In [57]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, roc_auc_score
from sklearn.decomposition import PCA

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
```

In [58]:

```
df=pd.read_csv((r'C:\Users\mmi\Desktop\cancer.csv'))
```

In [59]:

```
df.head(5)
df.drop('id',axis=1,inplace=True)
df.drop('Unnamed: 32',axis=1,inplace=True)
df
```

Out[59]:

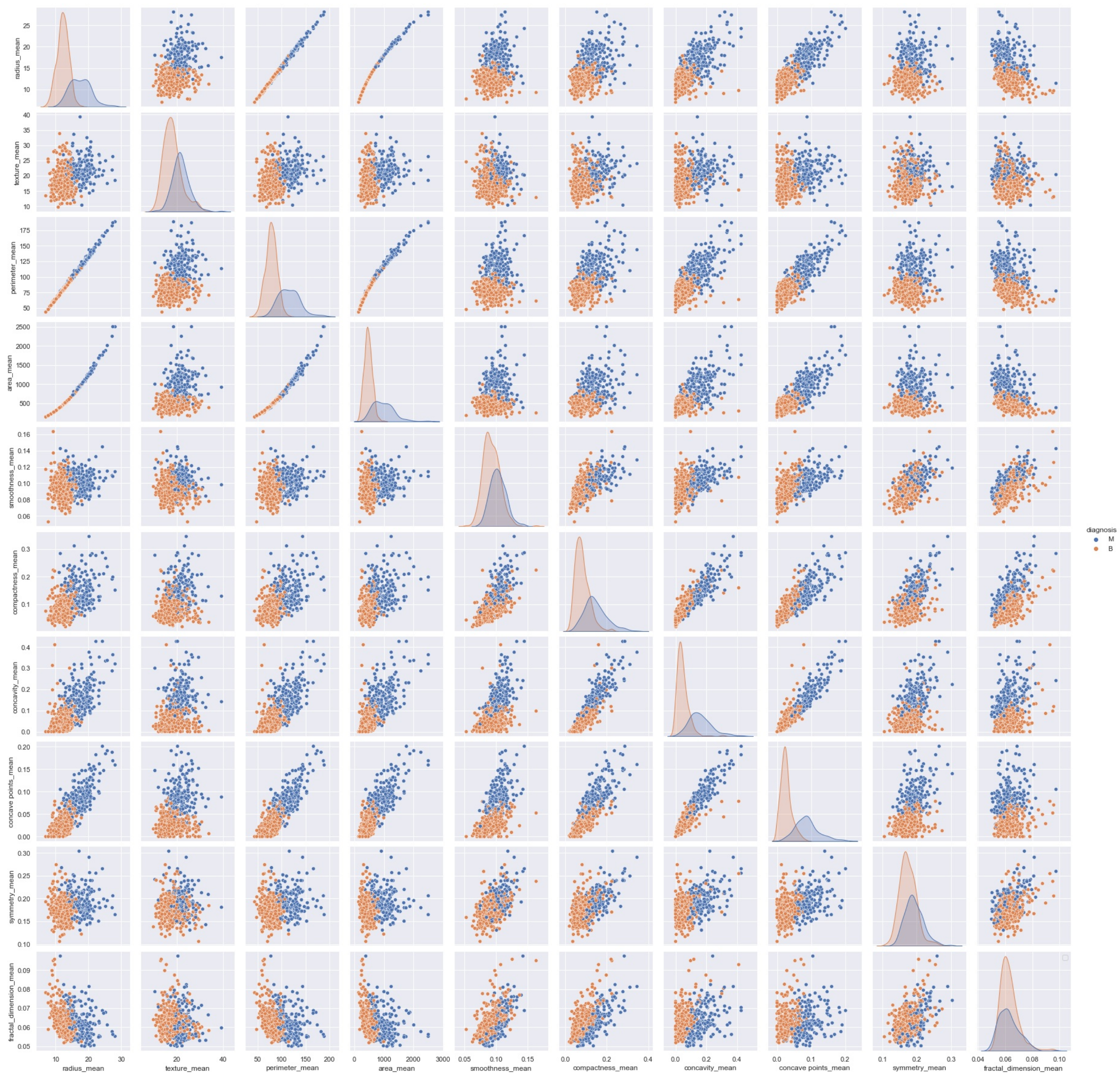
	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	conca points_mea
0	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.147
1	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.070
2	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.127
3	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.105
4	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.104
...	...	...	...	...	...	...	...	...	
564	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.138
565	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.097
566	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.053
567	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.152
568	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.000

569 rows × 31 columns

In [60]:

```
sns.set()
cols_to_pairplot = df.columns[:11]
sns.pairplot(df[cols_to_pairplot], hue="diagnosis")
plt.legend()
plt.show()
```

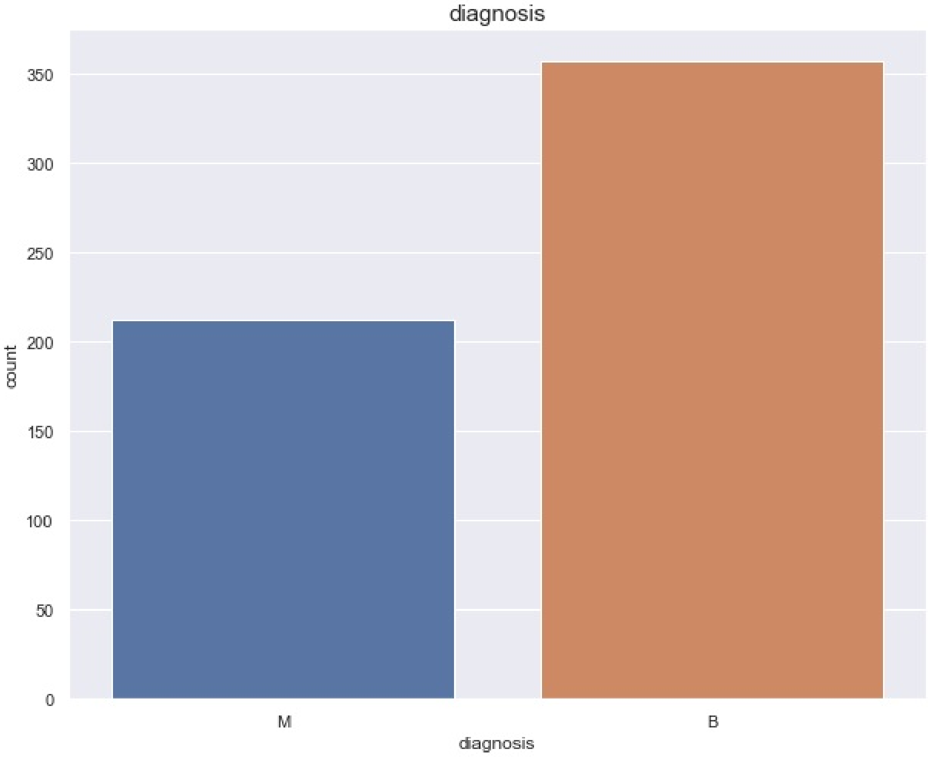
```
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are igno
red when legend() is called with no argument.
```

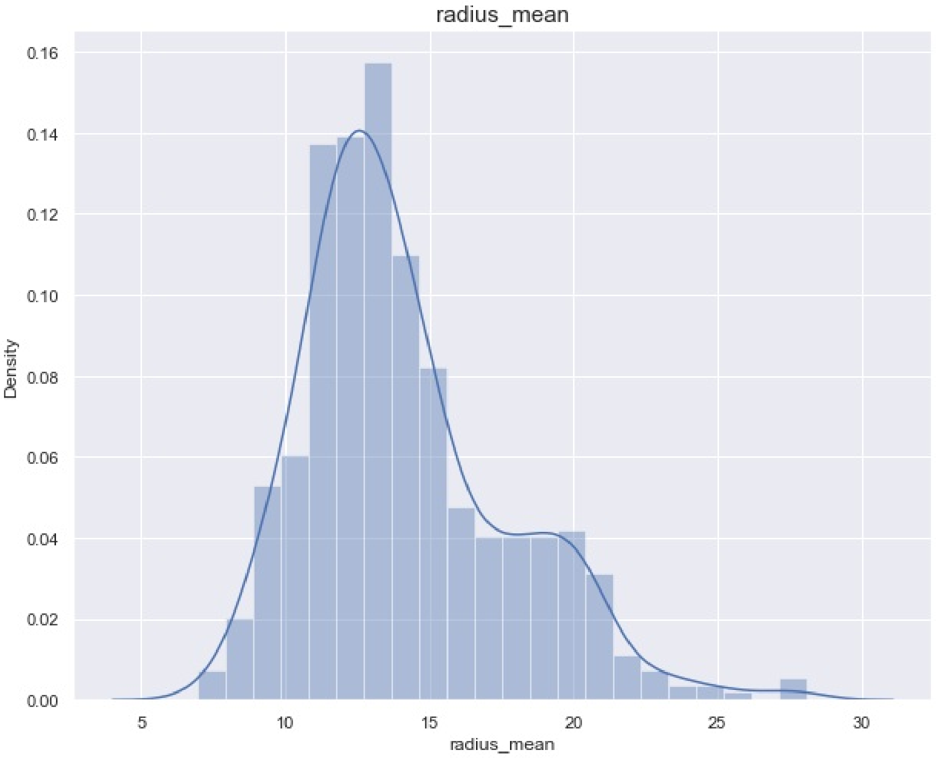


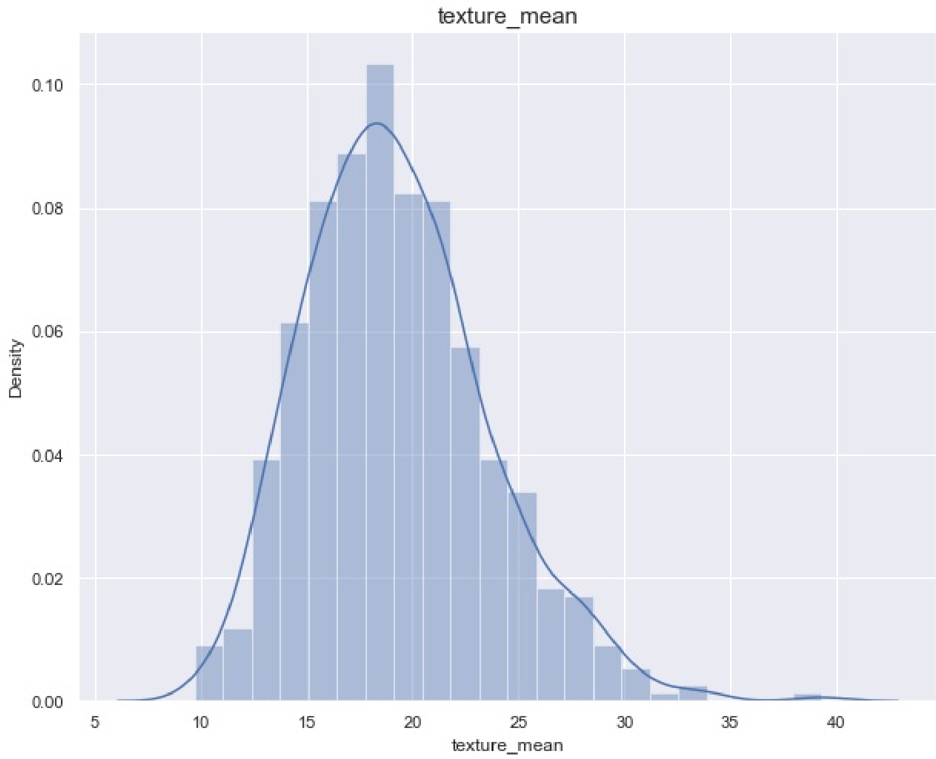
In [61]:

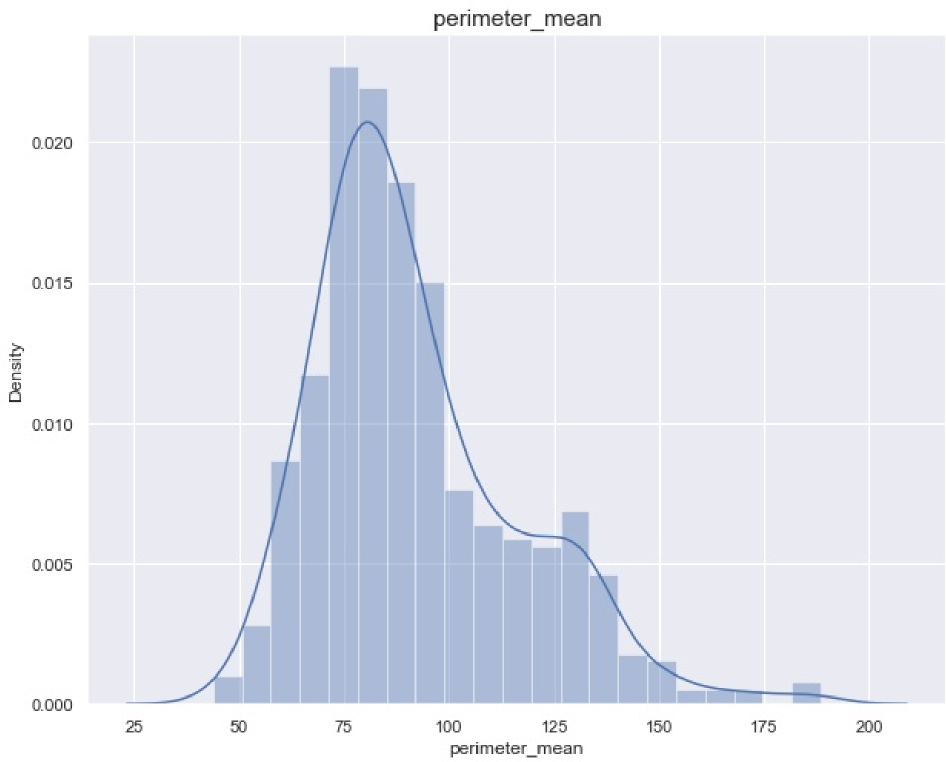
```
plt.figure(figsize=(10,8))
sns.countplot(df["diagnosis"])
plt.title("diagnosis", size=15)
plt.show()
```

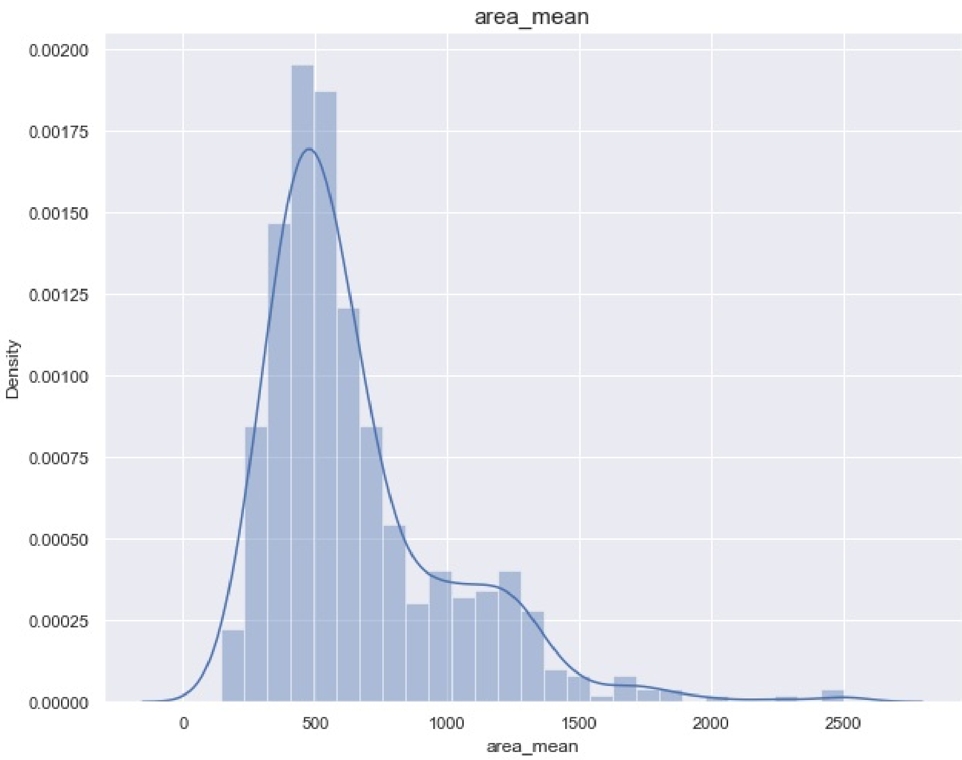
```python
for col in df.drop("diagnosis", axis=1).columns:
    plt.figure(figsize=(10,8))
    sns.distplot(df[col])
    plt.title(f"{col}", size=15)
    plt.show()
```

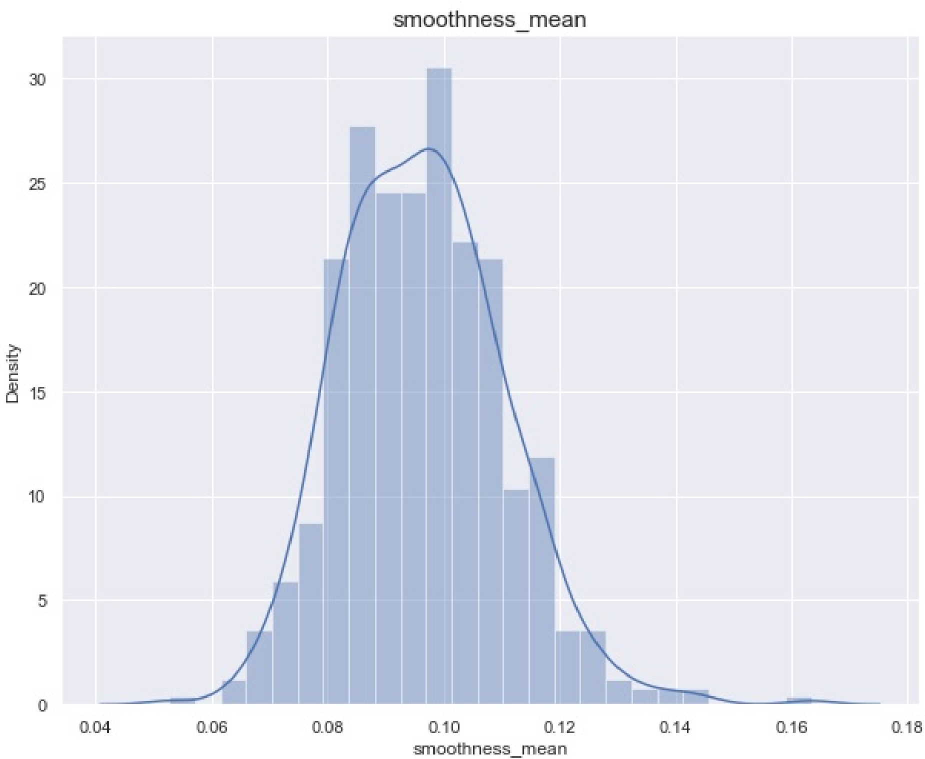


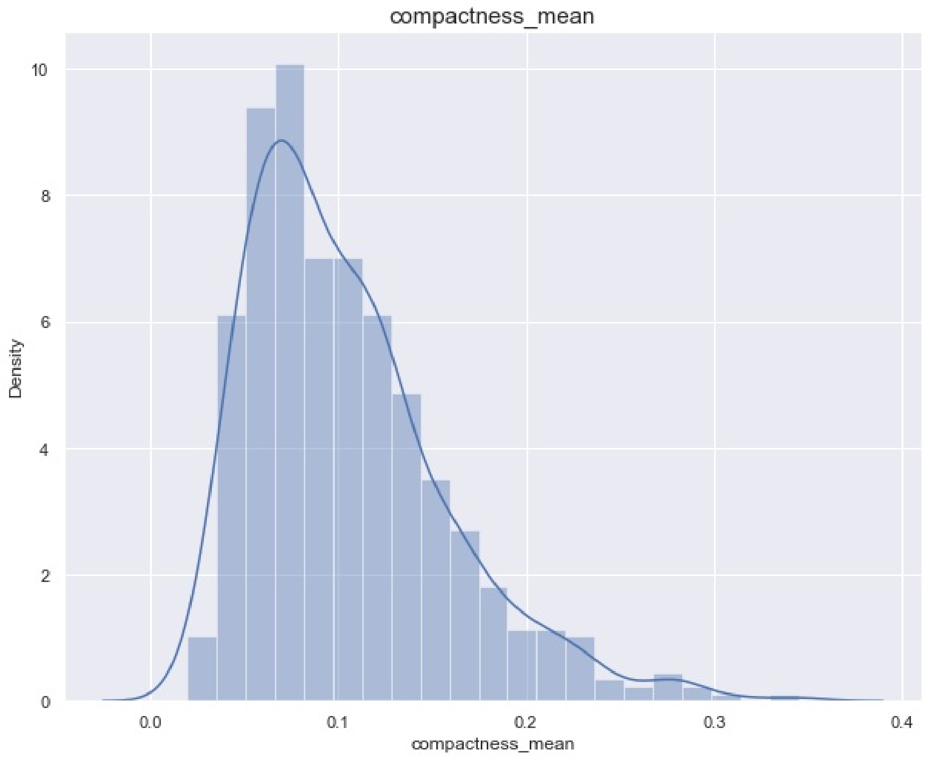


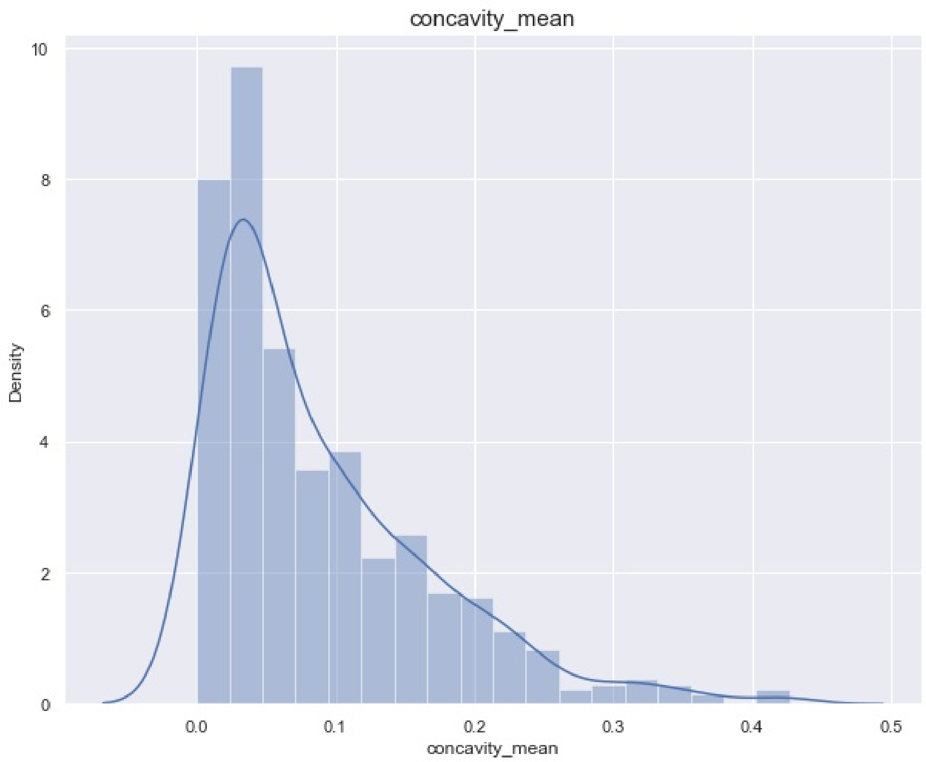

texture_mean
Density
0.10
0.08
0.06
0.04
0.02
0.00
5
10
15
20
25
30
35
40
texture_mean

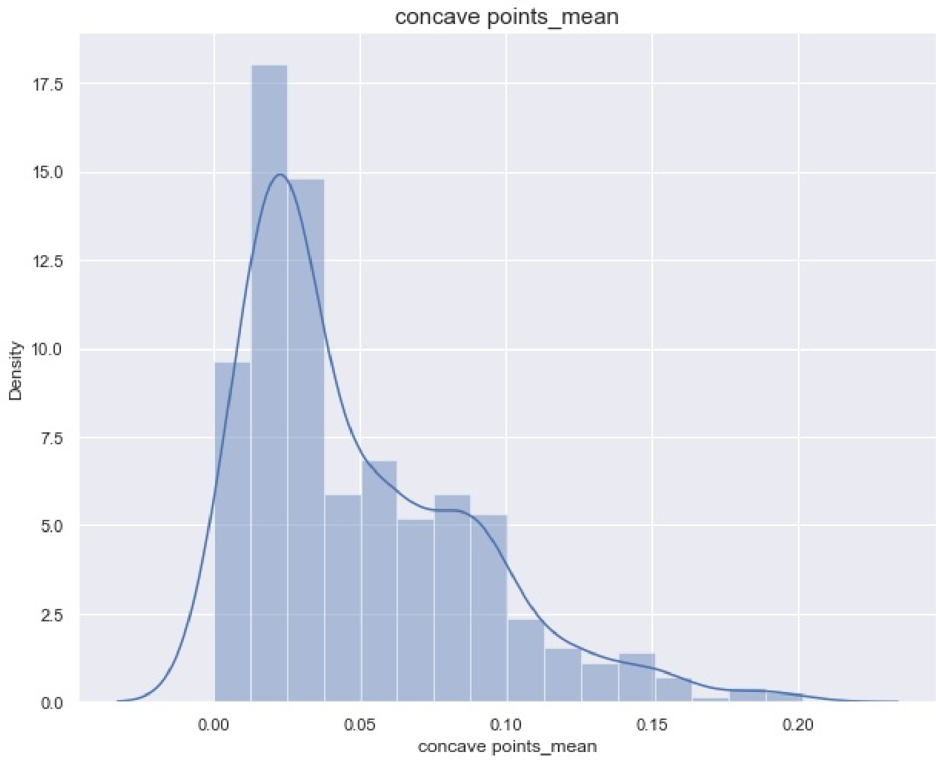

perimeter_mean
Density
0.020
0.015
0.010
0.005
0.000
25
50
75
100
125
150
175
200
perimeter_mean

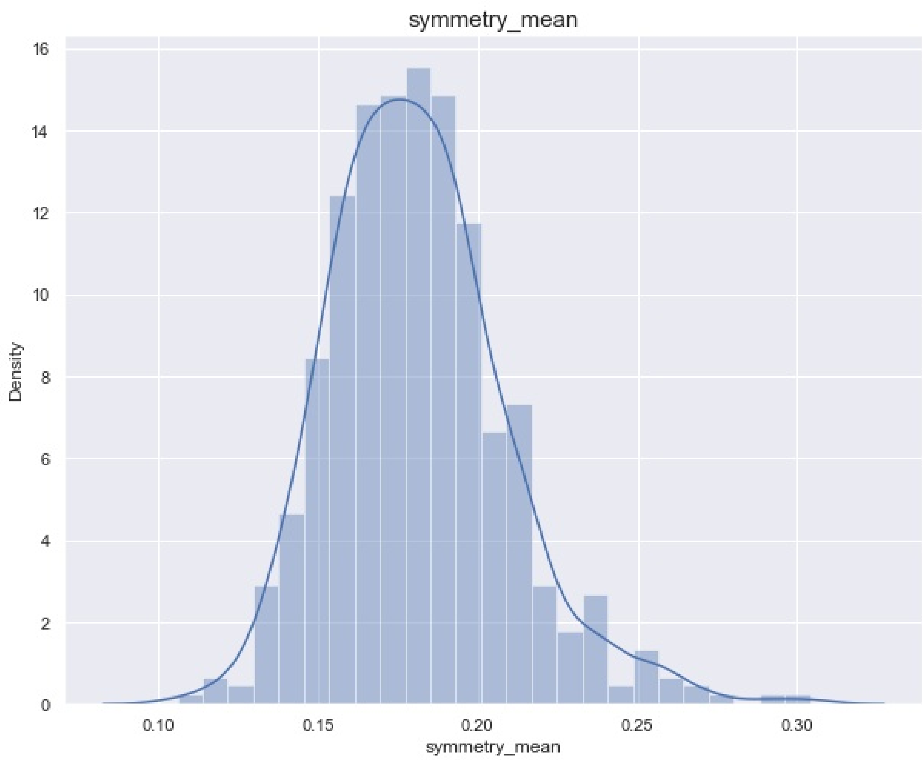

area_mean
0.00200
0.00175
0.00150
0.00125
0.00100
0.00075
0.00050
0.00025
0.00000
Density
0
500
1000
1500
2000
2500
area_mean

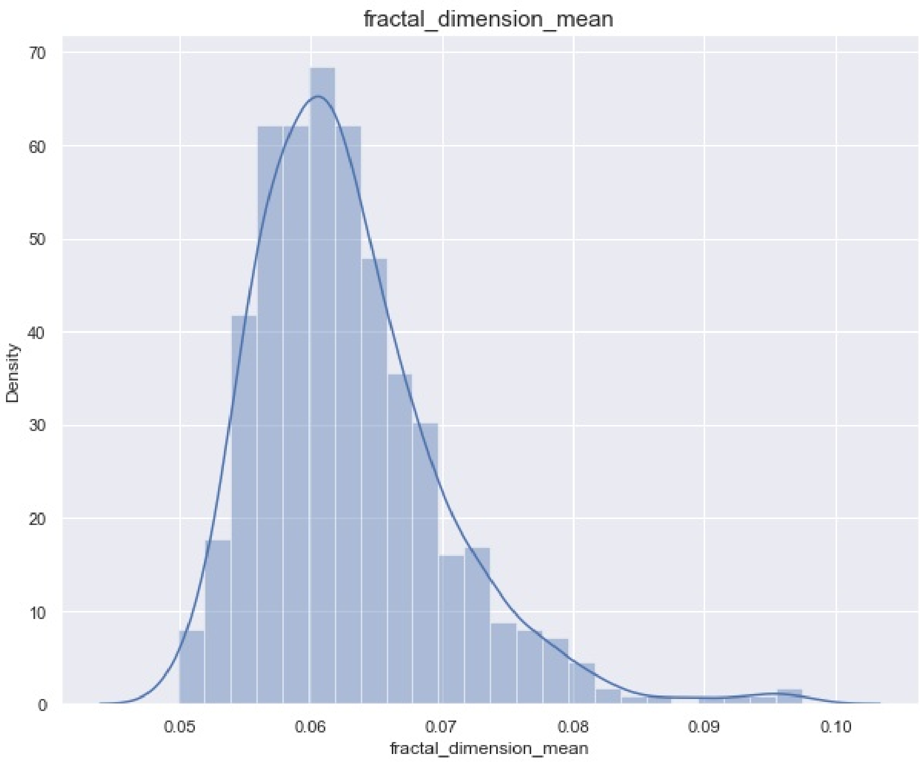

smoothness_mean
30
25
20
15
10
5
0
Density
0.04
0.06
0.08
0.10
0.12
0.14
0.16
0.18
smoothness_mean

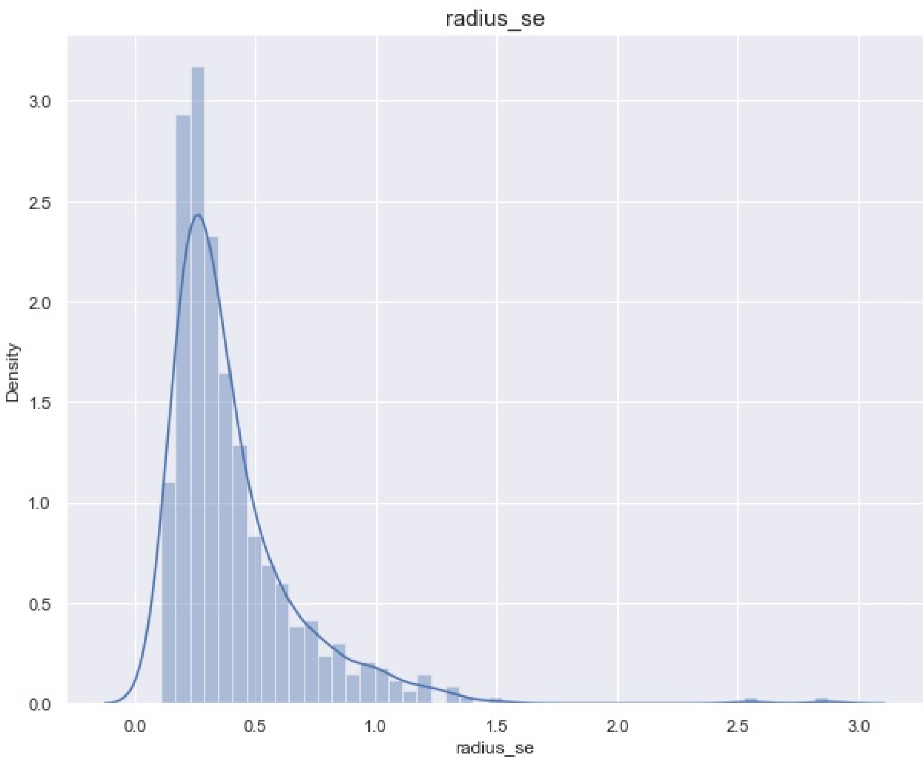

compactness_mean
Density
10
8
6
4
2
0
0.0
0.1
0.2
0.3
0.4
compactness_mean

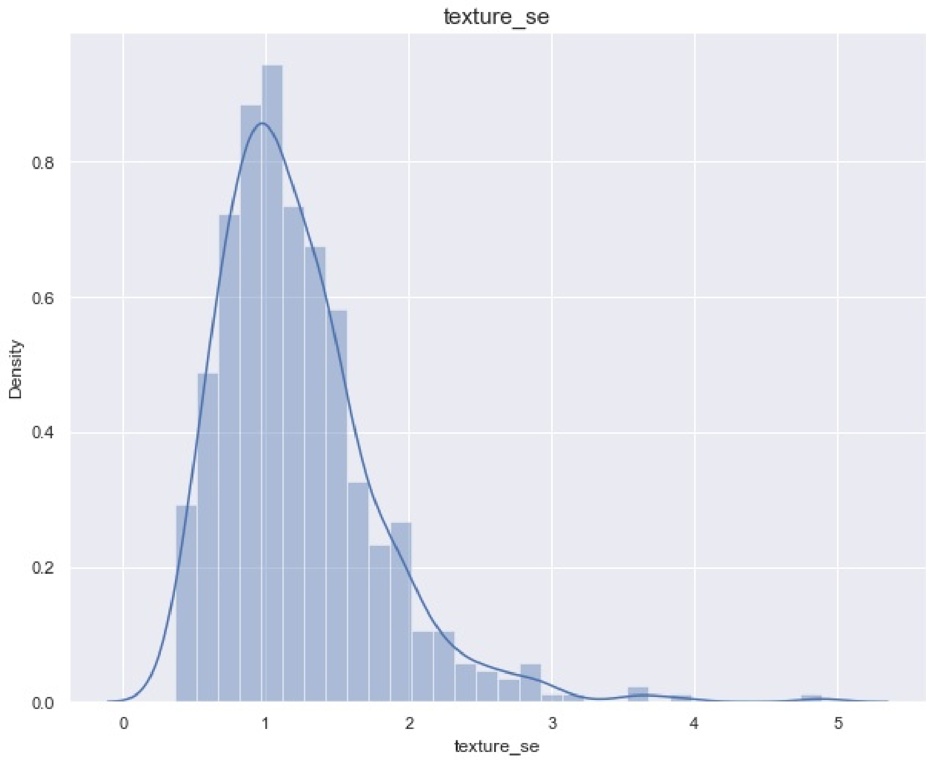

concavity_mean
Density
10
8
6
4
2
0
0.0
0.1
0.2
0.3
0.4
0.5
concavity_mean

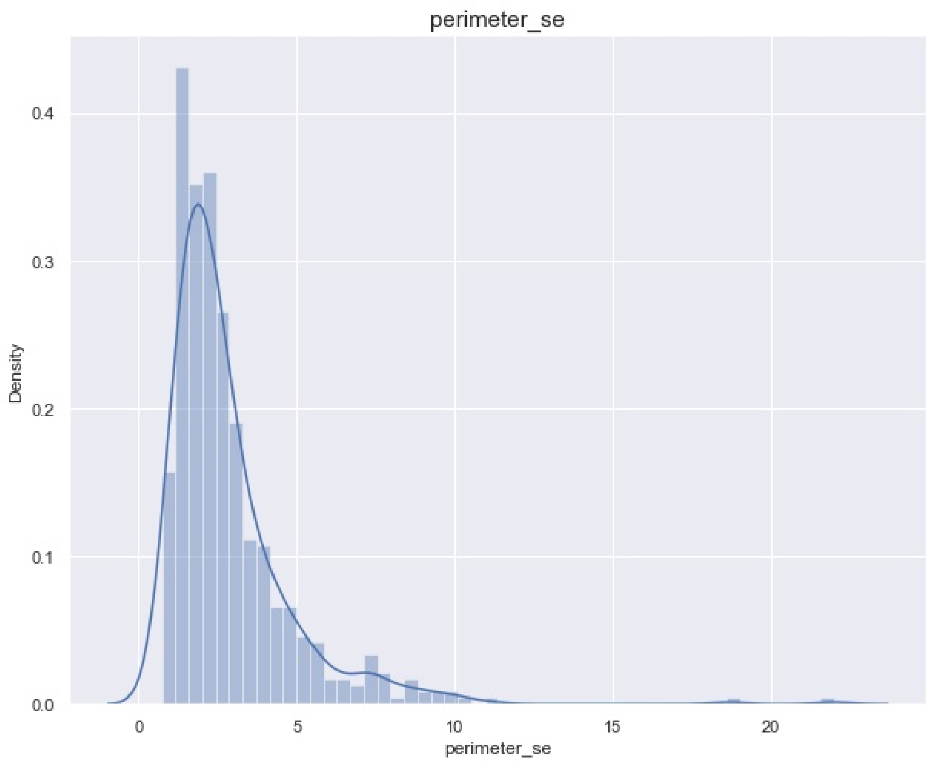

concave points_mean
Density
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0.00
0.05
0.10
0.15
0.20
concave points_mean

