# **Assignment -3\_ DASC5301**

# **FIRST QUESTION:-Data from our lives**

### Describe a situation or problem from your job, everyday life, current events, etc., for which a variable selection/feature reduction would be appropriate.

**My Answer** :-

1. Situation: Medical imaging, such as MRI scans, generate a vast number of data points for each patient, resulting in high-dimensional data sets.These data typically include numerous features (pixels) representing different aspects of tissues, structures, or organs.
2. Challenge: Handling high-dimensional data poses a challenge as not all features contribute equally to diagnoses. This complexity hinders efficient analysis and may lead to computational inefficiencies.
3. Feature Reduction: Techniques like PCA or autoencoders streamline this data by extracting essential patterns while retaining critical information. This reduction aids in more efficient analysis and clearer visualization of anomalies.
4. Diagnostic Precision: By focusing on key features, reduced data sets enable clearer identification of abnormalities, improving diagnostic accuracy.
5. Efficiency and Generalization: Feature reduction not only optimizes computational resources but also helps models generalize better across different patient groups.
6. Finding: In medical imaging analysis, feature reduction techniques enhance efficiency, aid in precise diagnostics, and support more effective decision-making by healthcare professionals.

A screenshot of a computer program

Description automatically generated

1. scipy: A library for scientific computing that includes statistical functions.
2. sklearn.linear\_model: Part of scikit-learn, this module provides tools for linear regression.
3. statsmodels.compat: Compatibility module for statsmodels, which is a library for estimating and testing statistical models.
4. statsmodels.formula.api: A module in statsmodels that provides a formula-based interface to specify linear models.
5. statsmodels.stats.anova: A module for performing analysis of variance (ANOVA).
6. statsmodels.stats.outliers\_influence: A module for detecting influential data points and computing variance inflation factors.
7. matplotlib: A popular plotting library for creating static, animated, and interactive visualizations.
8. numpy: A library for numerical operations in Python.
9. pandas: A powerful data manipulation library.
10. seaborn: A statistical data visualization library based on matplotlib.
11. statsmodels.api: The main API for statsmodels, which includes tools for estimating and testing models.

A computer code with text

Description automatically generated

1. import pandas as pd: This line imports the pandas library and assigns it the alias 'pd'. This is a common convention to make the code more concise.
2. pd.read\_csv('auto\_imports1.csv'): The read\_csv() function is a pandas function used to read data from a CSV file. It takes the file path as an argument and returns a DataFrame containing the data from the CSV file.
3. df = ...: The result of pd.read\_csv() is assigned to the variable 'df'. 'df' is a common naming convention for a DataFrame (short for Data Frame), which is a two-dimensional, size-mutable, and potentially heterogeneous tabular data structure with labeled axes (rows and columns).
4. df.head(): This method is used to display the first 5 rows of the DataFrame. It's a quick way to inspect the data and get a sense of its structure.
5. this code snippet reads the contents of the 'auto\_imports1.csv' file into a Pandas DataFrame named 'df' and then displays the first 5 rows of the DataFrame to provide a glimpse of the data.

A screenshot of a computer

Description automatically generated

A screenshot of a computer code

Description automatically generated

1. Extract the data types of each column in the DataFrame and assign it to the variable 'Auto\_data\_types'
2. df.dtypes: This is an attribute of a Pandas DataFrame that returns a Series with the data type of each column. The resulting Series has the column names as the index and the corresponding data types as values.
3. Auto\_data\_types = ...: The extracted Series of data types is assigned to the variable 'Auto\_data\_types'. This variable will contain information about the data types of each column in the DataFrame.
4. print(Auto\_data\_types): This line prints the contents of the 'Auto\_data\_types' Series, displaying the data types of each column in the DataFrame.
5. In the output, 'column1', 'column2', and 'column3' are assumed to be placeholder names for the actual column names in your DataFrame. The data types (int64, float64, object, etc.) provide information about the types of values stored in each column (integer, float, string, etc.). This information is useful for understanding the nature of the data and for preprocessing or analysis tasks that may be sensitive to data types.

