```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

#simple imputer
array = np.array([[1,2],[np.nan,3],[7,6]])
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan,strategy="mean")
imputed_array = imputer.fit_transform(array)
print(imputed_array)
```

```
[[1. 2.]
 [4. 3.]
 [7. 6.]]
```

```
#minmaxscaler
from sklearn.preprocessing import MinMaxScaler
array_2 = np.array([[1,2],[6,7],[3,4]])
scaler = MinMaxScaler()
scaled_array = scaler.fit_transform(array_2)
print(scaled_array)
```

```
[[0.  0. ]
 [1.  1. ]
 [0.4 0.4]]
```

```
data = pd.read_csv("iris.csv")
df = pd.DataFrame(data)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   sepallength  150 non-null    float64
 1   sepalwidth   150 non-null    float64
 2   petallength  150 non-null    float64
 3   petalwidth   150 non-null    float64
 4   class        150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

```
df_nonull = df[df.isnull()==False]
df_nonull
```

	sepallength	sepalwidth	petallength	petalwidth	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa

```
4             5.0          3.6           1.4          0.2     Iris-setosa
..            ...          ...           ...          ...             ...
145           6.7          3.0           5.2          2.3  Iris-virginica
146           6.3          2.5           5.0          1.9  Iris-virginica
147           6.5          3.0           5.2          2.0  Iris-virginica
148           6.2          3.4           5.4          2.3  Iris-virginica
149           5.9          3.0           5.1          1.8  Iris-virginica

[150 rows x 5 columns]
```

```
df_nonull.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   sepallength  150 non-null    float64
 1   sepalwidth   150 non-null    float64
 2   petallength  150 non-null    float64
 3   petalwidth   150 non-null    float64
 4   class        150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

```
df.head(5)
```

```
   sepallength  sepalwidth  petallength  petalwidth        class
0          5.1         3.5          1.4         0.2  Iris-setosa
1          4.9         3.0          1.4         0.2  Iris-setosa
2          4.7         3.2          1.3         0.2  Iris-setosa
3          4.6         3.1          1.5         0.2  Iris-setosa
4          5.0         3.6          1.4         0.2  Iris-setosa
```

```
df.tail(5)
```

```
     sepallength  sepalwidth  petallength  petalwidth           class
145          6.7         3.0          5.2         2.3  Iris-virginica
146          6.3         2.5          5.0         1.9  Iris-virginica
147          6.5         3.0          5.2         2.0  Iris-virginica
148          6.2         3.4          5.4         2.3  Iris-virginica
149          5.9         3.0          5.1         1.8  Iris-virginica
```

```
df["petallength"].dtype
```

```
dtype('float64')
```

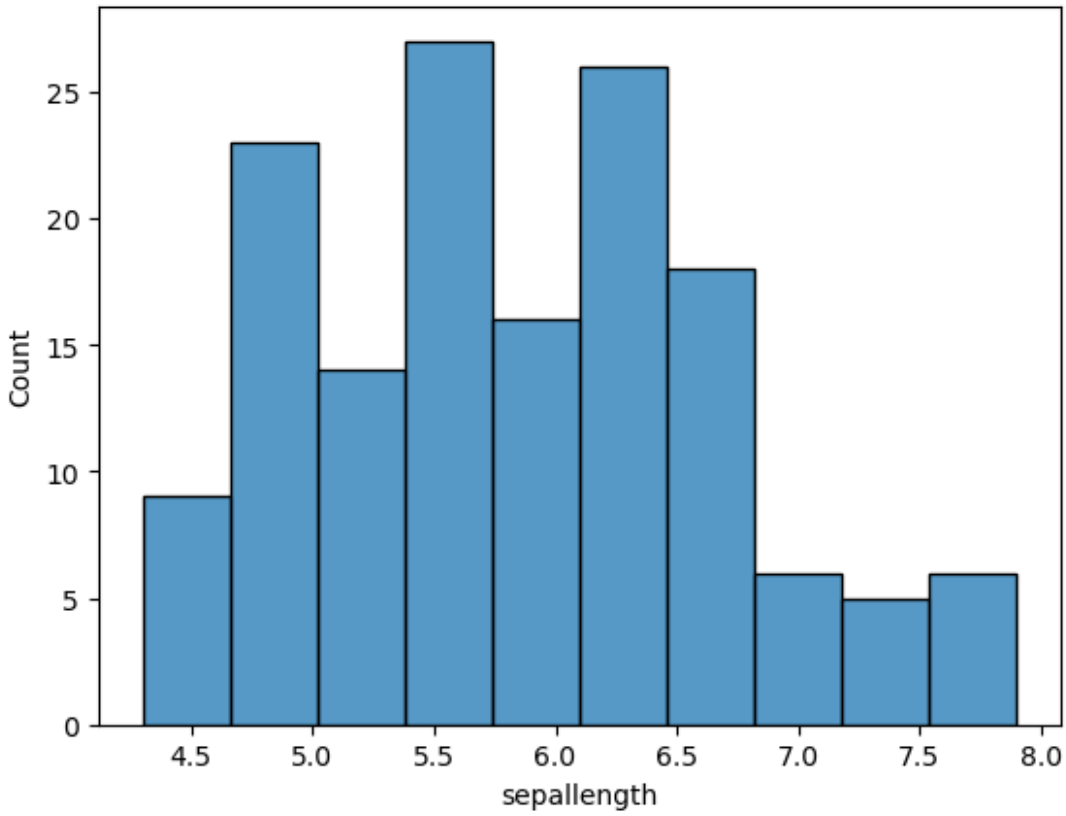
```
df["sepallength"].mean()
```

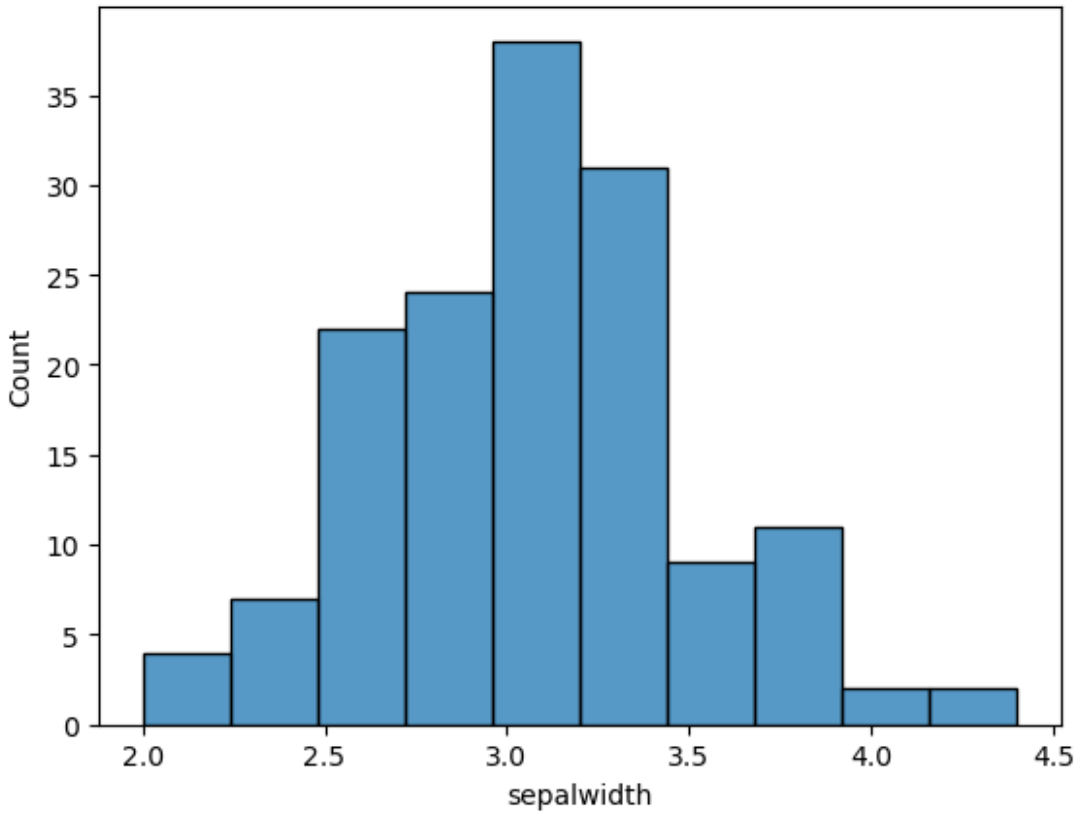
```
5.843333333333334
```

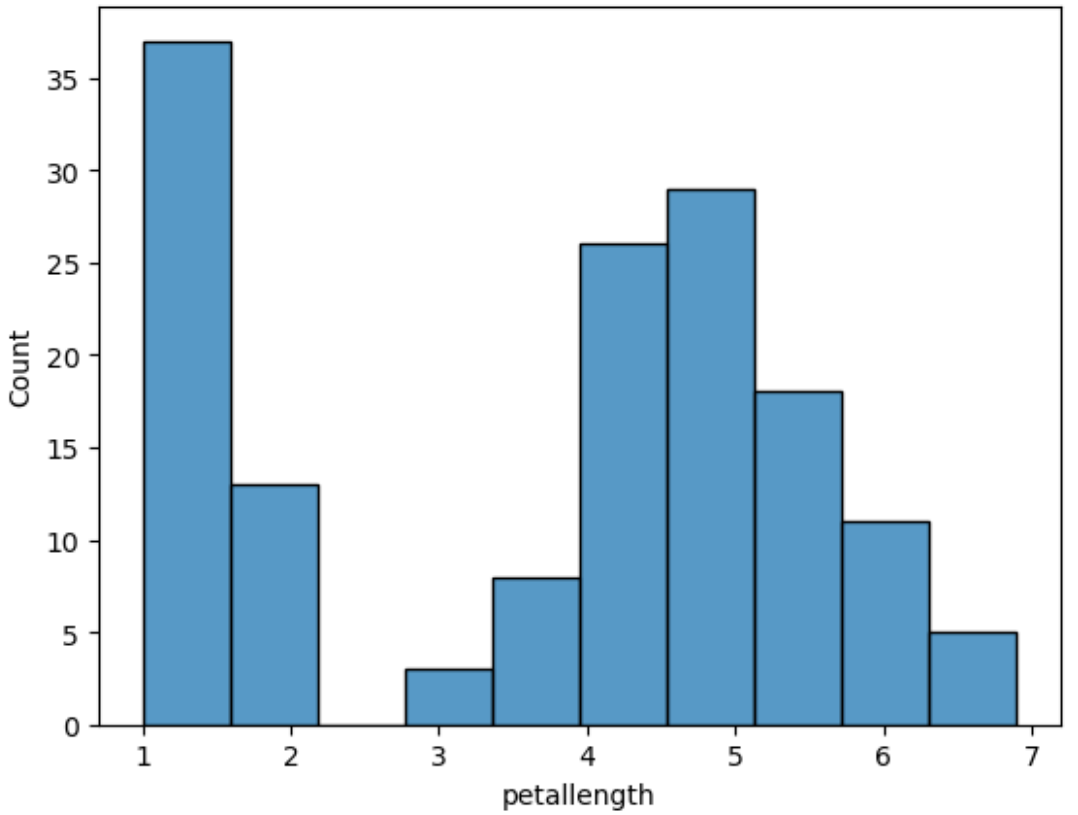
```
df["petalwidth"].std()
```

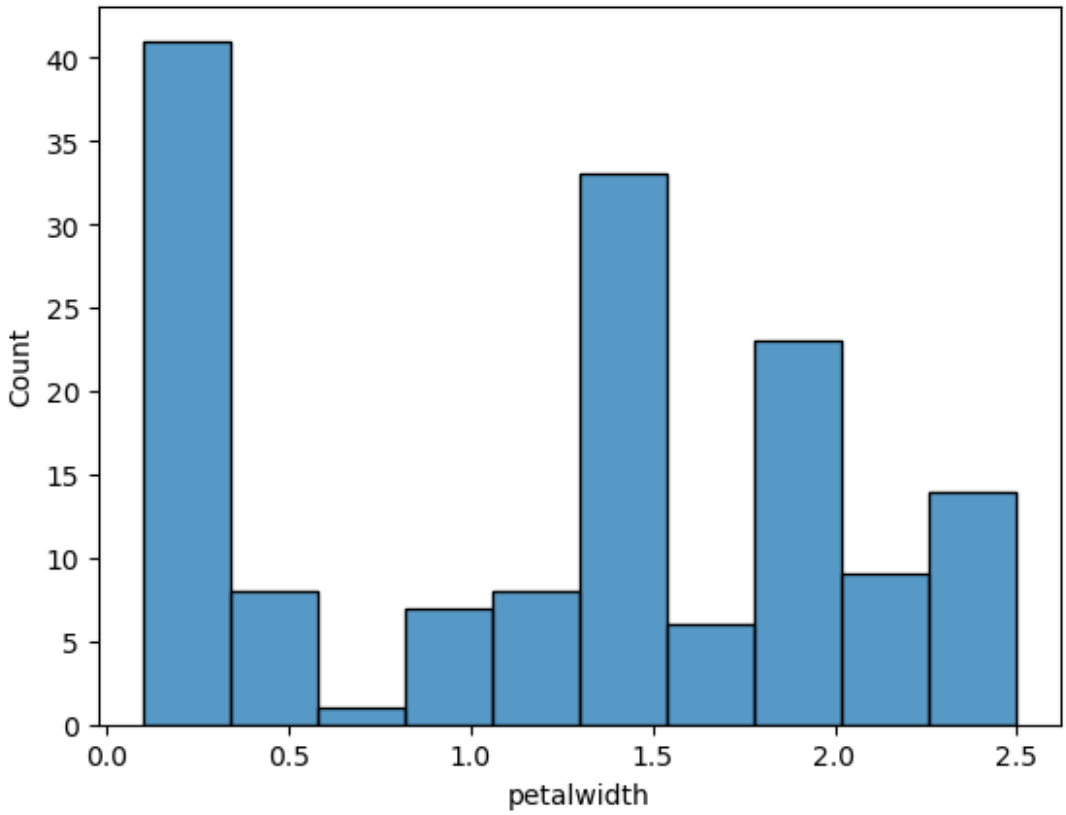
```
0.7631607417008414
```

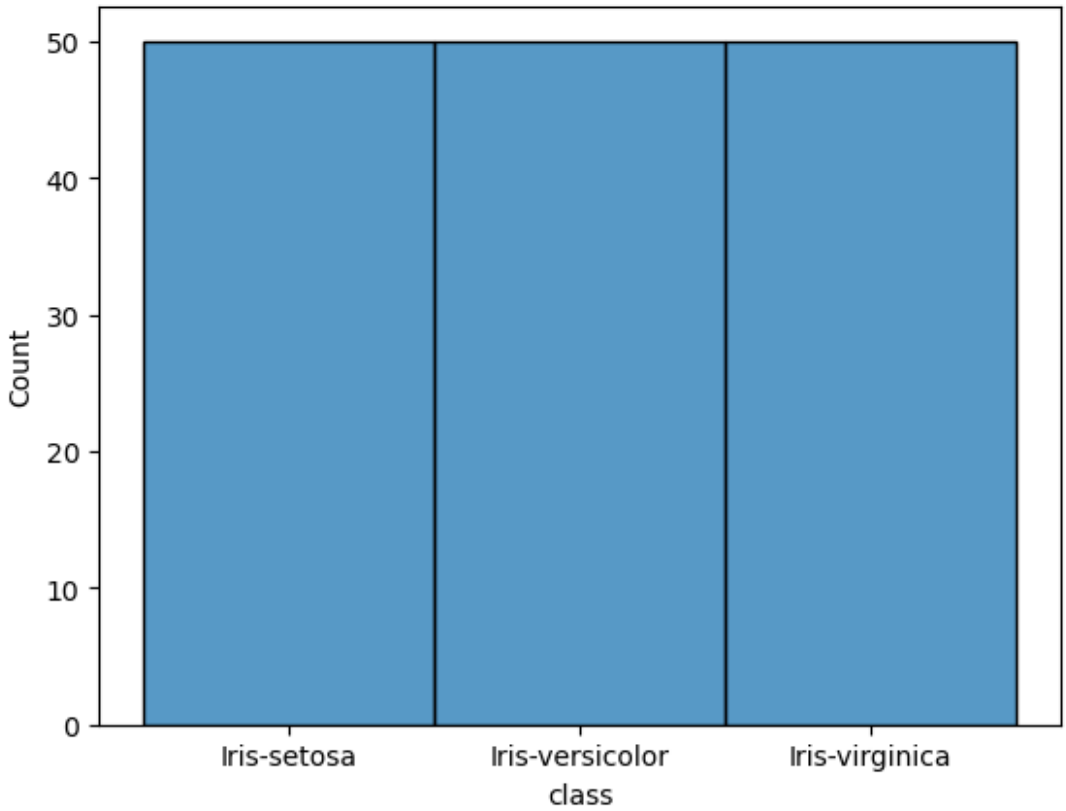
```
import seaborn as sns

for i in list(df.columns):
    sns.histplot(df[i],bins=10)
    plt.show()
```




