In [1]:

```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import KFold
```

Problem a

In [2]:

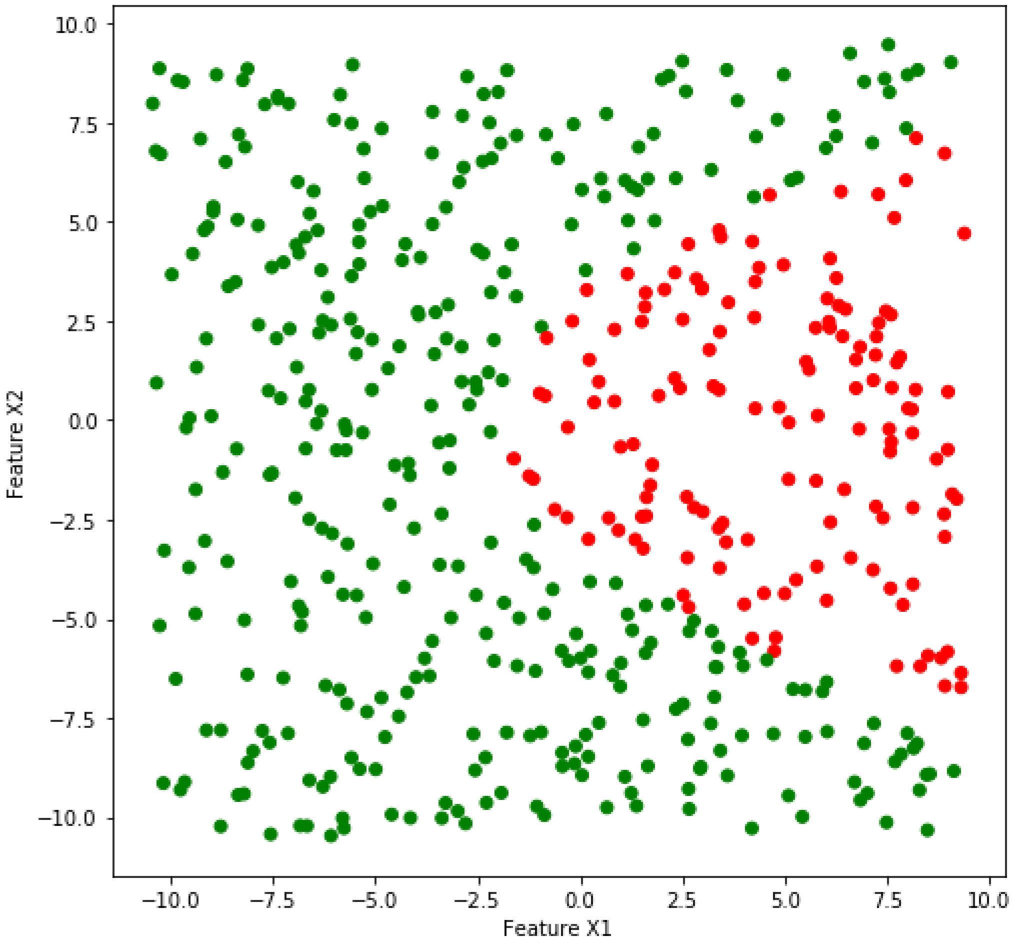
```
X1 = np.random.uniform(low=-10, high=10, size=500) - 0.5
X2 = np.random.uniform(low=-10, high=10, size=500) - 0.5
y = 1*((2 + X1 - 0.2*X2**2) > 0)
```

Problem b

In [3]:
```
# Define colors for the class labels: RED for 1, GREEN for 0
colors = ['green','red']

fig = plt.figure(figsize=(8,8))
plt.scatter(X1, X2, c=y, cmap=matplotlib.colors.ListedColormap(colors))
plt.xlabel('Feature X1')
plt.ylabel('Feature X2')
```

