In [ ]:
```
# DSO110 Final Group Project
```

In [ ]:
```
## Goal: Use the Wisconsin Breast Cancer Dataset to Classify a Tumor as Benign or Malignant
```

In [ ]:
```
# Wisconsin Breast Cancer Initial Data Exploration
```

In [1]:
```
# Import Packages
```

In [5]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.cluster import KMeans
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

In [3]:
```
# Import Data
```

In [6]:
```
from sklearn.datasets import load_breast_cancer
cancer_data = load_breast_cancer()
```

In [ ]:
```
# Explore Data
```

In [7]:
```
print(cancer_data)
```

```
{'data': array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
        1.189e-01],
       [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
```

```
        8.902e-02],
       [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
        8.758e-02],
       ...,
       [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
        7.820e-02],
       [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
        1.240e-01],
       [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
        7.039e-02]]), 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0,
       0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1]), 'frame': None, 'target_names': array(['malign
ant', 'benign'], dtype='<U9'), 'DESCR': '.. _breast_cancer_dataset:\n\nBreast cancer wisconsin (diagnostic) dat
aset\n--------------------------------------------\n\n**Data Set Characteristics:**\n\n    :Number of Instance
s: 569\n\n    :Number of Attributes: 30 numeric, predictive attributes and the class\n\n    :Attribute Informat
ion:\n        - radius (mean of distances from center to points on the perimeter)\n        - texture (standard
deviation of gray-scale values)\n        - perimeter\n        - area\n        - smoothness (local variation in
radius lengths)\n        - compactness (perimeter^2 / area - 1.0)\n        - concavity (severity of concave por
tions of the contour)\n        - concave points (number of concave portions of the contour)\n        - symmetry
\n        - fractal dimension ("coastline approximation" - 1)\n\n        The mean, standard error, and "worst"
or largest (mean of the three\n        worst/largest values) of these features were computed for each image,\n
resulting in 30 features.  For instance, field 0 is Mean Radius, field\n        10 is Radius SE, field 20 is Wo
```

```
rst Radius.\n\n        - class:\n                - WDBC-Malignant\n                - WDBC-Benign\n\n    :Summar
y Statistics:\n\n    ===================================== ====== ======\n
Min    Max\n    ===================================== ====== ======\n    radius (mean):
6.981  28.11\n    texture (mean):                       9.71   39.28\n    perimeter (mean):
43.79  188.5\n    area (mean):                          143.5  2501.0\n    smoothness (mean):
0.053  0.163\n    compactness (mean):                   0.019  0.345\n    concavity (mean):
0.0    0.427\n    concave points (mean):                0.0    0.201\n    symmetry (mean):
0.106  0.304\n    fractal dimension (mean):             0.05   0.097\n    radius (standard error):
0.112  2.873\n    texture (standard error):             0.36   4.885\n    perimeter (standard error):
0.757  21.98\n    area (standard error):                6.802  542.2\n    smoothness (standard error):
0.002  0.031\n    compactness (standard error):         0.002  0.135\n    concavity (standard error):
0.0    0.396\n    concave points (standard error):      0.0    0.053\n    symmetry (standard error):
0.008  0.079\n    fractal dimension (standard error):   0.001  0.03\n    radius (worst):
7.93   36.04\n    texture (worst):                      12.02  49.54\n    perimeter (worst):
50.41  251.2\n    area (worst):                         185.2  4254.0\n    smoothness (worst):
0.071  0.223\n    compactness (worst):                  0.027  1.058\n    concavity (worst):
0.0    1.252\n    concave points (worst):               0.0    0.291\n    symmetry (worst):
0.156  0.664\n    fractal dimension (worst):            0.055  0.208\n    =====================================
====== ======\n\n    :Missing Attribute Values: None\n\n    :Class Distribution: 212 - Malignant, 357 - Benign
\n\n    :Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian\n\n    :Donor: Nick Street\n\n
:Date: November, 1995\n\nThis is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.\nhttps://goo.g
l/U2Uwz2\n\nFeatures are computed from a digitized image of a fine needle\naspirate (FNA) of a breast mass.  Th
ey describe\ncharacteristics of the cell nuclei present in the image.\n\nSeparating plane described above was o
btained using\nMultisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree\nConstruction Via Linear Program
ming." Proceedings of the 4th\nMidwest Artificial Intelligence and Cognitive Science Society,\npp. 97-101, 199
2], a classification method which uses linear\nprogramming to construct a decision tree.  Relevant features\nwe
re selected using an exhaustive search in the space of 1-4\nfeatures and 1-3 separating planes.\n\nThe actual l
inear program used to obtain the separating plane\nin the 3-dimensional space is that described in:\n[K. P. Ben
nett and O. L. Mangasarian: "Robust Linear\nProgramming Discrimination of Two Linearly Inseparable Sets",\nOpti
mization Methods and Software 1, 1992, 23-34].\n\nThis database is also available through the UW CS ftp serve
r:\n\nftp ftp.cs.wisc.edu\ncd math-prog/cpo-dataset/machine-learn/WDBC/\n\n.. topic:: References\n\n   - W.N. S
treet, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction \n     for breast tumor diagnosis. IS&T/SP
IE 1993 International Symposium on \n     Electronic Imaging: Science and Technology, volume 1905, pages 861-87
0,\n     San Jose, CA, 1993.\n   - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and
\n     prognosis via linear programming. Operations Research, 43(4), pages 570-577, \n     July-August 1995.\n
- W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques\n     to diagnose breast cancer
from fine-needle aspirates. Cancer Letters 77 (1994) \n     163-171.', 'feature_names': array(['mean radius',
'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
```

```
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23'), 'filename': '/Users/dessiemiller91/opt/anac
onda3/lib/python3.9/site-packages/sklearn/datasets/data/breast_cancer.csv'}
```

In [8]:
```
print(cancer_data.keys())
```

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename'])
```

In [9]:
```
print("Feaures: ", cancer_data.feature_names)
```

```
Feaures:  ['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
```

In [10]:
```
print("Labels: ", cancer_data.target_names)
```

```
Labels:  ['malignant' 'benign']
```

In [11]:
```
print (cancer_data.DESCR[27:3130])
```

```
Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
```

```
    - concave points (number of concave portions of the contour)
    - symmetry
    - fractal dimension ("coastline approximation" - 1)

    The mean, standard error, and "worst" or largest (mean of the three
    worst/largest values) of these features were computed for each image,
    resulting in 30 features.  For instance, field 0 is Mean Radius, field
    10 is Radius SE, field 20 is Worst Radius.