Count
0
5
10
15
20
25
30
35
2.0
2.5
3.0
3.5
4.0
4.5
sepalwidth

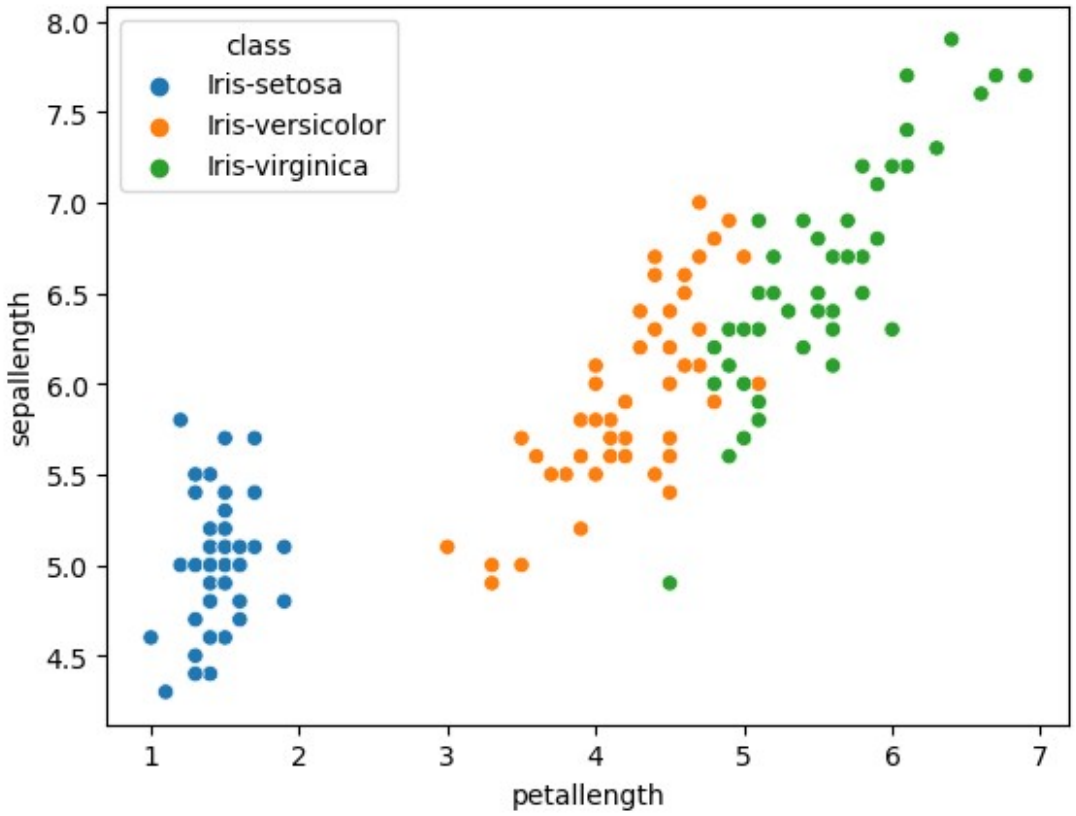

Count
0
5
10
15
20
25
30
35
1
2
3
4
5
6
7
petallength


Count
40
35
30
25
20
15
10
5
0
0.0
0.5
1.0
1.5
2.0
2.5
petalwidth


Count
50
40
30
20
10
0
Iris-setosa
Iris-versicolor
Iris-virginica
class

```
sns.scatterplot(data=df,x=df["petallength"],y=df["sepallength"],hue=df
["class"])
```

```
<Axes: xlabel='petallength', ylabel='sepallength'>
```




```
wine = pd.read_csv("WineQT.csv")
df_wine  = pd.DataFrame(wine)
df_wine.head()
```

```
   fixed_acidity  volatile_acidity  citric_acid  residual_sugar
chlorides  \
0            7.4              0.70         0.00             1.9
0.076
1            7.8              0.88         0.00             2.6
0.098
2            7.8              0.76         0.04             2.3
0.092
3           11.2              0.28         0.56             1.9
0.075
4            7.4              0.70         0.00             1.9
0.076

   free_sulfur_dioxide  total_sulfur_dioxide  density    pH  sulphates
\
```

```
0                 11.0                    34.0   0.9978  3.51       0.56

1                 25.0                    67.0   0.9968  3.20       0.68

2                 15.0                    54.0   0.9970  3.26       0.65

3                 17.0                    60.0   0.9980  3.16       0.58

4                 11.0                    34.0   0.9978  3.51       0.56
```

```
   alcohol  quality  Id
0      9.4        5   0
1      9.8        5   1
2      9.8        5   2
3      9.8        6   3
4      9.4        5   4
```

```
df_wine.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1143 entries, 0 to 1142
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed_acidity         1143 non-null   float64
 1   volatile_acidity      1143 non-null   float64
 2   citric_acid           1143 non-null   float64
 3   residual_sugar        1143 non-null   float64
 4   chlorides             1143 non-null   float64
 5   free_sulfur_dioxide   1143 non-null   float64
 6   total_sulfur_dioxide  1143 non-null   float64
 7   density               1143 non-null   float64
 8   pH                    1143 non-null   float64
 9   sulphates             1143 non-null   float64
 10  alcohol               1143 non-null   float64
 11  quality               1143 non-null   int64  
 12  Id                    1143 non-null   int64  
dtypes: float64(11), int64(2)
memory usage: 116.2 KB
```

```
df_wine.shape
```

```
(1143, 13)
```

```
df_wine.isnull().sum()
```

```
fixed_acidity           0
volatile_acidity        0
citric_acid             0
residual_sugar          0
```

```
chlorides               0
free_sulfur_dioxide     0
total_sulfur_dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
Id                      0
dtype: int64
```

```
for j in list(df_wine.columns):
    print("1st quartile:",np.percentile(df_wine[j],25))
    print("2nd quartile:",np.percentile(df_wine[j],75))
    print()
```

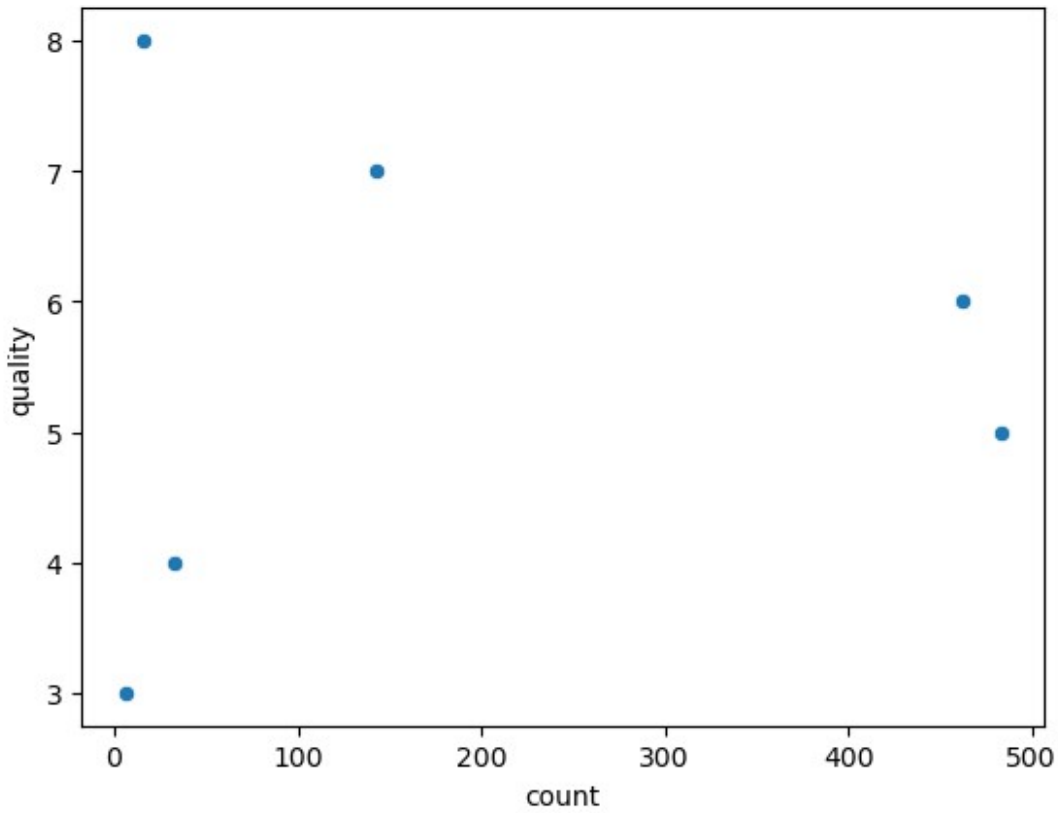
```
1st quartile: 7.1
2nd quartile: 9.1

1st quartile: 0.3925
2nd quartile: 0.64

1st quartile: 0.09
2nd quartile: 0.42

1st quartile: 1.9
2nd quartile: 2.6

1st quartile: 0.07
2nd quartile: 0.09

1st quartile: 7.0
2nd quartile: 21.0

1st quartile: 21.0
2nd quartile: 61.0

1st quartile: 0.99557
2nd quartile: 0.997845

1st quartile: 3.205
2nd quartile: 3.4

1st quartile: 0.55
2nd quartile: 0.73

1st quartile: 9.5
2nd quartile: 11.1

1st quartile: 5.0
2nd quartile: 6.0
```

```
1st quartile: 411.0
2nd quartile: 1209.5
```

```
df_plot = df_wine["quality"].value_counts().sort_index().reset_index()
df_plot.columns = ["quality","count"]
df_plot
```

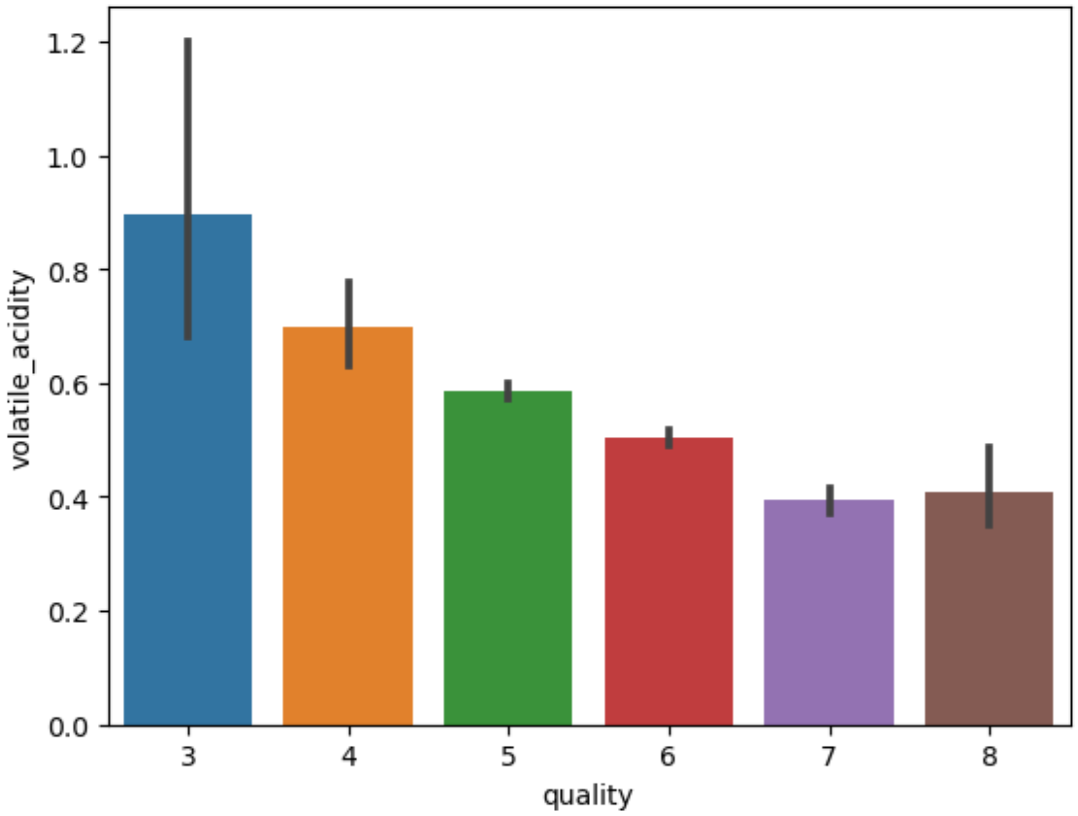
	quality	count
0	3	6
1	4	33
2	5	483
3	6	462
4	7	143
5	8	16

```
sns.scatterplot(data =
df_plot,x=df_plot["count"],y=df_plot["quality"])
plt.show()
```




```
sns.barplot(data=df_wine,x=df_wine["quality"],y=df_wine["volatile_acid
ity"])
```

```
<Axes: xlabel='quality', ylabel='volatile_acidity'>
```


1.2
1.0
0.8
0.6
0.4
0.2
0.0
volatile_acidity
3
4
5
6
7
8
quality

In [41]:
```
import numpy as np
```

Question 1

Robust Scaler is a feature scaling technique used in machine learning to standardize numerical input variables, especially when dealing with datasets that contain outliers. It uses the median and IQR to scale the data, making it less sensitive to extreme values compared to other scaling methods like StandardScaler.

Question 2

In [42]:
```
X = [[ 1., -2.,  2.],

   [ -2.,  1.,  3.],

    [ 4.,  1., -2.]]
X = np.array(X)
```

In [43]:
```
from sklearn.preprocessing import RobustScaler
r_scaler = RobustScaler()
scaled_X = r_scaler.fit_transform(X)
scaled_X
```

Out[43]:
```
array([[ 0. , -2. ,  0. ],
       [-1. ,  0. ,  0.4],
       [ 1. ,  0. , -1.6]])
```

Question 3

In [44]:
```
X = [[ 1., -1.,  2.],

  [ 2.,  0.,  0.],

  [ 0.,  1., -1.]]
X = np.array(X)
```

In [45]:
```
from sklearn.preprocessing import StandardScaler
s_scaler = StandardScaler()
scaled_X = s_scaler.fit_transform(X)
scaled_X
```

Out[45]:
```
array([[ 0.        , -1.22474487,  1.33630621],
       [ 1.22474487,  0.        , -0.26726124],
       [-1.22474487,  1.22474487, -1.06904497]])
```

Question 4

In [46]:
```
from sklearn.preprocessing import MaxAbsScaler
m_scaler = MaxAbsScaler()
scaled_X = m_scaler.fit_transform(X)
scaled_X
```

Out[46]:
```
array([[ 0.5, -1. ,  1. ],
       [ 1. ,  0. ,  0. ],
       [ 0. ,  1. , -0.5]])
```

Question 5

In [47]:
```
from sklearn.preprocessing import OneHotEncoder
arr = np.array([['Male', 1], ['Female', 3], ['Female', 2]])
enc = OneHotEncoder()
encoded_arr = enc.fit_transform(arr)
encoded_arr.toarray()
```

Out[47]:
```
array([[0., 1., 1., 0., 0.],
       [1., 0., 0., 0., 1.],
       [1., 0., 0., 1., 0.]])
```

Question 6

In [48]:
```
from sklearn.preprocessing import MultiLabelBinarizer

mlb = MultiLabelBinarizer()
encoded = mlb.fit_transform([{'sci-fi', 'thriller'}, {'comedy'}])
print(mlb.classes_)
print(encoded)
```

```
['comedy' 'sci-fi' 'thriller']
[[0 1 1]
 [1 0 0]]
```

Question 1

In [45]:
```
import matplotlib.pyplot as plt
import numpy as np

# Import datasets, classifiers and performance metrics
from sklearn import datasets, metrics, svm
from sklearn.model_selection import train_test_split
```

Loading the dataset

In [33]:
```
dataset = datasets.load_digits()
X = dataset['images']
y = dataset['target']
```

In [34]:
```
dataset
```

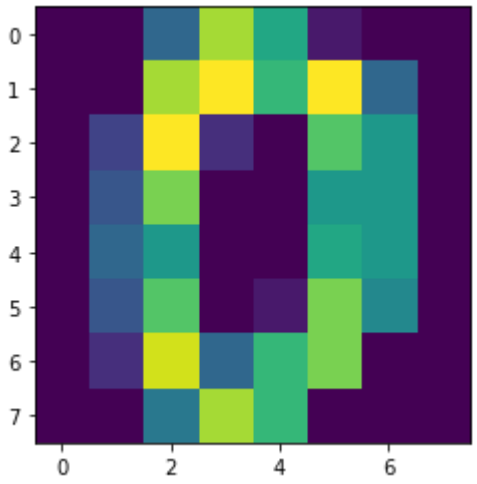
Out[34]:
```
{'data': array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
        [ 0.,  0.,  0., ..., 10.,  0.,  0.],
        [ 0.,  0.,  0., ..., 16.,  9.,  0.],
        ...,
        [ 0.,  0.,  1., ...,  6.,  0.,  0.],
        [ 0.,  0.,  2., ..., 12.,  0.,  0.],
        [ 0.,  0., 10., ..., 12.,  1.,  0.]]),
 'target': array([0, 1, 2, ..., 8, 9, 8]),
 'frame': None,
 'feature_names': ['pixel_0_0',
  'pixel_0_1',
  'pixel_0_2',
  'pixel_0_3',
  'pixel_0_4',
  'pixel_0_5',
  'pixel_0_6',
  'pixel_0_7',
  'pixel_1_0',
  'pixel_1_1',
```

In [32]:
```
X[0]
```

Out[32]:
```
array([[ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.],
       [ 0.,  0., 13., 15., 10., 15.,  5.,  0.],
       [ 0.,  3., 15.,  2.,  0., 11.,  8.,  0.],
       [ 0.,  4., 12.,  0.,  0.,  8.,  8.,  0.],
       [ 0.,  5.,  8.,  0.,  0.,  9.,  8.,  0.],
       [ 0.,  4., 11.,  0.,  1., 12.,  7.,  0.],
       [ 0.,  2., 14.,  5., 10., 12.,  0.,  0.],
       [ 0.,  0.,  6., 13., 10.,  0.,  0.,  0.]])
```

In [38]:
```
plt.imshow(X[0])
```

Out[38]:
```
<matplotlib.image.AxesImage at 0x74ce8a573880>
```



In [29]:
```
len(X),len(y)
```

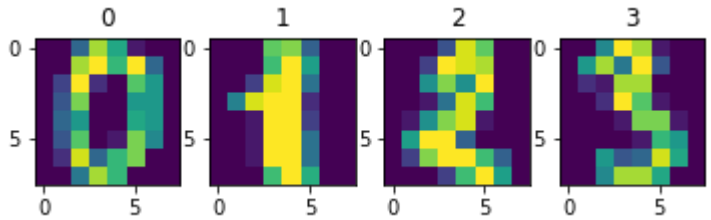
Out[29]:
```
(1797, 1797)
```

In [30]:
```
X.shape,y.shape
```

Out[30]:
```
((1797, 8, 8), (1797,))
```

Plotting and flattening of images

In [42]:
```
plt.title("First 4 images")
for i in range(1,5):
    plt.subplot(1,4,i)
    plt.title(y[i-1])
    plt.imshow(X[i-1])
```



In [47]:
```
flattened_X = []
for image in X:
    flattened_X.append(image.flatten())
flattened_X = np.array(flattened_X)
```

Out[47]:
```
(1797, 64)
```

Splitting and Training

In [48]:
```
X_train,X_test,y_train,y_test = train_test_split(flattened_X,y,test_s
```

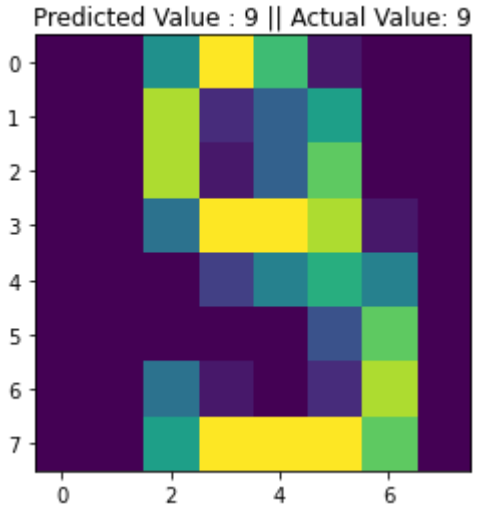
In [60]:
```
classifier = svm.SVC(gamma=0.001)
classifier.fit(X_train,y_train)
```

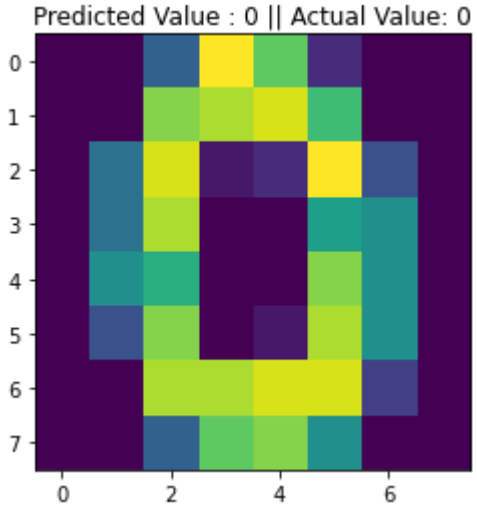
Out[60]:
```
SVC(gamma=0.001)
```

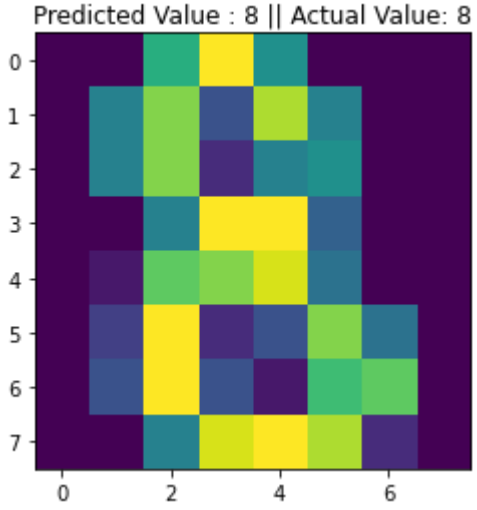
In [61]:
```
y_pred = classifier.predict(X_test)
```

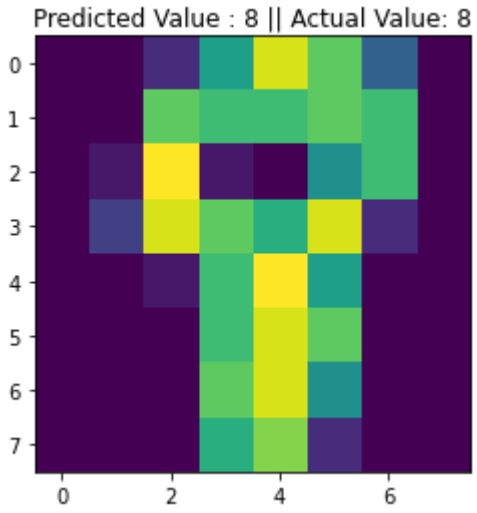
Evaluation

In [65]:
```
plt.title("First 4 images:")
for i in range(1,5):
    plt.title("Predicted Value : {} || Actual Value: {}".format(y_pre
    plt.imshow(X_test[i-1].reshape(8,8))
    plt.show()
```







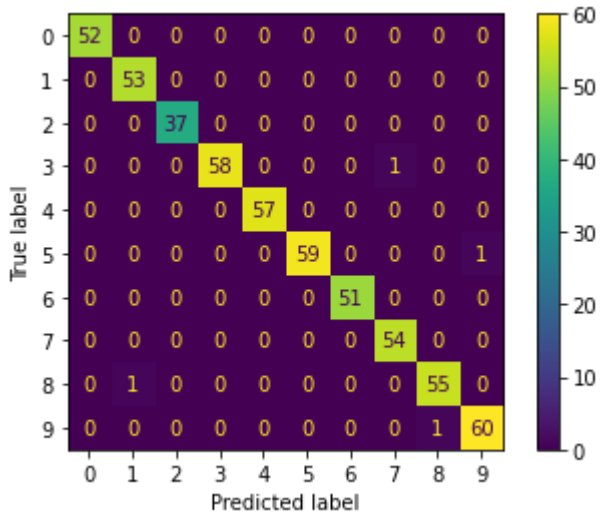


In [68]:
```
print(metrics.classification_report(y_test,y_pred))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	52
1	0.98	1.00	0.99	53
2	1.00	1.00	1.00	37
3	1.00	0.98	0.99	59
4	1.00	1.00	1.00	57
5	1.00	0.98	0.99	60
6	1.00	1.00	1.00	51
7	0.98	1.00	0.99	54
8	0.98	0.98	0.98	56
9	0.98	0.98	0.98	61
accuracy			0.99	540
macro avg	0.99	0.99	0.99	540
weighted avg	0.99	0.99	0.99	540

In [72]:
```
cm_plot = metrics.ConfusionMatrixDisplay(metrics.confusion_matrix(y_t
cm_plot.plot()

# this confusion matrix is a sparse matrix as it has a lot of zeroes.
```

Out[72]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0
x74ce8769fca0>
```



Question 2

In [77]:
```
from sklearn.feature_selection import SelectKBest,chi2,f_classif,f_re
X, y = datasets.load_digits(return_X_y=True)
```

In [78]:
```
X_new = SelectKBest(chi2, k=20).fit_transform(X, y)
X_new.shape
```

Out[78]:
```
(1797, 20)
```

In [80]:
```
X_new = SelectKBest(f_classif, k=20).fit_transform(X, y)
X_new.shape
```

```
/opt/anaconda3/lib/python3.9/site-packages/sklearn/feature_selectio
n/_univariate_selection.py:114: UserWarning: Features [ 0 32 39] ar
e constant.
  warnings.warn("Features %s are constant." % constant_features_id
x,
/opt/anaconda3/lib/python3.9/site-packages/sklearn/feature_selectio
n/_univariate_selection.py:116: RuntimeWarning: invalid value encou
ntered in true_divide
  f = msb / msw
```

Out[80]:
```
(1797, 20)
```

In [81]:
```
X_new = SelectKBest(f_regression, k=20).fit_transform(X, y)
X_new.shape
```

```
/opt/anaconda3/lib/python3.9/site-packages/sklearn/feature_selectio
n/_univariate_selection.py:302: RuntimeWarning: invalid value encou
ntered in true_divide
  corr /= X_norms
```

Out[81]:
```
(1797, 20)
```

Question 3

In [83]:
```
from sklearn.feature_selection import SelectPercentile, chi2 , f_clas
```

In [84]:
```
X_new = SelectPercentile(chi2, percentile=10).fit_transform(X, y)

X_new.shape
```

Out[84]:
```
(1797, 7)
```

In [85]:
```
X_new = SelectPercentile(f_classif, percentile=10).fit_transform(X, y
X_new.shape
```

```
/opt/anaconda3/lib/python3.9/site-packages/sklearn/feature_selectio
n/_univariate_selection.py:114: UserWarning: Features [ 0 32 39] ar
e constant.
  warnings.warn("Features %s are constant." % constant_features_id
x,
/opt/anaconda3/lib/python3.9/site-packages/sklearn/feature_selectio
n/_univariate_selection.py:116: RuntimeWarning: invalid value encou
ntered in true_divide
  f = msb / msw
```

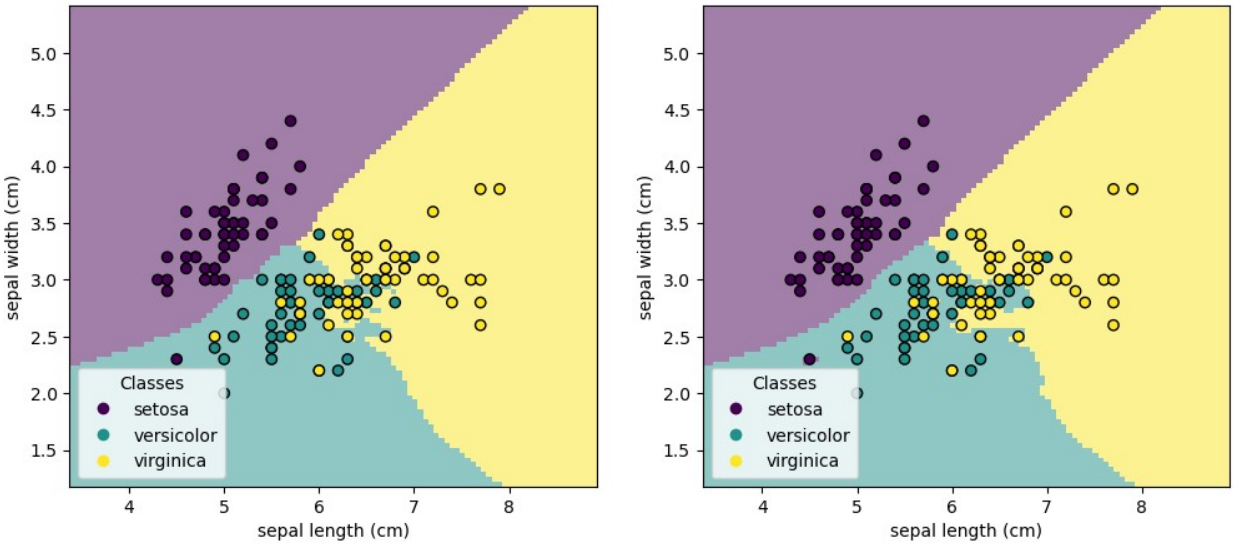
Out[85]:
```
(1797, 7)
```

In [86]:
```
X_new = SelectPercentile(f_regression, percentile=10).fit_transform(X

X_new.shape
```

```
/opt/anaconda3/lib/python3.9/site-packages/sklearn/feature_selectio
n/_univariate_selection.py:302: RuntimeWarning: invalid value encou
ntered in true_divide
  corr /= X_norms
```

Out[86]:
```
(1797, 7)
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets

data = datasets.load_iris()
X,y = datasets.load_iris(return_X_y=True)
from sklearn.model_selection import train_test_split
X = X[:,:2]

x_train,x_test,y_train,y_test =
train_test_split(X,y,test_size=0.3,random_state=42)

from sklearn.pipeline import Pipeline
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.inspection import DecisionBoundaryDisplay

clf = Pipeline(steps=[("scaler",StandardScaler()),
("knn",KNeighborsClassifier(n_neighbors=11))])

fig,axes = plt.subplots(nrows=1,ncols=2,figsize=(12,5))

for ax,weights in zip(axes,["uniform","distance"]):
    clf.set_params(knn__weights = weights).fit(x_train,y_train)

DecisionBoundaryDisplay.from_estimator(clf,x_test,response_method="pre
dict",plot_method="pcolormesh",xlabel=data.feature_names[0],ylabel=dat
a.feature_names[1],shading="auto",ax=ax,alpha=0.5)
    scatter = ax.scatter(X[:,0],X[:,1],c=y,edgecolors="k")
    ax.legend(
        scatter.legend_elements()[0], data.target_names,
        loc="lower left", title="Classes"
    )
```


sepal width (cm)
sepal length (cm)
Classes
setosa
versicolor
virginica

In [1]:
```
import numpy as np
from sklearn.datasets import fetch_california_housing
```

In [4]:
```
dataset = fetch_california_housing()
```

In [6]:
```
X = dataset.data
y = dataset.target
```

In [7]:
```
X = X[:2000]
y = y[:2000]
```

In [8]:
```
X.shape,y.shape
```

Out[8]:
```
((2000, 8), (2000,))
```

Select K best

In [9]:
```
from sklearn.feature_selection import SelectKBest, f_regression
X_k = SelectKBest(f_regression,k=3).fit_transform(X,y)
X_k.shape
```

Out[9]:
```
(2000, 3)
```

Select Percentile

In [10]:
```
from sklearn.feature_selection import SelectPercentile,f_regression
X_p = SelectPercentile(f_regression,percentile=30).fit_transform(X,y)
X_p.shape
```

Out[10]:
```
(2000, 3)
```

Generic Univariate Select

In [15]:
```
from sklearn.feature_selection import GenericUnivariateSelect

transformer = GenericUnivariateSelect(f_regression, mode='k_best', pa
X_g = transformer.fit_transform(X, y)
X_g.shape
```

Out[15]:
```
(2000, 3)
```

In [46]:
```
from sklearn.datasets import make_friedman1,fetch_california_housing
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
estimator = LinearRegression()
```

In [47]:
```
X, y = make_friedman1(n_samples=50, n_features=10, random_state=0)
```

RFE

In [48]:
```
X.shape,y.shape
```

Out[48]:
```
((50, 10), (50,))
```

In [49]:
```
selector = RFE(estimator, n_features_to_select=3, step=1)
selector = selector.fit(X,y)
```

In [50]:
```
selector.ranking_
```

Out[50]:
```
array([3, 2, 1, 1, 1, 4, 5, 8, 7, 6])
```

In [51]:
```
X = selector.transform(X)
X.shape
```

Out[51]:
```
(50, 3)
```

SelectFromModal

In [52]:
```
from sklearn.feature_selection import SelectFromModel
```

In [53]:
```
dataset = fetch_california_housing()
estimator = LinearRegression()
```

In [54]:
```
X = dataset.data
y = dataset.target
```

In [55]:
```
X.shape
```

Out[55]:
```
(20640, 8)
```

In [56]:
```
selector = SelectFromModel(estimator).fit(X, y)
print("Threshold: ",selector.threshold_)
```

```
Threshold:  0.25726693216557395
```

In [57]:
```
X = selector.transform(X)
X.shape
```

Out[57]:
```
(20640, 4)
```

SequentialFeatureSelection

In [58]:
```
from sklearn.feature_selection import SequentialFeatureSelector
```

In [59]:
```
sfs = SequentialFeatureSelector(estimator, n_features_to_select=3)
```

In [60]:
```
sfs.fit(X, y)
sfs.transform(X).shape
```

Out[60]:
```
(20640, 3)
```

In [48]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


# Data processing, modeling, and model evaluation
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score, classification_report, Confusio
```

In [49]:
```
dataset = pd.read_csv("ep.csv")
df = pd.DataFrame(dataset)
df.head()
```

Out[49]:

	Unnamed	X1	X2	X3	X4	X5	X6	X7	X8	X9	...	X170	X171	X172	X173
0	X21.V1.791	135	190	229	223	192	125	55	-9	-33	...	-17	-15	-31	-77
1	X15.V1.924	386	382	356	331	320	315	307	272	244	...	164	150	146	152
2	X8.V1.1	-32	-39	-47	-37	-32	-36	-57	-73	-85	...	57	64	48	19
3	X16.V1.60	-105	-101	-96	-92	-89	-95	-102	-100	-87	...	-82	-81	-80	-77
4	X20.V1.54	-9	-65	-98	-102	-78	-48	-16	0	-21	...	4	2	-12	-32

5 rows × 180 columns

In [50]:
```
df.pop('Unnamed')
```

Out[50]:
```
0         X21.V1.791
1         X15.V1.924
2            X8.V1.1
3          X16.V1.60
4          X20.V1.54
              ...
11495    X22.V1.114
11496    X19.V1.354
11497      X8.V1.28
11498    X10.V1.932
11499    X16.V1.210
Name: Unnamed, Length: 11500, dtype: object
```

In [51]:
```
df.head()
```

Out[51]:

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	...	X170	X171	X172	X173	X174
0	135	190	229	223	192	125	55	-9	-33	-38	...	-17	-15	-31	-77	-103
1	386	382	356	331	320	315	307	272	244	232	...	164	150	146	152	157
2	-32	-39	-47	-37	-32	-36	-57	-73	-85	-94	...	57	64	48	19	-12
3	-105	-101	-96	-92	-89	-95	-102	-100	-87	-79	...	-82	-81	-80	-77	-85
4	-9	-65	-98	-102	-78	-48	-16	0	-21	-59	...	4	2	-12	-32	-41

5 rows × 179 columns

In [52]:
```
df['y'] = [1 if x == 1 else 0 for x in df['y']]
```

In [53]:
```
df.head()
```

Out[53]:

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	...	X170	X171	X172	X173	X174
0	135	190	229	223	192	125	55	-9	-33	-38	...	-17	-15	-31	-77	-103
1	386	382	356	331	320	315	307	272	244	232	...	164	150	146	152	157
2	-32	-39	-47	-37	-32	-36	-57	-73	-85	-94	...	57	64	48	19	-12
3	-105	-101	-96	-92	-89	-95	-102	-100	-87	-79	...	-82	-81	-80	-77	-85
4	-9	-65	-98	-102	-78	-48	-16	0	-21	-59	...	4	2	-12	-32	-41

5 rows × 179 columns

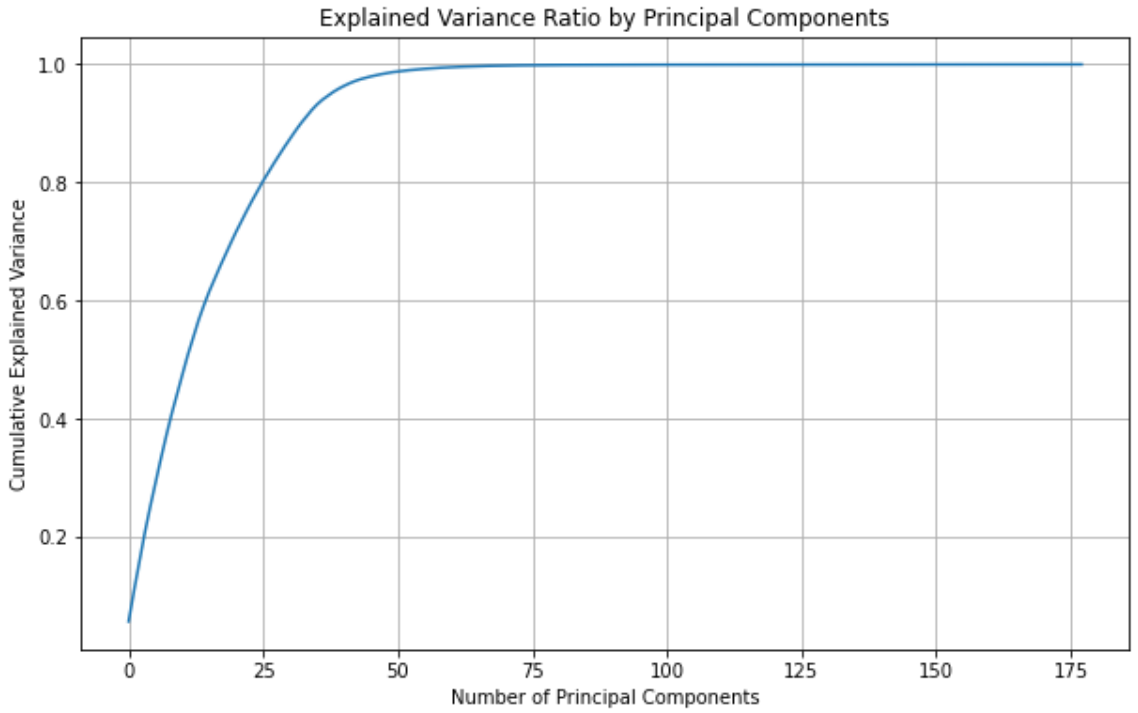
In [54]:
```
sum(df.isnull().sum())
```

Out[54]:
```
0
```

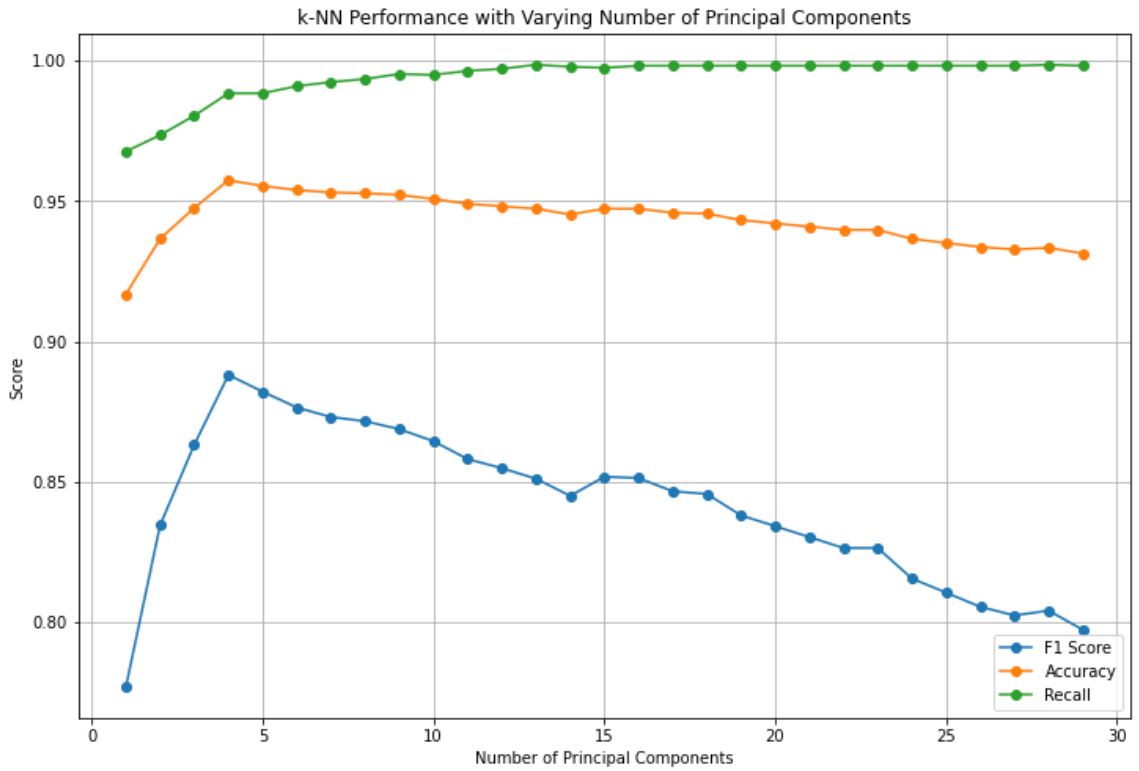
In [55]:
```
X = df.iloc[:,:-1]
y = df.iloc[:,-1]
```

In [61]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [57]:
```
pca = PCA(n_components=178)
X_pca = pca.fit_transform(X_scaled)

