Name - Arkadipta Mojumder
Registration Number - 22MCA0201
Digital Assessment 2

13-07-2023-penguin-classification

July 26, 2023

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score,classification_report
from sklearn.tree import DecisionTreeClassifier
```

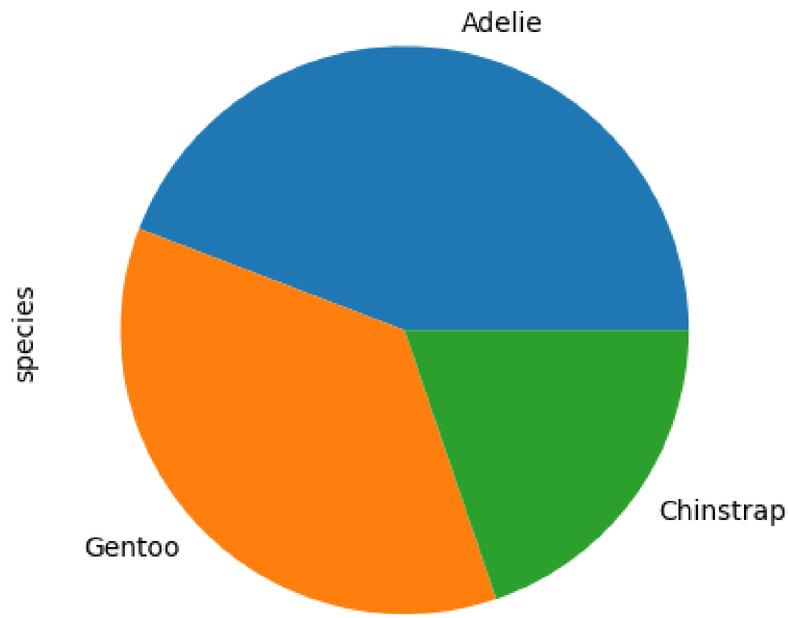
[ ]:
```
data = pd.read_csv('penguins_size.csv')
```

[ ]:
```
data.head()
```

[ ]:
```
  species     island  culmen_length_mm  culmen_depth_mm  flipper_length_mm  \
0  Adelie  Torgersen              39.1             18.7              181.0
1  Adelie  Torgersen              39.5             17.4              186.0
2  Adelie  Torgersen              40.3             18.0              195.0
3  Adelie  Torgersen               NaN              NaN                NaN
4  Adelie  Torgersen              36.7             19.3              193.0

   body_mass_g     sex
0       3750.0    MALE
1       3800.0  FEMALE
2       3250.0  FEMALE
3          NaN     NaN
4       3450.0  FEMALE
```

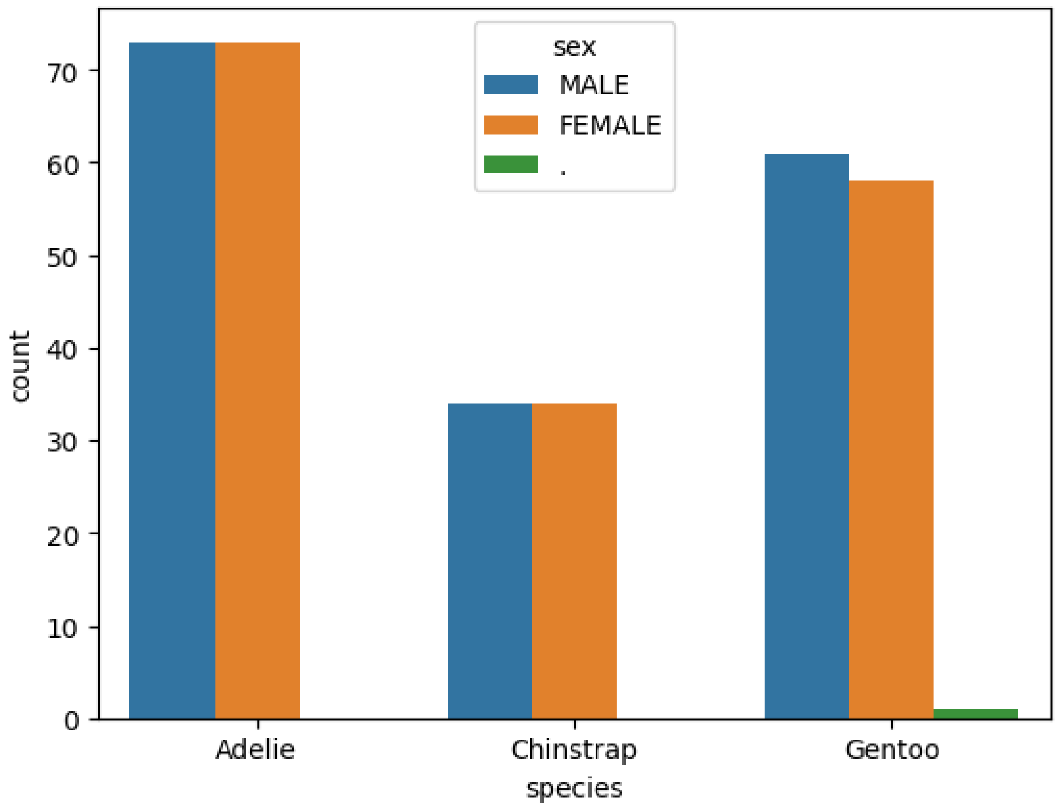
[ ]:
```
data.species.value_counts().plot(kind='pie')
```

[ ]:
```
<Axes: ylabel='species'>
```



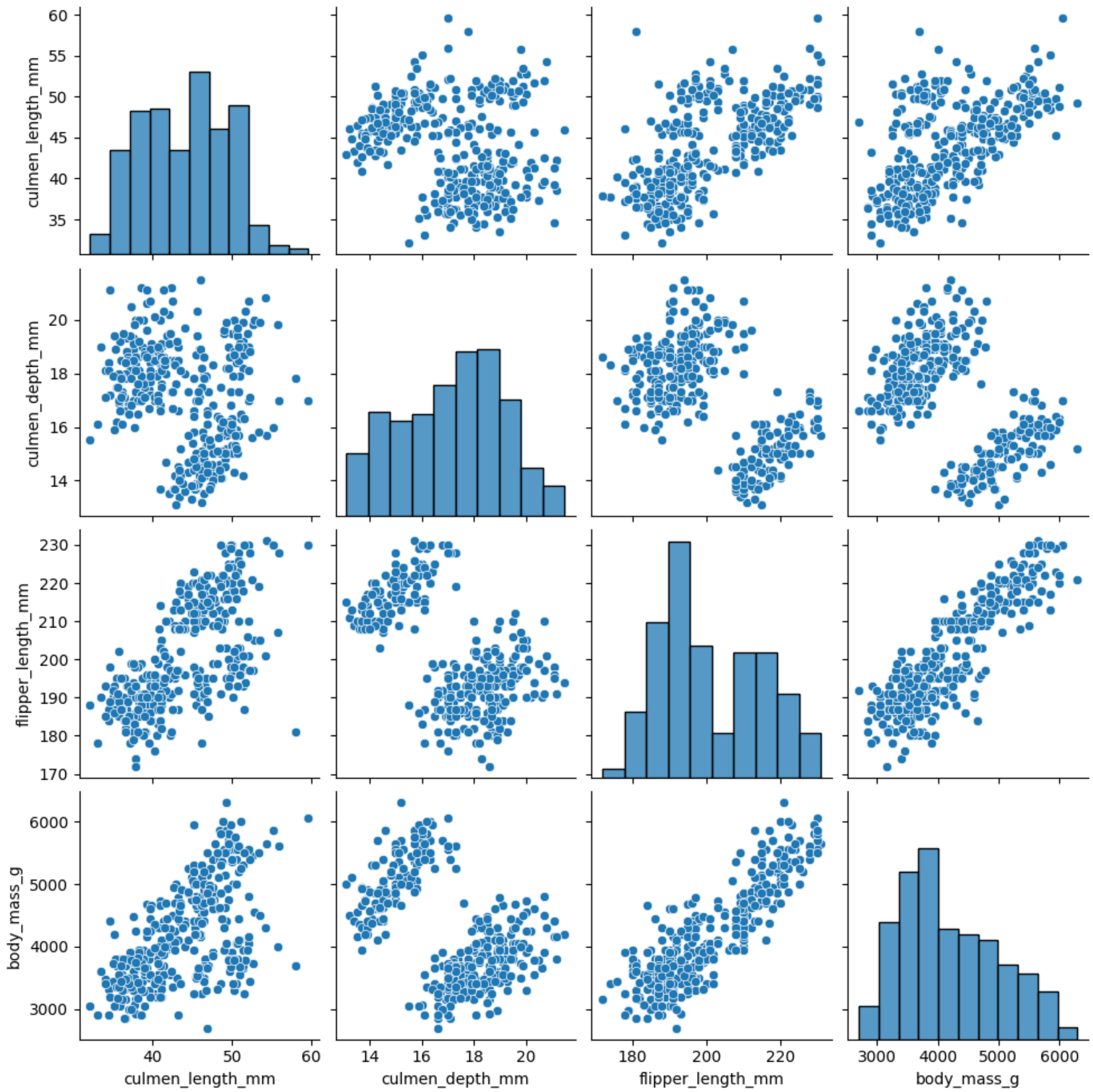
[ ]:
```
import seaborn as sns
sns.countplot(data=data,x='species',hue='sex')
```

[ ]: <Axes: xlabel='species', ylabel='count'>



[ ]:
```
sns.pairplot(data)
```

[ ]: <seaborn.axisgrid.PairGrid at 0x7a0e4c09c490>



[ ]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   species            344 non-null    object
 1   island             344 non-null    object
 2   culmen_length_mm   342 non-null    float64
 3   culmen_depth_mm    342 non-null    float64
 4   flipper_length_mm  342 non-null    float64
 5   body_mass_g        342 non-null    float64
```

```
 6   sex                 334 non-null    object
dtypes: float64(4), object(3)
memory usage: 18.9+ KB
```

[ ]:
```
data.describe()
```

[ ]:

	culmen_length_mm	culmen_depth_mm	flipper_length_mm	body_mass_g
count	342.000000	342.000000	342.000000	342.000000
mean	43.921930	17.151170	200.915205	4201.754386
std	5.459584	1.974793	14.061714	801.954536
min	32.100000	13.100000	172.000000	2700.000000
25%	39.225000	15.600000	190.000000	3550.000000
50%	44.450000	17.300000	197.000000	4050.000000
75%	48.500000	18.700000	213.000000	4750.000000
max	59.600000	21.500000	231.000000	6300.000000

[ ]:
```
data.isnull().sum()
```

[ ]:
```
species               0
island                0
culmen_length_mm      2
culmen_depth_mm       2
flipper_length_mm     2
body_mass_g           2
sex                  10
dtype: int64
```

[ ]:
```
data.dropna(subset=['sex'],inplace=True)
data.isnull().sum()
```

[ ]:
```
species              0
island               0
culmen_length_mm     0
culmen_depth_mm      0
flipper_length_mm    0
body_mass_g          0
sex                  0
dtype: int64
```

[ ]:
```
data['species'].value_counts()
```

[ ]:
```
Adelie       146
Gentoo       120
Chinstrap     68
Name: species, dtype: int64
```

[ ]:
```
data['sex'].value_counts()
```

[ ]:
```
MALE      168
FEMALE    165
.           1
Name: sex, dtype: int64
```

[ ]:
```
data=data[data.sex!='.']
```

[ ]:
```
data['sex'].value_counts()
```

[ ]:
```
MALE      168
FEMALE    165
Name: sex, dtype: int64
```

[ ]:
```
x=pd.get_dummies(data.drop('species',axis=1),columns=['island','sex'])
```

[ ]:
```
x.head()
```

[ ]:
```
   culmen_length_mm  culmen_depth_mm  flipper_length_mm  body_mass_g  \
0              39.1             18.7              181.0       3750.0
1              39.5             17.4              186.0       3800.0
2              40.3             18.0              195.0       3250.0
4              36.7             19.3              193.0       3450.0
5              39.3             20.6              190.0       3650.0

   island_Biscoe  island_Dream  island_Torgersen  sex_FEMALE  sex_MALE
0              0             0                 1           0         1
1              0             0                 1           1         0
2              0             0                 1           1         0
4              0             0                 1           1         0
5              0             0                 1           0         1
```

[ ]:
```
y=data['species']
```

[ ]:
```
from sklearn.preprocessing import LabelEncoder
```

[ ]:
```
le=LabelEncoder()
y=le.fit_transform(y)
y
```

[ ]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 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, 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,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2])
```

[ ]:
```python
from sklearn.model_selection import train_test_split
```

[ ]:
```python
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
```

[ ]:
```python
result=pd.DataFrame()
Models=[]
Accuracy=[]
```

[ ]:
```python
lR=LogisticRegression()
```

[ ]:
```python
knn=KNeighborsClassifier()
k_pred=knn.fit(x_train,y_train).predict(x_test)
print("Accuracy={}".format(accuracy_score(y_test,k_pred)))
```

```
Accuracy=0.8507462686567164
```

[ ]:
```python
kvl=range(1,10)
bestk=0
acc=0
b_acc=0
for k in kvl:
    knn=KNeighborsClassifier(n_neighbors=k)
    k_pred=knn.fit(x_train,y_train).predict(x_test)
    acc=accuracy_score(y_test,k_pred)
    if acc>b_acc:
        b_acc=acc
        bestk=k
print("Max accuracy:", b_acc)
print("Best k:", bestk)
```

```
Max accuracy: 0.8656716417910447
Best k: 1
```

[ ]:
```python
knn = KNeighborsClassifier(n_neighbors=bestk)
knn_pred = knn.fit(x_train, y_train).predict(x_test)
Models.append('K-Nearest Neighbor')
Accuracy.append(accuracy_score(y_test, knn_pred))
```

```
print("Accuracy score using K-Nearest Neighbor is: {}%".
 ↪format(accuracy_score(y_test, knn_pred)*100))
print(classification_report(y_test, knn_pred))
```

```
Accuracy score using K-Nearest Neighbor is: 86.56716417910447%
              precision    recall  f1-score   support

           0       0.94      0.81      0.87        37
           1       0.64      0.90      0.75        10
           2       0.90      0.95      0.93        20

    accuracy                           0.87        67
   macro avg       0.83      0.89      0.85        67
weighted avg       0.88      0.87      0.87        67
```

[ ]:
```
dst = DecisionTreeClassifier()
dst_pred = dst.fit(x_train, y_train).predict(x_test)
Models.append('Decision Trees')
Accuracy.append(accuracy_score(y_test, dst_pred))
print("Accuracy score using Decision Trees is: {}%".
 ↪format(accuracy_score(y_test, dst_pred)*100))
print(classification_report(y_test, dst_pred))
```

```
Accuracy score using Decision Trees is: 95.52238805970148%
              precision    recall  f1-score   support

           0       0.97      0.95      0.96        37
           1       0.91      1.00      0.95        10
           2       0.95      0.95      0.95        20

    accuracy                           0.96        67
   macro avg       0.94      0.97      0.95        67
weighted avg       0.96      0.96      0.96        67
```

[ ]:
```
result['models']=Models
result['accuracy']=Accuracy
```

[ ]:
```
result
```

[ ]:
```
               models  accuracy
0  K-Nearest Neighbor  0.865672
1      Decision Trees  0.955224
```

breast cancer-diagnosis-using-ml techniques

July 26, 2023

[ ]:
```python
import numpy as np
import pandas as pd
from pandas import Series, DataFrame
import seaborn as sns
import matplotlib.pyplot as plt
palette = 'magma'