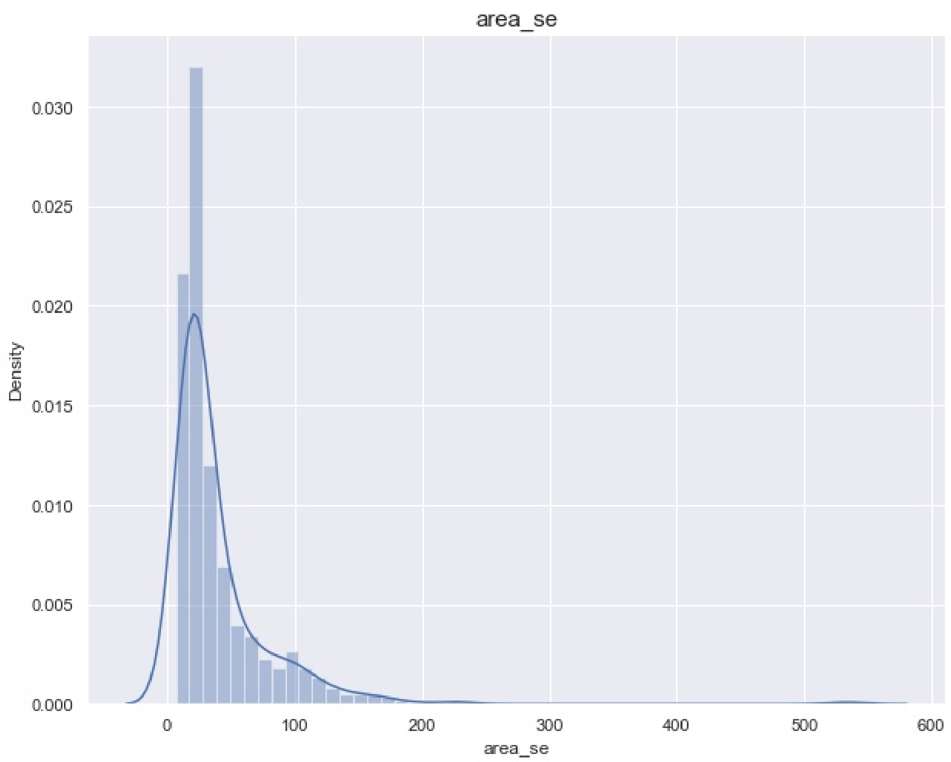

symmetry_mean
Density
16
14
12
10
8
6
4
2
0
0.10
0.15
0.20
0.25
0.30
symmetry_mean

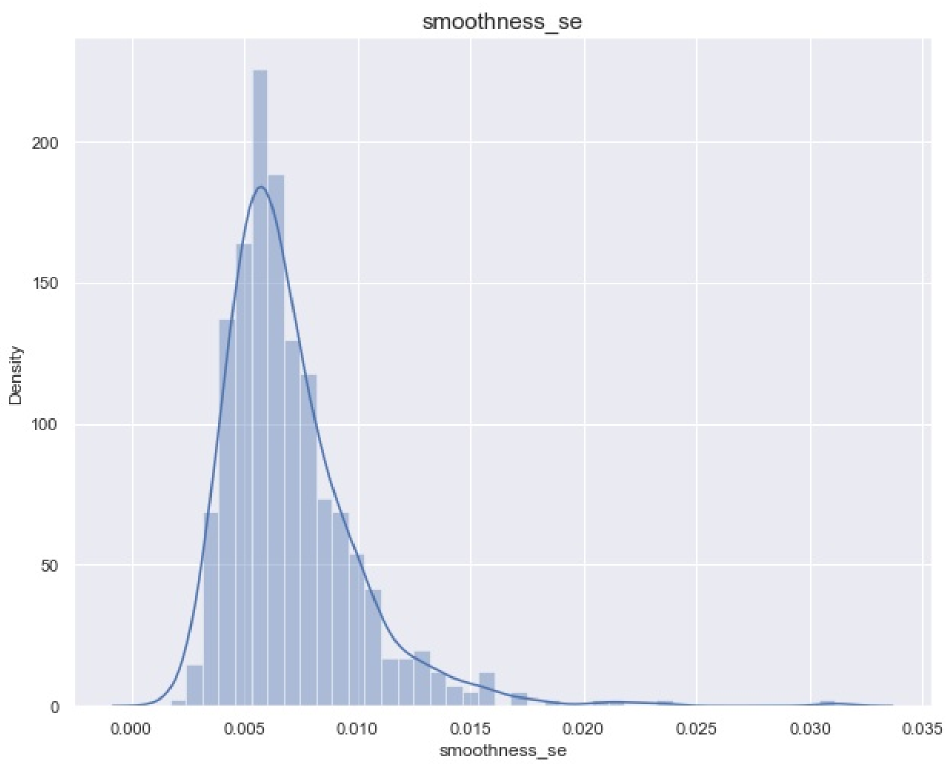

fractal_dimension_mean
Density
0
10
20
30
40
50
60
70
0.05
0.06
0.07
0.08
0.09
0.10
fractal_dimension_mean

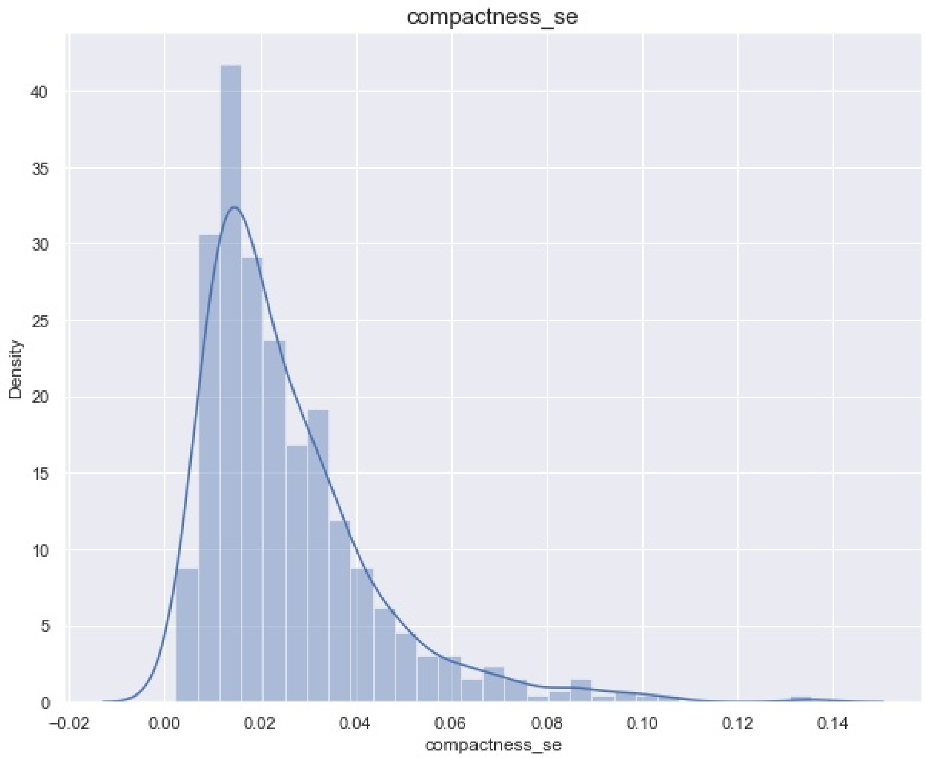

radius_se
Density
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
radius_se

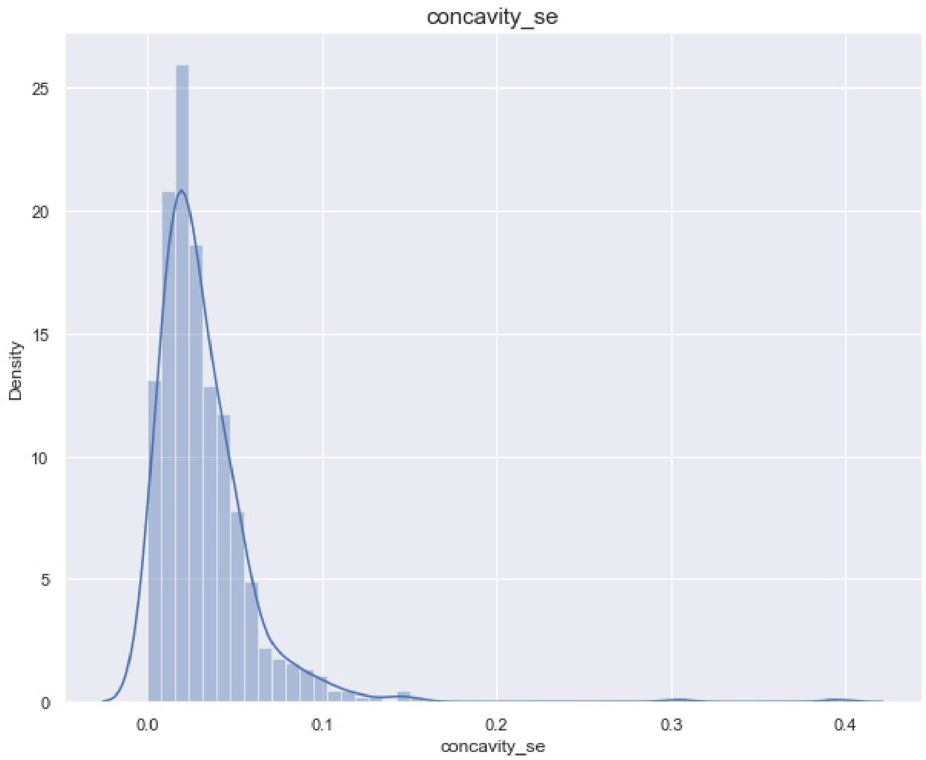

texture_se
Density
0.0
0.2
0.4
0.6
0.8
0
1
2
3
4
5
texture_se

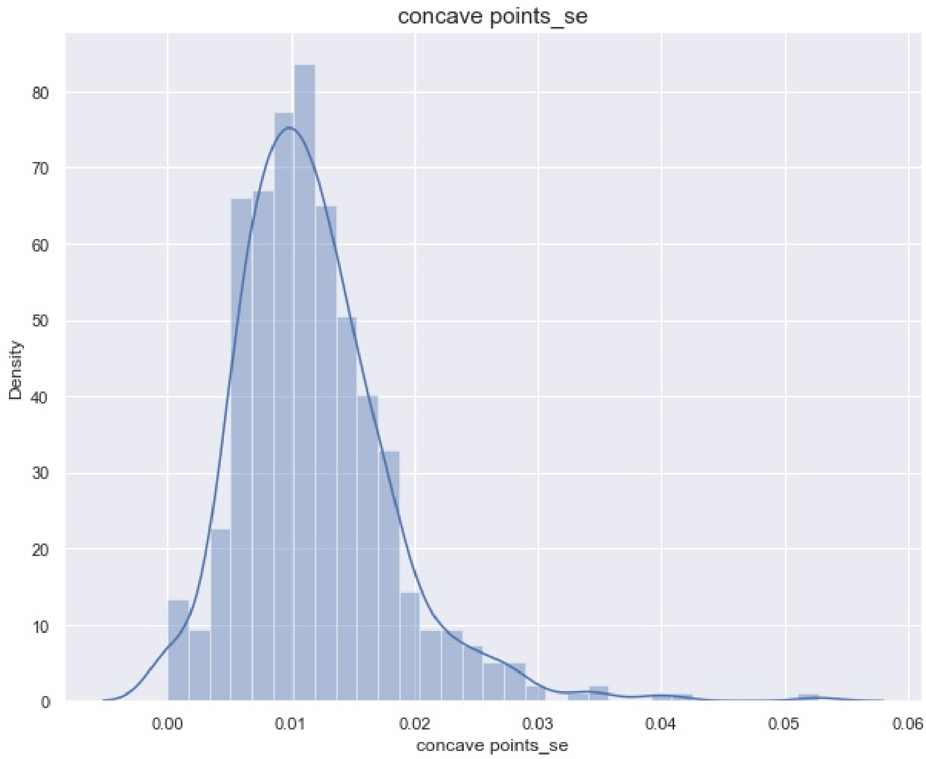

perimeter_se
Density
0.0
0.1
0.2
0.3
0.4
0
5
10
15
20
perimeter_se

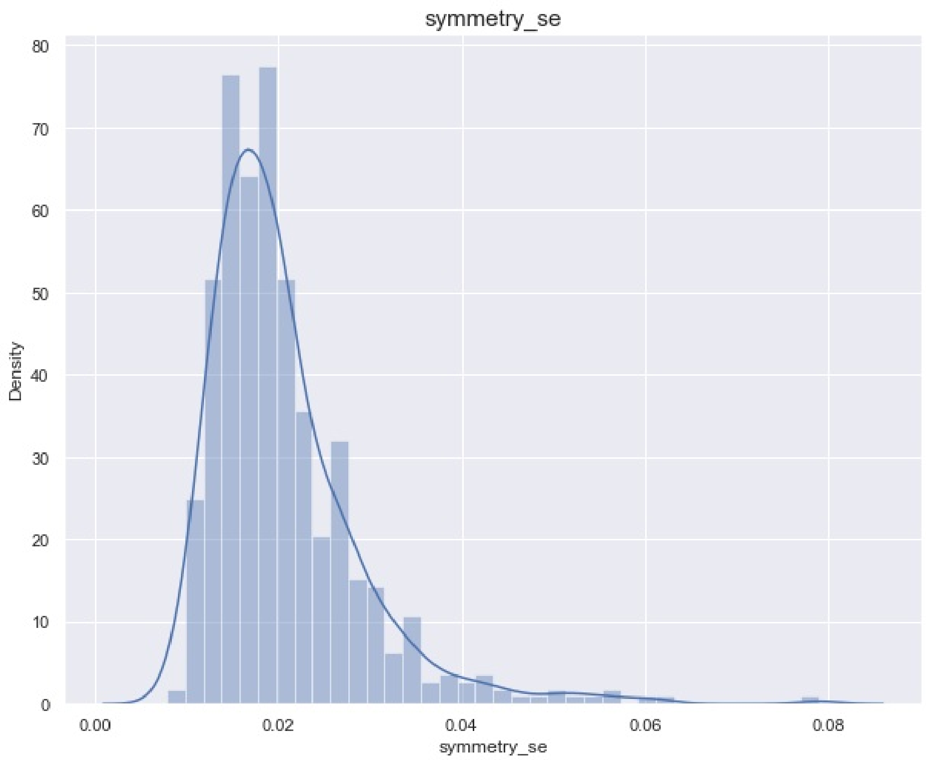

area_se
0.030
0.025
0.020
0.015
0.010
0.005
0.000
Density
0
100
200
300
400
500
600
area_se

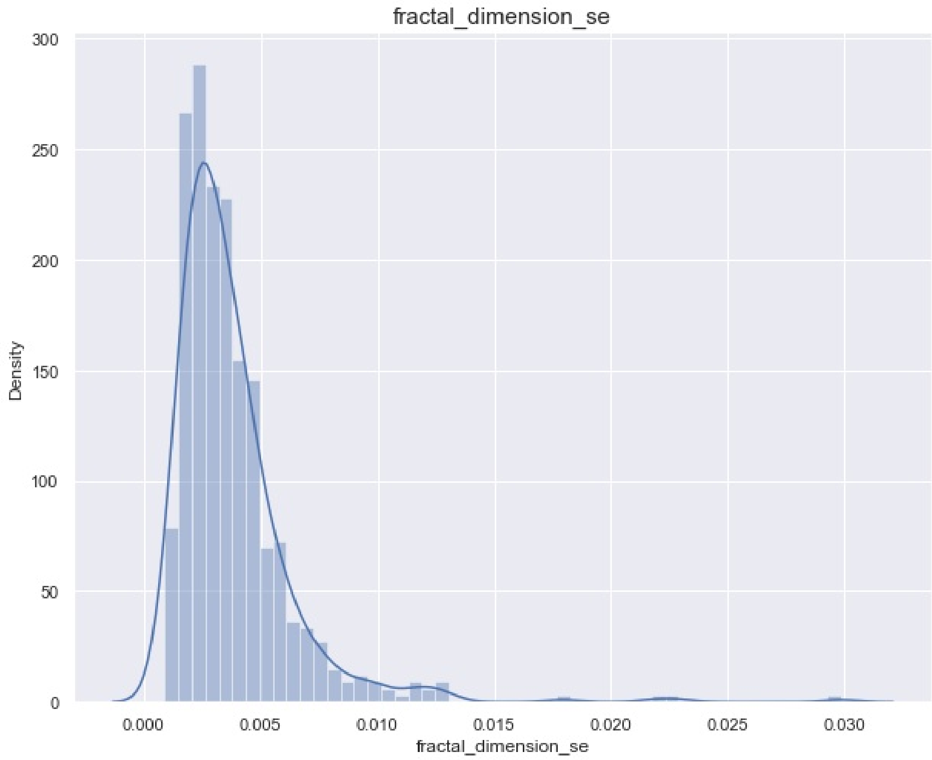

smoothness_se
200
150
100
50
0
Density
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035
smoothness_se

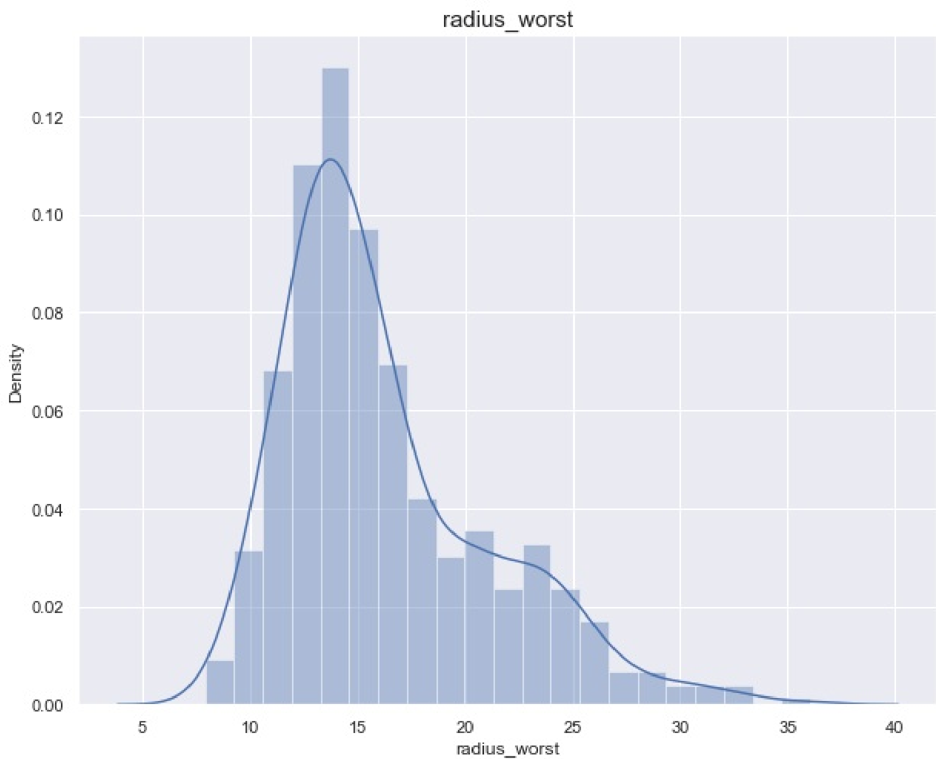

compactness_se
Density
0
5
10
15
20
25
30
35
40
−0.02
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
compactness_se

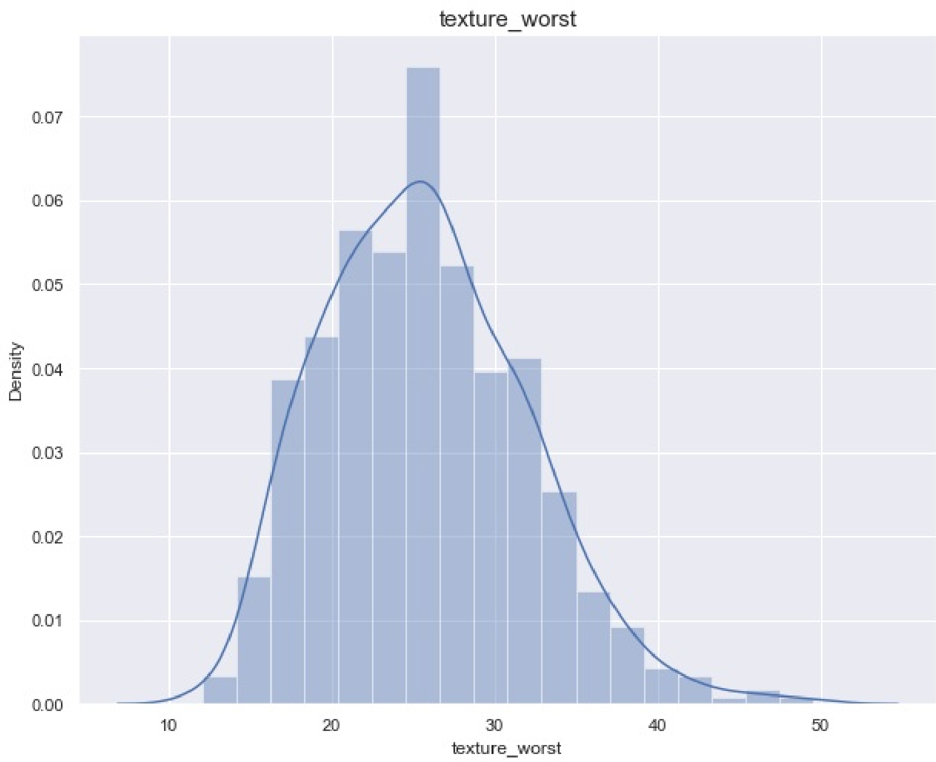

concavity_se
Density
0
5
10
15
20
25
0.0
0.1
0.2
0.3
0.4
concavity_se

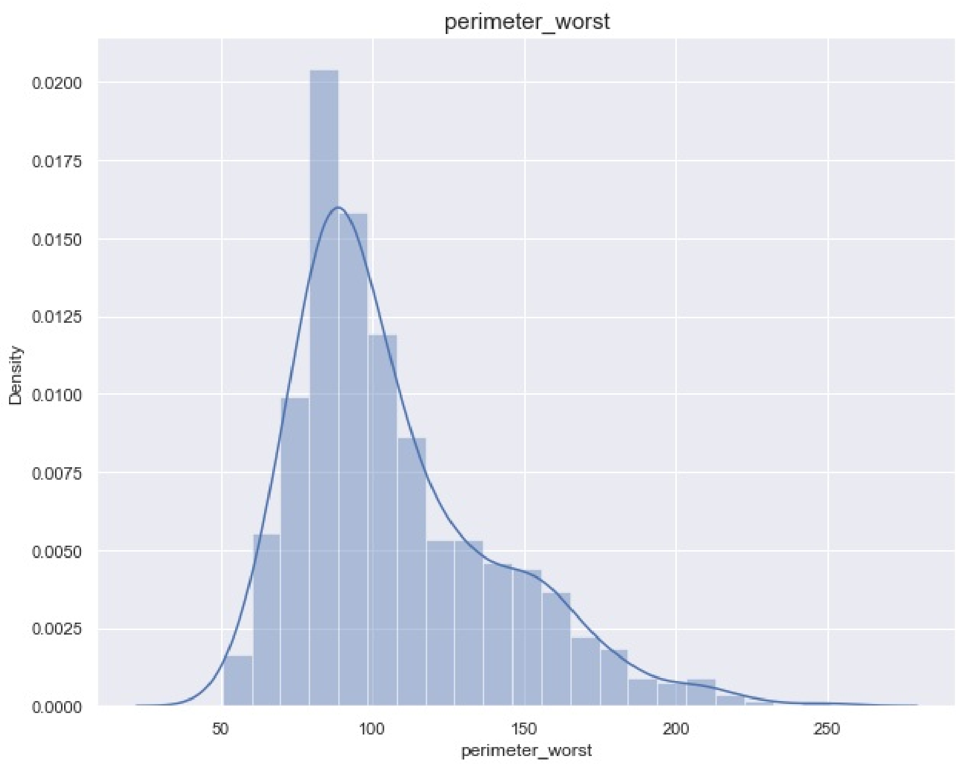

concave points_se
Density
0
10
20
30
40
50
60
70
80
0.00
0.01
0.02
0.03
0.04
0.05
0.06
concave points_se

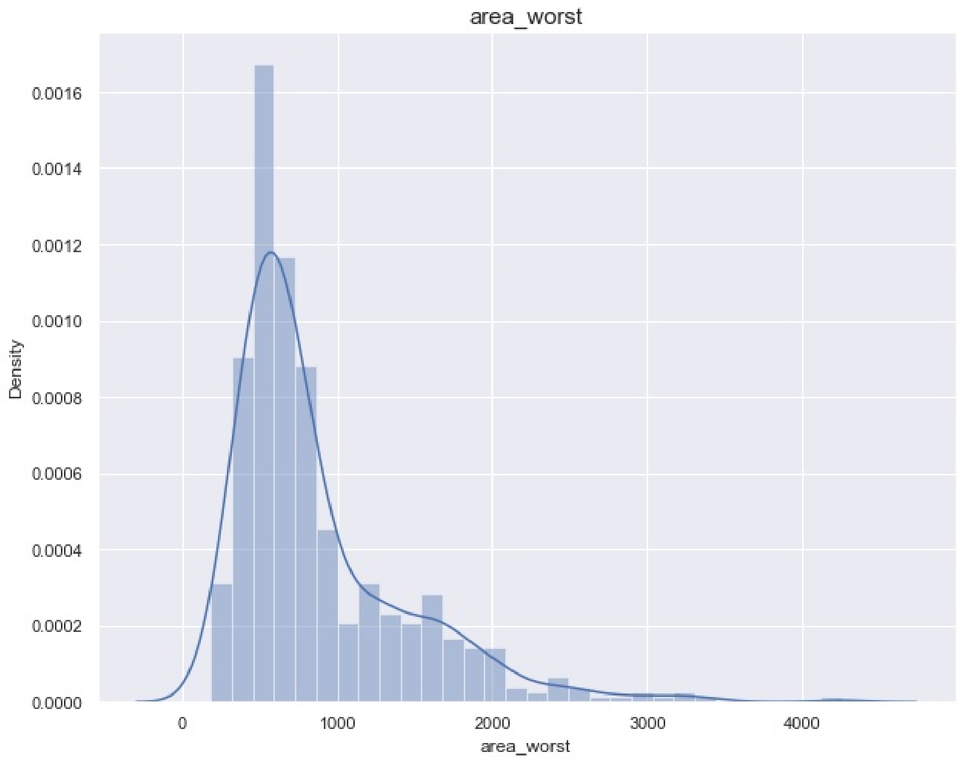

symmetry_se
Density
0
10
20
30
40
50
60
70
80
0.00
0.02
0.04
0.06
0.08
symmetry_se

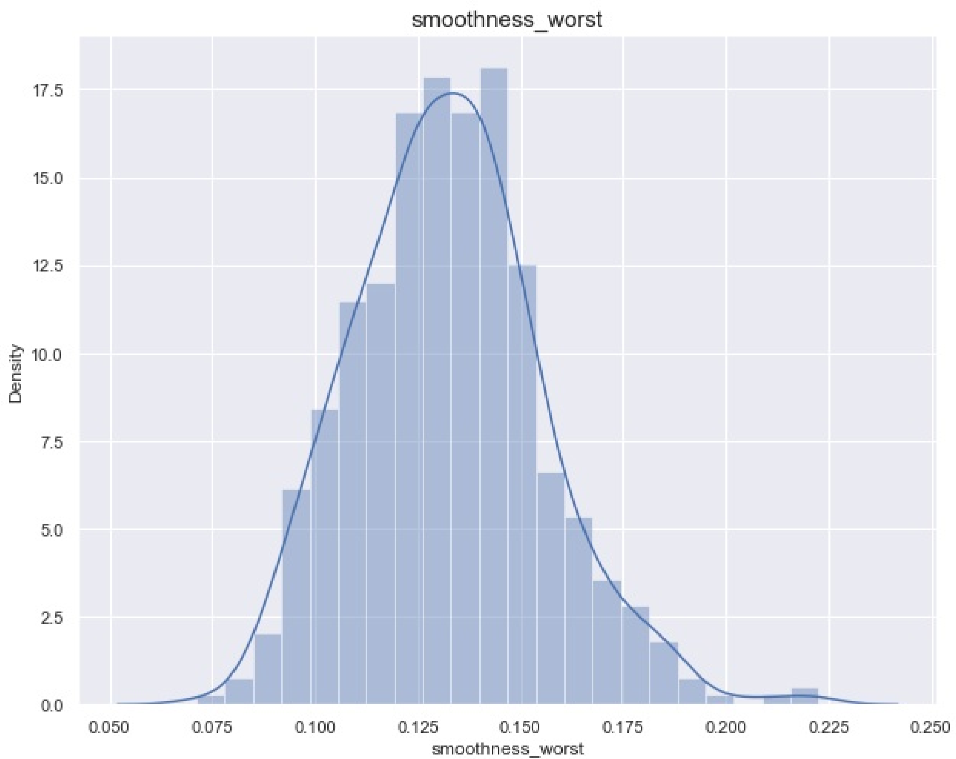

fractal_dimension_se
300
250
200
150
100
50
0
Density
0.000
0.005
0.010
0.015
0.020
0.025
0.030
fractal_dimension_se

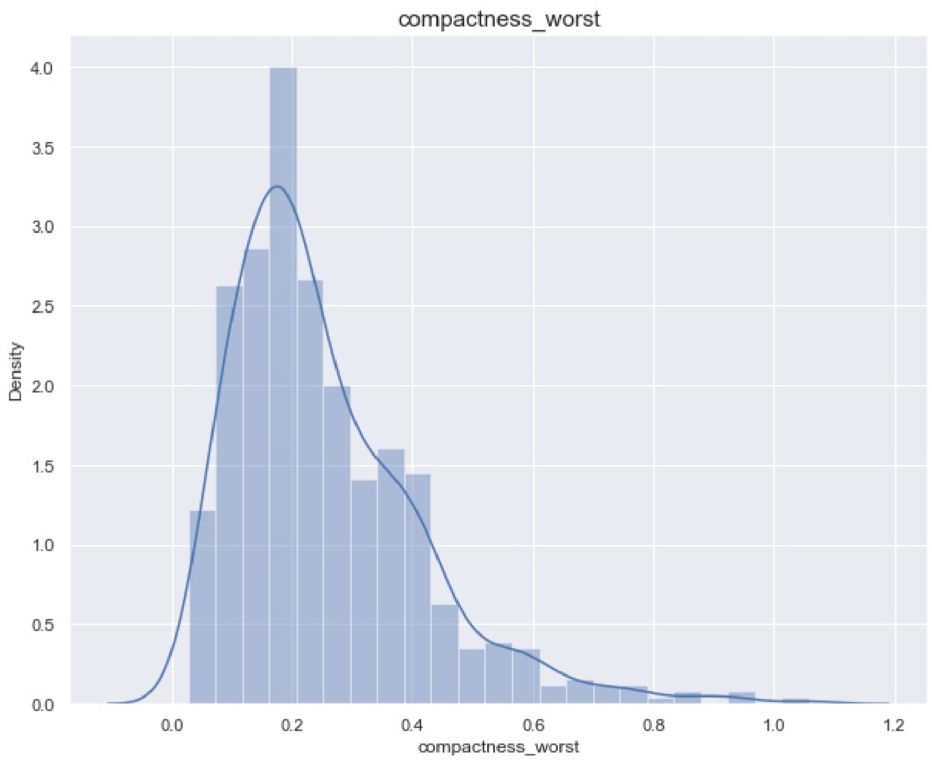

radius_worst
0.12
0.10
0.08
0.06
0.04
0.02
0.00
Density
5
10
15
20
25
30
35
40
radius_worst

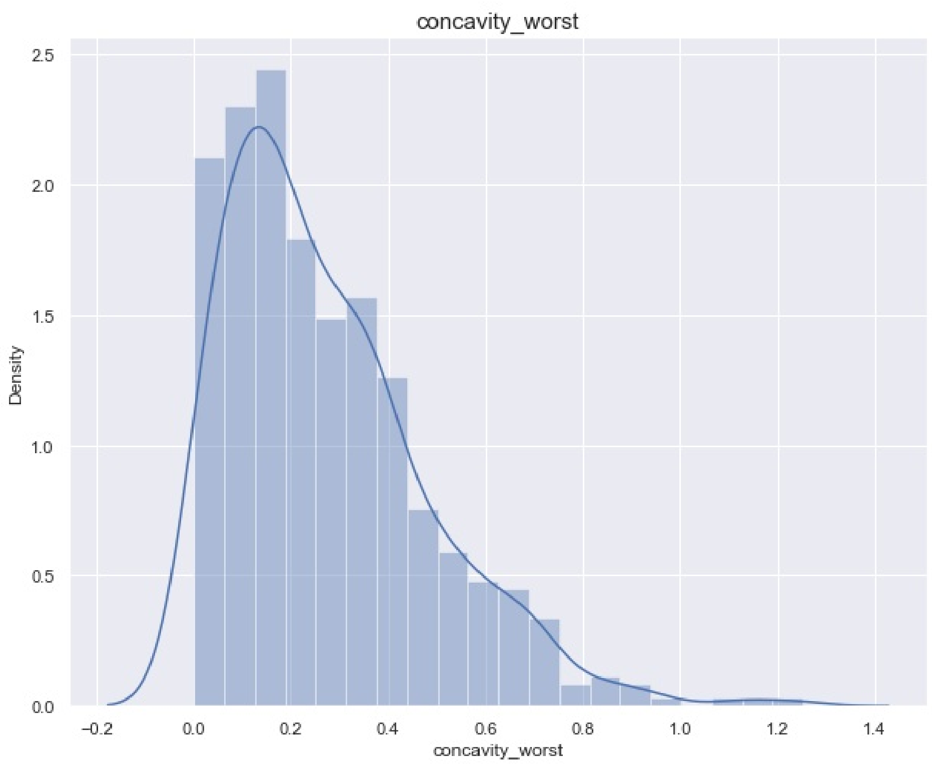

texture_worst
