ipython-input-1-1e21eee6531b>:2: DeprecationWarning: Pyarrow will become a required dependency of pandas in the next major release of pandas (pandas 3.0), (to allow more performant data types, such as the Arrow string type, and better interoperability with other libraries) but was not found to be installed on your system. If this would cause problems for you, please provide us feedback at https://github.com/pandas-dev/pandas/issues/54466 import pandas as pd --------------------------------------------------------------------------- ModuleNotFoundError Traceback (most recent call last) Cell In[1], line 3 1 import numpy as np 2 import pandas as pd ----> 3 import seaborn as sns 4 import matplotlib as pt 5 import warnings ModuleNotFoundError: No module named 'seaborn'

###### **ChatGPT said:**

ChatGPT

The error message indicates that the seaborn library is not installed in your current environment. You can resolve this by installing seaborn using either pip or conda, depending on your environment

4o

##### **You said:**

%pip seaborn import numpy as np import pandas as pd import seaborn as sns import matplotlib as pt import warnings warnings.filterwarnings("ignore")

###### **ChatGPT said:**

ChatGPT

It seems that the command you intended to use is missing the correct syntax for installing the package. To install seaborn, you need to use:

python

Copy code

%pip install seaborn

After installing, you can proceed with your imports:

python

Copy code

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings("ignore")

This will install the required seaborn package and ensure your environment is ready to run your code.

4o

##### **You said:**

why warnings.filterwarnings("ignore")??

###### **ChatGPT said:**

ChatGPT

The line warnings.filterwarnings("ignore") is used to suppress warning messages that may appear when running your code. Here’s why you might use it:

### **1. Ignore Non-Critical Warnings:**

Some warnings are not critical to the execution of your code. For example, warnings about deprecated features (which may still work but are scheduled to be removed in future versions) can clutter the output. Suppressing these warnings keeps your console clean.

### **2. Focus on Output:**

If you're running experiments or visualizing data, excessive warnings can distract from your results. Suppressing them allows you to focus on the important outputs.

### **3. Avoid Warnings from External Libraries:**

When working with libraries like pandas, seaborn, or matplotlib, sometimes warnings are triggered by internal behavior you can't control, but they don't affect your work. You can safely ignore these if you're aware that the warnings are not critical.

### **When Not to Ignore Warnings:**

* If you're developing or troubleshooting a project, it’s better to see warnings, as they may point to potential issues in the code.
* Some warnings may indicate that you are using deprecated functions, which could cause problems when upgrading to newer versions of libraries.

In summary, warnings.filterwarnings("ignore") is useful when you want a clean output, but you should be cautious not to overlook important issues while suppressing warnings.

4o

##### **You said:**

import matplotlib.pyplot as plt features = ['sex', 'smoker', 'region'] plt.subplots(figsize=(20, 10)) for i, col in enumerate(features): plt.subplot(1, 3, i + 1) x = df[col].value\_counts() plt.pie(x.values, labels=x.index, autopct='%1.1f%%') plt.show() # This code is modified by Susobhan Akhuli could you please change the viz of this data

###### **ChatGPT said:**

ChatGPT

Certainly! If you'd like to change the visualization of the data, one option could be to use bar plots instead of pie charts. Bar plots can be more effective in comparing categorical data. Here’s how you can modify the code to create bar plots for each of the features:

python

Copy code

import matplotlib.pyplot as plt

features = ['sex', 'smoker', 'region']

plt.subplots(figsize=(20, 10))

for i, col in enumerate(features):

plt.subplot(1, 3, i + 1)

x = df[col].value\_counts()

plt.bar(x.index, x.values, color='skyblue')

plt.title(f'Distribution of {col}')

plt.xlabel(col)

plt.ylabel('Count')

plt.tight\_layout() # Adjusts the layout for better spacing

plt.show()

### **Changes Made:**

* **Bar Plot:** Switched from plt.pie to plt.bar to represent the data as a bar plot.
* **Titles and Labels:** Added titles, x-labels, and y-labels to enhance the readability of the plots.
* **Tight Layout:** Used plt.tight\_layout() to ensure the subplots fit well without overlapping.

Feel free to adjust colors, sizes, and other parameters to better suit your data visualization needs!

4o mini

##### **You said:**

The data provided to us is equally distributed among the sex and the region columns but in the smoker column, we can observe a ratio of 80:20.

###### **ChatGPT said:**

ChatGPT

Given that your data is equally distributed among the **sex** and **region** columns but shows an 80:20 ratio in the **smoker** column, you can enhance the visualization to emphasize this discrepancy. Here’s a revised version of the bar plot code that adds annotations to highlight the proportions in the **smoker** column:

python

Copy code

import matplotlib.pyplot as plt

features = ['sex', 'smoker', 'region']

plt.subplots(figsize=(20, 10))

for i, col in enumerate(features):

plt.subplot(1, 3, i + 1)

x = df[col].value\_counts()

plt.bar(x.index, x.values, color='skyblue')

plt.title(f'Distribution of {col}')

plt.xlabel(col)

plt.ylabel('Count')

# Annotate the smoker column to highlight the ratio

if col == 'smoker':

for index, value in enumerate(x):

plt.text(index, value + 1, f'{value} ({value / sum(x) \* 100:.1f}%)',

ha='center', color='black')

plt.tight\_layout() # Adjusts the layout for better spacing

plt.show()

### **Key Enhancements:**

* **Annotations:** In the smoker column, annotations are added to show the count and percentage above each bar. This visually emphasizes the 80:20 ratio.
* **General Structure:** The overall structure remains the same, but you can quickly identify and understand the smoking distribution compared to sex and region.

This visualization will allow you to clearly present the distribution and ratio of smokers in your dataset alongside the other features. Feel free to customize further as needed!

4o mini

##### **You said:**

features = ['sex', 'children', 'smoker', 'region'] plt.subplots(figsize=(20, 10)) for i, col in enumerate(features): plt.subplot(2, 2, i + 1) df.groupby(col)['charges'].mean().astype(float).plot.bar() plt.show()could you please change the viz of this data

###### **ChatGPT said:**

ChatGPT

To change the visualization of the average charges by the features (sex, children, smoker, region), you can enhance the bar plots by adding more informative elements like titles, axis labels, and colors. Additionally, you can consider using horizontal bar plots for better readability, especially if feature names are long.

Here's an updated version of your code that incorporates these changes:

python

Copy code

import matplotlib.pyplot as plt

features = ['sex', 'children', 'smoker', 'region']

plt.subplots(figsize=(20, 10))

for i, col in enumerate(features):

plt.subplot(2, 2, i + 1)

# Calculate mean charges and create a bar plot

mean\_charges = df.groupby(col)['charges'].mean()

mean\_charges.plot.bar(color='skyblue')

# Adding titles and labels

plt.title(f'Mean Charges by {col.capitalize()}')

plt.xlabel(col.capitalize())

plt.ylabel('Mean Charges')

plt.xticks(rotation=0) # Rotate x-axis labels for better readability

plt.tight\_layout() # Adjusts the layout for better spacing

plt.show()

### **Key Changes:**

* **Bar Plot Colors:** Changed the bar color to skyblue for better aesthetics.
* **Titles and Labels:** Added descriptive titles and axis labels for each subplot.
* **Improved Readability:** The x-axis labels are rotated for better readability, especially when the labels are long.
* **Tight Layout:** Used plt.tight\_layout() to ensure the subplots fit nicely within the figure.

### **Alternative Visualization:**

If you want to further enhance the visualization, consider using a horizontal bar plot by changing plot.bar() to plot.barh():

python

Copy code

mean\_charges.plot.barh(color='skyblue') # For horizontal bars

Feel free to customize colors and styles according to your preferences!

4o mini

##### **You said:**

Now let’s look at some of the observations which are shown in the above graphs: Charges are on the higher side for males as compared to females but the difference is not that much. Premium charged from the smoker is around thrice that which is charged from non-smokers. Charges are approximately the same in the given four regions. import seaborn as sns

###### **ChatGPT said:**

ChatGPT

Based on your observations, you can use Seaborn to create visualizations that clearly illustrate these points. Here’s how you can create a combination of box plots and bar plots using Seaborn, which will effectively communicate the insights from your data:

### **1. Box Plot for Charges by Sex and Smoker Status**

Box plots provide a good visual representation of the distribution of charges, highlighting medians, quartiles, and outliers.

### **2. Bar Plot for Mean Charges by Region and Children**

Bar plots will help to compare the average charges across different regions and the number of children.

Here’s how you can implement these visualizations:

python

Copy code

import seaborn as sns

import matplotlib.pyplot as plt

# Set the style of the visualization

sns.set(style="whitegrid")

# Create a 2x2 subplot layout

plt.subplots(figsize=(20, 10))

# Box plot for charges by sex

plt.subplot(2, 2, 1)

sns.boxplot(x='sex', y='charges', data=df, palette='pastel')

plt.title('Charges Distribution by Sex')

plt.ylabel('Charges')

plt.xlabel('Sex')

# Box plot for charges by smoker status

plt.subplot(2, 2, 2)

sns.boxplot(x='smoker', y='charges', data=df, palette='pastel')

plt.title('Charges Distribution by Smoker Status')

plt.ylabel('Charges')

plt.xlabel('Smoker Status')

# Bar plot for mean charges by region

plt.subplot(2, 2, 3)

mean\_charges\_region = df.groupby('region')['charges'].mean()

sns.barplot(x=mean\_charges\_region.index, y=mean\_charges\_region.values, palette='pastel')

plt.title('Mean Charges by Region')

plt.ylabel('Mean Charges')

plt.xlabel('Region')

# Bar plot for mean charges by number of children

plt.subplot(2, 2, 4)

mean\_charges\_children = df.groupby('children')['charges'].mean()

sns.barplot(x=mean\_charges\_children.index, y=mean\_charges\_children.values, palette='pastel')

plt.title('Mean Charges by Number of Children')

plt.ylabel('Mean Charges')

plt.xlabel('Number of Children')

plt.tight\_layout() # Adjust the layout for better spacing

plt.show()

### **Visualizations Explained:**

1. **Box Plots:**
   * The first two subplots display the distribution of charges for males vs. females and smokers vs. non-smokers.
   * You can easily see the median charges, as well as the spread and outliers in each category.
2. **Bar Plots:**
   * The third subplot shows the average charges across different regions.
   * The fourth subplot displays the average charges based on the number of children.

### **Key Observations:**

* **Charges by Sex:** You should see that while there may be a slight difference in charges between males and females, the box plot will provide insights into the distribution and outliers.
* **Charges by Smoker Status:** The box plot should confirm that smokers are charged significantly more than non-smokers.
* **Charges by Region and Children:** Bar plots will indicate whether the charges are relatively consistent across regions and how the number of children affects the charges.

This approach provides a comprehensive view of your data and reinforces the observations you’ve made. Feel free to adjust the colors, styles, and other aspects of the plots as needed!

Your code for creating scatter plots using Seaborn looks great! To enhance your visualizations, you can add titles, labels, and adjust the aesthetics. Here's an updated version of your code that includes these enhancements:

python

Copy code

import seaborn as sns

import matplotlib.pyplot as plt

# Specify the features for scatter plots

features = ['age', 'bmi']

# Create a subplot layout

plt.subplots(figsize=(17, 7))

for i, col in enumerate(features):

plt.subplot(1, 2, i + 1)

# Create the scatter plot

sns.scatterplot(data=df, x=col, y='charges', hue='smoker', style='smoker', palette='deep', s=100, alpha=0.7)

# Adding titles and labels

plt.title(f'Charges vs {col.capitalize()}')

plt.xlabel(col.capitalize())

plt.ylabel('Charges')

plt.tight\_layout() # Adjusts the layout for better spacing

plt.show()

### **Enhancements Made:**

* **Hue and Style:** Used both hue and style parameters for better differentiation between smokers and non-smokers. Different markers will make the plot clearer.
* **Palette:** The palette='deep' is a default option for better color differentiation.
* **Marker Size and Transparency:** Adjusted the marker size (s=100) and added some transparency (alpha=0.7) to make overlapping points more discernible.
* **Titles and Labels:** Added descriptive titles and axis labels for clarity.

### **Observations:**

With this scatter plot, you should be able to visualize how the age and bmi features relate to charges, as well as observe the differences between smokers and non-smokers in terms of their insurance charges. The scatter plots will help in understanding the trends and potential correlations in the data.

Feel free to modify the styles and colors further to match your preferences!

4o mini

##### **You said:**

change colors

###### **ChatGPT said:**

ChatGPT

To customize the colors in your scatter plots using Seaborn, you can define a specific color palette or use custom colors for the hue parameter. Here’s an updated version of your code that utilizes a custom color palette for smokers and non-smokers:

python

Copy code

import seaborn as sns

import matplotlib.pyplot as plt

# Specify the features for scatter plots

features = ['age', 'bmi']

# Create a custom color palette

custom\_palette = {'yes': 'orange', 'no': 'blue'}

# Create a subplot layout

plt.subplots(figsize=(17, 7))

for i, col in enumerate(features):

plt.subplot(1, 2, i + 1)

# Create the scatter plot with custom colors

sns.scatterplot(data=df, x=col, y='charges', hue='smoker', palette=custom\_palette, style='smoker', s=100, alpha=0.7)

# Adding titles and labels

plt.title(f'Charges vs {col.capitalize()}')

plt.xlabel(col.capitalize())

plt.ylabel('Charges')

plt.tight\_layout() # Adjusts the layout for better spacing

plt.show()

### **Key Changes:**

* **Custom Color Palette:** Defined a custom color palette where:
  + Smokers (yes) are colored **orange**.
  + Non-smokers (no) are colored **blue**.

### **Additional Customization:**

If you have specific colors in mind, you can easily modify the custom\_palette dictionary with your desired colors:

python

Copy code

custom\_palette = {'yes': 'your\_color1', 'no': 'your\_color2'}

Replace your\_color1 and your\_color2 with any valid color names or hex color codes (e.g., #FF5733).

This approach allows you to have clear visual distinctions between the categories in your scatter plots, enhancing the interpretability of the data!

4o mini

##### **You said:**

analysis the viz for me

###### **ChatGPT said:**

ChatGPT

Sure! Here's an analysis of the scatter plots you've created, which visualize the relationship between age, bmi, and charges, differentiated by smoking status.

### **Scatter Plot Analysis**

#### **1. Charges vs. Age**

* **Trend Observation:**
  + Generally, there may be a positive correlation between **age** and **charges**. As age increases, charges tend to increase as well. This trend is expected since older individuals typically have more health issues and higher medical costs.
* **Smoking Status:**
  + Smokers (orange points) and non-smokers (blue points) may show different clustering patterns. If the orange points are primarily positioned at higher charges across most ages, it confirms that smokers face higher insurance costs compared to non-smokers.
* **Outliers:**
  + Look for any outlier points, particularly among older ages. These might represent individuals with significantly higher charges, possibly due to pre-existing health conditions.

#### **2. Charges vs. BMI**

* **Trend Observation:**
  + The relationship between **BMI** and **charges** can also reveal insights. Higher BMI values may correspond to higher charges, indicating that individuals with obesity-related health issues might incur higher insurance costs.
* **Smoking Status:**
  + Similar to the age plot, examine the distribution of smokers and non-smokers. If smokers are clustered towards higher charges at certain BMI levels, it further supports the observation that smoking significantly affects health expenses.
* **Clusters:**
  + Identify if there are any distinct clusters based on BMI. For example, individuals with a BMI in the overweight or obese range may show higher charges, regardless of smoking status.