%matplotlib inline
```

[ ]:
```python
data = pd.read_csv("data.csv")
data.head()
```

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

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0          0.11840           0.27760          0.3001              0.14710
1          0.08474           0.07864          0.0869              0.07017
2          0.10960           0.15990          0.1974              0.12790
3          0.14250           0.28390          0.2414              0.10520
4          0.10030           0.13280          0.1980              0.10430

   …  texture_worst  perimeter_worst  area_worst  smoothness_worst  \
0  …          17.33           184.60      2019.0            0.1622
1  …          23.41           158.80      1956.0            0.1238
2  …          25.53           152.50      1709.0            0.1444
3  …          26.50            98.87       567.7            0.2098
4  …          16.67           152.20      1575.0            0.1374

   compactness_worst  concavity_worst  concave points_worst  symmetry_worst  \
0             0.6656           0.7119                0.2654          0.4601
1             0.1866           0.2416                0.1860          0.2750
2             0.4245           0.4504                0.2430          0.3613
```

```
3              0.8663           0.6869                0.2575           0.6638
4              0.2050           0.4000                0.1625           0.2364

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

[5 rows x 33 columns]
```

[ ]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 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
 32   Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

Removing non-essential column

[ ]:
```python
data.drop("id", axis = 1, inplace = True)
```

[ ]:
```python
data.head()
```

[ ]:
```
  diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0         M        17.99         10.38          122.80     1001.0
1         M        20.57         17.77          132.90     1326.0
2         M        19.69         21.25          130.00     1203.0
3         M        11.42         20.38           77.58      386.1
4         M        20.29         14.34          135.10     1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0          0.11840           0.27760          0.3001              0.14710
1          0.08474           0.07864          0.0869              0.07017
2          0.10960           0.15990          0.1974              0.12790
3          0.14250           0.28390          0.2414              0.10520
4          0.10030           0.13280          0.1980              0.10430

   symmetry_mean  …  texture_worst  perimeter_worst  area_worst  \
0         0.2419  …          17.33           184.60      2019.0
1         0.1812  …          23.41           158.80      1956.0
2         0.2069  …          25.53           152.50      1709.0
3         0.2597  …          26.50            98.87       567.7
4         0.1809  …          16.67           152.20      1575.0

   smoothness_worst  compactness_worst  concavity_worst  concave points_worst  \
0            0.1622             0.6656           0.7119                0.2654
1            0.1238             0.1866           0.2416                0.1860
2            0.1444             0.4245           0.4504                0.2430
3            0.2098             0.8663           0.6869                0.2575
4            0.1374             0.2050           0.4000                0.1625

   symmetry_worst  fractal_dimension_worst  Unnamed: 32
0          0.4601                  0.11890          NaN
1          0.2750                  0.08902          NaN
2          0.3613                  0.08758          NaN
3          0.6638                  0.17300          NaN
4          0.2364                  0.07678          NaN
```

[5 rows x 32 columns]

```
[ ]: data = data[['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
            'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
            'concave points_mean']]
```

Converting Diagnosis to Dummy Variables

```
[ ]: malignant = pd.get_dummies(data['diagnosis'], drop_first=True)
     malignant
```

```
[ ]:      M
     0    1
     1    1
     2    1
     3    1
     4    1
     ..  ..
     564  1
     565  1
     566  1
     567  1
     568  0

     [569 rows x 1 columns]
```

```
[ ]: data = pd.concat([data, malignant], axis=1)
```

```
[ ]: data.drop('diagnosis', axis=1, inplace=True)
     data.head()
```

