In [1]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis, QuadraticDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, roc_curve, roc_auc_score

%matplotlib inline
plt.rcParams['figure.figsize'] = [16, 8]

# Import data
df = pd.read_csv("diabetes_binary_5050split_health_indicators_BRFSS2021.csv")
df.head()

# Split the dataframe into the output y and the input X and into train and test with a train size of 0.8
X = df.drop(['Diabetes_binary'], axis=1)
y = df['Diabetes_binary']
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, random_state=0)
```

In [2]:

```python
# Define transformations for numeric variables using ColumnTransformer
preprocessor = ColumnTransformer(transformers=[('num', StandardScaler(), X.columns)])

# Fit and transform the training data
X_train_processed = preprocessor.fit_transform(X_train)

# Transform the test data
X_test_processed = preprocessor.transform(X_test)
```

In [3]:

```python
# Linear Discriminant Analysis (LDA)
lda_model = LinearDiscriminantAnalysis()
lda_model.fit(X_train_processed, y_train)
y_pred_lda = lda_model.predict(X_test_processed)
error_lda = 1 - accuracy_score(y_test, y_pred_lda)
print('LDA Error: {:.2f}'.format(error_lda))

# Store the necessary variables for LDA
cols_to_store_lda = [i for i in range(len(lda_model.classes_)) if lda_model.classes_[i] == 1]
y_LDA_pred = lda_model.predict_proba(X_test_processed)[:, cols_to_store_lda]
y_LDA_true = (y_test.copy() == 1).copy()
```

```
LDA Error: 0.25
```

In [4]:

```python
# Quadratic Discriminant Analysis (QDA)
qda_model = QuadraticDiscriminantAnalysis()
qda_model.fit(X_train_processed, y_train)
y_pred_qda = qda_model.predict(X_test_processed)
error_qda = 1 - accuracy_score(y_test, y_pred_qda)
print('QDA Error: {:.2f}'.format(error_qda))

# Store the necessary variables for QDA
cols_to_store_qda = [i for i in range(len(qda_model.classes_)) if qda_model.classes_[i] == 1]
y_QDA_pred = qda_model.predict_proba(X_test_processed)[:, cols_to_store_qda]
y_QDA_true = (y_test.copy() == 1).copy()
```

```
QDA Error: 0.27
```

In [5]:

```python
# Logistic Regression
logreg_model = LogisticRegression()
logreg_model.fit(X_train_processed, y_train)
y_pred_logreg = logreg_model.predict(X_test_processed)
error_logreg = 1 - accuracy_score(y_test, y_pred_logreg)
print('Logistic Regression Error: {:.2f}'.format(error_logreg))

# Store the necessary variables for Logistic Regression
cols_to_store_logreg = [i for i in range(len(logreg_model.classes_)) if logreg_model.classes_[i] == 1]
y_logistic_pred = logreg_model.predict_proba(X_test_processed)[:, cols_to_store_logreg]
y_logistic_true = (y_test.copy() == 1).copy()
```

```
Logistic Regression Error: 0.25
```

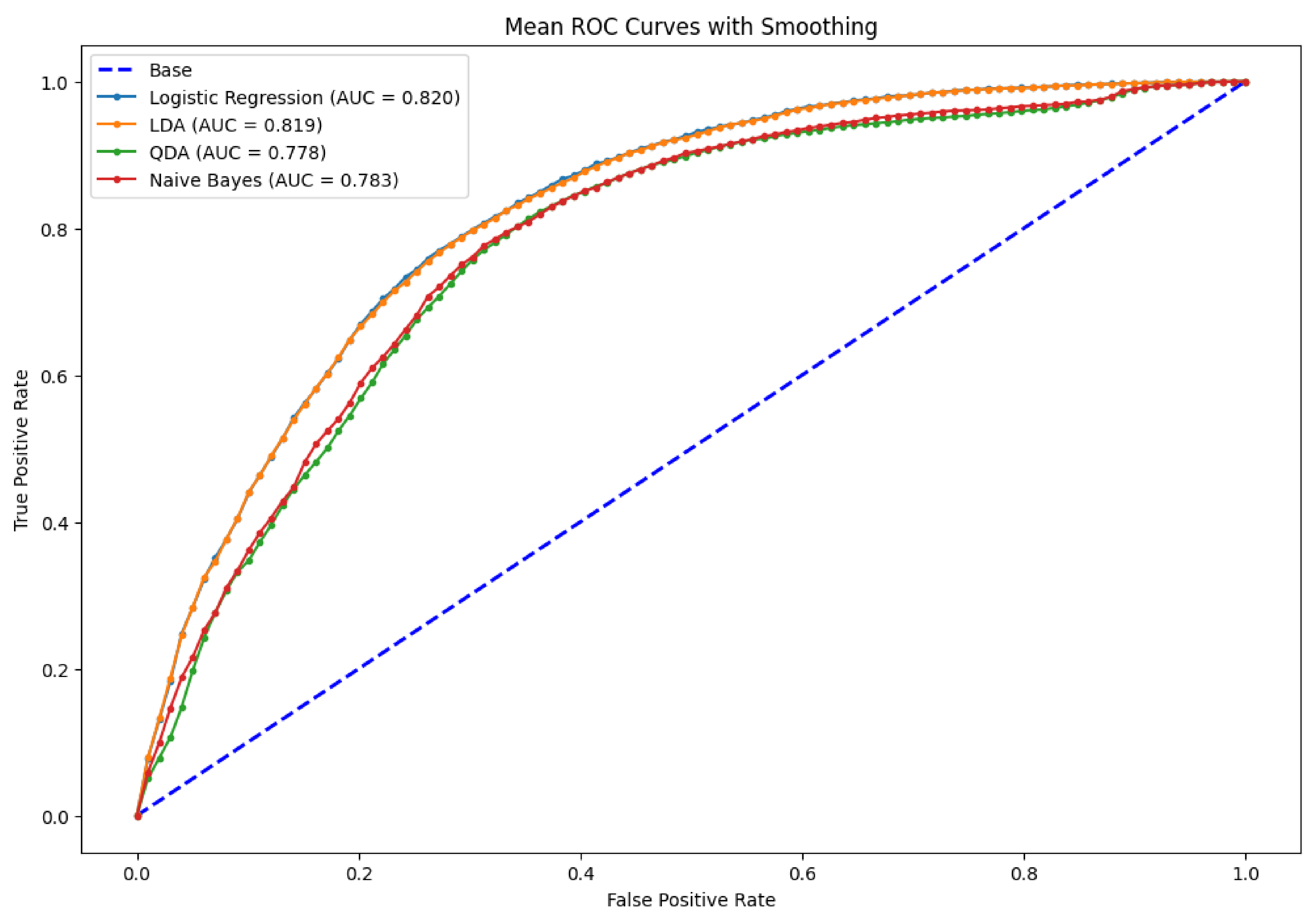
In [6]:

