FOUNDATIONS FOR DATA ANALYTICS(LAB-7)

NAME:SUDANA SHASHI KIRAN
REGD NUMBER : 21BCE8644

Python – Basic operations on Numpy & MLR performance

Numpy basic operations:

CODE:

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.stats.api as sms
```

```
# Creating an array
arr = np.array([1, 2, 3, 4, 5])
print("Array:", arr)
```

```
# Array dimensions
dimensions = arr.ndim
print("Array Dimensions:", dimensions)
```

```
# Array shape
shape = arr.shape
print("Array Shape:", shape)
```

```
# Array size
size = arr.size
print("Array Size:", size)
```

```
# Accessing elements
element = arr[2]
print("Element at Index 2:", element)
```

```
# Slicing
sliced_arr = arr[1:4]
print("Sliced Array:", sliced_arr)
```

```python
# Arithmetic operations
result = arr + 10
print("Result of Addition:", result)
```

```python
# Dot product
dot_product = np.dot(arr, arr)
print("Dot Product:", dot_product)
```

```python
# Element-wise multiplication
elementwise_product = arr * 2
print("Element-wise Multiplication:", elementwise_product)
```

```python
# Sum of elements
sum_elements = np.sum(arr)
print("Sum of Elements:", sum_elements)
```

```python
# Mean of elements
mean = np.mean(arr)
print("Mean of Elements:", mean)
```

```python
# Maximum element
max_value = np.max(arr)
print("Maximum Element:", max_value)
```

```python
# Minimum element
min_value = np.min(arr)
print("Minimum Element:", min_value)
```

```python
# Reshaping array
reshaped_arr = arr.reshape (5,1)
print("Reshaped Array:\n", reshaped_arr)
```

```python
# Transposing array
transposed_arr = arr.T
print("Transposed Array:\n", transposed_arr)
```

```python
# Broadcasting
broadcasted_arr = arr + 5
print("Broadcasted Array:\n", broadcasted_arr)
```

```python
# Finding unique values
unique_values = np.unique(arr)
print("Unique Values:", unique_values)
```

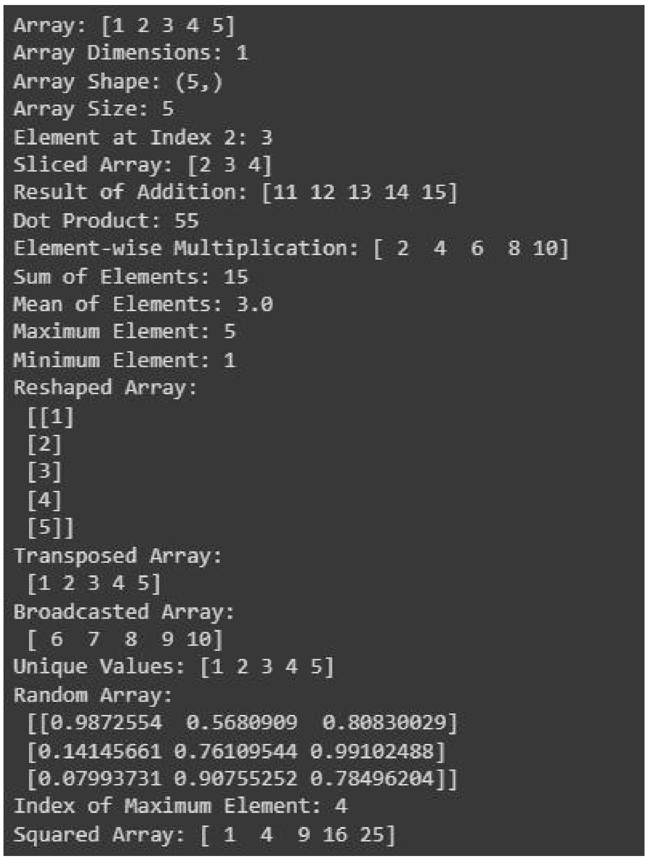
```python
# Generating random numbers
```

```
random_array = np.random.rand(3, 3)
print("Random Array:\n", random_array)
```

```
# Finding index of maximum element
max_index = np.argmax(arr)
print("Index of Maximum Element:", max_index)
```

```
# Applying a function to each element
squared_arr = np.square(arr)
print("Squared Array:", squared_arr)
```

OUTPUT:



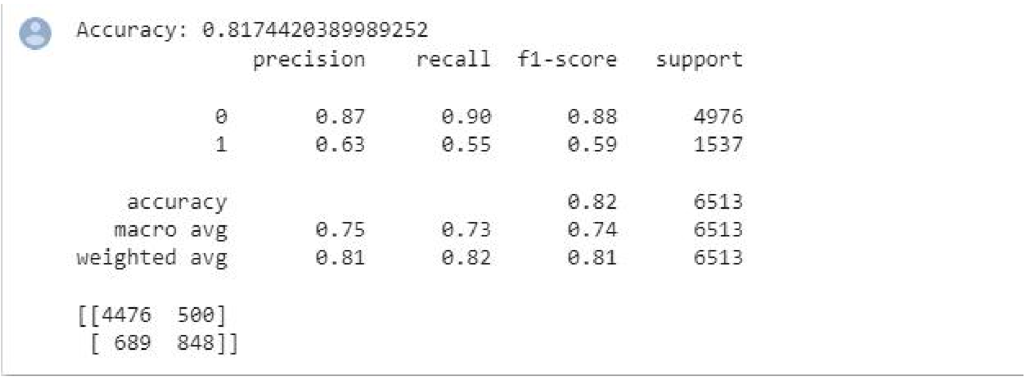
```
Array: [1 2 3 4 5]
Array Dimensions: 1
Array Shape: (5,)
Array Size: 5
Element at Index 2: 3
Sliced Array: [2 3 4]
Result of Addition: [11 12 13 14 15]
Dot Product: 55
Element-wise Multiplication: [ 2  4  6  8 10]
Sum of Elements: 15
Mean of Elements: 3.0
Maximum Element: 5
Minimum Element: 1
Reshaped Array:
 [[1]
 [2]
 [3]
 [4]
 [5]]
Transposed Array:
 [1 2 3 4 5]
Broadcasted Array:
 [ 6  7  8  9 10]
Unique Values: [1 2 3 4 5]
Random Array:
 [[0.9872554  0.5680909  0.80830029]
 [0.14145661 0.76109544 0.99102488]
 [0.07993731 0.90755252 0.78496204]]
Index of Maximum Element: 4
Squared Array: [ 1  4  9 16 25]
```

MLR performance:

K-Nearest Neighbours (KNN):
Supervised or unsupervised method for classification and regression, making predictions based on the majority class or average value of the k-nearest data points in feature space.

CODE:
```
from sklearn.neighbors import KNeighborsClassifier
knn_classifier = KNeighborsClassifier()
knn_classifier.fit(X_train, y_train)
y_pred = knn_classifier.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

EXPLAINATION OF CODE:
The code uses a K-Nearest Neighbors (KNN) Classifier from scikit-learn to create a model. It fits the classifier to the training data and predicts 'income' on the test set. The code then prints the accuracy score, classification report, and confusion matrix to evaluate the model's performance in predicting income levels.
KNN is a non-parametric algorithm used for both classification and regression tasks. It assigns the class label to a data point based on the majority class among its k-nearest neighbors, where k is a user-defined hyperparameter. KNN is simple to implement and can be effective when there is a clear separation between classes in the feature space.

OUTPUT:

```
Accuracy: 0.8174420389989252
              precision    recall  f1-score   support

           0       0.87      0.90      0.88      4976
           1       0.63      0.55      0.59      1537

    accuracy                           0.82      6513
   macro avg       0.75      0.73      0.74      6513
weighted avg       0.81      0.82      0.81      6513

[[4476  500]
 [ 689  848]]
```

Gaussian NB:

Gaussian Naive Bayes (Gaussian NB) is a simple and efficient probabilistic classification algorithm. It assumes features are normally distributed within each class and that features are conditionally independent given the class label. It works well with small datasets, especially for text classification tasks, but may not perform well with highly correlated features.

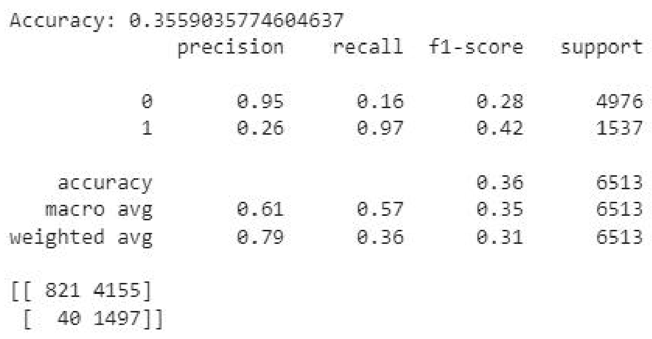
CODE:

```
from sklearn.naive_bayes import GaussianNB
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)
y_pred = nb_cl
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

EXPLANATION OF CODE:

The code snippet uses the Gaussian Naive Bayes classifier to perform classification, makes predictions, and evaluates the model's performance using accuracy, a classification report, and a confusion matrix.

OUTPUT:



```
Accuracy: 0.3559035774604637
              precision    recall  f1-score   support

           0       0.95      0.16      0.28      4976
           1       0.26      0.97      0.42      1537

    accuracy                           0.36      6513
   macro avg       0.61      0.57      0.35      6513
weighted avg       0.79      0.36      0.31      6513

[[ 821 4155]
 [  40 1497]]
```

MLP Classifier :

The MLP Classifier is a powerful feedforward neural network designed for classification tasks. It consists of multiple interconnected layers, including input, hidden, and output layers. Training involves backpropagation to minimize errors, and its flexibility allows handling complex non-linear relationships in data. Proper hyperparameter tuning is essential for optimal performance**.**

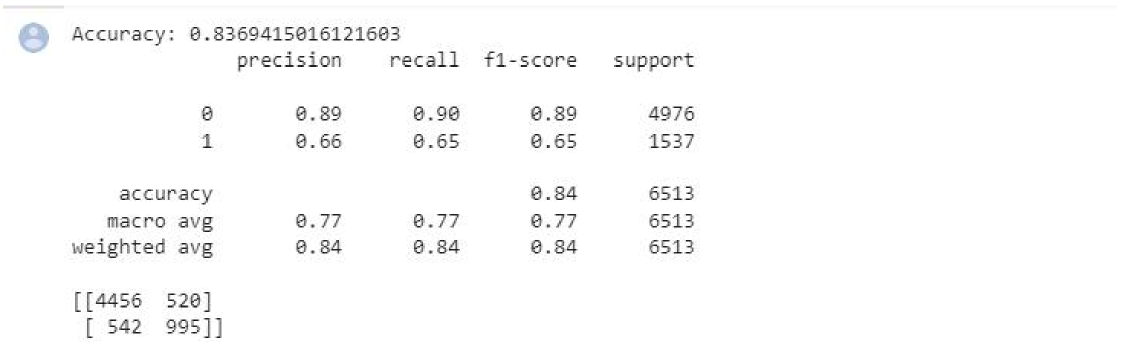
CODE:

```
from sklearn.neural_network import MLPClassifier
nn_classifier = MLPClassifier(hidden_layer_sizes=(100,), max_iter=100)
nn_classifier.fit(X_train, y_train)
y_pred = nn_classifier.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

EXPLANATION OF CODE:

The code uses the MLPClassifier from scikit-learn to create a neural network model. It sets the number of neurons in the hidden layer to 100 and the maximum number of iterations to 100. The model is then trained on the training data and predicts 'income' on the test set. The code prints the accuracy score, classification report, and confusion matrix to evaluate the model's performance in predicting income levels. MLPClassifier is a powerful neural network algorithm capable of handling complex non-linear relationships in data. The number of neurons and iterations can be adjusted to fine-tune the model's performance.

OUTPUT:



```
Accuracy: 0.8369415016121603
              precision    recall  f1-score   support

           0       0.89      0.90      0.89      4976
           1       0.66      0.65      0.65      1537

    accuracy                           0.84      6513
   macro avg       0.77      0.77      0.77      6513
weighted avg       0.84      0.84      0.84      6513

[[4456  520]
 [ 542  995]]
```

Gradient Boosting Machines (GBM) :

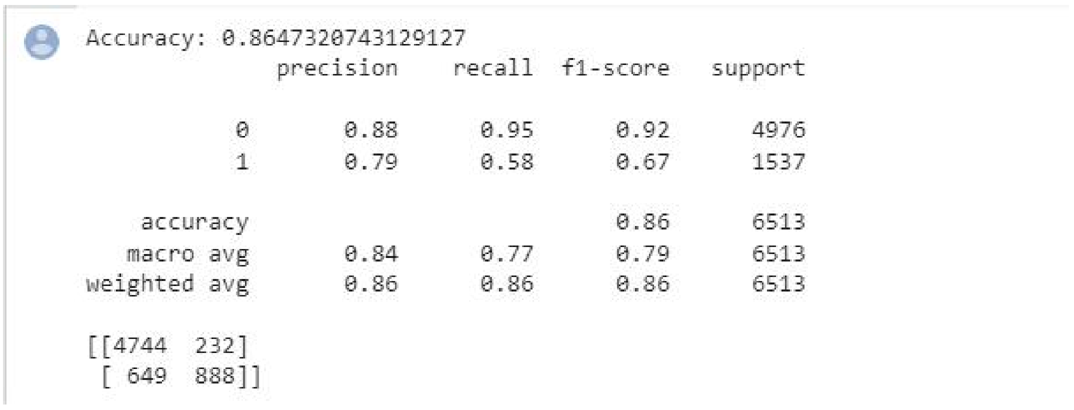
Boosting technique that constructs multiple decision trees, each learning from the mistakes of the previous one, achieving high predictive accuracy.

CODE:

```
from sklearn.ensemble import GradientBoostingClassifier
gbm_classifier = GradientBoostingClassifier()
gbm_classifier.fit(X_train, y_train)
y_pred = gbm_classifier.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

EXPLANATION OF CODE:
The code uses the GradientBoostingClassifier from scikit-learn to create an ensemble model called Gradient Boosting Machine (GBM). It fits the classifier to the training data and predicts 'income' on the test set. The code then prints the accuracy score, classification report, and confusion matrix to evaluate the model's performance in predicting income levels.
Gradient Boosting Classifier is an ensemble learning method that builds multiple weak learners (typically decision trees) sequentially, with each subsequent model correcting the errors made by the previous ones. It combines their predictions to produce a more accurate and robust model. GBM is known for its high predictive accuracy and ability to handle complex non-linear relationships in data, making it a popular choice for various classification tasks.

OUTPUT:



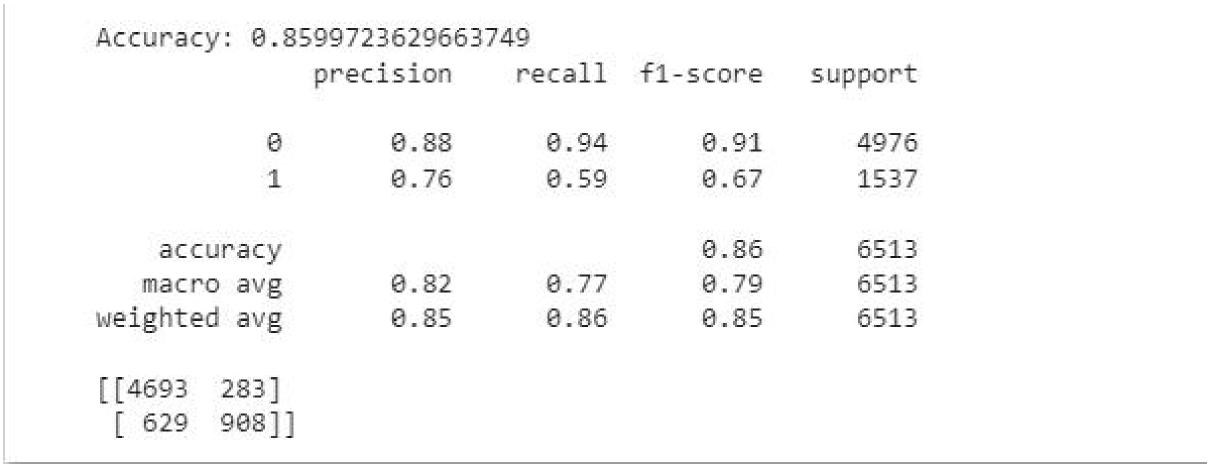
```
Accuracy: 0.8647320743129127
              precision    recall  f1-score   support

           0       0.88      0.95      0.92      4976
           1       0.79      0.58      0.67      1537

    accuracy                           0.86      6513
   macro avg       0.84      0.77      0.79      6513
weighted avg       0.86      0.86      0.86      6513

[[4744  232]
 [ 649  888]]
```

AdaBoost:
Boosting algorithm assigning higher weights to misclassified samples, focusing on challenging instances to enhance classification accuracy in ensemble learning.

CODE:

```
from sklearn.ensemble import AdaBoostClassifier
adaboost_classifier = AdaBoostClassifier()
adaboost_classifier.fit(X_train, y_train)
y_pred = adaboost_classifier.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

EXPLANATION OF CODE:
The code uses the AdaBoostClassifier from scikit-learn to create an ensemble model called AdaBoost. It fits the classifier to the training data and predicts 'income' on the test set. The code then prints the accuracy score, classification report, and confusion matrix to evaluate the model's performance in predicting income levels.
AdaBoost (Adaptive Boosting) is an ensemble learning method that combines multiple weak learners, typically decision trees, to build a strong model. It gives more weight to misclassified data points in each iteration, allowing subsequent weak learners to focus on those areas, improving overall performance. AdaBoost is effective for both binary and multi-class classification tasks, often achieving high accuracy with relatively simple models.

OUTPUT:


```
Accuracy: 0.8599723629663749
              precision    recall  f1-score   support

           0       0.88      0.94      0.91      4976
           1       0.76      0.59      0.67      1537

    accuracy                           0.86      6513
   macro avg       0.82      0.77      0.79      6513
weighted avg       0.85      0.86      0.85      6513

[[4693  283]
 [ 629  908]]
```

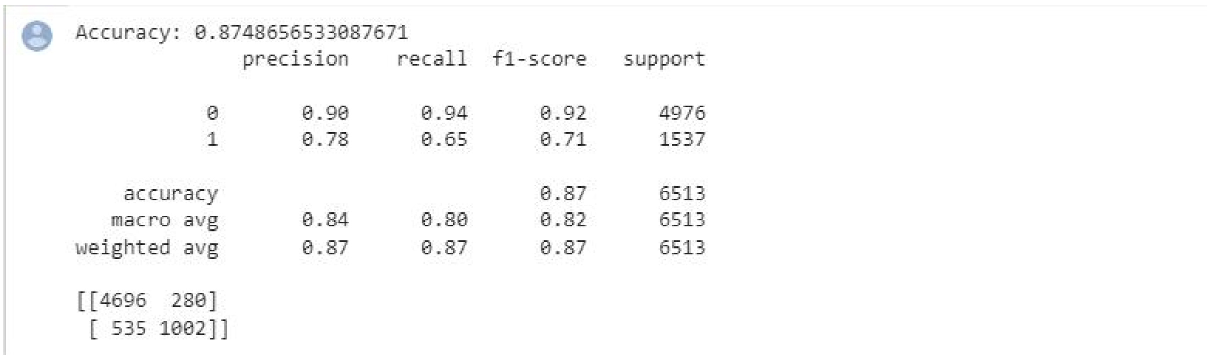
XGBoost:
Extreme Gradient Boosting, a highly efficient and scalable implementation of gradient boosting algorithm, popular for various machine learning tasks.

CODE:
```
from xgboost import XGBClassifier
xgb_classifier = XGBClassifier()
xgb_classifier.fit(X_train, y_train)
y_pred = xgb_classifier.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

EXPLANATION OF CODE:

The code uses the XGBClassifier from the XGBoost library to create an ensemble model called XGBoost. It fits the classifier to the training data and predicts 'income' on the test set. The code then prints the accuracy score, classification report, and confusion matrix to evaluate the model's performance in predicting income levels.
XGBoost (Extreme Gradient Boosting) is an advanced and efficient gradient boosting algorithm known for its speed and high accuracy. It is widely used in various machine learning competitions and real-world applications. XGBoost uses decision trees as base learners and employs techniques like regularization and parallel processing to improve model performance and prevent overfitting. It is particularly effective for structured/tabular data and can handle large datasets efficiently.

OUTPUT:



```
Accuracy: 0.8748656533087671
              precision    recall  f1-score   support

           0       0.90      0.94      0.92      4976
           1       0.78      0.65      0.71      1537

    accuracy                           0.87      6513
   macro avg       0.84      0.80      0.82      6513
weighted avg       0.87      0.87      0.87      6513

[[4696  280]
 [ 535 1002]]
```

LightGBM:

High-performance gradient boosting framework optimized for speed and scalability, suitable for large-scale machine learning applications.

CODE:

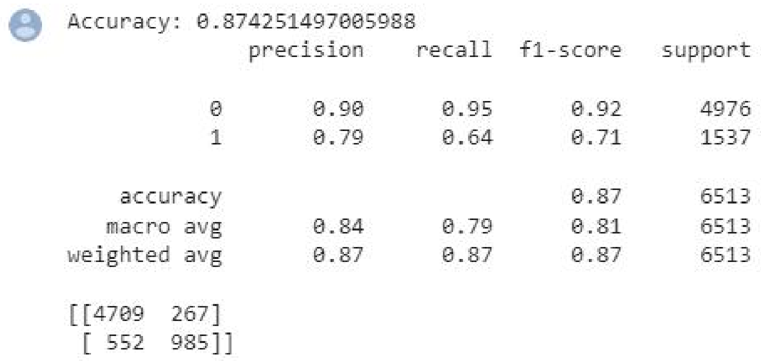
```
from lightgbm import LGBMClassifier
lgbm_classifier = LGBMClassifier()
lgbm_classifier.fit(X_train, y_train)
y_pred = lgbm_classifier.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

EXPLANATION OF CODE:

The code uses the LGBMClassifier from the LightGBM library to create an ensemble model called LightGBM. It fits the classifier to the training data and predicts 'income' on the test set. The code then prints the accuracy score, classification report, and confusion matrix to evaluate the model's performance in predicting income levels.

LightGBM is a gradient boosting framework known for its speed and efficiency. It is designed to handle large datasets and perform well on both binary and multi-class classification tasks. LightGBM uses a histogram-based approach for gradient boosting, which reduces memory usage and speeds up training. Due to its efficiency and accuracy, LightGBM is widely used in various data science competitions and real-world applications.

OUTPUT:



```
Accuracy: 0.874251497005988
              precision    recall  f1-score   support

           0       0.90      0.95      0.92      4976
           1       0.79      0.64      0.71      1537

    accuracy                           0.87      6513
   macro avg       0.84      0.79      0.81      6513
weighted avg       0.87      0.87      0.87      6513

[[4709  267]
 [ 552  985]]
```

K-Means Clustering:

Unsupervised algorithm partitioning data into K clusters by iteratively assigning points to the nearest centroid, useful for data clustering tasks based on similarity.

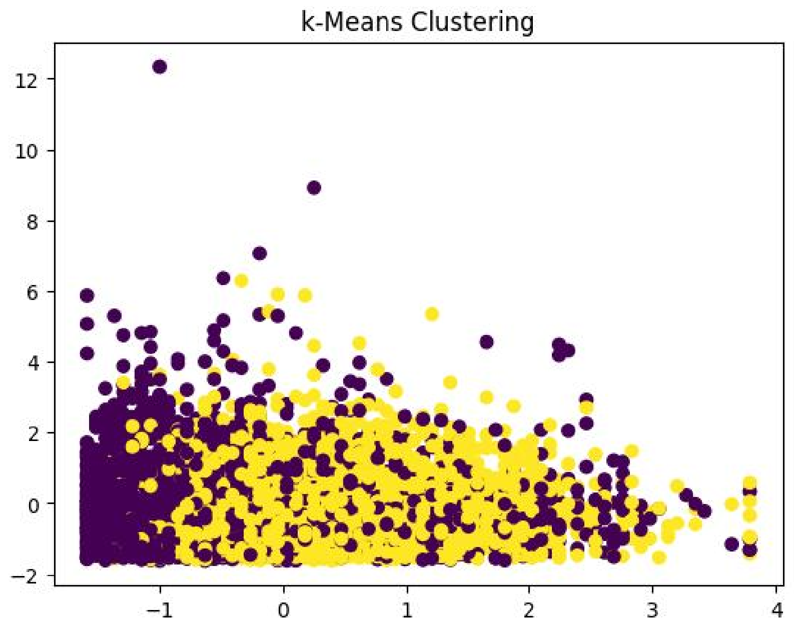
CODE:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA, FastICA
from sklearn.manifold import TSNE
from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN
from sklearn.mixture import GaussianMixture
from sklearn.neighbors import NearestNeighbors
from mlxtend.frequent_patterns import apriori, association_rules
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans.fit(X_train)
kmeans_clusters = kmeans.predict(X_test)
plt.scatter(X_test[:, 0], X_test[:, 1], c=kmeans_clusters, cmap='viridis')
plt.title("k-Means Clustering")
plt.show()
```

EXPLAINATION OF CODE:

The code performs clustering using k-Means algorithm on the test set 'X_test' after applying dimensionality reduction techniques such as PCA, FastICA, and t-SNE. It then visualizes the clustering results using a scatter plot, where data points are colored based on their cluster assignments. The k-Means algorithm is used to group data points into two clusters, specified by the 'n_clusters' parameter. The plot provides a visual representation of how the data points are grouped together based on the k-Means clustering algorithm.

OUTPUT:



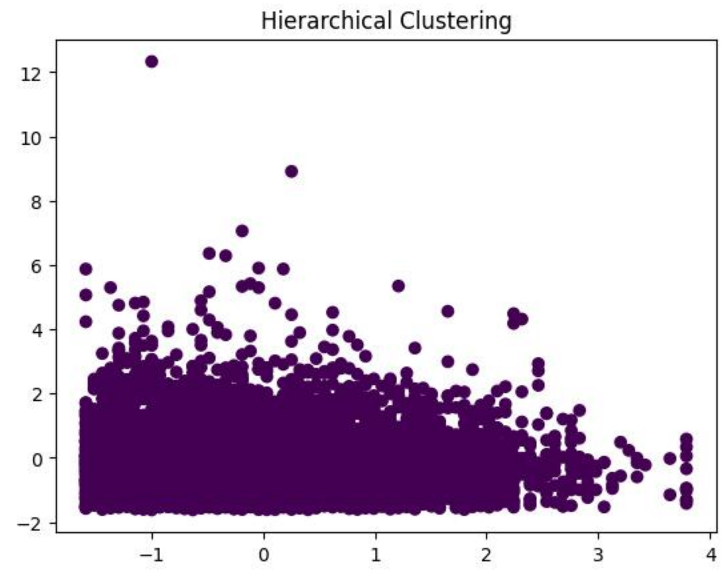
Hierarchical Clustering:
Unsupervised technique forming a tree-like structure of nested clusters by iteratively merging or splitting data points based on similarity, revealing hierarchical relationships in data.

CODE:
```
hierarchical = AgglomerativeClustering(n_clusters=2)
hierarchical_clusters = hierarchical.fit_predict(X_test)
plt.scatter(X_test[:, 0], X_test[:, 1], c=hierarchical_clusters, cmap='viridis')
plt.title("Hierarchical Clustering")
plt.show()
```

EXPLANATION OF CODE:
The code performs hierarchical clustering on the test set 'X_test' using the AgglomerativeClustering algorithm. It groups data points into two clusters, specified by the 'n_clusters' parameter. The resulting cluster assignments are then visualized using a scatter plot, where data points are colored based on

their cluster memberships. The plot provides a visual representation of how the data points are grouped together based on the hierarchical clustering algorithm.

OUTPUT:



Gaussian Mixture Models (GMM):
Unsupervised probabilistic model that assumes data points are generated from a combination of multiple Gaussian distributions, helpful for clustering with overlapping clusters and complex patterns.

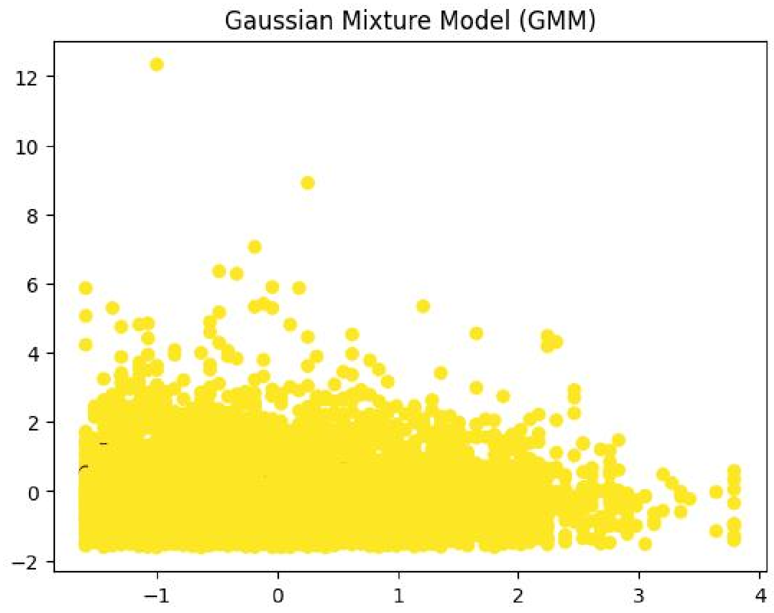
CODE:

```
gmm = GaussianMixture(n_components=2, random_state=42)
gmm.fit(X_train)
gmm_clusters = gmm.predict(X_test)
plt.scatter(X_test[:, 0], X_test[:, 1], c=gmm_clusters, cmap='viridis')
plt.title("Gaussian Mixture Model (GMM)")
plt.show()
```

EXPLAINATION OF CODE:
The code uses the Gaussian Mixture Model (GMM) algorithm to perform clustering on the test set 'X_test'. It specifies two components (clusters) using the 'n_components' parameter. The GMM fits a probabilistic model to the data, estimating the underlying Gaussian distributions of each cluster. The resulting

cluster assignments are visualized using a scatter plot, where data points are colored based on their GMM cluster memberships. The plot provides a visual representation of how the data points are grouped together based on the Gaussian Mixture Model algorithm.

OUTPUT:



Dimensionality Reduction (Principal Component Analysis):

PCA is a technique used to reduce the number of variables in a dataset while preserving important information.

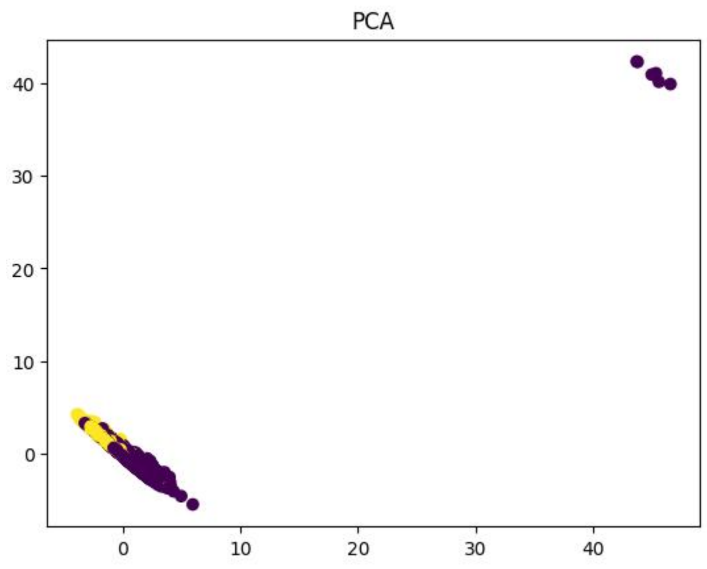
CODE:

```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_test)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y_test, cmap='viridis')
plt.title("PCA")
plt.show()
```

EXPLAINATION OF CODE:

The code applies Principal Component Analysis (PCA) to reduce the dimensionality of the test set 'X_test' to two principal components. It transforms the data into a 2-dimensional space that captures the most important patterns in the data. The resulting transformed data is visualized using a scatter plot, where data points are colored based on their true 'income' labels ('y_test'). The plot provides a visual representation of how the data points are distributed in the reduced 2D space after PCA. PCA helps to visualize the data and identify potential clusters or patterns.

OUTPUT:

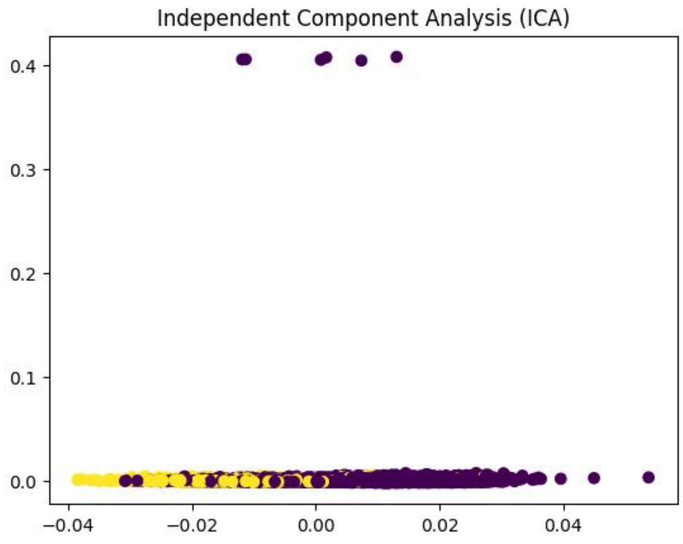


Independent Component Analysis (ICA):
Unsupervised method separating a mixture of signals into statistically independent components, useful for blind source separation and feature extraction in signal processing and image analysis.

CODE:
```
ica = FastICA(n_components=2)
X_ica = ica.fit_transform(X_test)
plt.scatter(X_ica[:, 0], X_ica[:, 1], c=y_test, cmap='viridis')
plt.title("Independent Component Analysis (ICA)")
plt.show()
```

EXPLAINATION OF CODE:
The code applies Independent Component Analysis (ICA) to the test set 'X_test' to extract two independent components from the data. ICA aims to separate mixed signals into statistically independent components, which can reveal hidden patterns in the data. The resulting transformed data is visualized using a scatter plot, where data points are colored based on their true 'income' labels ('y_test'). The plot provides a visual representation of how the data points are distributed in the 2D space after ICA, highlighting any potential independent patterns in the data. ICA is particularly useful when dealing with mixed signals or finding meaningful latent factors in the data.

OUTPUT:



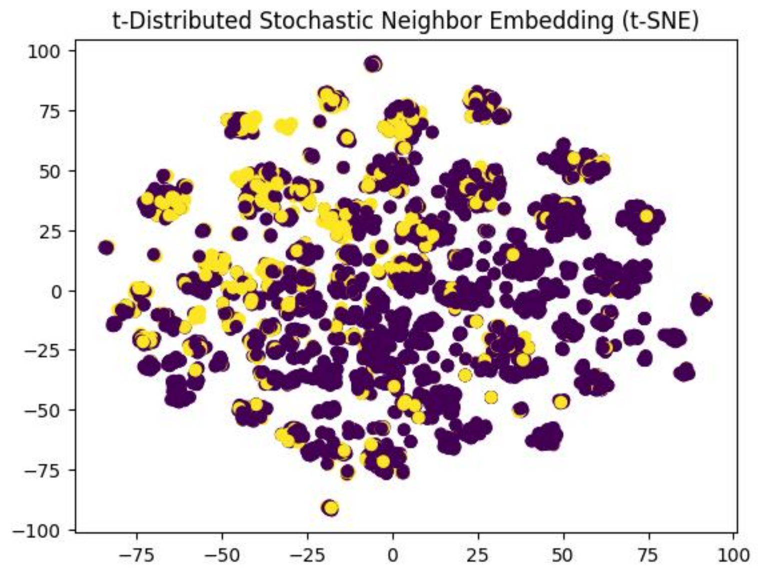
t-Distributed Stochastic Neighbor Embedding (t-SNE):
Non-linear dimensionality reduction method emphasizing preservation of local similarities, effectively visualizing high-dimensional data in a lower-dimensional space.

CODE:

```
tsne = TSNE(n_components=2, random_state=42)
X_tsne = tsne.fit_transform(X_test)
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y_test, cmap='viridis')
plt.title("t-Distributed Stochastic Neighbor Embedding (t-SNE)")
plt.show()
```

EXPLAINATION OF CODE:
The code applies t-Distributed Stochastic Neighbor Embedding (t-SNE) to the test set 'X_test' to reduce its dimensionality to two components. t-SNE is a non-linear dimensionality reduction technique that aims to preserve local similarities between data points. The resulting transformed data is visualized using a scatter plot, where data points are colored based on their true 'income' labels ('y_test'). The plot provides a visual representation of how the data points are distributed in the 2D space after t-SNE, highlighting any potential clusters or patterns in the data. t-SNE is often used for visualization and can reveal intricate structures in the data that may not be easily captured by other dimensionality reduction methods.

OUTPUT:



DBSCAN (Density-Based Spatial Clustering of Applications with Noise): Unsupervised clustering algorithm grouping data based on density, forming clusters in regions of high density while identifying outliers as noise, applicable to spatial data with varying cluster shapes and sizes.

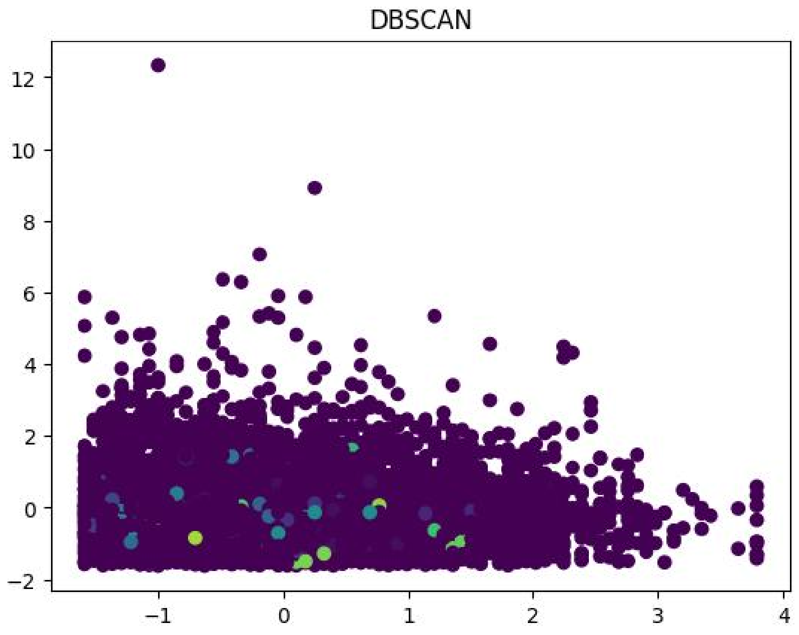
CODE:

```
dbscan = DBSCAN(eps=0.5, min_samples=5)
dbscan_clusters = dbscan.fit_predict(X_test)
plt.scatter(X_test[:, 0], X_test[:, 1], c=dbscan_clusters, cmap='viridis')
plt.title("DBSCAN")
plt.show()
```

EXPLAINATION OF CODE:

The code applies Density-Based Spatial Clustering of Applications with Noise (DBSCAN) to the test set 'X_test'. DBSCAN is a density-based clustering algorithm that groups data points based on their density and identifies noise points as well. It requires two hyperparameters: 'eps' specifies the maximum distance between two points to be considered neighbors, and 'min_samples' determines the minimum number of points needed to form a dense region. The resulting cluster assignments from DBSCAN are visualized using a scatter plot, where data points are colored based on their cluster memberships. The plot provides a visual representation of how the data points are grouped together based on their density and proximity in the 2D space. DBSCAN is effective in identifying arbitrary-shaped clusters and can handle noisy data points as outliers, making it suitable for various clustering tasks.

OUTPUT:



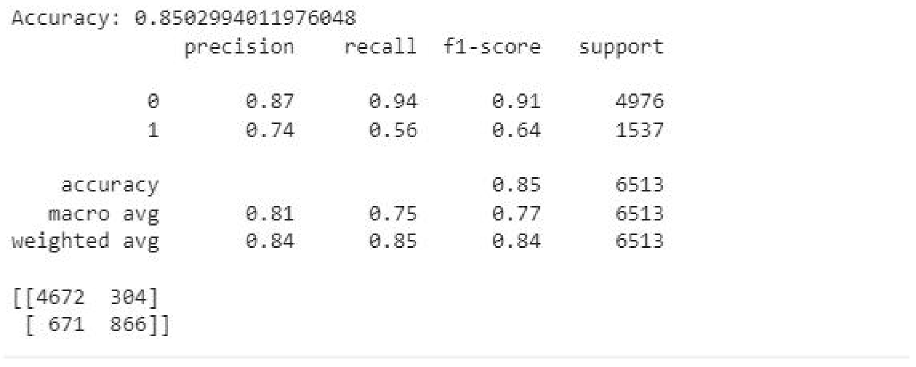
Supervised SVM model:

Powerful ML algorithm for classification/regression. Finds optimal hyperplane to separate data, maximizes margin, handles high-dimensional data, and uses kernels for non-linearity. Versatile for tasks like image recognition, text classification, and bioinformatics.

CODE:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.svm import SVC
from sklearn.semi_supervised import SelfTrainingClassifier
svm_classifier = SVC(kernel='linear', C=1.0, random_state=42)
svm_classifier.fit(X_train, y_train)
semi_supervised_svm = SelfTrainingClassifier(svm_classifier)
semi_supervised_svm.fit(X_train, y_train)
y_pred = semi_supervised_svm.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

EXPLAINATION OF CODE:
The code performs semi-supervised learning using a Support Vector Machine (SVM) classifier with a linear kernel on the test set 'X_test'. It first trains the SVM classifier on the labeled training data ('X_train' and 'y_train'). Then, it applies SelfTrainingClassifier to incorporate unlabeled data points into the model during training.
SelfTrainingClassifier is a semi-supervised learning approach that uses unlabeled data to improve model performance. It assigns pseudo-labels to the unlabeled data points based on the model's predictions and incorporates them into the training process. This way, the model benefits from both labeled and unlabeled data to make predictions.
After training the semi-supervised SVM, it predicts 'income' on the test set ('X_test') and evaluates the model's performance by printing the accuracy score, classification report, and confusion matrix. Semi-supervised learning can be useful when labeled data is limited, and leveraging unlabeled data can enhance model generalization and performance.

OUTPUT:

```
Accuracy: 0.8502994011976048
              precision    recall  f1-score   support

           0       0.87      0.94      0.91      4976
           1       0.74      0.56      0.64      1537

    accuracy                           0.85      6513
   macro avg       0.81      0.75      0.77      6513
weighted avg       0.84      0.85      0.84      6513

[[4672  304]
 [ 671  866]]
```

Label propagation model:
Semi-supervised learning method that propagates labels across data points, assigning labels to unlabeled instances based on similarity or affinity with labeled data, effectively using the unlabeled data to improve classification accuracy, particularly in scenarios with limited labeled data**.**

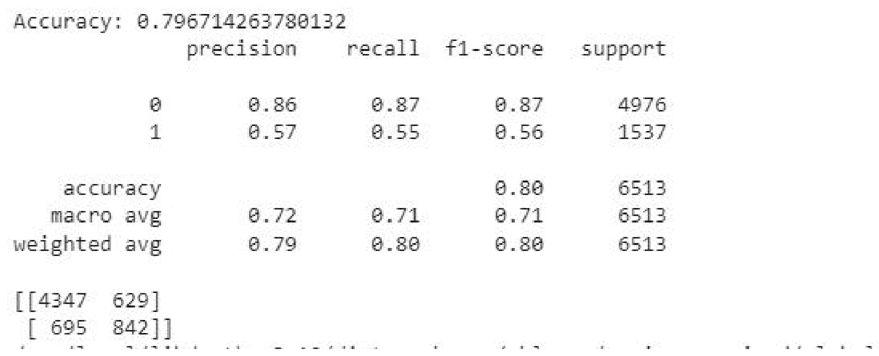
CODE:
```
import pandas as pd
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.semi_supervised import LabelPropagation
label_propagation = LabelPropagation()
label_propagation.fit(X_train, y_train)
y_pred = label_propagation.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

EXPLAINATION OF CODE:

The code uses the LabelPropagation algorithm, a semi-supervised learning approach, to predict 'income' labels for the test set 'X_test.' It leverages both labeled and unlabeled data to propagate labels in a graph-based manner, making predictions based on data similarity. The code then evaluates the model's performance by printing the accuracy score, classification report, and confusion matrix. This approach is useful when labeled data is scarce, as it allows utilizing unlabeled data to enhance the model's predictions and generalization capabilities.

OUTPUT:



```
Accuracy: 0.796714263780132
              precision    recall  f1-score   support

           0       0.86      0.87      0.87      4976
           1       0.57      0.55      0.56      1537

    accuracy                           0.80      6513
   macro avg       0.72      0.71      0.71      6513
weighted avg       0.79      0.80      0.80      6513

[[4347  629]
 [ 695  842]]
```