In [545…

```python
#Homework 3 complete
#Took me approximately 5 days to figure this out!
import matplotlib.pyplot as plot
import pandas as pd
import seaborn as sns
import numpy as np
from sklearn import metrics
from sklearn.datasets import load_breast_cancer
from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
```

In [546…

```python
# Problem 1
# loads imported cancer dataset
breast = load_breast_cancer()
```

In [547…

```python
breast_data = breast.data
breast_data.shape
```

Out[547…

```
(569, 30)
```

In [548…

```python
#Visualizes dataset
breast_input = pd.DataFrame(breast_data)
breast_input.head()
```

Out[548…

	0	1	2	3	4	5	6	7	8	9	...	20	21	22	23	24	25	26
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	184.60	2019.0	0.1622	0.6656	0.7119
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	158.80	1956.0	0.1238	0.1866	0.2416
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	152.50	1709.0	0.1444	0.4245	0.4504
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	98.87	567.7	0.2098	0.8663	0.6869
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	152.20	1575.0	0.1374	0.2050	0.4000

5 rows × 30 columns

In [571…

```python
# Generates dependent variable
breast_labels = breast.target
breast_labels.shape
labels = np.reshape(breast_labels, (569, 1))
```

In [572…

```python
# Adds labels to the dataset
dataset = pd.DataFrame(np.concatenate([breast_data, labels], axis=1))
features = breast.feature_names
features
```

Out[572…

```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

In [573…

```python
#Generates features and labels of dataset
features_labels = np.append(features, 'label')
```

In [574…

```python
#Generates labels
dataset.columns = features_labels
dataset.head()
```

Out[574…

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	98.87	567.7
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0

5 rows × 31 columns

In [575…

```python
n = breast_data.shape[1]
x = dataset.values[:, :n]
y = dataset.values[:, n]
```

In [576…

```python
 # Processes cost function for linear regression
# Computes value of cost function J for each iteration
 # This fuunction takes into account on each iteration
def gradient_descent(x, y, theta, alpha, iterations):
    cost_history = np.zeros(iterations)
    acc_history = np.zeros(iterations)
    m = x.shape[0]
    x = np.hstack((np.ones((m, 1)), x.reshape(m, n)))

    for i in range(iterations):
        predictions = np.divide(1, 1 + np.exp(-1 * x.dot(theta)))
        theta = theta - (alpha / m) * x.transpose().dot(predictions - y)

        log1 = np.multiply(y, np.log(predictions))
        log2 = np.multiply(1 - y, np.log(1 - predictions))
        cost_history[i] = -1 / m * np.sum(log1 + log2)

        # Computes accuracy for each iteration
        acc_history[i] = (m - np.sum(np.abs(np.round(predictions) - y))) / m

    return theta, cost_history, acc_history
```

In [577…

```python
#Implements Min max scaling
mins = MinMaxScaler()
x = mins.fit_transform(x)
```

In [578…

```python
#Standardization implementation
strds = StandardScaler()
x = strds.fit_transform(x)
```

In [579…

```python
#80% and 20% split used for evaluation
np.random.seed(0)
x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=0.8,
                                                    test_size=0.2,
                                                    random_state=np.random)
```

In [580…

```python
# Initializes theta, the learning rate alpha, and the number of iterations
theta = np.zeros(x.shape[1] + 1)
alpha = 0.1
it = 1500
```

In [581…

```python
theta, train_cost, train_acc = gradient_descent(x_train, y_train, theta, alpha, it)
test_cost, test_acc = gradient_descent(x_test, y_test, theta, alpha, it)[1:]
```

In [560…

```python
# Evaluates model using accuracy evaluation metric
print('Accuracy of model:', test_acc[0])
```

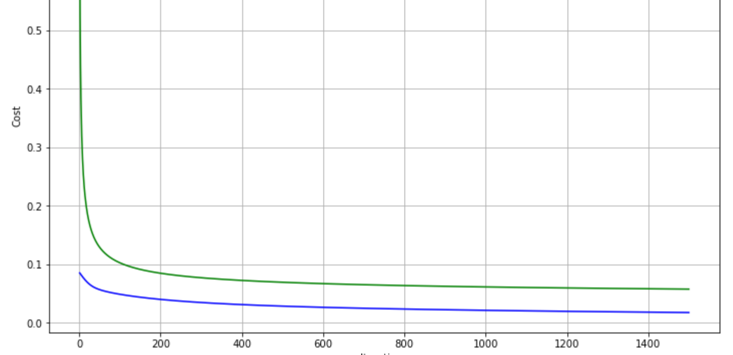
```
Accuracy of model: 0.9649122807017544
```

In [582…

```python
# Plots training and test cost history for the logistic regression
plot.figure(1)
plot.plot(np.linspace(1, it, it), train_cost, color='green',
         label='Training cost')
plot.plot(np.linspace(1, it, it), test_cost, color='blue',
         label='Test cost')
plot.rcParams['figure.figsize'] = (12, 8)
plot.grid()
plot.xlabel('Iterations')
plot.ylabel('Cost')
plot.title('Graph of Test and Train input variable for logistic regression')
plot.legend()
```

Out[582…