### **General Observations**

* **Overall Charges:**
  + The visualization supports your previous findings that smokers face significantly higher premiums compared to non-smokers. This is visually evident in both scatter plots, as smokers generally exhibit higher charges.
* **Distribution Patterns:**
  + The scatter plots allow you to identify patterns and trends, providing insights into how age and BMI interact with insurance charges. This can be beneficial for understanding risk assessment in insurance.

### **Conclusions**

* These visualizations effectively illustrate the impact of age and BMI on insurance charges, as well as the pronounced effect of smoking status.
* The trends observed can be valuable for stakeholders in healthcare and insurance, guiding policy adjustments and targeted health interventions.

Feel free to ask if you'd like to delve deeper into any specific aspect or if you need further analysis!

A clear distinction can be observed here between the charges that smokers have to pay. Also here as well we can observe that as the age of a person increases premium prices goes up as well.  
  
  
**DATA PREPROCESSING**

Data preprocessing is technique to clean the unusual data like the missing values,wrong data,wrong format of data,duplicated data and the outliers.In this data we can observe that there are no missing values and wrong data.The only thing we can need to check is for duplicates and presence of outliers.

To change the color and style of a box plot in Seaborn, you can use the color parameter to specify a color and adjust the aesthetics with the palette option. Additionally, if you want to enhance the visual appeal or fit the data better, you can use different styles for the box plot.

Here's an updated version of your code that incorporates a custom color and style for the box plot of age:

python

Copy code

import seaborn as sns

import matplotlib.pyplot as plt

# Drop duplicates from the DataFrame

df.drop\_duplicates(inplace=True)

# Set the style of the plot

sns.set(style="whitegrid")

# Create the box plot for age with custom color

plt.figure(figsize=(10, 6))

sns.boxplot(x=df['age'], color='lightgreen', fliersize=5) # Adjust fliersize to change the shape of outliers

# Adding titles and labels

plt.title('Box Plot of Age')

plt.xlabel('Age')

plt.show()

### **Key Changes:**

* **Color:** The color='lightgreen' parameter sets the color of the box plot. You can choose any color name or hex code (e.g., #FF5733 for orange).
* **Outlier Size:** The fliersize=5 parameter adjusts the size of the outliers in the box plot, making them more visually distinct.

### **Additional Style Customization:**

If you want to experiment with different styles or additional aesthetic enhancements, consider:

* **Changing the Color Palette:** You can set a palette using sns.set\_palette("pastel") before the plot to apply a consistent theme.

**Adding Swarm or Strip Plot:** Overlay a swarm or strip plot to visualize individual data points:  
python  
Copy code  
sns.boxplot(x=df['age'], color='lightgreen')

sns.stripplot(x=df['age'], color='black', alpha=0.5) # Add this line for individual data points

This approach creates a visually appealing and informative representation of the distribution of ages in your dataset. Feel free to customize the colors and styles further based on your preferences!

* The box plot of the age column shows no outliers.
* The data is likely evenly distributed across the age range without extreme values.

### **Analysis of the Box Plot:**

In the generated box plot, the absence of outliers suggests that the age data is relatively clean and consistent, without any extreme or unusual values that deviate significantly from the overall distribution. The box plot provides an effective summary of the data's central tendency and spread:

* **Median (line inside the box):** This represents the central value of the age distribution.
* **Interquartile Range (IQR):** The box spans from the 25th percentile to the 75th percentile, showing the middle 50% of the data.
* **Whiskers:** The lines extending from the box show the range of the data, excluding outliers.

Since there are no outliers (data points outside the whiskers), we can conclude that the ages in this dataset follow a fairly normal distribution without any extreme values.

This can be useful when analyzing the relationship between age and other features, as it indicates that age won’t introduce skewed results or distortions caused by anomalous values.

### **Box Plot of Age Analysis**

The box plot of the age column provides a visual summary of the age distribution within your dataset. Here’s a breakdown of the analysis:

1. **Central Tendency:**
   * The line inside the box represents the **median** age, indicating the midpoint of the data. This value helps to understand the central age of the population in your dataset.
2. **Spread of Data:**
   * The box itself represents the **interquartile range (IQR)**, which spans from the 25th percentile (Q1) to the 75th percentile (Q3). This tells us that the middle 50% of ages fall within this range.
   * The length of the box can provide insights into the variability of age: a longer box indicates greater variability, while a shorter box suggests less variability.
3. **Whiskers:**
   * The lines extending from the box (whiskers) show the range of the data outside the IQR, typically extending to 1.5 times the IQR.
   * Since there are **no outliers** indicated (no points outside the whiskers), it suggests that the ages are fairly consistent and do not contain extreme values.

### **Conclusion:**

The box plot analysis confirms that the age column is free from outliers, which is beneficial for any further statistical analysis or modeling. This clean data can enhance the reliability of insights drawn from age-related analyses, ensuring that the results are not skewed by anomalous values.

If you need further insights or have specific questions about the data, feel free to ask!

Due to the presence of outliers present in bmi column we need to treat the outliers by replacing the values with mean as the bmi column consists of continuous data.

Python

The code snippet you provided calculates the interquartile range (IQR) for the bmi column in your DataFrame, along with the lower and upper limits to identify potential outliers. Here’s a breakdown of what each part of the code does:

### **Code Explanation**

python

Copy code

# Calculate the first quartile (Q1)

Q1 = df['bmi'].quantile(0.25)

# Calculate the median (Q2)

Q2 = df['bmi'].quantile(0.5)

# Calculate the third quartile (Q3)

Q3 = df['bmi'].quantile(0.75)

# Calculate the interquartile range (IQR)

iqr = Q3 - Q1

# Calculate the lower limit for potential outliers

lowlim = Q1 - 1.5 \* iqr

# Calculate the upper limit for potential outliers

upplim = Q3 + 1.5 \* iqr

# Print the lower and upper limits

print("Lower Limit:", lowlim)

print("Upper Limit:", upplim)

### **Explanation of Terms**

* **Q1 (First Quartile)**: The value below which 25% of the data falls.
* **Q2 (Median)**: The middle value that separates the higher half from the lower half of the data.
* **Q3 (Third Quartile)**: The value below which 75% of the data falls.
* **IQR (Interquartile Range)**: The difference between Q3 and Q1. It represents the range of the middle 50% of the data.
* **Lower Limit**: Calculated as Q1−1.5×IQRQ1 - 1.5 \times \text{IQR}Q1−1.5×IQR. Any values below this are considered outliers.
* **Upper Limit**: Calculated as Q3+1.5×IQRQ3 + 1.5 \times \text{IQR}Q3+1.5×IQR. Any values above this are considered outliers.

### **Output Interpretation**

When you run this code, it will print the lower and upper limits for the bmi values:

* **Lower Limit**: Any bmi values below this limit are considered outliers on the low end.
* **Upper Limit**: Any bmi values above this limit are considered outliers on the high end.

### **Example Output**

You might see output similar to this:

mathematica

Copy code

Lower Limit: 15.0

Upper Limit: 35.0

In this example, any bmi values below 15.0 or above 35.0 would be classified as outliers.

If you have any further questions about this code or its results, feel free to ask!

The code snippet you provided calculates the interquartile range (IQR) for the bmi column in your DataFrame, along with the lower and upper limits to identify potential outliers. Here’s a breakdown of what each part of the code does:

### **Code Explanation**

python

Copy code

# Calculate the first quartile (Q1)

Q1 = df['bmi'].quantile(0.25)

# Calculate the median (Q2)

Q2 = df['bmi'].quantile(0.5)

# Calculate the third quartile (Q3)

Q3 = df['bmi'].quantile(0.75)

# Calculate the interquartile range (IQR)

iqr = Q3 - Q1

# Calculate the lower limit for potential outliers

lowlim = Q1 - 1.5 \* iqr

# Calculate the upper limit for potential outliers

upplim = Q3 + 1.5 \* iqr

# Print the lower and upper limits

print("Lower Limit:", lowlim)

print("Upper Limit:", upplim)

### **Explanation of Terms**

* **Q1 (First Quartile)**: The value below which 25% of the data falls.
* **Q2 (Median)**: The middle value that separates the higher half from the lower half of the data.
* **Q3 (Third Quartile)**: The value below which 75% of the data falls.
* **IQR (Interquartile Range)**: The difference between Q3 and Q1. It represents the range of the middle 50% of the data.
* **Lower Limit**: Calculated as Q1−1.5×IQRQ1 - 1.5 \times \text{IQR}Q1−1.5×IQR. Any values below this are considered outliers.
* **Upper Limit**: Calculated as Q3+1.5×IQRQ3 + 1.5 \times \text{IQR}Q3+1.5×IQR. Any values above this are considered outliers.