plt.figure(figsize=(10, 6))
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Explained Variance Ratio by Principal Components')
plt.grid(True)
plt.show()
```

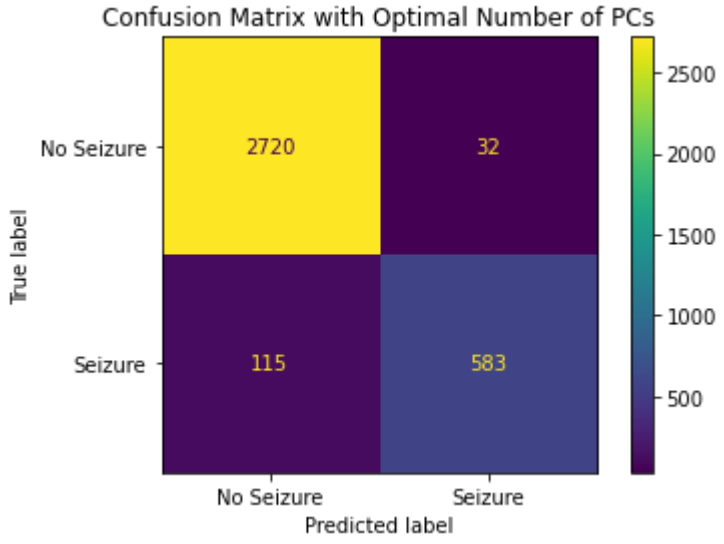


In [58]:
```
X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_si

results = {'num_pcs': [], 'f1_score': [], 'accuracy': [], 'recall': |

for n_pcs in range(1, 20):
    X_train_reduced = X_train[:, :n_pcs]
    X_test_reduced = X_test[:, :n_pcs]

    knn = KNeighborsClassifier()
    knn.fit(X_train_reduced, y_train)

    y_pred = knn.predict(X_test_reduced)
    f1 = f1_score(y_test, y_pred)
    accuracy = np.mean(y_pred == y_test)
    recall = classification_report(y_test, y_pred, output_dict=True)|

    results['num_pcs'].append(n_pcs)
    results['f1_score'].append(f1)
    results['accuracy'].append(accuracy)
    results['recall'].append(recall)


results_df = pd.DataFrame(results)

plt.figure(figsize=(12, 8))
plt.plot(results_df['num_pcs'], results_df['f1_score'], marker='o', l
plt.plot(results_df['num_pcs'], results_df['accuracy'], marker='o', l
plt.plot(results_df['num_pcs'], results_df['recall'], marker='o', lab
plt.xlabel('Number of Principal Components')
plt.ylabel('Score')
plt.title('k-NN Performance with Varying Number of Principal Componer
plt.legend()
plt.grid(True)
plt.show()
```



In [64]:
```
print(optimal_n_pcs)

X_train_opt = X_train[:, :optimal_n_pcs]
X_test_opt = X_test[:, :optimal_n_pcs]

knn_opt = KNeighborsClassifier()
knn_opt.fit(X_train_opt, y_train)

y_pred_opt = knn_opt.predict(X_test_opt)
accuracy_opt = np.mean(y_pred_opt == y_test)
f1_opt = f1_score(y_test, y_pred_opt)
report_opt = classification_report(y_test, y_pred_opt)

cm = confusion_matrix(y_test, y_pred_opt)

cm_display = ConfusionMatrixDisplay(confusion_matrix=cm, display_labe

plt.figure(figsize=(8, 8))
cm_display.plot(values_format='d')
plt.title('Confusion Matrix with Optimal Number of PCs')
plt.show()

print(f"Optimal number of PCs: {optimal_n_pcs}")
print(f"Accuracy: {accuracy_opt:.4f}")
print(f"F1 Score: {f1_opt:.4f}")
print("Classification Report:")
print(report_opt)
```

```
4

<Figure size 576x576 with 0 Axes>
```



```
Optimal number of PCs: 4
Accuracy: 0.9574
F1 Score: 0.8880
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.99      0.97      2752
           1       0.95      0.84      0.89       698

    accuracy                           0.96      3450
   macro avg       0.95      0.91      0.93      3450
weighted avg       0.96      0.96      0.96      3450
```

In [ ]:

eda-lab-6-b

August 19, 2024

[30]:
```
import numpy as np
import pandas as pd
```

[31]:
```
data = pd.read_csv('Data.csv')
data.head()
```

[31]:

	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

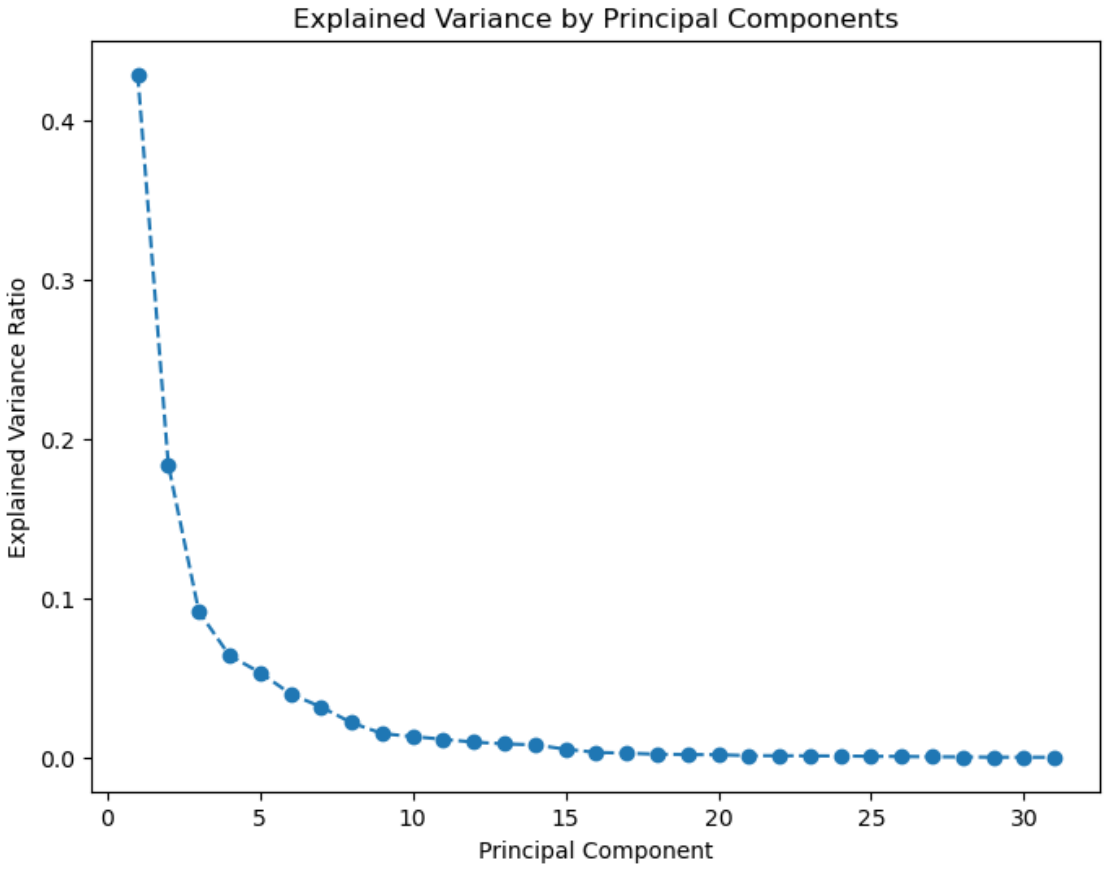
```
2                0.08758          NaN
3                0.17300          NaN
4                0.07678          NaN

[5 rows x 33 columns]
```

[32]:
```
data = data.drop(columns='Unnamed: 32')
```

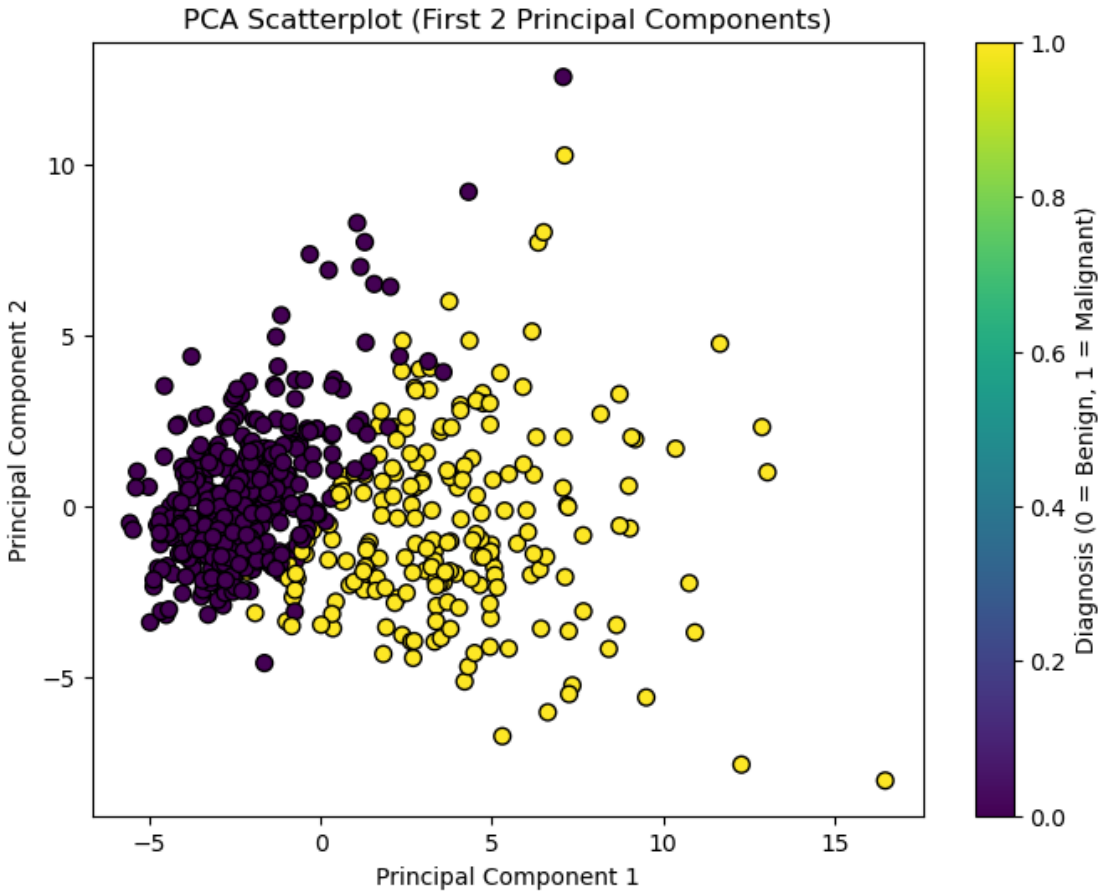
[33]:
```
X = data.drop(columns='diagnosis')
y = [0 if row == 'B' else 1 for row in data['diagnosis']]
```

[34]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

[35]:
```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

pca = PCA()
X_pca = pca.fit_transform(X_scaled)

explained_variance = pca.explained_variance_ratio_

# Plot the explained variance
plt.figure(figsize=(8, 6))
plt.plot(range(1, len(explained_variance) + 1), explained_variance, marker='o',␣
 ↪linestyle='--')
plt.title('Explained Variance by Principal Components')
plt.xlabel('Principal Component')
plt.ylabel('Explained Variance Ratio')
plt.show()
```



[37]:
```
pca_2 = PCA(n_components=2)
X_pca_2 = pca_2.fit_transform(X_scaled)

plt.figure(figsize=(8, 6))
plt.scatter(X_pca_2[:, 0], X_pca_2[:, 1], c=y, cmap='viridis', edgecolor='k',␣
↪s=50)
plt.title('PCA Scatterplot (First 2 Principal Components)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar(label='Diagnosis (0 = Benign, 1 = Malignant)')
plt.show()
```


PCA Scatterplot (First 2 Principal Components)
Principal Component 2
Principal Component 1
Diagnosis (0 = Benign, 1 = Malignant)
10
5
0
−5
−5
0
5
10
15
1.0
0.8
0.6
0.4
0.2
0.0

```
import numpy as np
import pandas as pd

dataset = pd.read_csv("diabetes.csv")
df = pd.DataFrame(dataset)
df.head()
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin
BMI  \
0            6      148             72             35        0  33.6

1            1       85             66             29        0  26.6

2            8      183             64              0        0  23.3

3            1       89             66             23       94  28.1

4            0      137             40             35      168  43.1


   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

```
df.shape
```

```
(768, 9)
```

```
X = dataset.drop(columns='Outcome')
y = dataset['Outcome']

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

chi2_selector = SelectKBest(chi2, k=4)
X_kbest = chi2_selector.fit_transform(X, y)

selected_features = chi2_selector.get_support(indices=True)
selected_feature_names = X.columns[selected_features]
print("Top 4 features selected using Chi-Squared test:",
selected_feature_names)
```

```
Top 4 features selected using Chi-Squared test: Index(['Glucose',
'Insulin', 'BMI', 'Age'], dtype='object')
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE

model = LogisticRegression(max_iter=1000)
```

```
rfe = RFE(estimator=model, n_features_to_select=3)

rfe.fit(X, y)

rfe_selected_features = rfe.get_support(indices=True)
rfe_selected_feature_names = X.columns[rfe_selected_features]
print("Top 3 features selected using RFE:",
rfe_selected_feature_names)
```

```
Top 3 features selected using RFE: Index(['Pregnancies', 'BMI',
'DiabetesPedigreeFunction'], dtype='object')
```

lab7

August 26, 2024

[39]:
```
import pandas as pd
import numpy as np
```

0.1 Task 1

[40]:
```
from sklearn.datasets import fetch_california_housing
housing = fetch_california_housing()
X = housing.data
y = housing.target
```

[41]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,␣
 ↪random_state=42)
```

0.2 Task 2

[42]:
```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train,y_train)
lr.score(X_test,y_test)
```

[42]:
```
0.5957702326061662
```

0.3 Task 3

[43]:
```
from sklearn.datasets import load_diabetes
X,y = load_diabetes(return_X_y=True)
```

[44]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.40,␣
 ↪random_state=42)
```

0.4 Task 4

[45]:
```
lr = LinearRegression()
lr.fit(X_train,y_train)
lr.score(X_test,y_test)
```

[45]: 0.515743631390243

0.5 Task 6

[46]:
```
lr.coef_
```

[46]:
```
array([  18.08799763, -227.04344876,  592.27723487,  361.54123241,
       -655.90738774,  353.71636413,   14.41265469,  142.87369371,
        594.01542882,   31.67317969])
```

[47]:
```
lr.intercept_
```

[47]: 148.92850837170067

0.6 Task 5

[48]:
```
lr = LinearRegression(fit_intercept=False)
lr.fit(X_train,y_train)
lr.score(X_test,y_test)
```

[48]: -3.7861097351892816

0.7 Task 7

[49]:
```
from sklearn.datasets import load_diabetes
X,y = load_diabetes(return_X_y=True)
```

[50]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.40,␣
↪random_state=42)
```

[60]:
```
from sklearn.linear_model import SGDRegressor
sgdreg = SGDRegressor(max_iter=10000)
sgdreg.fit(X_train,y_train)
sgdreg.score(X_test,y_test)
```

[60]: 0.5093237977527326

lab8

September 2, 2024

[123]:
```
import pandas as pd
import numpy as np
```

[124]:
```
column_name=["Age","Sex","cp","trestbps","chol","fbs","restecg","thalach","exang","oldpeak","s
```

[125]:
```
dataset = pd.read_csv("processed.cleveland.data",names=column_name,header=None)
```

[126]:
```
df = pd.DataFrame(dataset)
df.tail()
```

[126]:

	Age	Sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	\
298	45.0	1.0	1.0	110.0	264.0	0.0	0.0	132.0	0.0	1.2	
299	68.0	1.0	4.0	144.0	193.0	1.0	0.0	141.0	0.0	3.4	
300	57.0	1.0	4.0	130.0	131.0	0.0	0.0	115.0	1.0	1.2	
301	57.0	0.0	2.0	130.0	236.0	0.0	2.0	174.0	0.0	0.0	
302	38.0	1.0	3.0	138.0	175.0	0.0	0.0	173.0	0.0	0.0	

	slope	ca	thal	num
298	2.0	0.0	7.0	1
299	2.0	2.0	7.0	2
300	2.0	1.0	7.0	3
301	2.0	1.0	3.0	1
302	1.0	?	3.0	0

[127]:
```
df.isnull().sum()
```

[127]:
```
Age         0
Sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
```

```
ca          0
thal        0
num         0
dtype: int64
```

[128]:
```
df['num'] = [1 if int(num) in [2,3,4] else 0 for num in df['num']]
```

[129]:
```
df['thal'].unique()
```

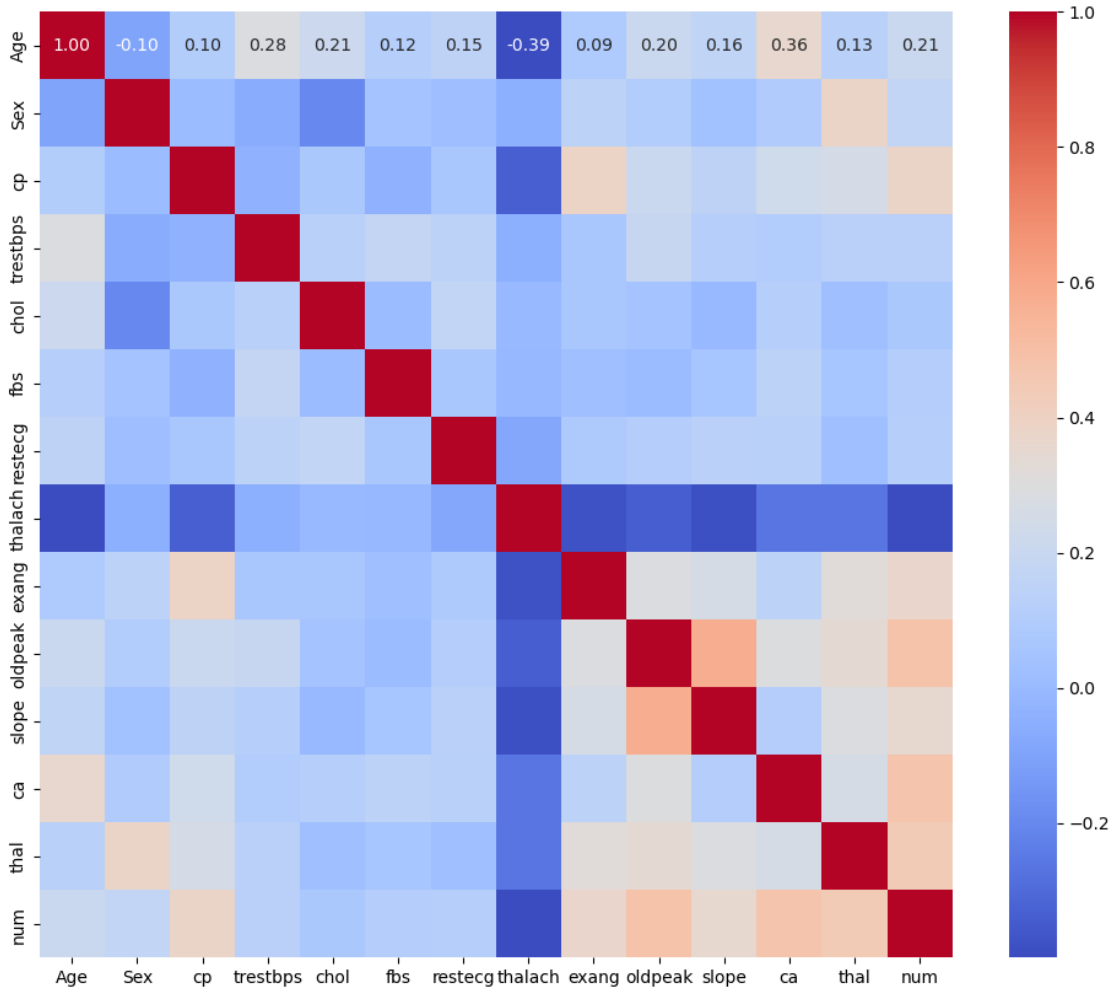
[129]:
```
array(['6.0', '3.0', '7.0', '?'], dtype=object)
```

[130]:
```
df['ca'] = [np.nan if val == '?' else val for val in df['ca']]
df['ca'] = [int(float(val)) if val in ['0.0', '3.0', '2.0', '1.0'] else val for␣
 ↪val in df['ca']]
df['ca'].fillna(value=df['ca'].mean(),inplace=True)
```

[131]:
```
df['thal'] = [np.nan if val == '?' else val for val in df['thal']]
df['thal'] = [int(float(val)) if val in ['6.0', '3.0', '7.0'] else val for val␣
 ↪in df['thal']]
df['thal'].fillna(value=df['ca'].mean(),inplace=True)
```

[133]:
```
# Separate features and target
X = df.drop(columns='num')
y = df['num']
```

[135]:
```
import seaborn as sns
import matplotlib.pyplot as plt

corr = df.corr()

# Plot the heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt='.2f')
plt.show()
```



[136]:
```
from sklearn.model_selection import train_test_split

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[139]:
```
from sklearn.preprocessing import StandardScaler

# Normalize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[143]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
```

```
model = LogisticRegression()
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)

print(f"Training Score: {model.score(X_train_scaled, y_train)}")

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Training Score: 0.9008264462809917
Confusion Matrix:
[[34  7]
 [ 7 13]]

Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.83      0.83        41
           1       0.65      0.65      0.65        20

    accuracy                           0.77        61
   macro avg       0.74      0.74      0.74        61
weighted avg       0.77      0.77      0.77        61
```

[144]:
```
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV

# Define parameter grid
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear', 'saga']
}

# Randomized Search
random_search = RandomizedSearchCV(LogisticRegression(max_iter=1000),␣
 ↪param_distributions=param_grid, n_iter=10, cv=5, random_state=42)
random_search.fit(X_train_scaled, y_train)
print("Best parameters from RandomizedSearchCV:")
print(random_search.best_params_)

# Grid Search
grid_search = GridSearchCV(LogisticRegression(max_iter=1000), param_grid, cv=5)
```

```
grid_search.fit(X_train_scaled, y_train)
print("Best parameters from GridSearchCV:")
print(grid_search.best_params_)
```

```
Best parameters from RandomizedSearchCV:
{'solver': 'liblinear', 'penalty': 'l1', 'C': 10}
Best parameters from GridSearchCV:
{'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
```

lab8b

September 2, 2024

[135]:
```
import numpy as np# Summary of results
results = {
    'OLS': ols_mse,
    'Ridge': ridge_mse,
    'Lasso': lasso_mse
}

best_model = min(results, key=results.get)
print(f"The model with the highest accuracy (lowest MSE) is: {best_model}")

import pandas as pd
```

```
The model with the highest accuracy (lowest MSE) is: Lasso
```

[136]:
```
dataset = pd.read_csv("Hitters.csv")
df = pd.DataFrame(dataset)
df
```

[136]:

	Unnamed: 0	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	\
0	-Andy Allanson	293	66	1	30	29	14	1	293	
1	-Alan Ashby	315	81	7	24	38	39	14	3449	
2	-Alvin Davis	479	130	18	66	72	76	3	1624	
3	-Andre Dawson	496	141	20	65	78	37	11	5628	
4	-Andres Galarraga	321	87	10	39	42	30	2	396	
..	…	…	…	…	…	…	…	…	…	
317	-Willie McGee	497	127	7	65	48	37	5	2703	
318	-Willie Randolph	492	136	5	76	50	94	12	5511	
319	-Wayne Tolleson	475	126	3	61	43	52	6	1700	
320	-Willie Upshaw	573	144	9	85	60	78	8	3198	
321	-Willie Wilson	631	170	9	77	44	31	11	4908	

	CHits	…	CRuns	CRBI	CWalks	League	Division	PutOuts	Assists	\
0	66	…	30	29	14	A	E	446	33	
1	835	…	321	414	375	N	W	632	43	
2	457	…	224	266	263	A	W	880	82	
3	1575	…	828	838	354	N	E	200	11	
4	101	…	48	46	33	N	E	805	40	
..	…	…	…	…	…	…	…	…	…	

```
317    806  …    379   311      138       N        E     325        9
318   1511  …    897   451      875       A        E     313      381
319    433  …    217    93      146       A        W      37      113
320    857  …    470   420      332       A        E    1314      131
321   1457  …    775   357      249       A        W     408        4

     Errors  Salary  NewLeague
0        20     NaN          A
1        10   475.0          N
2        14   480.0          A
3         3   500.0          N
4         4    91.5          N
..      …     …        …
317       3   700.0          N
318      20   875.0          A
319       7   385.0          A
320      12   960.0          A
321       3  1000.0          A

[322 rows x 21 columns]
```

[137]:
```
df.dtypes
```

[137]:
```
Unnamed: 0     object
AtBat           int64
Hits            int64
HmRun           int64
Runs            int64
RBI             int64
Walks           int64
Years           int64
CAtBat          int64
CHits           int64
CHmRun          int64
CRuns           int64
CRBI            int64
CWalks          int64
League         object
Division       object
PutOuts         int64
Assists         int64
Errors          int64
Salary        float64
NewLeague      object
dtype: object
```

[138]:
```
df.pop('Unnamed: 0')
```

[138]:
```
0          -Andy Allanson
1             -Alan Ashby
2           -Alvin Davis
3          -Andre Dawson
4       -Andres Galarraga
              ...
317         -Willie McGee
318      -Willie Randolph
319       -Wayne Tolleson
320        -Willie Upshaw
321        -Willie Wilson
Name: Unnamed: 0, Length: 322, dtype: object
```

[139]:
```
df.head()
```

[139]:

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns	\
0	293	66	1	30	29	14	1	293	66	1	30	
1	315	81	7	24	38	39	14	3449	835	69	321	
2	479	130	18	66	72	76	3	1624	457	63	224	
3	496	141	20	65	78	37	11	5628	1575	225	828	
4	321	87	10	39	42	30	2	396	101	12	48	

	CRBI	CWalks	League	Division	PutOuts	Assists	Errors	Salary	NewLeague
0	29	14	A	E	446	33	20	NaN	A
1	414	375	N	W	632	43	10	475.0	N
2	266	263	A	W	880	82	14	480.0	A
3	838	354	N	E	200	11	3	500.0	N
4	46	33	N	E	805	40	4	91.5	N

[140]:
```
df.describe()
```

[140]:

	AtBat	Hits	HmRun	Runs	RBI	Walks	\
count	322.000000	322.000000	322.000000	322.000000	322.000000	322.000000	
mean	380.928571	101.024845	10.770186	50.909938	48.027950	38.742236	
std	153.404981	46.454741	8.709037	26.024095	26.166895	21.639327	
min	16.000000	1.000000	0.000000	0.000000	0.000000	0.000000	
25%	255.250000	64.000000	4.000000	30.250000	28.000000	22.000000	
50%	379.500000	96.000000	8.000000	48.000000	44.000000	35.000000	
75%	512.000000	137.000000	16.000000	69.000000	64.750000	53.000000	
max	687.000000	238.000000	40.000000	130.000000	121.000000	105.000000	

	Years	CAtBat	CHits	CHmRun	CRuns	\
count	322.000000	322.00000	322.000000	322.000000	322.000000	
mean	7.444099	2648.68323	717.571429	69.490683	358.795031	
std	4.926087	2324.20587	654.472627	86.266061	334.105886	
min	1.000000	19.00000	4.000000	0.000000	1.000000	
25%	4.000000	816.75000	209.000000	14.000000	100.250000	

```
50%      6.000000   1928.00000   508.000000   37.500000   247.000000
75%     11.000000   3924.25000  1059.250000   90.000000   526.250000
max     24.000000  14053.00000  4256.000000  548.000000  2165.000000

              CRBI       CWalks      PutOuts     Assists      Errors  \
count   322.000000   322.000000   322.000000  322.000000  322.000000
mean    330.118012   260.239130   288.937888  106.913043    8.040373
std     333.219617   267.058085   280.704614  136.854876    6.368359
min       0.000000     0.000000     0.000000    0.000000    0.000000
25%      88.750000    67.250000   109.250000    7.000000    3.000000
50%     220.500000   170.500000   212.000000   39.500000    6.000000
75%     426.250000   339.250000   325.000000  166.000000   11.000000
max    1659.000000  1566.000000  1378.000000  492.000000   32.000000

            Salary
count   263.000000
mean    535.925882
std     451.118681
min      67.500000
25%     190.000000
50%     425.000000
75%     750.000000
max    2460.000000
```

[141]:
```
df.isnull().sum()
```

[141]:
```
AtBat         0
Hits          0
HmRun         0
Runs          0
RBI           0
Walks         0
Years         0
CAtBat        0
CHits         0
CHmRun        0
CRuns         0
CRBI          0
CWalks        0
League        0
Division      0
PutOuts       0
Assists       0
Errors        0
Salary       59
NewLeague     0
dtype: int64
```

[142]:
```
df['Salary'].fillna(value=df['Salary'].mean(),inplace=True)
```

[143]:
```
from scipy import stats
import numpy as np

# Calculate Z-scores
z_scores = np.abs(stats.zscore(df.select_dtypes(include=np.number)))

threshold = 3
outliers = (z_scores > threshold)

print(np.where(outliers)[0])

df = df.drop(index=np.where(outliers)[0])
```

```
[ 30  32  73  73  80  82  82  84  96 100 112 113 114 121 136 163 163 179
 179 180 189 217 229 235 236 236 236 236 236 248 249 249 249 249 260 272
 274 276 278 292 302 302 302 302 302 306 310 313 315 320]
```

[144]:
```
df.dtypes
```

[144]:
```
AtBat          int64
Hits           int64
HmRun          int64
Runs           int64
RBI            int64
Walks          int64
Years          int64
CAtBat         int64
CHits          int64
CHmRun         int64
CRuns          int64
CRBI           int64
CWalks         int64
League        object
Division      object
PutOuts        int64
Assists        int64
Errors         int64
Salary       float64
NewLeague     object
dtype: object
```

[145]:
```
# Convert categorical variables into dummy/indicator variables
df = pd.get_dummies(df, columns=['Division', 'League',␣
 ↪'NewLeague'],drop_first=True)
```

[146]:
```
df
```

[146]:

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	\
0	293	66	1	30	29	14	1	293	66	1	
1	315	81	7	24	38	39	14	3449	835	69	
2	479	130	18	66	72	76	3	1624	457	63	
3	496	141	20	65	78	37	11	5628	1575	225	
4	321	87	10	39	42	30	2	396	101	12	
..	…	…	…	…	…	…	…	…	…	…	
316	221	53	2	21	23	22	8	1063	283	15	
317	497	127	7	65	48	37	5	2703	806	32	
318	492	136	5	76	50	94	12	5511	1511	39	
319	475	126	3	61	43	52	6	1700	433	7	
321	631	170	9	77	44	31	11	4908	1457	30	

	CRuns	CRBI	CWalks	PutOuts	Assists	Errors	Salary	Division_W	\
0	30	29	14	446	33	20	535.925882	False	
1	321	414	375	632	43	10	475.000000	True	
2	224	266	263	880	82	14	480.000000	True	
3	828	838	354	200	11	3	500.000000	False	
4	48	46	33	805	40	4	91.500000	False	
..	…	…	…	…	…	…	…	…	
316	107	124	106	325	58	6	535.925882	False	
317	379	311	138	325	9	3	700.000000	False	
318	897	451	875	313	381	20	875.000000	False	
319	217	93	146	37	113	7	385.000000	True	
321	775	357	249	408	4	3	1000.000000	True	

	League_N	NewLeague_N
0	False	False
1	True	True
2	False	False
3	True	True
4	True	True
..	…	…
316	True	True
317	True	True
318	False	False
319	False	False
321	False	False

[287 rows x 20 columns]

[147]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import mean_squared_error
```

```
# Prepare features and target variable
X = df.drop('Salary', axis=1)  # Assuming 'Salary' is the target variable
y = df['Salary']

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# OLS Regression
ols_model = LinearRegression()
ols_model.fit(X_train, y_train)
ols_predictions = ols_model.predict(X_test)
ols_mse = mean_squared_error(y_test, ols_predictions)

# Ridge Regression
ridge_model = Ridge(alpha=1.0)
ridge_model.fit(X_train, y_train)
ridge_predictions = ridge_model.predict(X_test)
ridge_mse = mean_squared_error(y_test, ridge_predictions)

# Lasso Regression
lasso_model = Lasso(alpha=0.1)
lasso_model.fit(X_train, y_train)
lasso_predictions = lasso_model.predict(X_test)
lasso_mse = mean_squared_error(y_test, lasso_predictions)

print(f"OLS Mean Squared Error: {ols_mse}")
print(f"Ridge Mean Squared Error: {ridge_mse}")
print(f"Lasso Mean Squared Error: {lasso_mse}")
```

```
OLS Mean Squared Error: 59666.2269633068
Ridge Mean Squared Error: 59599.27851989654
Lasso Mean Squared Error: 59594.05119739496
```

```
/opt/anaconda3/lib/python3.11/site-
packages/sklearn/linear_model/_coordinate_descent.py:631: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 5.386e+06, tolerance: 2.068e+03
  model = cd_fast.enet_coordinate_descent(
```

[148]:
```
# Summary of results
results = {
    'OLS': ols_mse,
    'Ridge': ridge_mse,
    'Lasso': lasso_mse
}
```

```
best_model = min(results, key=results.get)
print(f"The model with the highest accuracy (lowest MSE) is: {best_model}")
```

```
The model with the highest accuracy (lowest MSE) is: Lasso
```

[ ]:

```
# Common imports
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import SGDClassifier
from sklearn.utils import shuffle
import matplotlib.pyplot as plt

# Fetch the MNIST dataset from openml
mnist = fetch_openml('mnist_784', version=1)

# Store samples in X and labels in y
X, y = mnist["data"], mnist["target"]

# Convert labels to integers
y = y.astype(np.int32)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/datasets/_openml.py:1022: FutureWarning: The default value of `parser` will change f
  warn(
```

```
# Splitting the dataset: 70% training, 30% testing
train_size = int(0.7 * len(X))

x_train, x_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]



last_training_sample_digit = list(y_train)[-1]
print(f"The last training sample is of digit {last_training_sample_digit}.")
```

```
The last training sample is of digit 6.
```

```
# Collect digit-6 and digit-9 images
six_nine_indices = (y_train == 6) | (y_train == 9)
x_train_69 = x_train[six_nine_indices]
y_train_69 = y_train[six_nine_indices]

# Set labels: 1 for digit 6, 0 for digit 9
y_train_69 = (y_train_69 == 6).astype(int)

# Shuffle the data
x_train_69, y_train_69 = shuffle(x_train_69, y_train_69, random_state=1729)

# Similarly, create x_test_69 and y_test_69
six_nine_indices_test = (y_test == 6) | (y_test == 9)
x_test_69 = x_test[six_nine_indices_test]
y_test_69 = y_test[six_nine_indices_test]

y_test_69 = (y_test_69 == 6).astype(int)



sum_train_labels = np.sum(y_train_69)
sum_test_labels = np.sum(y_test_69)

print(f"Sum of labels in y_train_69: {sum_train_labels}")
print(f"Sum of labels in y_test_69: {sum_test_labels}")
```

```
Sum of labels in y_train_69: 4855
Sum of labels in y_test_69: 2021
```

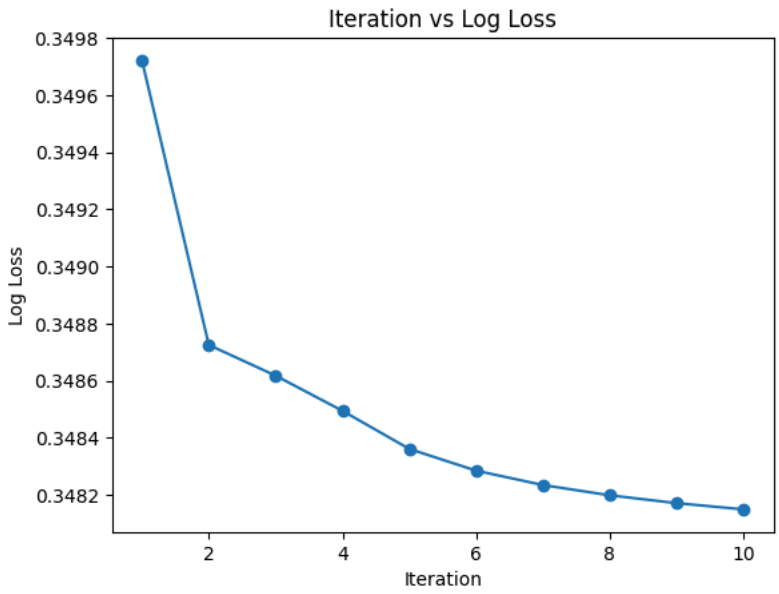
```
# Apply StandardScaler
scaler = StandardScaler()
x_train_69Tf = scaler.fit_transform(x_train_69)

# Mean and standard deviation of the zeroth sample and feature
mean_zeroth_sample = np.mean(x_train_69Tf[0])
mean_zeroth_feature = np.mean(x_train_69Tf[:, 0])
std_zeroth_sample = np.std(x_train_69Tf[0])
std_zeroth_feature = np.std(x_train_69Tf[:, 0])

results_tuple = (mean_zeroth_sample, mean_zeroth_feature, std_zeroth_sample, std_zeroth_feature)
print(results_tuple)
```

```
(0.1285224869548995, 0.0, 3.888657702544401, 0.0)
```

```
# Train Logistic Regression model using SGDClassifier
sgd_clf = SGDClassifier(loss='log_loss', max_iter=10, random_state=10, learning_rate='constant', eta0=0.01)
sgd_clf.fit(x_train_69Tf, y_train_69)
```

```
SGDClassifier(eta0=0.01, learning_rate='constant', loss='log_loss', max_iter=10,
              random_state=10)
```

```
# Train Logistic Regression model using SGDClassifier
sgd_clf = SGDClassifier(loss='log_loss', max_iter=10, random_state=10, learning_rate='constant', eta0=0.01)

# Initialize the loss list to capture the loss after each iteration
losses = []
for _ in range(10):
    sgd_clf.partial_fit(x_train_69Tf, y_train_69, classes=np.unique(y_train_69))

    # Calculate the log loss
    y_pred_prob = sgd_clf.decision_function(x_train_69Tf)
    log_loss_value = np.mean(np.log(1 + np.exp(-y_train_69 * y_pred_prob)))

    # Append the log loss value for this iteration
    losses.append(log_loss_value)

# Plotting the iteration vs loss curve
import matplotlib.pyplot as plt
plt.plot(range(1, 11), losses, marker='o')
plt.xlabel('Iteration')
plt.ylabel('Log Loss')
plt.title('Iteration vs Log Loss')
plt.show()
```

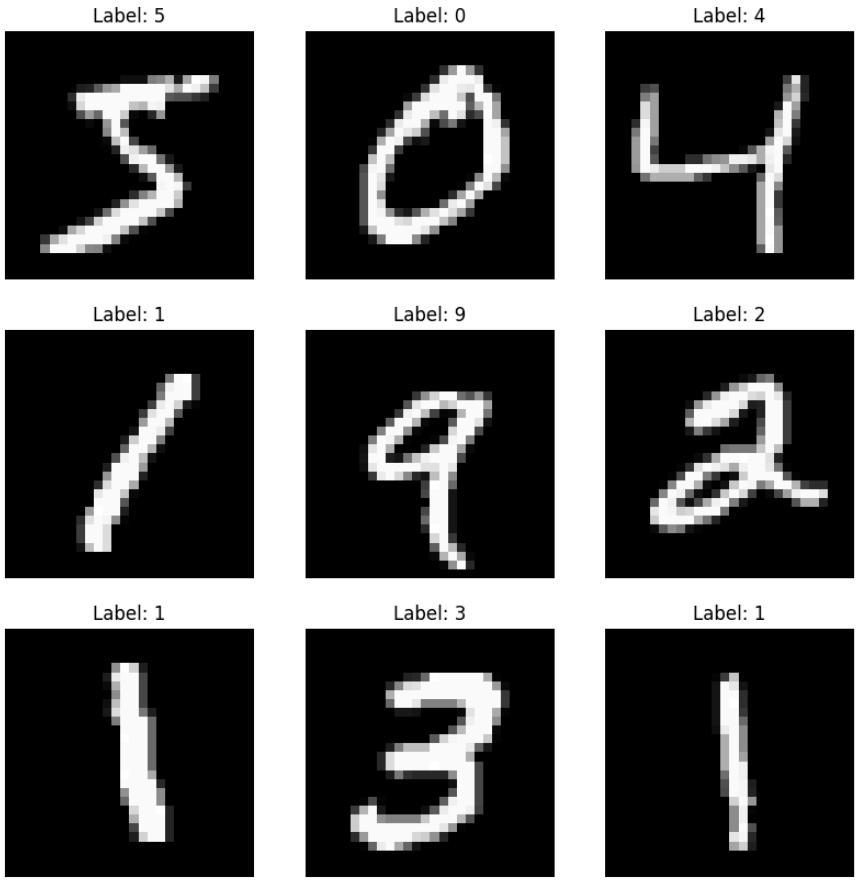


Start coding or generate with AI.

```
# Import basic libraries
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Import the libraries for performing classification
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import cross_val_score, train_test_split, GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix


# Fetch the MNIST dataset from openml
from sklearn.datasets import fetch_openml

mnist = fetch_openml('mnist_784', version=1)
X, y = mnist["data"], mnist["target"]

# Convert target labels to integers
y = y.astype(np.int8)

# Check the shapes of the dataset
print("Shape of X (data):", X.shape)  # Should be (70000, 784)
print("Shape of y (labels):", y.shape)  # Should be (70000,)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/datasets/_openml.py:1022: FutureWarning: The default value of `parser` will change f
  warn(
Shape of X (data): (70000, 784)
Shape of y (labels): (70000,)
```

```
# Plot the first few images from the dataset
plt.figure(figsize=(10, 10))
for i in range(9):
    plt.subplot(3, 3, i+1)
    # Access the i-th row of the DataFrame using .iloc[]
    plt.imshow(X.iloc[i].values.reshape(28, 28), cmap='gray')
    plt.title(f"Label: {y[i]}")
    plt.axis('off')
plt.show()
```




```
# Flatten each input image into a vector of length 784 and normalize the data
X = X / 255.0  # Normalize

# Check the shape of the dataset
print("Shape after flattening and normalizing, X:", X.shape)
```

```
Shape after flattening and normalizing, X: (70000, 784)
```

```
# Splitting the dataset
X_train, X_test, y_train, y_test = X[:10000], X[10000:12000], y[:10000], y[10000:12000]

# Check the shape of the training and testing sets
print("Training data shape:", X_train.shape)
print("Testing data shape:", X_test.shape)
```

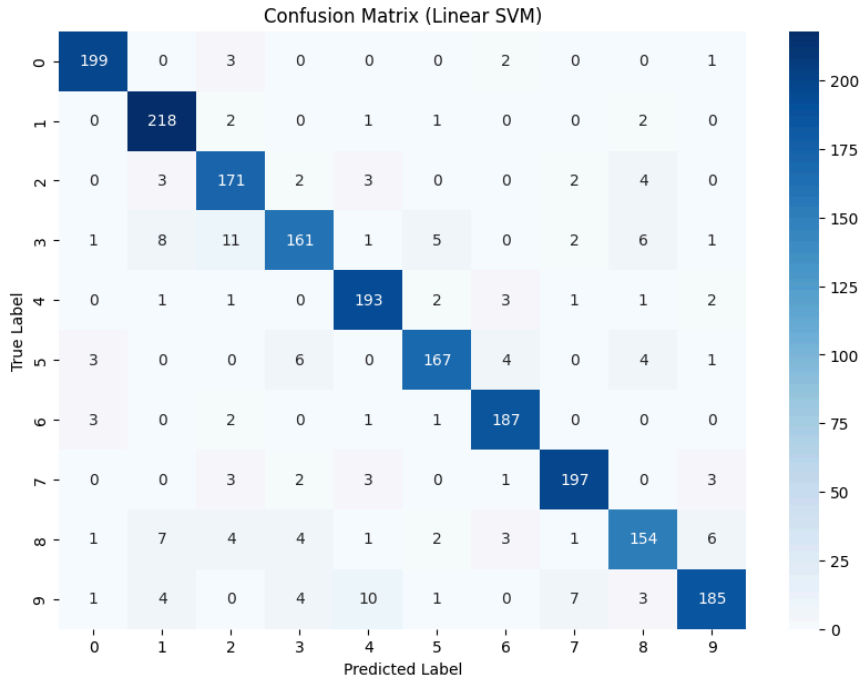
```
Training data shape: (10000, 784)
Testing data shape: (2000, 784)
```

```
# Train Linear SVM using a pipeline with MinMaxScaler and SVC with a linear kernel
pipe_1 = Pipeline([('scaler', MinMaxScaler()),
                   ('classifier', SVC(kernel='linear', C=1))])

# Fit the model
pipe_1.fit(X_train, y_train.ravel())

# Evaluate using cross-validation
acc = cross_val_score(pipe_1, X_train, y_train.ravel(), cv=2)
print("Training Accuracy with Linear SVM: {:.2f} %".format(acc.mean() * 100))
```

```
Training Accuracy with Linear SVM: 91.07 %
```

```
# Predicting on test data
y_pred = pipe_1.predict(X_test)
```

```
# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plotting heatmap for confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix (Linear SVM)")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

# Classification report
print("Classification Report for Linear SVM:\n")
print(classification_report(y_test, y_pred))
```

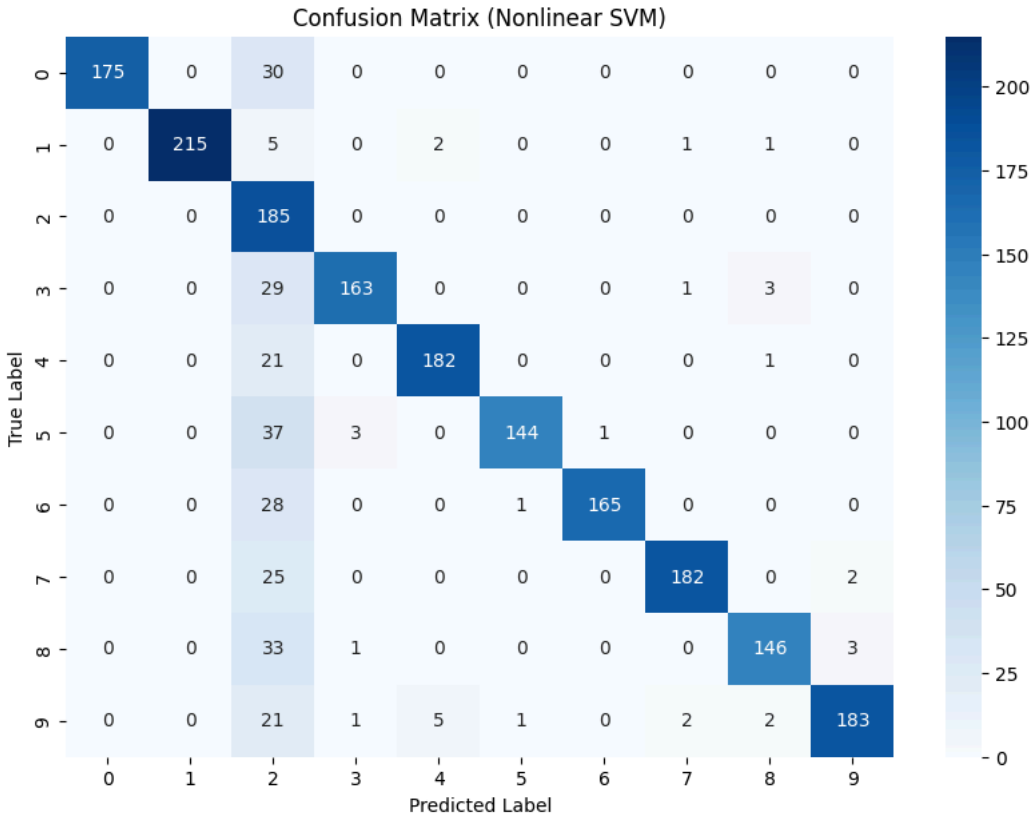



```
Classification Report for Linear SVM:

              precision    recall  f1-score   support

           0       0.96      0.97      0.96       205
           1       0.90      0.97      0.94       224
           2       0.87      0.92      0.90       185
           3       0.90      0.82      0.86       196
           4       0.91      0.95      0.93       204
           5       0.93      0.90      0.92       185
           6       0.94      0.96      0.95       194
           7       0.94      0.94      0.94       209
           8       0.89      0.84      0.86       183
           9       0.93      0.86      0.89       215

    accuracy                           0.92      2000
   macro avg       0.92      0.91      0.91      2000
weighted avg       0.92      0.92      0.92      2000
```

```
# Train Nonlinear SVM using a pipeline with MinMaxScaler and SVC with RBF kernel
pipe_2 = Pipeline([('scaler', MinMaxScaler()),
                   ('classifier', SVC(kernel='rbf', gamma=0.1, C=1))])

# Fit the model
pipe_2.fit(X_train, y_train.ravel())

# Evaluate using cross-validation
acc = cross_val_score(pipe_2, X_train, y_train.ravel(), cv=2)
print("Training Accuracy with Nonlinear SVM: {:.2f} %".format(acc.mean() * 100))
```

```
Training Accuracy with Nonlinear SVM: 82.87 %
```

```
# Predicting on test data
y_pred_rbf = pipe_2.predict(X_test)

# Confusion matrix
conf_matrix_rbf = confusion_matrix(y_test, y_pred_rbf)

# Plotting heatmap for confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix_rbf, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix (Nonlinear SVM)")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

# Classification report
print("Classification Report for Nonlinear SVM:\n")
print(classification_report(y_test, y_pred_rbf))
```

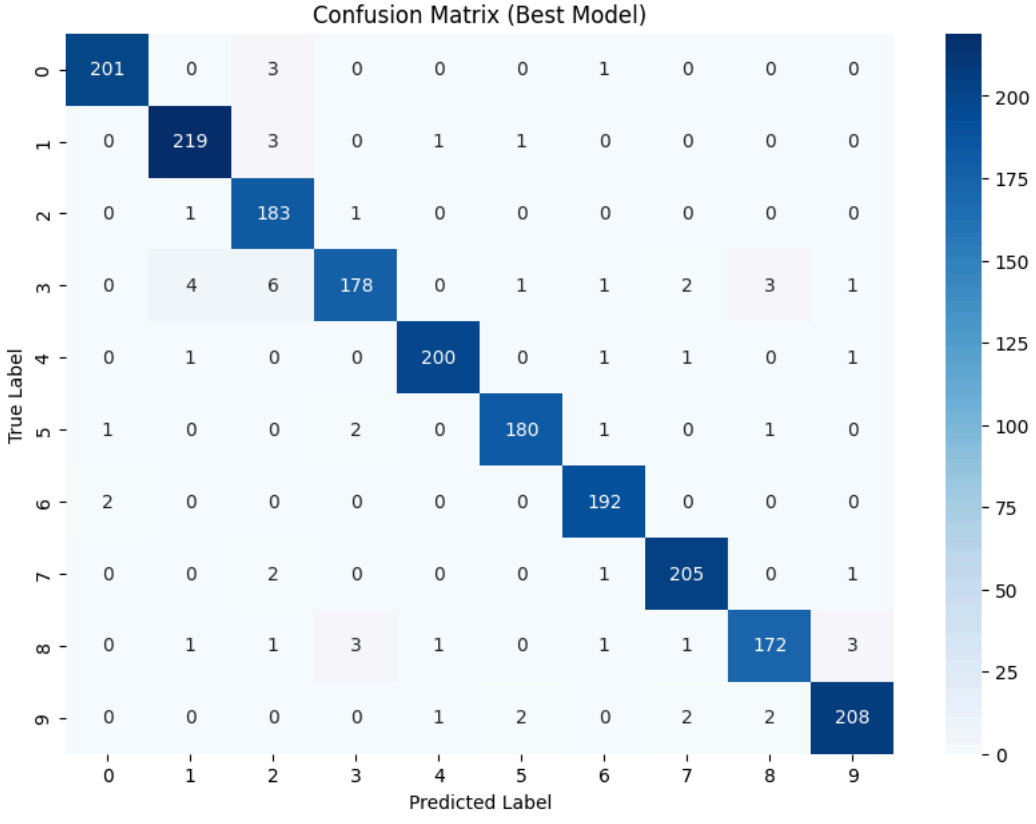



```
Classification Report for Nonlinear SVM:

              precision    recall  f1-score   support

           0       1.00      0.85      0.92       205
           1       1.00      0.96      0.98       224
           2       0.45      1.00      0.62       185
           3       0.97      0.83      0.90       196
           4       0.96      0.89      0.93       204
           5       0.99      0.78      0.87       185
           6       0.99      0.85      0.92       194
           7       0.98      0.87      0.92       209
           8       0.95      0.80      0.87       183
           9       0.97      0.85      0.91       215

    accuracy                           0.87      2000
   macro avg       0.93      0.87      0.88      2000
weighted avg       0.93      0.87      0.89      2000
```

```
# Define parameter grid
param_grid = {
    'classifier__C': [0.1, 1, 10],
    'classifier__gamma': [0.01, 0.1, 1]
}

# Set up the pipeline
pipe = Pipeline([('scaler', MinMaxScaler()),
                 ('classifier', SVC(kernel='rbf'))])

# Use GridSearchCV to find the best parameters
grid_search = GridSearchCV(pipe, param_grid, cv=3, scoring='accuracy')
grid_search.fit(X_train, y_train)
```

```
# Best parameters
print("Best parameters found by GridSearch:", grid_search.best_params_)

# Evaluate best model
best_model = grid_search.best_estimator_
y_pred_best = best_model.predict(X_test)

# Confusion matrix and classification report for the best model
conf_matrix_best = confusion_matrix(y_test, y_pred_best)

plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix_best, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix (Best Model)")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

# Print classification report
print("Classification Report for Best Model:\n")
print(classification_report(y_test, y_pred_best))
```

```
Best parameters found by GridSearch: {'classifier__C': 10, 'classifier__gamma': 0.01}
```



lab10

September 16, 2024

[135]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
```

[136]:
```
dataset = pd.read_csv("general_data.csv")
df = pd.DataFrame(dataset)
df.head()
```

[136]:

	Age	Attrition	BusinessTravel	Department	DistanceFromHome	\
0	51	No	Travel_Rarely	Sales	6	
1	31	Yes	Travel_Frequently	Research & Development	10	
2	32	No	Travel_Frequently	Research & Development	17	
3	38	No	Non-Travel	Research & Development	2	
4	32	No	Travel_Rarely	Research & Development	10	

	Education	EducationField	EmployeeCount	EmployeeID	Gender	…	\
0	2	Life Sciences	1	1	Female	…	
1	1	Life Sciences	1	2	Female	…	
2	4	Other	1	3	Male	…	
3	5	Life Sciences	1	4	Male	…	
4	1	Medical	1	5	Male	…	

	NumCompaniesWorked	Over18	PercentSalaryHike	StandardHours	\
0	1.0	Y	11	8	
1	0.0	Y	23	8	
2	1.0	Y	15	8	
3	3.0	Y	11	8	
4	4.0	Y	12	8	

	StockOptionLevel	TotalWorkingYears	TrainingTimesLastYear	YearsAtCompany	\
0	0	1.0	6	1	
1	1	6.0	3	5	
2	3	5.0	2	5	
3	3	13.0	5	8	
4	2	9.0	2	6	

	YearsSinceLastPromotion	YearsWithCurrManager

```
0                       0                       0
1                       1                       4
2                       0                       3
3                       7                       5
4                       0                       4

[5 rows x 24 columns]
```

[137]:
```
df.isnull().sum()
```

[137]:
```
Age                       0
Attrition                 0
BusinessTravel            0
Department                0
DistanceFromHome          0
Education                 0
EducationField            0
EmployeeCount             0
EmployeeID                0
Gender                    0
JobLevel                  0
JobRole                   0
MaritalStatus             0
MonthlyIncome             0
NumCompaniesWorked       19
Over18                    0
PercentSalaryHike         0
StandardHours             0
StockOptionLevel          0
TotalWorkingYears         9
TrainingTimesLastYear     0
YearsAtCompany            0
YearsSinceLastPromotion   0
YearsWithCurrManager      0
dtype: int64
```

[138]:
```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy='mean')

num_companies_reshaped = df[['NumCompaniesWorked']].values

df['NumCompaniesWorked'] = imputer.fit_transform(num_companies_reshaped)

df['NumCompaniesWorked'] = df['NumCompaniesWorked'].squeeze()
```

[139]:
```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy='mean')

num_companies_reshaped = df[['TotalWorkingYears']].values

df['TotalWorkingYears'] = imputer.fit_transform(num_companies_reshaped)

df['TotalWorkingYears'] = df['TotalWorkingYears'].squeeze()
```

[140]:
```
df.isnull().sum()
```

[140]:
```
Age                       0
Attrition                 0
BusinessTravel            0
Department                0
DistanceFromHome          0
Education                 0
EducationField            0
EmployeeCount             0
EmployeeID                0
Gender                    0
JobLevel                  0
JobRole                   0
MaritalStatus             0
MonthlyIncome             0
NumCompaniesWorked        0
Over18                    0
PercentSalaryHike         0
StandardHours             0
StockOptionLevel          0
TotalWorkingYears         0
TrainingTimesLastYear     0
YearsAtCompany            0
YearsSinceLastPromotion   0
YearsWithCurrManager      0
dtype: int64
```

[141]:
```
df.columns
```

[141]:
```
Index(['Age', 'Attrition', 'BusinessTravel', 'Department', 'DistanceFromHome',
       'Education', 'EducationField', 'EmployeeCount', 'EmployeeID', 'Gender',
       'JobLevel', 'JobRole', 'MaritalStatus', 'MonthlyIncome',
       'NumCompaniesWorked', 'Over18', 'PercentSalaryHike', 'StandardHours',
       'StockOptionLevel', 'TotalWorkingYears', 'TrainingTimesLastYear',
       'YearsAtCompany', 'YearsSinceLastPromotion', 'YearsWithCurrManager'],
      dtype='object')
```

[142]:
```
df.head()
```

[142]:

	Age	Attrition	BusinessTravel	Department	DistanceFromHome	\
0	51	No	Travel_Rarely	Sales	6	
1	31	Yes	Travel_Frequently	Research & Development	10	
2	32	No	Travel_Frequently	Research & Development	17	
3	38	No	Non-Travel	Research & Development	2	
4	32	No	Travel_Rarely	Research & Development	10	

	Education	EducationField	EmployeeCount	EmployeeID	Gender	…	\
0	2	Life Sciences	1	1	Female	…	
1	1	Life Sciences	1	2	Female	…	
2	4	Other	1	3	Male	…	
3	5	Life Sciences	1	4	Male	…	
4	1	Medical	1	5	Male	…	

	NumCompaniesWorked	Over18	PercentSalaryHike	StandardHours	\
0	1.0	Y	11	8	
1	0.0	Y	23	8	
2	1.0	Y	15	8	
3	3.0	Y	11	8	
4	4.0	Y	12	8	

	StockOptionLevel	TotalWorkingYears	TrainingTimesLastYear	YearsAtCompany	\
0	0	1.0	6	1	
1	1	6.0	3	5	
2	3	5.0	2	5	
3	3	13.0	5	8	
4	2	9.0	2	6	

	YearsSinceLastPromotion	YearsWithCurrManager
0	0	0
1	1	4
2	0	3
3	7	5
4	0	4

[5 rows x 24 columns]

[143]:
```
df_business = pd.get_dummies(df['BusinessTravel'], prefix='BusinessTravel').
↪astype(int)
df = pd.concat([df.drop('BusinessTravel', axis=1), df_business], axis=1)
```

[144]:
```
df_department = pd.get_dummies(df['Department'], prefix='Department').
↪astype(int)
df = pd.concat([df.drop('Department', axis=1), df_department], axis=1)
```

[145]:
```
df_education = pd.get_dummies(df['EducationField'], prefix='EducationField').
↪astype(int)
df = pd.concat([df.drop('EducationField', axis=1), df_education], axis=1)
```

[146]:
```
df_gender = pd.get_dummies(df['Gender'], prefix='Gender').astype(int)
df = pd.concat([df.drop('Gender', axis=1), df_gender], axis=1)
```

[147]:
```
df_job_level = pd.get_dummies(df['JobLevel'], prefix='JobLevel').astype(int)
df = pd.concat([df.drop('JobLevel', axis=1), df_job_level], axis=1)
```

[148]:
```
df_job_role = pd.get_dummies(df['JobRole'], prefix='JobRole').astype(int)
df = pd.concat([df.drop('JobRole', axis=1), df_job_role], axis=1)
```

[149]:
```
df_marital = pd.get_dummies(df['MaritalStatus'], prefix='MaritalStatus').
↪astype(int)
df = pd.concat([df.drop('MaritalStatus', axis=1), df_marital], axis=1)
```

[150]:
```
df['Over18'] = [1 if value == 'Y' else 0 for value in df['Over18']]
```

[151]:
```
df.head()
```

[151]:

	Age	Attrition	DistanceFromHome	Education	EmployeeCount	EmployeeID	\
0	51	No	6	2	1	1	
1	31	Yes	10	1	1	2	
2	32	No	17	4	1	3	
3	38	No	2	5	1	4	
4	32	No	10	1	1	5	

	MonthlyIncome	NumCompaniesWorked	Over18	PercentSalaryHike	…	\
0	131160	1.0	1	11	…	
1	41890	0.0	1	23	…	
2	193280	1.0	1	15	…	
3	83210	3.0	1	11	…	
4	23420	4.0	1	12	…	

	JobRole_Laboratory Technician	JobRole_Manager	\
0	0	0	
1	0	0	
2	0	0	
3	0	0	
4	0	0	

	JobRole_Manufacturing Director	JobRole_Research Director	\
0	0	0	
1	0	0	
2	0	0	
3	0	0	

```
4                                0                          0

   JobRole_Research Scientist  JobRole_Sales Executive  \
0                           0                        0
1                           1                        0
2                           0                        1
3                           0                        0
4                           0                        1

   JobRole_Sales Representative  MaritalStatus_Divorced  \
0                             0                       0
1                             0                       0
2                             0                       0
3                             0                       0
4                             0                       0

   MaritalStatus_Married  MaritalStatus_Single
0                      1                     0
1                      0                     1
2                      1                     0
3                      1                     0
4                      0                     1

[5 rows x 48 columns]
```

[152]:
```
X,y = df.drop('Attrition',axis=1),df['Attrition']
```

[153]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[154]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[155]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

model = LogisticRegression()

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print(confusion_matrix(y_test, y_pred))
```

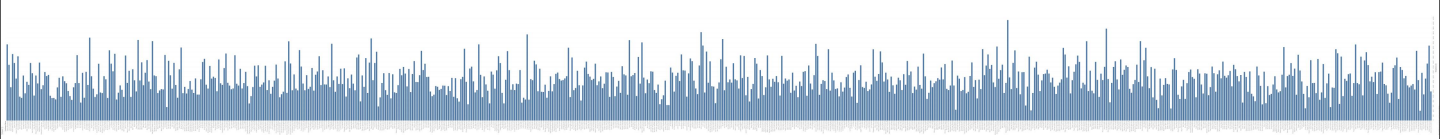
```
print(classification_report(y_test, y_pred))
print(model.score(X_test,y_test))
```

```
[[722  19]
 [120  21]]
              precision    recall  f1-score   support

          No       0.86      0.97      0.91       741
         Yes       0.53      0.15      0.23       141

    accuracy                           0.84       882
   macro avg       0.69      0.56      0.57       882
weighted avg       0.80      0.84      0.80       882

