Report

First: I imported pandas to read the csv file then imported sklearn.model_selection to split the data then imported sklearn.naive_bayes to train the data and test it then imported sklearn.metrics to evaluate the data and get the accuracy, classification report, and confusion matrix.

main.py

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

Second: I read the csv from the url and put there headers in .columns function then searched for the data with question mark on it and replace it with NA then dropped the rows with NA in it and then chooses the target column and converted >50k to 1 and <50k to 0.

```python
# Load and Preprocess the Data
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data"
adult_data = pd.read_csv(url, header=None)
adult_data.columns = ['age', 'workclass', 'fnlwgt', 'education', 'education-num', 'marital-status', 'occupation',
                      'relationship', 'race', 'sex', 'capital-gain', 'capital-loss', 'hours-per-week', 'native-country',
                      'income']

adult_data.replace('?', pd.NA, inplace=True)  # Replace '?' with NA
adult_data.dropna(inplace=True)  # Drop rows with missing values
adult_data['income'] = adult_data['income'].apply(lambda x: 1 if x.strip() == '>50K' else 0)  # Convert income to binary
```

Third: I turned the categorical data to dummy code which is 0s and 1s like a table to recognize each row then separated the features from target and splits them to train and test with 20% from the data and specified the random state to 74 to remain the same results every time with the same training set and testing set (74 calculated to ensure that it is the best one in range from 1 to 100) .

```
# Encode categorical variables
adult_data = pd.get_dummies(adult_data, columns=['workclass', 'education', 'marital-status', 'occupation',
                                                 'relationship', 'race', 'sex', 'native-country'])

# Separate features and target variable
X = adult_data.drop('income', axis=1)  # features
y = adult_data['income']  # target

# Split the Data into Training and Testing Sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=74)
# random_state to insure the outputs are the same at every function call
```

Fourth: trained the Naïve Bayes Classifier with GaussianNB() function from sklearn.naive_bayes with the training data set then predicted the test features and got there target then calculated the accuracy with function accuracy_score and give it the parameters y_test which is the target of the test set and y_predict which is the predicted target from the test features we calculated.

```
# Train the Naive Bayes Classifier
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)

# Evaluate the Model
y_pred = nb_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("\nAccuracy:", accuracy, "\n")
```

Fifth: calculated the confusion matrix with confusion_matrix function and plotted it and computed the sensitivity and specificity from the confusion matrix.

```
conf_matrix = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = conf_matrix.ravel()

# Confusion matrix
plt.figure(figsize=(8, 6))
sns.set(font_scale=1.4)
sns.heatmap([[tn, fp], [fn, tp]], annot=True, fmt='g', cmap='Blues',
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()

# Compute Sensitivity (True Positive Rate)
sensitivity = tp / (tp + fn)

# Compute Specificity (True Negative Rate)
specificity = tn / (tn + fp)

print("Sensitivity (True Positive Rate):", sensitivity)
print("Specificity (True Negative Rate):", specificity, "\n")
```

Sixth: I calculated the posterior probability of making over 50k a year with function predict_proba from GaussianNB() which calculates the probability of each feature given class and then calculate the positive class probability (>50k or 1) then getting the mean of it then printed the classification report.

```
# Compute the posterior probability of making over 50K a year
posterior_probs = nb_classifier.predict_proba(X_test)
positive_class_prob = posterior_probs[:, 1]  # Probability of the positive class


# Assuming the positive class corresponds to making over 50K a year
mean_positive_prob = positive_class_prob.mean()

print("Mean posterior probability of making over 50K a year:", mean_positive_prob, "\n")

print(classification_report(y_test, y_pred))
```