimators': 50}
The best training score is 0.938:
[0.9186813186813187, 0.9406593406593406, 0.9670329670329672,
0.9648351648351647, 0.9626373626373625, 0.9670329670329669,
0.9538461538461538, 0.956043956043956, 0.9604395604395604,
0.9494505494505494, 0.945054945054945, 0.945054945054945,
0.9494505494505494, 0.9428571428571428, 0.9384615384615385]
[0.956140350877193, 0.956140350877193, 0.956140350877193,
0.9473684210526315, 0.956140350877193, 0.9473684210526315,
0.9649122807017544, 0.956140350877193, 0.9385964912280702,
0.9385964912280702, 0.9385964912280702, 0.9298245614035088,
0.9298245614035088, 0.9210526315789473, 0.9385964912280702]
[0.20870310068130493, 0.24198606967926023, 0.23404119968414308,
0.23571539878845216, 0.25382235765457156, 0.25549466609954835,
0.26541335821151735, 0.27812360286712645, 0.2831190872192383,
0.2843164086341858, 0.3045900464057922, 0.28680178165435793,
0.31214831829071044, 0.3212070345878601, 0.3311116552352905]
```

No of Components in PCA versus Model Accuracy/Training Time

```
PCA_curves(PCA_cv_score, PCA_test_score, PCA_cv_training_time)
```



6. 4. 2. logistic Regression with PCA (14 components)

```
i =PCA_test_score.index(max(PCA_test_score))
pca = PCA(n_components=(i+2)*2,
svd_solver="full",random_state=random_state)
X_PCA_train = pca.fit_transform(X_train)
X_PCA_test = pca.transform(X_test)
# Model Selection
cv_results, best_param, best_result =
modelselection(classifier_rf,parameters_rf, scoring, X_PCA_train)
rf_PCA = RandomForestClassifier(n_estimators =
best_param['n_estimators'],
                                criterion = best_param['criterion'],
                                bootstrap = best_param['bootstrap'],
                                random_state=random_state)
rf_PCA.fit(X_PCA_train, y_train)
# Metrics
rf_PCA_metrics = metrics(X_PCA_test,rf_PCA)
```

```
The best parameters for classifier is {'bootstrap': False,
'criterion': 'entropy', 'n_estimators': 100}
The best training score is 0.956:
Accuracy=0.956
Sensitivity=0.953
Specificity=0.958
Precision=0.932
F-measure=0.943
```

6. 5. Random Forest with RFE (Recursive features elimination)

```
X_train_selected = X_train[:,rfecv.get_support()]
X_test_selected = X_test[:,rfecv.get_support()]

cv_results, best_param, best_result =
modelselection(classifier_rf,parameters_rf, scoring_rf,
X_train_selected)

# Classifier with the best hyperparameters
rf_RFE = RandomForestClassifier(n_estimators =
best_param['n_estimators'],
                                criterion = best_param['criterion'],
                                bootstrap = best_param['bootstrap'],
                                random_state=random_state)
rf_RFE.fit(X_train_selected, y_train)

# Metrics
rf_RFE_metrics = metrics(X_test_selected ,rf_RFE)
```

```
The best parameters for classifier is {'bootstrap': True, 'criterion':
'entropy', 'n_estimators': 150}
The best training score is 0.967:
Accuracy=0.965
Sensitivity=0.930
```

```
Specificity=0.986
Precision=0.976
F-measure=0.952
```

6. 6. Model Performance Plot

```
models_metrics = {'logisticRegression': [round(elem, 3) for elem in
logReg_metrics],
                  'logReg+PCA': [round(elem, 3) for elem in
logReg_PCA_metrics],
                  'RandomForest' : [round(elem, 3) for elem in
rf_metrics],
                  'Rand+PCA' : [round(elem, 3) for elem in
rf_PCA_metrics],
                  'Rand+RFE' : [round(elem, 3) for elem in
rf_RFE_metrics]
                }
index=['Accuracy','Sensitivity','Specificity','Precision', 'F-
measure']
df_scores = pd.DataFrame(data = models_metrics, index=index)
ax = df_scores.plot(kind='bar', figsize = (15,6), ylim = (0.90, 1.02),

                    color = ['gold', 'mediumturquoise', 'darkorange',
'lightgreen','cyan'],
                    rot = 0, title ='Models performance (test
scores)',
                    edgecolor = 'grey', alpha = 0.5)
ax.legend(loc='upper center', ncol=5, title="models")
for container in ax.containers:
    ax.bar_label(container)
plt.show()
```