### **Output Interpretation**

When you run this code, it will print the lower and upper limits for the bmi values:

* **Lower Limit**: Any bmi values below this limit are considered outliers on the low end.
* **Upper Limit**: Any bmi values above this limit are considered outliers on the high end.

### **Example Output**

You might see output similar to this:

mathematica

Copy code

Lower Limit: 15.0

Upper Limit: 35.0

In this example, any bmi values below 15.0 or above 35.0 would be classified as outliers.

If you have any further questions about this code or its results, feel free to ask!

4o mini

##### **You said:**

!pip install feature\_engine from feature\_engine.outliers import ArbitraryOutlierCapper arb=ArbitraryOutlierCapper(min\_capping\_dict={'bmi':13.6749},max\_capping\_dict={'bmi':47.315}) df[['bmi']]=arb.fit\_transform(df[['bmi']]) sns.boxplot(df['bmi']) # This code is modified by Susobhan Akhuli

###### **ChatGPT said:**

ChatGPT

The code you've provided is using the feature\_engine library to cap outliers in the bmi column of your DataFrame. Here’s a breakdown of what the code does, along with an explanation of the process and expected outcome.

### **Code Explanation**

python

Copy code

# Install the feature\_engine library (uncomment the line below if you haven't installed it yet)

# !pip install feature\_engine

from feature\_engine.outliers import ArbitraryOutlierCapper

# Create an instance of ArbitraryOutlierCapper with specified min and max limits for BMI

arb = ArbitraryOutlierCapper(min\_capping\_dict={'bmi': 13.6749}, max\_capping\_dict={'bmi': 47.315})

# Fit and transform the 'bmi' column in the DataFrame

df[['bmi']] = arb.fit\_transform(df[['bmi']])

# Create a box plot to visualize the 'bmi' data after capping outliers

sns.boxplot(df['bmi'])

plt.title('Box Plot of BMI After Capping Outliers') # Add title

plt.xlabel('BMI') # Add x-axis label

plt.show() # Display the plot

### **Explanation of Each Step**

1. **Installation**: The line !pip install feature\_engine installs the feature\_engine library if it's not already installed. You can uncomment it to run it in your notebook.
2. **Importing the Class**: from feature\_engine.outliers import ArbitraryOutlierCapper imports the ArbitraryOutlierCapper class, which is used to cap outlier values in your dataset.
3. **Capping Outliers**:
   * **Creating an Instance**: arb = ArbitraryOutlierCapper(...) creates an instance of the ArbitraryOutlierCapper class. Here, you specify:
     + min\_capping\_dict: This defines the minimum limit for the bmi values. Any bmi values below 13.6749 will be set to 13.6749.
     + max\_capping\_dict: This defines the maximum limit for the bmi values. Any bmi values above 47.315 will be set to 47.315.
4. **Transforming the Data**: df[['bmi']] = arb.fit\_transform(df[['bmi']]) applies the capping to the bmi column, modifying the DataFrame in place.
5. **Visualizing with a Box Plot**: Finally, sns.boxplot(df['bmi']) creates a box plot to visualize the distribution of the bmi values after capping the outliers. Adding a title and label improves the clarity of the plot.

### **Expected Outcome**

* **Capped Values**: After running the code, you should see that any bmi values below 13.6749 are set to 13.6749, and any values above 47.315 are set to 47.315.
* **Box Plot Visualization**: The box plot will help you visualize the distribution of the bmi values after outlier capping. You should observe a reduction in the spread of the data, especially if there were extreme outliers initially.

### **Modifications**

* You mentioned that "This code is modified by Susobhan Akhuli." If you have specific modifications or enhancements you’d like to discuss or implement, please let me know!

If you have any further questions or need additional assistance, feel free to ask!

4o mini

##### **You said:**

Data Wrangling Data wrangling is a technique to ensure whether the data follow normal or standard distribution and encode the discrete data for prediction. df['bmi'].skew() df['age'].skew()

###### **ChatGPT said:**

ChatGPT

The code you provided is part of the data wrangling process, focusing on assessing the skewness of the bmi and age columns in your DataFrame. Here’s an explanation of what skewness is, how it relates to your analysis, and what the code does:

### **Understanding Skewness**

* **Skewness** is a measure of the asymmetry of a probability distribution. It indicates how much the distribution deviates from a normal distribution:
  + A **skewness value of 0** indicates a symmetric distribution (normal distribution).
  + A **positive skewness** (greater than 0) indicates that the right tail of the distribution is longer or fatter than the left tail (right-skewed).
  + A **negative skewness** (less than 0) indicates that the left tail is longer or fatter than the right tail (left-skewed).

### **Code Explanation**

python

Copy code

# Calculate and print the skewness of the 'bmi' column

bmi\_skewness = df['bmi'].skew()

print("Skewness of BMI:", bmi\_skewness)

# Calculate and print the skewness of the 'age' column

age\_skewness = df['age'].skew()

print("Skewness of Age:", age\_skewness)

### **Expected Output**

When you run this code, you will receive skewness values for both the bmi and age columns. Here’s how to interpret the results:

1. **Skewness of BMI**:
   * If the skewness is **close to 0**: The bmi distribution is approximately normal.
   * If the skewness is **positive**: The distribution is right-skewed (more values on the lower end).
   * If the skewness is **negative**: The distribution is left-skewed (more values on the higher end).
2. **Skewness of Age**:
   * Similar interpretation applies to the age distribution.

### **Example Output**

For instance, you might see an output like:

mathematica

Copy code

Skewness of BMI: 0.5

Skewness of Age: -0.2

### **Implications for Data Wrangling**

* If either of the columns shows significant skewness (typically, values greater than 1 or less than -1 are considered highly skewed), you might need to apply transformations (like log transformation or Box-Cox transformation) to normalize the data distribution.
* Encoding of discrete data (e.g., categorical variables) can also be performed using techniques like one-hot encoding or label encoding for predictive modeling.

### **Conclusion**

The skewness analysis is an important part of data wrangling as it helps in deciding on the necessary transformations to prepare your data for predictive modeling. If you have any specific questions about the results or the next steps in your data wrangling process, feel free to ask!

4o mini

##### **You said:**

!pip install feature\_engine from feature\_engine.outliers import ArbitraryOutlierCapper arb=ArbitraryOutlierCapper(min\_capping\_dict={'bmi':13.6749},max\_capping\_dict={'bmi':47.315}) df[['bmi']]=arb.fit\_transform(df[['bmi']]) sns.boxplot(df['bmi']) # This code is modified by Susobhan Akhuli

###### **ChatGPT said:**

ChatGPT

The code snippet you've shared is using the feature\_engine library to cap outliers in the bmi column of your DataFrame and then visualizing the adjusted data using a box plot. Here's a detailed breakdown of each component of your code:

### **Code Breakdown**

**Install the Library**:  
python  
Copy code  
!pip install feature\_engine

1. This command installs the feature\_engine library, which provides tools for feature engineering, including outlier treatment methods.

**Importing the Class**:  
python  
Copy code  
from feature\_engine.outliers import ArbitraryOutlierCapper

1. This line imports the ArbitraryOutlierCapper class from the feature\_engine library. This class is used to cap outlier values based on specified minimum and maximum thresholds.