```python
# Naive Bayes
nb_model = GaussianNB()
nb_model.fit(X_train_processed, y_train)
y_pred_nb = nb_model.predict(X_test_processed)
error_nb = 1 - accuracy_score(y_test, y_pred_nb)
print('Naive Bayes Error: {:.2f}'.format(error_nb))

# Store the necessary variables for Naive Bayes
cols_to_store_nb = [i for i in range(len(nb_model.classes_)) if nb_model.classes_[i] == 1]
y_GNB_pred = nb_model.predict_proba(X_test_processed)[:, cols_to_store_nb]
y_GNB_true = (y_test.copy() == 1).copy()
```

```
Naive Bayes Error: 0.28
```

In [7]:

```python
from sklearn.metrics import roc_curve, roc_auc_score
import numpy as np

# Function to smooth the ROC curve
def smooth_roc_curve(fpr, tpr, smoothness=100):
    fpr_smooth = np.linspace(0, 1, smoothness)
    tpr_smooth = np.interp(fpr_smooth, fpr, tpr)
    return fpr_smooth, tpr_smooth

# Calculate ROC curves
lr_fpr, lr_tpr, _ = roc_curve(y_logistic_true, y_logistic_pred)
lr_fpr1, lr_tpr1, _ = roc_curve(y_LDA_true, y_LDA_pred)
lr_fpr2, lr_tpr2, _ = roc_curve(y_QDA_true, y_QDA_pred)
lr_fpr3, lr_tpr3, _ = roc_curve(y_GNB_true, y_GNB_pred)

# Smooth the ROC curves
smooth_lr_fpr, smooth_lr_tpr = smooth_roc_curve(lr_fpr, lr_tpr)
smooth_lr_fpr1, smooth_lr_tpr1 = smooth_roc_curve(lr_fpr1, lr_tpr1)
smooth_lr_fpr2, smooth_lr_tpr2 = smooth_roc_curve(lr_fpr2, lr_tpr2)
smooth_lr_fpr3, smooth_lr_tpr3 = smooth_roc_curve(lr_fpr3, lr_tpr3)

# Plot the ROC curves with adjusted linewidth and different colors
plt.figure(figsize=(12, 8))

# Print AUC in the legend
plt.plot([0, 1], [0, 1], linestyle='--', color='blue', lw=2, label='Base')
plt.plot(smooth_lr_fpr, smooth_lr_tpr, marker='.', label=f'Logistic Regression (AUC = {roc_auc_score(y_logistic_true, y_logistic_pred):.3f})')
plt.plot(smooth_lr_fpr1, smooth_lr_tpr1, marker='.', label=f'LDA (AUC = {roc_auc_score(y_LDA_true, y_LDA_pred):.3f})')
plt.plot(smooth_lr_fpr2, smooth_lr_tpr2, marker='.', label=f'QDA (AUC = {roc_auc_score(y_QDA_true, y_QDA_pred):.3f})')
plt.plot(smooth_lr_fpr3, smooth_lr_tpr3, marker='.', label=f'Naive Bayes (AUC = {roc_auc_score(y_GNB_true, y_GNB_pred):.3f})')

# Axis Labels
plt.legend()
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Mean ROC Curves with Smoothing')
plt.show()
```


Mean ROC Curves with Smoothing
Base
Logistic Regression (AUC = 0.820)
LDA (AUC = 0.819)
QDA (AUC = 0.778)
Naive Bayes (AUC = 0.783)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

```
In [ ]:
```