Kernel PCA AND Applications

Submitted By:

Aiza Ateeq (BSM-20-27)

Ayesha Shahzad (BSM-20-52)

Supervised By:

Dr. Fiza Zafar

Center for Advanced Studies in Pure and Applied Mathematics

Bahauddin Zakariya University, Multan, Pakistan

2020-2024

Submission Certificate

A project titled: "**Kernel PCA and Applications**" has been completed by **Aiza Ateeq and Ayesha Shahzad** under the supervision of **Dr.Fiza Zafar** . Report of this study is hereby submitted in partial fulfillment of requirements for the degree of "**'BS MATHEMTHICS (2020-2024)**".

Signature

_____

StudentName

_____

Roll No.

_____

Supervisor

_____

Acceptance Certificate

We, hereby accept this report of the project "**Kernel PCA and Applications**" submitted by **Aiza Ateeq and Ayesha Shahzad** under the supervision of **Dr. Fiza Zafar** as conforming to the required standards.

TeacherName

Supervisor

Associate Professor (Tenured)

CASPAM, B.Z. University Multan.

(Internal Examiner)

(External Examiner)

In-charge Examinations (CASPAM)

Dr. Muhammad Ibrahim

Assistant Professor

CASPAM, B.Z. University Multan.

Acknowledgements

I would like to express my heartfelt gratitude to all those who have contributed to the successful completion of this project.

Firstly, I would like to thank my supervisor, Dr. Fiza Zafar, for their invaluable guidance, support, and encouragement throughout the project. Their expertise and feedback have been instrumental in shaping my research and helped me achieve my goals.

I would also like to thank CASPAM for providing me with the resources and facilities necessary to carry out this project. I am grateful to my family and friends for their unwavering support and encouragement throughout my academic journey. Their constant motivation and belief in me have been the driving force behind my success.

Finally, I would like to thank all my colleagues and friends who have been a source of inspiration, motivation, and support. Their feedback and suggestions have been invaluable, and I could not have completed this project without their help. Thank you all once again for your support, encouragement, and invaluable contributions.

Abstract

In today's data-driven world, the ability to efficiently extract meaningful information from complex datasets is paramount. Kernel Principal Component Analysis (Kernel PCA) is an extension of Principal Component Analysis (PCA) that employs kernel methods to enable nonlinear dimensionality reduction. Unlike traditional PCA, which relies on linear transformations, Kernel PCA maps data into a higher-dimensional feature space using a kernel function, allowing for the capture of complex structures and patterns within the data. This mapping transforms the input space into a feature space where linear PCA can then be applied. Kernel PCA is particularly useful for data that is not linearly separable, providing more flexibility and effectiveness in tasks such as denoising, clustering, and classification.

Contents

Chapter 1

A Comprehensive Exploration of KPCA

1.1 Context and Background

Too much of anything is good for nothing!

Suppose you are working on a large-scale data science project. What happens when the given data set has too many variables? There are a few possible situations that you might come across. For instance, you find that most of the variables are correlated on analysis, and you become indecisive about what to do; hence you lose patience and decide to run a model on the whole data. This returns poor accuracy, and you feel terrible and start thinking of some strategic method to find a few important variables. Principal Component Analysis (PCA) is good at solving this problem, but it struggles when the connections between variables are complicated. That's where Kernel Principal Component Analysis (KPCA) is used. This is the most common scenario in machine learning projects. Statistical techniques such as Kernel principal component analysis (KPCA) help to overcome such difficulties.

1.2 Previous Studies on PCA and KPCA

PCA was invented in 1901 by Karl Pearson (LI, 1901), who formulated the analysis as finding "lines and planes of closest fit to systems of points in space." PCA was briefly mentioned by Fisher and MacKenzie as more suitable than analysis of variance for the modeling of response data. Hotelling further developed PCA to its present stage.

PCA as applied in evaluation of water quality is a statistical analysis method that translates many indexes of water quality into a few comprehensive indexes using a mathematical method. This method can represent most information provided by the original indexes, so as to simplify implementation of water quality monitoring systems. It is a superior comprehensive method for use in evaluation of water quality.

Kernel PCA (KPCA) is a nonlinear form of PCA and at first, it was introduced by Schölkopf et al. (1998). The basic idea behind KPCA is the transformation of original realization space into a new high-dimensional space that is called feature space. This transformation is a nonlinear projection defined by a kernel function and then it would be possible to capture the higher-order statistics in the feature space. Afterward, PCA is applied to projected realizations in the feature space and then the realizations in feature space are projected back onto the original space.

1.3 Principal Component Analysis (PCA)

Principal Component Analysis is an unsupervised machine learning algorithm that is used for the dimensionality reduction. It is a statistical process that converts the correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation. These new transformed features are called the Principal Components. This reduces model complexity as the addition of each new feature negatively impacts model performance. It is one of the popular tools that is used in exploratory data analysis and predictive modeling.

This statistical technique involves both linear algebra and matrix operations, and it transforms the original dataset into a new coordinate system that is structured by the principal components. The eigenvectors and eigenvalues from the covariance matrix that underpin the principal components allow for the analysis of these linear transformations.

In short, it can extract the most informative features from large datasets while preserving the most relevant information from the initial dataset. By projecting a high-dimensional dataset into a smaller feature space, PCA also minimizes, or altogether eliminates, common issues such as multicollinearity and overfitting. Multicollinearity occurs when two or more independent variables are highly correlated with one another, which can be problematic for causal modeling. Overfit models will generalize poorly to new data, diminishing their value altogether.

1.3.1 Theory and Concept

The PCA computation process is summarized in the steps below, showing that how the principal components are calculated and how they relate to the original data.

1. **Standardize the range of continuous initial variables:** Since PCA can bias towards specific features, it is important to evaluate whether normalization of data is needed. Data should reflect a normal distribution with a mean of zero and a standard deviation of one. In this step, the mean values of the variables are calculated and subtracted from the original dataset so that each variable contributes equally to the analysis. This value is then divided by the standard deviation for each variable so that all variables use the same scale.

2. **Compute the covariance matrix to identify correlations:** Covariance (cov) measures how strongly correlated two or more variables are. The covariance matrix summarizes the covariances associated with all pair combinations of the initial variables in the dataset. Computing the covariance matrix helps identify the relationships between the variables–that is, how the variables vary from the mean with respect to each other. This data matrix is a symmetric matrix, meaning the variable combinations can be represented as d × d, where d is the number of dimensions. For example, for a 3-dimensional dataset, there would be 3 × 3 or 9 variable combinations in the covariance matrix.

   The sign of the variables in the matrix tells us whether combinations are correlated:

   - **Positive** (the variables are correlated and increase or decrease at the same time)

- **Negative** (the variables are not correlated, meaning that one decreases while the other increases)
- **Zero** (the variables are not related to each other)

3. **Compute the eigenvectors and eigenvalues of the covariance matrix:** Here, we calculate the eigenvectors (principal components) and eigenvalues of the covariance matrix. As eigenvectors, the principal components represent the directions of maximum variance in the data. The eigenvalues represent the amount of variance in each component. Ranking the eigenvectors by eigenvalue identifies the order of principal components.

4. **Select the principal components:** Here, we decide which components to keep and those to discard. Components with low eigenvalues typically will not be as significant. Scree plots usually plot the proportion of total variance explained and the cumulative proportion of variance. These metrics help one to determine the optimal number of components to retain. The point at which the Y axis of eigenvalues or total variance explained creates an "elbow" will generally indicate how many PCA components that we want to include.

5. **Transform the data into the new coordinate system** Finally, the data is transformed into the new coordinate system defined by the principal components. That is, the feature vector created from the eigenvectors of the covariance matrix projects the data onto the new axes defined by the principal components. This creates new data, capturing most of the information but with fewer dimensions than the original dataset.

1.3.2 Where PCA is useful?

Applying PCA can help preprocess or extract the most informative features from datasets with many variables. Preprocessing reduces complexity while preserving relevant information. Common scenarios that use PCA include:

1. **Image compression:** PCA reduces image dimensionality while retaining essential information. It helps create compact representations of images, making them easier to store and transmit.

2. **Data visualization:** PCA helps to visualize high-dimensional data by projecting it into a lower-dimensional space, such as a 2D or 3D plot. This simplifies data interpretation and exploration.

3. **Noise filtering:** PCA can remove noise or redundant information from data by focusing on the principal components that capture the underlying patterns.

4. **Feature Extraction:** PCA extracts the most significant features from the data by identifying the directions of maximum variance. These principal components can be used as input features for subsequent machine learning algorithms, reducing computational complexity and improving model performance.

5. **Clustering and Classification:** PCA can improve clustering and classification performance by reducing the dimensionality of the feature space. By removing redundant or correlated features, PCA can enhance the discriminative power of clustering and classification algorithms, leading to more accurate results.

1.3.3 Limitations of PCA

1. PCA is at a disadvantage if the data has not been standardized before applying the algorithm to it. PCA transforms original data into data that is relevant to the principal components of that data, which means that the new data variables cannot be interpreted in the same ways that the originals were. They are linear interpretations of the original variables. Also, if PCA is not performed properly, there is a high likelihood of information loss.

2. PCA relies on a linear model. If a dataset has a pattern hidden inside it that is nonlinear, then PCA can actually steer the analysis in the complete opposite direction of progress.

1.4 Kernel Principal Component Analysis (KPCA)

Kernel PCA (KPCA) is indeed a nonlinear extension of Principal Component Analysis (PCA) that utilizes kernel functions to transform data into higher-dimensional spaces, where it becomes easier to separate classes or identify patterns.

It is an extension of the classical Principal Component Analysis (PCA) algorithm, which is a linear method that identifies the most significant features or components of a dataset. KPCA applies a nonlinear mapping function to the data before applying PCA, allowing it to capture more complex and nonlinear relationships between the data points. KPCA can also handle high-dimensional datasets with many features by reducing the dimensionality of the data while preserving the most important information.

1.4.1 Mercer's Theorem

Mercer's Theorem states that: "If a kernel function K is symmetric, continuous, and leads to a positive semi-definite matrix P, then there exists a function ϕ that maps x_i and x_j into another space (possibly with much higher dimensions) such that

$$K(x_i, x_j) = \phi(x_i)^T \phi(x_j).$$"

In Kernel PCA, Mercer's theorem is applied during the construction of the kernel matrix and subsequent eigenvalue decomposition. Kernel matrix K, tells us that each entry K_{ij} represents the similarity between data points x_i and x_j. The kernel function $K(x_i, x_j)$ plays a crucial role here. Mercer's theorem ensures that the chosen kernel function satisfies the necessary conditions for positive semi-definiteness. This ensures that the kernel matrix is symmetric and positive semi-definite, which is essential for performing eigenvalue decomposition.

After constructing the kernel matrix K, the next step is to perform eigenvalue decomposition on K to find the principal components of the dataset. Mercer's theorem guarantees that the eigenvalues obtained from this decomposition are real and non-negative, and the corresponding eigenvectors can be used to project the data into a lower-dimensional space. Mercer's theorem is needed in Kernel PCA because it helps us transform data in a way that allows us to find patterns and relationships that might be hidden in its original form.

1.4.2 Theory and Concept

1. **Data Representation**

   Let the set of n data points be represented by

   $$X = \{x_1, x_2, \ldots, x_n\},$$

   where $x_i \in \mathbb{R}^d$, $i = 1, \ldots, n$. It is assumed that the data in X is centered in the original space $\mathbb{R}^d$, so that

   $$\sum_{i=1}^{n} x_i = 0$$

   This condition can be met by using the formula

   $$\tilde{x}_k = x_k - \frac{1}{n} \sum_{i=1}^{n} x_i,$$

   where $\tilde{x}_k$ is the k-th data point in the new, centered set.

2. **Mapping to Feature Space (F)**

   Let Φ be a mapping from the original space $\mathbb{R}^d$ of the data vectors, into an inner product space F, so that

   $$\Phi : \mathbb{R}^d \to F,$$

   where the space F is referred to as the feature space. Thus, $\Phi(x_i)$ represents the image of the data vector $x_i \in \mathbb{R}^d$ in feature space. The kernel matrix K is constructed using inner products of the mapped data points in the feature space.

   $$K_{ij} = (\Phi(x_i) \cdot \Phi(x_j)), \quad i, j = 1, \ldots, n.$$

3. **Kernel Matrix (K)**

   Interestingly, it is not necessary to know the values of $\Phi(x_i)$ and $\Phi(x_j)$ explicitly in order to compute the kernel matrix K. In other words, in order to map the data points into F, the mapping Φ is not required to be known. Instead, the elements of K can be calculated by computing the pairwise relation of all points $x_i, x_j \in X$ in the original space $\mathbb{R}^d$. This is motivated by Mercer's Theorem and done through a kernel function $k(x_i, x_j)$, so that

   $$K_{ij} = k(x_i, x_j), \quad i, j = 1, \ldots, n.$$

   This way the features space F and the mapping Φ are implicitly defined by the kernel function that is used. This is known as "the kernel trick" and is the perhaps most central part of the kernel PCA method.

4. **Centering in Feature Space**

   The data points represented in the kernel matrix K are assumed to be centered in feature space, in the sense that

   $$\sum_{i=1}^{n} \Phi(x_i) = 0.$$

5. **Eigenvectors and Eigenvalues**

   The eigenvectors of the centered kernel matrix $\tilde{K}$ can be obtained by solving the equation

   $$\tilde{K}\mathbf{V} = \Lambda\mathbf{V},$$

   where the columns of $\mathbf{V}$ represent the normalized eigenvectors and Λ is a matrix in which the diagonal consists of the corresponding eigenvalues and all other elements in Λ are zero. The eigenvectors in $\mathbf{V}$ are assumed to be ordered according to the size of their corresponding eigenvalues, in descending order. Also the eigenvalues in Λ are sorted in descending order.

6. **Projection to Low-Dimensional Space**

   The representation of the low-dimensional points $\{\chi_1, \ldots, \chi_n\}$, with $\chi_i \in \mathbb{R}^l$, $i = 1, \ldots, n$, can be obtained by projecting the kernel matrix $\tilde{K}$ onto the l eigenvectors that have the largest corresponding eigenvalues, by using the formula

   $$\chi_i = \tilde{K}_i\mathbf{V}_l,$$

   where $\tilde{K}_i$ is the i-th row of $\tilde{K}$, and $\mathbf{V}_l$ is the matrix consisting of the l first eigenvector columns in $\mathbf{V}$.

1.4.3 Different Kernel Functions

The popular kernels include:

- **Linear kernel:**

  $$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^\top \mathbf{x}_j$$

- **Polynomial kernel:**

  $$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^\top \mathbf{x}_j + c)^d$$

  where $c \geq 0$ and d is the degree of the polynomial, usually greater than 2.

- **RBF (Radial Basis Function) kernel:** Sometimes referred to as the Gaussian kernel.

  $$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma \|\|\mathbf{x}_i - \mathbf{x}_j\|\|^2\right)$$

  where $\gamma = \frac{1}{2\sigma^2}$.

- **Sigmoid kernel:**

  $$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh\left((\mathbf{x}_i^\top \mathbf{x}_j + c)\right)$$

  where $c \geq 0$ and $\gamma = \frac{1}{2\sigma^2}$.

- **Cosine kernel:**

  $$K(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{x}_i^\top \mathbf{x}_j}{\|\|\mathbf{x}_i\|\|\|\|\mathbf{x}_j\|\|}$$

1.4.4 Why KPCA?

Some of the advantages of kernel PCA over traditional PCA are:

- **Nonlinear transformation** – It has the ability to capture complex and nonlinear relationships
- **Higher-dimensional transformation** – By mapping data into a higher-dimensional space, kernel PCA can create a more expressive representation, potentially leading to better separation of classes or clusters
- **Flexibility** – By capturing nonlinear patterns, it's more flexible and adaptable to various data types. Thus, kernel PCA is used for many domains, including image recognition and speech processing

1.5 Gradient Boosting Algorithm

Boosting is a powerful ensemble technique in machine learning. Unlike traditional models that learn from the data independently, boosting combines the predictions of multiple weak learners to create a single, more accurate strong learner. Boosting is one kind of ensemble Learning method which trains the model sequentially and each new model tries to correct the previous model. It combines several weak learners into strong learners.

Gradient Boosting is an ensemble technique. It is primarily used in classification and regression tasks. It provides a forecast model consisting of a collection of weaker prediction models, mostly decision trees. Gradient Boosting is mainly of two types depending on the target columns:

1. **Gradient Boosting Regressor:** It is used when the columns are continuous.
2. **Gradient Boosting Classifier:** It is used when the target columns are categorical.

1.5.1 How Gradient Boosting Works?

Gradient boosting involves three elements:

1. A loss function to be optimized.
2. A weak learner to make predictions.
3. An additive model to add weak learners to minimize the loss function.

1. **Loss Function:** The loss function depends on the type of problem being solved. For example, regression may use a squared error and classification may use logarithmic loss.
2. **Weak Learner:** Decision trees are used as the weak learner in gradient boosting.

   Specifically regression trees are used that output real values for splits and whose output can be added together, allowing subsequent models outputs to be added and "correct" the residuals in the predictions.

Trees are constructed in a greedy manner, choosing the best split points based on purity scores like Gini or to minimize the loss.

Initially, such as in the case of AdaBoost, very short decision trees were used that only had a single split, called a decision stump. Larger trees can be used generally with 4-to-8 levels.

It is common to constrain the weak learners in specific ways, such as a maximum number of layers, nodes, splits or leaf nodes.

This is to ensure that the learners remain weak, but can still be constructed in a greedy manner.

3. **Additive Model:** Trees are added one at a time, and existing trees in the model are not changed.

   A gradient descent procedure is used to minimize the loss when adding trees.

   Traditionally, gradient descent is used to minimize a set of parameters, such as the coefficients in a regression equation or weights in a neural network. After calculating error or loss, the weights are updated to minimize that error.

   Instead of parameters, we have weak learner sub-models or more specifically decision trees. After calculating the loss, to perform the gradient descent procedure, we must add a tree to the model that reduces the loss (i.e. follow the gradient). We do this by parameterizing the tree, then modify the parameters of the tree and move in the right direction by (reducing the residual loss.

   Generally this approach is called functional gradient descent or gradient descent with functions.

   The output for the new tree is then added to the output of the existing sequence of trees in an effort to correct or improve the final output of the model.

   A fixed number of trees are added or training stops once loss reaches an acceptable level or no longer improves on an external validation dataset.

Chapter 2

Nonlinear Dimensionality Reduction: Performance Comparison of PCA and KPCA in Gradient Boosting Models

2.1 Problem Statement

In today's competitive business environment, companies face significant pressure to operate efficiently, especially in marketing. To tackle this challenge, create a predictive model that uses customer demographics—age, gender, and salary—to pinpoint potential buyers likely to make purchases. Emphasize the importance of Kernel Principal Component Analysis (KPCA) in detecting non linear patterns within marketing data. By adopting KPCA, businesses can gain valuable insights, improve how they target customers, and reduce unnecessary spending, leading to a more streamlined and cost-effective marketing strategy.

2.1.1 Cost of Inefficient Targeting

1. **Advertising Costs:** Businesses often pay for advertising space, whether it's through traditional mediums like TV, radio, or print, or through online channels like social media platforms or websites. When ads are shown to individuals who are less likely to convert, these advertising expenditures may not yield a return on investment (ROI), resulting in wasted funds.

2. **Opportunity Cost:** Resources spent on advertising to less promising customer segments could have been allocated elsewhere to activities with higher potential for generating revenue, such as targeting more promising customer segments or investing in product development or customer service improvements.

3. **Efficiency Loss:** Inefficient use of resources can lead to lower overall efficiency in marketing efforts. Instead of focusing on reaching out to individuals who are more likely to convert, businesses may end up spreading their resources thinly across a broad audience, diluting the impact of their marketing campaigns.

4. **Reputation Damage:** Excessive or irrelevant advertising can also have negative effects on a business's reputation. Bombarding individuals with ads that are not relevant to their interests or needs can lead to annoyance or frustration, potentially harming the brand's image and driving customers away.

By leveraging predictive modeling techniques like KPCA to identify and target the most promising customer segments, businesses can minimize these risks and ensure that their advertising efforts are more focused, efficient, and cost-effective. This ultimately leads to better utilization of resources and improved ROI on marketing investments.

2.1.2 How this project is useful for businesses?

This project offers several benefits for businesses:

1. **Targeted Marketing:** By accurately identifying potential customers who are more likely to make purchases, businesses can tailor their marketing campaigns and promotions to focus on these high-conversion prospects. This targeted approach ensures that marketing resources are allocated efficiently, leading to higher returns on investment.

2. **Cost Reduction:** Through the utilization of Kernel Principal Component Analysis (KPCA), the project enables businesses to uncover intricate nonlinear patterns within their marketing datasets. By understanding these patterns, organizations can refine their customer targeting strategies, optimizing their marketing efforts and reducing unnecessary expenditure on ineffective advertising channels.

3. **Improved Sales Performance:** By deploying predictive models based on customer demographic attributes, businesses can enhance their sales performance. By concentrating their efforts on customers with a higher likelihood of purchasing, companies can increase conversion rates and boost overall sales revenue.

4. **Competitive Advantage:** Implementing advanced analytical techniques like KPCA provides businesses with a competitive edge. By leveraging cutting-edge methods for targeting, organizations can stay ahead of competitors and adapt more effectively to changes in consumer behavior.

5. **Data-Driven Decision Making:** This project promotes a data-driven approach to marketing decision-making. By relying on insights derived from the predictive model, businesses can make informed choices regarding resource allocation, campaign optimization, and overall marketing strategy, leading to more successful outcomes.

Overall, this project empowers businesses to optimize their marketing efforts, reduce costs, improve sales performance, and gain a competitive advantage in the marketplace.

2.2 Overview of "Social-Networking Ads" dataset

I obtained this dataset from Kaggle https://www.kaggle.com/datasets/akram24/social-network-ads , a popular platform for data science competitions and datasets. The dataset contains information about users who are potentially targeted for social networking ads. Researchers and data scientists can utilize this dataset for various analyses, including exploratory data analysis (EDA), dimensionality reduction, and predictive modeling. Here's a brief overview:

- **User ID:** A unique identifier for each user, likely serving as the primary key.
- **Gender:** Indicates the gender of the user, with values being either male or female.
- **Age:** Represents the age of the user, provided as numerical values.
- **Estimated Salary:** Indicates the estimated salary of the user, also provided as numerical values. Purchased: A binary variable indicating whether the user made a purchase after being exposed to social networking ads.

2.3 Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is an essential step in the data analysis process, allowing us to understand the structure and characteristics of the dataset. In this section, we'll perform EDA on the "Social-Networking Ads" dataset to gain insights into user demographics and purchase behavior.

```
#Importing relevant libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA, KernelPCA
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix, f1_score, precision_score, recall_score
```

We begin by importing essential libraries required for exploratory data analysis and subsequent analysis. The pandas library is utilized for data manipulation and analysis, while numpy provides support for numerical operations. Visualizations are facilitated by matplotlib.pyplot and seaborn, allowing us to create insightful plots to understand the dataset's characteristics better. Additionally, we import modules from the scikit-learn library, including `train_test_split` for splitting the data into training and testing sets, StandardScaler for feature scaling, PCA and KernelPCA for dimensionality reduction techniques, and GradientBoostingClassifier for building our machine learning model. Evaluation metrics such as accuracy, precision, recall, and F1-score are computed using functions from sklearn.metrics.

```
# Load the dataset
data = pd.read_csv("Social_Network_Ads.csv")
```

```
# The code below prints the first few rows of the dataframe 'data'
data.head()
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

This code snippet serves as the foundational step in the data preprocessing phase of the project. It imports the dataset "Social_Network_Ads.csv" into a pandas DataFrame named 'data', facilitating subsequent data manipulation and analysis. The utilization of the .head() method allows for a quick initial exploration of the dataset by displaying the first few rows, enabling a preliminary understanding of its structure and contents. This systematic approach aids in establishing a robust analytical framework for the subsequent stages of the project, aligning with best practices in data science and statistical analysis.

```
# The code below retrieves the shape of the dataframe 'data'
data.shape
```

```
(400, 5)
```

```
# The code below retrieves the total number of elements in the dataframe 'data'
data.size
```

```
2000
```

This code segment demonstrates two fundamental operations applied to the pandas DataFrame named 'data'. The first line utilizes the .shape attribute to retrieve the dimensions of the DataFrame, returning a tuple representing the number of rows and columns. In this instance, the output "(400, 5)" indicates that the DataFrame contains 400 rows and 5 columns. This operation is pivotal for understanding the dataset's overall size and structure, providing essential context for subsequent analyses.

The second line showcases the .size attribute, which calculates the total number of elements within the DataFrame by multiplying the number of rows by the number of columns. The output "2000" represents the total count of elements present in the DataFrame 'data'. This operation complements the .shape attribute by offering a comprehensive perspective on the dataset's scale, aiding in resource allocation and computational planning for data processing tasks.

```
# The code below provides concise summary information about the dataframe 'data'
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   User ID          400 non-null    int64
 1   Gender           400 non-null    object
 2   Age              400 non-null    int64
 3   EstimatedSalary  400 non-null    int64
 4   Purchased        400 non-null    int64
dtypes: int64(4), object(1)
memory usage: 15.8+ KB
```

The output generated by the code snippet using the .info() method delivers a concise yet comprehensive overview of the DataFrame 'data'. It begins by indicating that the DataFrame consists of 400 entries indexed from 0 to 399, denoting the range of indices utilized. The summary then proceeds to enumerate the five data columns present in the DataFrame, detailing various attributes for each. For instance, it reveals that all columns contain 400 non-null values, suggesting that there are no missing or empty entries within the dataset. Moreover, it specifies the data type for each column, with four columns represented as integers (int64) and one column as an

object (likely indicating categorical or string data). The memory usage information indicates that the DataFrame consumes approximately 15.8 KB of memory, which is valuable for optimizing memory allocation, especially when dealing with larger datasets. Overall, this summary provides essential insights into the dataset's structure, completeness, and memory utilization, laying a solid foundation for subsequent data analysis and manipulation tasks.

```
# The code below generates descriptive statistics for the dataframe 'data'
data.describe()
```

	User ID	Age	EstimatedSalary	Purchased
count	4.000000e+02	400.000000	400.000000	400.000000
mean	1.569154e+07	37.655000	69742.500000	0.357500
std	7.165832e+04	10.482877	34096.960282	0.479864
min	1.556669e+07	18.000000	15000.000000	0.000000
25%	1.562676e+07	29.750000	43000.000000	0.000000
50%	1.569434e+07	37.000000	70000.000000	0.000000
75%	1.575036e+07	46.000000	88000.000000	1.000000
max	1.581524e+07	60.000000	150000.000000	1.000000

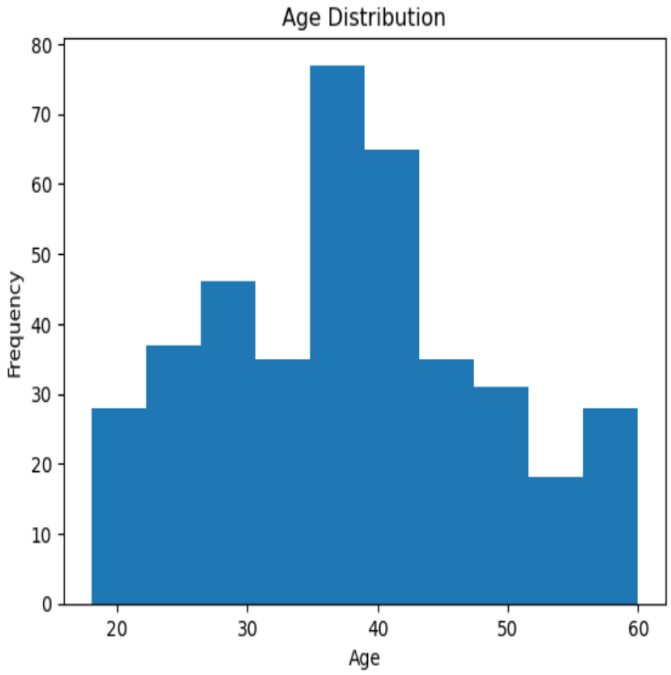
The code snippet utilizes the .describe() method to generate descriptive statistics for the DataFrame 'data', offering valuable insights into its numerical attributes. The resulting output presents a tabulated summary consisting of various statistical metrics for each numeric column in the DataFrame.

For the 'User ID' column, the statistics are represented in scientific notation, indicating the count of entries (400), mean value (1.569154e+07), standard deviation (7.165832e+04), minimum value (1.556669e+07), 25th percentile (1.562676e+07), median (1.569434e+07), 75th percentile (1.575036e+07), and maximum value (1.581524e+07).

Similarly, the 'Age', 'EstimatedSalary', and 'Purchased' columns each have their respective statistics displayed. These include the count of non-null entries, mean value, standard deviation, minimum and maximum values, as well as the 25th, 50th (median), and 75th percentiles.

```
# Age distribution
data['Age'].hist()
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.grid(False)  # Remove grid lines
plt.show()
```

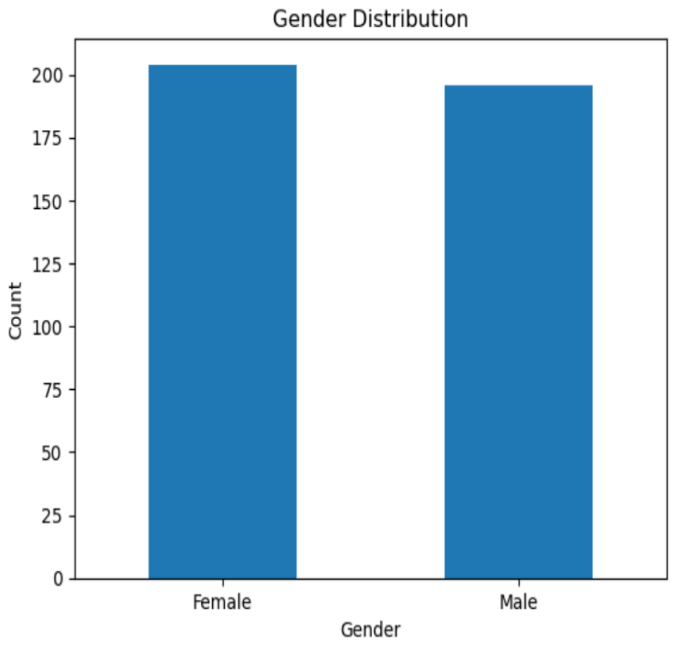
By calling the .hist() method on the 'Age' column, a histogram is generated, providing insights into the frequency or count of individuals belonging to different age groups. The subsequent lines of code set titles for the plot, label the axes appropriately, and remove grid lines for clarity. Ultimately, this visualization offers a clear and concise representation of the age distribution within the dataset, aiding in exploratory data analysis and understanding demographic patterns.



The graph made from the code shows that most people in the dataset are between 35 and 42 years old. This tells us that there's a big group of people in that age range compared to others. Knowing this helps with things like deciding what products to sell or which ads to show because it tells you who might be interested. It's like having a map that guides you to understand the age group of the people you're dealing with, helping you make smarter decisions in different areas like marketing or research.

```
# Gender distribution
data['Gender'].value_counts().plot(kind='bar', rot=0)
plt.title('Gender Distribution')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.show()
```

This code creates a bar plot to visualize the distribution of genders within the dataset stored in the 'Gender' column. By calling the .value_counts() method on the 'Gender' column, it calculates the frequency of each gender category. The resulting counts are then plotted as bars using .plot(kind='bar', rot=0), with the 'rot=0' parameter ensuring that the x-axis labels are not rotated. The plot is titled 'Gender Distribution', with appropriate labels for the x-axis ('Gender') and y-axis ('Count'). This visualization provides a clear representation of the gender distribution in the dataset, facilitating insights into demographic proportions and aiding decision-making processes in areas such as marketing segmentation or diversity analysis.

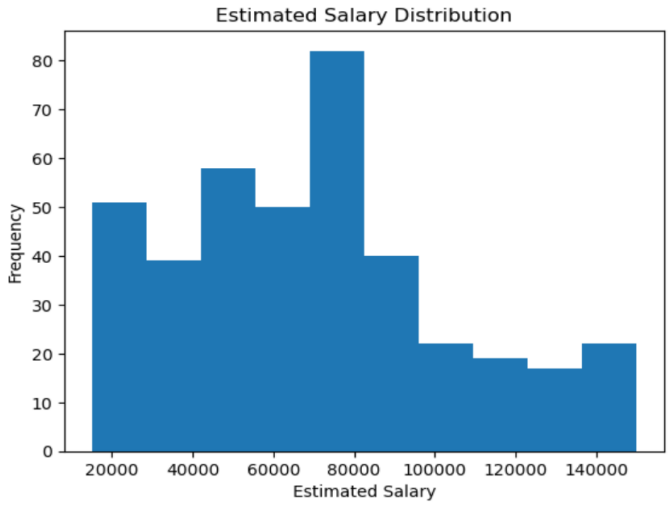


The bar plot generated from the code depicts a relatively balanced gender distribution within the dataset, with both females and males accounting for around 200 occurrences each. Although there is a slight difference in counts, with females slightly outnumbering males, the overall distribution remains fairly equitable. This observation suggests that the dataset exhibits a relatively equal representation of genders, without any significant skew towards one gender over the other. For instance, it could reflect a fair representation of gender demographics in a social context or indicate inclusive data collection practices. Furthermore, understanding this gender distribution can inform targeted strategies in marketing, product development, or policy-making to ensure inclusivity and effectiveness across diverse gender groups.

```
# The code below creates a histogram of the 'EstimatedSalary' column in the dataframe 'data'
data['EstimatedSalary'].hist()

# Adding title to the histogram
plt.title('Estimated Salary Distribution')

# Adding label to the x-axis
plt.xlabel('Estimated Salary')

# Adding label to the y-axis
plt.ylabel('Frequency')

# Turning off grid lines
plt.grid(False)

# Displaying the histogram
plt.show()
```

The provided code segment generates a histogram representing the distribution of estimated salaries within the dataset, focusing on the 'EstimatedSalary' column of the DataFrame 'data'. The histogram visually displays the frequency or count of individuals falling within specific salary ranges, offering insights into the salary distribution across the dataset. Additionally, the code includes annotations to enhance the clarity and interpretability

of the histogram. A title, 'Estimated Salary Distribution', provides description of the plotted data, aiding in immediate comprehension. Axis labels are added to denote the salary values on the x-axis ('Estimated Salary') and the corresponding frequency of occurrence on the y-axis ('Frequency'). Furthermore, grid lines are disabled ('plt.grid(False)') to declutter the visualization and improve readability. Overall, this histogram serves as a valuable exploratory tool, facilitating the analysis of salary trends and patterns within the dataset.




```
X = data.drop(columns=['User ID', 'Purchased'])  # Drop User ID as it's not useful for modeling
y = data['Purchased']
```

```
# Perform one-hot encoding for the 'Gender' column
data_encoded = pd.get_dummies(data, columns=['Gender'], drop_first=True)
```

```
#Show first five rows of new dataframe
data_encoded.head()
```

	User ID	Age	EstimatedSalary	Purchased	AgeGroup	Gender_Male
0	15624510	19	19000	0	18-24	True
1	15810944	35	20000	0	25-34	True
2	15668575	26	43000	0	25-34	False
3	15603246	27	57000	0	25-34	False
4	15804002	19	76000	0	18-24	True

This code block serves several purposes:

1. **Separating Features and Target Variable:** In most machine learning tasks, we need to separate the input features from the target variable. Here, the features are stored in X, and the target variable (indicating whether the user purchased or not) is stored in y.

2. **One-Hot Encoding for Categorical Variable:** Machine learning algorithms typically require numerical input data. Since 'Gender' is a categorical variable, it needs to be converted into numerical format for modeling. One-hot encoding converts categorical variables into binary vectors, making them suitable for machine learning algorithms.

3. **Dropping Unnecessary Columns:** The 'User ID' column is likely just an identifier and doesn't contain useful predictive information for modeling. Similarly, we remove the 'Purchased' column from the features since it's what we want to predict.

4. **Showing First Five Rows of the New DataFrame:** This allows us to visually inspect the transformed DataFrame to ensure that the one-hot encoding was performed correctly and to get a quick overview of the data.

```
# Specify x and y
X = data_encoded[['Gender_Male', 'Age', 'EstimatedSalary']]
y = data_encoded['Purchased']
```

```
# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

This code block further prepares the data for modeling by performing the following steps:

1. **Specifying Features and Target Variable:** These features are selected based on their potential relevance to predicting whether a customer will make a purchase. 'Gender_Male', 'Age', and 'EstimatedSalary' are commonly considered factors that can influence purchasing behavior.

2. **Splitting Data into Train and Test Sets:** Splitting the data into separate training and testing sets allows for the evaluation of the model's performance on unseen data. The training set is used to train the model, while the testing set is used to assess its performance.

3. **Standardizing the Features:** Standardization ensures that all features are on the same scale, which can be important for certain machine learning algorithms, such as those based on distance metrics. Standardizing the features helps the algorithm converge faster and can improve the model's performance.

```
# PCA
pca = PCA(n_components=2)  # You can adjust the number of components as needed
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)
```

This code block applies Principal Component Analysis (PCA) to the standardized features:

PCA is used to reduce the dimensionality of the feature space while preserving the most important information. By specifying the number of components, we determine the dimensionality of the transformed feature space. These transformed features contain the principal components extracted from the original features. This transformation reduces the computational complexity and can improve the performance of machine learning algorithms.

Principal Component Analysis (PCA) is applied to the feature space, not the target variable space. PCA aims to find the directions (principal components) that capture the most variance in the feature space. It's used to reduce

the dimensionality of the feature space while retaining most of the information. Since the target variable is not part of the feature space but rather represents the outcome we want to predict, PCA does not apply to it. The target variable typically represents discrete categories or values that indicate the classes or labels of the data (e.g., whether a customer purchased a product or not). Unlike the continuous feature space, the target variable does not exhibit variance that PCA aims to capture.

In summary, PCA is applied to the features (X) to reduce dimensionality and capture the most important information, while the target variable (y) remains unchanged as it represents the outcome we want to predict.

```
# Kernel PCA
kpca = KernelPCA(n_components=2, kernel='rbf')  # You can experiment with different kernels
X_train_kpca = kpca.fit_transform(X_train_scaled)
X_test_kpca = kpca.transform(X_test_scaled)
```

In this code block, Kernel Principal Component Analysis (KPCA) is applied to the standardized features:

KPCA is a nonlinear extension of PCA that can capture complex relationships in the data. The radial basis function (RBF) kernel is commonly used in KPCA for its flexibility in capturing nonlinear patterns. Transforming Training and Testing Data: Similar to PCA, these transformed features contain the principal components extracted from the original features. However, KPCA applies a kernel function to map the data into a higher-dimensional space before computing the principal components, allowing it to capture nonlinear relationships that PCA cannot. Overall, this code block applies KPCA to the standardized features, reducing their dimensionality while capturing nonlinear patterns in the data. The transformed features can then be used as input for training machine learning models. KPCA offers a powerful alternative to PCA when dealing with datasets containing nonlinear relationships..

```
# Train Gradient Boosting model
gradboost_pca = GradientBoostingClassifier()
gradboost_pca.fit(X_train_pca, y_train)

gradboost_kpca = GradientBoostingClassifier()
gradboost_kpca.fit(X_train_kpca, y_train)

# Predictions
y_pred_pca_gb = gradboost_pca.predict(X_test_pca)
y_pred_kpca_gb = gradboost_kpca.predict(X_test_kpca)
```

This code block trains two Gradient Boosting models:

1. **Training Gradient Boosting Model with PCA-transformed Data:** This model learns to make predictions based on the principal components extracted from the original features using PCA. It captures the most important information in the data while reducing dimensionality.

2. **Training Gradient Boosting Model with KPCA-transformed Data:** This model learns to make predictions based on the principal components extracted from the original features using KPCA, which can capture nonlinear patterns in the data that PCA might miss.

3. **Making Predictions:** These predictions allow for the evaluation of each model's performance on unseen data. By comparing the predictions made by the models trained on PCA- and KPCA-transformed data, you

can assess which dimensionality reduction technique yields better predictive performance.

Overall, this code block trains two Gradient Boosting models using PCA- and KPCA-transformed data, respectively, and generates predictions to evaluate their performance.

```
# Evaluate models
accuracy_pca_gb = accuracy_score(y_test, y_pred_pca_gb)
accuracy_kpca_gb = accuracy_score(y_test, y_pred_kpca_gb)
```

```
print("Accuracy with PCA:", accuracy_pca_gb)
print("Accuracy with kPCA:", accuracy_kpca_gb)
```

```
Accuracy with PCA: 0.825
Accuracy with kPCA: 0.925
```

The code block evaluates the performance of the Gradient Boosting models trained on PCA and KPCA-transformed data by computing their accuracy scores on the testing set. Here's what the output indicates:

The accuracy of the Gradient Boosting model trained on PCA-transformed data is 82.5%. This means that the model correctly predicts the target variable (whether a customer purchased or not) for approximately 82.5% of the observations in the testing set. The accuracy of the Gradient Boosting model trained on KPCA-transformed data is 92.5%. This indicates that the model trained on KPCA-transformed data performs better, achieving a higher accuracy of 92.5% on the testing set compared to the PCA-transformed model.

In summary, the output suggests that the model trained on KPCA-transformed data outperforms the model trained on PCA-transformed data in terms of accuracy. This aligns with our earlier findings where we observed that KPCA outperformed PCA, indicating that KPCA might be more effective in capturing the underlying patterns in the data.

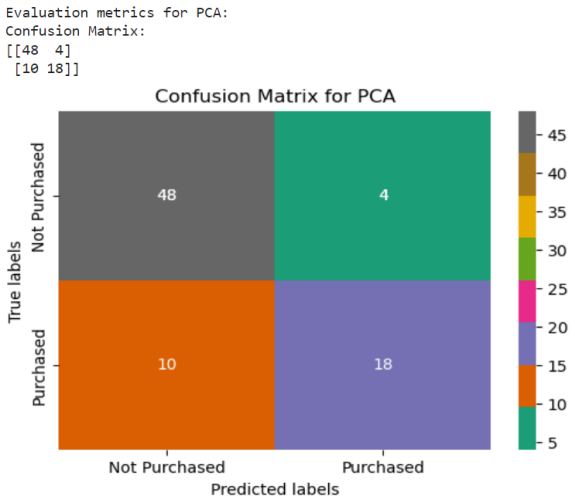
```
# Define labels for clarity
labels = ['Not Purchased', 'Purchased']

# Function to dynamically adjust font color based on background color
def get_text_color(value):
    return 'white' if value < 50 else 'black'  # Adjust the threshold as needed

# Evaluate PCA model
print("Evaluation metrics for PCA:")
print("Confusion Matrix:")
conf_matrix_pca = confusion_matrix(y_test, y_pred_pca_gb)
print(conf_matrix_pca)
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix_pca, annot=True, cmap='Dark2', fmt='g', cbar=True, xticklabels=labels, yticklabels=labels)
# Manually annotate the heatmap with values and adjust font color
for i in range(len(conf_matrix_pca)):
    for j in range(len(conf_matrix_pca[i])):
        plt.text(j+0.5, i+0.5, str(conf_matrix_pca[i][j]),\
        ha='center', va='center', fontsize=10, color=get_text_color(conf_matrix_pca[i][j]))
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix for PCA')
plt.show()
```

This code block visualizes the evaluation metrics for the Gradient Boosting model trained on PCA-transformed data, specifically focusing on the confusion matrix:

The line defines labels for the confusion matrix to clarify which class corresponds to "Not Purchased" and "Purchased." The get_text_color function dynamically adjusts the font color based on the background color of

each cell in the heatmap. This function helps improve readability by ensuring that text remains visible against both light and dark backgrounds. Evaluation Metrics for PCA Model: Purpose: The next code block evaluates the performance of the Gradient Boosting model trained on PCA-transformed data. By examining the confusion matrix, we can assess how well the model predicts each class ("Not Purchased" and "Purchased") and identify any potential misclassifications. The sns.heatmap function creates a heatmap visualization of the confusion matrix. Heatmaps provide a visually intuitive way to represent the confusion matrix, making it easier to interpret the distribution of true positive, false positive, true negative, and false negative predictions. The plt.text function manually annotates the heatmap with the values from the confusion matrix. Adding text labels to each cell of the heatmap enhances its interpretability by displaying the exact counts of true positive, false positive, true negative, and false negative predictions. Various plot customization options such as axis labels, title, and color map settings are applied to enhance the visual presentation of the confusion matrix. Customizing the plot ensures that it effectively communicates the evaluation metrics for the PCA model in a clear and informative manner.

Overall, this code block provides a comprehensive visualization of the evaluation metrics for the PCA model, allowing for a detailed analysis of its performance in predicting the target variable classes.



The output shows the evaluation metrics for the Gradient Boosting model trained on PCA-transformed data, specifically focusing on the confusion matrix:

- **True Negatives (TN):** The value 48 in the top-left corner of the confusion matrix represents the number of observations where the model correctly predicted "Not Purchased" (the negative class) and the actual label was also "Not Purchased".
- **False Positives (FP):** The value 4 in the top-right corner represents the number of observations where the model incorrectly predicted "Purchased" (the positive class) but the actual label was "Not Purchased".
- **False Negatives (FN):** The value 10 in the bottom-left corner represents the number of observations where the model incorrectly predicted "Not Purchased" but the actual label was "Purchased".

- **True Positives (TP):** The value 18 in the bottom-right corner represents the number of observations where the model correctly predicted "Purchased" and the actual label was also "Purchased".

In summary:

The model correctly predicted "Not Purchased" for 48 observations.

The model incorrectly predicted "Purchased" for 4 observations that were actually "Not Purchased".

The model incorrectly predicted "Not Purchased" for 10 observations that were actually "Purchased".

The model correctly predicted "Purchased" for 18 observations.

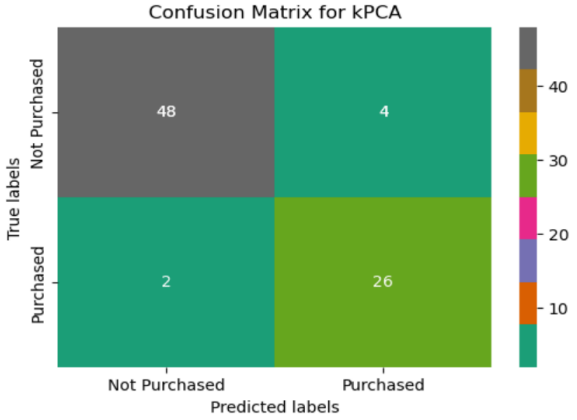
```
# Evaluate KPCA model
print("\nEvaluation metrics for kPCA:")
print("Confusion Matrix:")
conf_matrix_kpca = confusion_matrix(y_test, y_pred_kpca_gb)
print(conf_matrix_kpca)
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix_kpca, annot=True, cmap='Dark2', fmt='g', cbar=True, xticklabels=labels, yticklabels=labels)
# Manually annotate the heatmap with values and adjust font color
for i in range(len(conf_matrix_kpca)):
    for j in range(len(conf_matrix_kpca[i])):
        plt.text(j+0.5, i+0.5, str(conf_matrix_kpca[i][j]),\
        ha='center', va='center', fontsize=10, color=get_text_color(conf_matrix_kpca[i][j]))
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix for kPCA')
plt.show()
```

This code block evaluates the performance of the Gradient Boosting model trained on KPCA-transformed data and visualizes the evaluation metrics using a confusion matrix.

By examining the confusion matrix, we can assess how well the model predicts each class ("Not Purchased" and "Purchased") based on the KPCA-transformed features. The sns.heatmap function creates a heatmap visualization of the confusion matrix for the KPCA model. Heatmaps provide a visually intuitive way to represent the confusion matrix, making it easier to interpret the distribution of true positive, false positive, true negative, and false negative predictions. The plt.text function manually annotates the heatmap with the values from the confusion matrix for the KPCA model. Adding text labels to each cell of the heatmap enhances its interpretability by displaying the exact counts of true positive, false positive, true negative, and false negative predictions. Various plot customization options such as axis labels, title, and color map settings are applied to enhance the visual presentation of the confusion matrix for the KPCA model. Customizing the plot ensures that it effectively communicates the evaluation metrics for the KPCA model in a clear and informative manner.

Overall, this code block provides a comprehensive visualization of the evaluation metrics for the KPCA model, allowing for a detailed analysis of its performance in predicting the target variable classes.

```
Evaluation metrics for kPCA:
Confusion Matrix:
[[48  4]
 [ 2 26]]
```



The output shows the evaluation metrics for the Gradient Boosting model trained on KPCA-transformed data, focusing on the confusion matrix:

- **True Negatives (TN):** The value 48 in the top-left corner of the confusion matrix represents the number of observations where the model correctly predicted "Not Purchased" (the negative class) and the actual label was also "Not Purchased".
- **False Positives (FP):** The value 4 in the top-right corner represents the number of observations where the model incorrectly predicted "Purchased" (the positive class) but the actual label was "Not Purchased".
- **False Negatives (FN):** The value 2 in the bottom-left corner represents the number of observations where the model incorrectly predicted "Not Purchased" but the actual label was "Purchased".
- **True Positives (TP):** The value 26 in the bottom-right corner represents the number of observations where the model correctly predicted "Purchased" and the actual label was also "Purchased".

In summary:

_The model correctly predicted "Not Purchased" for 48 observations.

_The model incorrectly predicted "Purchased" for 4 observations that were actually "Not Purchased".

_The model incorrectly predicted "Not Purchased" for 2 observations that were actually "Purchased".

_The model correctly predicted "Purchased" for 26 observations.

2.4 Comparison with the confusion matrix for the PCA model:

The KPCA model has fewer false negatives (2 vs. 10) and more true positives (26 vs. 18), indicating that it performs better at correctly identifying purchasers. However, the KPCA model has the same number of false positives (4) and true negatives (48) as the PCA model.

```python
#Evaluate PCA model
print("F1 Score:", f1_score(y_test, y_pred_pca_gb))
print("Precision Score:", precision_score(y_test, y_pred_pca_gb))
print("Recall Score:", recall_score(y_test, y_pred_pca_gb))

print("----------------------")

# Evaluate KPCA model
print("F1 Score:", f1_score(y_test, y_pred_kpca_gb))
print("Precision Score:", precision_score(y_test, y_pred_kpca_gb))
print("Recall Score:", recall_score(y_test, y_pred_kpca_gb))
```

```
F1 Score: 0.7200000000000001
Precision Score: 0.8181818181818182
Recall Score: 0.6428571428571429
----------------------
F1 Score: 0.896551724137931
Precision Score: 0.8666666666666667
Recall Score: 0.9285714285714286
```

2.5 Evaluation Metrics:

These are evaluation metrics for both the PCA and KPCA models:

2.5.1 For PCA Model:

- **F1 Score:** 0.72

  This is a weighted average of precision and recall. It indicates the balance between precision (the number of true positives divided by the sum of true positives and false positives) and recall (the number of true positives divided by the sum of true positives and false negatives). A higher F1 score indicates better performance in terms of both precision and recall.

- **Precision Score:** 0.818

  Precision is the proportion of true positive predictions among all positive predictions made by the model. A higher precision score indicates fewer false positives.

- **Recall Score:** 0.643

  Recall (also known as sensitivity) is the proportion of true positives that were correctly identified by the model among all actual positives in the data. A higher recall score indicates fewer false negatives.

2.5.2 For KPCA Model:

- **F1 Score:** 0.897
- **Precision Score:** 0.867
- **Recall Score:** 0.929

In summary:

The KPCA model outperforms the PCA model in all three metrics: F1 score, precision, and recall. This indicates that the KPCA model achieves a better balance between precision and recall, resulting in higher overall

performance. Specifically, the KPCA model has higher precision, recall, and F1 score, indicating better predictive capability and fewer misclassifications compared to the PCA model.

Chapter 3

Disease Prediction using Kernel PCA

3.1 Problem Statement

The objective of this project is to enhance prediction accuracy of machine learning model by employing Kernel Principal Component Analysis (KPCA). By applying KPCA on patient health data, we aim to uncover non-linear relationships and complex patterns that are not easily detectable through conventional methods. The refined dataset will be used to develop a robust predictive model capable of assessing heart attack risk with higher precision. This approach seeks to improve early diagnosis and prevention strategies, ultimately contributing to better patient care and outcomes.

3.2 Introduction

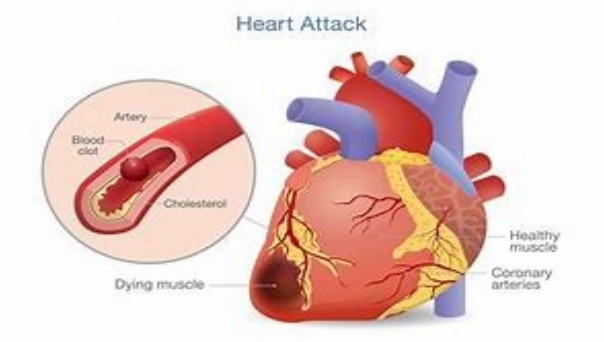
3.2.1 What is Heart Attack?

A heart attack happens when blood flow to the heart is blocked or cut off. If there's not sufficient oxygen-rich blood flowing to the heart, it can cause damage to the affected area. As a result, the heart muscle begins to die.

When your heart isn't getting the blood and oxygen it needs to function properly, it can put you at a higher risk of heart failure and other serious complications.A heart attack is a life threatening medical emergency. The sooner you can get medical treatment that restores normal blood flow to your heart, the better your chance of a successful

outcome.

3.2.2 How does it occur?



- Formation of plaque in coronary arteries.
- Plaque rupture leads to blood clot formation.
- Blood clot blocks blood flow in coronary artery.
- Reduced or blocked blood flow deprives heart muscle of oxygen.
- Resulting tissue damage or cell death manifests as a heart attack.

3.2.3 What are the symptoms of heart attack?

The general symptoms for a heart attack can include:

- Chest pain or discomfort
- Shortness of breath
- Pain in your arm, shoulder, or neck
- Nausea
- Sweating
- Lightheadedness or dizziness
- Fatigue
- Upper body pain
- Trouble breathing

3.3 Recognizing Variables In The Dataset

- **age** - Age of the person.
- **sex** - Gender of the person.
- **cp** - Chest pain type.
- **trtbps** - Resting blood pressure (in mm Hg).
- **chol** - Cholesterol in mg/dl fetched via BMI sensor.
- **fbs** - Fasting blood sugar > 120 mg/dl (1 = true; 0 = false).
- **restecg** - Resting electrocardiographic results.
- **thalach** - Maximum heart rate achieved.
- **exang** - Exercise induced angina (1 = yes; 0 = no).
- **oldpeak** - Previous peak.
- **slope** - Slope of the peak exercise ST segment.
- **ca** - Number of major vessels (0-3) colored by fluoroscopy.
- **thal** - Thalassemia (3 = normal; 6 = fixed defect; 7 = reversible defect).
- **target** - Target variable (1 = presence of heart disease; 0 = absence of heart disease).

3.4 Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is an essential step in the data analysis process, allowing us to understand the structure and characteristics of the dataset. In this section, we'll perform EDA on the "Heart attack analysis and prediction" dataset.

3.4.1 Required Python Libraries

```
import warnings

# Suppress all warnings
warnings.filterwarnings("ignore")
```

```
#Importing relevant libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA, KernelPCA
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix, f1_score, precision_score, recall_score
```

The code warnings.filterwarnings("ignore") is used to suppress all warnings in a Python program. This means that if there are any warning messages that Python would normally display (such as deprecation warnings or runtime warnings), they will be ignored and not shown to the user. This can help keep the output clean, but it also means you won't see potentially important warnings about issues in your code. We begin by importing essential libraries required for exploratory data analysis and subsequent analysis. pandas and numpy libraries are used for data manipulation and numerical operations. matplotlib.pyplot and seaborn are used for creating visualizations and plotting data. We import module from scikit-learn library including train-test-split for split the data into training and testing sets. StandardScaler for standardize the features. PCA and Kernel PCA for dimensionality reduction. GradientBoostingClassifier for a machine learning model for classification tasks. accuracy-score, classification-report, confusion-matrix, f1-score, precision-score, recall-score for evaluating the performance of the model using various metrics

3.4.2 Loading The Dataset:

```
# Load the dataset
data = pd.read_csv("heart.csv")
```

```
data.rename(columns={'output': 'heart_attack'}, inplace=True)
```

```
# The code below prints the first few rows of the dataframe 'data'
data.head()
```

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	heart_attack
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

The line data = pd.read-csv("heart.csv") is used to load a dataset named "heart.csv" into a pandas Data Frame. The function pd.read-csv("heart.csv") reads the data from the CSV file "heart.csv". The data is stored in a pandas Data Frame, which is a 2-dimensional, size-mutable, and potentially heterogeneous tabular data structure with labeled axes (rows and columns) .The code in the second cell used to rename the column of the DataFrame .The code data.head() is used to display the first few rows of the DataFrame data. This method is a built-in function of a pandas Data Frame .By default, data.head() returns the first 5 rows of the Data Frame. You can also specify a different number of rows to display by passing an integer as an argument (e.g., data.head(10) for the first 10 rows).

3.4.3 Shape and size of dataset:

```
# The code below retrieves the shape of the dataframe 'data'
data.shape
```

```
(303, 14)
```

```
# The code below retrieves the total number of elements in the dataframe 'data'
data.size
```

```
4242
```

The code data.shape is used to retrieve the dimensions of the DataFrame data. Specifically, this attribute is a built-in property of a pandas DataFrame .The shape attribute returns a tuple containing two values: the number of rows and the number of columns in the DataFrame. In this instance data.shape would return (303, 14) indicates that the DataFrame contains 303 rows and 14 columns. This is useful for understanding the size of the dataset. The code data.size is used to retrieve the total number of elements in the DataFrame data. Specifically ,this attribute is a built-in property of a pandas DataFrame.The size attribute returns the total count of all elements in the DataFrame, which is the product of the number of rows and the number of columns. In this instance the DataFrame data has 303 rows and 14 columns, data.size would return 303 * 14 = 4242. This provides a quick way to know the total number of data points in the DataFrame.

3.4.4 Information of the dataset:

```
# The code below provides concise summary information about the dataframe 'data'
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   age           303 non-null    int64  
 1   sex           303 non-null    int64  
 2   cp            303 non-null    int64  
 3   trtbps        303 non-null    int64  
 4   chol          303 non-null    int64  
 5   fbs           303 non-null    int64  
 6   restecg       303 non-null    int64  
 7   thalachh      303 non-null    int64  
 8   exng          303 non-null    int64  
 9   oldpeak       303 non-null    float64
 10  slp           303 non-null    int64  
 11  caa           303 non-null    int64  
 12  thall         303 non-null    int64  
 13  heart_attack  303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

The code data.info() provides a concise summary of the DataFrame data. Specifically, this method is a built-in function of a pandas DataFrame. The info() method outputs a summary that includes the total number of entries (non-null values) in each column ,the data type of each column ,memory usage information, additionally, it provides an overall count of non-null entries and the data types present in the DataFrame. The summary then proceeds to enumerate the 14 data columns present in the DataFrame, detailing various attributes for each. For instance, it reveals that all columns contain 303 non-null values, suggesting that there are no missing or empty entries within the dataset. Moreover, it specifies the data type for each column, with 13 columns represented as integers (int64) and 1 column represented as float (float64). The memory usage information indicates that the DataFrame consumes

approximately 33.3 KB of memory, which is valuable for optimizing memory allocation, especially when dealing with larger datasets. Overall, this summary provides essential insights into the dataset's structure, completeness, and memory utilization, laying a solid foundation for subsequent data analysis and manipulation tasks.

3.4.5 Descriptive statistics for the dataset:

```
# The code below generates descriptive statistics for the dataframe 'data'
data.describe()
```

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.0
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	0.729373	2.31
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	1.022606	0.61
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000	2.0
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000	2.0
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000	3.0
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	3.0

The code data.describe() generates descriptive statistics for the DataFrame data. Specifically, this method is a built-in function of a pandas DataFrame.The describe() method calculates various summary statistics for each numerical column in the DataFrame, including count (number of non-null value), mean (average value),std(standard deviation),min (minimum value),25th per, 50th per, 75th per (25th, 50th, and 75th percentiles),max(maximum value). For the 'age' column, the statistics are represented in scientific notation, indicating the count of entries (303.000000), mean value (54.366337), standard deviation (9.082101), minimum value (29.000000), 25th percentile (47.500000), median (55.000000), 75th percentile (61.000000), and maximum value (77.000000). Similarly, the 'sex', 'cp', 'trtbps' and remaining columns given in the DataFrame each have their respective statistics displayed. These include the count of non-null entries, mean value, standard deviation, minimum and maximum values, as well as the 25th, 50th (median), and 75th percentiles.

3.4.6 Demographic analysis:

```
# Filter the dataset to include only patients with heart disease (output == 1)
disease_patients = data[data['heart_attack'] == 1]

# Create age bins
age_bins = pd.cut(disease_patients['age'], bins=[0, 20, 40, 60, 80, 100])
disease_patients['age_bin'] = age_bins

# Plot
plt.figure(figsize=(10, 6))
sns.countplot(data=disease_patients, x='age_bin', palette='husl')
plt.xlabel('Age Range')
plt.ylabel('Count of Patients with Heart Disease')
plt.title('Distribution of Heart Disease Patients by Age Range')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

This code segment creates a visualization of the distribution of patients with heart disease across different age ranges.`disease_patients = data[data['heart_attack'] == 1]`: This filters the original dataset (`data`) to include only the rows where the value in the 'heart_attack' column is equal to 1, indicating patients with heart disease. The resulting subset is stored in a new DataFrame called `disease_patients`. `age_bins=pd.cut(disease_patients['age'], bins=[0, 20, 40, 60, 80, 100])`.

This creates age bins using the `pd.cut()` function. It bins the ages of the patients from the

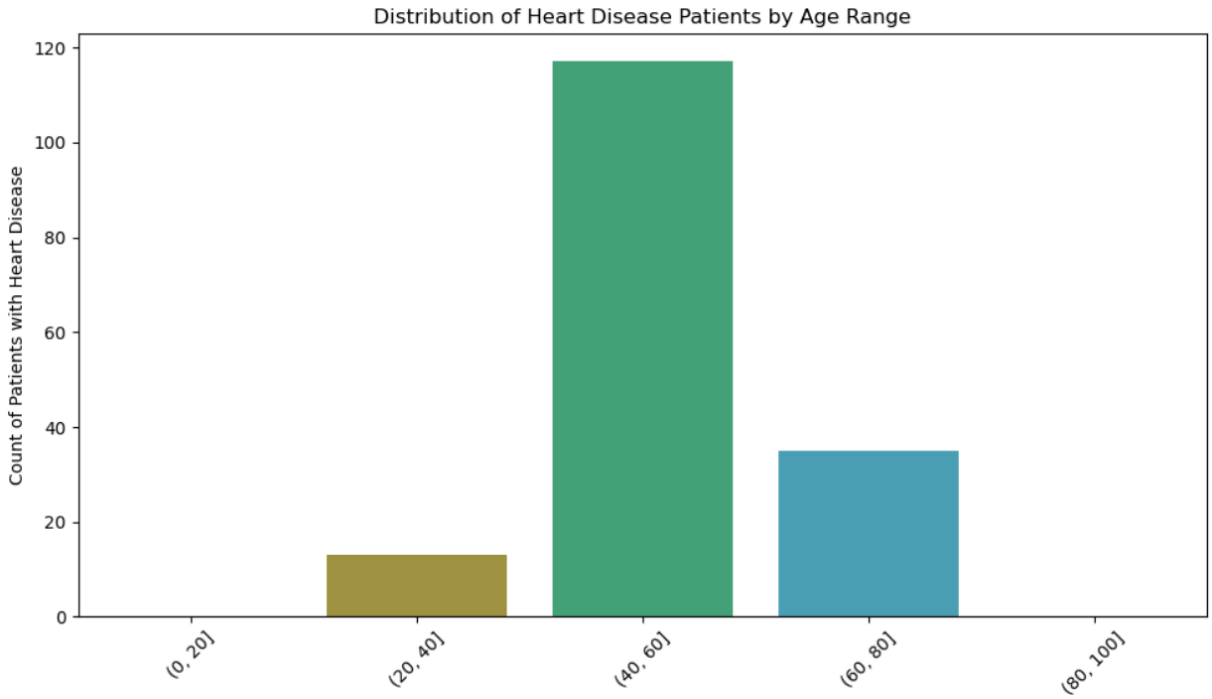
`disease_patients` DataFrame into intervals defined by the specified bins parameter. In this case, the bins are defined as [0, 20), [20, 40), [40, 60), [60, 80), and [80, 100). `disease_patients['age_bin'] = age_bins`.This adds a new column named 'age_bin' to the `disease_patients` DataFrame, containing the age bins computed in the previous step.`plt.figure(figsize=(10, 6))`.Sets the figure size for the plot.

`sns.countplot(data=disease_patients, x='age_bin', palette='husl')`:

Creates a count plot using seaborn (`sns`). It counts the number of occurrences of each age bin and plots them along the x-axis.`plt.xlabel('Age Range')`: Sets the label for the x-axis as 'Age Range'.

`plt.ylabel('Count of Patients with Heart Disease')`: Sets the label for the y-axis as 'Count of Patients with Heart Disease'.`plt.title('Distribution of Heart Disease Patients by Age Range')`: Sets the title of the plot. `plt.xticks(rotation=45)`: Rotates the x-axis labels by 45 degrees for better readability.`plt.tight_layout()`: Adjusts the layout to prevent overlapping of plot elements.`plt.show()`: Displays the plot.

Overall, this code segment provides a clear visualization of how the number of patients with heart disease is distributed across different age ranges, aiding in understanding the demographic distribution of heart disease patients. .

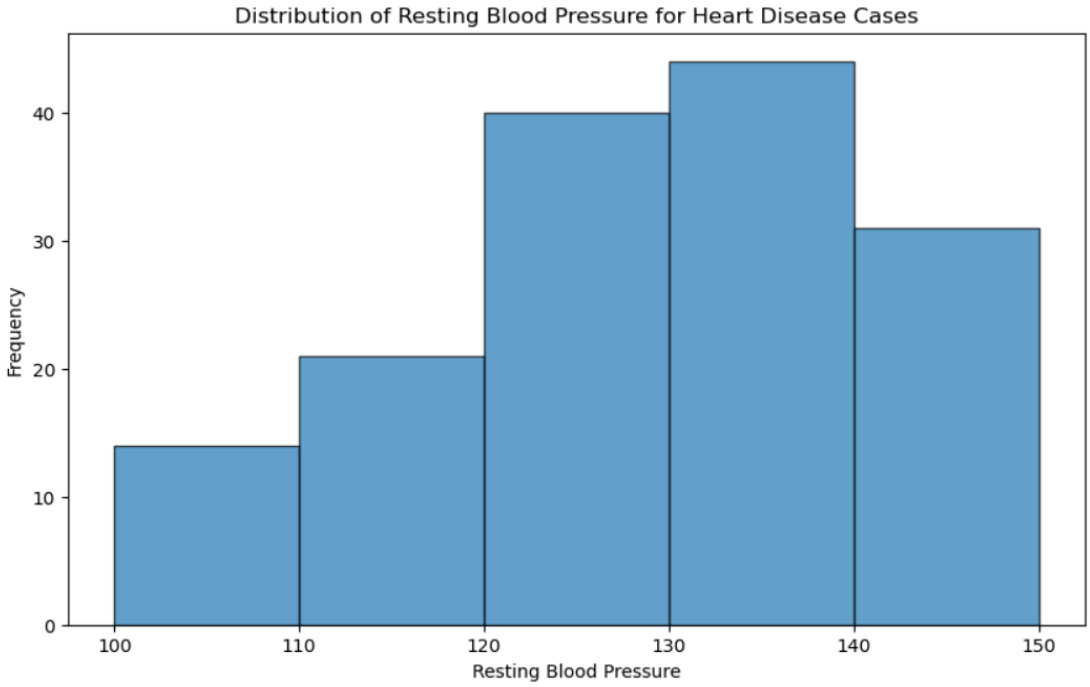


This bar chart provides a visual representation of the distribution of heart disease patients across different age ranges. The title of the chart is "Distribution of Heart Disease Patients by Age Range," indicating that the chart shows how heart disease cases are distributed among various age groups. The x-axis is labeled with age ranges in intervals of 20 years. The ranges are (0, 20],(20, 40],(40, 60],(60, 80] and (80, 100].These intervals represent the age groups of the patients. The y-axis shows the count of patients who have heart disease within each age range. The counts start from 0 and go up to 120. Each bar represents the number of heart disease patients in a specific age range. The first bar (0, 20] has a count close to 0, indicating no patients in this age range. The second bar (20, 40] shows a small count, indicating a relatively low number of patients. The third bar (40, 60] is the tallest, indicating that the majority of heart disease patients fall within this age range, with a count around 120.The fourth bar (60, 80] shows a moderate count, less than the (40, 60] range but higher than the first two ranges. The fifth bar (80, 100] has no count, indicating that there are no patients in this age range. The chart indicates that heart disease is most prevalent among patients aged 40-60, with significantly fewer cases in the other age ranges. This information could be useful for healthcare providers and policymakers to target prevention and treatment efforts more effectively.The visual representation helps quickly understand the age distribution of heart disease patients, highlighting the age group most affected by this condition.

3.4.7 Analysis of resting blood pressure with heart attack

```
# Assuming the columns for resting blood pressure and target variable are named 'trtbps' and 'output'
# Filter the data to include only rows where the target variable indicates heart disease (assuming 1 indicates heart disease)
heart_disease_data = data[data['heart_attack'] == 1]

# Define the bin range
bins = [100, 110, 120, 130, 140, 150]

# Plot a histogram
plt.figure(figsize=(10, 6))
plt.hist(heart_disease_data['trtbps'], bins=bins, edgecolor='k', alpha=0.7)
plt.title('Distribution of Resting Blood Pressure for Heart Disease Cases')
plt.xlabel('Resting Blood Pressure')
plt.ylabel('Frequency')
plt.xticks(bins)  # Set the x-ticks to match the bin edges
plt.show()
```

This code is about analyzing data related to heart disease. The code starts by filtering the dataset to include only rows where the target variable indicates heart disease. It assumes that a value of 1 in the 'heart -attack' column represents heart disease. It then defines bins for grouping the resting blood pressure values. These bins are defined by certain ranges of blood pressure levels. Next, it creates a histogram of the resting blood pressure values for cases where heart disease is present. The histogram visualizes the distribution of resting blood pressure within those cases. Various plot customizations are applied, such as setting the figure size, adding titles and labels to the axes, and setting the x-ticks to match the bin edges for clarity .Finally, it displays the histogram. Overall, this code helps visualize how resting blood pressure is distributed among individuals with heart disease, providing insights into potential relationships between blood pressure levels and heart disease.



This histogram visualizes the distribution of resting blood pressure levels among heart disease patients. The title of the graph is "Distribution of Resting Blood Pressure for Heart Disease Cases," indicating that the chart shows how resting blood pressure values are spread among individuals with heart disease. The x-axis represents the resting blood pressure levels, measured in mmHg (millimeters of mercury). The values range from 100 to 150, divided into bins of equal width (likely intervals of 10 mmHg).The y-axis shows the frequency, which is the number of heart disease patients within each resting blood pressure range. The frequency values start from 0 and go up to 50. Each bar represents the number of heart disease patients whose resting blood pressure falls within the corresponding range .The first bar (100-110) has a height around 15, indicating that approximately 15 patients have resting blood pressure in this range. The second bar (110-120) shows a height around 20, indicating that approximately 20 patients fall in this range. The third bar (120-130) rises significantly to about 40, suggesting that this is a common resting blood pressure range among the patients .The fourth bar (130-140) is the tallest, also around 40, showing that many patients have resting blood pressure in this range. The fifth bar (140-150) has a height around 35, indicating a slightly lower but still substantial number of patients in this range.The histogram shows that the majority of heart disease patients have resting blood pressure levels between 120 and 140 mmHg. This range represents the most common blood pressure levels among the patients in the dataset. Fewer patients have resting blood pressure below 120 mmHg or above 140 mmHg. This histogram provides a clear visual representation of the resting blood pressure distribution among heart disease patients, highlighting that most of them have blood pressure in the mid-ranges, specifically between 120 and 140 mmHg. This information can be useful for healthcare providers to understand typical blood pressure levels in heart disease patients and tailor treatment and monitoring strategies accordingly.

```
# Specify x and y
X = data[[ 'age', 'sex','cp','trtbps','chol','fbs','restecg','thalachh','exng','oldpeak','slp','caa','thall']]
y = data['heart_attack']
```

```
# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

The first cell snippet specifies the feature matrix X and the target variable y for a machine learning task using a dataset. X is defined as a subset of the original dataset, containing specific columns that are chosen as features for the model. The selected columns are 'age' (Age of the individual), 'sex' (Gender of the individual), 'cp' (Chest pain type), 'trtbps' (Resting blood pressure), 'chol' (Serum cholesterol), 'fbs' (Fasting blood sugar), 'restecg' (Resting electrocardiographic results), 'thalachh' (Maximum heart rate achieved), 'exng' (Exercise induced angina), 'oldpeak' (ST depression induced by exercise relative to rest), 'slp' (The slope of the peak exercise ST segment), 'caa' (Number of major vessels colored by fluoroscopy), 'thall' (Thalassemia). y is defined as the 'heart_attack' column from the dataset, which represents the target variable. This column typically indicates whether or not the individual has heart disease (for example, 1 might indicate the presence of heart disease and 0 might indicate absence). In summary, X contains the input features that will be used to predict y, which is the target variable representing the presence or absence of heart disease. This setup is commonly used in supervised machine learning tasks.The second cell splits the dataset into training and testing sets for a machine learning task.`train_test_split` from the `sklearn.model_selection` module is used to perform the split. X(features) and y(target variable) are provided as inputs to the function. `test_size=0.2` specifies that 20% of the data will be allocated to the test set, while the remaining 80% will be used for training the model. `random_state=42` is a seed for the random number generator. Setting this ensures that the split is reproducible, meaning you will get the same split each time you run the code. `X_train` features for the training set (80% of the data), `X_test` features for the test set (20% of the data), `y_train` target variable for the training set (80% of the data), and `X_test` target variable for the test set (20% of the data). In summary, this code prepares the data for training and evaluating a machine learning model by splitting it into training and testing sets, ensuring that the model can be trained on one portion of the data and evaluated on a separate, unseen portion.The third cell standardizes the features of the training and testing sets. The `StandardScaler` from `sklearn.preprocessing` is used to standardize the features. `scaler = StandardScaler()` creates an instance of the `StandardScaler`. `X_train_scaled = scaler.fit_transform(X_train)` computes the mean and standard deviation of the training set and scales the data accordingly. `fit_transform` both fits the scaler to the data and transforms the data in one step. `X_test_scaled = scaler.transform(X_test)` scales the test set using the mean and standard deviation calculated from the training set. `transform` is used to ensure the test data is scaled consistently with the training data. In summary, this code ensures that the features in both the training and testing sets have a mean of 0

and a standard deviation of 1, which can improve the performance of many machine learning algorithms.

3.4.8 Dimensionality reduction by PCA and KPCA:

```
# PCA
pca = PCA(n_components=2)  # You can adjust the number of components as needed
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)
```

This code applies Principal Component Analysis (PCA) to reduce the dimensionality of the dataset.

`pca = PCA(n_components=2)` creates a PCA object that will reduce the dataset to 2 principal components.`X_train_pca=pca.fit_transform(X_train_scaled)` fits the PCA model to the standardized training data and then transforms it, reducing its dimensionality to 2 components.

`X_test_pca = pca.transform(X_test_scaled)` transforms the standardized test data using the already fitted PCA model, ensuring consistency in dimensionality reduction.In summary, this code reduces the number of features in the dataset to 2 principal components, which can help in visualizing the data and may improve the performance of some machine learning algorithms by removing noise and redundancy.

```
# Kernel PCA
kpca = KernelPCA(n_components=2, kernel='rbf')  # You can experiment with different kernels
X_train_kpca = kpca.fit_transform(X_train_scaled)
X_test_kpca = kpca.transform(X_test_scaled)
```

This code applies Kernel Principal Component Analysis (Kernel PCA) to reduce the dimensionality of the dataset using a nonlinear kernel. `kpca = KernelPCA(n_components=2, kernel='rbf')` creates a Kernel PCA object to reduce the dataset to 2 components using the Radial Basis Function (RBF) kernel. Different kernels can be experimented with to see which works best for the data.

`X_train_kpca = kpca.fit_transform(X_train_scaled)` fits the Kernel PCA model to the standardized training data and then transforms it, reducing its dimensionality to 2 components. `X_test_kpca = kpca.transform(X_test_scaled)` transforms the standardized test data using the already fitted Kernel PCA model, ensuring consistency in dimensionality reduction.

In summary, this code reduces the number of features in the dataset to 2 components using Kernel PCA with an RBF kernel, which allows capturing nonlinear relationships in the data that standard PCA might miss.

3.4.9 Gradient Boosting model:

```
# Train Gradient Boosting model
gradboost_pca = GradientBoostingClassifier()
gradboost_pca.fit(X_train_pca, y_train)

gradboost_kpca = GradientBoostingClassifier()
gradboost_kpca.fit(X_train_kpca, y_train)

# Predictions
y_pred_pca_gb = gradboost_pca.predict(X_test_pca)
y_pred_kpca_gb = gradboost_kpca.predict(X_test_kpca)
```

This code trains two Gradient Boosting classifiers on datasets reduced by PCA and Kernel PCA, and then makes predictions.`gradboost_pca = GradientBoostingClassifier()` creates a Gradient Boosting classifier.`gradboost_pca.fit(X_train_pca, y_train)` trains the classifier using the training data reduced by PCA. `gradboost_kpca = GradientBoostingClassifier()` creates another Gradient Boosting classifier.`gradboost_kpca.fit(X_train_kpca, y_train)` trains this classifier using the training data reduced by Kernel PCA.`y_pred_pca_gb = gradboost_pca.predict(X_test_pca)` predicts the target values for the test set reduced by PCA.

`y_pred_kpca_gb = gradboost_kpca.predict(X_test_kpca)` predicts the target values for the test set reduced by Kernel PCA.This code trains and evaluates Gradient Boosting classifiers on both PCA and Kernel PCA reduced data.

3.4.10 Accuracy of PCA and KPCA trained on Gradiend Boosting model:

```
# Evaluate models
accuracy_pca_gb = accuracy_score(y_test, y_pred_pca_gb)
accuracy_kpca_gb = accuracy_score(y_test, y_pred_kpca_gb)
```

```
print("Accuracy with PCA:", accuracy_pca_gb)
print("Accuracy with kPCA:", accuracy_kpca_gb)
```

```
Accuracy with PCA: 0.819672131147541
Accuracy with kPCA: 0.8524590163934426
```

The code block evaluates the performance of the Gradient Boosting models trained on PCA and KPCA transformed data by computing their accuracy scores on the testing set.The accuracy of the Gradient Boosting model trained on PCA-transformed data is 82%. This means that the model correctly predicts the target variable (whether a customer purchased or not) for approximately 82% of the observations in the testing set.The accuracy of the Gradient Boosting model trained on KPCA transformed data is 85%. This indicates that the model trained on KPCA-transformed data performs better, achieving a higher accuracy of 85% on the testing set compared to the PCA-transformed model.In summary, the output suggests that the model trained on KPCA-transformed data outperforms the model trained on PCA-transformed data in terms of accuracy. This aligns with our earlier findings

where we observed that KPCA outperformed PCA, indicating that KPCA might be more effective in capturing the underlying patterns in the data.

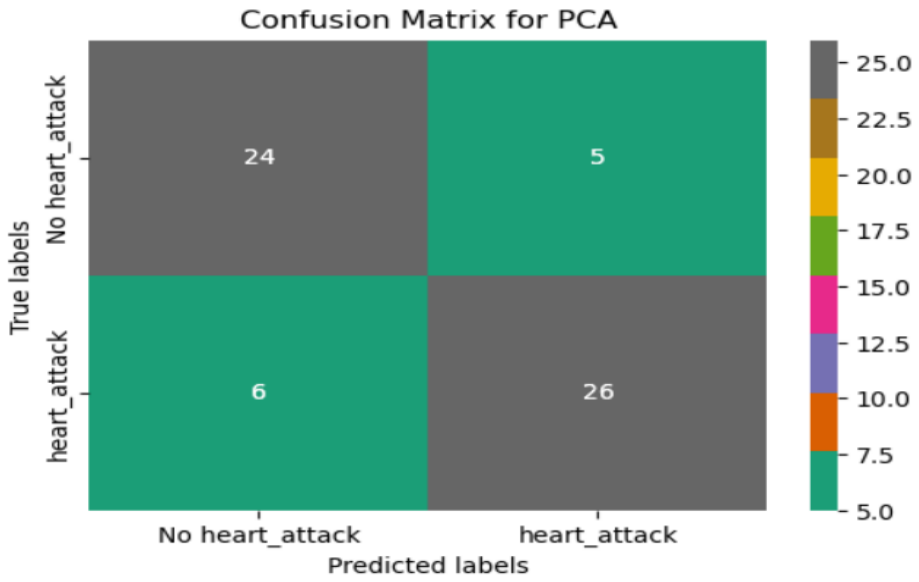
3.4.11 Evaluation metrics for PCA model:

```
# Define labels for clarity
labels = ['No heart_attack', 'heart_attack']

# Function to dynamically adjust font color based on background color
def get_text_color(value):
    return 'white' if value < 50 else 'black'  # Adjust the threshold as needed

# Evaluate PCA model
print("Evaluation metrics for PCA:")
print("Confusion Matrix:")
conf_matrix_pca = confusion_matrix(y_test, y_pred_pca_gb)
print(conf_matrix_pca)
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix_pca, annot=True, cmap='Dark2', fmt='g', cbar=True, xticklabels=labels, yticklabels=labels)
# Manually annotate the heatmap with values and adjust font color
for i in range(len(conf_matrix_pca)):
    for j in range(len(conf_matrix_pca[i])):
        plt.text(j+0.5, i+0.5, str(conf_matrix_pca[i][j]),\
        ha='center', va='center', fontsize=10, color=get_text_color(conf_matrix_pca[i][j]))
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix for PCA')
plt.show()
```

This code block visualizes the evaluation metrics for the Gradient Boosting model trained on PCA transformed data, specifically focusing on the confusion matrix.The line defines labels for the confusion matrix to clarify which class corresponds to “No heart_attack” and “heart_attack”. The `get_text_color` function dynamically adjusts the font color based on the background color of each cell in the heatmap. This function helps improve readability by ensuring that text remains visible against both light and dark backgrounds.

Evaluation Metrics for PCA Model (Purpose): The next code block evaluates the performance of the Gradient Boosting model trained on PCA-transformed data. By examining the confusion matrix, we can assess how well the model predicts each class (“No heart_attack” and “heart_attack”) and identify any potential misclassifications. The `sns.heatmap` function creates a heatmap visualization of the confusion matrix. Heatmaps provide a visually intuitive way to represent the confusion matrix, making it easier to interpret the distribution of true positive, false positive, true negative, and false negative predictions. The `plt.text` function manually annotates the heatmap with the values from the confusion matrix. Adding text labels to each cell of the heatmap enhances its interpretability by displaying the exact counts of true positive, false positive, true negative, and false negative predictions. Various plot customization options such as axis labels, title, and color map settings are applied to enhance the visual presentation of the confusion matrix. Customizing the plot ensures that it effectively communicates the evaluation metrics for the PCA model in a clear and informative manner. Overall, this code block provides a comprehensive visualization of the evaluation metrics for the PCA model, allowing for a detailed analysis of its performance in predicting the target variable classes.

```
Evaluation metrics for PCA:
Confusion Matrix:
[[24  5]
 [ 6 26]]
```



The output shows the evaluation metrics for the Gradient Boosting model trained on PCA-transformed data, specifically focusing on the confusion matrix.The value 24 in the top-left corner of the confusion matrix represents the number of observations where the model correctly predicted “No heart_attack” (the negative class) and the actual label was also “No heart_attack”.The value 5 in the top-right corner represents the number of observations where the model incorrectly predicted “heart_attack” (the positive class) but the actual label was “No heart_attack”. The value 6 in the bottom-left corner represents the number of observations where the model incorrectly predicted “No heart_attack” but the actual label was “heart_attack”.The value 26 in the bottom-right corner represents the number of observations where the model correctly predicted “heart_attack” and the actual label was also “heart_attack”.

In summary:

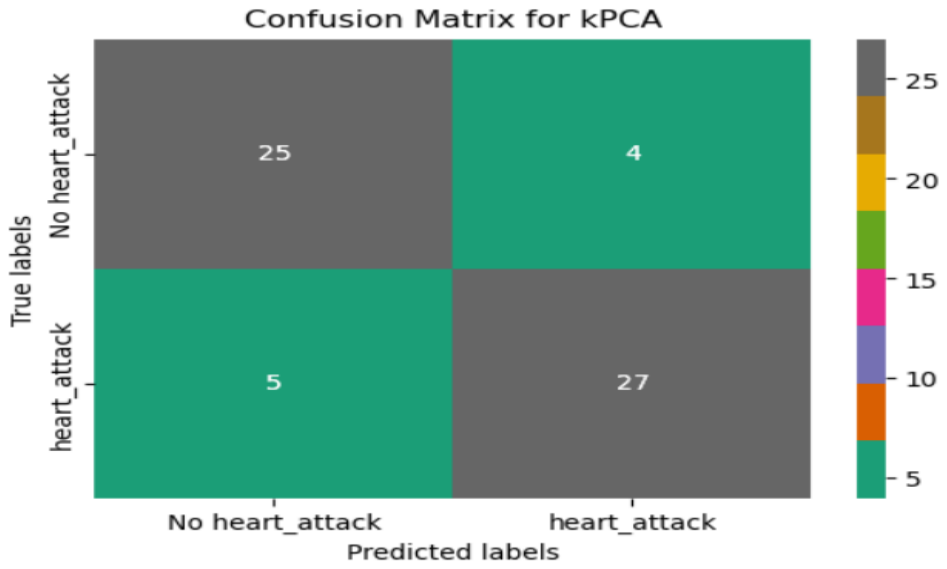
- The model correctly predicted “No heart_attack” for 24 observations.
- The model incorrectly predicted “heart_attack” for 5 observations that were actually “No heart_attack”.
- The model incorrectly predicted “No heart_attack” for 6 observations that were actually “heart_attack”.
- The model correctly predicted “heart_attack” for 26 observations.

3.4.12 Evaluation metrics for KPCA model:

```
# Evaluate KPCA model
print("\nEvaluation metrics for kPCA:")
print("Confusion Matrix:")
conf_matrix_kpca = confusion_matrix(y_test, y_pred_kpca_gb)
print(conf_matrix_kpca)
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix_kpca, annot=True, cmap='Dark2', fmt='g', cbar=True, xticklabels=labels, yticklabels=labels)
# Manually annotate the heatmap with values and adjust font color
for i in range(len(conf_matrix_kpca)):
    for j in range(len(conf_matrix_kpca[i])):
        plt.text(j+0.5, i+0.5, str(conf_matrix_kpca[i][j]),\
        ha='center', va='center', fontsize=10, color=get_text_color(conf_matrix_kpca[i][j]))
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix for kPCA')
plt.show()
```

This code block evaluates the performance of the Gradient Boosting model trained on KPCA-transformed data and visualizes the evaluation metrics using a confusion matrix. By examining the confusion matrix, we can assess how well the model predicts each class (“No heart_attack” and “heart_attack”) based on the KPCA-transformed features. The `sns.heatmap` function creates a heatmap visualization of the confusion matrix for the KPCA model. Heatmaps provide a visually intuitive way to represent the confusion matrix, making it easier to interpret the distribution of true positive, false positive, true negative, and false negative predictions. The `plt.text` function manually annotates the heatmap with the values from the confusion matrix for the KPCA model. Adding text labels to each cell of the heatmap enhances its interpretability by displaying the exact counts of true positive, false positive, true negative, and false negative predictions.Various plot customization options such as axis labels, title, and color map settings are applied to enhance the visual presentation of the confusion matrix for the KPCA model. Customizing the plot ensures that it effectively communicates the evaluation metrics for the KPCA model in a clear and informative manner.Overall, this code block provides a comprehensive visualization of the evaluation metrics for the KPCA model, allowing for a detailed analysis of its performance in predicting the target variable classes.

```
Evaluation metrics for kPCA:
Confusion Matrix:
[[25  4]
 [ 5 27]]
```



The output shows the evaluation metrics for the Gradient Boosting model trained on KPCA-transformed data, focusing on the confusion matrix.The value 25 in the top-left corner of the confusion matrix represents the number of observations where the model correctly predicted “No heart_attack” (the negative class) and the actual label was also “No heart_attack”.The value 4 in the top-right corner represents the number of observations where the model incorrectly predicted “heart_attack” (the positive class) but the actual label was “No heart_attack”.The value 5 in the bottom-left corner represents the number of observations where the model incorrectly predicted “No heart_attack” but the actual label was “heart_attack”.The value 27 in the bottom-right corner represents the number of observations where the model correctly predicted “heart_attack” and the actual label was also “heart_attack”. In summary:

- The model correctly predicted “No heart_attack” for 25 observations.
- The model incorrectly predicted “heart_attack” for 4 observations that were actually “No heart_attack”.
- The model incorrectly predicted “No heart_attack” for 5 observations that were actually “heart_attack”.
- The model correctly predicted “heart_attack” for 27 observations.

```python
#Evaluate PCA model
print("F1 Score:", f1_score(y_test, y_pred_pca_gb))
print("Precision Score:", precision_score(y_test, y_pred_pca_gb))
print("Recall Score:", recall_score(y_test, y_pred_pca_gb))

print("----------------------")

# Evaluate KPCA model
print("F1 Score:", f1_score(y_test, y_pred_kpca_gb))
print("Precision Score:", precision_score(y_test, y_pred_kpca_gb))
print("Recall Score:", recall_score(y_test, y_pred_kpca_gb))
```

```
F1 Score: 0.8253968253968254
Precision Score: 0.8387096774193549
Recall Score: 0.8125
----------------------
F1 Score: 0.8571428571428571
Precision Score: 0.8709677419354839
Recall Score: 0.84375
```

This code evaluates the performance of the Gradient Boosting models trained on PCA and Kernel PCA reduced data using various metrics. Here's a simple explanation:

1. **Evaluate PCA Model:**

   - 0.8253968253968254 is the F1 Score for the PCA model's predictions. The F1 Score is the harmonic mean of precision and recall.
   - 0.8387096774193549 is the Precision Score for the PCA model's predictions. Precision is the ratio of true positive predictions to the total predicted positives.
   - 0.8125 is the Recall Score for the PCA model's predictions. Recall is the ratio of true positive predictions to the total actual positives.

   Prints the F1 Score, Precision Score, and Recall Score for the PCA model. Prints a separator line for better readability.

2. **Evaluate KPCA Model:**

   - 0.8571428571428571 is the F1 Score for the KPCA model's predictions.
   - 0.8709677419354839 is the Precision Score for the KPCA model's predictions.
   - 0.84375 is the Recall Score for the KPCA model's predictions.

   Prints the F1 Score, Precision Score, and Recall Score for the KPCA model.

In summary, this code computes and prints the F1, Precision, and Recall scores to evaluate and compare the performance of the models trained using PCA and Kernel PCA.

References

1. https://microbiozindia.com/understanding-the-mathematics-behind-principal-component-a#:~:text=Principal%20Component%20Analysis%20(PCA)%20is%20a%20widely%20used%20dimensionality%20reduction,axes%2C%20referred%20as%20principal%20components
2. https://statisticsglobe.com/advantages-disadvantages-pca
3. https://www.baeldung.com/cs/kernel-principal-component-analysis#:~:text=Some%20of%20the%20advantages%20of,separation%20of%20classes%20or%20clusters
4. http://www.diva-portal.org/smash/get/diva2:402792/fulltext01.pdf
5. https://nirpyresearch.com/pca-kernel-pca-explained/
6. http://www.stats.org.uk/pca/
7. https://www.sciencedirect.com/science/article/pii/S0920410520302229
8. https://www.ibm.com/topics/principal-component-analysis
9. https://www.geeksforgeeks.org/ml-introduction-to-kernel-pca/
10. https://en.wikipedia.org/wiki/Principal_component_analysis#Properties_and_limitations_of_PCA
11. https://link.springer.com/chapter/10.1007/978-3-642-03664-4_94
12. https://www.sciencedirect.com/topics/agricultural-and-biological-sciences/principal-component-analysis#:~:text=PCA%20was%20invented%20in%201901,the%20modeling%20of%20response%20data
13. https://compphysics.github.io/MachineLearningMSU/doc/pub/svm/html/._svm-bs020.html#:~:text=An%20important%20theorem%20for%20us,T%CF%95(xj).
14. https://www.datacamp.com/tutorial/guide-to-the-gradient-boosting-algorithm
15. https://www.analytixlabs.co.in/blog/gradient-boosting-algorithm/