    - class:
            - WDBC-Malignant
            - WDBC-Benign

:Summary Statistics:

===================================== ====== ======
                                       Min    Max
===================================== ====== ======
radius (mean):                        6.981  28.11
texture (mean):                       9.71   39.28
perimeter (mean):                     43.79  188.5
area (mean):                          143.5  2501.0
smoothness (mean):                    0.053  0.163
compactness (mean):                   0.019  0.345
concavity (mean):                     0.0    0.427
concave points (mean):                0.0    0.201
symmetry (mean):                      0.106  0.304
fractal dimension (mean):             0.05   0.097
radius (standard error):              0.112  2.873
texture (standard error):             0.36   4.885
perimeter (standard error):           0.757  21.98
area (standard error):                6.802  542.2
smoothness (standard error):          0.002  0.031
compactness (standard error):         0.002  0.135
concavity (standard error):           0.0    0.396
concave points (standard error):      0.0    0.053
symmetry (standard error):            0.008  0.079
fractal dimension (standard error):   0.001  0.03
radius (worst):                       7.93   36.04
texture (worst):                      12.02  49.54
perimeter (worst):                    50.41  251.2
area (worst):                         185.2  4254.0
smoothness (worst):                   0.071  0.223
compactness (worst):                  0.027  1.058
concavity (worst):                    0.0    1.252
```

```
    concave points (worst):                0.0    0.291
    symmetry (worst):                      0.156  0.664
    fractal dimension (worst):             0.055  0.208
    ====================================== ====== ======

    :Missing Attribute Values: None

    :Class Distribution: 212 - Malignant, 357 - Be
```

In [ ]:
```
# Check for Missing Data
```

In [ ]:
```
## Feature Variables Data Frame
```

In [12]:
```
cancer_features = pd.DataFrame(cancer_data.data, columns = cancer_data.feature_names)
```

In [13]:
```
cancer_features.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 30 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
```

```
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
dtypes: float64(30)
memory usage: 133.5 KB
```

In [ ]:
```
### The data is clean and there are no missing values. Proceed to next step.
```

In [ ]:
```
## Create Target Variable Data Frame
```

In [14]:
```
cancer_target = pd.DataFrame(cancer_data.target, columns=['target'])
```

In [15]:
```
cancer_target['target'].value_counts()
```

Out[15]:
```
1    357
0    212
Name: target, dtype: int64
```

In [ ]:
```
### The distribution of the target variable is 212 (Malignant) and 357 (Benign). "Benign" and "Malignant" are p
```

In [ ]:
```
### Merge Features and Target Variables Together
```

In [16]:
```
cancerDF = pd.concat([cancer_features, cancer_target], axis=1)
```

In [17]:
```
cancerDF['target'] = cancerDF['target'].apply(lambda x: "Benign"
                                              if x == 1 else "Malignant")
```

In [18]:
```
cancerDF.head()
```

Out[18]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.60	2
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158.80	1
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152.50	1
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	98.87	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	152.20	1

5 rows × 31 columns

In [19]:
```
cancerDF.describe()
```

Out[19]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mea fract dimensi
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.0000
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	0.0627
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	0.0070
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	0.0499
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	0.0577
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	0.0615
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	0.0661
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	0.0974

8 rows × 30 columns

In [ ]:
```
# Exploratory Analysis
```

In [ ]:
```
## Count Plot of Each Diagnosis
```

In [20]:
```
sns.set_style('darkgrid')
```

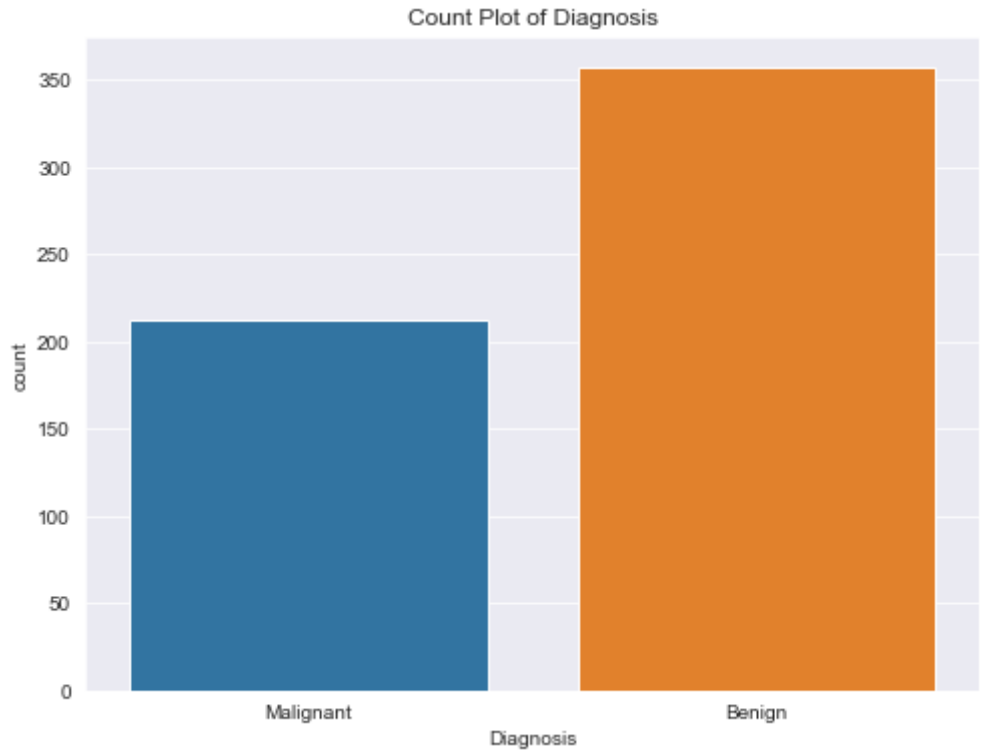
In [21]:
```
cancerDF['target'].value_counts()
```

Out[21]:
```
Benign       357
Malignant    212
Name: target, dtype: int64
```

In [22]:
```
plt.figure(figsize=(8,6))
sns.countplot(cancerDF['target'])
plt.xlabel("Diagnosis")
plt.title("Count Plot of Diagnosis")
```

```
/Users/dessiemiller91/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass
the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data
`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[22]:
```
Text(0.5, 1.0, 'Count Plot of Diagnosis')
```



In [ ]:
```
## Distribution of Features
```

In [ ]:
```
### Now we will take a look at the distribution of each feature to see how they are different between "Malignan
```

In [23]:
```
from sklearn.preprocessing import StandardScaler
```

In [24]:
```
scaler = StandardScaler()
scaler.fit(cancer_features)

features_scaled = scaler.transform(cancer_features)
features_scaled = pd.DataFrame(data=features_scaled,
                               columns=cancer_features.columns)

cancer_scaled = pd.concat([features_scaled, cancerDF['target']], axis=1)
```

In [ ]:
```
### Let's unpivot the dataframe from wide to long format.
```

In [25]:
```
cancer_scaled_melt = pd.melt(cancer_scaled, id_vars='target',
                             var_name='features', value_name='value')
cancer_scaled_melt.head(3)
```

Out[25]:

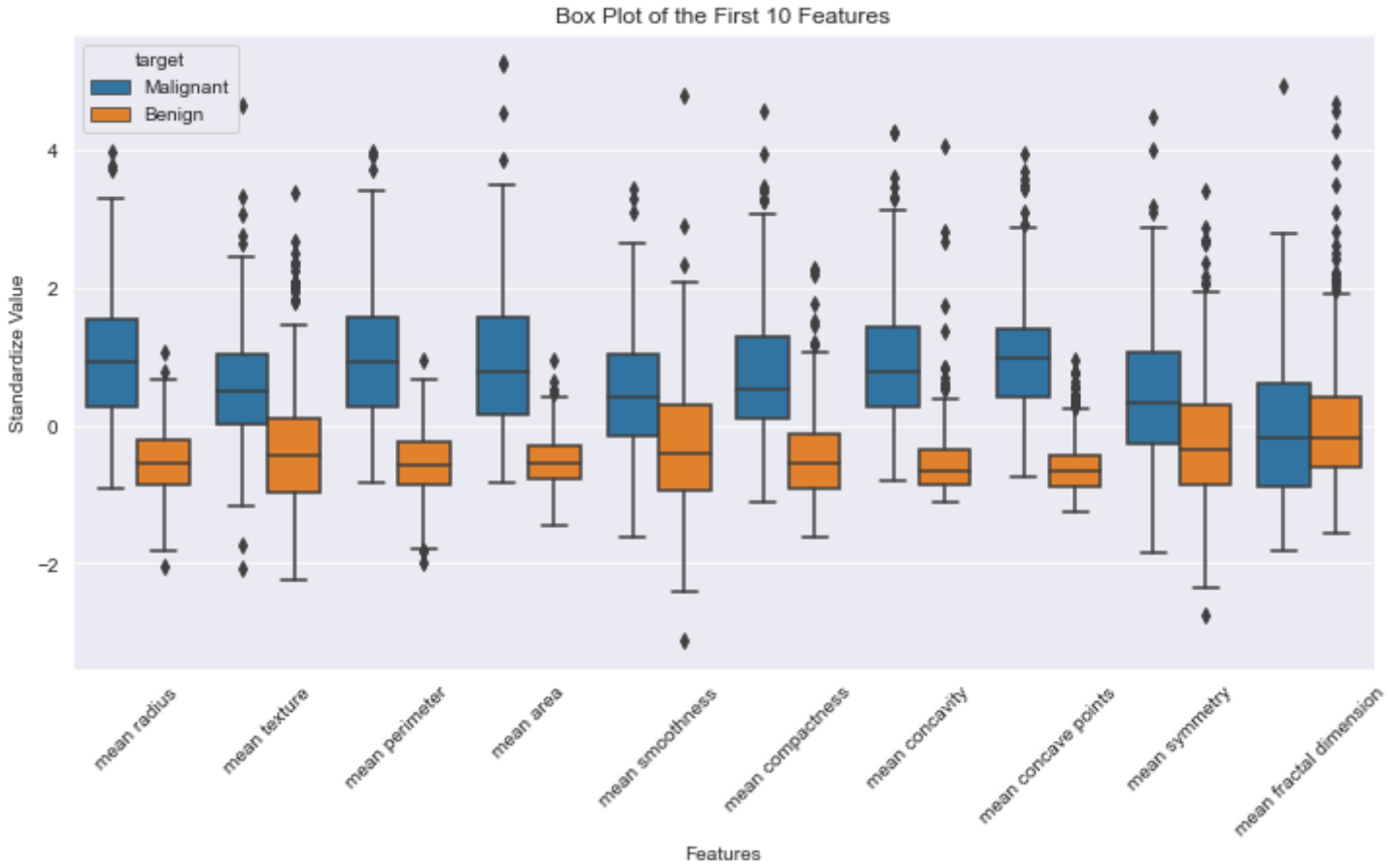
	target	features	value
0	Malignant	mean radius	1.097064
1	Malignant	mean radius	1.829821
2	Malignant	mean radius	1.579888

In [ ]:
```
### There are 30 features so a box plot will be created for each batch of 10 features. Box plots are useful in
```

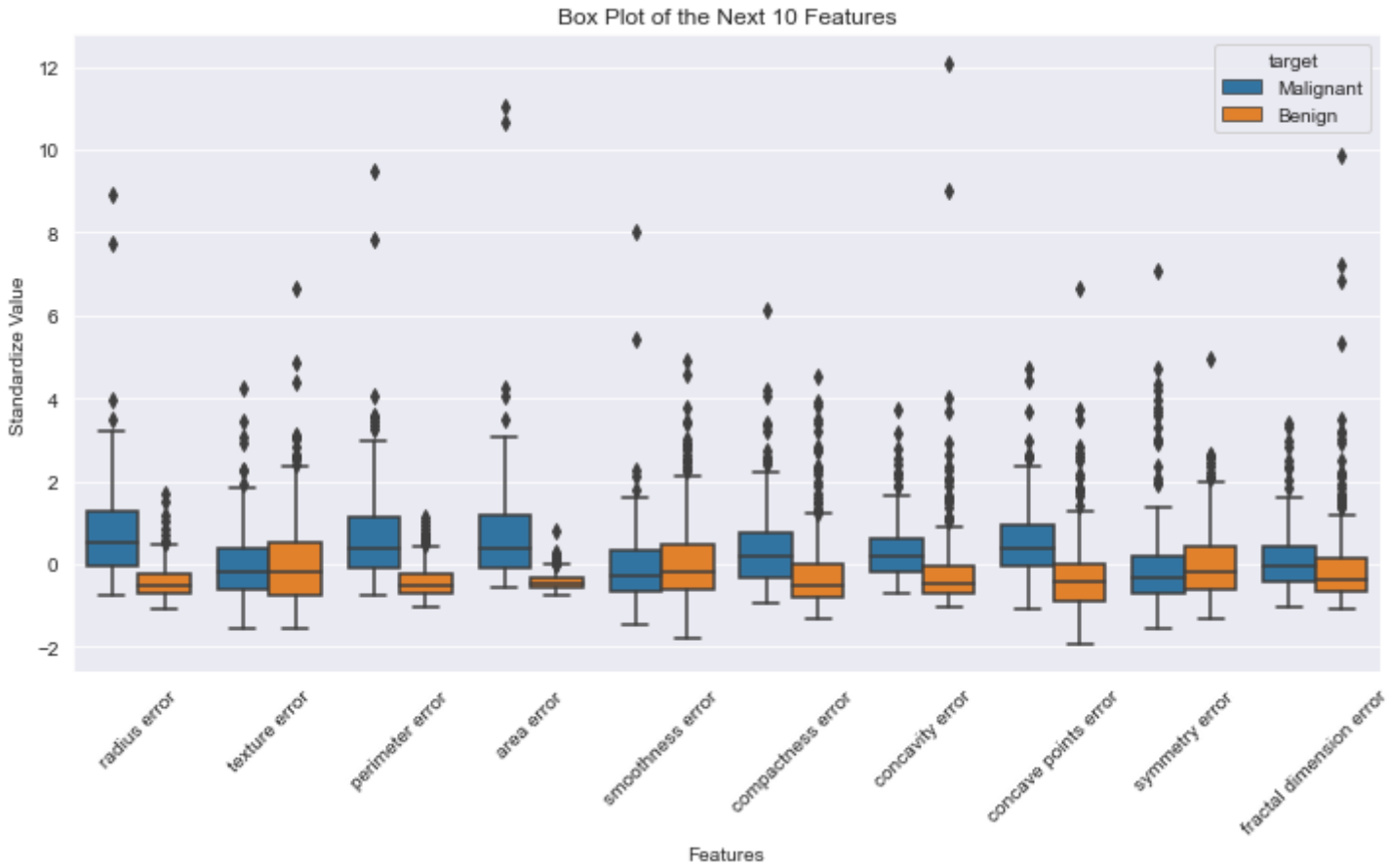
In [ ]:
```
### Define Box Plot Features
```

In [26]:
```
def box_plot(features, name):
    """
    This function creates box plots of features given in the argument.
    """
    # Create query
    query = ''
    for x in features:
        query += "features == '" + str(x) + "' or "
    query = query[0:-4]

    # Create data for visualization
    data = cancer_scaled_melt.query(query)

    # Plot figure
    plt.figure(figsize=(12, 6))
    sns.boxplot(x='features', y='value', hue='target', data=data)
    plt.xticks(rotation=45)
    plt.title(name)
    plt.xlabel("Features")
    plt.ylabel("Standardize Value")
```

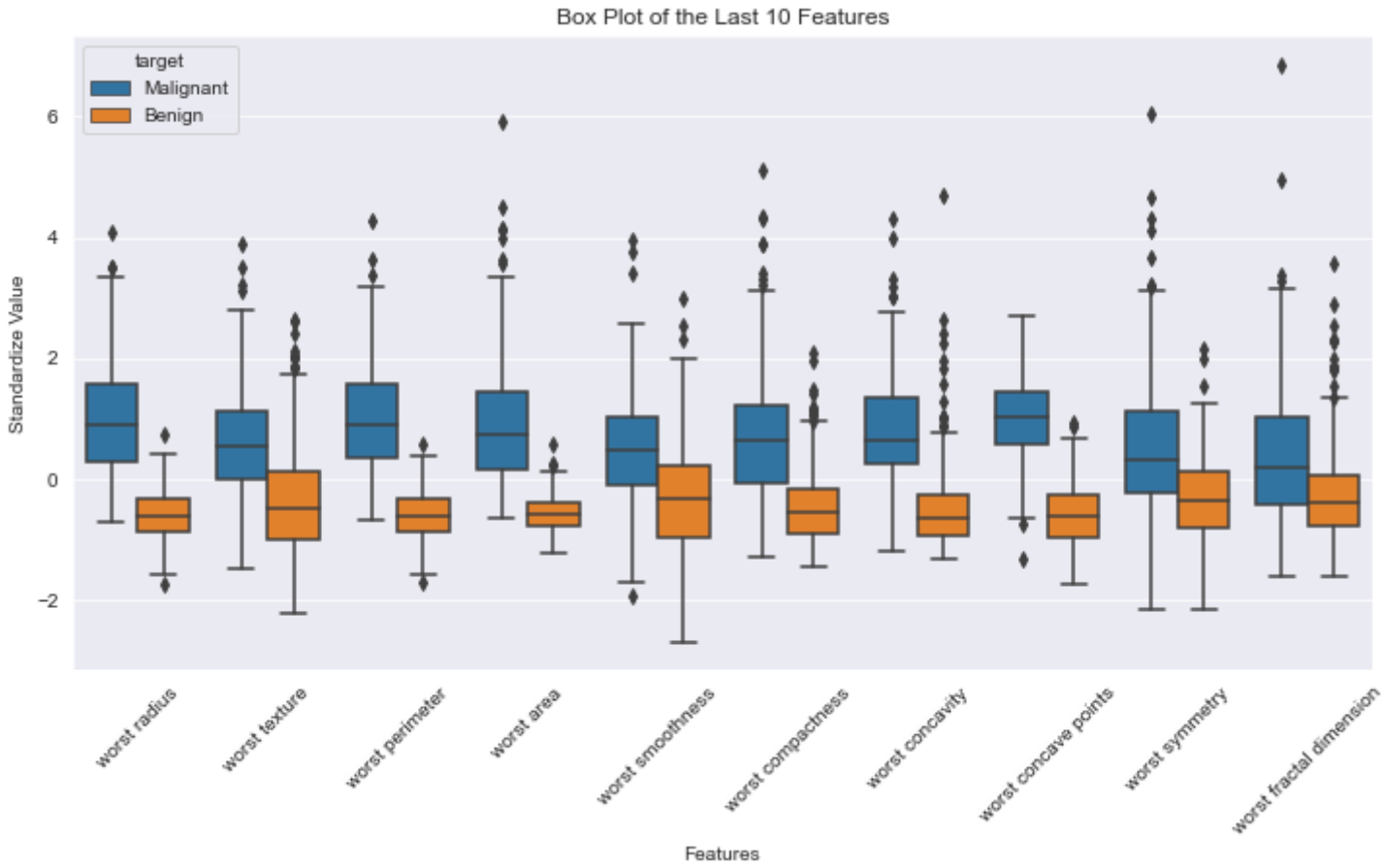
In [27]:
```
box_plot(cancerDF.columns[0:10], "Box Plot of the First 10 Features")
```



In [28]:
```
box_plot(cancerDF.columns[10:20], "Box Plot of the Next 10 Features")
```



In [29]:
```
box_plot(cancerDF.columns[20:30], "Box Plot of the Last 10 Features")
```



In [58]:
```
### The median of some features are very different between "Malignant" and "Benin," such as mean radius, mean a
```

In [59]:
```
### There are also some distributions that are similar between "Malignant" and "Benin," such as mean smoothness
```

In [ ]:
```
### Some features look to be highly correlated with each other, such as mean perimeter & mean area, mean concav
```

In [ ]:
```
## Correlation Plots
```

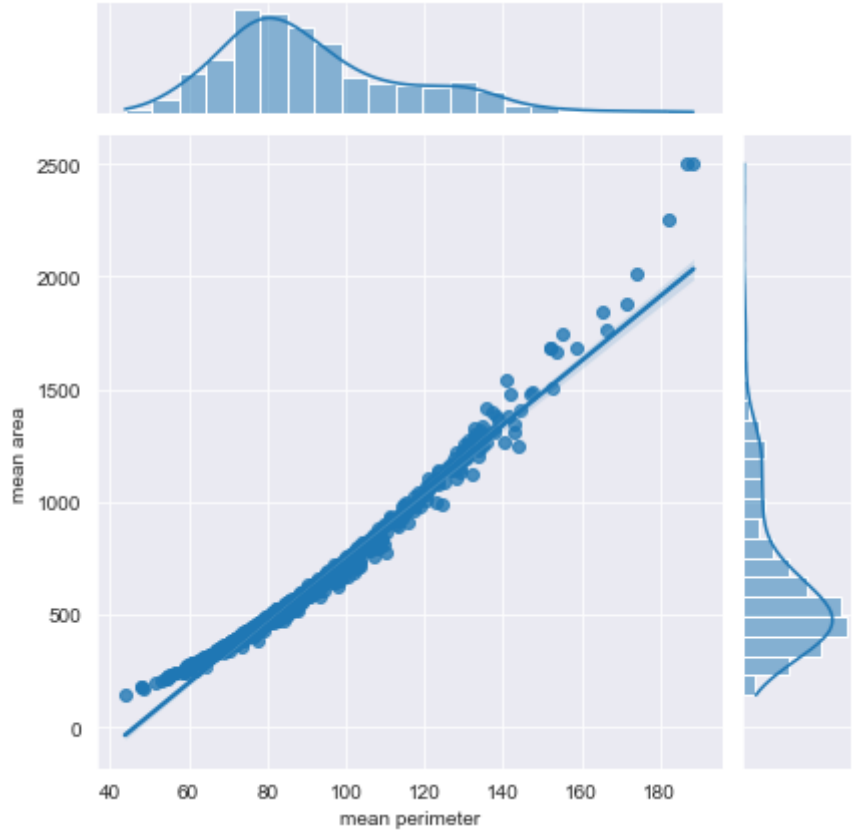
In [ ]:
```
### Let's explore the correlation of three examples above with jointplots.
```

In [ ]:
```
### Jointplots will be used to show how the dependednt variable (y) varies with the independent plot (x), the c
```

In [30]:
```
def correlation(var):
    """
    1. Print correlation
    2. Create jointplot
    """
    # Print correlation
    print("Correlation: ", cancerDF[[var[0], var[1]]].corr().iloc[1, 0])

    # Create jointplot
    plt.figure(figsize=(6, 6))
    sns.jointplot(cancerDF[(var[0])], cancerDF[(var[1])], kind='reg')
```

In [31]:
```
correlation(['mean perimeter', 'mean area'])
```

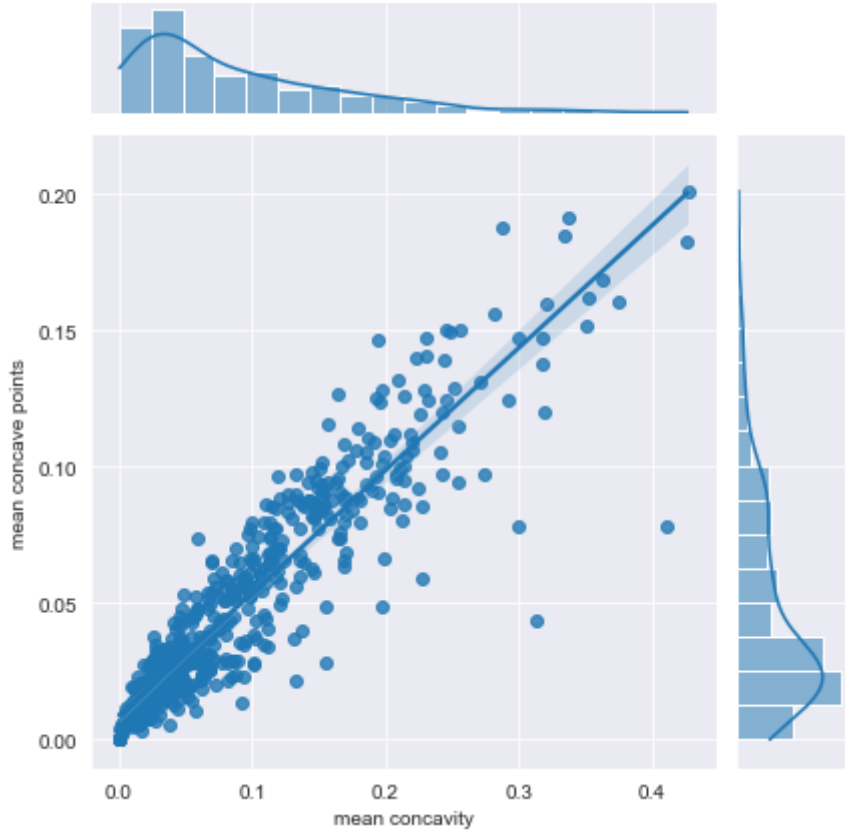
```
Correlation:  0.9865068039913907
/Users/dessiemiller91/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass
the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `d
ata`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
<Figure size 432x432 with 0 Axes>
```



In [ ]:
```
### The above plot indicates postively skewed data and strong correlation between the variables.
```

In [32]:
```
correlation(['mean concavity', 'mean concave points'])
```

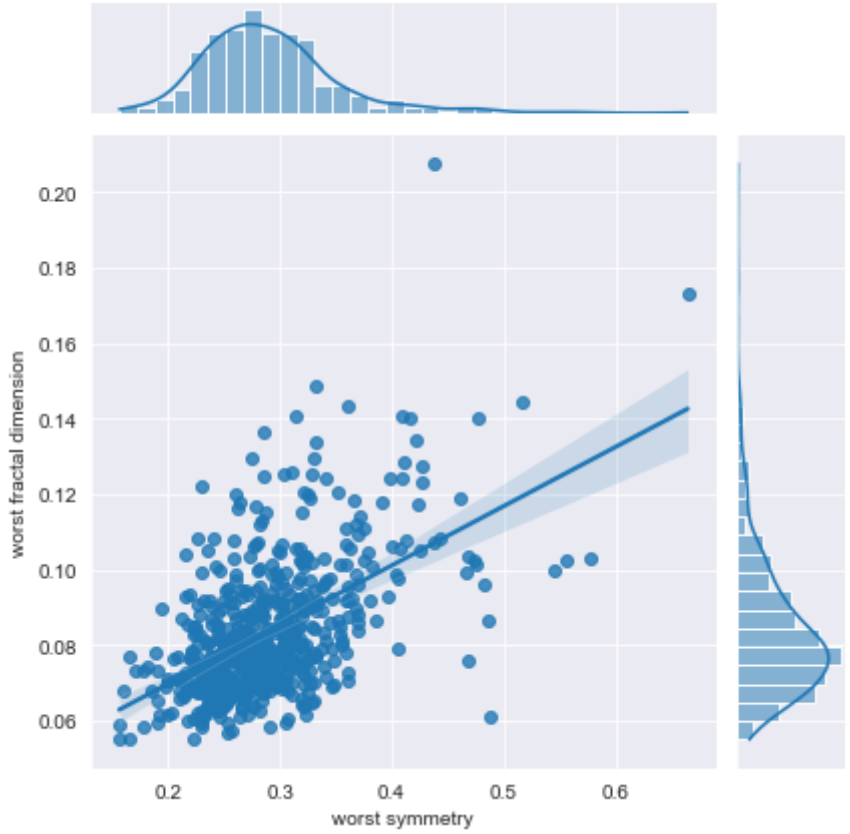
```
Correlation:  0.9213910263788588
/Users/dessiemiller91/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
<Figure size 432x432 with 0 Axes>
```



In [ ]:
```
### The above plot indicates postively skewed data and strong correlation between the variables.
```

In [33]:
```
correlation(['worst symmetry', 'worst fractal dimension'])
```

```
Correlation:  0.537848206253609
/Users/dessiemiller91/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
<Figure size 432x432 with 0 Axes>
```



In [ ]:
```
### The above plot indicates postively skewed data (more bell-shaped than the 2 previous plots) and strong corr
```

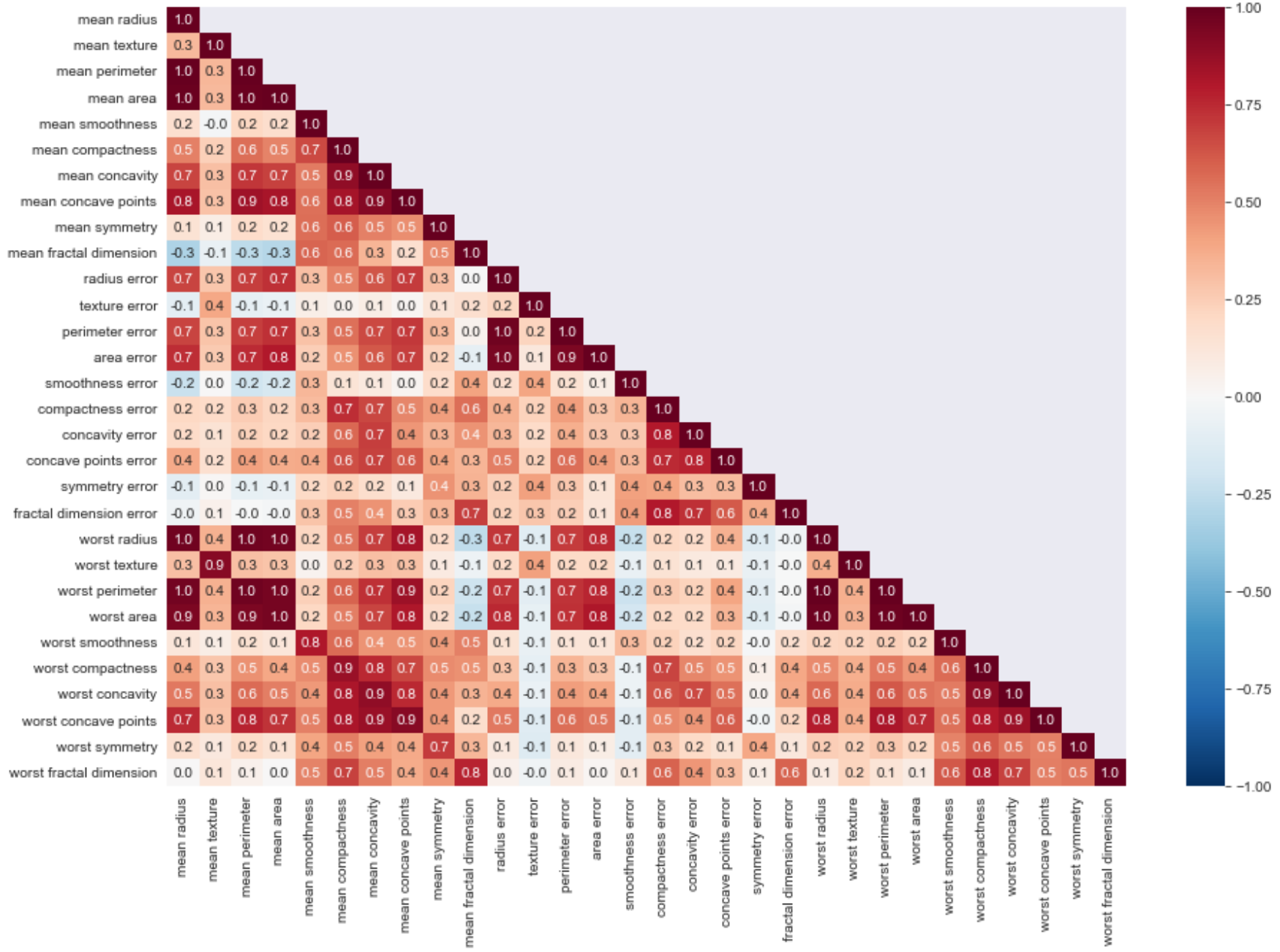
In [ ]:
```
## Heat Maps to Further Illuatrate Correlation
```

In [35]:
```
# Create correlation matrix
corr_mat = cancerDF.corr()

# Create mask
mask = np.zeros_like(corr_mat, dtype=np.bool)
mask[np.triu_indices_from(mask, k=1)] = True

# Plot heatmap
plt.figure(figsize=(15, 10))
sns.heatmap(corr_mat, annot=True, fmt='.1f',
```

```
            cmap='RdBu_r', vmin=-1, vmax=1,
            mask=mask)
```

```
/var/folders/l4/3yv91dvj78l_fby7dphbw4th0000gn/T/ipykernel_14486/606264296.py:5: DeprecationWarning: `np.bool`
is a deprecated alias for the builtin `bool`. To silence this warning, use `bool` by itself. Doing this will no
t modify any behavior and is safe. If you specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#de
precations
  mask = np.zeros_like(corr_mat, dtype=np.bool)
```

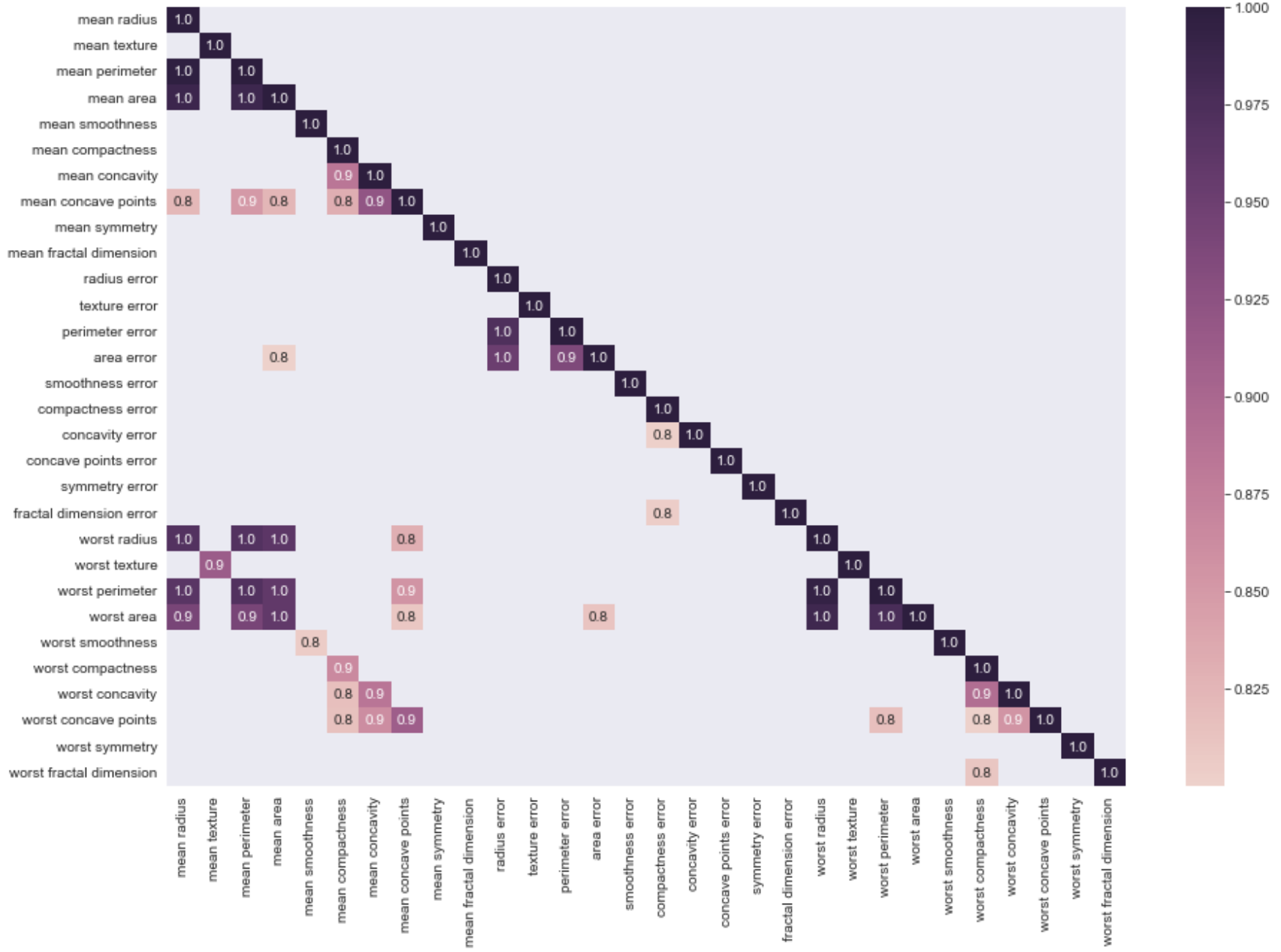
Out[35]:
```
<AxesSubplot:>
```



In [ ]:
```
### From the heat map, we can see that the dependent variables in the dataset are indeed highly correlated. Wha
```

In [36]:
```
plt.figure(figsize=(15, 10))
sns.heatmap(corr_mat[corr_mat > 0.8], annot=True,
            fmt='.1f', cmap=sns.cubehelix_palette(200), mask=mask)
```

Out[36]:
```
<AxesSubplot:>
```



In [ ]:
```
### There's work to be done with feature selection, which is the next step for the next section: Create Model.
```

In [ ]:
```
# Create Model
```

In [ ]:
```
## Decisiion Trees & Random Forests
```

In [ ]:
```
## Feature Selection
```

In [ ]:
```
### I will use Univariate Feature Selection to choose 5 features with the k highest scores. I chose 5 because f
```