Density
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
10
20
30
40
50
texture_worst

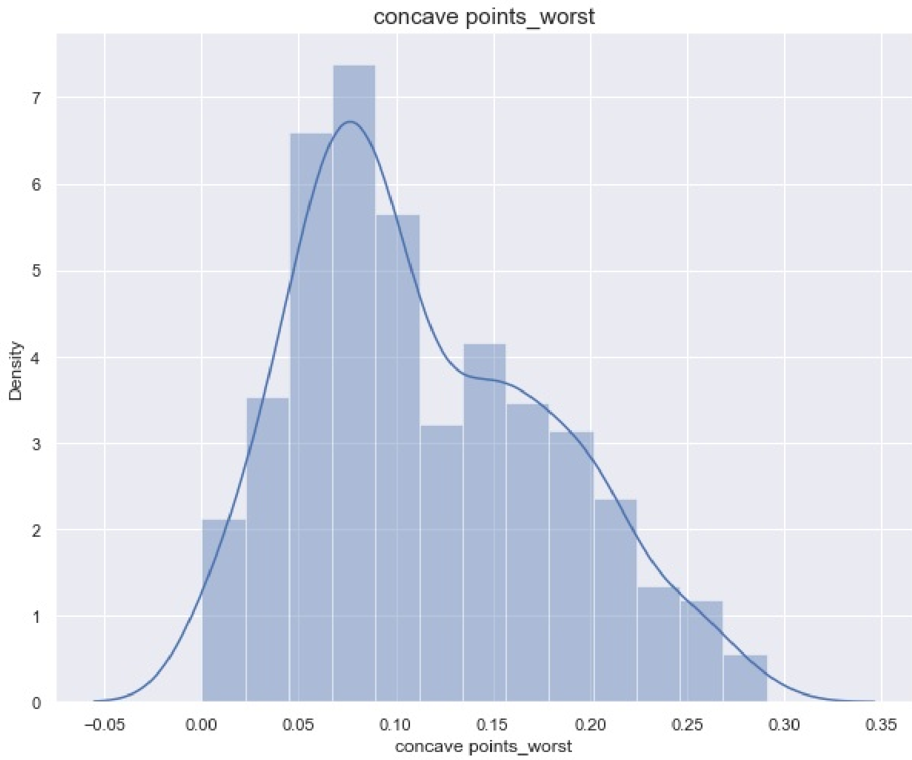

perimeter_worst
Density
0.0000
0.0025
0.0050
0.0075
0.0100
0.0125
0.0150
0.0175
0.0200
50
100
150
200
250
perimeter_worst

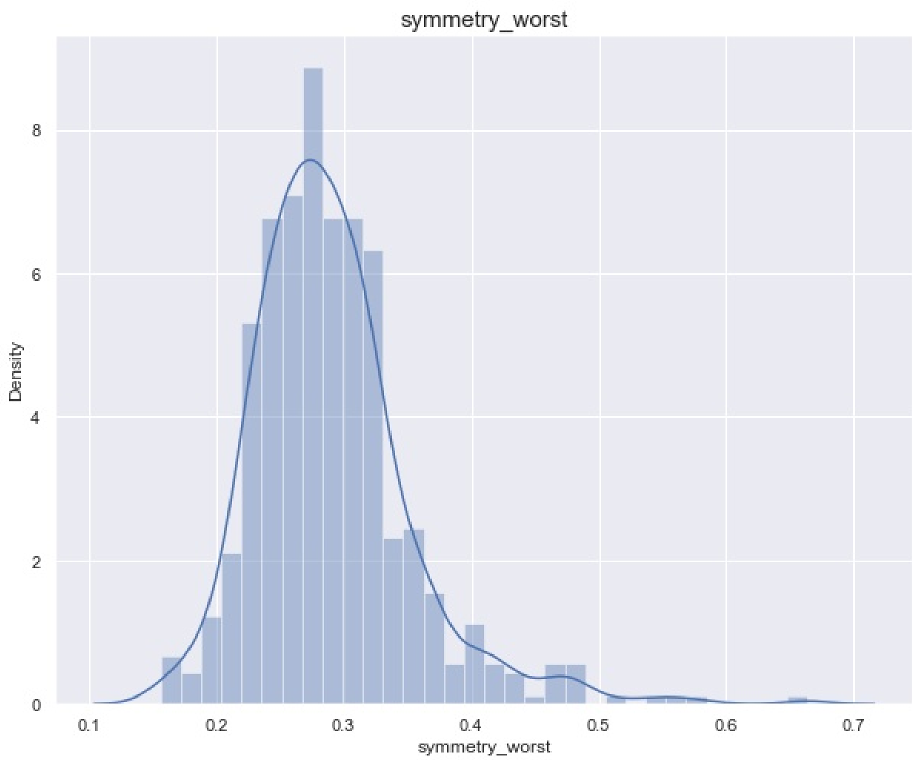

area_worst
0.0016
0.0014
0.0012
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
Density
0
1000
2000
3000
4000
area_worst

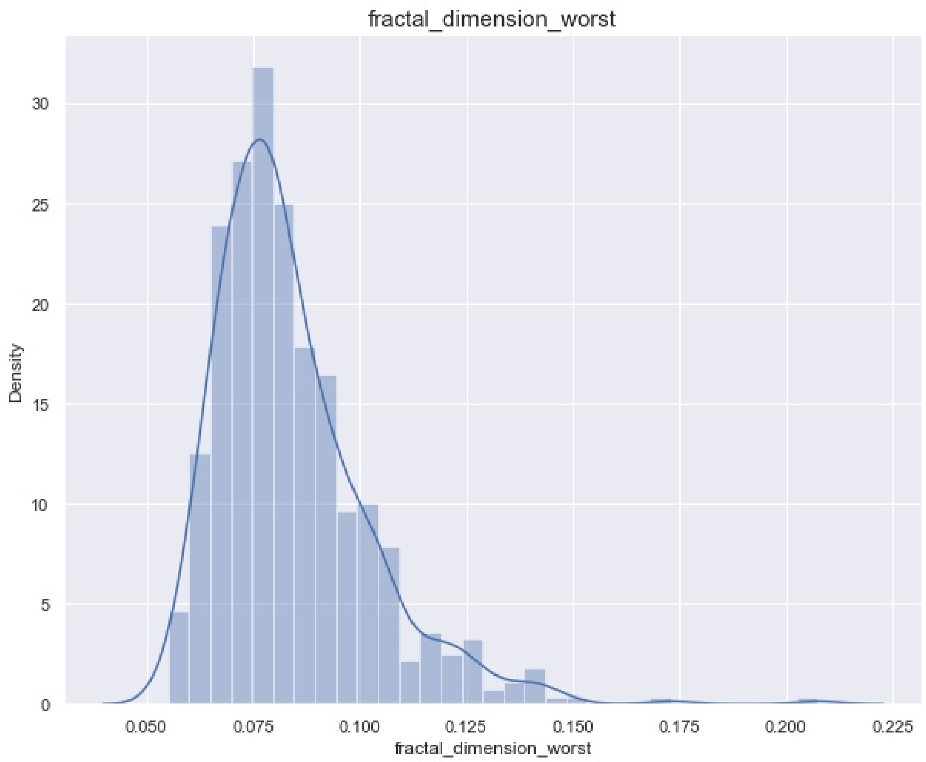

smoothness_worst
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
Density
0.050
0.075
0.100
0.125
0.150
0.175
0.200
0.225
0.250
smoothness_worst


compactness_worst
Density
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
compactness_worst

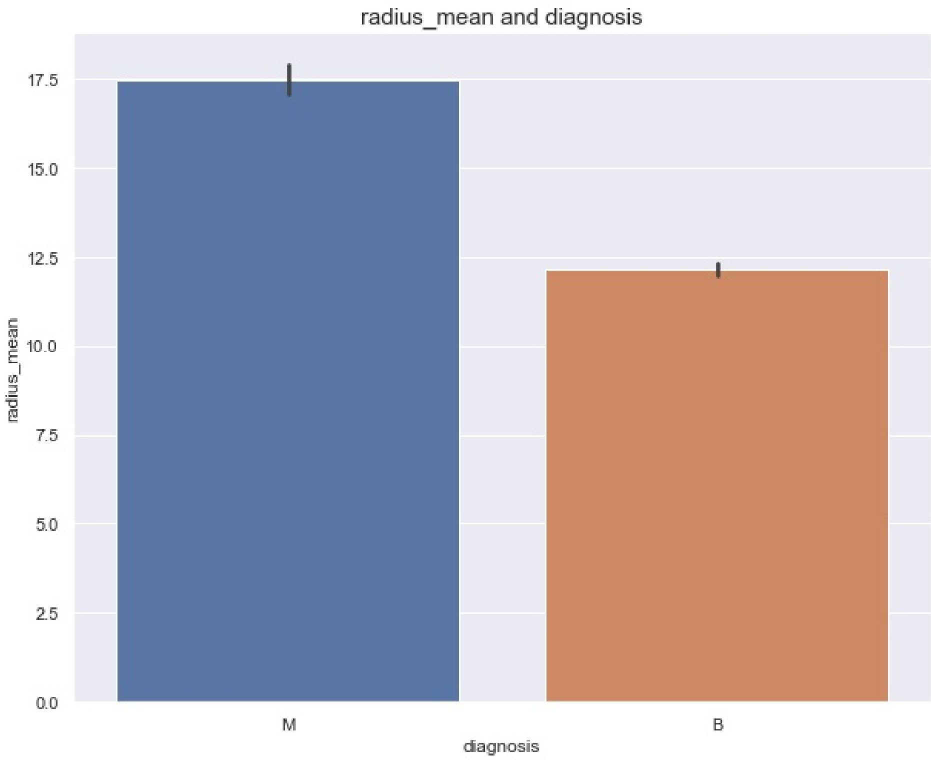

concavity_worst
Density
2.5
2.0
1.5
1.0
0.5
0.0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
concavity_worst

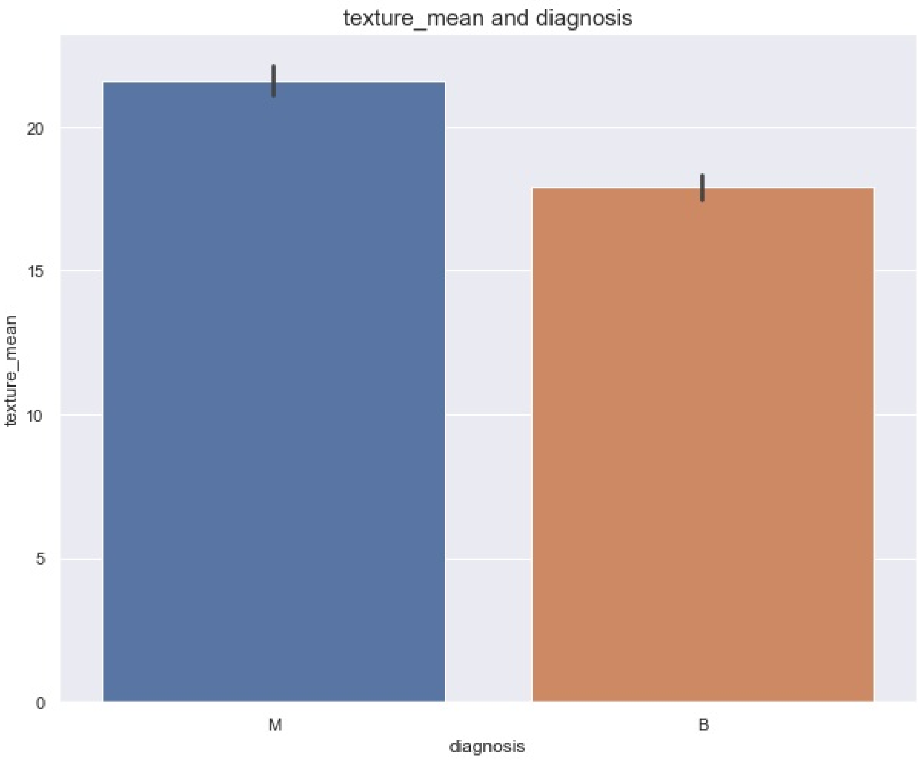

concave points_worst
Density
0
1
2
3
4
5
6
7
−0.05
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
concave points_worst


symmetry_worst
Density
0
2
4
6
8
0.1
0.2
0.3
0.4
0.5
0.6
0.7
symmetry_worst

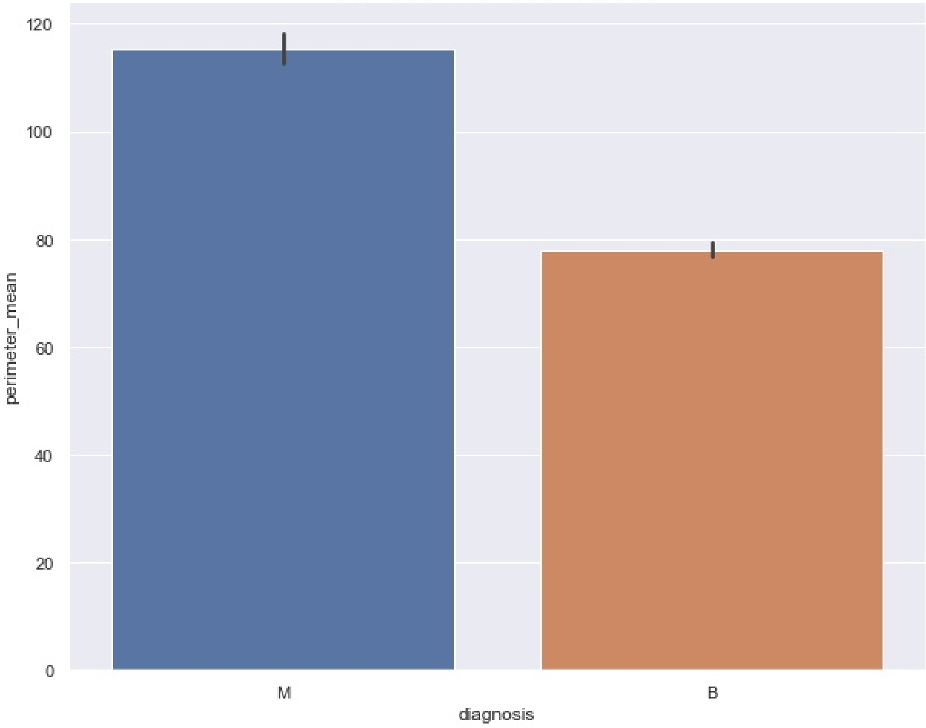


In [62]:
```
for col in df.drop("diagnosis", axis=1).columns:
    plt.figure(figsize=(10,8))
    sns.barplot(x=df["diagnosis"], y=df[col])
    plt.title(f"{col} and diagnosis", size=15)
    plt.show()
```

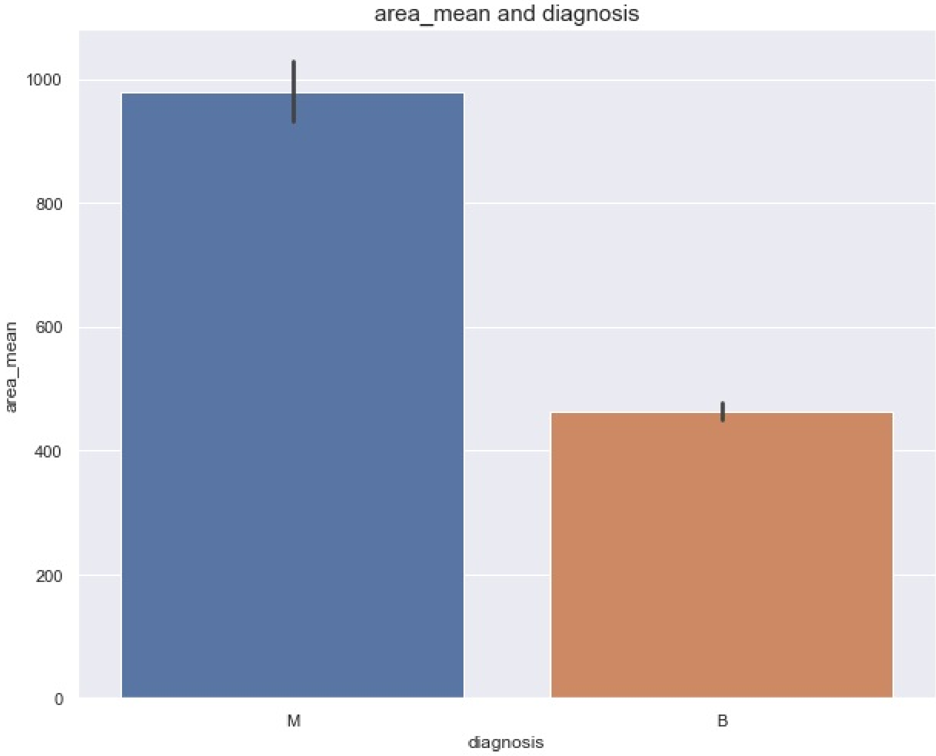

radius_mean and diagnosis
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
radius_mean
M
B
diagnosis

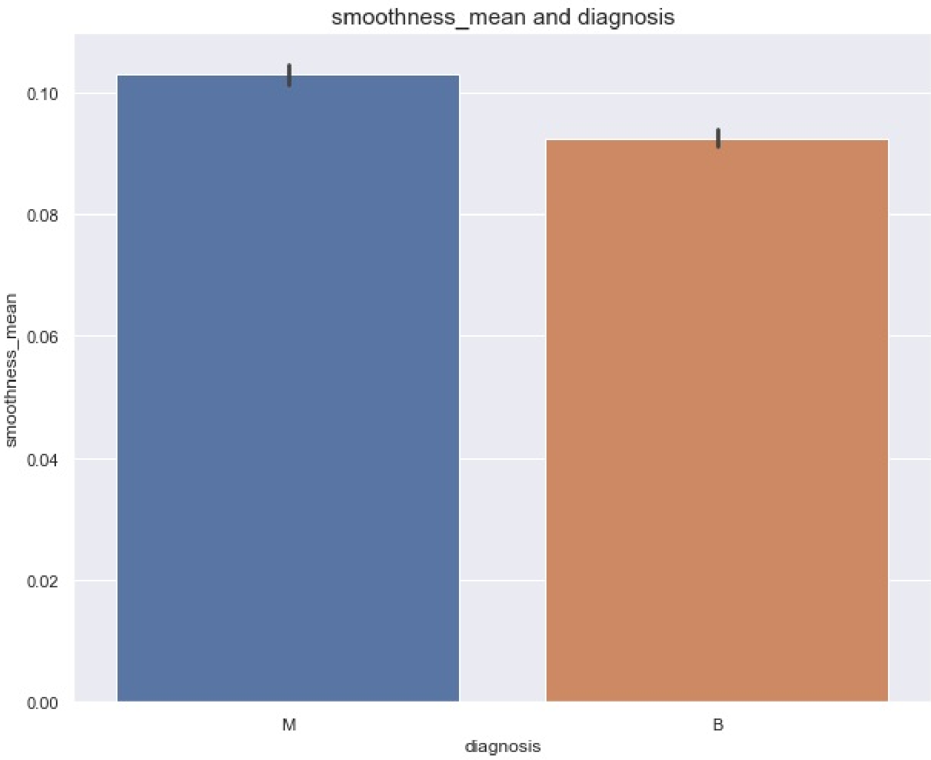

texture_mean and diagnosis
20
15
10
5
0
texture_mean
M
B
diagnosis

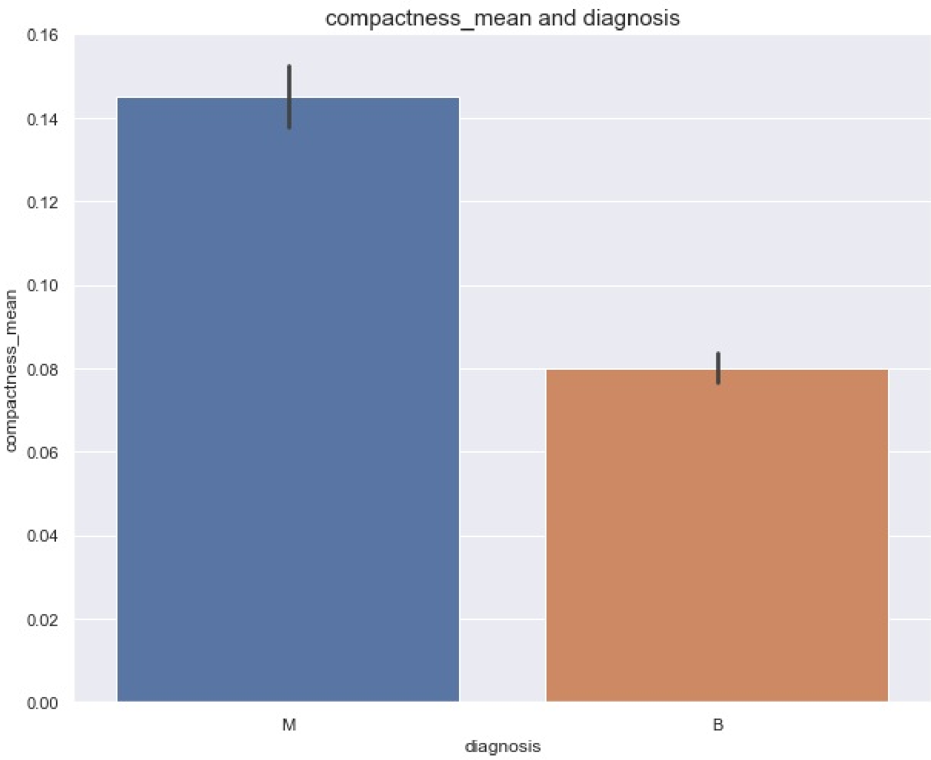
perimeter_mean and diagnosis

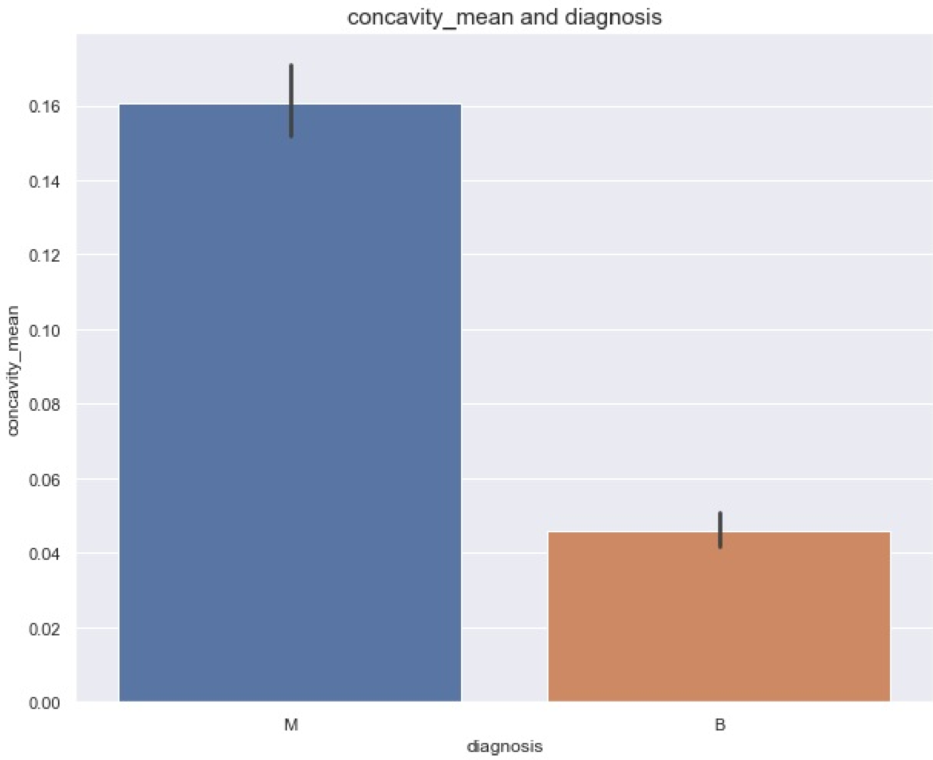


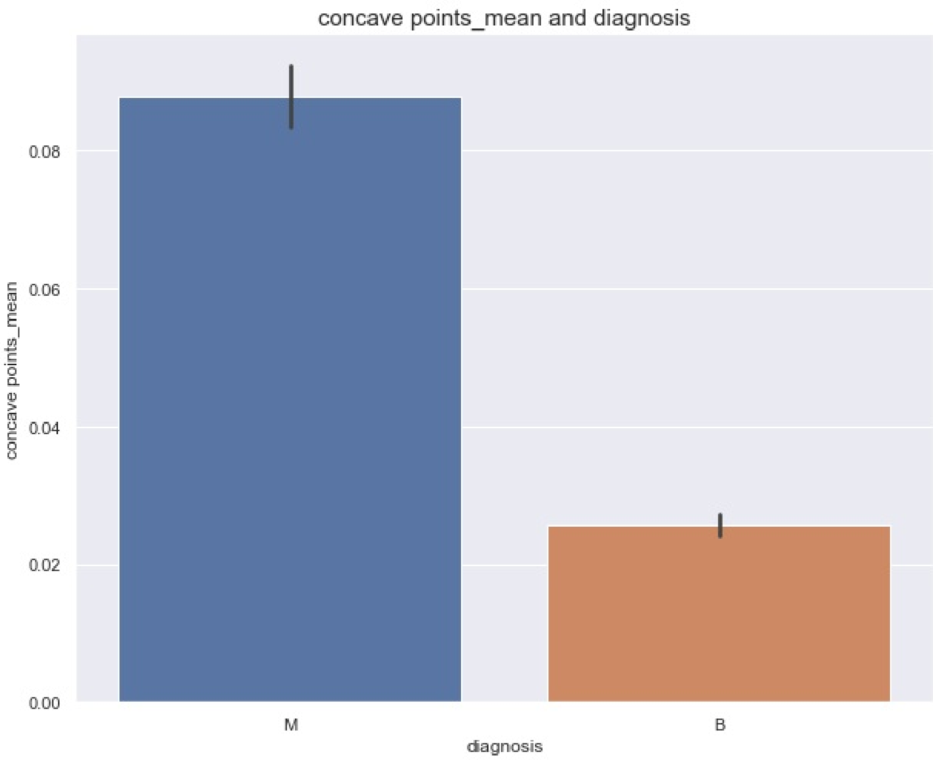
area_mean and diagnosis

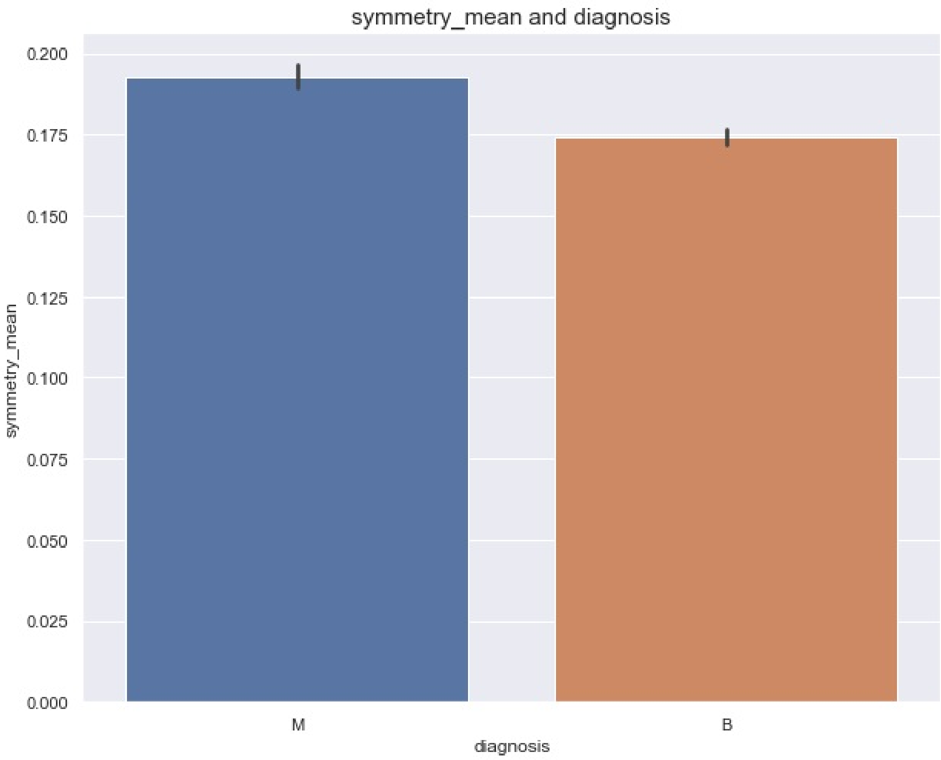


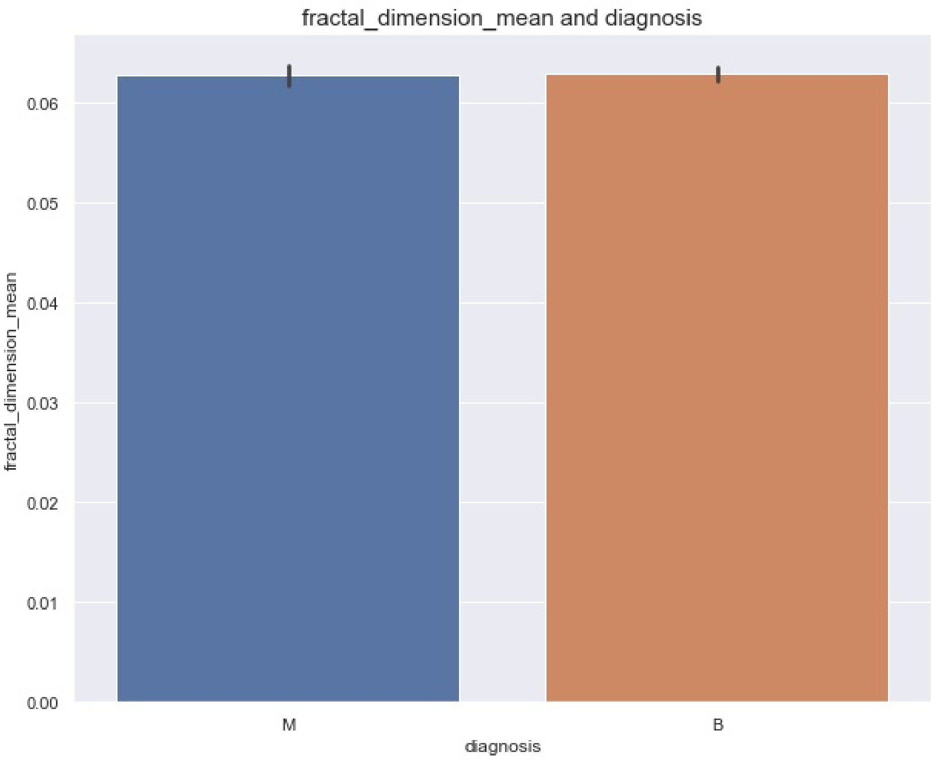

smoothness_mean and diagnosis
0.10
0.08
0.06
0.04
0.02
0.00
smoothness_mean
M
B
diagnosis

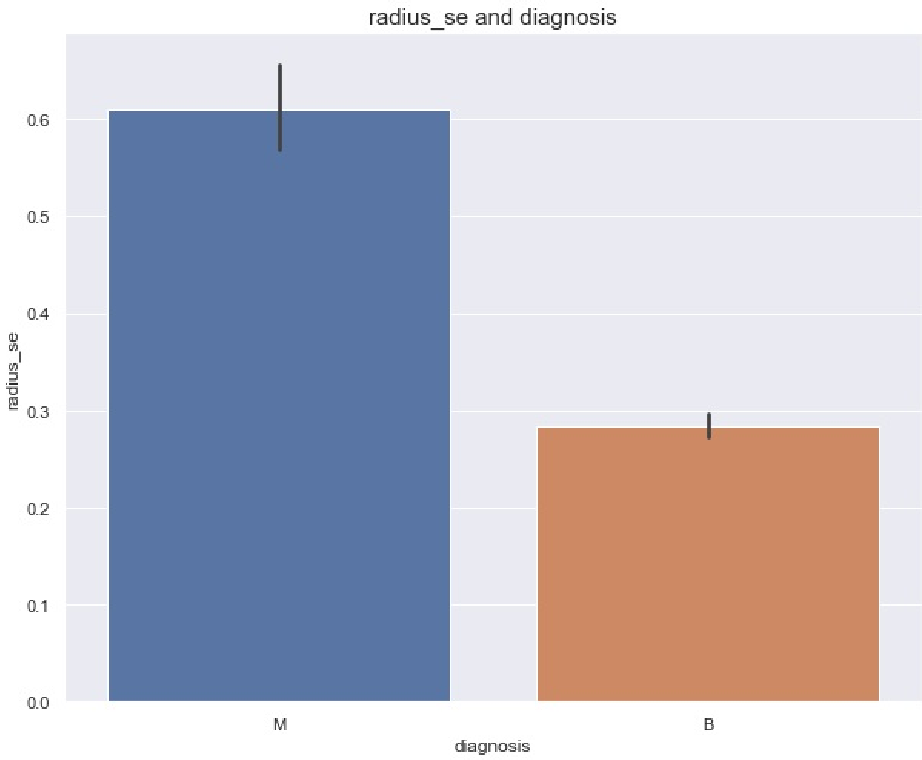

compactness_mean and diagnosis
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
compactness_mean
M
B
diagnosis

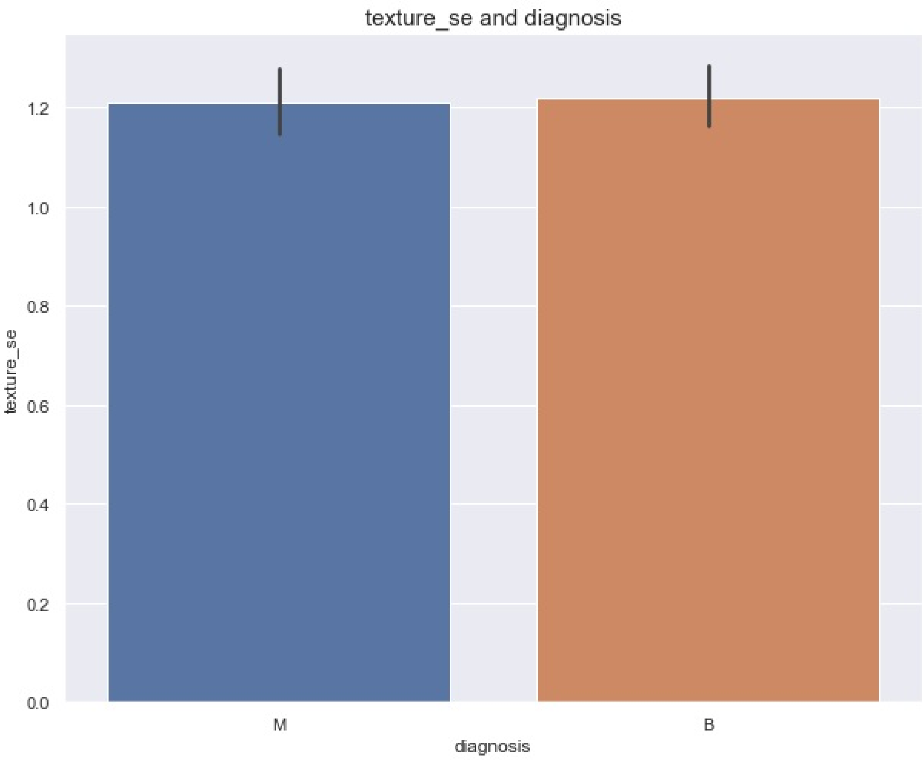

concavity_mean and diagnosis
concavity_mean
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
M
B
diagnosis

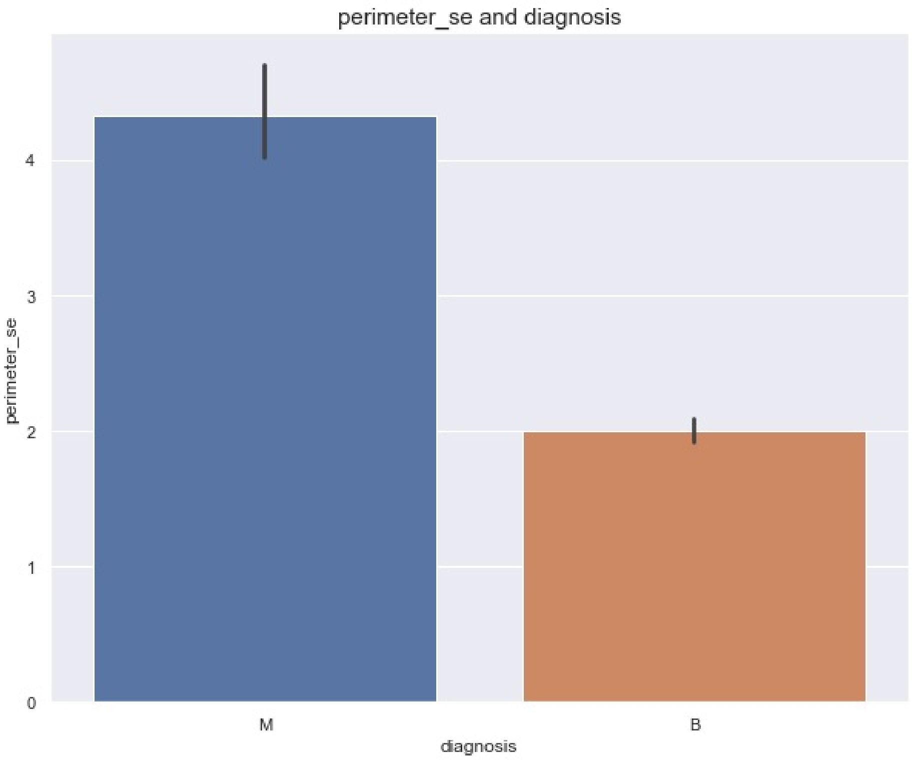

concave points_mean and diagnosis
concave points_mean
0.08
0.06
0.04
0.02
0.00
M
B
diagnosis

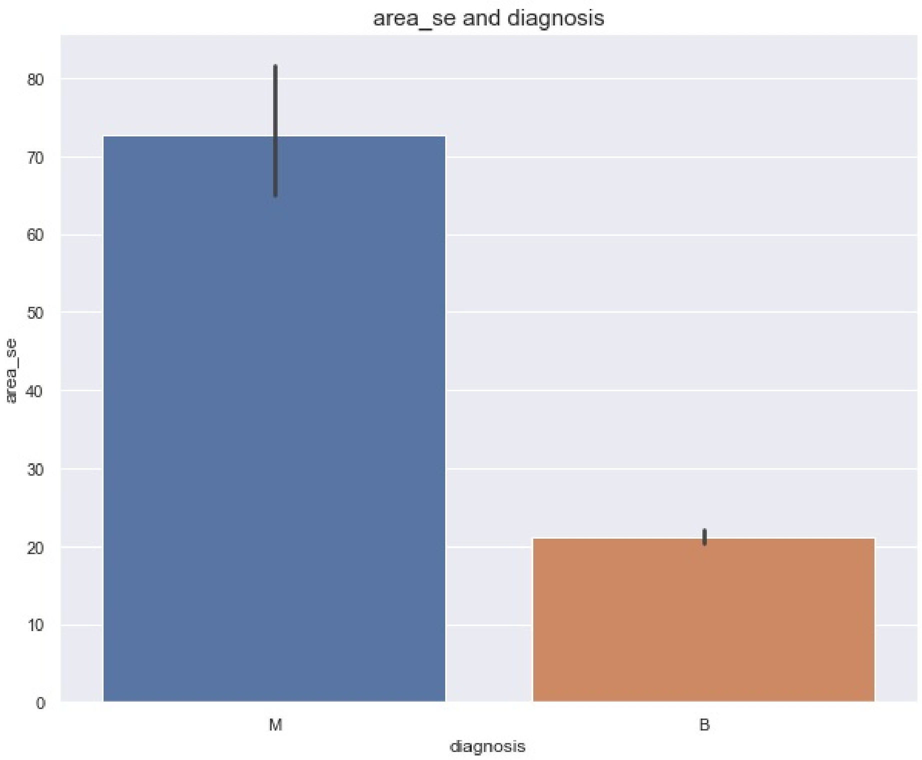

symmetry_mean and diagnosis
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
symmetry_mean
M
B
diagnosis

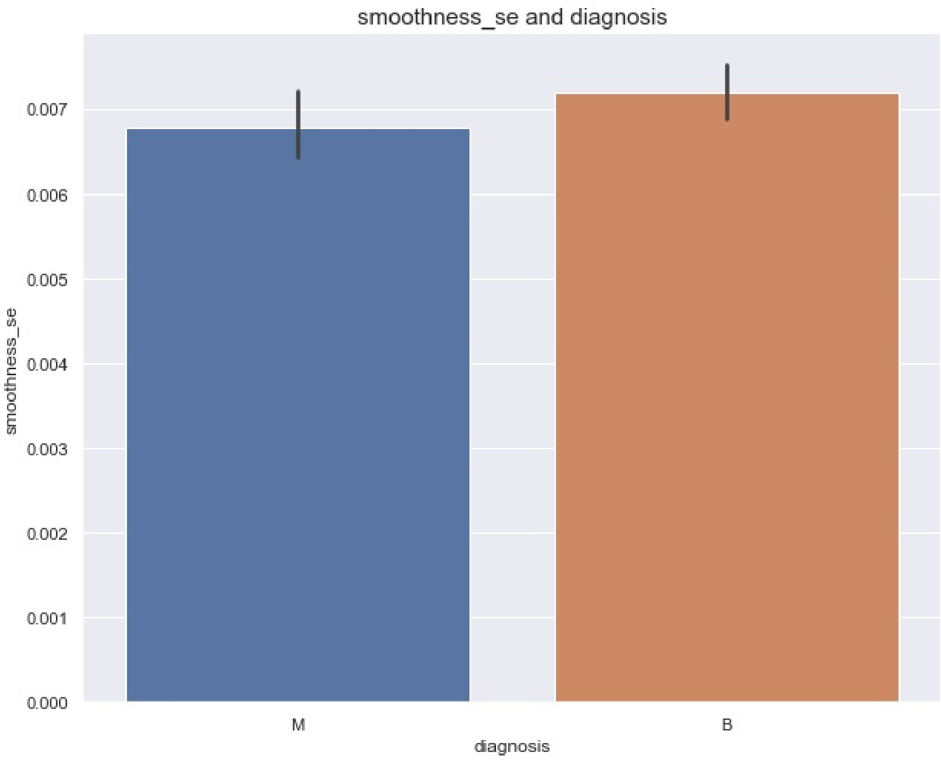

fractal_dimension_mean and diagnosis
0.06
0.05
0.04
0.03
0.02
0.01
0.00
fractal_dimension_mean
M
B
diagnosis

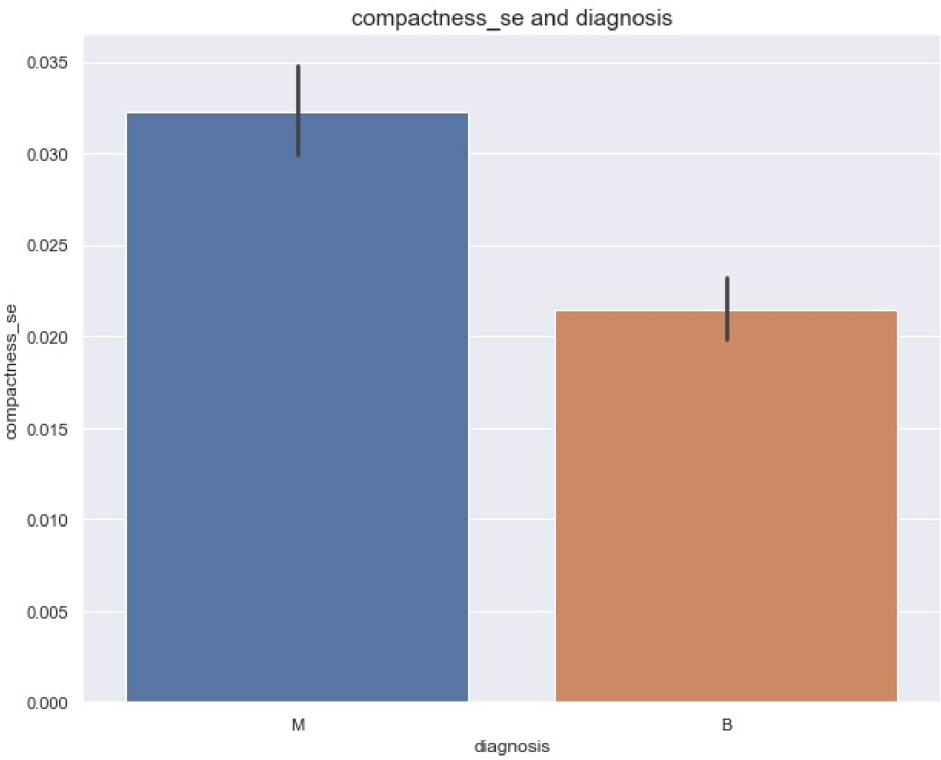

radius_se and diagnosis
radius_se
0.0
0.1
0.2
0.3
0.4
0.5
0.6
M
B
diagnosis

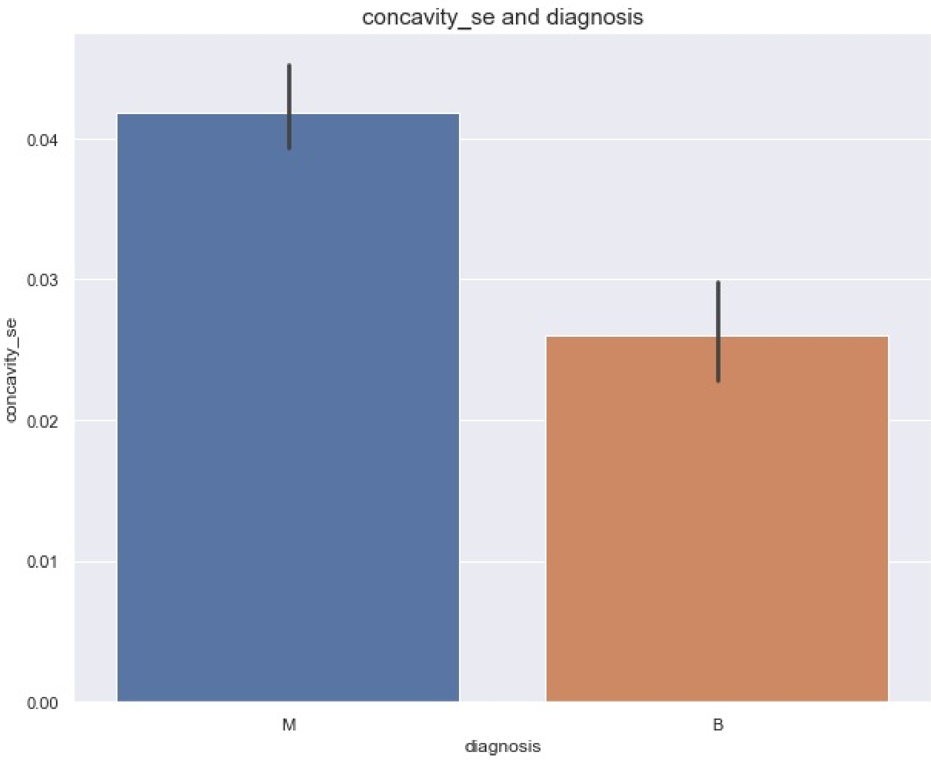

texture_se and diagnosis
texture_se
0.0
0.2
0.4
0.6
0.8
1.0
1.2
M
B
diagnosis

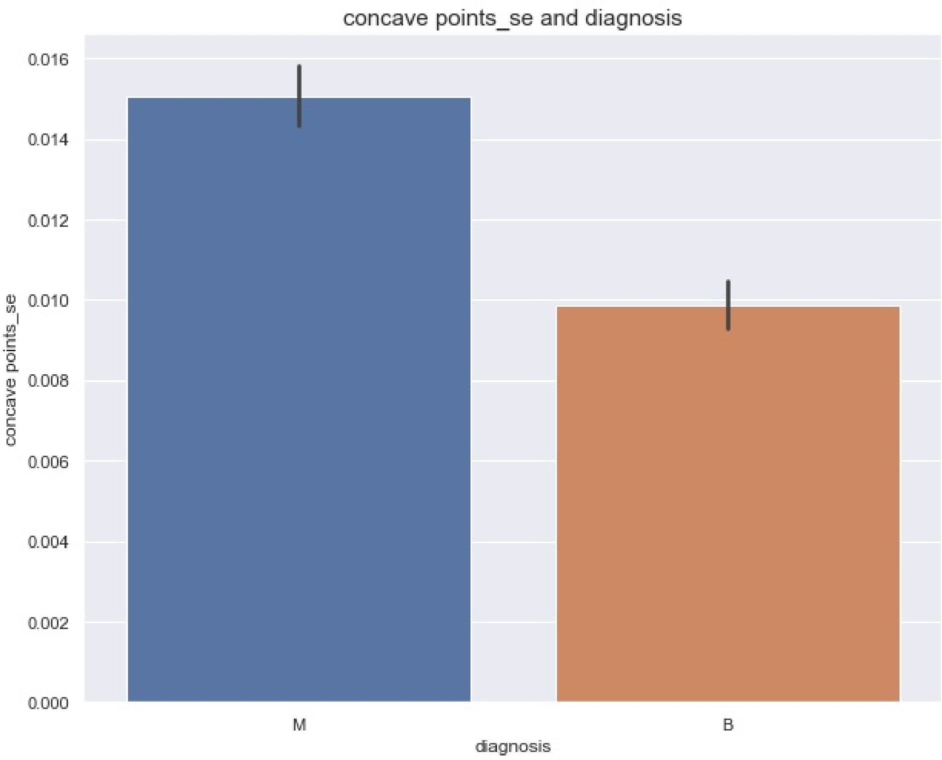

perimeter_se and diagnosis
perimeter_se
0
1
2
3
4
M
B
diagnosis

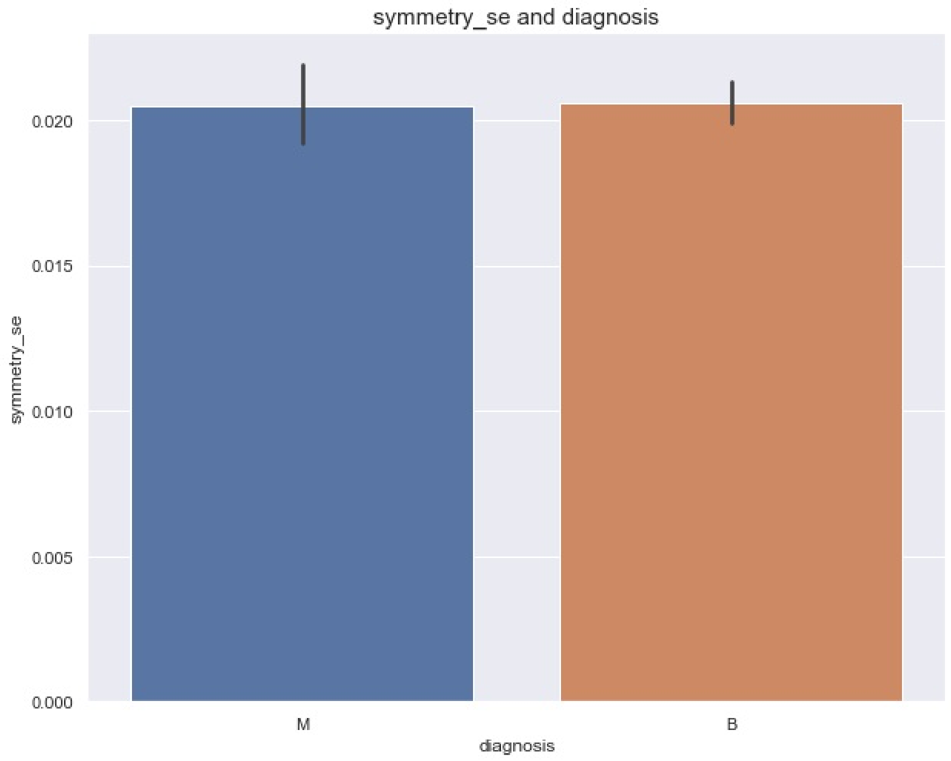

area_se and diagnosis
area_se
0
10
20
30
40
50
60
70
80
M
B
diagnosis

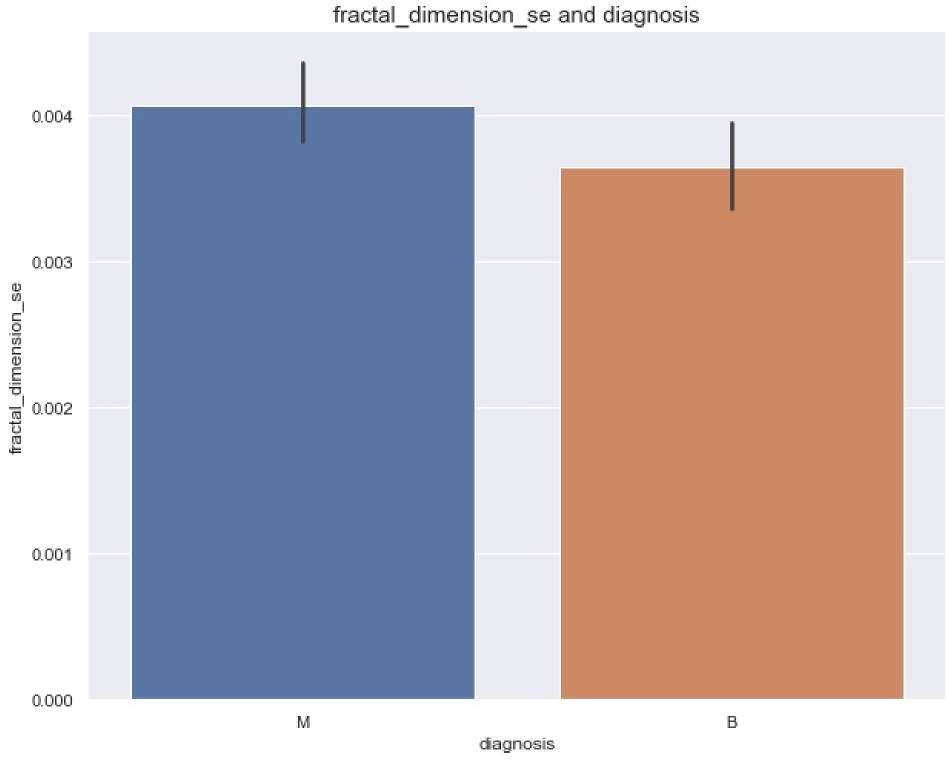

smoothness_se and diagnosis
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
smoothness_se
M
B
diagnosis

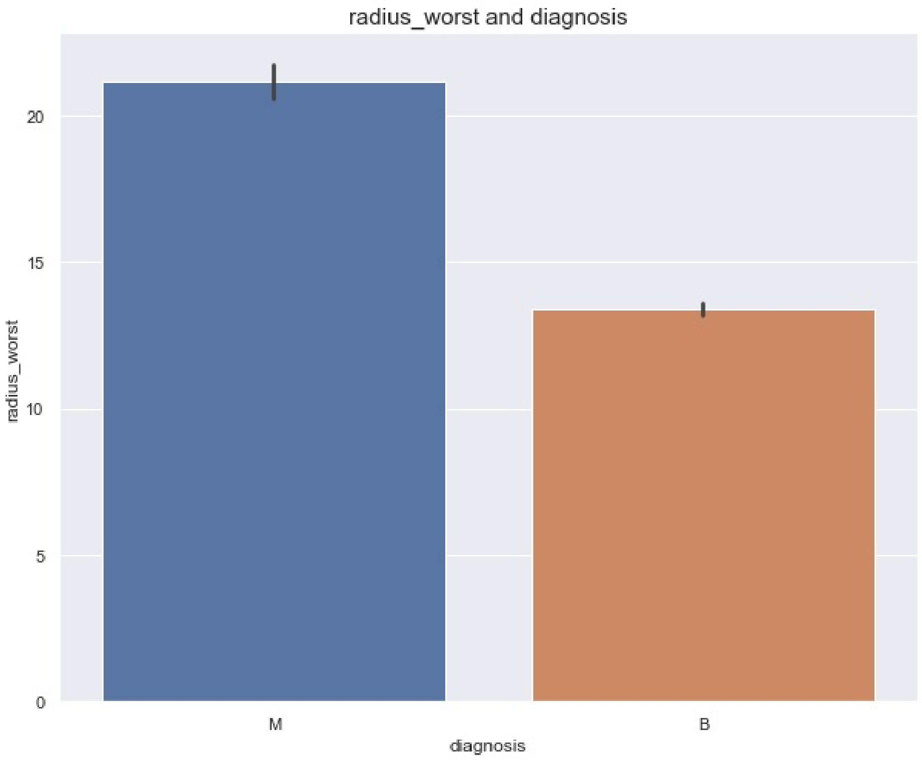

compactness_se and diagnosis
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
compactness_se
M
B
diagnosis

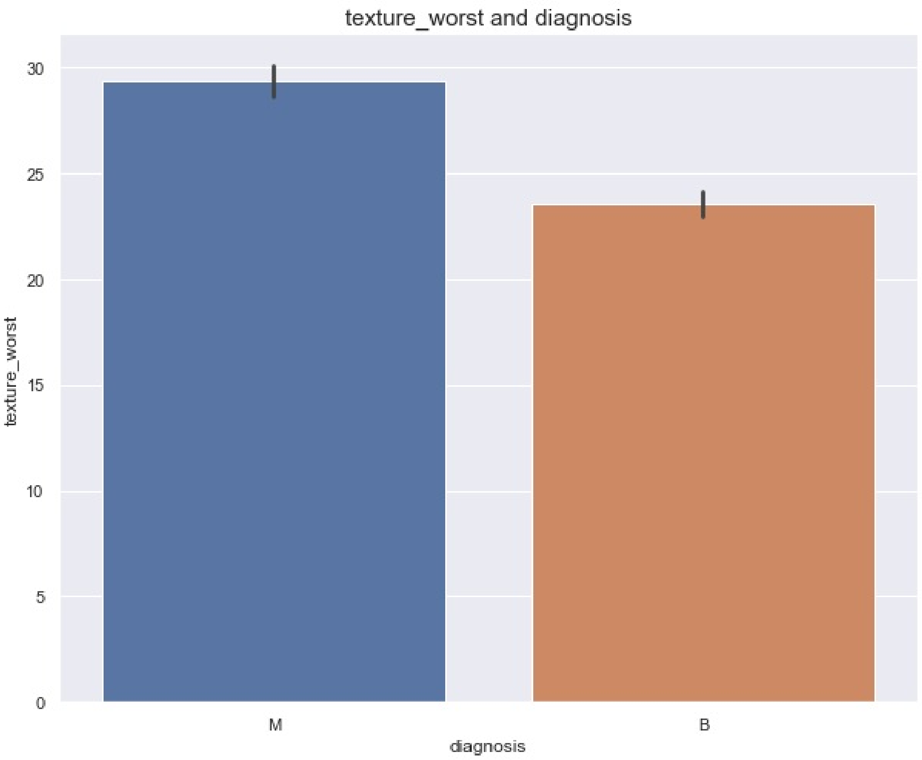

concavity_se and diagnosis
0.04
0.03
0.02
0.01
0.00
concavity_se
M
B
diagnosis

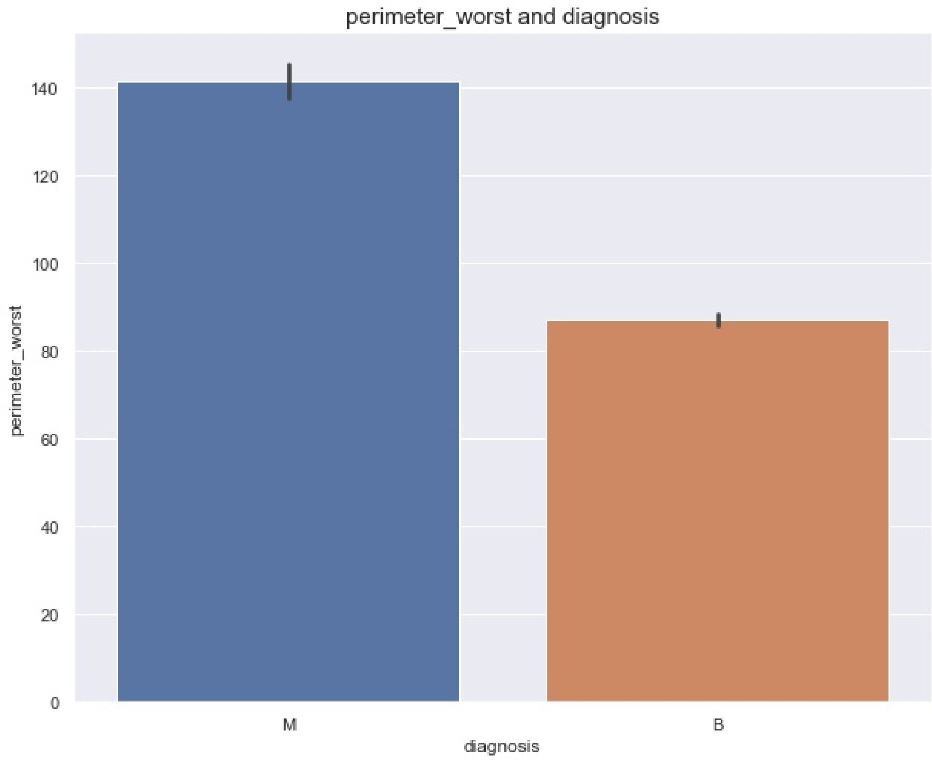

concave points_se and diagnosis
0.016
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
concave points_se
M
B
diagnosis

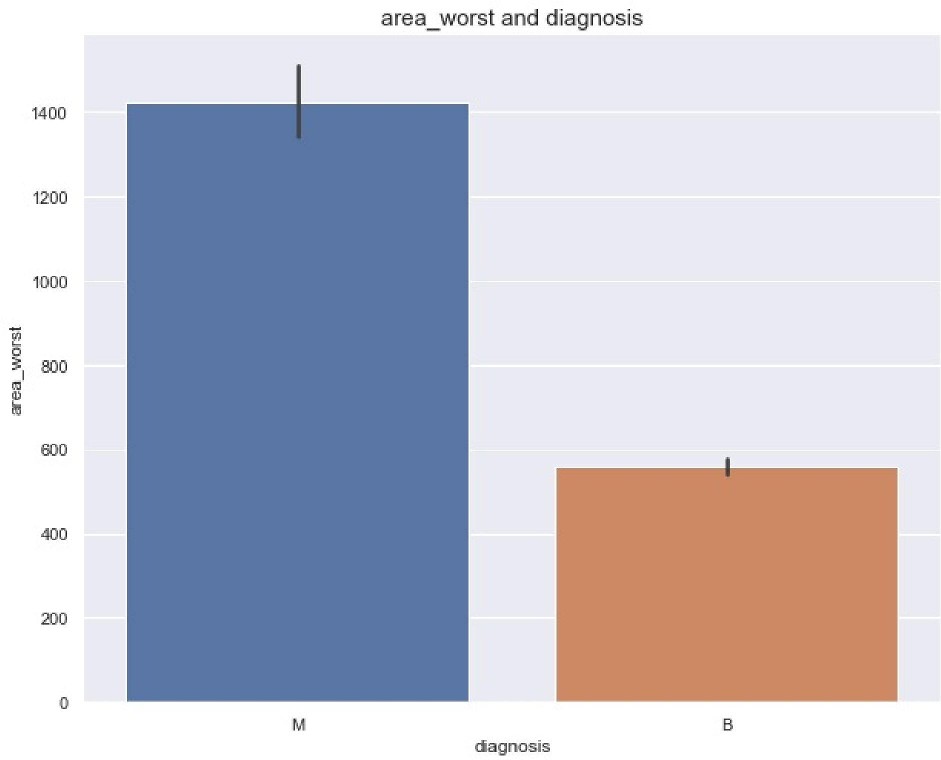

symmetry_se and diagnosis
0.020
0.015
0.010
0.005
0.000
symmetry_se
M
B
diagnosis

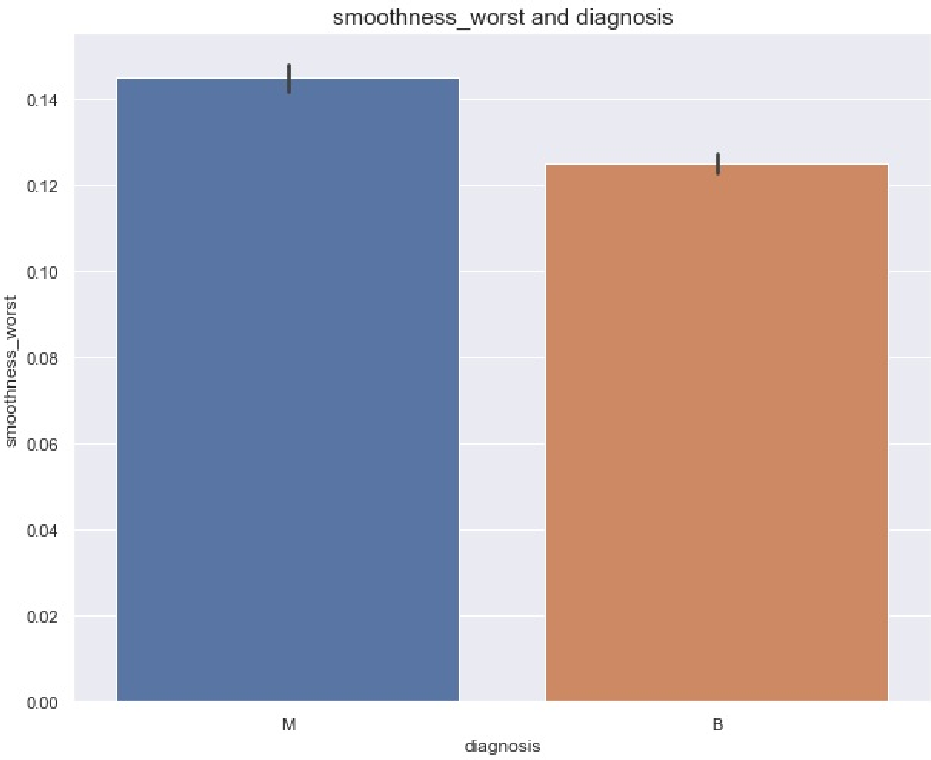

fractal_dimension_se and diagnosis
0.004
0.003
0.002
0.001
0.000
fractal_dimension_se
M
B
diagnosis

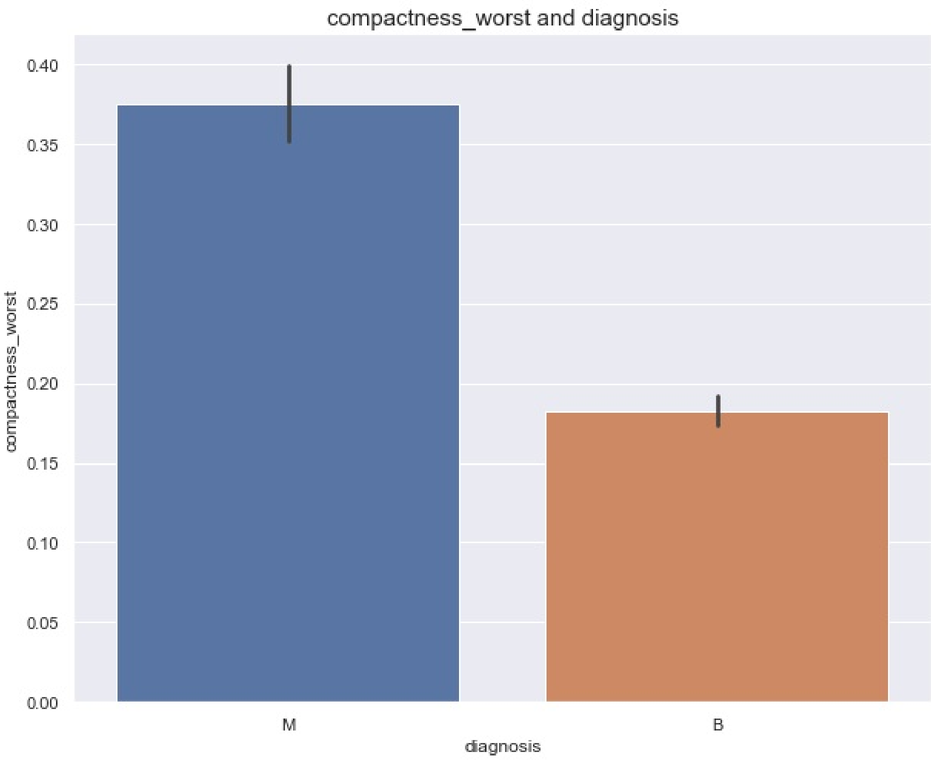

radius_worst and diagnosis
radius_worst
0
5
10
15
20
M
B
diagnosis

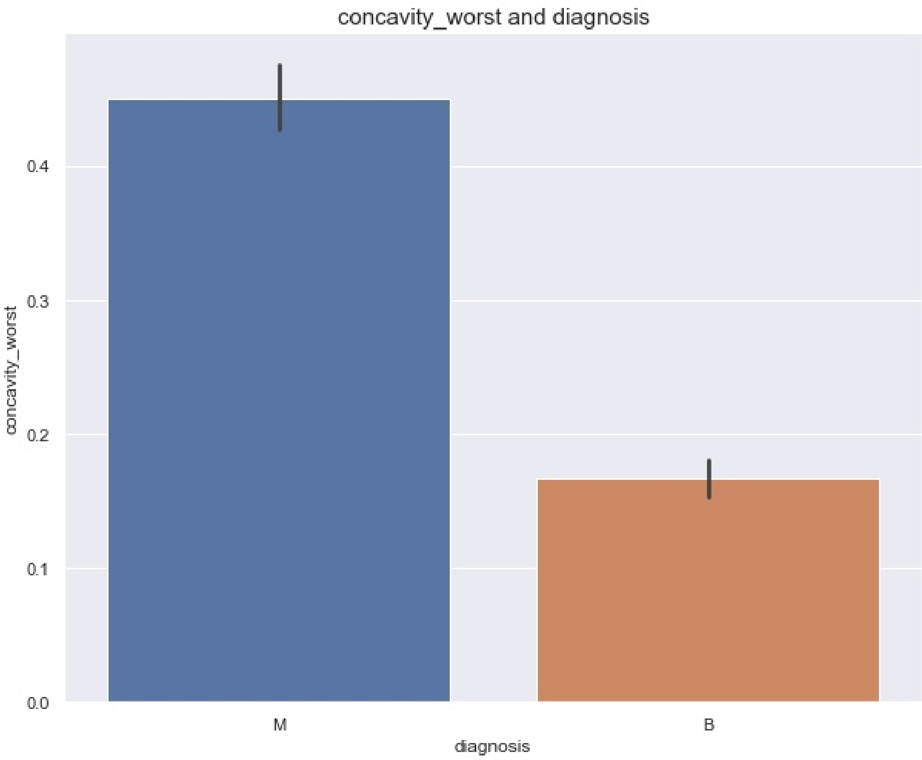

texture_worst and diagnosis
texture_worst
0
5
10
15
20
25
30
M
B
diagnosis

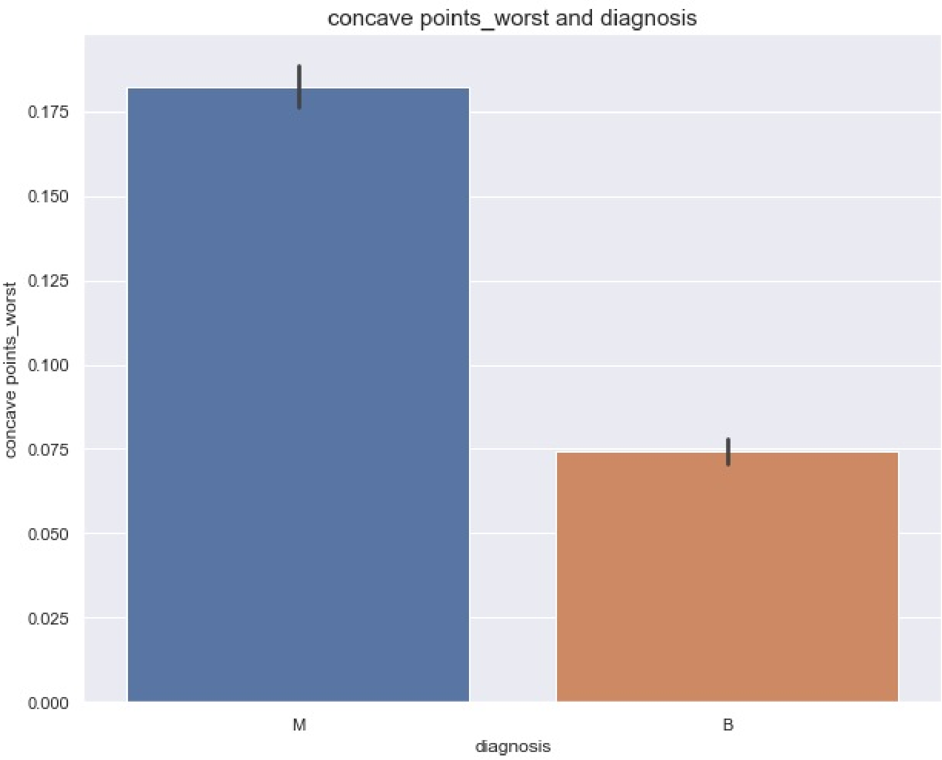

perimeter_worst and diagnosis
perimeter_worst
140
120
100
80
60
40
20
0
M
B
diagnosis

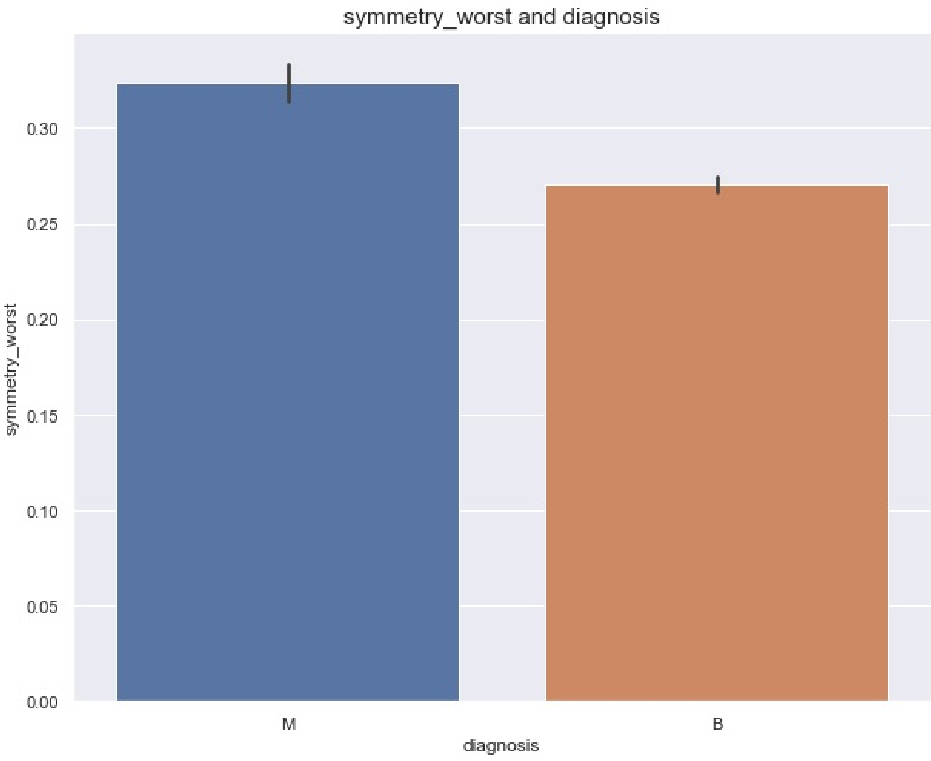

area_worst and diagnosis
area_worst
1400
1200
1000
800
600
400
200
0
M
B
diagnosis

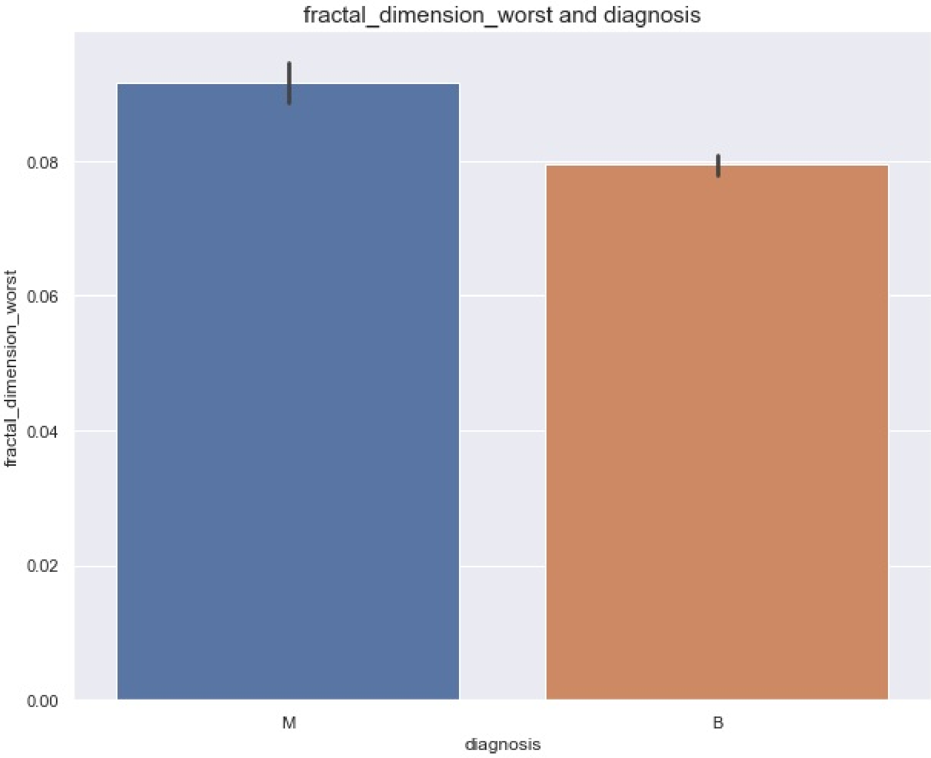

smoothness_worst and diagnosis
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
smoothness_worst
M
B
diagnosis


compactness_worst and diagnosis
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
compactness_worst
M
B
diagnosis


concavity_worst and diagnosis
0.4
0.3
0.2
0.1
0.0
concavity_worst
M
B
diagnosis

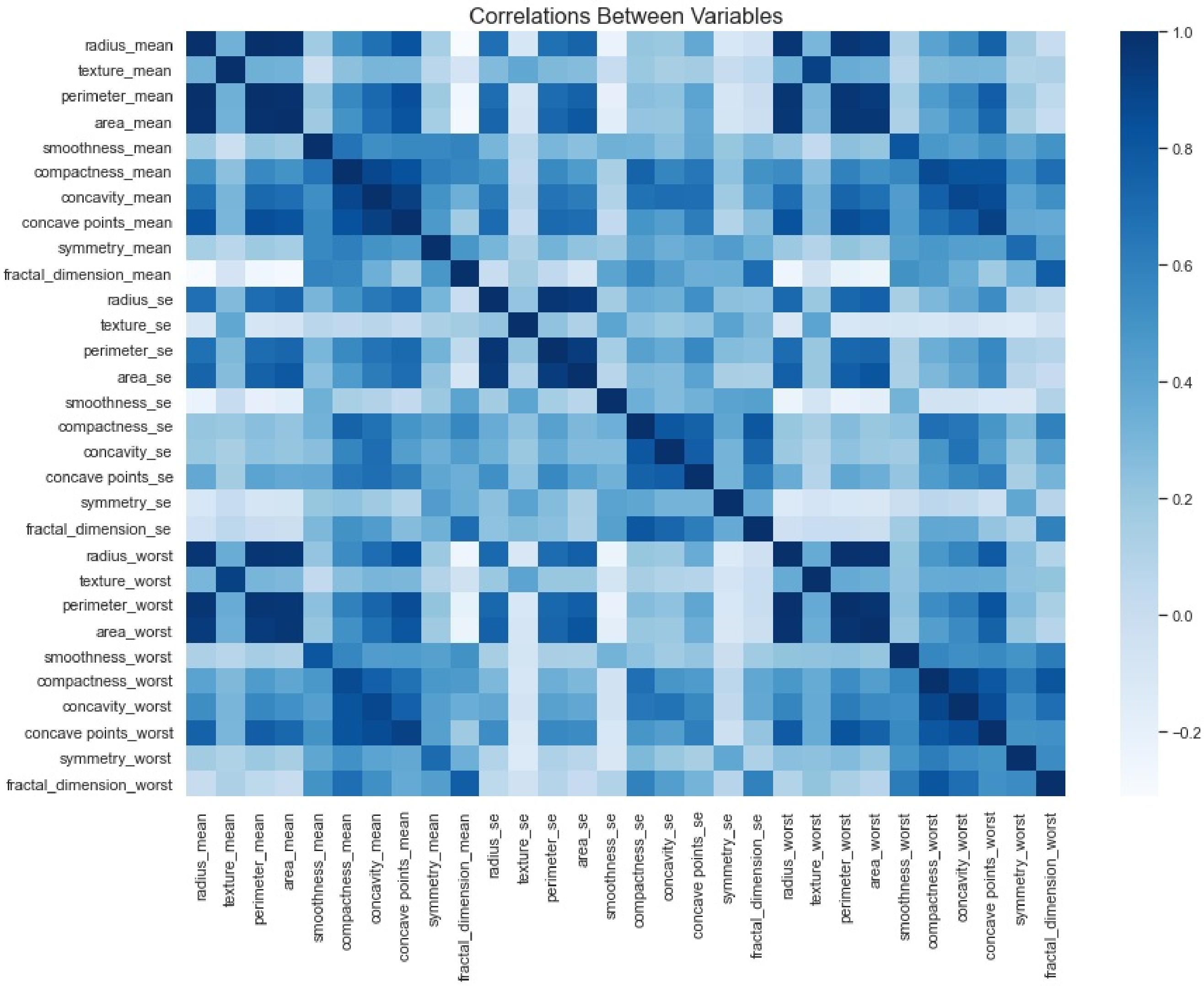

concave points_worst and diagnosis
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
concave points_worst
M
B
diagnosis





In [63]:

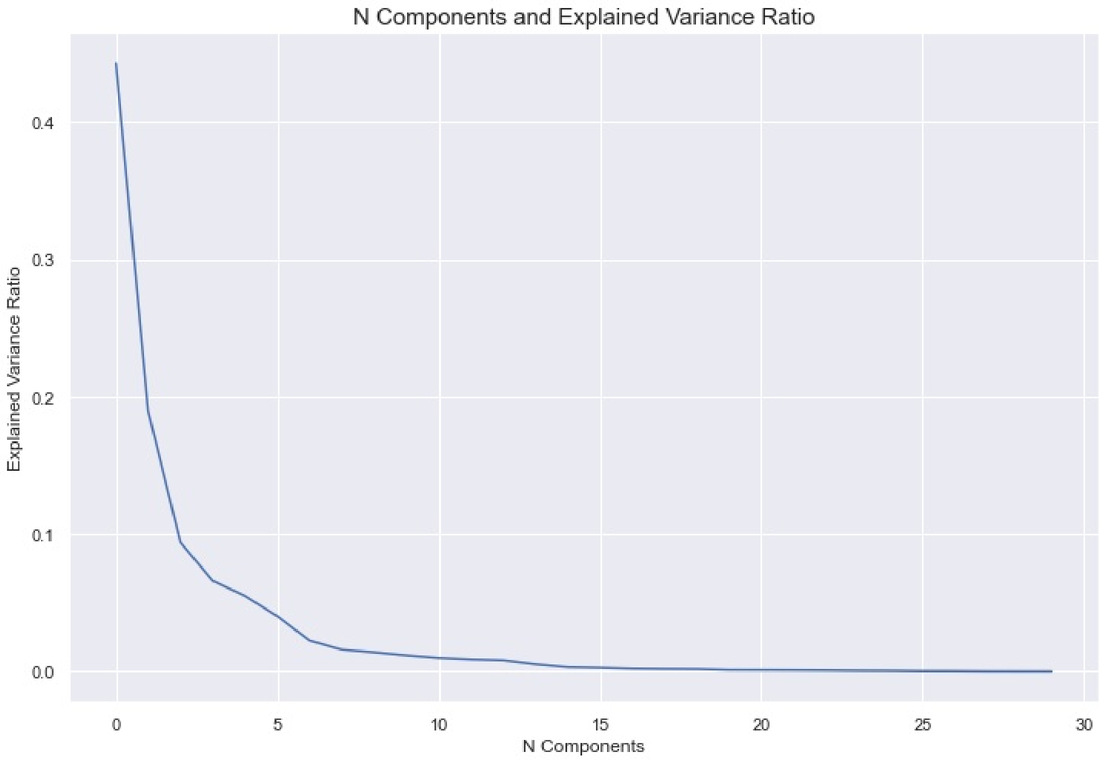
```
plt.figure(figsize=(14,10))
sns.heatmap(df.corr(), cmap="Blues")
plt.title("Correlations Between Variables", size=16)
plt.show()
```



In [64]:
```
X = df.drop("diagnosis", axis=1)
y = df["diagnosis"].replace({"B": 0, "M": 1})
```

In [65]:
```
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