0.8424036281179138
```

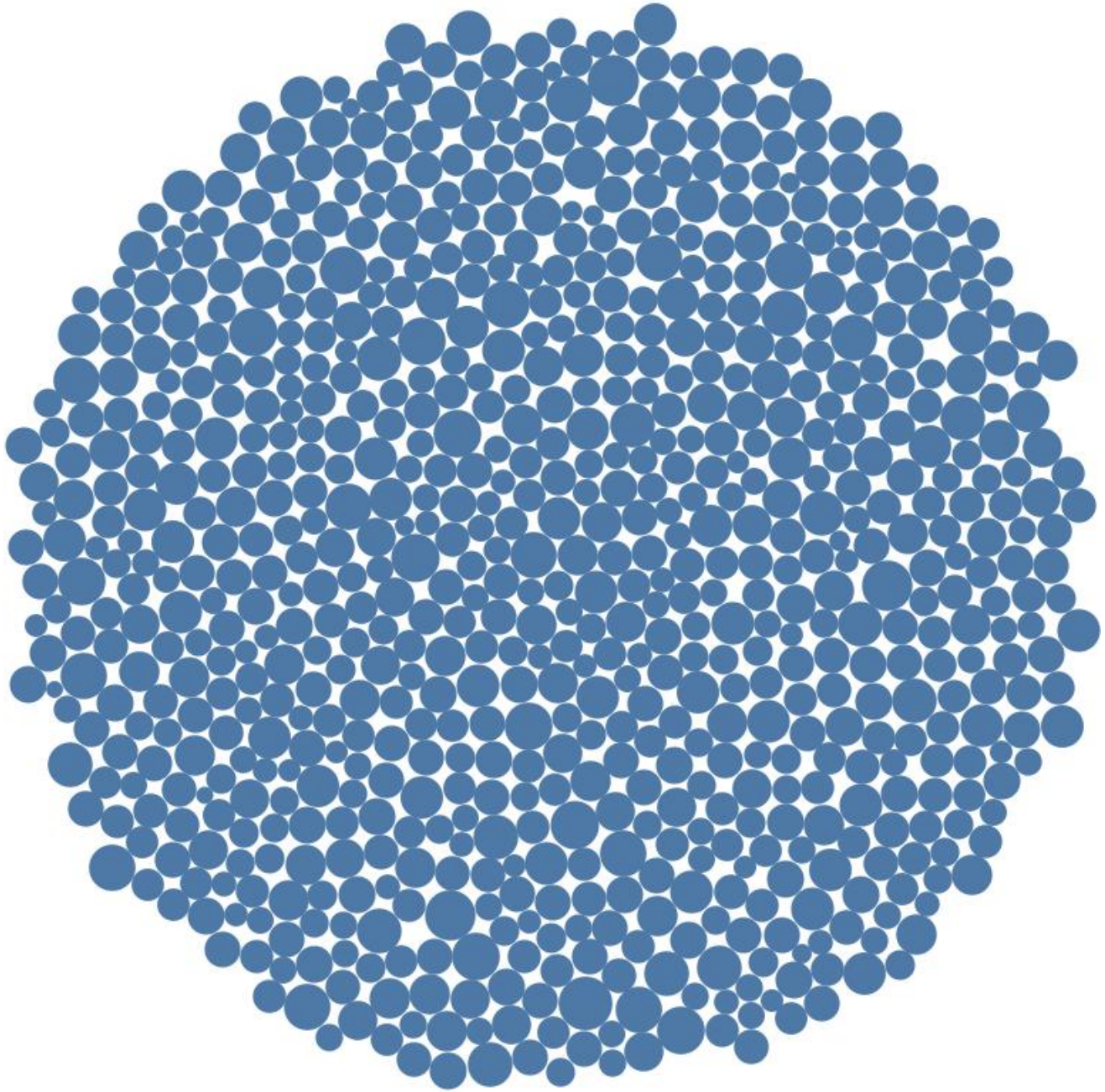
[ ]:

Book1

File created on: 01-10-2024 11:00:59

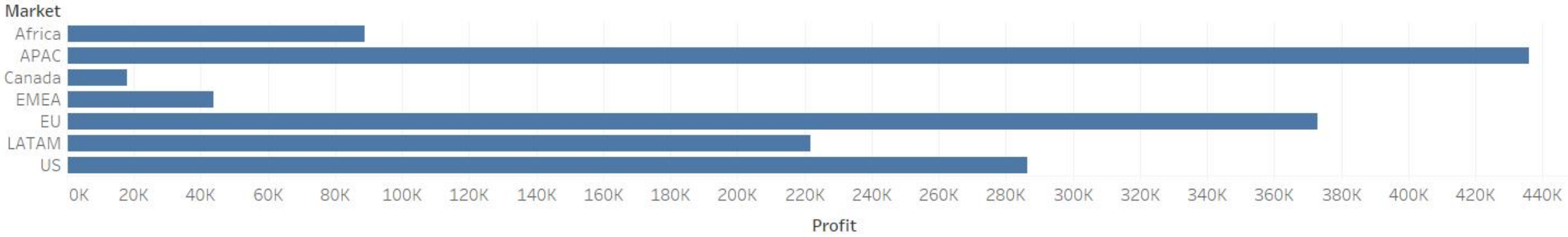


Sheet 3



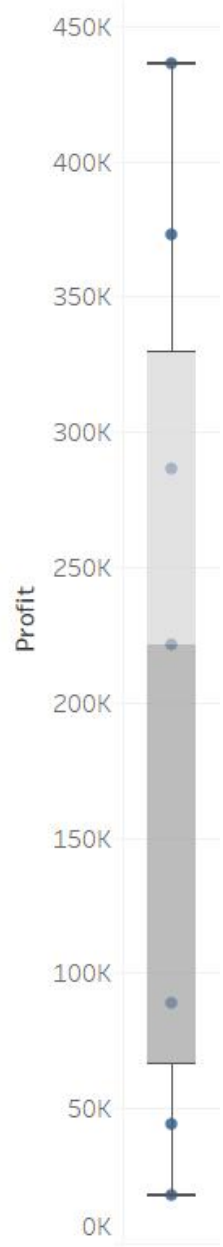
Customer Name. Size shows sum of Shipping Cost. The marks are labeled by Customer Name.

Sheet 4



Sum of Profit for each Market.

Sheet 5



Sum of Profit.
Details are shown
for Market.

Sheet 6

Market (color) and sum of Profit (size).

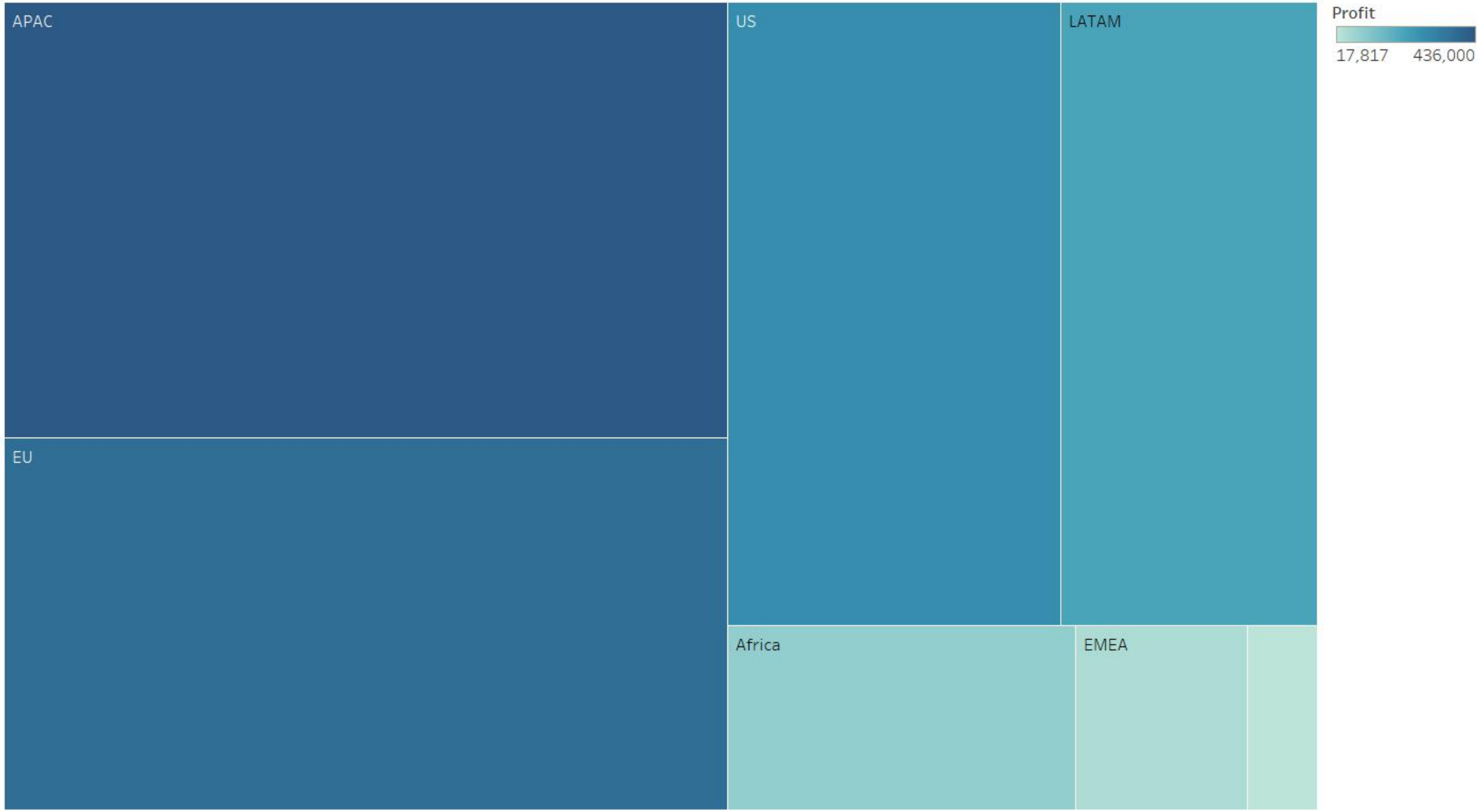
Profit

1,467,457

Market

- Africa
- APAC
- Canada
- EMEA
- EU
- LATAM
- US

Sheet 7



Market. Color shows sum of Profit. Size shows sum of Profit. The marks are labeled by Market.

Sheet 8

Market (color) and sum of Sales (size).

Sales

12,642,502

Market

- Africa
- APAC
- Canada
- EMEA
- EU
- LATAM
- US

LAB2

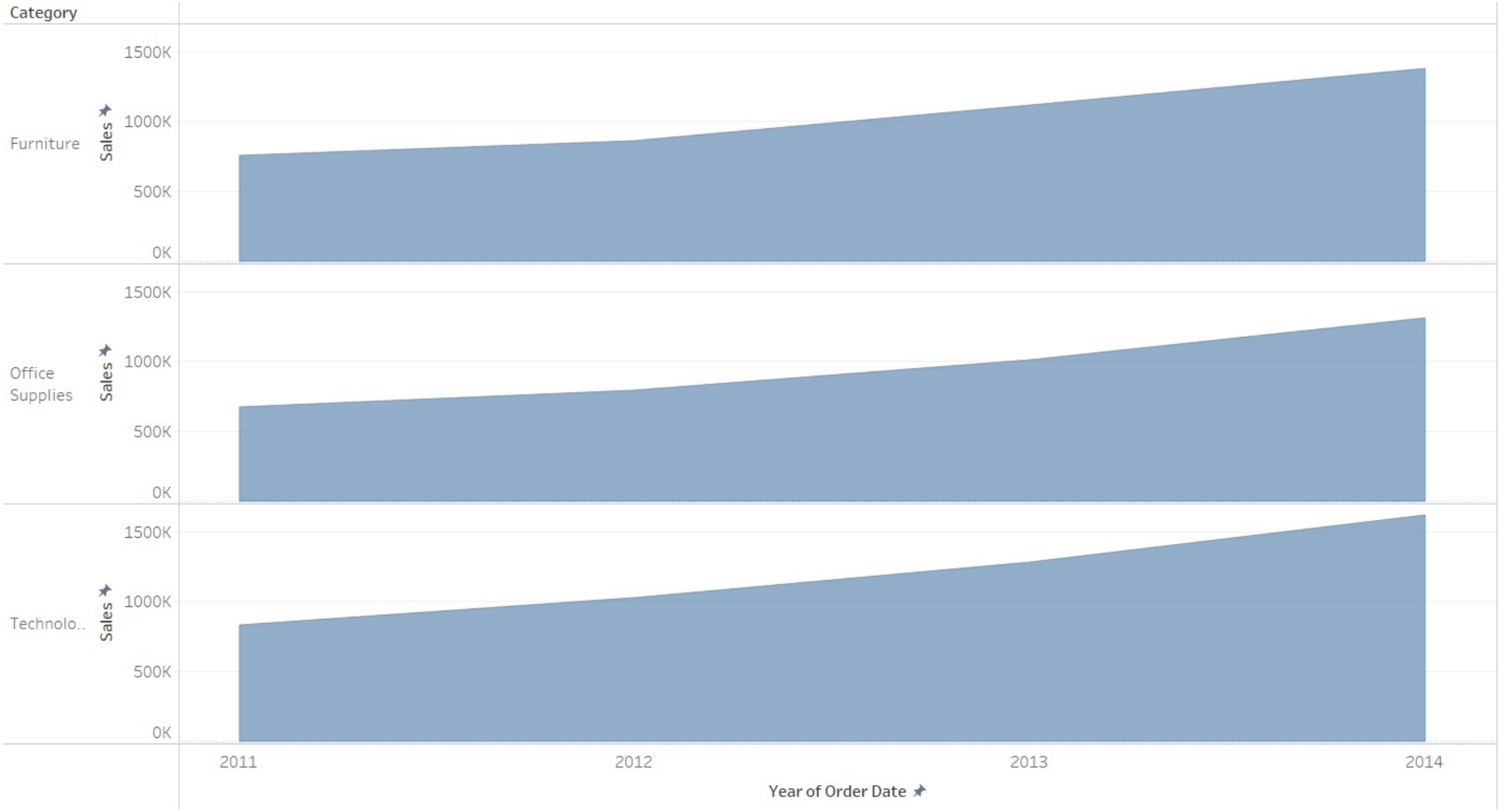
File created on: 13-10-2024 16:11:43

Key Perfomance Indicators

Sales	Profit	Discount	Profit per Order	Quantity	Sales per Custom..	Shipping Cost
12,642,502	1,467,457	7,330	59	178,312	15,903	1,352,816

Sales, Profit, Discount, Profit per Order, Quantity, Sales per Customer and Shipping Cost.

Area Charts



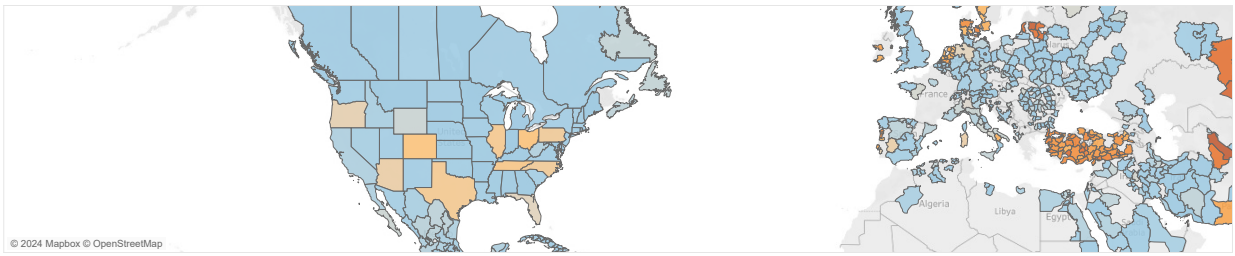
The plot of sum of Sales for Order Date Year broken down by Category.

Key Perfomance Indicators

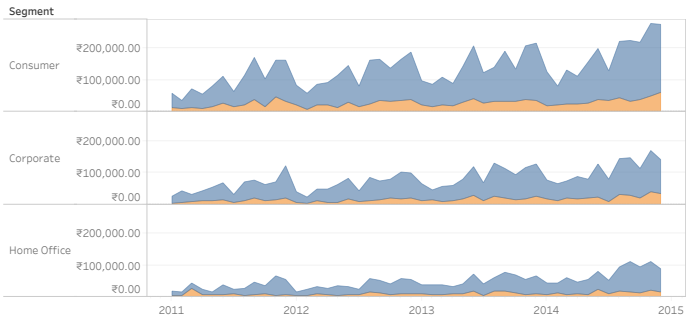
Profit Ratio
-2.244 0.500

Sales	Profit	Profit Ratio	Profit Per Order	Sales per Customer	Avg. Discount	Quantity
₹12,642,501.91	₹1,467,457.29	11.6%	59	₹15,902.52	14.29%	178,312

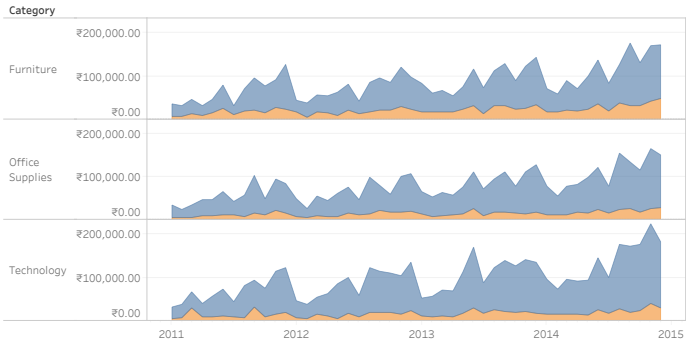
Profit Ratio by Geography



Sales by Segment



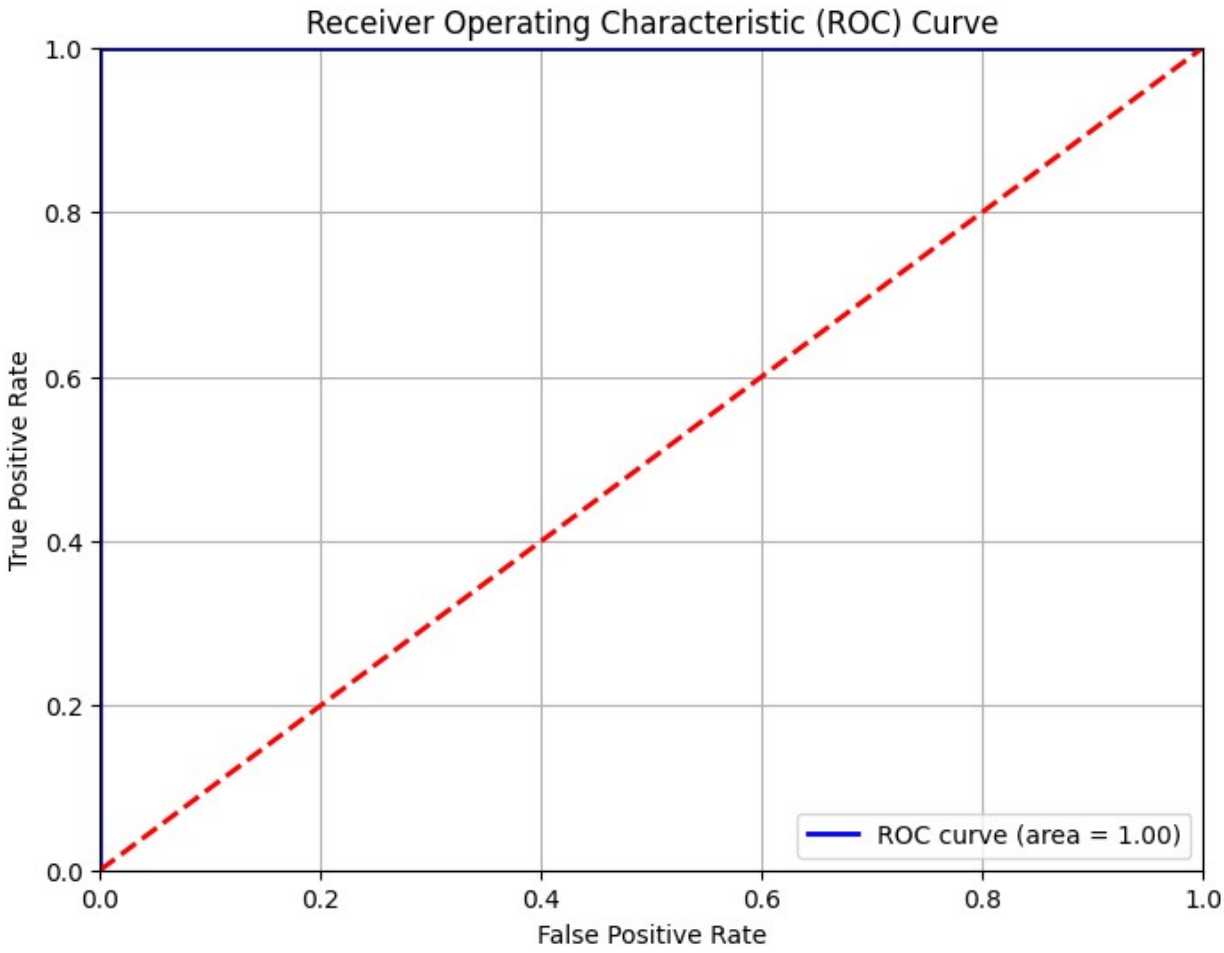
Sales by Category



```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, auc

iris = load_iris()
X = iris.data[:, :2]  # Select sepal length and sepal width
y = iris.target

binary_mask = (y == 0) | (y == 1)  # Only select classes 0 and 1
X_binary = X[binary_mask]
y_binary = y[binary_mask]

X_train, X_test, y_train, y_test = train_test_split(X_binary,
y_binary, test_size=0.3, random_state=42)

# Step 4: Train a logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)
```

```
LogisticRegression()
```

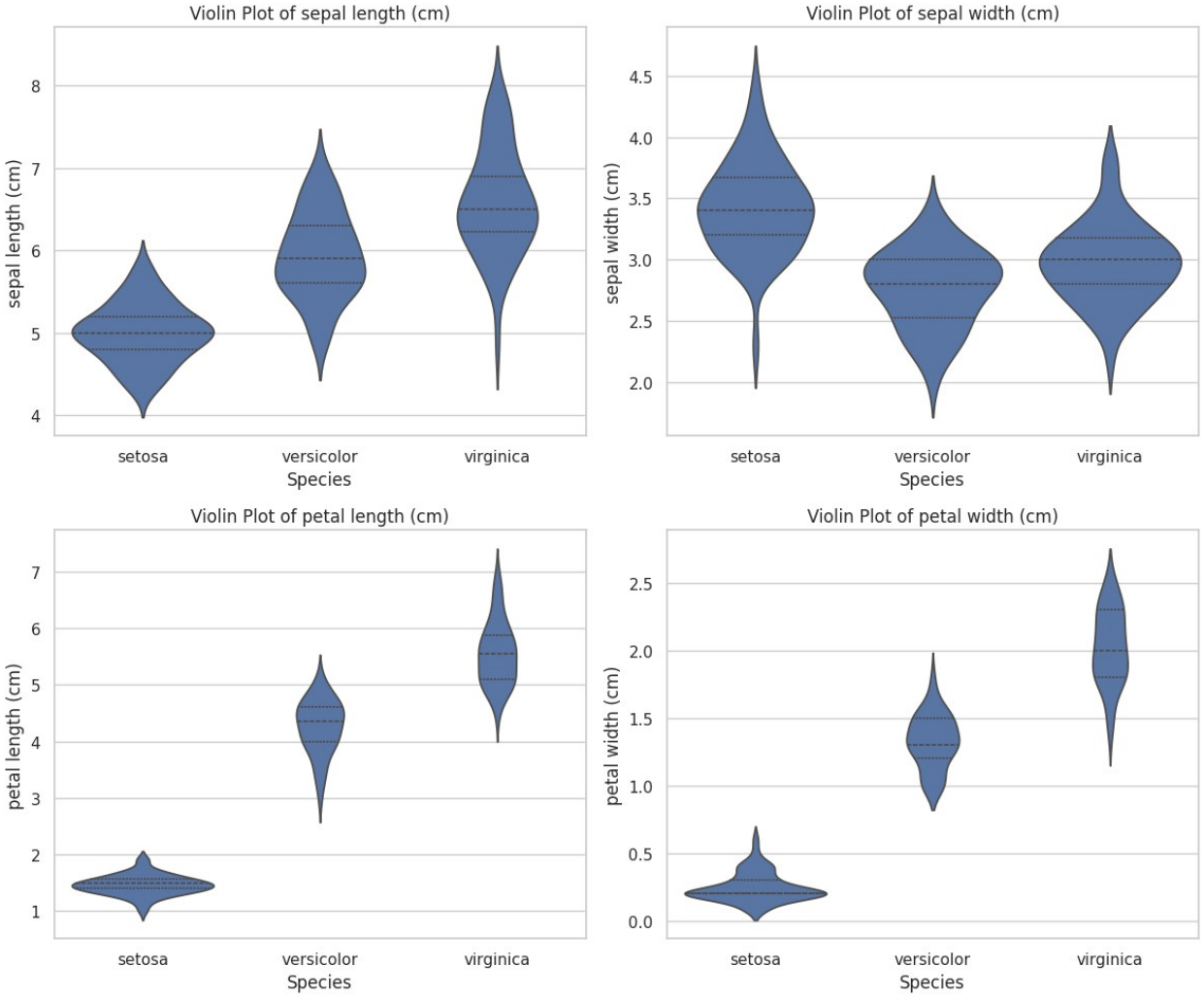
```
y_score = model.predict_proba(X_test)[:, 1]

fpr, tpr, thresholds = roc_curve(y_test, y_score)
roc_auc = auc(fpr, tpr)

# Step 7: Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (area =
{:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--')  #
Diagonal line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.grid()
plt.show()
```


Receiver Operating Characteristic (ROC) Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 1.00)

```
roc_auc
```

```
1.0
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets

iris = datasets.load_iris()
X = iris.data  # Features
y = iris.target  # Target labels
species = iris.target_names[y]

iris_df = pd.DataFrame(X, columns=iris.feature_names)
iris_df['species'] = species

sns.set(style="whitegrid")

# Create a figure with subplots
plt.figure(figsize=(12, 10))

# Violin plot for each feature
features = iris.feature_names

for i, feature in enumerate(features):
    plt.subplot(2, 2, i + 1)  # Create a subplot for each feature
    sns.violinplot(x='species', y=feature, data=iris_df,
inner='quartile')
    plt.title(f'Violin Plot of {feature}')
    plt.xlabel('Species')
    plt.ylabel(feature)

# Adjust layout
plt.tight_layout()
plt.show()
```


Violin Plot of sepal length (cm)
sepal length (cm)
8
7
6
5
4
setosa
versicolor
virginica
Species
Violin Plot of sepal width (cm)
sepal width (cm)
4.5
4.0
3.5
3.0
2.5
2.0
setosa
versicolor
virginica
Species
Violin Plot of petal length (cm)
petal length (cm)
7
6
5
4
3
2
1
setosa
versicolor
virginica
Species
Violin Plot of petal width (cm)
petal width (cm)
2.5
2.0
1.5
1.0
0.5
0.0
setosa
versicolor
virginica
Species