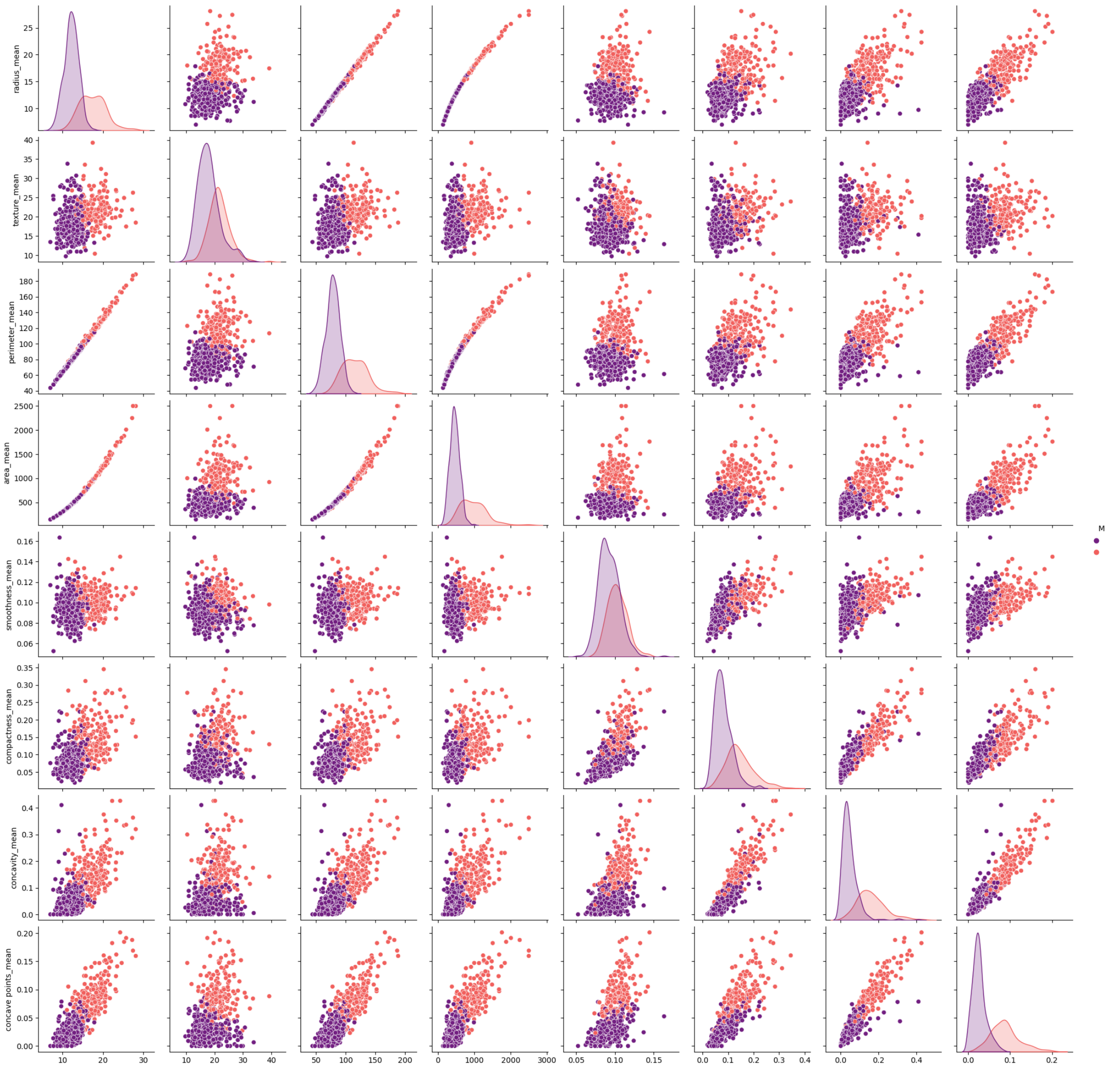
```
[ ]:    radius_mean  texture_mean  perimeter_mean  area_mean  smoothness_mean  \
     0        17.99         10.38          122.80     1001.0          0.11840
     1        20.57         17.77          132.90     1326.0          0.08474
     2        19.69         21.25          130.00     1203.0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

        compactness_mean  concavity_mean  concave points_mean  M
     0           0.27760          0.3001              0.14710  1
     1           0.07864          0.0869              0.07017  1
     2           0.15990          0.1974              0.12790  1
     3           0.28390          0.2414              0.10520  1
     4           0.13280          0.1980              0.10430  1
```

Exploratory Data Analysis

[ ]:
```
plt.figure(figsize=(10,6))
sns.pairplot(data, hue='M', palette=palette)
```

