In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [31]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
df = pd.read_csv('data.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	cor
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	

5 rows × 33 columns

In [4]:
```
df.tail()
```

Out[4]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	cor
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	
566	926954	M	16.60	28.08	108.30	858.1	0.08455	
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	
568	92751	B	7.76	24.54	47.92	181.0	0.05263	

5 rows × 33 columns

In [5]:
```
df.shape
```

Out[5]: (569, 33)

In [6]:
```
df.describe().T
```

Out[6]:

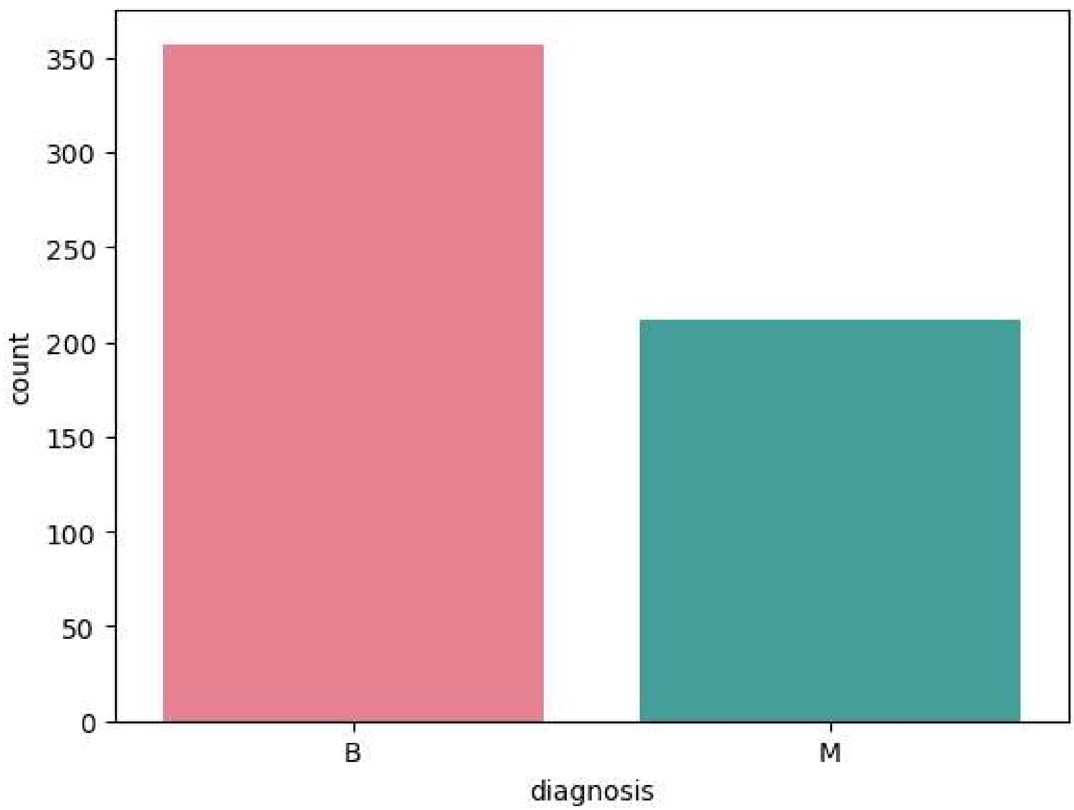
	count	mean	std	min	25%	50%
id	569.0	3.037183e+07	1.250206e+08	8670.000000	869218.000000	906024.000000
radius_mean	569.0	1.412729e+01	3.524049e+00	6.981000	11.700000	13.370000
texture_mean	569.0	1.928965e+01	4.301036e+00	9.710000	16.170000	18.840000
perimeter_mean	569.0	9.196903e+01	2.429898e+01	43.790000	75.170000	86.240000
area_mean	569.0	6.548891e+02	3.519141e+02	143.500000	420.300000	551.100000
smoothness_mean	569.0	9.636028e-02	1.406413e-02	0.052630	0.086370	0.095870
compactness_mean	569.0	1.043410e-01	5.281276e-02	0.019380	0.064920	0.092630
concavity_mean	569.0	8.879932e-02	7.971981e-02	0.000000	0.029560	0.061540
concave points_mean	569.0	4.891915e-02	3.880284e-02	0.000000	0.020310	0.033500
symmetry_mean	569.0	1.811619e-01	2.741428e-02	0.106000	0.161900	0.179200
fractal_dimension_mean	569.0	6.279761e-02	7.060363e-03	0.049960	0.057700	0.061540
radius_se	569.0	4.051721e-01	2.773127e-01	0.111500	0.232400	0.324200
texture_se	569.0	1.216853e+00	5.516484e-01	0.360200	0.833900	1.108000
perimeter_se	569.0	2.866059e+00	2.021855e+00	0.757000	1.606000	2.287000
area_se	569.0	4.033708e+01	4.549101e+01	6.802000	17.850000	24.530000
smoothness_se	569.0	7.040979e-03	3.002518e-03	0.001713	0.005169	0.006380
compactness_se	569.0	2.547814e-02	1.790818e-02	0.002252	0.013080	0.020450
concavity_se	569.0	3.189372e-02	3.018606e-02	0.000000	0.015090	0.025890
concave points_se	569.0	1.179614e-02	6.170285e-03	0.000000	0.007638	0.010930
symmetry_se	569.0	2.054230e-02	8.266372e-03	0.007882	0.015160	0.018730
fractal_dimension_se	569.0	3.794904e-03	2.646071e-03	0.000895	0.002248	0.003187
radius_worst	569.0	1.626919e+01	4.833242e+00	7.930000	13.010000	14.970000
texture_worst	569.0	2.567722e+01	6.146258e+00	12.020000	21.080000	25.410000
perimeter_worst	569.0	1.072612e+02	3.360254e+01	50.410000	84.110000	97.660000
area_worst	569.0	8.805831e+02	5.693570e+02	185.200000	515.300000	686.500000
smoothness_worst	569.0	1.323686e-01	2.283243e-02	0.071170	0.116600	0.131300
compactness_worst	569.0	2.542650e-01	1.573365e-01	0.027290	0.147200	0.211900
concavity_worst	569.0	2.721885e-01	2.086243e-01	0.000000	0.114500	0.226700
concave points_worst	569.0	1.146062e-01	6.573234e-02	0.000000	0.064930	0.099930
symmetry_worst	569.0	2.900756e-01	6.186747e-02	0.156500	0.250400	0.282200
fractal_dimension_worst	569.0	8.394582e-02	1.806127e-02	0.055040	0.071460	0.080040
Unnamed: 32	0.0	NaN	NaN	NaN	NaN	NaN

In [7]:
```
df.diagnosis.unique()
```
Out[7]:
```
array(['M', 'B'], dtype=object)
```

In [8]:
```
df['diagnosis'].value_counts()
```
Out[8]:
```
B    357
M    212
Name: diagnosis, dtype: int64
```

In [11]:
```
#pip install seaborn matplotlib
```

In [12]:
```

# convert it to a categorical variable
df['diagnosis'] = df['diagnosis'].astype('category')

#  create the count plot
sns.countplot(data=df, x='diagnosis', palette='husl')

# Show the plot
plt.show()

```



In [14]:
```
#sns.countplot(df['diagnosis'], palette='husl')
```

In [ ]:
```

```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

clean and prepare the data

In [15]:
```
df.drop('id',axis=1,inplace=True)
df.drop('Unnamed: 32',axis=1,inplace=True)
```

In [16]:
```
df.head()
```

Out[16]:

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

5 rows × 31 columns

In [17]:
```
df['diagnosis'] = df['diagnosis'].map({'M':1,'B':0})
df.head()
```

Out[17]:

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

5 rows × 31 columns

In [18]:
```
df.isnull().sum()
```

Out[18]:
```
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
dtype: int64
```

```
#def diagnosis_value(diagnosis):
    if diagnosis == 'M':
        return 1
    else:
        return 0

#df['diagnosis'] = df['diagnosis'].apply(diagnosis_value)
```

In [19]:

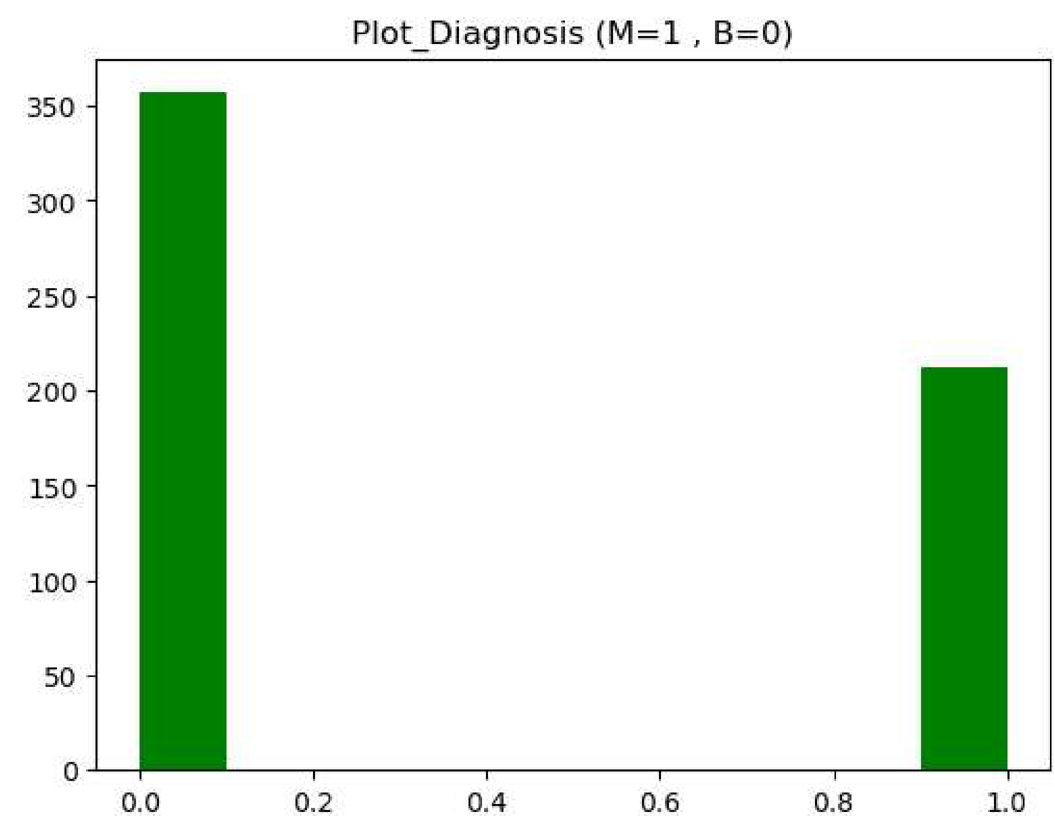
```
df.corr()
```

concave points_mean	0.822529	0.293464	0.850977	0.823269	0.553695
symmetry_mean	0.147741	0.071401	0.183027	0.151293	0.557775
fractal_dimension_mean	-0.311631	-0.076437	-0.261477	-0.283110	0.584792
radius_se	0.679090	0.275869	0.691765	0.732562	0.301467
texture_se	-0.097317	0.386358	-0.086761	-0.066280	0.068406
perimeter_se	0.674172	0.281673	0.693135	0.726628	0.296092
area_se	0.735864	0.259845	0.744983	0.800086	0.246552
smoothness_se	-0.222600	0.006614	-0.202694	-0.166777	0.332375
compactness_se	0.206000	0.191975	0.250744	0.212583	0.318943
concavity_se	0.194204	0.143293	0.228082	0.207660	0.248396
concave points_se	0.376169	0.163851	0.407217	0.372320	0.380676
symmetry_se	-0.104321	0.009127	-0.081629	-0.072497	0.200774

radius_mean , perimeter _mean, area_mean have a high correlation with malignant tumor

In [20]:

```
plt.hist(df['diagnosis'], color='g')
plt.title('Plot_Diagnosis (M=1 , B=0)')
plt.show()
```