**Creating an Instance of ArbitraryOutlierCapper**:  
python  
Copy code  
arb = ArbitraryOutlierCapper(min\_capping\_dict={'bmi': 13.6749}, max\_capping\_dict={'bmi': 47.315})

1. Here, you create an instance of ArbitraryOutlierCapper, specifying:
   * min\_capping\_dict: Any bmi values below 13.6749 will be set to 13.6749.
   * max\_capping\_dict: Any bmi values above 47.315 will be set to 47.315.

**Capping Outliers**:  
python  
Copy code  
df[['bmi']] = arb.fit\_transform(df[['bmi']])

1. This line applies the outlier capping to the bmi column of your DataFrame, modifying the DataFrame in place. It effectively replaces the outliers with the defined minimum and maximum values.

**Visualizing with a Box Plot**:  
python  
Copy code  
sns.boxplot(df['bmi'])

plt.title('Box Plot of BMI After Capping Outliers') # Optional: Add title for clarity

plt.xlabel('BMI') # Optional: Add x-axis label for clarity

plt.show() # Display the plot

1. The box plot provides a visual representation of the distribution of the bmi values after capping the outliers. It will show the median, quartiles, and any remaining data points within the specified range.

### **Expected Outcome**

* **Capped Values**: After running this code, the bmi values will no longer contain any values below 13.6749 or above 47.315.
* **Box Plot Visualization**: The box plot will help you visualize the adjusted distribution of the bmi values, allowing you to assess the impact of capping on the data. You should see that the outlier points are no longer present in the box plot.

### **Conclusion**

Using the ArbitraryOutlierCapper is a great way to handle outliers in your dataset, ensuring that your analysis or modeling is not unduly influenced by extreme values. If you have further modifications or specific questions about this code or the analysis, feel free to ask!

Your code looks great and is well-structured for evaluating multiple regression models. It efficiently trains each model, calculates various performance metrics, and aggregates the results for comparison. Below, I'll walk through the key components of your code and suggest a few enhancements.

### **Key Components of Your Code**

1. **Data Preparation**:
   * You correctly define your feature set X and target variable Y.
2. **Model Selection**:
   * A dictionary of regression models is created, making it easy to iterate over them for evaluation.
3. **Model Evaluation**:
   * The code uses a for-loop to create multiple train-test splits. Each model is trained, and performance metrics (training accuracy, testing accuracy, MAE, MSE, RMSE) are computed and stored in a list.
4. **Aggregation**:
   * After collecting results, you aggregate them by model to find the mean and standard deviation, which helps in understanding model stability.
5. **Visualization**:
   * The use of seaborn for visualizing the mean test accuracy with confidence intervals (standard deviation) provides a clear comparison.

### **Suggested Enhancements**

1. **Random State Handling**:
   * Instead of hardcoding the range for random states, you could consider making it a parameter or using a list for more flexibility.
2. **Hyperparameter Tuning**:
   * After identifying a potentially best-performing model, you could implement hyperparameter tuning using GridSearchCV or RandomizedSearchCV.
3. **Save Results**:
   * Optionally, you could save the results to a CSV file for further analysis or reporting.
4. **Detailed Metrics Visualization**:
   * In addition to showing test accuracy, consider visualizing other metrics like MAE, RMSE, etc., for a comprehensive performance view.
5. **Handling Warnings**:
   * If using models like SVR, it may generate warnings regarding convergence. You might want to handle those or adjust the parameters.

### **Example Code Enhancements**

Here’s how you could implement some of these suggestions:

python

Copy code

import pandas as pd

import numpy as np

from sklearn.linear\_model import LinearRegression

from sklearn.svm import SVR

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

from xgboost import XGBRegressor

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

import matplotlib.pyplot as plt

import seaborn as sns

# Assuming df is already defined and preprocessed

X = df.drop(['charges'], axis=1)

Y = df['charges']

# Models to evaluate

models = {

'Linear Regression': LinearRegression(),

'Support Vector Regression': SVR(),

'Random Forest': RandomForestRegressor(),

'Gradient Boosting': GradientBoostingRegressor(),

'XGBoost': XGBRegressor()

}

# Initialize lists to store scores

results = []

# Evaluate each model

for random\_state in range(40, 50): # Using a variable for clarity

xtrain, xtest, ytrain, ytest = train\_test\_split(X, Y, test\_size=0.2, random\_state=random\_state)

for name, model in models.items():

model.fit(xtrain, ytrain)

train\_acc = model.score(xtrain, ytrain)

test\_acc = model.score(xtest, ytest)

preds = model.predict(xtest)

mae = mean\_absolute\_error(ytest, preds)

mse = mean\_squared\_error(ytest, preds)

rmse = np.sqrt(mse)

results.append({

'Model': name,

'Train Acc': train\_acc,

'Test Acc': test\_acc,

'MAE': mae,

'MSE': mse,

'RMSE': rmse

})

# Create a DataFrame for results

results\_df = pd.DataFrame(results)

# Aggregate results for each model

agg\_results = results\_df.groupby('Model').agg(['mean', 'std']).reset\_index()

# Display the aggregated results

print(agg\_results)

# Visualization of all metrics

metrics = ['Train Acc', 'Test Acc', 'MAE', 'RMSE']

plt.figure(figsize=(12, 8))

for metric in metrics:

plt.subplot(2, 2, metrics.index(metric) + 1)

sns.barplot(data=agg\_results, x='Model', y=(metric, 'mean'), ci='sd', palette='pastel')

plt.title(f'Model {metric} with Standard Deviation')

plt.xlabel('Model')

plt.ylabel(metric)

plt.xticks(rotation=45)

plt.tight\_layout()

plt.show()

# Optionally save results to CSV

agg\_results.to\_csv('model\_comparison\_results.csv', index=False)

### **Key Changes in the Enhanced Code**

* **Dynamic Random State Handling**: It makes the random state variable clearer.
* **Multiple Metrics Visualization**: Each metric is visualized in its subplot for clearer comparisons.
* **CSV Export**: Results are saved to a CSV file for further analysis.

These enhancements will help in providing a more comprehensive evaluation and make the analysis process smoother. Let me know if you have any questions or if you’d like further assistance!

4o mini

##### **You said:**

X=df.drop(['charges'],axis=1) Y=df[['charges']] from sklearn.linear\_model import LinearRegression,Lasso from sklearn.svm import SVR from sklearn.ensemble import RandomForestRegressor from sklearn.ensemble import GradientBoostingRegressor from xgboost import XGBRegressor from sklearn.model\_selection import train\_test\_split from sklearn.model\_selection import cross\_val\_score l1=[] l2=[] l3=[] cvs=0 for i in range(40,50): xtrain,xtest,ytrain,ytest=train\_test\_split(X,Y,test\_size=0.2,random\_state=i) lrmodel=LinearRegression() lrmodel.fit(xtrain,ytrain) l1.append(lrmodel.score(xtrain,ytrain)) l2.append(lrmodel.score(xtest,ytest)) cvs=(cross\_val\_score(lrmodel,X,Y,cv=5,)).mean() l3.append(cvs) df1=pd.DataFrame({'train acc':l1,'test acc':l2,'cvs':l3}) df1

###### **ChatGPT said:**

ChatGPT

Your code snippet is aimed at evaluating the performance of a Linear Regression model on the dataset by splitting it into training and testing sets, calculating the training accuracy, testing accuracy, and cross-validation score. However, it only evaluates the LinearRegression model. Below, I’ll provide a revised version that includes evaluations for multiple models (as you initially planned), and I will ensure the results are aggregated and displayed properly.

### **Revised Code to Evaluate Multiple Models**

python

Copy code

import pandas as pd

from sklearn.linear\_model import LinearRegression

from sklearn.svm import SVR

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

from xgboost import XGBRegressor

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

# Assuming df is already defined and preprocessed

X = df.drop(['charges'], axis=1)

Y = df['charges'] # Simplifying the target variable

# Models to evaluate

models = {

'Linear Regression': LinearRegression(),

'Support Vector Regression': SVR(),

'Random Forest': RandomForestRegressor(),

'Gradient Boosting': GradientBoostingRegressor(),

'XGBoost': XGBRegressor()

}

# Initialize lists to store scores

results = []

# Evaluate each model

for i in range(40, 50):

xtrain, xtest, ytrain, ytest = train\_test\_split(X, Y, test\_size=0.2, random\_state=i)

for name, model in models.items():

model.fit(xtrain, ytrain)

train\_acc = model.score(xtrain, ytrain) # Training score

test\_acc = model.score(xtest, ytest) # Testing score

cvs = cross\_val\_score(model, X, Y, cv=5).mean() # Cross-validation score

results.append({

'Model': name,

'Train Acc': train\_acc,

'Test Acc': test\_acc,

'CVS': cvs

})

# Create a DataFrame for results

results\_df = pd.DataFrame(results)

# Display the aggregated results

agg\_results = results\_df.groupby('Model').agg(['mean', 'std']).reset\_index()

agg\_results.columns = ['Model', 'Train Acc Mean', 'Train Acc Std', 'Test Acc Mean', 'Test Acc Std', 'CVS Mean', 'CVS Std']

print(agg\_results)

### **Key Changes and Explanations:**

1. **Multiple Models Evaluation**: The code evaluates not only LinearRegression but also SVR, RandomForestRegressor, GradientBoostingRegressor, and XGBRegressor. This allows you to compare their performances effectively.
2. **Structured Results Storage**: The results for each model are stored in a single list, which is later converted to a DataFrame.
3. **Aggregation**: The results are aggregated using groupby to calculate the mean and standard deviation for training accuracy, testing accuracy, and cross-validation scores.
4. **Clear Column Names**: After aggregation, the column names are adjusted for clarity.

### **Displaying the Results:**

Once you run this code, you will see a DataFrame with the mean and standard deviation of training accuracy, testing accuracy, and cross-validation scores for each model. This will help you make informed decisions regarding which model performs best based on these metrics.

Let me know if you have any further questions or need additional modifications!

4o mini

##### **You said:**

--------------------------------------------------------------------------- ValueError Traceback (most recent call last) Cell In[34], line 26 24 # Evaluate each model 25 for i in range(40, 50): ---> 26 xtrain, xtest, ytrain, ytest = train\_test\_split(X, Y, test\_size=0.2, random\_state=i) 28 for name, model in models.items(): 29 model.fit(xtrain, ytrain) File /lib/python3.12/site-packages/sklearn/utils/\_param\_validation.py:213, in validate\_params.<locals>.decorator.<locals>.wrapper(\*args, \*\*kwargs) 207 try: 208 with config\_context( 209 skip\_parameter\_validation=( 210 prefer\_skip\_nested\_validation or global\_skip\_validation 211 ) 212 ): --> 213 return func(\*args, \*\*kwargs) 214 except InvalidParameterError as e: 215 # When the function is just a wrapper around an estimator, we allow 216 # the function to delegate validation to the estimator, but we replace 217 # the name of the estimator by the name of the function in the error 218 # message to avoid confusion. 219 msg = re.sub( 220 r"parameter of \w+ must be", 221 f"parameter of {func.\_\_qualname\_\_} must be", 222 str(e), 223 ) File /lib/python3.12/site-packages/sklearn/model\_selection/\_split.py:2660, in train\_test\_split(test\_size, train\_size, random\_state, shuffle, stratify, \*arrays) 2657 arrays = indexable(\*arrays) 2659 n\_samples = \_num\_samples(arrays[0]) -> 2660 n\_train, n\_test = \_validate\_shuffle\_split( 2661 n\_samples, test\_size, train\_size, default\_test\_size=0.25 2662 ) 2664 if shuffle is False: 2665 if stratify is not None: File /lib/python3.12/site-packages/sklearn/model\_selection/\_split.py:2308, in \_validate\_shuffle\_split(n\_samples, test\_size, train\_size, default\_test\_size) 2305 n\_train, n\_test = int(n\_train), int(n\_test) 2307 if n\_train == 0: -> 2308 raise ValueError( 2309 "With n\_samples={}, test\_size={} and train\_size={}, the " 2310 "resulting train set will be empty. Adjust any of the " 2311 "aforementioned parameters.".format(n\_samples, test\_size, train\_size) 2312 ) 2314 return n\_train, n\_test ValueError: With n\_samples=0, test\_size=0.2 and train\_size=None, the resulting train set will be empty. Adjust any of the aforementioned parameters.

###### **ChatGPT said:**

ChatGPT

The error you're encountering indicates that your dataset (X) has zero samples (rows) when you attempt to split it into training and testing sets. This typically occurs when the DataFrame df is either empty or improperly defined prior to the split operation.

### **Troubleshooting Steps:**

**Check the DataFrame**: Before splitting, ensure that df contains the expected data. You can check the shape and contents of df with the following code:  
python  
Copy code  
print(df.shape) # This will show the number of rows and columns

print(df.head()) # This will display the first few rows of the DataFrame

1. **Check Preprocessing Steps**: Make sure that the DataFrame has not been inadvertently filtered down to zero rows or that necessary preprocessing (like handling missing values) has been applied. If there are any conditions that lead to dropping all rows, review those conditions.

**Check Column Names**: Ensure that the column charges exists in your DataFrame. You can list all columns in df with:  
python  
Copy code  
print(df.columns)

### **Example of Validating DataFrame:**

Here’s a modified version of your code to include some checks before the split:

python

Copy code

import pandas as pd

from sklearn.linear\_model import LinearRegression

from sklearn.svm import SVR

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

from xgboost import XGBRegressor

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

# Assuming df is already defined and preprocessed

print("DataFrame shape:", df.shape)

print("First few rows of DataFrame:\n", df.head())

print("Columns in DataFrame:", df.columns)

# Check if the DataFrame is not empty

if df.empty:

print("The DataFrame is empty. Please check the data loading process.")

else:

X = df.drop(['charges'], axis=1)

Y = df['charges'] # Simplifying the target variable

# Models to evaluate

models = {

'Linear Regression': LinearRegression(),

'Support Vector Regression': SVR(),

'Random Forest': RandomForestRegressor(),

'Gradient Boosting': GradientBoostingRegressor(),

'XGBoost': XGBRegressor()

}

# Initialize lists to store scores

results = []

# Evaluate each model

for i in range(40, 50):

xtrain, xtest, ytrain, ytest = train\_test\_split(X, Y, test\_size=0.2, random\_state=i)

for name, model in models.items():

model.fit(xtrain, ytrain)

train\_acc = model.score(xtrain, ytrain) # Training score

test\_acc = model.score(xtest, ytest) # Testing score

cvs = cross\_val\_score(model, X, Y, cv=5).mean() # Cross-validation score

results.append({

'Model': name,

'Train Acc': train\_acc,

'Test Acc': test\_acc,

'CVS': cvs

})

# Create a DataFrame for results

results\_df = pd.DataFrame(results)

# Display the aggregated results

agg\_results = results\_df.groupby('Model').agg(['mean', 'std']).reset\_index()

agg\_results.columns = ['Model', 'Train Acc Mean', 'Train Acc Std', 'Test Acc Mean', 'Test Acc Std', 'CVS Mean', 'CVS Std']

print(agg\_results)

### **Key Changes:**

* **Print Statements**: Added to display the DataFrame's shape and contents before processing.
* **Empty Data Check**: A conditional check to avoid executing the model evaluation if the DataFrame is empty.

Run the modified code and observe the output of the print statements to diagnose the issue. If the DataFrame is indeed empty or if it doesn't contain the necessary columns, you'll need to revisit the data loading and preprocessing steps. Let me know if you need further assistance!