```
<matplotlib.legend.Legend at 0x21d15a7f0d0>
```



In [562…

```python
#Performs logistic regression for training
classifier = LogisticRegression(random_state = 0);
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)
y_pred[0:9]
```

Out[562…

```
array([0., 1., 1., 1., 1., 1., 1., 1., 1.])
```

In [563…

```python
#Creates confusion matrix
cnf_matrix = confusion_matrix(y_test, y_pred)
cnf_matrix
```

Out[563…

```
array([[45,  2],
       [ 2, 65]], dtype=int64)
```

In [564…

```python
#Prints Precision accuracy and recall values
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
```

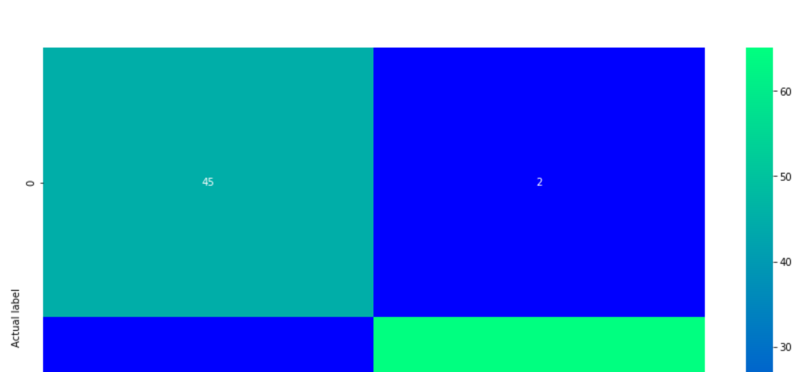
```
Accuracy: 0.9649122807017544
Precision: 0.9701492537313433
Recall: 0.9701492537313433
```

In [569…

```python
class_names = [0,1]
fig, axis = plt.subplots()
tick_marks = np.arange(len(class_names))
plot.xticks(tick_marks, class_names)
plot.yticks(tick_marks, class_names)
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="winter" ,fmt='g')
ax.xaxis.set_label_position("top")
plot.tight_layout()
plot.title('Confusion matrix', y=1.1)
plot.ylabel('Actual label')
plot.xlabel('Predicted label')
```

Out[569…

```
Text(0.5, 51.0, 'Predicted label')
```

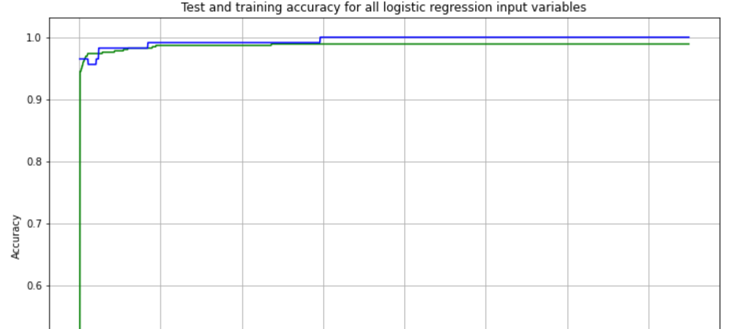


In [566…

```python
# This function plots the test and training accuracy of all input variables
plot.figure(2)
plot.plot(np.linspace(1, it, it), train_acc, color='green',
         label='Training accuracy')
plot.plot(np.linspace(1, it, it), test_acc, color='blue',
         label='Test accuracy')
plot.rcParams['figure.figsize'] = (12, 8)
plot.grid()
plot.xlabel('Iterations')
plot.ylabel('Accuracy')
plot.title('Test and training accuracy for all logistic regression input variables')
plot.legend()
```

Out[566…

```
<matplotlib.legend.Legend at 0x21d15918430>
```



In [567…

```python
#Number 2
#Attempts PCA feature extrection
pca = PCA()
pcs = pca.fit_transform(x)

#This function splits into two variables dependent and independant
x = dataset.iloc[:, 0:29].values
y = dataset.iloc[:, 30].values

# Standardizing the features
x = StandardScaler().fit_transform(x)
```

In [568…

```python
# Performs 80% and 20% split of the labelled data into training and test sets
x_train_p, x_test_p, y_train_p, y_test_p = train_test_split(pcs, y, train_size=0.8,
                                                            test_size=0.2,
                                                            random_state=np.random)
```

In [514…

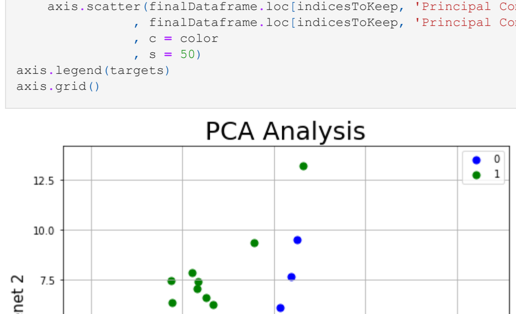
```python
# Initializes evaluation metrics  PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['Principal Component 1', 'Principal Component 2'])
```

In [515…

```python
finalDataframe = pd.concat([principalDf, dataset[['label']]], axis = 1)
```

In [516…

```python
#Code to plot data
fig = plot.figure(figsize = (8,8))
axis = fig.add_subplot(1,1,1)
axis.set_xlabel('Principal Component 1', fontsize = 15)
axis.set_ylabel('Principal Componenet 2', fontsize = 15)
axis.set_title('PCA Analysis', fontsize = 25)
targets = [0, 1]
colors = ['b', 'g']
for target, color in zip(targets,colors):
    indicesToKeep = finalDataframe['label'] == target
    axis.scatter(finalDataframe.loc[indicesToKeep, 'Principal Component 1']
               , finalDataframe.loc[indicesToKeep, 'Principal Component 2']
               , c = color
               , s = 50)
axis.legend(targets)
axis.grid()
```



In [517…

```python
# Iteratively trains and evaluates model for principal components
 # Performs logistic regression by instantiating LogisticRegression object
accur = 0
optimalk = 0
for i in range(k):

    regress = LogisticRegression()
    regress.fit(x_train_p[:, :i + 1], y_train_p)
    y_pred = regress.predict(x_test_p[:, :i + 1])

    accuracy[i] = metrics.accuracy_score(y_test_p, y_pred)
    precision[i] = metrics.precision_score(y_test_p, y_pred)
    recall[i] = metrics.recall_score(y_test_p, y_pred)

    if accuracy[i] > optimalk:
        accur = accuracy[i]
        optimalk = i + 1
```

In [518…

```python
# Displays optimal K and corresponding accuracy, precision, and recall
print('Optimal value of K:', optimalk)
print('Accuracy:', optimalk)
print('Precision:', precision[optimalk - 1])
print('Recall:', recall[optimalk - 1])
```

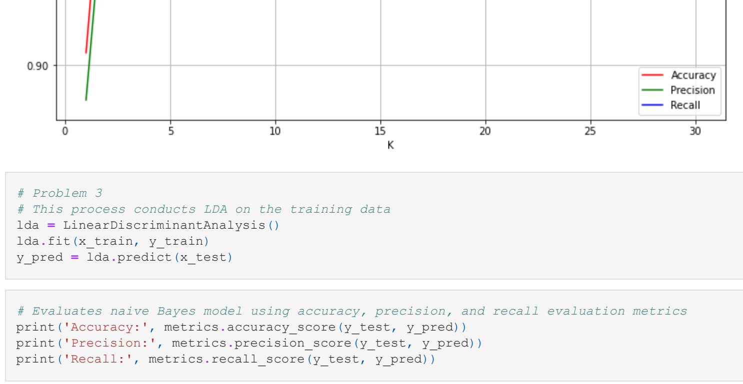
```
Optimal value of K: 1
Accuracy: 1
Precision: 0.8904109589041096
Recall: 0.9558823529411765
```

In [519…

```python
# Plots accuracy, precision, and recall for varying numbers of principal components
plot.figure(3)
plot.plot(np.linspace(1, k, k), accuracy, color='red',
         label='Accuracy')
plot.plot(np.linspace(1, k, k), precision, color='green',
         label='Precision')
plot.plot(np.linspace(1, k, k), recall, color='blue',
         label='Recall')
plot.rcParams['figure.figsize'] = (10, 6)
plot.grid()
plot.xlabel('K')
plot.ylabel('value')
plot.title('K principal coponents')
plot.legend()
```

Out[519…

```
<matplotlib.legend.Legend at 0x21d15373d60>
```



In [520…

```python
# Problem 3
# This process conducts LDA on the training data
lda = LinearDiscriminantAnalysis()
lda.fit(x_train, y_train)
y_pred = lda.predict(x_test)
```

In [521…

```python
# Evaluates naive Bayes model using accuracy, precision, and recall evaluation metrics
print('Accuracy:', metrics.accuracy_score(y_test, y_pred))
print('Precision:', metrics.precision_score(y_test, y_pred))
print('Recall:', metrics.recall_score(y_test, y_pred))
```

```
Accuracy: 0.9649122807017544
Precision: 0.9436619718309859
Recall: 1.0
```

In [522…

```python
# Problem 4
# Transforms input data using linear discriminant function from LDA
lds = lda.fit_transform(x, y)
```

In [523…

```python
# Performs 80% and 20% split of the labelled data into training and test sets
x_trainl, x_testl, y_trainl, y_testl = train_test_split(lds, y, train_size=0.8,
                                                        test_size=0.2,
                                                        random_state=np.random)
```

In [524…

```python
# Performs logistic regression by instantiating LogisticRegression object
regres = LogisticRegression()
regres.fit(x_trainl, y_trainl)
y_pred = regres.predict(x_testl)
```

In [525…

```python
print('Accuracy:', metrics.accuracy_score(y_testl, y_pred))
print('Precision:', metrics.precision_score(y_testl, y_pred))
print('Recall:', metrics.recall_score(y_testl, y_pred))
plt.show()
```

```
Accuracy: 0.9736842105263158
Precision: 0.9878048780487805
Recall: 0.9759036144578314
```

In [ ]:

In [ ]: