DATA SCIENCE

Classification Problems with Logistic Regression, KNN, Decision Tree

IINTRODUCTION: 3 Types of Problems in Data Science:

1

Regression

The estimation of a continuous dependent variables or response from a list of input variables (features).

Supervised learning algorithm.

2

Classifiction

The process of separating and organizing data into relevant groups (classes) based on their shared characteristics.

Supervised learning algorithm.

3

Clustering

The process of segregating groups with similar traits and assigning them into clusters.

Unsupervised learning algorithm.

PROBLEM STATEMENT: Classification Problem:

1

Dataset

Dataset from Kaggle.
Dataset on breast cancer.
Total features: 32 features.
Total objects: 569.

2

Goals

Predict / Classify tumors into malignant (cancerous) or benign (non-cancerous):

- 1 (Malignant).
- 0 (Benign).

3

Ideas

Perform classification methods on the dataset to create the models.

Use evaluation metrics to interpret the goodness of the model and to compare among models of different methods.

METHODS: 3 Classification Methods / Algorithms:

1

Logistic Regression

The algorithm find a relationship between features and probability of particular outcome.

Advantages:

- Easy to implement and interpret.
- Performs well on low-dimensional data.

Disadvantages:

- Overfits on high dimensional data.

2

kNN

The algorithm assumes that similar things exist in close proximity and finds n number of neighbors

Advantages:

- Simple and easy to implement.
- Can be used for classification, and regression

Disadvantages:

- Get significantly slower as the number of examples, predictors / independent variables increase.

3

Decision Tree

The algorithm creates a training model that can use to predict the class or value of the target variable by learning simple decision rules inferred from training data.

Advantages:

- Less effort for data preprocessing.
- Very intuitive and easy to explain.

Disadvantages:

- Non-robust.
- Overfitting.

PROGRESS:

1

Data Exporation

Overview of dataset.

Check for missing values.

Check for multicollinearity.

Visualize important features.

IMPLEMENTATION + INTERPRETATION: Data Exploration:

Overview of dataset

No missing value.

32 features:

- 1 categorical (Diagnosis): also 'Result' variable in this dataset.
- 31 numerical.

```
In [73]: #look for any missing values
         data_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), int64(1), object(1)
memory usage: 142.4+ KB
```

IMPLEMENTATION + INTERPRETATION: Data Exploration:

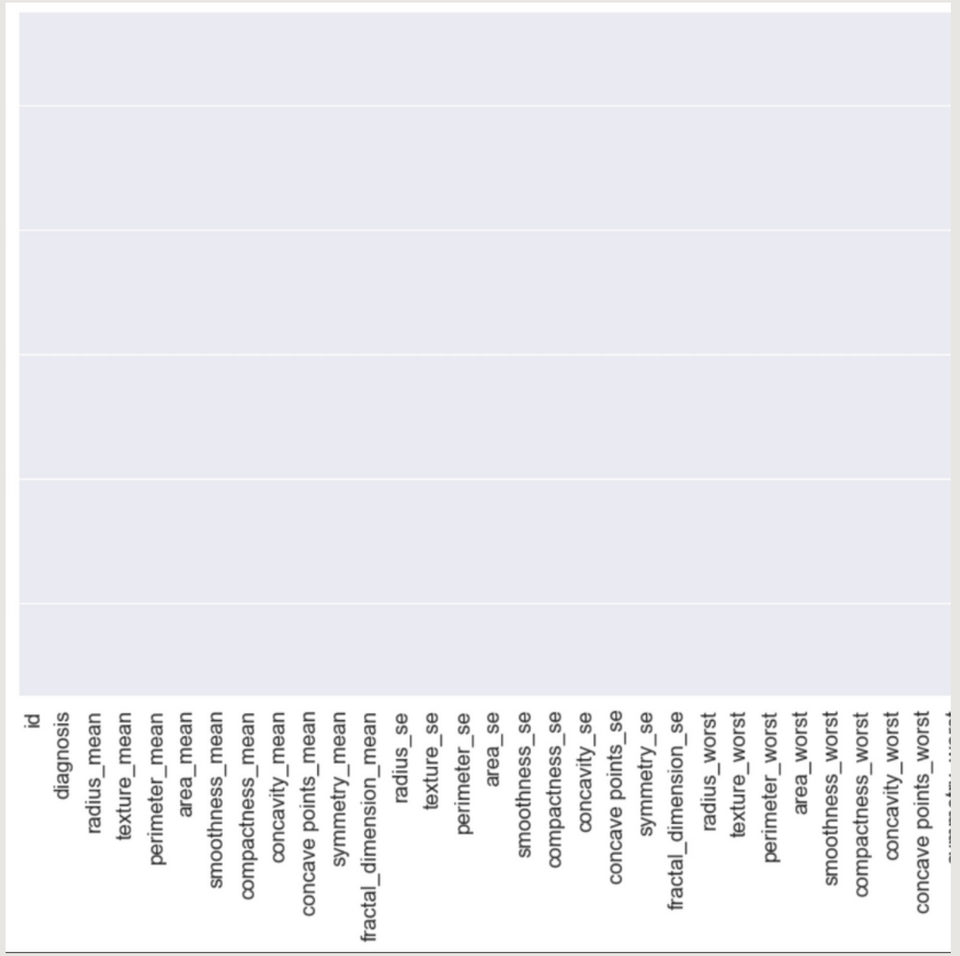
Missing value

We used the isnull() to see which values came up as null so that we could take care of them before making out models.

The dataset turned out to not have any missing data as shown in the pictures on the right.

```
data_df.isnull().sum()
```

```
Output exceeds the size limit. Open the full output data in a text editor
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
...
concavity_worst           0
concave points_worst      0
symmetry_worst            0
fractal_dimension_worst   0
dtype: int64
```



IMPLEMENTATION + INTERPRETATION: Data Exploration:

Multicollinearity

Correlation Heatmap

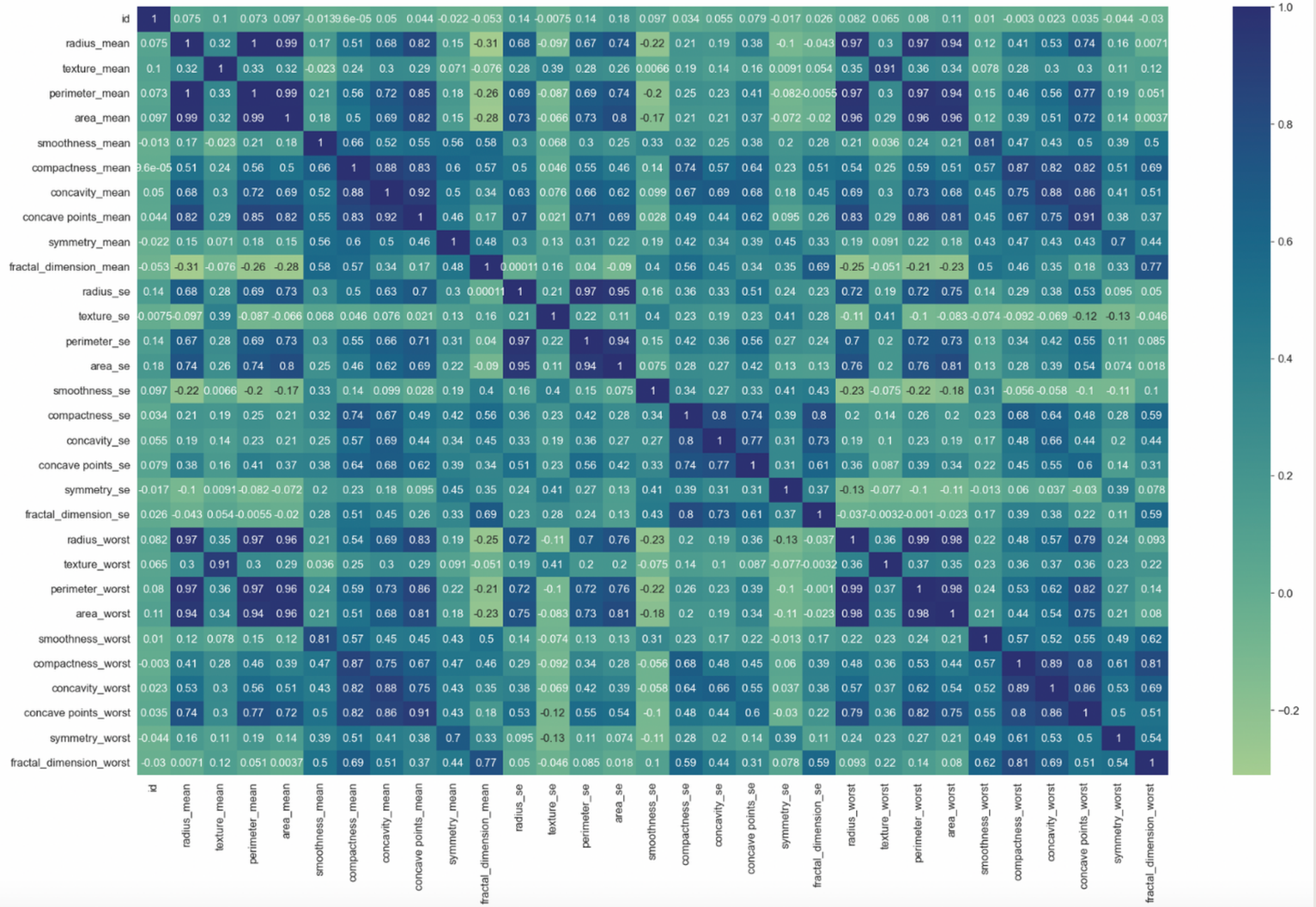
Strong positive relationship:

perimeter_mean & area_mean, concave_points_mean & concavity_mean, area_se & perimeter_se, radius_worst & perimeter_worst, area_worst & perimeter_worst

Negative relationship:

perimeter_mean & symmetry_mean, symmetry_se & radius_worst, fractal_dimension_mean & perimeter_mean, area_worst & fractal_dimension_worst, radius_worst & symmetry worst

```
In [77]: plt.figure(figsize=(25,15))
         sns.heatmap(data_df.corr(), annot=True, cmap = 'crest')
         plt.show()
```



Dataset: Attributes:

- id
- diagnosis (B = Benign, M = Malignant)

Mean

- radius
- texture
- perimeter
- area
- smoothness
- compactness
- concavity
- concave_points
- symmetry
- fractal_dimension

Standard Error(SE)

- radius
- texture
- perimeter
- area
- smoothness
- compactness
- concavity
- concave_points
- symmetry
- fractal_dimension

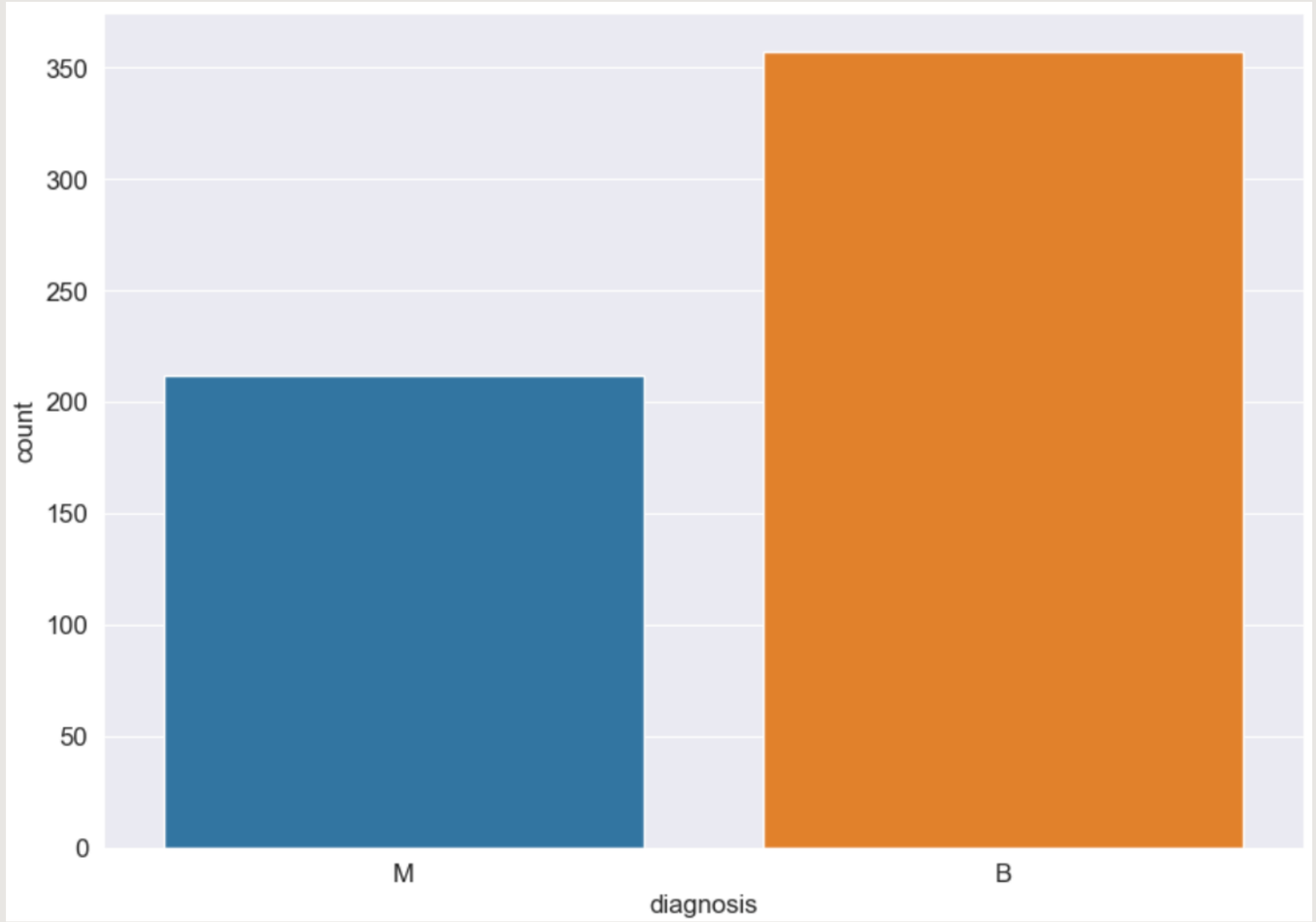
Worst

- radius
- texture
- perimeter
- area
- smoothness
- compactness
- concavity
- concave_points
- symmetry
- fractal_dimension

IMPLEMENTATION + INTERPRETATION: Data Exploration:

Visualize important features:

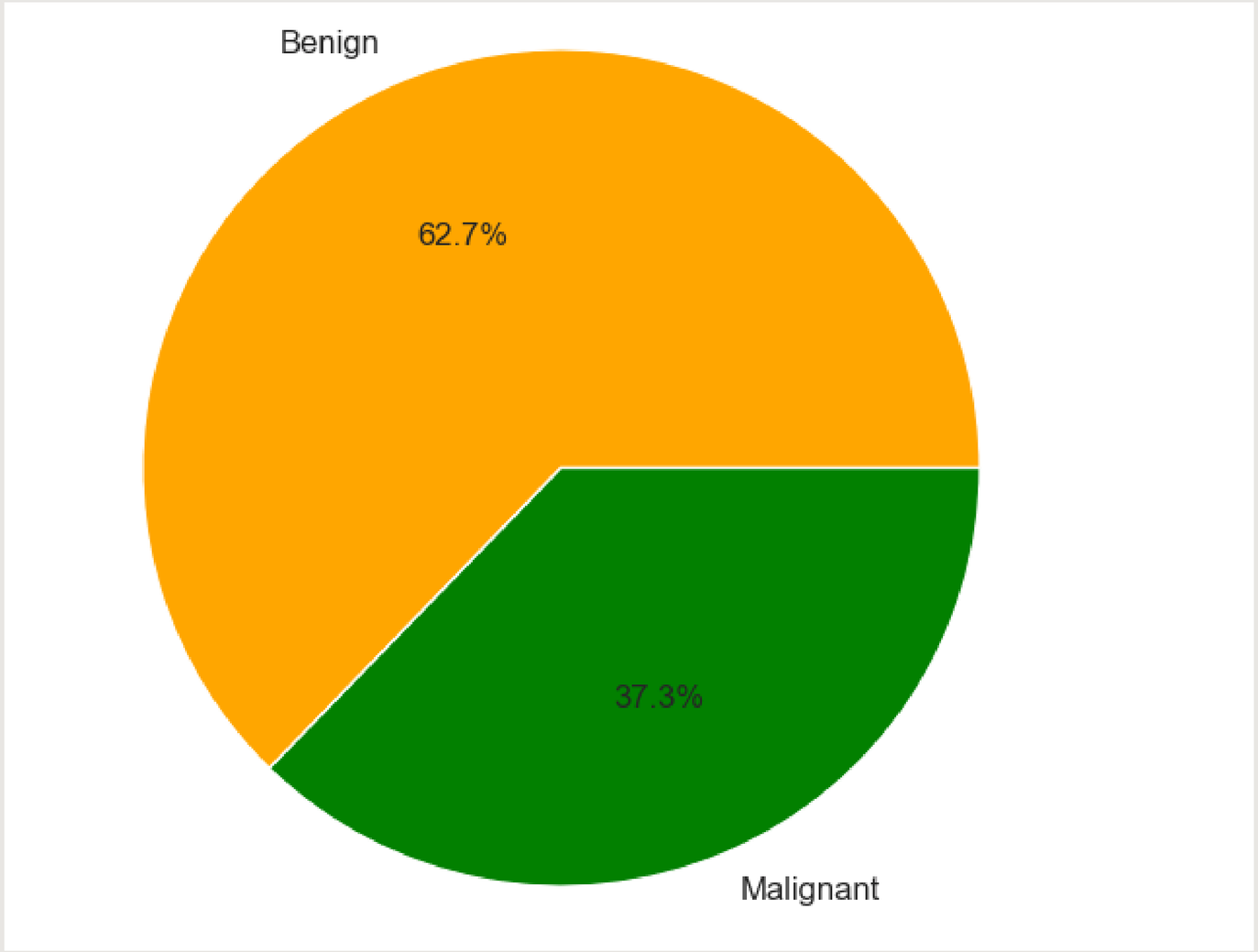
Count Plot:
We wanted to see the split between malignant and benign in the dataset so we made a count plot to see which diagnosis was higher. Looking at the graph to the right we see that benign (B) is much greater than malignant (M).



IMPLEMENTATION + INTERPRETATION: Data Exploration:

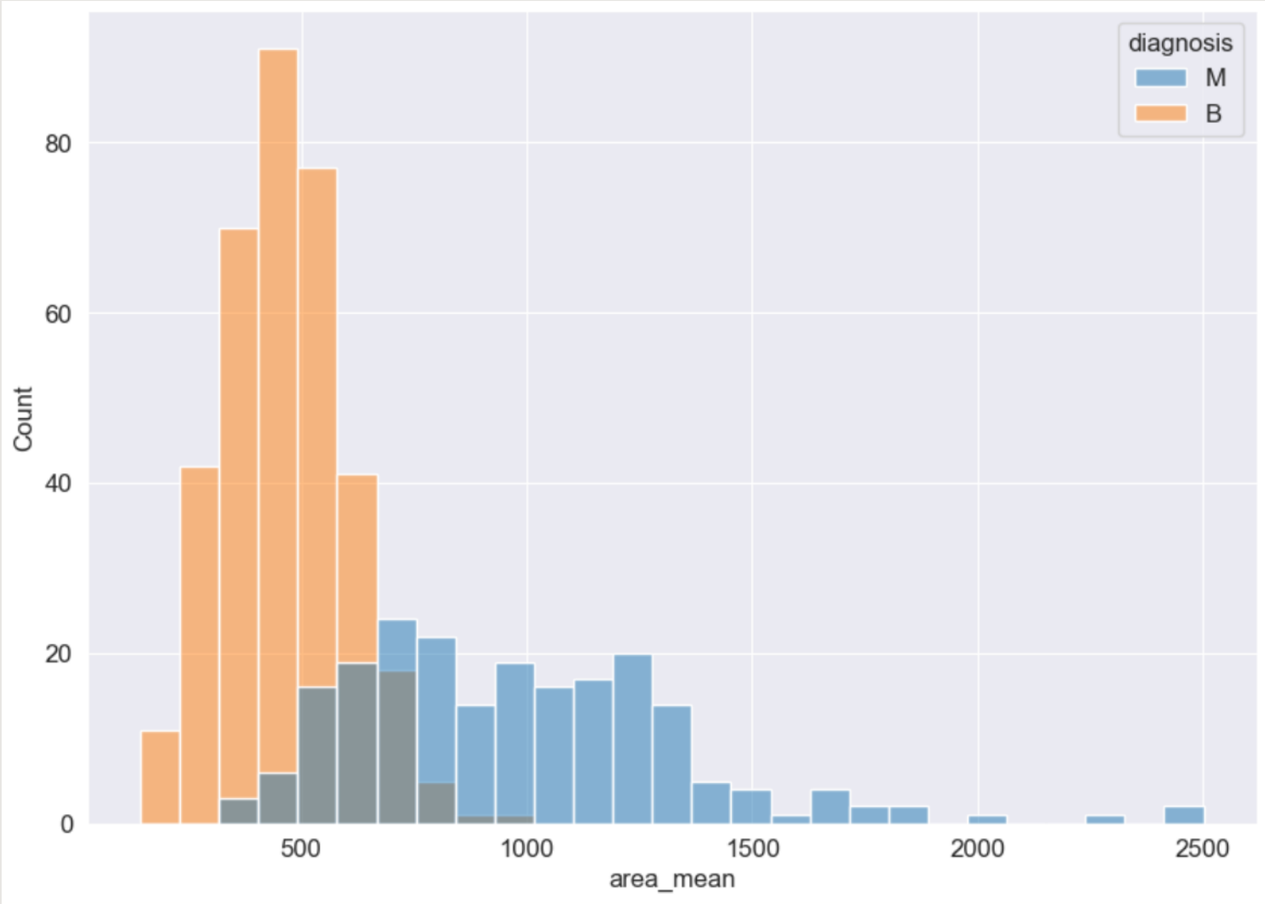
Pie Chart:
The purpose of creating a piechart was to look for accurate percentages for each type of diagnosis. Thus, benign is 62.7% while Malignant is 37.3% .

The models can be biased toward the 'benign' class.



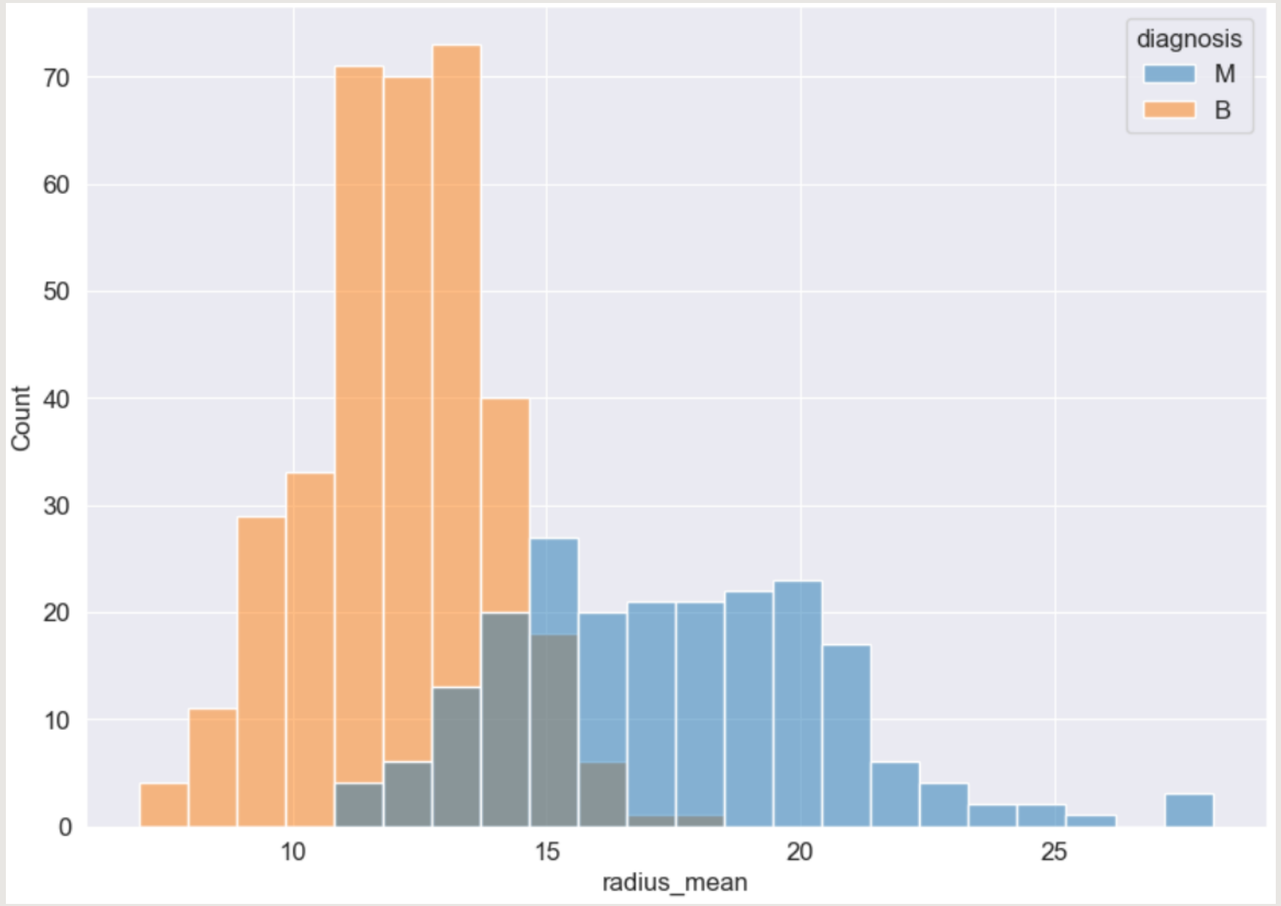
IMPLEMENTATION + INTERPRETATION: Data Exploration:

Hist Plot:
We wanted to see which predictors would be good to see the separation between malignant or benign so we made hist plots to see the separation. The way we saw this was to look at the amount of overlap between the two diagnoses. The graphs to the right with the least amount of overlap would be good predictors, in this case, we saw radius_mean, area_mean, and perimeter_mean to be the best predictors. The graph on the right is area_mean.



IMPLEMENTATION + INTERPRETATION: Data Exploration:

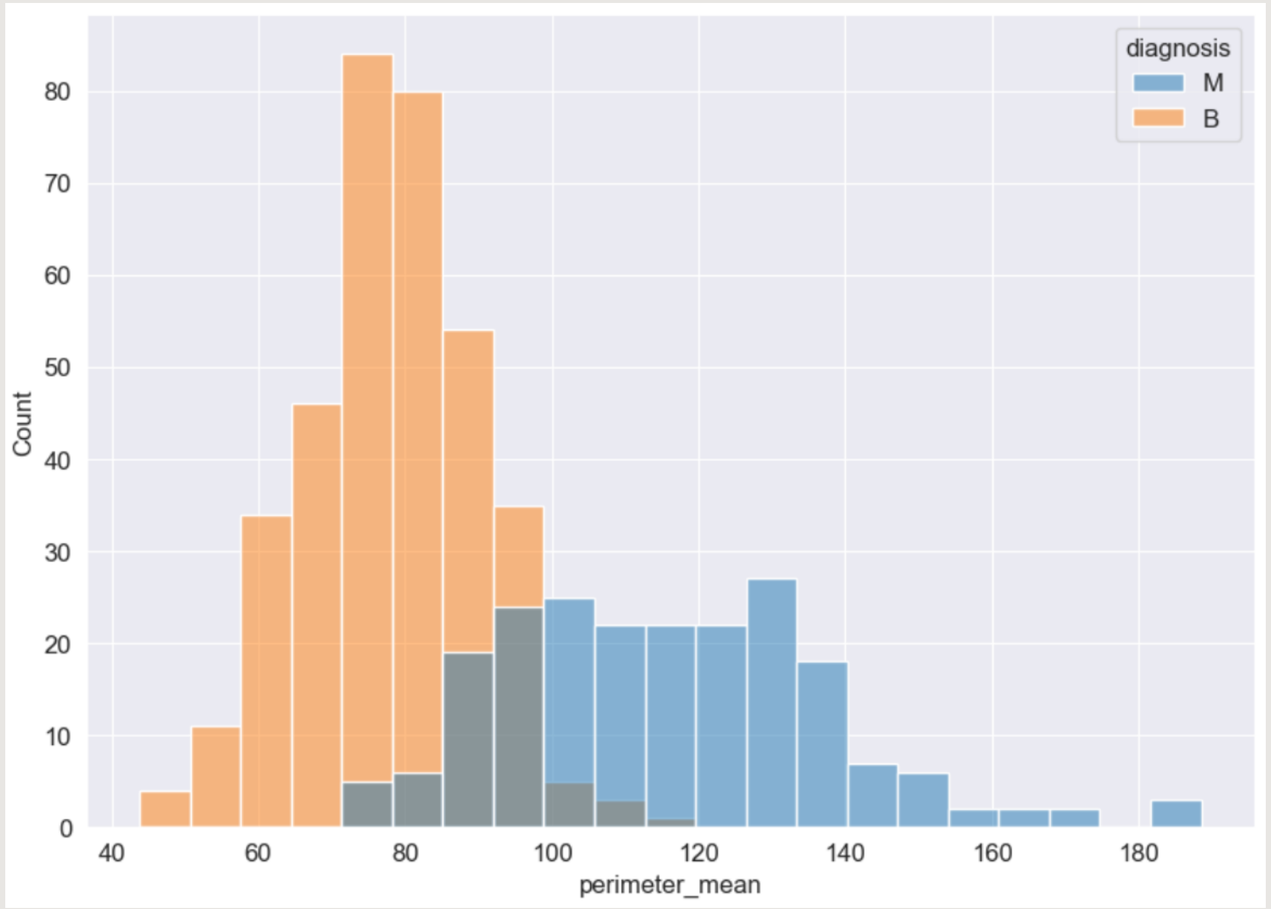
Hist Plot:
The graph on the right is radius_mean.



IMPLEMENTATION + INTERPRETATION: Data Exploration:

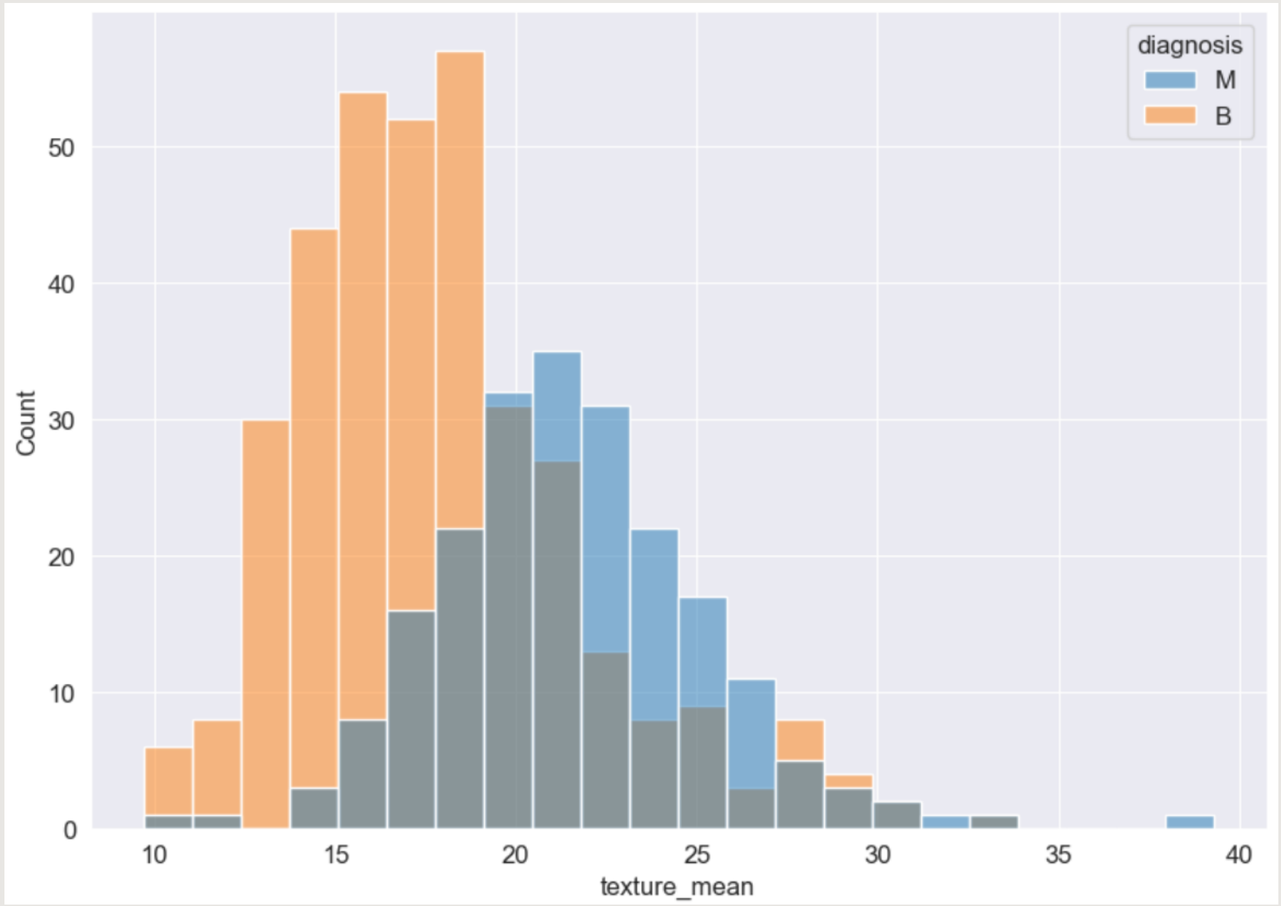
Hist Plot:

The graph on the right is perimeter_mean.



IMPLEMENTATION + INTERPRETATION: Data Exploration:

Hist Plot:
The graph on the right is texture_mean. As we can see from the graph this is not a good predictor since it has a lot of overlap between the two diagnoses.



Pair Plot:

We used variables such as radius_se, texture_se, perimeter_se, area_se, smoothness_se, compactness_se, concavity_se, concave points_se, symmetry_se, and fractal_dimension_se to create pair plot with diagnosis parameter.

For the most of the above mentioned variables, there has been a clear differentiation between benign and malignant diagnoses.

PROGRESS:

1

Data Exporation

Overview of dataset.

Check for missing values.

Check for multicollinearity.

Visualize important features.

2

Data Preprocessing

Take care of missing values if any.

Drop unimportant features.

Dummy variables for categorical variable:

- 'B': 0.
- 'M': 1.

Scale the dataset.

Split training / testing sets.

IMPLEMENTATION + INTERPRETATION: Data Preprocessing:

Drop 'id' Feature + Dummy Variables for 'Diagnosis' Feature:

```
In [20]: #DATA PREPROCESSING
         data_df.drop('id', axis=1, inplace=True)

In [21]: diagnosis_mappings = {'B': 0, 'M': 1}

In [22]: data_df['diagnosis'] = data_df['diagnosis'].map(diagnosis_mappings)

In [23]: data_df.info()

In [24]: data_df.head()
```

Out[24]:

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

5 rows × 31 columns

IMPLEMENTATION + INTERPRETATION: Data Preprocessing:

Scale Data using Standard Scaler:

In [25]:
```
X = data_df.drop('diagnosis', axis=1)
y = data_df['diagnosis']

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled_df = pd.DataFrame(data=X_scaled, columns=X.columns)
```

In [26]:
```
X_scaled_df.head()
```

Out[26]:

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

5 rows × 30 columns

IMPLEMENTATION + INTERPRETATION: Data Preprocessing:

Split Training and Testing Sets:

In [27]:
```
# split data into training and testing sets:

X_train, X_test, y_train, y_test = train_test_split(X_scaled_df, y, test_size=0.3, random_state=42)
```

In [28]:
```
X_train
```

Out[28]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_di
149	-0.109996	-0.321053	-0.158542	-0.198772	-1.204139	-0.769070	-0.753164	-0.919018	-1.236277	
124	-0.215082	-0.674768	-0.241747	-0.288361	-1.794101	-0.589220	-0.098925	-0.539588	-1.422476	
421	0.159817	-1.235591	0.257479	0.003444	0.479635	1.502076	0.705598	0.363201	1.001751	
195	-0.345728	-0.688730	-0.388796	-0.393877	-1.206274	-0.960480	-0.628619	-0.648697	0.063458	
545	-0.144078	0.916946	-0.196849	-0.232332	-0.277565	-0.698760	-0.741488	-0.631673	-0.538947	
...	...	...	...	...	...	...	...	...	...	
71	-1.488033	-1.082004	-1.366651	-1.168611	0.104593	0.924055	-0.034392	-0.521016	0.329977	
106	-0.706426	-0.223317	-0.691956	-0.689379	1.269571	-0.050051	-0.227236	-0.362899	-0.038768	
270	0.046211	-0.574704	-0.068748	-0.063392	-2.282296	-1.470464	-1.023849	-1.100607	-1.108494	
435	-0.041833	0.076875	-0.034972	-0.157532	0.686015	0.169787	0.298817	0.405245	-0.520693	
102	-0.553058	0.286311	-0.607516	-0.557982	-1.155035	-1.212155	-0.815688	-0.805266	-0.265127	

398 rows × 30 columns

PROGRESS:

1

Data Exporation

Overview of dataset.

Check for missing values.

Check for multicollinearity.

Visualize important features.

2

Data Preprocessing

Take care of missing values if any.

Drop unimportant feature.

Dummy variables for categorical variable:

- 'B': 0.
- 'M': 1.

Scale the dataset.

Split training / testing sets.

3

Model + Evaluation

Logistic Regression

IMPLEMENTATION + INTERPRETATION: Logistic Regression:

```
In [30]: #Logistic Regression
         logmodel = LogisticRegression(max_iter=1000)
         logmodel.fit(X_train,y_train)

Out[30]: LogisticRegression(max_iter=1000)

In [31]: log_pred = logmodel.predict(X_test)

In [32]: print(classification_report(y_test,log_pred))
```

	precision	recall	f1-score	support
0	0.99	0.98	0.99	108
1	0.97	0.98	0.98	63
accuracy			0.98	171
macro avg	0.98	0.98	0.98	171
weighted avg	0.98	0.98	0.98	171

Interpretation of Result:
Overall, the model gives us the high accuracy of 0.98.

F1-Score: 0.99 for class 0, 0.98 for class 1.

Slightly better prediction for class 0.

PROGRESS:

1

Data Exporation

Overview of dataset.

Check for missing values.

Check for multicollinearity.

Visualize important features.

2

Data Preprocessing

Take care of missing values if any.

Drop unimportant feature.

Dummy variables for categorical variable:

- 'B': 0.
- 'M': 1.

Scale the dataset.

Split training / testing sets.

3

Model + Evaluation

Logistic Regression

KNN

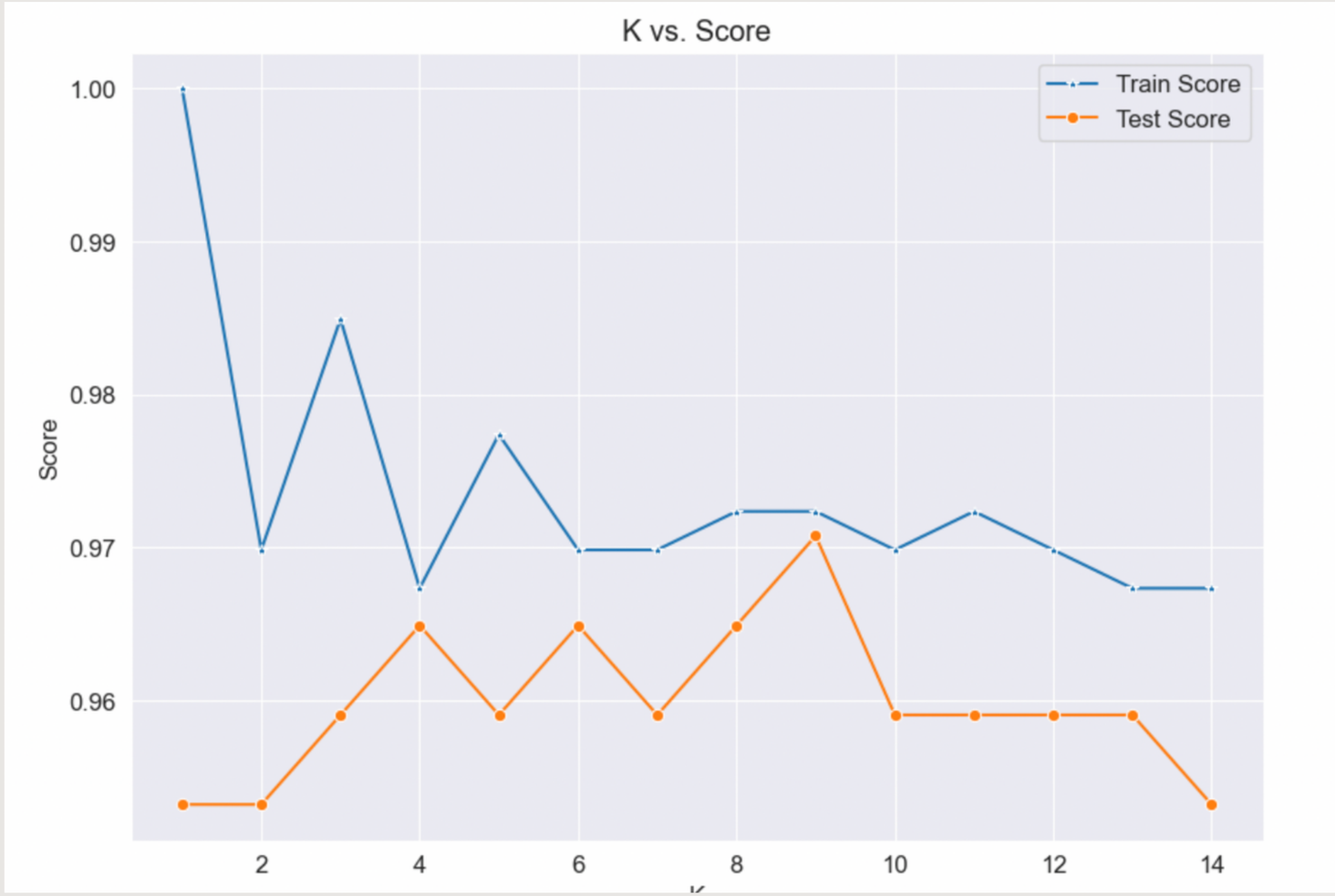
IMPLEMENTATION + INTERPRETATION: KNN: Find the best K:

In [33]:
```
#KNN
test_scores = []
train_scores = []

# testing k values from 1-14
for i in range(1,15):
    knn = KNeighborsClassifier(i)
    knn.fit(X_train,y_train)
    # append scores.
    train_scores.append(knn.score(X_train,y_train))
    test_scores.append(knn.score(X_test,y_test))
```

In [34]:
```
sns.lineplot(x=range(1,15), y=train_scores, marker='*', label='Train Score')
sns.lineplot(x=range(1,15), y=test_scores, marker='o', label='Test Score')
plt.title('K vs. Score')
plt.xlabel('K')
plt.ylabel('Score')
plt.show()
```

IMPLEMENTATION + INTERPRETATION: KNN: K = 9



IMPLEMENTATION + INTERPRETATION: KNN: K = 9

In [47]:
```
#Checking to see which K score is the best
#Turns out the best K score is 9
max=0
index=0
for i in range(1,15):
    knn = KNeighborsClassifier(i)

    knn.fit(X_train,y_train)
    testing = knn.score(X_test,y_test)
    if testing > max:
        max = testing
        index = i

knn = KNeighborsClassifier(9)

knn.fit(X_train,y_train)
knn.score(X_test,y_test)
```

Out[47]: 0.9707602339181286

In [48]:
```
knn_pred = knn.predict(X_test)

print(classification_report(y_test, knn_pred))
```

	precision	recall	f1-score	support
0	0.97	0.98	0.98	108
1	0.97	0.95	0.96	63
accuracy			0.97	171
macro avg	0.97	0.97	0.97	171
weighted avg	0.97	0.97	0.97	171

Interpretation of Result:

The KNN classification report suggested that the model is slightly better at predicting benign samples compared to malignant samples.

This is because of the higher precision, recall, and F1 score.

The precision for both classes is 97 percent which means when the model predicts a sample to be M or B it's correct 97 percent of the time.

For recall the scores mean that when there are actual M or B samples in the testing set the model can identify it correctly around 95 to 98 percent of the time.

Lastly, the F1 score means that it takes into account both precision and recall and shows that the model is performing well in both classes.

PROGRESS:

1

Data Exporation

Overview of dataset.

Check for missing values.

Check for multicollinearity.

Visualize important features.

2

Data Preprocessing

Take care of missing values if any.

Drop unimportant feature.

Dummy variables for categorical variable:

- 'B': 0.
- 'M': 1.

Scale the dataset.

Split training / testing sets.

3

Model + Evaluation

Logistic Regression
KNN
Decision Tree

IMPLEMENTATION + INTERPRETATION: Decision Tree:

```
In [39]: #Decision Trees

         from sklearn.tree import DecisionTreeClassifier

         clf_gini = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=0)

         # fit the model
         clf_gini.fit(X_train, y_train)
```

```
Out[39]: DecisionTreeClassifier(max_depth=3, random_state=0)
```

```
In [40]: y_pred_gini = clf_gini.predict(X_test)
```

```
In [44]: print(classification_report(y_test, y_pred_gini))
```

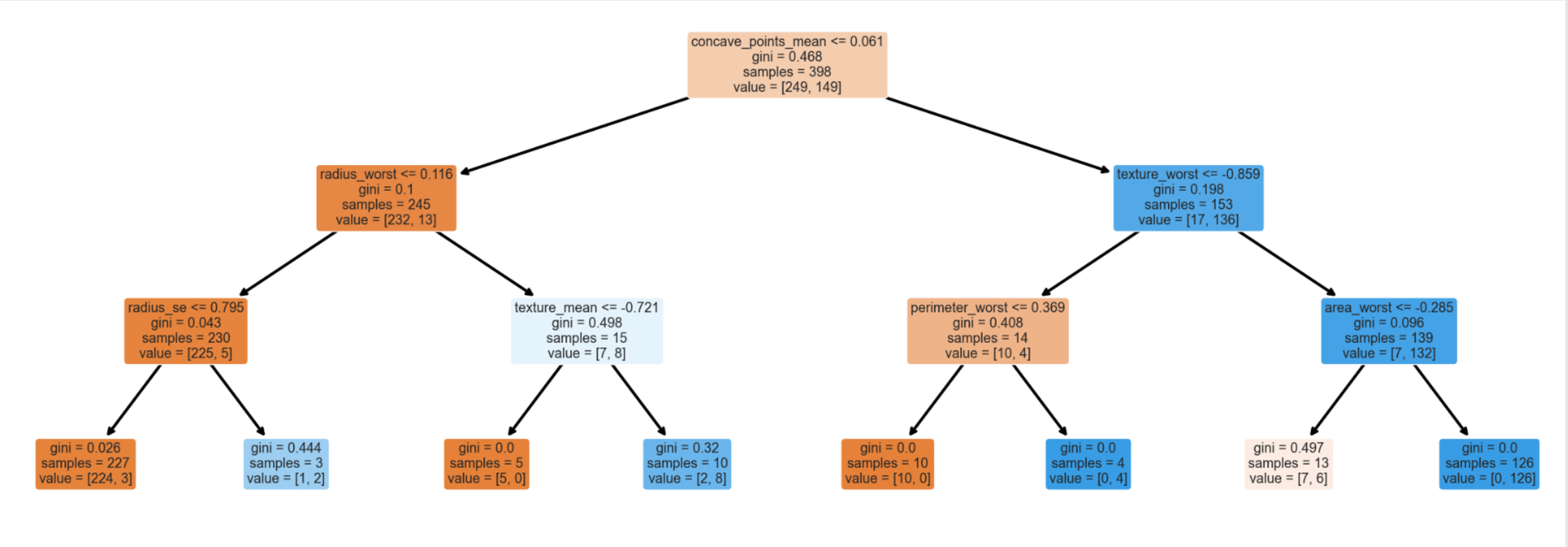
	precision	recall	f1-score	support
0	0.97	0.97	0.97	108
1	0.95	0.95	0.95	63
accuracy			0.96	171
macro avg	0.96	0.96	0.96	171
weighted avg	0.96	0.96	0.96	171

Interpretation of Result:

Accuracy: 0.96.
F1-Score: 0.97 for class 0; 0.95 for class 1.

According to the KNN classification report, the model classified benign samples slightly more accurately than malignant samples.
The precision for class 0 (Benign) is 0.97 while for class 1 (Malignant) is 0.95. That means, the class 0 has more correct positive predictions.

IMPLEMENTATION + INTERPRETATION: Visualize Decision Tree:



MODEL COMPARISON: Metrics on Testing Set:

1

Metrics: Accuracy:

A metric that measures the number of correct predictions by the model in relation to the total number of predictions made.

2

Conclusion:

Logistic Regression: 0.9824561403508771
KNN: 0.9707602339181286
Decision Tree: 0.9649122807017544

Logistic Regression has the highest accuracy (0.99). Hence, it is the best model for this dataset.
All 3 methods give a high accuracies (>0.95) which indicates that all 3 methods are able to satisfy the goal of classification.

RESULTS / MODEL EVALUATION:

1

Metrics: F1- Score

A metric that combines the precision and recall scores of a model.

It computes how many times a model made a correct prediction across the entire dataset.

2

Conclusion

Logistic Regression: 0.99 (0), 0.98 (1)

KNN: 0.98 (0), 0.96 (1)

Decision Tree: 0.97 (0), 0.95 (1)

Slightly bias toward class 0.

All 3 methods have high F1-scores.

Logistic Regression has highest values, overall.

RESULTS / MODEL EVALUATION:

1

Metrics: CrossValidation Score:

A metric that measures the average of the accuracy of each fold for training dataset.

2

Conclusion

Logistic Regression: 0.9747435897435898
KNN: 0.959679487179487 1
Decision Tree: 0.9221153846153847

Overall, all 3 methods give >0.90 values.
Logistic Regression with the highest values, following by KNN, and lastly, Decision Tree.

DICUSSION:

1

Metrics:

Accuracy.

F-1 Score.

Cross Validation Score.

2

Final Conclusion:

All three methods are able to give a good classification model for this dataset.
Logistic Regression is the best out of them.

THANK YOU

Classification Problems with Logistic Regression, KNN, Decision Tree

THE END