4o mini

##### **You said:**

Screenshot-2023-07-29-102756 Scores for various random\_state number After dividing the data into training and validation data it is considered a better practice to achieve stable and fast training of the model.We have identified the best random\_state number for this data set as 42 .Now we fix this random\_state and try with different ml algorithms for better score or accuracy. Now let’s train some state-of-the-art machine learning models on the training data and then use the validation data for choosing the best out of them for prediction. xtrain,xtest,ytrain,ytest=train\_test\_split(X,Y,test\_size=0.2,random\_state=42)

###### **ChatGPT said:**

ChatGPT

Memory updated

It looks like you're ready to proceed with training various machine learning models using the fixed random\_state of 42. Here's how you can set up the training process with different algorithms:

python

Copy code

import pandas as pd

from sklearn.linear\_model import LinearRegression, Lasso

from sklearn.svm import SVR

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

from xgboost import XGBRegressor

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

# Assuming df is already defined and preprocessed

X = df.drop(['charges'], axis=1)

Y = df['charges']

# Split the data

xtrain, xtest, ytrain, ytest = train\_test\_split(X, Y, test\_size=0.2, random\_state=42)

# Initialize models

models = {

'Linear Regression': LinearRegression(),

'Lasso Regression': Lasso(),

'Support Vector Regression': SVR(),

'Random Forest': RandomForestRegressor(),

'Gradient Boosting': GradientBoostingRegressor(),

'XGBoost': XGBRegressor()

}

# Evaluate each model

results = []

for name, model in models.items():

model.fit(xtrain, ytrain)

preds = model.predict(xtest)

mae = mean\_absolute\_error(ytest, preds)

mse = mean\_squared\_error(ytest, preds)

rmse = np.sqrt(mse)

results.append({

'Model': name,

'MAE': mae,

'MSE': mse,

'RMSE': rmse

})

# Create a DataFrame for results

results\_df = pd.DataFrame(results)

# Display the results

print(results\_df)

This code snippet evaluates various models using the fixed random\_state of 42 and calculates the mean absolute error (MAE), mean squared error (MSE), and root mean squared error (RMSE) for each model on the test set. You can then choose the model with the best performance based on these metrics.

**Training Score**: This is the R2R^2R2 score for the training data. A value closer to 1 means that the model explains a high proportion of the variance in the training data.

**Testing Score**: This is the R2R^2R2 score for the test data. It gives an indication of how well the model generalizes to unseen data.

**Cross-Validation Score**: This is the average R2R^2R2 score from 5-fold cross-validation, providing a more robust estimate of the model’s performance.

The output indicates how well the LinearRegression model performed on the given dataset, with the data split using a random\_state of 42.

Here's a breakdown of the results:

* **Training Score**: 0.7295 – This shows that the model explains about 72.95% of the variance in the training data. This indicates the model fits the training data reasonably well.
* **Testing Score**: 0.8062 – The model explains about 80.62% of the variance in the test data. A higher score on the test set compared to the training set suggests the model generalizes well to unseen data.
* **Cross-Validation Score**: 0.7471 – The average score from 5-fold cross-validation suggests that, on average, the model explains 74.71% of the variance in different subsets of the data, providing a more robust estimate of the model’s performance.

The results indicate that the model has a decent fit and generalizes well to the validation set. Would you like to proceed with other models to compare performance?

xtrain,xtest,ytrain,ytest=train\_test\_split(X,Y,test\_size=0.2,random\_state=42) lrmodel=LinearRegression() lrmodel.fit(xtrain,ytrain) print(lrmodel.score(xtrain,ytrain)) print(lrmodel.score(xtest,ytest)) print(cross\_val\_score(lrmodel,X,Y,cv=5,).mean()) Output: Linear Regression: 0.7295415541376445 0.8062391115570589 0.7470697972809902

###### **ChatGPT said:**

ChatGPT

The output shows the performance of the LinearRegression model as follows:

* **Training Score**: 0.7295 – The model explains 72.95% of the variance in the training data, indicating a reasonably good fit.
* **Testing Score**: 0.8062 – The model explains 80.62% of the variance in the test data, which shows it generalizes well to unseen data.
* **Cross-Validation Score**: 0.7471 – The average score from 5-fold cross-validation suggests that the model can explain 74.71% of the variance across multiple data splits, providing a more reliable estimate of performance.

The performance metrics suggest that this linear model fits the data well and performs consistently on unseen data. Would you like to proceed with testing other models or continue with further analysis on this model?

4o

##### **You said:**

from sklearn.metrics import r2\_score svrmodel=SVR() svrmodel.fit(xtrain,ytrain) ypredtrain1=svrmodel.predict(xtrain) ypredtest1=svrmodel.predict(xtest) print(r2\_score(ytrain,ypredtrain1)) print(r2\_score(ytest,ypredtest1)) print(cross\_val\_score(svrmodel,X,Y,cv=5,).mean())

###### **ChatGPT said:**

ChatGPT

The SVR model is being evaluated using the R² score on both the training and test data, as well as cross-validation. Here's a breakdown of what the code does:

* **Train the SVR model** on the training data.
* **Predict values** for both the training (xtrain) and test (xtest) sets.
* **Calculate the R² score** for both sets using r2\_score. The R² score tells how well the model is performing in explaining the variance of the dependent variable.
* **Perform cross-validation** on the whole dataset to get a more robust evaluation of the model.

Here’s what the output of the code might look like once you run it:

python

Copy code

from sklearn.metrics import r2\_score

svrmodel = SVR()

svrmodel.fit(xtrain, ytrain)

ypredtrain1 = svrmodel.predict(xtrain)

ypredtest1 = svrmodel.predict(xtest)

# R² Scores

print(r2\_score(ytrain, ypredtrain1)) # Training R² score

print(r2\_score(ytest, ypredtest1)) # Testing R² score

# Cross-validation score

print(cross\_val\_score(svrmodel, X, Y, cv=5).mean())

Expected output (hypothetical):

yaml

Copy code

Training R² score: 0.55 # (Example: May vary based on actual dataset)

Testing R² score: 0.50 # (Example: May vary based on actual dataset)

Cross-Validation score: 0.48 # (Example: May vary based on actual dataset)

Let me know if you run the code and encounter any specific results!

4o

##### **You said:**

from sklearn.metrics import r2\_score svrmodel=SVR() svrmodel.fit(xtrain,ytrain) ypredtrain1=svrmodel.predict(xtrain) ypredtest1=svrmodel.predict(xtest) print(r2\_score(ytrain,ypredtrain1)) print(r2\_score(ytest,ypredtest1)) print(cross\_val\_score(svrmodel,X,Y,cv=5,).mean()) Output: SVR: -0.10151474302536445 -0.1344454720199666 -0.10374591327267262

###### **ChatGPT said:**

ChatGPT

It looks like the **SVR model** is not performing well on the given dataset. Let's break down the results:

* **Training R² score (-0.101)**: A negative R² value indicates that the model is performing worse than a horizontal line (i.e., the mean prediction) on the training data. This suggests that the SVR model is underfitting the training data.
* **Testing R² score (-0.134)**: The negative R² score on the test data confirms that the model is also performing poorly on unseen data. It fails to generalize.
* **Cross-Validation score (-0.104)**: The mean cross-validation score is also negative, indicating that the model consistently underperforms across different subsets of the dataset.

Let's break down what you're doing in the code and what you should expect in terms of results for each step:

### **Step 1: Initial RandomForestRegressor Training**

python

Copy code

rfmodel = RandomForestRegressor(random\_state=42)

rfmodel.fit(xtrain, ytrain)

ypredtrain2 = rfmodel.predict(xtrain)

ypredtest2 = rfmodel.predict(xtest)

# R² Scores

print(r2\_score(ytrain, ypredtrain2)) # Training R² Score

print(r2\_score(ytest, ypredtest2)) # Testing R² Score

# Cross-validation score

print(cross\_val\_score(rfmodel, X, Y, cv=5).mean())

* **Training R² Score**: This will likely be very close to 1 because Random Forests typically overfit the training data.
* **Testing R² Score**: This shows how well the model generalizes. If it’s significantly lower than the training score, there may be overfitting.
* **Cross-Validation Score**: The mean score across 5 different folds, giving you a better sense of the model's generalization performance.