In [ ]:
```
1
```

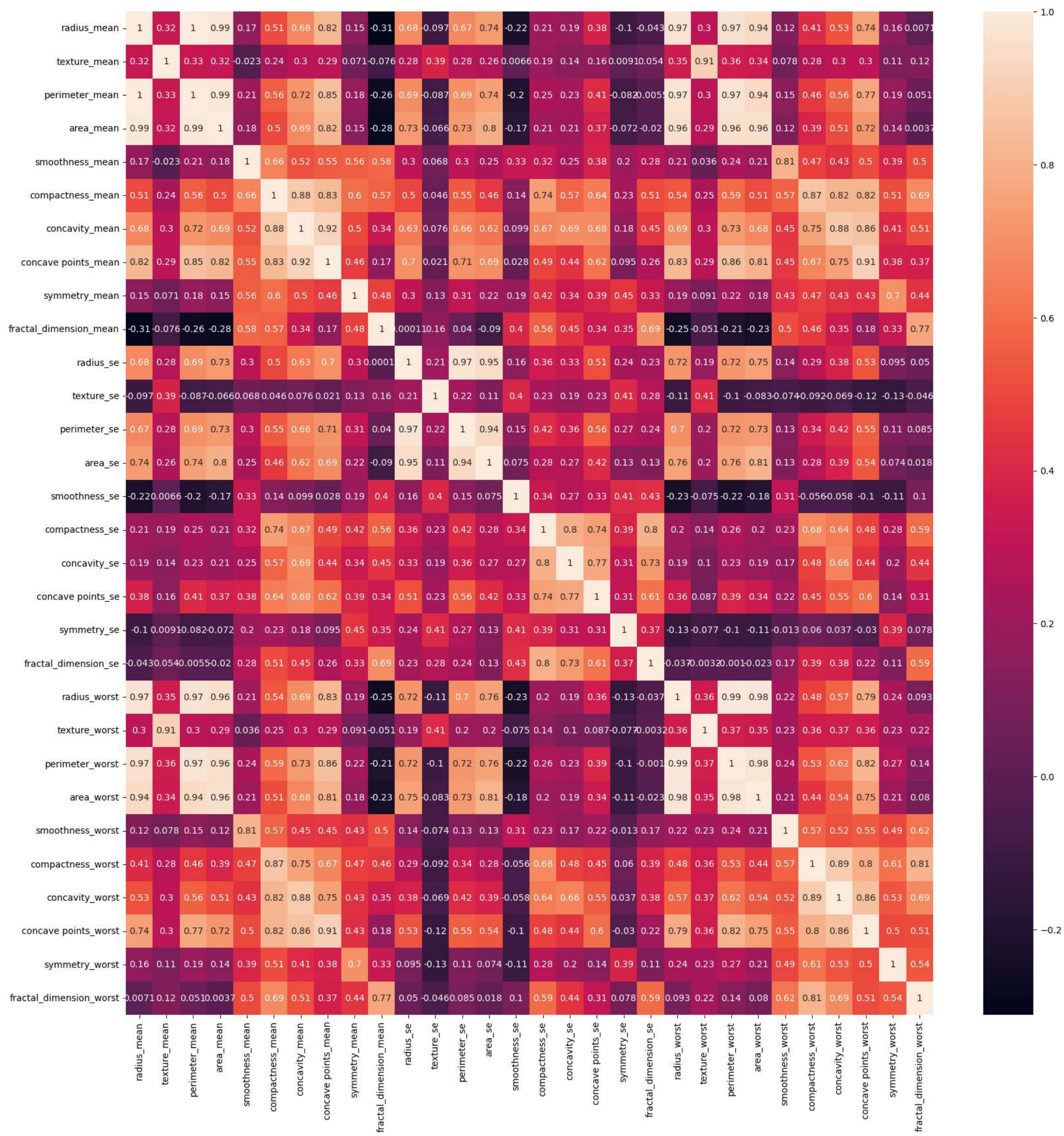
In [ ]:
```
1
```

In [ ]:
```
1
```

In [21]:
```
1 plt.figure(figsize=(20,20))
2 sns.heatmap(df.corr(), annot=True)
```

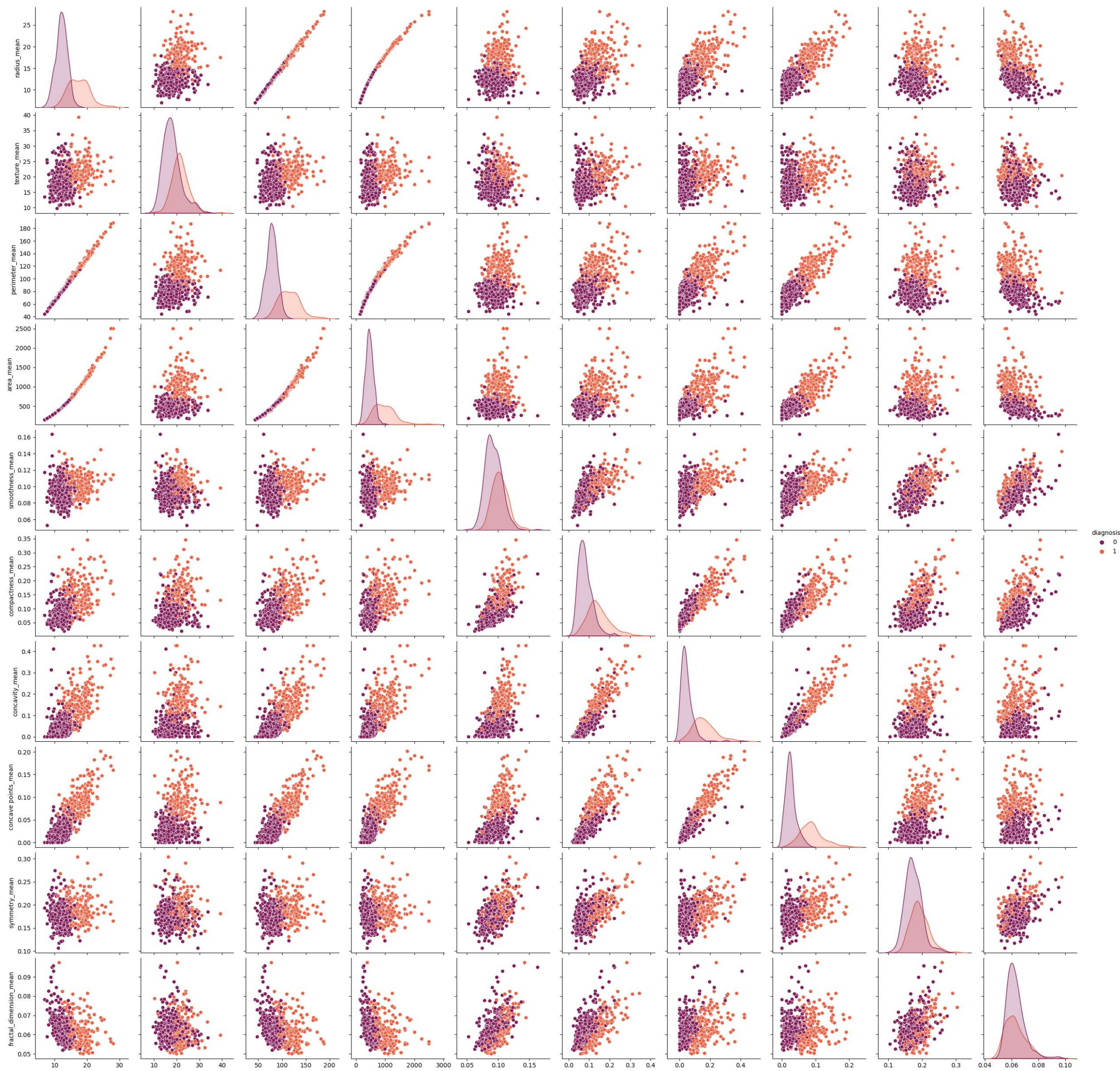
```
C:\Users\Pritik\AppData\Local\Temp\ipykernel_9152\357754966.py:2: FutureWarning: Th
e default value of numeric_only in DataFrame.corr is deprecated. In a future versio
n, it will default to False. Select only valid columns or specify the value of nume
ric_only to silence this warning.
  sns.heatmap(df.corr(), annot=True)
```