Out[3]:
```
Text(0,0.5,'Feature X2')
```



Problem c

In [4]:

```
# Create design matrix with predictors
predictors = {
             'X1': X1,
             'X2': X2
             }
data = pd.DataFrame(predictors)

# Create array of column names
column_names = list(data.columns)
column_names.insert(0, 'Intercept')

# Split data into training and testing datasets
X_train, X_test, y_train, y_test = train_test_split(data, y, test_size=0.20)

# Create Logistic Classifier model and fit to training data
Logit_classifier = LogisticRegression()
Logit_classifier.fit(X_train, y_train)

values = []
values.append(Logit_classifier.intercept_[0])
for coeff in (Logit_classifier.coef_[0]):
    values.append(coeff)

# Create table of model prediction results
results = {
        'Attributes': column_names,
        'Coefficient Beta_i': values,
       }
ModelResults = pd.DataFrame(results)
ModelResults.round(4) # Round values in table to 4-decimal places
```

Out[4]:

	Attributes	Coefficient Beta_i
0	Intercept	-1.3785
1	X1	0.3589
2	X2	0.0729

Problem d

In [5]:

```
# Make predictions on the training data
y_train_pred = Logit_classifier.predict(X_train)
# Make the classifications 1 or 0 depending on probability is greater than or equal to 0.5
y_train_pred = 1*(y_train_pred >= 0.5)
```

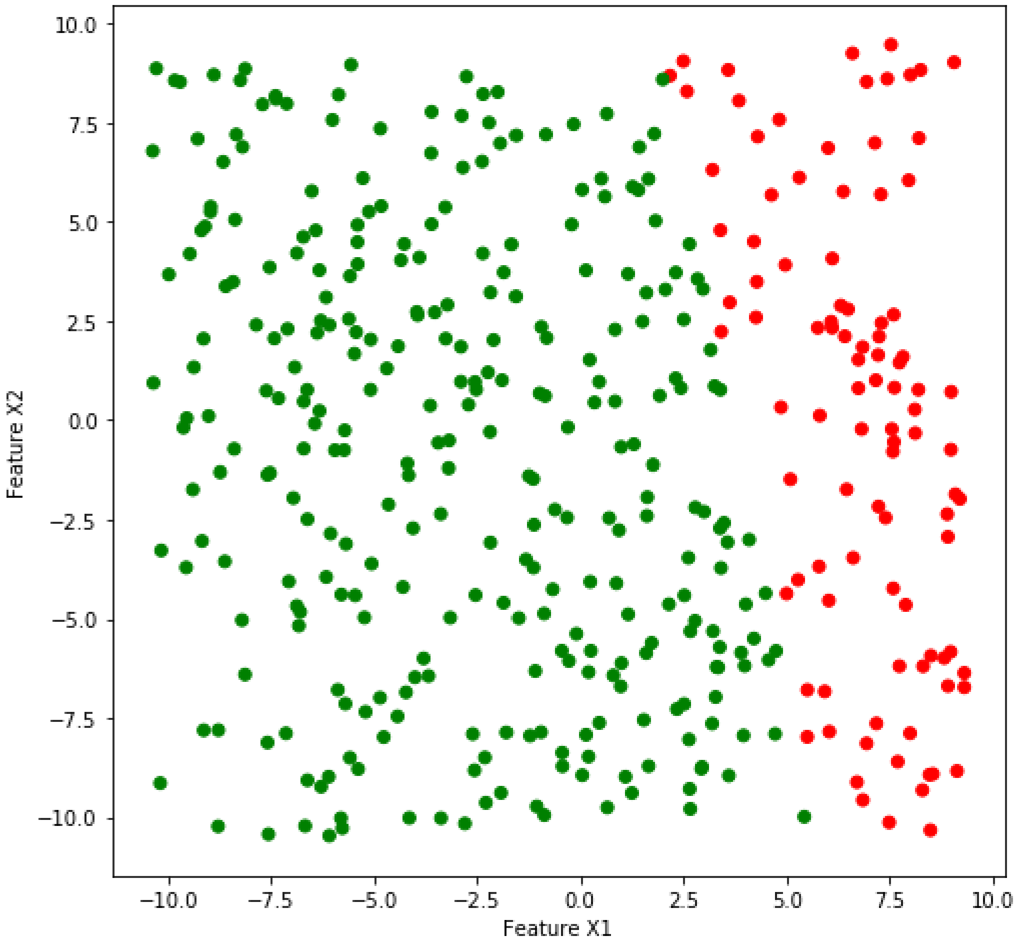
In [6]:

```
# Define colors for the class labels: RED for 1, GREEN for 0
colors = ['green','red']

fig = plt.figure(figsize=(8,8))
plt.scatter(X_train['X1'], X_train['X2'], c=y_train_pred, cmap=matplotlib.colors.ListedColormap(colors))
plt.xlabel('Feature X1')
plt.ylabel('Feature X2')
```