In [66]:
```
pca = PCA()
pca.fit(X)
plt.figure(figsize=(12,8))
plt.plot(pca.explained_variance_ratio_)
plt.title("N Components and Explained Variance Ratio", size=15)
plt.xlabel("N Components")
plt.ylabel("Explained Variance Ratio")
plt.show()
```



In [67]:
```
pca = PCA(n_components = 5)
X = pca.fit_transform(X)
pca.explained_variance_ratio_.sum()
```

Out[67]:
```
0.8473427431679001
```

In [68]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

In [69]:
```
models = pd.DataFrame(columns=["Model", "Accuracy Score"])
```

In [70]:
```
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
predictions = log_reg.predict(X_test)
score = accuracy_score(y_test, predictions)
print("Accuracy Score:", score)
new_row = {"Model": "LogisticRegression", "Accuracy Score": score}
models = models.append(new_row, ignore_index=True)
```

```
Accuracy Score: 0.9883040935672515
```

In [71]:
```
rfc = RandomForestClassifier()
rfc.fit(X_train, y_train)
predictions = rfc.predict(X_test)
score = accuracy_score(y_test, predictions)
print("Accuracy Score:", score)
new_row = {"Model": "RandomForestClassifier", "Accuracy Score": score}
models = models.append(new_row, ignore_index=True)
```

```
Accuracy Score: 0.9707602339181286
```

In [72]:
```
gbc = GradientBoostingClassifier()
gbc.fit(X_train, y_train)
predictions = gbc.predict(X_test)
score = accuracy_score(y_test, predictions)
print("Accuracy Score:", score)
new_row = {"Model": "GradientBoostingClassifier", "Accuracy Score": score}
models = models.append(new_row, ignore_index=True)
```

```
Accuracy Score: 0.9766081871345029
```

In [73]:
```
svc = SVC()
svc.fit(X_train, y_train)
predictions = svc.predict(X_test)
score = accuracy_score(y_test, predictions)
print("Accuracy Score:", score)
new_row = {"Model": "SVC", "Accuracy Score": score}
models = models.append(new_row, ignore_index=True)
models.sort_values(by="Accuracy Score", ascending=False)
```