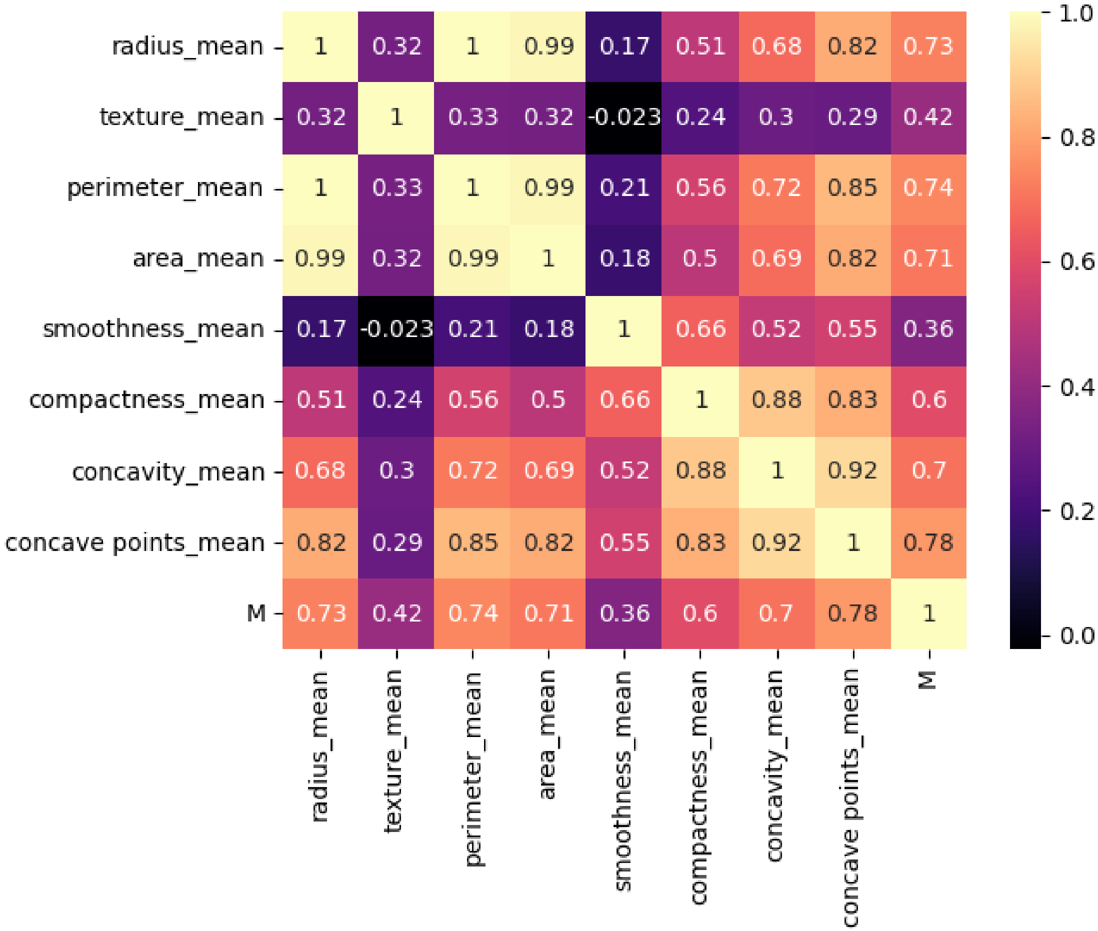
[ ]: <seaborn.axisgrid.PairGrid at 0x7ab78c10c820>

<Figure size 1000x600 with 0 Axes>



[ ]:
```
sns.heatmap(data.corr(), annot=True, cmap=palette)
```

[ ]: <Axes: >



Standarize the variables

[ ]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(data.drop('M', axis=1))
```

[ ]: StandardScaler()

[ ]:
```
scaled_features = scaler.transform(data.drop('M', axis=1))
```

[ ]:
```
df_feat = pd.DataFrame(scaled_features, columns=data.columns[:-1])
df_feat.head()
```

[ ]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	\
0	1.097064	-2.073335	1.269934	0.984375	1.568466	
1	1.829821	-0.353632	1.685955	1.908708	-0.826962	
2	1.579888	0.456187	1.566503	1.558884	0.942210	
3	-0.768909	0.253732	-0.592687	-0.764464	3.283553	