In [38]:
```
from sklearn.feature_selection import SelectKBest, chi2
feature_selection = SelectKBest(chi2, k=5)
feature_selection.fit(cancer_features, cancer_target)
selected_features = cancer_features.columns[feature_selection.get_support()]
print("The five selected features are: ", list(selected_features))
```

```
The five selected features are:  ['mean perimeter', 'mean area', 'area error', 'worst perimeter', 'worst area']
```

In [39]:
```
X = pd.DataFrame(feature_selection.transform(cancer_features),
                 columns=selected_features)
X.head()
```

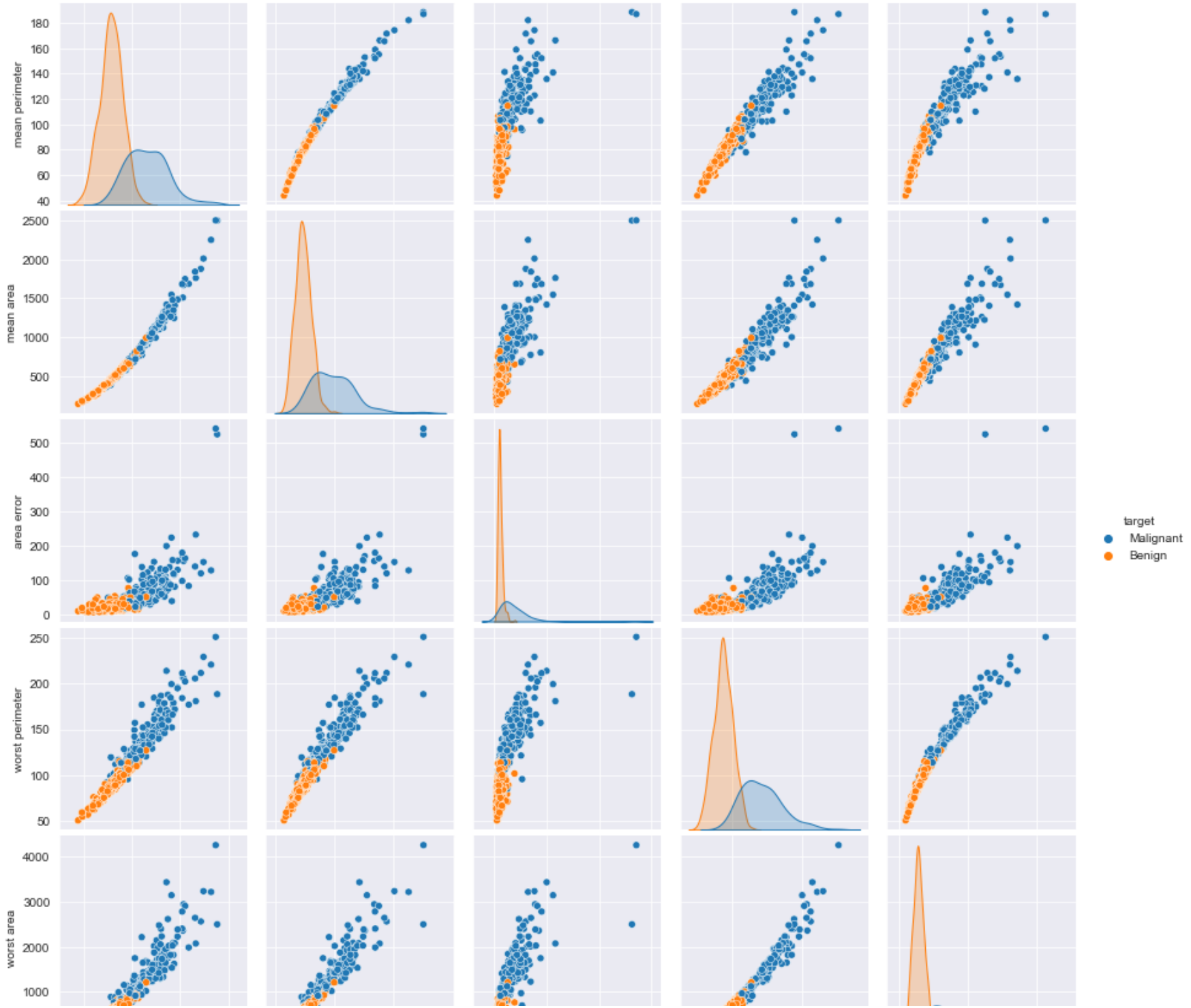
Out[39]:

	mean perimeter	mean area	area error	worst perimeter	worst area
0	122.80	1001.0	153.40	184.60	2019.0
1	132.90	1326.0	74.08	158.80	1956.0
2	130.00	1203.0	94.03	152.50	1709.0
3	77.58	386.1	27.23	98.87	567.7
4	135.10	1297.0	94.44	152.20	1575.0

In [ ]:
```
### Let's create a pairplot to see how different these features are in "malignant" and in "benign."
```

In [40]:
```
sns.pairplot(pd.concat([X, cancerDF['target']], axis=1), hue='target')
```

Out[40]:
```
<seaborn.axisgrid.PairGrid at 0x7fcab76f8850>
```


mean perimeter
mean area
area error
worst perimeter
worst area
target
Malignant
Benign

In [ ]:
```
### Looking good! There is not much variation between the 2 variables in the above pairplots. Let's continue on
```

In [ ]:
```
## Train Test Split
```

In [57]:
```
y = cancer_target['target']
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=42)
```

In [ ]:
```
## Decision Tree
```

In [ ]:
```
### Create initial decision tree.
```

In [48]:
```
decisionTree = DecisionTreeClassifier(random_state=42)
decisionTree.fit(X_train, y_train)
```

Out[48]:
```
DecisionTreeClassifier(random_state=42)
```

In [ ]:
```
### Assess the model and read confusion matrix.
```

In [46]:
```
treePredictions = decisionTree.predict(X_test)
```

In [50]:
```
treePredictions = decisionTree.predict(X_test)
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, treePredictions))
print (classification_report(y_test, treePredictions))
```

```
[[ 61   6]
 [ 14 107]]
              precision    recall  f1-score   support
```

```
              precision    recall  f1-score   support

           0       0.81      0.91      0.86        67
           1       0.95      0.88      0.91       121

    accuracy                           0.89       188
   macro avg       0.88      0.90      0.89       188
weighted avg       0.90      0.89      0.89       188
```

In [ ]:
```
## Random Forest Classifier
```

In [61]:
```
forest = RandomForestClassifier(n_estimators=200, random_state=42)
forest.fit(X_train, y_train)
```

Out[61]:
```
RandomForestClassifier(n_estimators=200, random_state=42)
```

In [ ]:
```
### Evaluate model fit
```

In [62]:
```
forestPredictions = forest.predict(X_test)
print(confusion_matrix(y_test, forestPredictions))
print(classification_report(y_test, forestPredictions))
```

```
[[ 65   2]
 [  6 115]]
              precision    recall  f1-score   support

           0       0.92      0.97      0.94        67
           1       0.98      0.95      0.97       121

    accuracy                           0.96       188
   macro avg       0.95      0.96      0.95       188
weighted avg       0.96      0.96      0.96       188
```

In [ ]:
```
### The accuracy rate is approximately 96%. The model only makes 7 wrong predictions out of 188. The chosen fea
```

In [ ]:
```
## k-Nearest Neighbors (KNN) Classification
```

In [ ]:
```
### Goal: The KNN analysis will help predict whehter cancer is "malignant" or "benign" and get a better read or
```

In [ ]:
```
## Scale Data
```

In [63]:
```
scaler = StandardScaler()
scaler.fit(cancerDF.drop('target', axis=1))
scaledVariables = scaler.transform(cancerDF.drop('target',axis=1))
cancerScaled = pd.DataFrame(scaledVariables, columns=cancerDF.columns[:-1])
```

In [64]:
```
cancerScaled.head()
```

Out[64]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius
0	1.097064	-2.073335	1.269934	0.984375	1.568466	3.283515	2.652874	2.532475	2.217515	2.255747	...	1.886690
1	1.829821	-0.353632	1.685955	1.908708	-0.826962	-0.487072	-0.023846	0.548144	0.001392	-0.868652	...	1.805927
2	1.579888	0.456187	1.566503	1.558884	0.942210	1.052926	1.363478	2.037231	0.939685	-0.398008	...	1.511870
3	-0.768909	0.253732	-0.592687	-0.764464	3.283553	3.402909	1.915897	1.451707	2.867383	4.910919	...	-0.281464
4	1.750297	-1.151816	1.776573	1.826229	0.280372	0.539340	1.371011	1.428493	-0.009560	-0.562450	...	1.298575

5 rows × 30 columns

In [ ]:
```
## Create X and Y Datasets
```

In [65]:
```
X = cancerScaled
y = cancerDF['target']
```

In [ ]:
```
## Train Test Split
```

In [66]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.33, random_state=42)
```

In [ ]:
```
## KNN Analysis
```

In [69]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
```

In [70]:
```
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
pred = knn.predict(X_test)
```

In [ ]:
```
## Interpret KNN Predictions
```

In [71]:
```
print(confusion_matrix(y_test, pred))
```

```
[[116   5]
 [  6  61]]
```

In [72]:
```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

      Benign       0.95      0.96      0.95       121
   Malignant       0.92      0.91      0.92        67

    accuracy                           0.94       188
   macro avg       0.94      0.93      0.94       188
weighted avg       0.94      0.94      0.94       188
```

In [ ]:
```
### Looking at the precision column, the KNN algorithm was 95% correct about predicting cancer being "benign" a
```

In [ ]:
```
## Choose the Best Fit Model
```

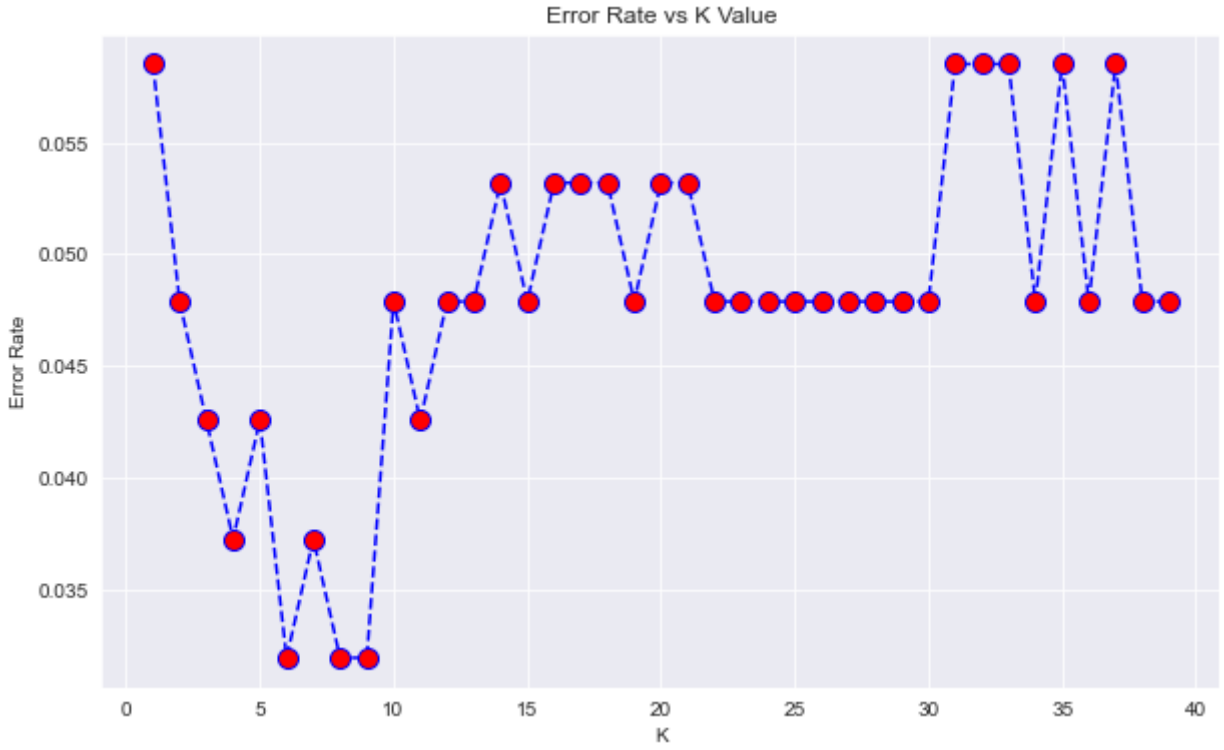
In [73]:
```
errorRate = []
for i in range(1,40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
```

```
    predI = knn.predict(X_test)
    errorRate.append(np.mean(predI != y_test))
```

In [ ]:
```
### Plot above model.
```

In [74]:
```
plt.figure(figsize=(10,6))
plt.plot(range(1,40), errorRate, color='blue', linestyle='dashed', marker='o', markerfacecolor='red', markersiz
plt.title('Error Rate vs K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

Out[74]:
```
Text(0, 0.5, 'Error Rate')
```



In [ ]:
```
### In this case, 6, 8, and 9 are all k values that are equally low. Almost no error there!
```

In [ ]:
```
## Run the Final Model
```

In [90]:
```
knn = KNeighborsClassifier(n_neighbors=8)
knn.fit(X_train, y_train)
pred = knn.predict(X_test)
```

In [91]:
```
print(confusion_matrix(y_test, pred))
```

```
[[119   2]
 [  4  63]]
```

In [92]:
```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

      Benign       0.97      0.98      0.98       121
   Malignant       0.97      0.94      0.95        67

    accuracy                           0.97       188
   macro avg       0.97      0.96      0.96       188
weighted avg       0.97      0.97      0.97       188
```

In [ ]:
```
### After running the KNN again with k=8, the KNN algorithm was able to predict the whether cancer was "maligna
```

In [ ]:
```
# Summary
```

In [ ]:
```
### In the first part of this project, I explored the data to get a feel for what I was dealing with and how be
```

In [ ]:
```
### Next, I did exploratory data analysis to better understand each of 30 original features, how they might be
```

In [ ]:
```
### Then I selected 5 best features for my model using univariate feature selection, and performed both Decisic
```

In [ ]:
```
### Additionally, I performed k-Nearest Neighbors (KNN) Classification where the main goal was predict whehter
```

In [ ]:
```
### The initial accuracy rate for the KNN algorithm was 95% correct about predicting cancer being "benign" and
```

In [ ]:
```
### After running the KNN again with k=8, the KNN algorithm was able to predict whether cancer was "malignant"
```

In [ ]:
```
### Finally, the Wisconsin Breast Cancer dataset was easy to work with and for machine learning models to class
```

In [ ]:
```
# Citation: Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvi
```