Out[6]:

```
Text(0,0.5,'Feature X2')
```



Problem e

In [7]:
```
# Create design matrix with predictors
predictors = {
            'X1': X1,
            'X2^2': X2**2,
            'X2': X2
            }
data = pd.DataFrame(predictors)

# Create array of column names
column_names = list(data.columns)
column_names.remove('X2')
column_names.insert(0, 'Intercept')

# Split data into training and testing datasets
X_train, X_test, y_train, y_test = train_test_split(data, y, test_size=0.20)

# Create Logistic Classifier model and fit to training data
Logit_classifier = LogisticRegression()
Logit_classifier.fit(X_train[['X1','X2^2']], y_train)

values = []
values.append(Logit_classifier.intercept_[0])
for coeff in (Logit_classifier.coef_[0]):
    values.append(coeff)

# Create table of model prediction results
results = {
        'Attributes': column_names,
        'Coefficient Beta_i': values,
       }
ModelResults = pd.DataFrame(results)
ModelResults.round(4) # Round values in table to 4-decimal places
```

Out[7]:

	Attributes	Coefficient Beta_i
0	Intercept	2.8874
1	X1	1.8874
2	X2^2	-0.3596

Problem f

In [8]:

```
# Make predictions on the training data
y_train_pred = Logit_classifier.predict(X_train[['X1','X2^2']])
# Make the classifications 1 or 0 depending on probability is greater than or equal to 0.5
y_train_pred = 1*(y_train_pred >= 0.5)
```

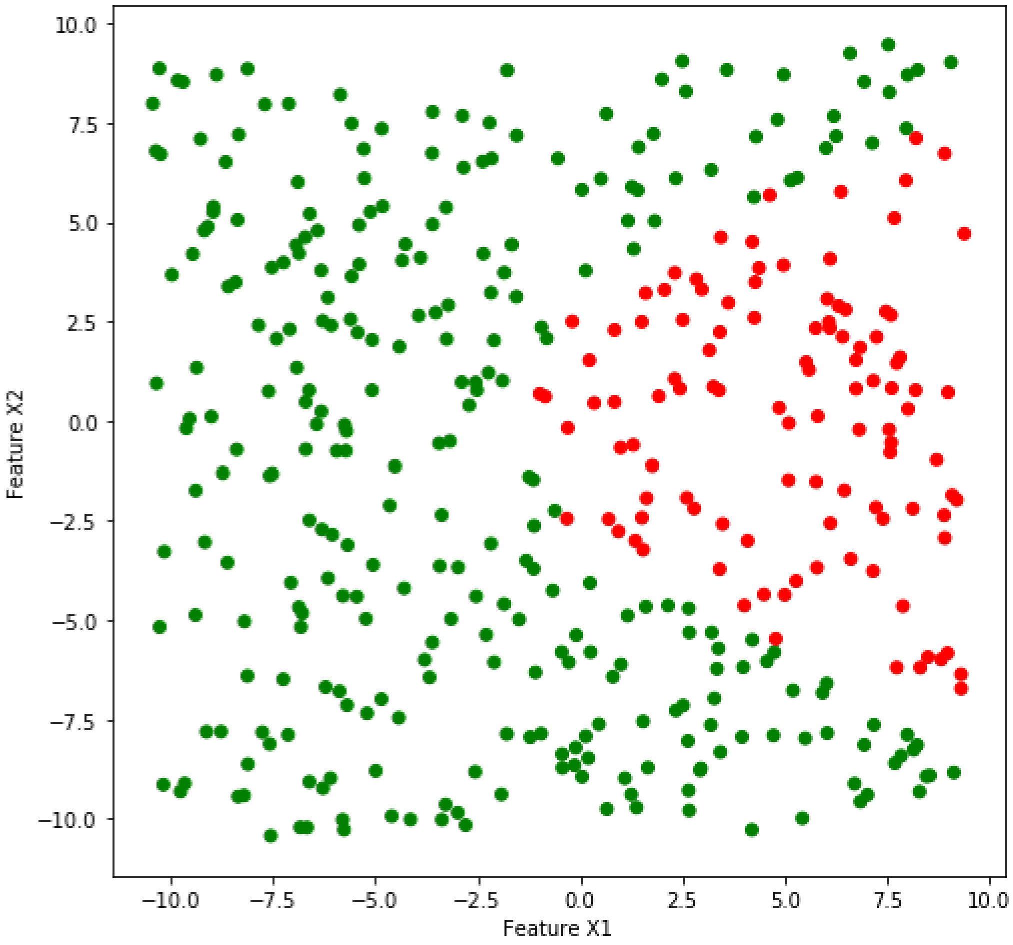
In [9]:

```
# Define colors for the class labels: RED for 1, GREEN for 0
colors = ['green','red']

fig = plt.figure(figsize=(8,8))
plt.scatter(X_train['X1'], X_train['X2'], c=y_train_pred, cmap=matplotlib.colors.ListedColormap(colors))
plt.xlabel('Feature X1')
plt.ylabel('Feature X2')
```

Out[9]:

```
Text(0,0.5,'Feature X2')
```



Problem g

In [10]:

```
# Create design matrix with predictors
predictors = {
            'X1': X1,
            'X2': X2
            }
data = pd.DataFrame(predictors)

# Create array of column names
column_names = list(data.columns)
column_names.insert(0, 'Intercept')

# Split data into training and testing datasets
X_train, X_test, y_train, y_test = train_test_split(data, y, test_size=0.20)

# Create Support Vector Classifier model and fit to training data
SV_classifier = SVC(kernel='linear')
SV_classifier.fit(X_train, y_train);

values = []
values.append(SV_classifier.intercept_[0])
for coeff in (SV_classifier.coef_[0]):
    values.append(coeff)

# Create table of model prediction results
results = {
        'Attributes': column_names,
        'Coefficient Beta_i': values,
       }
ModelResults = pd.DataFrame(results)
ModelResults.round(4) # Round values in table to 4-decimal places
```

Out[10]:

	Attributes	Coefficient Beta_i
0	Intercept	-0.8213
1	X1	0.2464
2	X2	0.0534

In [11]:

```
# Make predictions on the training data
y_train_pred = SV_classifier.predict(X_train)
# Make the classifications 1 or 0 depending on probability is greater than or equal to 0.5
y_train_pred = 1*(y_train_pred >= 0.5)
```

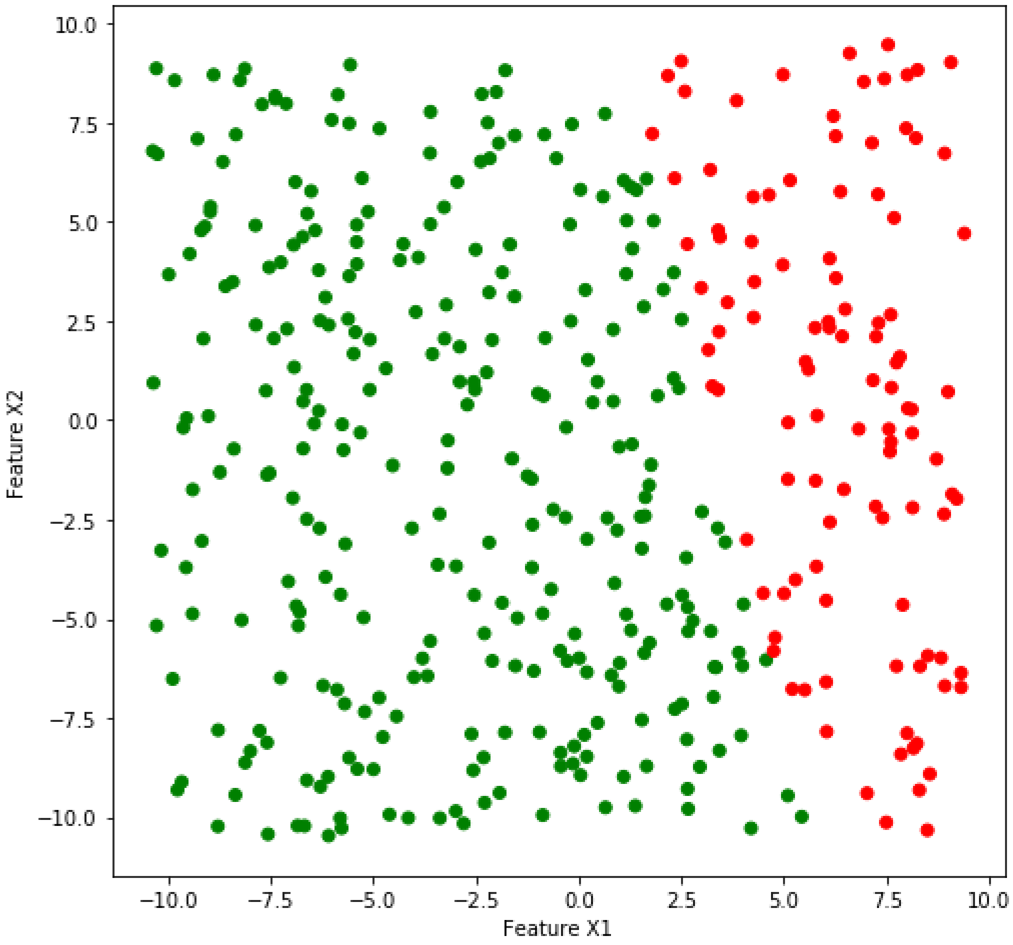
In [12]:

```
# Define colors for the class labels: RED for 1, GREEN for 0
colors = ['green','red']

fig = plt.figure(figsize=(8,8))
plt.scatter(X_train['X1'], X_train['X2'], c=y_train_pred, cmap=matplotlib.colors.ListedColormap(colors))
plt.xlabel('Feature X1')
plt.ylabel('Feature X2')
```

Out[12]:

```
Text(0,0.5,'Feature X2')
```



Problem h

In [13]:

```
# Create design matrix with predictors
predictors = {
            'X1': X1,
            'X2': X2
            }
data = pd.DataFrame(predictors)

# Create array of column names
column_names = list(data.columns)
column_names.insert(0, 'Intercept')

# Split data into training and testing datasets
X_train, X_test, y_train, y_test = train_test_split(data, y, test_size=0.20)

# Create Support Vector Classifier model and fit to training data
# SV_classifier = SVC(kernel='poly', degree=2)
SV_classifier = SVC(kernel='rbf')
SV_classifier.fit(X_train, y_train);

# values = []
# values.append(SV_classifier.intercept_[0])
# for coeff in (SV_classifier.coef_[0]):
#     values.append(coeff)

# # Create table of model prediction results
# results = {
#         'Attributes': column_names,
#         'Coefficient Beta_i': values,
#        }
# ModelResults = pd.DataFrame(results)
# ModelResults.round(4) # Round values in table to 4-decimal places
```

In [14]:

```
# Make predictions on the training data
y_train_pred = SV_classifier.predict(X_train)
# Make the classifications 1 or 0 depending on probability is greater than or equal to 0.5
y_train_pred = 1*(y_train_pred >= 0.5)
```

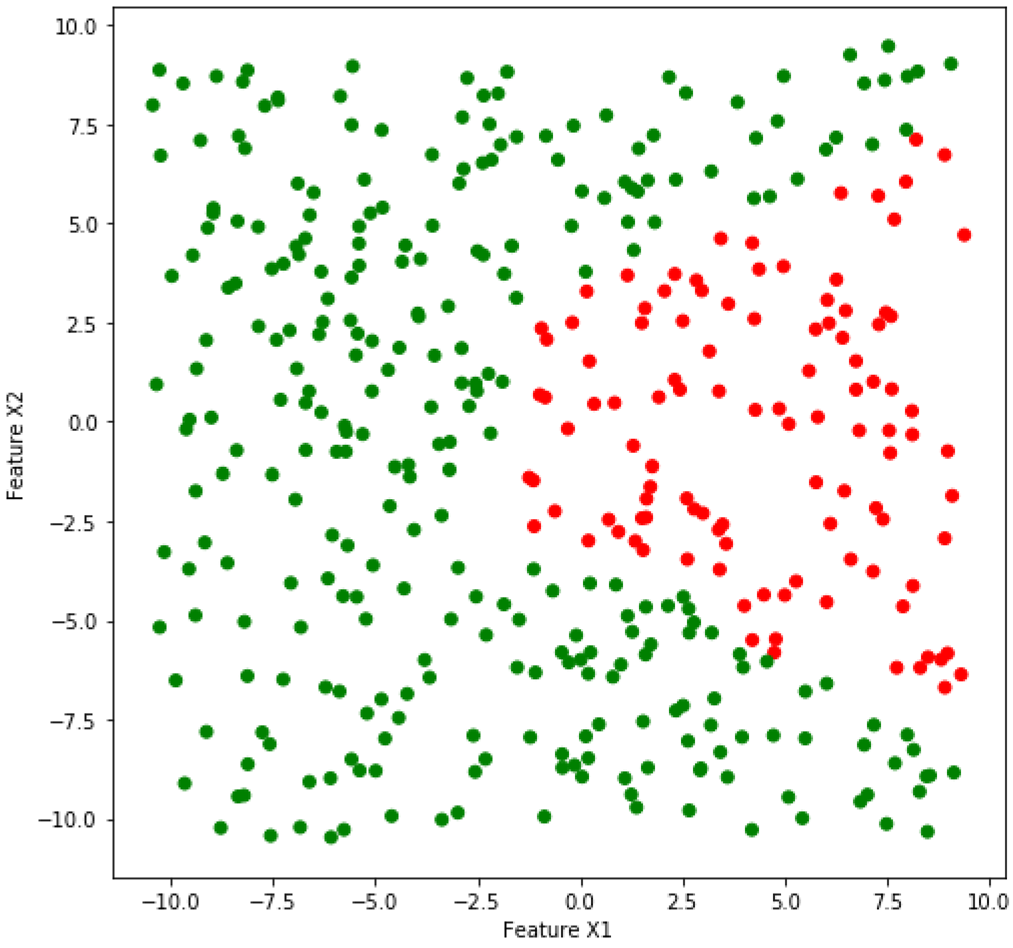
In [15]:

```
# Define colors for the class labels: RED for 1, GREEN for 0
colors = ['green','red']

fig = plt.figure(figsize=(8,8))
plt.scatter(X_train['X1'], X_train['X2'], c=y_train_pred, cmap=matplotlib.colors.ListedColormap(colors))
plt.xlabel('Feature X1')
plt.ylabel('Feature X2')
```

Out[15]:

```
Text(0,0.5,'Feature X2')
```



Problem i

In [16]:

```
# In part g), the support vector classifier with a linear kernel
# has a linear decision boundary, while in part h), the support
# vector classifier with a radial basis function kernel is able to capture
# quadratic-shaped decision boundary.
```

In [ ]: