**PCA Stuff WorkFlow**

**1. Importing libraries:**

* **numpy** (as **np**): For numerical operations and arrays.
* **pandas** (as **pd**): For data manipulation and analysis.
* **seaborn** (as **sns**): For data visualization.
* **sklearn.metrics**: For calculating evaluation metrics.
* **plotly.express** (as **px**): For interactive visualizations.
* **matplotlib.pyplot** (as **plt**): A library for creating static, animated, and interactive visualizations in Python.
* Specific modules from scikit-learn (**sklearn**):
  + **PCA**: A class for performing Principal Component Analysis.
  + **StandardScaler**: A class for standardizing features by removing the mean and scaling to unit variance.
  + **KNeighborsClassifier**: A class implementing the k-nearest neighbors classifier.
  + **RandomForestClassifier**: A class implementing the random forest classifier.
  + **LogisticRegression**: A class implementing logistic regression.
  + **train\_test\_split**: A function for splitting data into training and testing sets.

1. **Importing data**:
   * Reads a CSV file into a pandas DataFrame (**df**).
2. **Data exploration**:

* **cols**: A list of column names representing different body measurements.
* **target**: A list of target labels representing different clothing sizes.
* **target\_num**: A list of target numbers assigned to each clothing size.
  + Prints the first few rows of the DataFrame using **df.head()**.
  + Computes descriptive statistics of the DataFrame using **df.describe()**.

1. **Data preprocessing**:
   * Scales the data using **StandardScaler()** from **sklearn.preprocessing**.
   * Fits the scaler on the DataFrame (**df**) using **scaler.fit(df)**.
   * Transforms the data to obtain scaled features using **scaler.transform(df)** and stores it in **scaled\_data**.
2. **Data visualization**:
   * Creates a scatter plot using the first two columns of **scaled\_data** (**scaled\_data[:, 0]** and **scaled\_data[:, 1]**) as the x and y coordinates, respectively.
   * Colors the points based on the corresponding target values (**target\_num**).
   * Displays the plot using **plt.show()**.
3. **Data splitting**:
   * Splits the scaled data (**scaled\_data**) and target values (**target\_num**) into training and testing sets using **train\_test\_split()** from **sklearn.model\_selection**.
   * The training set consists of 80% of the data, and the remaining 20% is used for testing.
   * The split is performed in a stratified manner, preserving the distribution of target values.
4. **Logistic Regression before PCA**:
   * Creates an instance of **LogisticRegression()** from **sklearn.linear\_model**.
   * Fits the logistic regression model on the training data using **reg\_model.fit(x\_train, y\_train)**.
   * Computes the accuracy score of the model on the test data using **reg\_model.score(x\_test, y\_test)** and stores it in **reg\_score**.
   * Makes predictions on the test data using **reg\_model.predict(x\_test)** and stores the predicted labels in **reg\_pred**.
   * Computes the confusion matrix using **metrics.confusion\_matrix(y\_test, reg\_pred)** and stores it in **reg\_cm**.
   * Computes various evaluation metrics (accuracy, precision, recall, F1 score) using functions from **metrics** module.
   * Prints the evaluation metrics.
5. **Random Forest before PCA**:
   * Similar to logistic regression, but uses **RandomForestClassifier()** from **sklearn.ensemble**.
   * Fits the random forest model, computes the evaluation metrics, and prints them.
6. **K-Nearest Neighbors (KNN) before PCA**:
   * Similar to logistic regression and random forest, but uses **KNeighborsClassifier()** from **sklearn.neighbors**.
   * Fits the KNN model, computes the evaluation metrics, and prints them.
7. **PCA (Principal Component Analysis)**:
   * The code initializes a PCA object with a variance threshold of **0.95**. This means that the resulting transformed data will retain 95% of the variance in the original data and set the test size to 20% and random state to **42** for reproducibility.
   * The **fit** method is called on the PCA object to perform the actual PCA on the **scaled\_data**, which is assumed to be the input dataset.
   * The **transform** method is used to transform the **scaled\_data** into the new feature space defined by the principal components. The transformed data is stored in **x\_pca**.
   * The shape of **x\_pca** is printed to see the number of samples and the reduced dimensionality.
   * Visualize the principal components (EigenVectors) as a heatmap using **sns.heatmap** from the seaborn library.
8. **Logistic Regression after PCA**:
   * The code splits the transformed data **x\_pca** and the target variable **target\_num** into training and testing sets using the **train\_test\_split** function from scikit-learn.
   * A logistic regression model (**LogisticRegression**) is created and trained on the training data (**pca\_x\_train** and **pca\_y\_train**) using the **fit** method.
   * The accuracy score of the model is computed on the test data (**pca\_x\_test** and **pca\_y\_test**) using the **score** method.
   * The predicted values for the test data are obtained using the **predict** method.
   * A confusion matrix (**pca\_reg\_cm**) is computed to evaluate the performance of the logistic regression model.
   * Various performance metrics such as accuracy, precision, recall, and F1 score are computed using functions from the **metrics** module of scikit-learn.
   * The confusion matrix is visualized using a heatmap from the **seaborn** library.
9. **Random Forest Classifier after PCA**:
   * Similar to the logistic regression section, a random forest classifier (**RandomForestClassifier**) is created and trained on the transformed data (**pca\_x\_train** and **pca\_y\_train**).
   * The accuracy score, predicted values, confusion matrix, and performance metrics are computed and displayed similarly to the logistic regression section.
10. **K-Nearest Neighbors Classifier after PCA**:
    * A k-nearest neighbors classifier (**KNeighborsClassifier**) is created with **n\_neighbors=5**.
    * The classifier is trained on the transformed data (**pca\_x\_train** and **pca\_y\_train**).
    * The accuracy score, predicted values, confusion matrix, and performance metrics are computed and displayed similarly to the previous sections.
11. **Visualization**:
    * The scatter plot of the transformed data (**x\_pca**) is displayed using **plt.scatter** from Matplotlib. The colors represent different classes in the **target\_num** variable.
    * A 3D scatter plot of the transformed data is created using the **scatter\_3d** function from Plotly Express. The colors represent different classes.
12. **PCA Components and Explained Variance:**
    * Print the principal components **(pca.components\_**) obtained from PCA.
    * Print the explained variance ratio for each principal component (**pca.explained\_variance\_ratio\_**).
    * Create a DataFrame (**df\_**) to display the principal components with their corresponding features.
    * Plot a heatmap of the principal components using **sns.heatmap** from the seaborn library.
13. **Explained Variance and Cumulative Variance:**
    * Calculate the explained variance ratios for each principal component.
    * Calculate the cumulative explained variance by taking the cumulative sum of the explained variance ratios.
    * Create DataFrames (**explained\_variance\_df** and **cumulative\_variance\_df**) to display the explained variance and cumulative variance.
    * Visualize the explained variance ratios and cumulative variance using bar plots.
14. **Scatter Plot of Loadings:**
    * Create a 3D scatter plot of the loadings of each feature on the principal components.
    * The loadings represent the correlation between each feature and the principal components.
15. **Scatter Plot of Scores:**
    * Create a 3D scatter plot of the principal components (**PC1**, **PC2**, and **PC3**) colored by the class labels (Class).
    * Visualize the distribution of data points in the reduced-dimensional space.