Out[21]: <Axes: >



In [22]:
```python
# generate a scatter plot matrix with the "mean" columns
cols = ['diagnosis',
        'radius_mean',
        'texture_mean',
        'perimeter_mean',
        'area_mean',
        'smoothness_mean',
        'compactness_mean',
        'concavity_mean',
        'concave points_mean',
        'symmetry_mean',
        'fractal_dimension_mean']

sns.pairplot(data=df[cols], hue='diagnosis', palette='rocket')
```

Out[22]: <seaborn.axisgrid.PairGrid at 0x1fe46f61010>



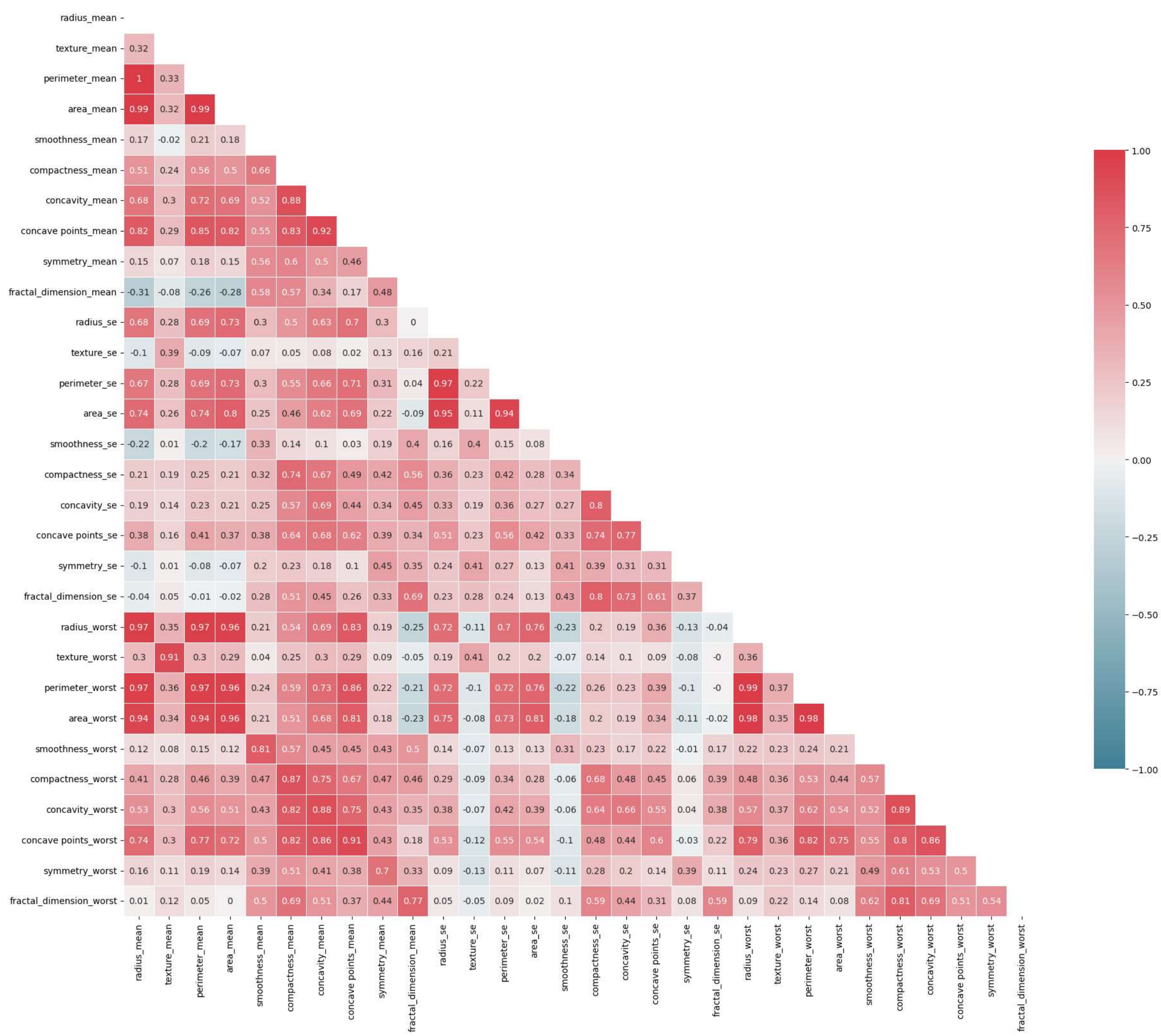
almost perfectly linear patterns between the radius, perimeter and area attributes are hinting at the presence of multicollinearity between these variables. (they are highly linearly related) Another set of variables that possibly imply multicollinearity are the concavity, concave_points and compactness.

```
Multicollinearity is a problem as it undermines the significance of independent varibales and we fix it
by removing the highly correlated predictors from the model
Use Partial Least Squares Regression (PLS) or Principal Components Analysis, regression methods that cut the number
of predictors to a smaller set of uncorrelated components.
```

In [30]:

```python
# Generate and visualize the correlation matrix
corr = df.corr().round(2)

# Mask for the upper triangle
mask = np.zeros_like(corr, dtype=np.bool_(True))
mask[np.triu_indices_from(mask)] = True

# Set figure size
f, ax = plt.subplots(figsize=(20, 20))

# Define custom colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap
sns.heatmap(corr, mask=mask, cmap=cmap, vmin=-1, vmax=1, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5}, annot=True)

plt.tight_layout()
```

```
C:\Users\Pritik\AppData\Local\Temp\ipykernel_9152\250676228.py:2: FutureWarning: Th
e default value of numeric_only in DataFrame.corr is deprecated. In a future versio
n, it will default to False. Select only valid columns or specify the value of nume
ric_only to silence this warning.
  corr = df.corr().round(2)
```



```
we can verify the presence of multicollinearity between some of the variables.
For instance, the radius_mean column has a correlation of 1 and 0.99 with perimeter_mean and area_mean columns, respectively.
This is because the three columns essentially contain the same information, which is the physical size of the observation
(the cell).
Therefore we should only pick ONE of the three columns when we go into further analysis.
```

```
Another place where multicollienartiy is apparent is between the "mean" columns and the "worst" column.
For instance, the radius_mean column has a correlation of 0.97 with the radius_worst column.
```

also there is multicollinearity between the attributes compactness, concavity, and concave points. So we can choose just ONE out of these, I am going for Compactness.

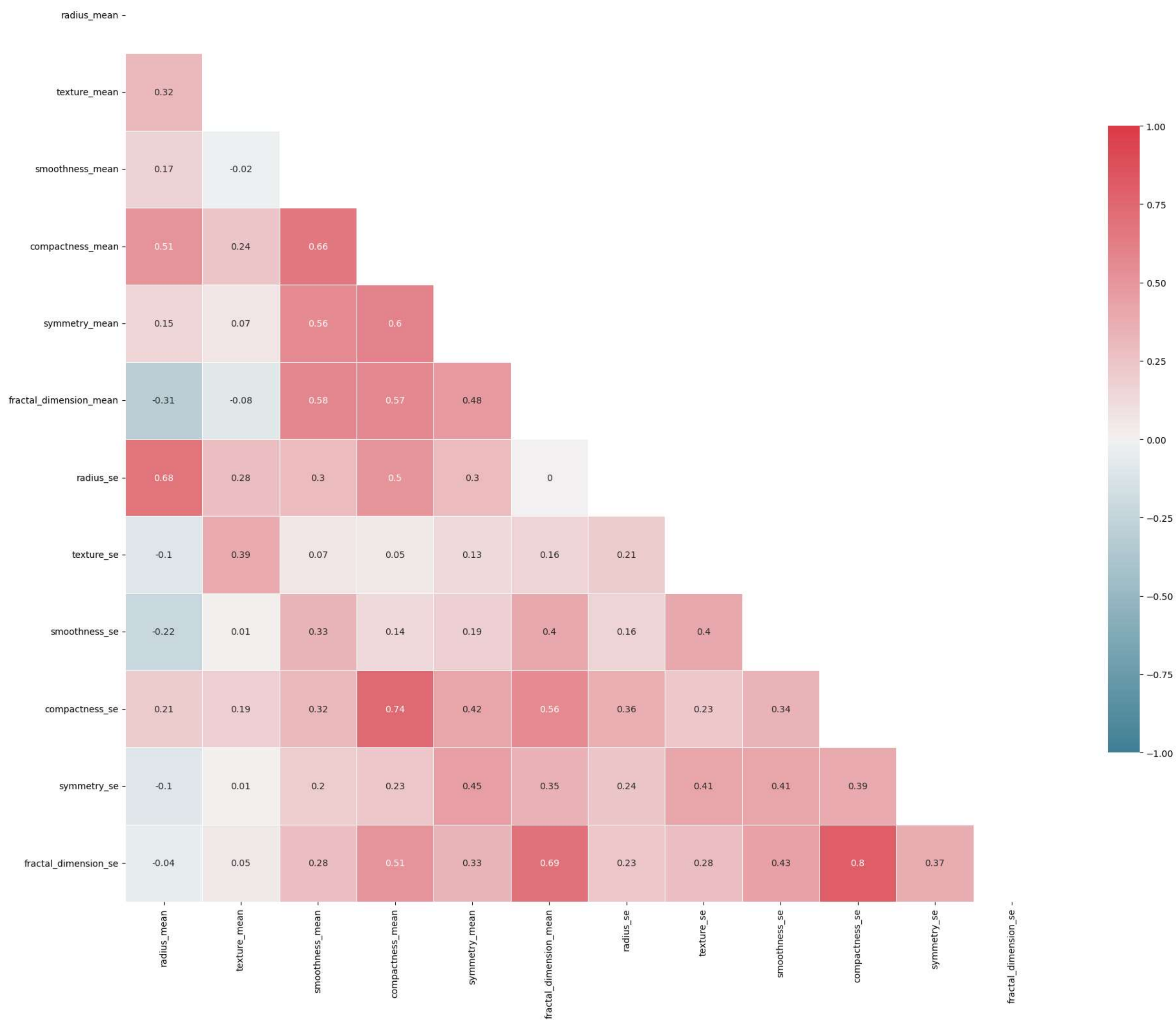
In [32]:

```
# first, drop all "worst" columns
cols = ['radius_worst',
        'texture_worst',
        'perimeter_worst',
        'area_worst',
        'smoothness_worst',
        'compactness_worst',
        'concavity_worst',
        'concave points_worst',
        'symmetry_worst',
        'fractal_dimension_worst']
df = df.drop(cols, axis=1)

# then, drop all columns related to the "perimeter" and "area" attributes
cols = ['perimeter_mean',
        'perimeter_se',
        'area_mean',
        'area_se']
df = df.drop(cols, axis=1)

# lastly, drop all columns related to the "concavity" and "concave points" attri
cols = ['concavity_mean',
        'concavity_se',
        'concave points_mean',
        'concave points_se']
df = df.drop(cols, axis=1)

# verify remaining columns
df.columns
```

Out[32]:

```
Index(['diagnosis', 'radius_mean', 'texture_mean', 'smoothness_mean',
       'compactness_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'smoothness_se', 'compactness_se',
       'symmetry_se', 'fractal_dimension_se'],
      dtype='object')
```

In [34]:
```
# Draw the heatmap again, with the new correlation matrix
corr = df.corr().round(2)
mask = np.zeros_like(corr, dtype=np.bool_(True))
mask[np.triu_indices_from(mask)] = True

f, ax = plt.subplots(figsize=(20, 20))
sns.heatmap(corr, mask=mask, cmap=cmap, vmin=-1, vmax=1, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5}, annot=True)
plt.tight_layout()
```



Building Model

In [35]:
```
X=df.drop(['diagnosis'],axis=1)
y = df['diagnosis']
```

In [36]:
```
from sklearn.model_selection import train_test_split
```

In [37]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=40
```

Feature Scaling

In [38]:
```
from sklearn.preprocessing import StandardScaler
ss=StandardScaler()

X_train=ss.fit_transform(X_train)
X_test=ss.fit_transform(X_test)
```

Models and finding out the Best one

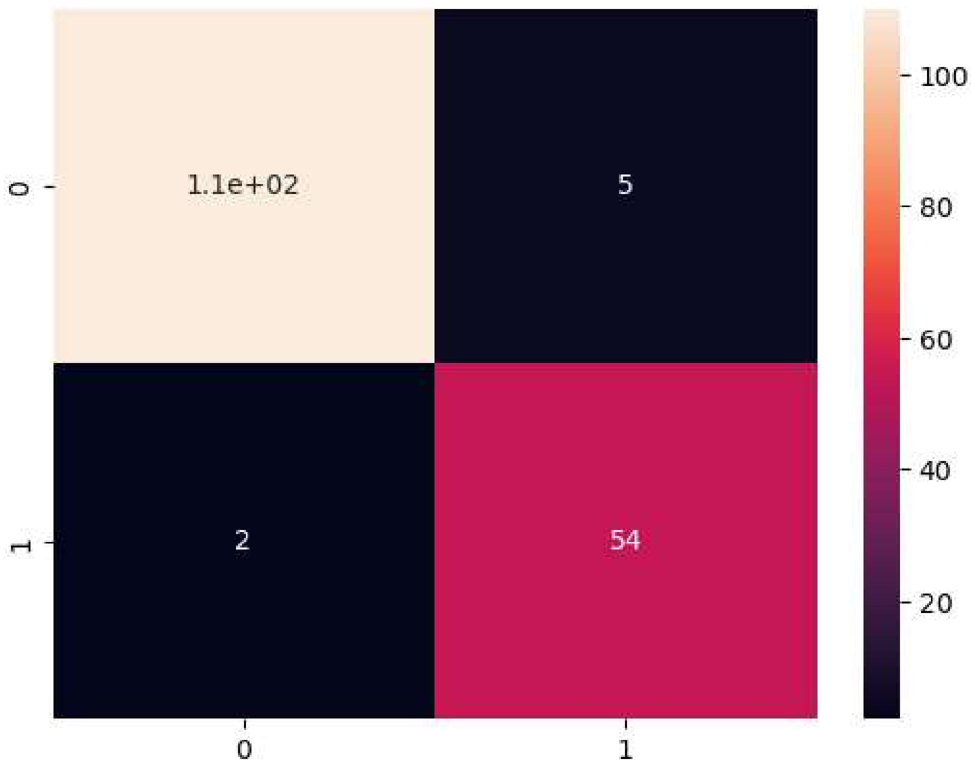
Logistic Regression

In [39]:
```
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()

model1=lr.fit(X_train,y_train)
prediction1=model1.predict(X_test)
```

In [40]:
```
from sklearn.metrics import confusion_matrix

cm=confusion_matrix(y_test,prediction1)
cm
```

Out[40]:
```
array([[110,   5],
       [  2,  54]], dtype=int64)
```

In [41]:
```
sns.heatmap(cm,annot=True)
plt.savefig('h.png')
```



In [42]:
```
TP=cm[0][0]
TN=cm[1][1]
FN=cm[1][0]
FP=cm[0][1]
print('Testing Accuracy:',(TP+TN)/(TP+TN+FN+FP))
```

```
Testing Accuracy: 0.9590643274853801
```

In [43]:
```
from sklearn.metrics import accuracy_score
```

In [44]:
```
accuracy_score(y_test,prediction1)
```

Out[44]:
```
0.9590643274853801
```

Decision Tree

In [45]:
```
from sklearn.tree import DecisionTreeClassifier

dtc=DecisionTreeClassifier()
model2=dtc.fit(X_train,y_train)
prediction2=model2.predict(X_test)
cm2= confusion_matrix(y_test,prediction2)
```

In [46]:
```
cm2
```

Out[46]:
```
array([[106,   9],
       [  8,  48]], dtype=int64)
```

In [47]:
```
accuracy_score(y_test,prediction2)
```
Out[47]: 0.9005847953216374

Random Forest

In [48]:
```
from sklearn.ensemble import RandomForestClassifier

rfc=RandomForestClassifier()
model3 = rfc.fit(X_train, y_train)
prediction3 = model3.predict(X_test)
confusion_matrix(y_test, prediction3)
```
Out[48]:
```
array([[109,   6],
       [  5,  51]], dtype=int64)
```

In [49]:
```
accuracy_score(y_test, prediction3)
```
Out[49]: 0.935672514619883

In [50]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, prediction3))
```
```
              precision    recall  f1-score   support

           0       0.96      0.95      0.95       115
           1       0.89      0.91      0.90        56

    accuracy                           0.94       171
   macro avg       0.93      0.93      0.93       171
weighted avg       0.94      0.94      0.94       171
```

In [51]:
```
print(classification_report(y_test, prediction1))

print(classification_report(y_test, prediction2))
```
```
              precision    recall  f1-score   support

           0       0.98      0.96      0.97       115
           1       0.92      0.96      0.94        56

    accuracy                           0.96       171
   macro avg       0.95      0.96      0.95       171
weighted avg       0.96      0.96      0.96       171

              precision    recall  f1-score   support

           0       0.93      0.92      0.93       115
           1       0.84      0.86      0.85        56

    accuracy                           0.90       171
   macro avg       0.89      0.89      0.89       171
weighted avg       0.90      0.90      0.90       171
```

K Nearest Neighbor (K NN)

Support Vector Machine

Naive Bayes

In [52]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
```

In [53]:
```
models=[]

models.append(('KNN', KNeighborsClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))

```

In [54]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

In [57]:
```
# evaluate each model
from sklearn.model_selection import KFold, cross_val_score

results = []
names = []

for name, model in models:
    kfold = KFold(n_splits=10, shuffle=True, random_state=40)
    cv_results = cross_val_score(model, X_train, y_train, cv=kfold, scoring='acc
    results.append(cv_results)
    names.append(name)

    msg = '%s: %f (%f)' % (name, cv_results.mean(), cv_results.std())
    print(msg)

```

```
KNN: 0.901987 (0.044061)
NB: 0.899744 (0.064079)
SVM: 0.909615 (0.045167)
```

In [ ]:
```

```

In [58]:

```
# make predictions on validation datasets

SVM = SVC()
SVM.fit(X_train, y_train)
predictions= SVM.predict(X_test)
print(accuracy_score(y_test, predictions))
print(classification_report(y_test, predictions))
print(confusion_matrix(y_test, predictions))
```

```
0.9649122807017544
              precision    recall  f1-score   support

           0       0.97      0.97      0.97       115
           1       0.95      0.95      0.95        56

    accuracy                           0.96       171
   macro avg       0.96      0.96      0.96       171
weighted avg       0.96      0.96      0.96       171

[[112   3]
 [  3  53]]
```

We are getting the best accuracy with SVM which is 96.4% , the model is predicting with 96% accuracy on our test data

In [ ]:

```

```

In [ ]:

```

```