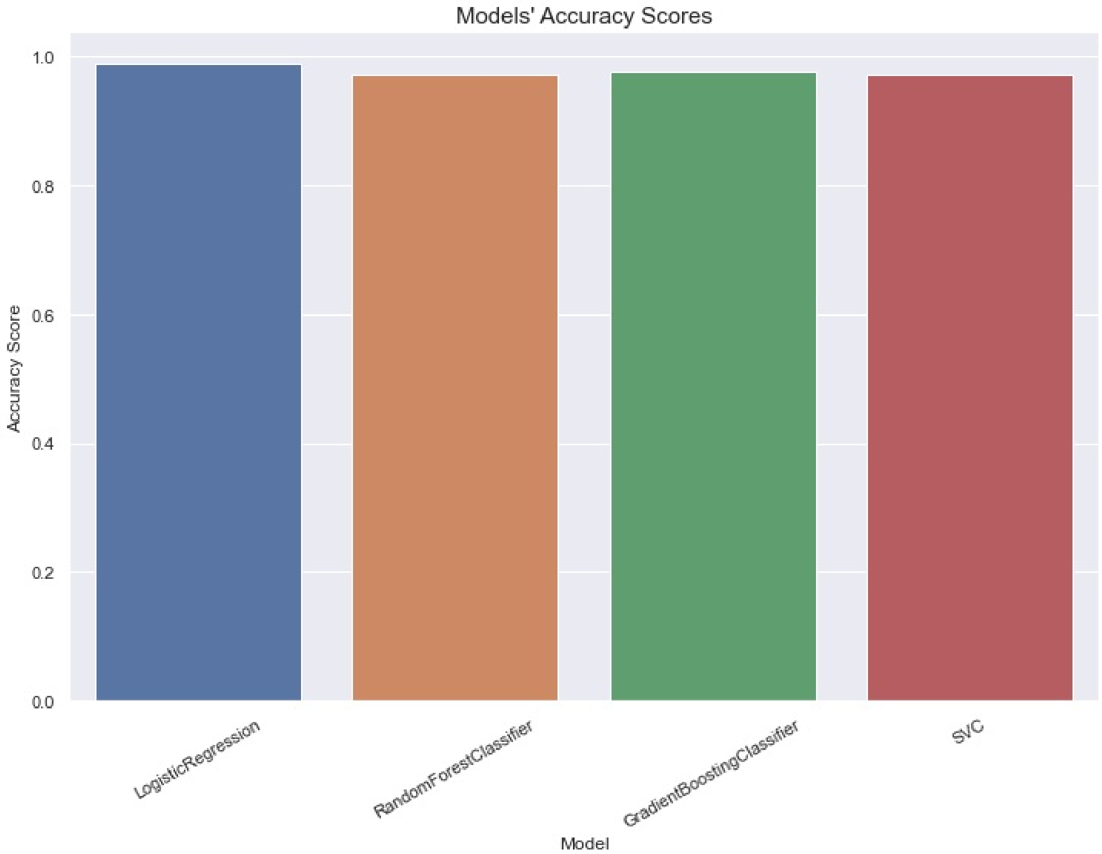
```
Accuracy Score: 0.9707602339181286
```

Out[73]:

	Model	Accuracy Score
0	LogisticRegression	0.988304
2	GradientBoostingClassifier	0.976608
1	RandomForestClassifier	0.97076
3	SVC	0.97076

In [74]:

```
plt.figure(figsize=(12,8))
sns.barplot(x=models["Model"], y=models["Accuracy Score"])
plt.title("Models' Accuracy Scores", size=15)
plt.xticks(rotation=30)
plt.show()
```



In [75]:

```
def visualize_roc_auc_curve(model, model_name):
    pred_prob = model.predict_proba(X_test)
    fpr, tpr, thresh = roc_curve(y_test, pred_prob[:,1], pos_label=1)

    score = roc_auc_score(y_test, pred_prob[:, 1])

    plt.figure(figsize=(10,8))
    plt.plot(fpr, tpr, linestyle="--", color="orange", label="ROC curve (area = %0.5f)" % score)
    plt.plot([0, 1], [0, 1], color="navy", lw=2, linestyle="--")

    plt.title(f"{model_name} ROC Curve", size=15)
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.legend(loc="lower right", prop={'size': 15})
    plt.show()
```

In [76]:

```
tuned_models = pd.DataFrame(columns=["Model", "Accuracy Score"])
param_grid_log_reg = {"C": [0.0001, 0.001, 0.01, 0.1, 1, 10]}
```

In [77]:

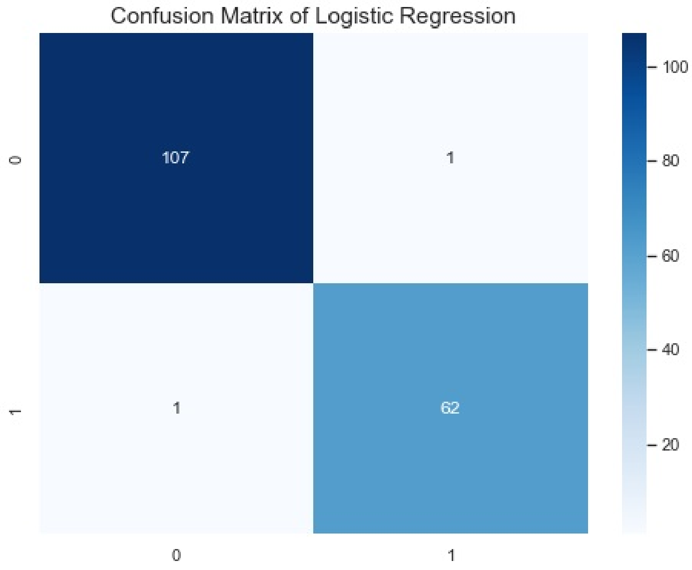
```
grid_log_reg = GridSearchCV(LogisticRegression(), param_grid_log_reg, scoring="accuracy", cv=5, verbose=0, n_jo
grid_log_reg.fit(X_train, y_train)
log_reg_params = grid_log_reg.best_params_
log_reg = LogisticRegression(**log_reg_params)
log_reg.fit(X_train, y_train)
predictions = log_reg.predict(X_test)
score = accuracy_score(y_test, predictions)
print("Accuracy Score:", score)

new_row = {"Model": "LogisticRegression", "Accuracy Score": score}
tuned_models = tuned_models.append(new_row, ignore_index=True)
```

```
Accuracy Score: 0.9883040935672515
```

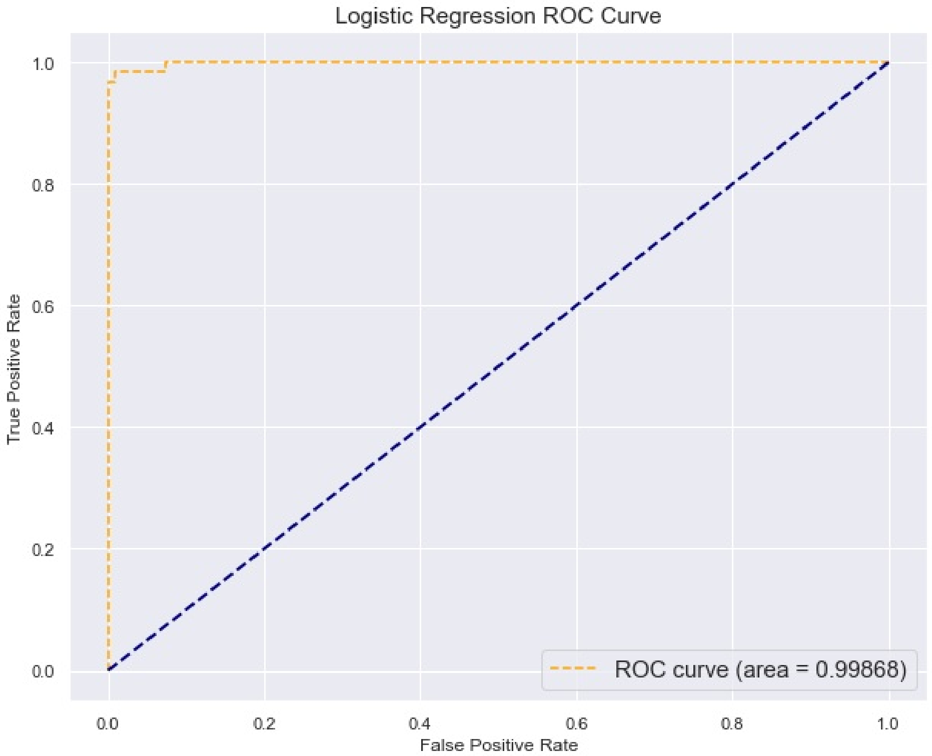
In [78]:

```
plt.figure(figsize=(8,6))
sns.heatmap(confusion_matrix(y_test, predictions), annot=True, cmap="Blues", fmt="d")
plt.title("Confusion Matrix of Logistic Regression", size=15)
```

```
plt.show()
```



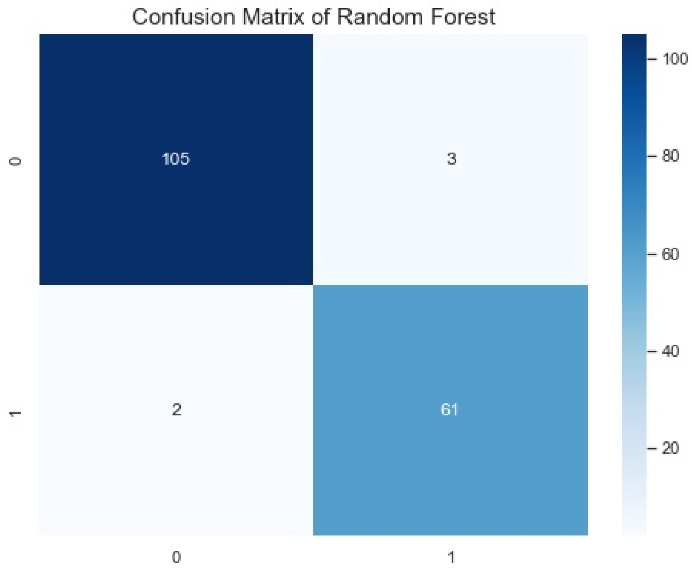
In [79]:
```
visualize_roc_auc_curve(log_reg, "Logistic Regression")
```



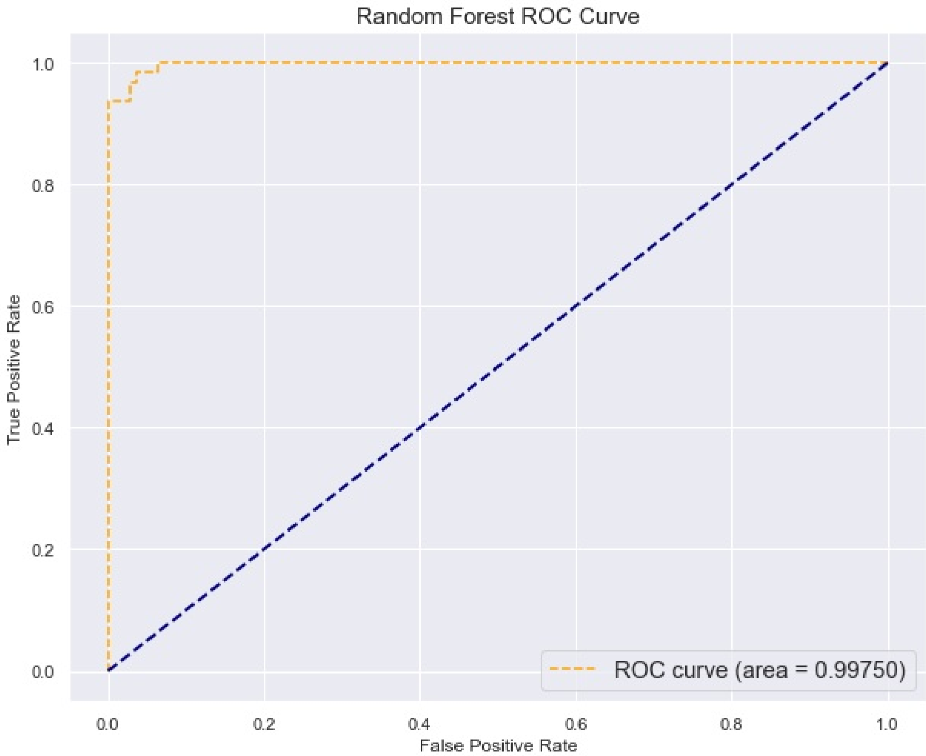
In [80]:
```
param_grid_rfc = {"min_samples_split": [2, 3, 10],
                  "min_samples_leaf": [1, 3, 10],
                  "n_estimators" :[100, 200, 500]}
grid_rfc = GridSearchCV(RandomForestClassifier(), param_grid_rfc, scoring="accuracy", cv=5, verbose=0, n_jobs=-
grid_rfc.fit(X_train, y_train)
rfc_params = grid_rfc.best_params_
rfc = RandomForestClassifier(**rfc_params)
rfc.fit(X_train, y_train)
predictions = rfc.predict(X_test)
score = accuracy_score(y_test, predictions)
print("Accuracy Score:", score)
new_row = {"Model": "RandomForestClassifier", "Accuracy Score": score}
tuned_models = tuned_models.append(new_row, ignore_index=True)
```

```
Accuracy Score: 0.9707602339181286
```

In [81]:
```
plt.figure(figsize=(8,6))
sns.heatmap(confusion_matrix(y_test, predictions), annot=True, cmap="Blues", fmt='d')
plt.title("Confusion Matrix of Random Forest", size=15)
plt.show()
```



In [82]:
```
visualize_roc_auc_curve(rfc, "Random Forest")
```

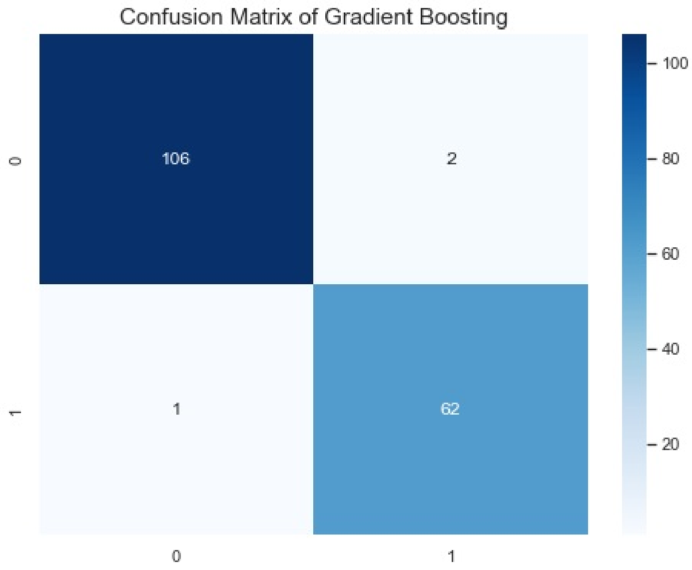


In [83]:
```
param_grid_gbc = {'n_estimators' : [100, 200, 500],
                  'learning_rate': [0.1, 0.05, 0.01],
                  'max_depth': [2, 3, 6],
                  'min_samples_leaf': [1, 2, 5]}
```

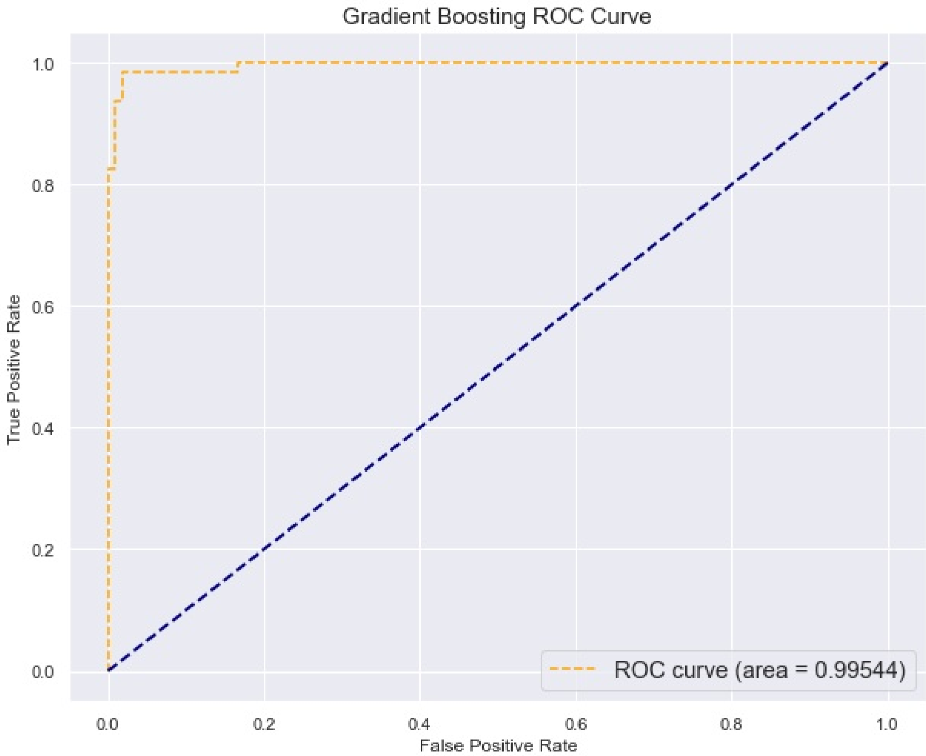
In [84]:
```
grid_gbc = GridSearchCV(GradientBoostingClassifier(), param_grid_gbc, scoring="accuracy", cv=5, verbose=0, n_jo

grid_gbc.fit(X_train, y_train)
gbc_params = grid_gbc.best_params_
gbc = GradientBoostingClassifier(**gbc_params)
gbc.fit(X_train, y_train)
predictions = gbc.predict(X_test)
score = accuracy_score(y_test, predictions)
print("Accuracy Score:", score)

new_row = {"Model": "GradientBoostingClassifier", "Accuracy Score": score}
tuned_models = tuned_models.append(new_row, ignore_index=True)
```

```
Accuracy Score: 0.9824561403508771
```

In [85]:
```
plt.figure(figsize=(8,6))
sns.heatmap(confusion_matrix(y_test, predictions), annot=True, cmap="Blues", fmt='d')
plt.title("Confusion Matrix of Gradient Boosting", size=15)
plt.show()
```



In [86]:
```
visualize_roc_auc_curve(gbc, "Gradient Boosting")
```



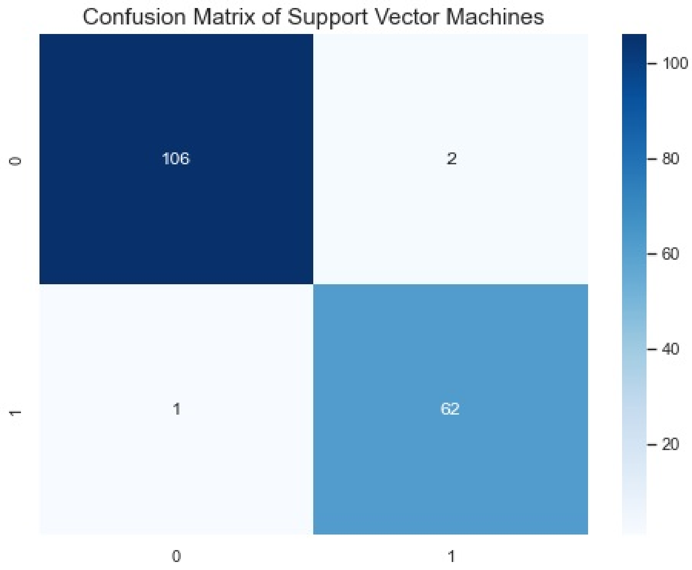
In [87]:
```
param_grid_svc = {'gamma': [ 0.001, 0.01, 0.1, 1, 10],
                  'C': [1, 10, 50, 100, 200, 300, 500, 1000]}
grid_svc = GridSearchCV(SVC(), param_grid_svc, scoring="accuracy", cv=5, verbose=0, n_jobs=-1)
grid_svc.fit(X_train, y_train)
```

Out[87]:
```
GridSearchCV(cv=5, estimator=SVC(), n_jobs=-1,
             param_grid={'C': [1, 10, 50, 100, 200, 300, 500, 1000],
                         'gamma': [0.001, 0.01, 0.1, 1, 10]},
             scoring='accuracy')
```

In [88]:
```
svc_params = grid_svc.best_params_
svc = SVC(**svc_params)
svc.fit(X_train, y_train)
predictions = svc.predict(X_test)
score = accuracy_score(y_test, predictions)
print("Accuracy Score:", score)

new_row = {"Model": "SVC", "Accuracy Score": score}
tuned_models = tuned_models.append(new_row, ignore_index=True)
```

```
Accuracy Score: 0.9824561403508771
```

In [89]:
```
plt.figure(figsize=(8,6))
sns.heatmap(confusion_matrix(y_test, predictions), annot=True, cmap="Blues", fmt='d')
plt.title("Confusion Matrix of Support Vector Machines", size=15)
plt.show()
```

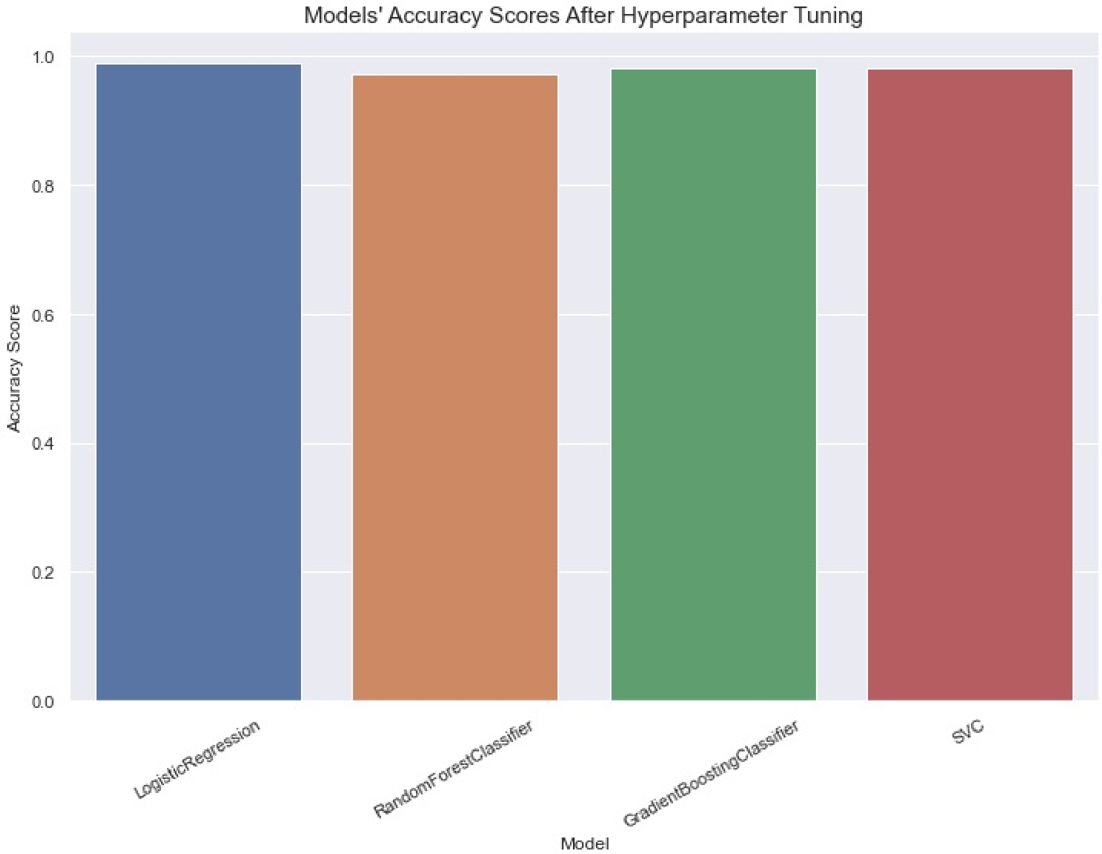


In [90]:
```
tuned_models.sort_values(by="Accuracy Score", ascending=False)
```

Out[90]:

	Model	Accuracy Score
0	LogisticRegression	0.988304
2	GradientBoostingClassifier	0.982456
3	SVC	0.982456
1	RandomForestClassifier	0.97076

In [91]:
```
plt.figure(figsize=(12, 8))
sns.barplot(x=tuned_models["Model"], y=tuned_models["Accuracy Score"])
plt.title("Models' Accuracy Scores After Hyperparameter Tuning", size=15)
plt.xticks(rotation=30)
plt.show()
```



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js