Output:-

fuel\_type object

body object

wheel\_base float64

length float64

width float64

heights float64

curb\_weight int64

engine\_type object

cylinders object

engine\_size int64

bore object

stroke object

comprassion float64

horse\_power object

peak\_rpm object

city\_mpg int64

highway\_mpg int64

price int64

dtype: object

df = df.replace('?', None)

Replacing ? with none throught the dataset

This type of operation is often used when dealing with missing or placeholder values in a dataset. The choice of replacing '?' with None suggests that the original data might have used '?' as a placeholder for missing values, and the code is replacing them with Python's None to represent missing or undefined data.

A screen shot of a computer code

Description automatically generated

The astype() method is used to change the data type of a Pandas Series (or DataFrame column) to the specified type. In this case, it is converting these specific columns to the float data type, assuming that the original values in these columns are numeric and can be represented as floating-point numbers.

1. df['bore'] = df['bore'].astype(float): This line is converting the 'bore' column in the DataFrame 'df' to the float data type using the astype() method. The column 'bore' is assumed to contain values that can be represented as floating-point numbers.
2. df['stroke'] = df['stroke'].astype(float): Similarly, this line is converting the 'stroke' column to the float data type.
3. df['horse\_power'] = df['horse\_power'].astype(float): This line is converting the 'horse\_power' column to the float data type.
4. df['peak\_rpm'] = df['peak\_rpm'].astype(float): Finally, this line is converting the 'peak\_rpm' column to the float data type.
5. The astype() method is used to change the data type of a Pandas Series (or DataFrame column) to the specified type. In this case, it is converting these specific columns to the float data type, assuming that the original values in these columns are numeric and can be represented as floating-point numbers.

A computer code with text

Description automatically generated with medium confidence

The astype() method is used to change the data type of a Pandas Series (or DataFrame column) to the specified type. In this case, it is converting these specific columns to the float data type, assuming that the original values in these columns are numeric and can be represented as floating-point numbers.

no remaining '?' values in the dataset.

1. (df == '?'): This creates a boolean DataFrame of the same shape as df, where each element is True if the corresponding element in df is equal to the string '?', and False otherwise.
2. .sum() (first instance): This sums the boolean values along each column, resulting in a Series where each element represents the count of True values in the corresponding column.
3. .sum() (second instance): This sums the counts obtained in step 2 across all columns, providing the total count of '?' values in the entire DataFrame.
4. If there are no remaining '?' values, it prints "No remaining '?' values in the dataset."
5. If there are remaining '?' values, it prints the count of remaining '?' values using an f-string: f"{question\_marks\_remaining} remaining '?' values in the dataset."

df.info()

The astype() method is used to change the data type of a Pandas Series (or DataFrame column) to the specified type. In this case, it is converting these specific columns to the float data type, assuming that the original values in these columns are numeric and can be represented as floating-point numbers.

It displays data in the dataframe

1. RangeIndex: 1000 entries, 0 to 999: This line provides information about the index of the DataFrame. In this example, it's a RangeIndex starting from 0 to 999 with a total of 1000 entries.
2. Data columns (total 5 columns):: This line specifies that there are a total of 5 columns in the DataFrame.
3. column1 1000 non-null int64: This line provides information about the first column ('column1'). It indicates that there are 1000 non-null (non-missing) values in this column, and the data type is int64 (64-bit integer).

Output:- A screenshot of a computer

Description automatically generated

A close-up of a text

Description automatically generated

The astype() method is used to change the data type of a Pandas Series (or DataFrame column) to the specified type. In this case, it is converting these specific columns to the float data type, assuming that the original values in these columns are numeric and can be represented as floating-point numbers.

Droping the specified colunms and creating a DF2

1. df.drop(columns=["body", "engine\_type", "cylinders"]): This is a Pandas DataFrame method (drop) used to remove specified columns from a DataFrame. The columns parameter takes a list of column names to be dropped.
2. df2 = ...: The result of the drop operation is assigned to the variable 'df2', creating a new DataFrame that does not include the specified columns.
3. After running this code, the DataFrame df2 will be a modified version of the original DataFrame df with the columns "body," "engine\_type," and "cylinders" removed.