```
4    1.750297    -1.151816        1.776573   1.826229          0.280372

   compactness_mean  concavity_mean  concave points_mean
0          3.283515        2.652874             2.532475
1         -0.487072       -0.023846             0.548144
2          1.052926        1.363478             2.037231
3          3.402909        1.915897             1.451707
4          0.539340        1.371011             1.428493
```

Split the Data into Training and Testing Set

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

[ ]:
```
X = df_feat
y = data['M']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
```

Prediction and Evaluation

[ ]:
```
from sklearn.neighbors import KNeighborsClassifier
```

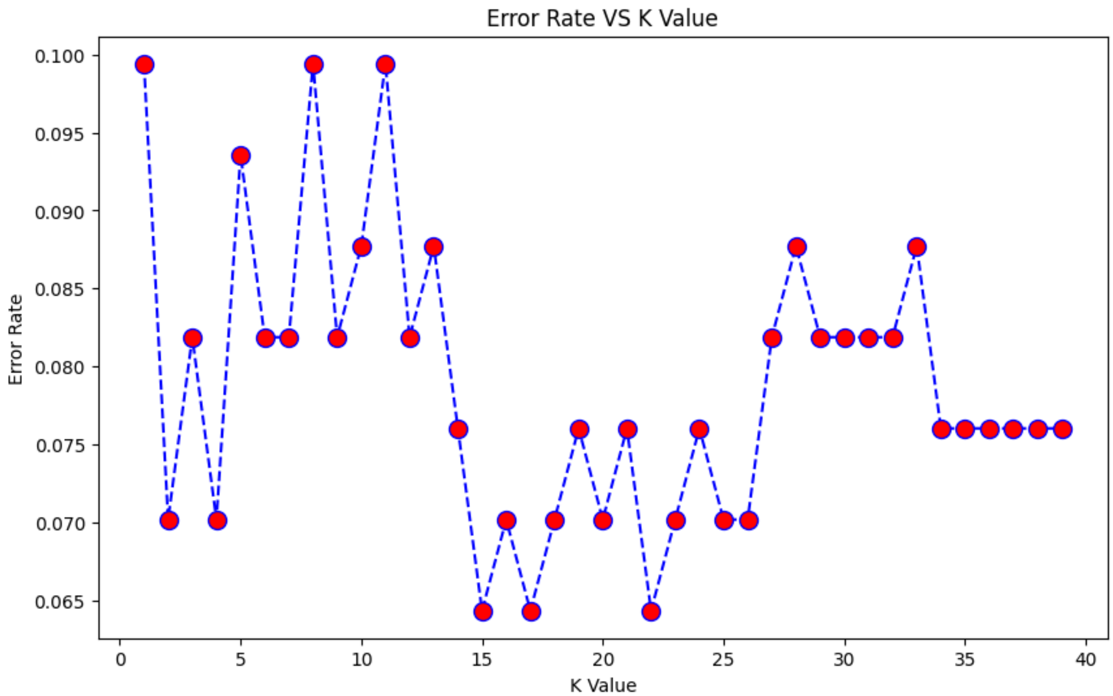
Create a loop to train KNN models with different k values and track the error rate for each model in a list.

[ ]:
```
error_rate = []
for i in range(1,40):

    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))
```

Generate a plot based on the data in the list

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

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



0.0.1 Set k to 17, then train the model and make predictions.

```
[ ]: knn = KNeighborsClassifier(n_neighbors=17)
     knn.fit(X_train, y_train)
     pred = knn.predict(X_test)
```

Evaluate the model using a classification report and a confusion matrix

```
[ ]: from sklearn.metrics import classification_report, confusion_matrix
```

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

```
[[100   5]
 [  6  60]]
```

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

```
              precision    recall  f1-score   support

           0       0.94      0.95      0.95       105
           1       0.92      0.91      0.92        66

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

```
weighted avg       0.94      0.94      0.94       171
```