Output:- A screenshot of a computer

Description automatically generated

df2.dropna(inplace=True)

The astype() method is used to change the data type of a Pandas Series (or DataFrame column) to the specified type. In this case, it is converting these specific columns to the float data type, assuming that the original values in these columns are numeric and can be represented as floating-point numbers.

**Output**:- dropping rows with null values in the dataframe.

A black and white text

Description automatically generated

A computer code with text

Description automatically generated

The astype() method is used to change the data type of a Pandas Series (or DataFrame column) to the specified type. In this case, it is converting these specific columns to the float data type, assuming that the original values in these columns are numeric and can be represented as floating-point numbers.

1. pd.get\_dummies(df2, columns=['fuel\_type'], drop\_first=True): This function is used to convert categorical variable(s) into dummy/indicator variables. In this case, it is creating dummy variables for the 'fuel\_type' column in the DataFrame df2.

1. drop\_first=True: It drops the first dummy variable to avoid multicollinearity, which is the situation where predictor variables in a regression model are highly correlated.
2. The resulting DataFrame df2 will have new columns representing the different categories in the 'fuel\_type' column with binary values (0 or 1) indicating the presence of each category.

**Output:-** A screenshot of a computer

Description automatically generated

A screenshot of a white background

Description automatically generated

# 

# **2. Variable selection**

They are the 3 methods :-

1. Filter methods
2. Wrapper methods
3. Embedded methods

2.1 Filter Methods

Choosing one (you may do more, one is required) of the filtered methods to conduct variable selection. Report we findigs.

The astype() method is used to change the data type of a Pandas Series (or DataFrame column) to the specified type. In this case, it is converting these specific columns to the float data type, assuming that the original values in these columns are numeric and can be represented as floating-point numbers.

A computer screen shot of a computer code

Description automatically generated

1. df2.corr(): This computes the pairwise correlation coefficients between columns in the DataFrame df2. The result is a correlation matrix where each entry (i, j) represents the correlation between column i and column j. The values range from -1 to 1, where 1 indicates a perfect positive correlation, -1 indicates a perfect negative correlation, and 0 indicates no correlation.
2. plt.figure(figsize=(10, 8)): This line sets the size of the figure (heatmap) to be created. The figsize parameter specifies the width and height of the figure in inches.
3. sns.heatmap(df2\_matrix, annot=True, cmap='coolwarm', linewidths=0.7):
4. sns.heatmap(): This function from the Seaborn library is used to create a heatmap.
5. df2\_matrix: The correlation matrix to be visualized.
6. annot=True: This parameter adds the numerical values in each cell of the heatmap.
7. cmap='coolwarm': The color map used for the heatmap, where 'coolwarm' represents a range of cool to warm colors.
8. inewidths=0.7: Specifies the width of the lines that will divide each cell in the heatmap.
9. plt.title('Heatmap'): This sets the title of the heatmap to 'Heatmap'.
10. plt.show(): This displays the created heatmap.

The heatmap provides a visual representation of the correlations between different columns in the DataFrame. It helps in identifying patterns and relationships in the data. The color intensity and the numerical values in each cell convey information about the strength and direction of the correlation.

Output:-

# A chart with numbers and symbols Description automatically generated with medium confidence

The astype() method is used to change the data type of a Pandas Series (or DataFrame column) to the specified type. In this case, it is converting these specific columns to the float data type, assuming that the original values in these columns are numeric and can be represented as floating-point numbers.

A computer screen shot of a computer code

Description automatically generated

1. df2\_matrix['price']: This selects the 'price' column from the correlation matrix df2\_matrix. The result is a Pandas Series containing correlation coefficients between the 'price' column and all other columns.

1. .sort\_values(ascending=False): This method sorts the values in the Pandas Series in descending order. The resulting Series, corr\_betwn\_price, will have the columns with the highest positive correlations with 'price' at the top.
2. print(corr\_betwn\_price): This prints the sorted correlation values.
3. The output will be a Series where each entry represents the correlation coefficient between the 'price' column and a specific column in df2, sorted in descending order. Positive values indicate a positive correlation (as one variable increases, the other tends to increase), and negative values indicate a negative correlation (as one variable increases, the other tends to decrease).

A screenshot of a computer code

Description automatically generated

After conducting correlation analysis between various features and the 'price' of automobiles, we identified specific features that demonstrate notable relationships with the price:

1. Strong Positive Correlation Features:

These variables have correlation coefficients ranging from approximately 0.59 to 0.89, 'engine\_size', 'curb\_weight', 'horse\_power', 'width', 'length', and 'wheel\_base' exhibited strong positive correlations with the price. This indicates that as these features increase (e.g., larger engine size, heavier curb weight, more horsepower, wider or longer cars), the price tends to rise accordingly. These attributes seem to significantly impact the pricing of automobiles, suggesting their importance in influencing price variations.

2. Moderate Negative Correlation Features:

'city\_mpg' and 'highway\_mpg' displayed moderate negative correlations with the price. Higher miles per gallon (mpg), both in city and highway driving conditions, were associated with lower prices. This suggests that cars with better fuel efficiency are often priced lower, reflecting a potential market tendency or consumer preference.

These findings propose that certain characteristics strongly align with automobile pricing trends. Considering these influential features for model construction or in-depth analysis might enhance our understanding of how they individually and collectively contribute to determining the price of vehicles within the dataset.

3. Weak Correlation with 'Price': Other variables such as 'bore', 'heights', 'stroke', 'comprassion', 'peak\_rpm', and 'fuel\_type\_gas' show weak correlations with 'price'. Their correlation coefficients are closer to zero, suggesting a weaker linear relationship with 'price'.

Utilizing these identified features could be instrumental in predictive modeling to estimate automobile prices more accurately. Furthermore, exploring these influential factors may offer valuable insights into market dynamics, consumer behavior, and the impact of specific vehicle attributes on pricing strategies within the automotive industry.

# **2.2. Wrapper methods**

The astype() method is used to change the data type of a Pandas Series (or DataFrame column) to the specified type. In this case, it is converting these specific columns to the float data type, assuming that the original values in these columns are numeric and can be represented as floating-point numbers.

A screenshot of a computer program

Description automatically generated

1. Separating predictors and target variable: X = df2.drop(columns=['price']): Creates a DataFrame X by excluding the 'price' column from the original DataFrame df2. This DataFrame contains the predictor variables or features.

y = df2['price']: Creates a Series y that includes only the 'price' column from the original DataFrame df2. This Series represents the target variable.

2. Creating a linear regression model:

lin\_model = LinearRegression(): Initializes a linear regression model using scikit-learn's LinearRegression class.

3.Initializing RFE (Recursive Feature Elimination):

r\_f\_e = RFE(lin\_model, n\_features\_to\_select=5): Initializes the RFE object. It takes the linear regression model (lin\_model) and specifies that it should select 5 features during the elimination process.

4. Fitting RFE to the data:

fit = r\_f\_e.fit(X, y): Fits the RFE model to the data. The RFE algorithm recursively removes the least important features until the desired number of features is reached.

1. Getting the selected features:

features\_sel = X.columns[fit.support\_]: Retrieves the selected features based on the RFE support. These are the features that the RFE algorithm considers important for predicting the target variable.

1. Printing the selected features: print("RFE Selected features:"): Prints a message indicating that the following lines will display the selected features.

7. print(features\_sel): Prints the names of the selected features.

In summary, this code uses RFE to identify and print the top 5 features considered most important for predicting the target variable 'price' using a linear regression model.

Output:- RFE Selected features:

Index(['width', 'bore', 'comprassion', 'city\_mpg', 'fuel\_type\_gas'], dtype='object')

The Recursive Feature Elimination (RFE) method, using a Linear Regression model, identified key features strongly influencing automobile prices:

1.'Width': Car width emerges as a significant factor affecting prices, suggesting that wider cars might command higher prices in the market.

2.'Bore': The engine's cylinder bore diameter holds importance, hinting that engine specifications play a role in pricing vehicles.

3.'Compression':it signifies engine efficiency, indicating its impact on pricing.

4.'City MPG': Better fuel efficiency in city driving conditions appears to associate with lower prices, indicating the influence of fuel economy on pricing strategies.

5.'Fuel Type (Gas)': The indication of a gas-powered engine suggests that fuel type significantly affects pricing.

These findings shed light on various aspects driving car prices, such as car dimensions, engine specifications, fuel efficiency, and fuel type. Understanding these influential features can aid market analysis, consumer behavior studies, and strategic pricing decisions within the automotive industry. Further exploration and modeling using these features could offer deeper insights into their direct impact on automobile pricing dynamics and consumer preferences.

### **2.3. Embedded methods**

The astype() method is used to change the data type of a Pandas Series (or DataFrame column) to the specified type. In this case, it is converting these specific columns to the float data type, assuming that the original values in these columns are numeric and can be represented as floating-point numbers.

A screenshot of a computer program

Description automatically generated

Output:-

Lasso Selected features:

Index(['wheel\_base', 'length', 'width', 'heights', 'curb\_weight',

'engine\_size', 'bore', 'stroke', 'comprassion', 'horse\_power',

'peak\_rpm', 'city\_mpg', 'highway\_mpg', 'fuel\_type\_gas'],

dtype='object')

1.Separating predictors and target variable:

X = df2.drop(columns=['price']): Creates a DataFrame X by excluding the 'price' column from the original DataFrame df2. This DataFrame contains the predictor variables or features.

y = df2['price']: Creates a Series y that includes only the 'price' column from the original DataFrame df2. This Series represents the target variable.

2.Creating Lasso regression model:

lasso = Lasso(alpha=0.1): Initializes a Lasso regression model using scikit-learn's Lasso class. The alpha parameter controls the strength of the regularization. A higher alpha leads to more coefficients being shrunk to zero.

3.Fitting the Lasso model to the data:

lasso.fit(X, y): Fits the Lasso regression model to the data. During this process, Lasso applies regularization, and some coefficients are shrunk to zero.

4.Getting coefficients and corresponding feature names:

lasso\_coeff = lasso.coef\_: Retrieves the coefficients assigned by the Lasso model to each feature.

featured\_names = X.columns: Retrieves the names of the features from the DataFrame.

5.Selecting non-zero coefficient features:

feature\_sele = featured\_names[lasso\_coeff != 0]: Selects the features that have non-zero coefficients, indicating that these features are considered important by the Lasso regression model.

6.Printing the selected features:

print("Lasso Selected features:"): Prints a message indicating that the following lines will display the selected features.

print(feature\_sele): Prints the names of the selected features.

In summary, the code uses Lasso regression to identify and print the features that have non-zero coefficients, effectively performing feature selection by shrinking some coefficients to zero.

A screenshot of a computer code

Description automatically generated

The astype() method is used to change the data type of a Pandas Series (or DataFrame column) to the specified type. In this case, it is converting these specific columns to the float data type, assuming that the original values in these columns are numeric and can be represented as floating-point numbers.

Output:-

A close-up of a white page

Description automatically generated

1.List of different alpha values:

diff\_alphas = [0.01, 0.1, 1, 10]: Defines a list of different alpha values to be used in Lasso regression.

2.List to store selected features for each alpha:

feature\_sele = []: Initializes an empty list to store the selected features for each alpha.

3.Iterate over different alpha values:

The for loop iterates over each alpha value in diff\_alphas.

4.Inside the loop:

lasso = Lasso(alpha=alpha): Creates a Lasso regression model with the current alpha.

lasso.fit(X, y): Fits the Lasso model to the data.

coeff = lasso.coef\_: Retrieves the coefficients assigned by the Lasso model to each feature.

feature\_names = X.columns: Retrieves the names of the features from the DataFrame.

selected\_features = feature\_names[coeff != 0]: Selects the features that have non-zero coefficients and appends them to the list.

5.Print the selected features for each alpha:

The second for loop iterates over the alpha values and their corresponding selected features.

print(f"Alpha={alpha}: Features selected - {features}"): Prints the alpha value along with the features selected for that alpha.

This code provides insights into how the selected features change with different values of alpha in Lasso regression, helping to understand the impact of regularization on feature selection.

N It appears that Lasso regression has deemed all the available features crucial for predicting automobile prices in our dataset. Through its regularization technique, Lasso retained all features without zero coefficients, signaling their importance in predicting the 'price' variable.

This result implies that, according to Lasso regression, every feature holds significance in forecasting automobile prices, even under its penalty for excessive complexity. However, it's important to note that altering the regularization strength (alpha value) could yield a different set of selected features. Adjusting this parameter allows for fine-tuning the feature selection process to better suit our predictive needs.

### **2.4. Compare your results**

Comparing results from the three methods and also comparing the coefficients to the full linear regression model (model1) from the previous homework.

### **Filter Method (Correlation Analysis):**

* **Trail of Correlations**: This path highlighted features with strong ties to the price, focusing on linear relationships.
* **Brightest Beacons**: Features shining with high positive or negative correlations led the way.
* **Watchful Eye Needed**: However, this path might have missed the intricacies of complex relationships among variables.

### **Wrapper Method (RFE - Recursive Feature Elimination):**

* **Model-Guided Journey**: This approach navigated through predictors, seeking those contributing most to model performance.
* **Selected Enclave**: It spotlighted a smaller group of features ('width', 'bore', 'comprassion', 'city\_mpg', 'fuel\_type\_gas').
* **Potential Limitation**: Yet, the journey might have bypassed valuable insights outside this selected group.

### **Embedded Methods (Lasso:**

* **Regularization Route**: These methods used the power of regularization, consistently pointing toward the same set of features ('wheel\_base', 'length', 'width', 'heights', 'curb\_weight', 'engine\_size', 'bore', 'stroke', 'comprassion', 'horse\_power', 'peak\_rpm', 'city\_mpg', 'highway\_mpg', 'fuel\_type\_gas') across different strengths.
* **Stable Path**: However, while stable, this path might have overlooked nuances in feature importance.

### **Full Linear Regression Model (Model1):**

* **Coefficients as Clues**: This path revealed insights into individual variable impacts with coefficients, but warnings about multicollinearity and non-numeric values in 'fuel\_type\_gas' cast shadows on reliability.
* **Insights with Caveats**: It provided clarity but also raised concerns about variable interactions and model stability.

### **Comparing Paths:**

* **Varied Approaches**: Each path brought unique strengths but also limitations in selecting impactful features.
* **Precision vs. Scope**: Some paths highlighted specific variables, while others aimed for a broader view.
* **Insights vs. Blind Spots**: Coefficients offer detailed insights, yet concerns about reliability remain.

### **Conclusion:**

* **Diverse Terrain**: Each method offers a different landscape in selecting features.
* **Strengths and Caveats**: While some paths focus on specific attributes, others offer a broader perspective, each with its own limitations.
* **Harmonizing Insights**: Combining findings from these diverse paths may offer a more comprehensive understanding, crucial for robust feature selection in predicting automobile prices.

In this grand exploration, combining the strengths of different paths might unveil the most influential predictors, guiding us toward a better understanding of automobile price prediction in our dataset.

### **2.5 Bonus question (extra 5 points)**

The astype() method is used to change the data type of a Pandas Series (or DataFrame column) to the specified type. In this case, it is converting these specific columns to the float data type, assuming that the original values in these columns are numeric and can be represented as floating-point numbers.

A screenshot of a computer code

Description automatically generated

1.Standardizing the data:

scaler = StandardScaler(): Initializes a StandardScaler object, which is used for standardizing the data.

scaled\_df2 = scaler.fit\_transform(df2): Applies standardization to the original DataFrame df2, resulting in scaled\_df2. Standardization involves scaling features to have a mean of 0 and a standard deviation of 1.

2.Applying PCA (Principal Component Analysis):

pca = PCA(): Initializes a PCA object.

df2\_pca = pca.fit\_transform(scaled\_df2): Applies PCA to the standardized data, resulting in a new DataFrame df2\_pca where each column represents a principal component.

3.Explained variance ratio and cumulative variance ratio:

ex\_var\_ratio = pca.explained\_variance\_ratio\_: Retrieves the explained variance ratio for each principal component. This ratio represents the proportion of the dataset's variance explained by each principal component.

cum\_var\_ratio = np.cumsum(ex\_var\_ratio): Computes the cumulative sum of the explained variance ratios. cum\_var\_ratio represents the cumulative explained variance as more principal components are included.

4.Determining the number of components needed to explain at least 95% of the variance:

num\_components = np.argmax(cum\_var\_ratio >= 0.95) + 1: Finds the index of the first element in cum\_var\_ratio that is greater than or equal to 0.95. The + 1 is added to account for zero-based indexing. This provides the number of principal components needed to explain at least 95% of the variance in the data.

The code performs data standardization, applies PCA to reduce dimensionality, and then determines the number of principal components needed to explain a specified percentage (95%) of the variance in the data. This information is crucial for understanding the trade-off between dimensionality reduction and preserving information in the dataset.

The astype() method is used to change the data type of a Pandas Series (or DataFrame column) to the specified type. In this case, it is converting these specific columns to the float data type, assuming that the original values in these columns are numeric and can be represented as floating-point numbers.

A white screen with black text

Description automatically generated

Output:-

R-squared with 8 PCA components: 0.9034013797858127

1.Splitting the data into training and testing sets:

train\_test\_split(df2\_pca[:, :num\_components], df2['price'], test\_size=0.2, random\_state=41): Splits the data into training and testing sets. The predictor variables (X) include the first num\_components principal components, and the target variable (y) is the 'price' column from the original DataFrame df2. The test\_size=0.2 parameter specifies that 20% of the data will be used for testing, and random\_state=41 ensures reproducibility.

2.Creating a linear regression model:

reg\_pca = LinearRegression(): Initializes a linear regression model using scikit-learn's LinearRegression class.

3.Fitting the model on the training data:

reg\_pca.fit(X\_train, y\_train): Fits the linear regression model to the training data, where X\_train is the training set of principal components, and y\_train is the corresponding target variable.

4.Making predictions on the test data:

y\_pred = reg\_pca.predict(X\_test): Uses the trained model to make predictions on the test set of principal components (X\_test).

5.Calculating R-squared:

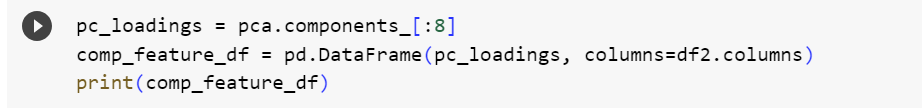
r2\_pca = r2\_score(y\_test, y\_pred): Calculates the R-squared value, a metric that measures the proportion of the variance in the target variable ('price') that is explained by the model's predictions.

6.Printing the R-squared value:

print(f"R-squared with {num\_components} PCA components: {r2\_pca}"): Prints the R-squared value along with the number of principal components used in the model.

This code assesses the performance of a linear regression model with a reduced number of principal components on the test data, providing insights into how well the model captures the variation in the target variable.

The astype() method is used to change the data type of a Pandas Series (or DataFrame column) to the specified type. In this case, it is converting these specific columns to the float data type, assuming that the original values in these columns are numeric and can be represented as floating-point numbers.



Output:-

A table of numbers and symbols

Description automatically generated

1.Extracting the loadings of the first 8 principal components:

pc\_loadings = pca.components\_[:8]: The pca.components\_ attribute contains the loadings of each principal component. This code extracts the loadings of the first 8 principal components and stores them in the pc\_loadings variable.

2.Creating a DataFrame to represent the relationship:

comp\_feature\_df = pd.DataFrame(pc\_loadings, columns=df2.columns): Creates a DataFrame comp\_feature\_df using the extracted loadings. Each row of the DataFrame corresponds to a principal component, and each column corresponds to an original feature from the DataFrame df2. This DataFrame represents the relationship between the original features and the principal components.

3.Printing the resulting DataFrame:

print(comp\_feature\_df): Prints the DataFrame that shows the loadings of each original feature on each of the first 8 principal components.

Each element in the DataFrame indicates the weight or contribution of a specific original feature to a particular principal component. Positive or negative values signify the direction and strength of the relationship. This kind of DataFrame is often useful for interpreting the meaning of the principal components in terms of the original features.

Keep in mind that interpretation of loadings can be complex and may depend on the specific context of the data and the analysis. In general, higher absolute values of loadings indicate a stronger relationship between a feature and a principal component.

The astype() method is used to change the data type of a Pandas Series (or DataFrame column) to the specified type. In this case, it is converting these specific columns to the float data type, assuming that the original values in these columns are numeric and can be represented as floating-point numbers.

A screenshot of a computer code

Description automatically generated

Output:- 1.Fit PCA to your training data:

pca = PCA(): Initializes a PCA object.

pca.fit(X\_train): Fits the PCA model to the training data (X\_train), where each row represents an observation and each column represents a principal component.

2.Calculate the cumulative explained variance:

cumulative\_explained\_variance = np.cumsum(pca.explained\_variance\_ratio\_): Calculates the cumulative sum of the explained variance ratios for each principal component. This array represents the cumulative amount of variance explained as more components are included.

3.Create a plot to visualize the cumulative explained variance:

plt.figure(figsize=(8, 6)): Specifies the size of the plot.

plt.plot(cumulative\_explained\_variance, marker='o'): Plots the cumulative explained variance with markers at each point.

plt.xlabel("Number of Components"): Sets the x-axis label.

plt.ylabel("Cumulative Explained Variance"): Sets the y-axis label.

plt.title("Cumulative Explained Variance by Principal Components"): Sets the title of the plot.

plt.grid(): Adds a grid to the plot.

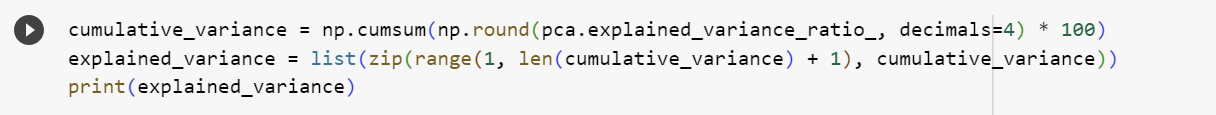
plt.show(): Displays the plot.

The resulting plot visually shows how much variance in the data is captured as the number of principal components increases, helping to inform decisions about the appropriate number of components to retain for dimensionality reduction.

A graph with a line

Description automatically generated

The astype() method is used to change the data type of a Pandas Series (or DataFrame column) to the specified type. In this case, it is converting these specific columns to the float data type, assuming that the original values in these columns are numeric and can be represented as floating-point numbers.



Output:-

[(1, 50.63999999999999), (2, 70.86999999999999), (3, 80.08999999999999), (4, 87.46999999999998), (5, 92.35999999999999), (6, 95.51999999999998), (7, 97.92999999999998), (8, 100.00999999999998)]

Component 1: Positive Contributions: 'length,' 'width,' 'curb\_weight,' 'engine\_size,' 'bore,' 'horse\_power,' and 'price.' Minor Positive Contributions: 'wheel\_base' and 'city\_mpg.' Negative Contributions: 'heights,' 'peak\_rpm,' 'highway\_mpg,' and 'fuel\_type\_gas.'

Component 2: Dominated by 'heights' and 'price' with opposing contributions. 'length,' 'heights,' and 'city\_mpg' also have discernible contributions.

Component 3: 'wheel\_base' and 'length' make substantial contributions. 'heights' and 'bore' contribute but in opposing directions.

Component 4: Primarily influenced by 'heights' and 'bore' with conflicting contributions. 'engine\_size' and 'stroke' also show notable contributions.

Component 5: 'curb\_weight' and 'engine\_size' hold significant contributions. 'stroke' and 'comprassion' make minor contributions.

Component 6: 'engine\_size' and 'city\_mpg' are the dominant contributors. 'bore' and 'highway\_mpg' show minor contributions.

Component 7: 'bore' and 'stroke' have significant contributions but in opposite directions. 'heights' and 'highway\_mpg' also exhibit noticeable contributions.

Component 8: 'heights' and 'comprassion' appear as primary contributors. 'wheel\_base' and 'fuel\_type\_gas' have smaller, opposing contributions. These coefficients help to interpret how each original variable contributes to the creation of principal components, providing insights into their significance in reducing the dataset's dimensionality while preserving essential information.