### **Step 2: Hyperparameter Tuning with Grid Search**

python

Copy code

from sklearn.model\_selection import GridSearchCV

# Defining parameter grid for tuning n\_estimators

param\_grid = {'n\_estimators': [10, 40, 50, 98, 100, 120, 150]}

estimator = RandomForestRegressor(random\_state=42)

# Grid Search to find the best n\_estimators

grid = GridSearchCV(estimator, param\_grid, scoring="r2", cv=5)

grid.fit(xtrain, ytrain)

print(grid.best\_params\_) # Output the best value for n\_estimators

* **Grid Search**: This step performs cross-validation for each value in param\_grid and chooses the best n\_estimators based on R² score.

### **Step 3: Refitting the RandomForestRegressor with Optimal Parameters**

python

Copy code

rfmodel = RandomForestRegressor(random\_state=42, n\_estimators=120)

rfmodel.fit(xtrain, ytrain)

ypredtrain2 = rfmodel.predict(xtrain)

ypredtest2 = rfmodel.predict(xtest)

# Evaluating the new model's performance

print(r2\_score(ytrain, ypredtrain2)) # Training R² Score

print(r2\_score(ytest, ypredtest2)) # Testing R² Score

print(cross\_val\_score(rfmodel, X, Y, cv=5).mean()) # Cross-validation score

* **New Training and Testing Scores**: After tuning n\_estimators, you should expect to see an improvement in the test R² score and the cross-validation score. The training score may slightly decrease, but this is often a good sign, as it means the model is not overfitting as much.

### **Expected Output:**

* **Training R²**: Will likely be very high, near 1.0 for both models (before and after tuning). This is expected for Random Forest because it often overfits the training data.
* **Test R²**: After tuning, you should expect the test R² to increase, showing that the model generalizes better.
* **Cross-Validation Score**: This should also improve after hyperparameter tuning, which reflects a more stable model performance across different folds.

Let me know how your results compare!

Your approach here is training a **GradientBoostingRegressor** model, followed by hyperparameter tuning with GridSearchCV to find the best values for n\_estimators and learning\_rate. Let's break down your code step by step to understand what’s happening.

### **Step 1: Initial Training of GradientBoostingRegressor**

python

Copy code

gbmodel = GradientBoostingRegressor()

gbmodel.fit(xtrain, ytrain)

ypredtrain3 = gbmodel.predict(xtrain)

ypredtest3 = gbmodel.predict(xtest)

# Evaluate the model

print(r2\_score(ytrain, ypredtrain3)) # Training R² Score

print(r2\_score(ytest, ypredtest3)) # Testing R² Score

print(cross\_val\_score(gbmodel, X, Y, cv=5).mean()) # Cross-validation score

* **Training R² Score**: Since Gradient Boosting builds trees sequentially to reduce the residual errors, this score should be fairly high, but not as extreme as Random Forest.
* **Testing R² Score**: This reflects how well the model generalizes to unseen data. A value close to the training score means good generalization.
* **Cross-Validation Score**: This score is computed using 5-fold cross-validation, providing a robust evaluation of the model.

### **Step 2: Hyperparameter Tuning using Grid Search**

python

Copy code

from sklearn.model\_selection import GridSearchCV

# Define the parameter grid for n\_estimators and learning\_rate

param\_grid = {

'n\_estimators': [10, 15, 19, 20, 21, 50],

'learning\_rate': [0.1, 0.19, 0.2, 0.21, 0.8, 1]

}

estimator = GradientBoostingRegressor()

# Perform grid search

grid = GridSearchCV(estimator, param\_grid, scoring="r2", cv=5)

grid.fit(xtrain, ytrain)

# Output the best parameters

print(grid.best\_params\_)

* **Grid Search**: This will train the model using different combinations of n\_estimators and learning\_rate, choosing the pair that maximizes the R² score.
* **Best Parameters**: After training, the best n\_estimators and learning\_rate values will be printed. For your code, you expect n\_estimators=19 and learning\_rate=0.2 to be the best combination.

### **Step 3: Refitting with Optimal Parameters**

python

Copy code

# Refit the model with the best parameters from Grid Search

gbmodel = GradientBoostingRegressor(n\_estimators=19, learning\_rate=0.2)

gbmodel.fit(xtrain, ytrain)

ypredtrain3 = gbmodel.predict(xtrain)

ypredtest3 = gbmodel.predict(xtest)

# Evaluate the model again with optimized parameters

print(r2\_score(ytrain, ypredtrain3)) # Training R² Score

print(r2\_score(ytest, ypredtest3)) # Testing R² Score

print(cross\_val\_score(gbmodel, X, Y, cv=5).mean()) # Cross-validation score

* **New Training and Testing R² Scores**: After optimizing with n\_estimators=19 and learning\_rate=0.2, you should see some improvements in both the training and testing R² scores. The training score may not change significantly, but the testing score should ideally improve, indicating better generalization.
* **Cross-Validation Score**: This score should also improve after hyperparameter tuning.

### **Expected Output:**

1. **Initial Training:**
   * Training R²: Should be high (close to 1).
   * Testing R²: Should be slightly lower, but reasonably close to the training R² score.
   * Cross-Validation: Should be a robust estimate of model performance on unseen data.
2. **Grid Search:**
   * The best parameters for n\_estimators and learning\_rate should be printed.
3. **After Hyperparameter Tuning:**
   * You should see an improvement in both the testing R² score and the cross-validation score, reflecting better generalization and overall performance.

Let me know how these steps work out for you!

You're now training and tuning an **XGBoost Regressor**, which is a powerful algorithm for regression tasks. Here's what your code is doing step by step, along with the expected output and results:

### **Step 1: Initial Training of XGBoost Regressor**

python

Copy code

xgmodel = XGBRegressor()

xgmodel.fit(xtrain, ytrain)

ypredtrain4 = xgmodel.predict(xtrain)

ypredtest4 = xgmodel.predict(xtest)

# Evaluate the model

print(r2\_score(ytrain, ypredtrain4)) # Training R² Score

print(r2\_score(ytest, ypredtest4)) # Testing R² Score

print(cross\_val\_score(xgmodel, X, Y, cv=5).mean()) # Cross-validation score

* **Training R² Score**: XGBoost usually fits the training data very well, so expect a high R² score close to 1.
* **Testing R² Score**: This is the key metric to watch for. If it's close to the training R² score, it means the model is generalizing well to unseen data.
* **Cross-Validation Score**: This score gives you a robust measure of how the model performs across different data splits.

### **Step 2: Hyperparameter Tuning with GridSearchCV**

python

Copy code

from sklearn.model\_selection import GridSearchCV

# Define the parameter grid for n\_estimators, max\_depth, and gamma

param\_grid = {

'n\_estimators': [10, 15, 20, 40, 50],

'max\_depth': [3, 4, 5],

'gamma': [0, 0.15, 0.3, 0.5, 1]

}

estimator = XGBRegressor()

# Perform grid search

grid = GridSearchCV(estimator, param\_grid, scoring="r2", cv=5)

grid.fit(xtrain, ytrain)

# Output the best parameters

print(grid.best\_params\_)

* **Grid Search**: This will search for the best combination of n\_estimators, max\_depth, and gamma that maximizes the R² score.
* **Best Parameters**: The best combination of hyperparameters will be printed. In your case, it expects n\_estimators=15, max\_depth=3, and gamma=0.

### **Step 3: Refitting the XGBoost Regressor with Best Parameters**

python

Copy code

# Refit the model with the best parameters from Grid Search

xgmodel = XGBRegressor(n\_estimators=15, max\_depth=3, gamma=0)

xgmodel.fit(xtrain, ytrain)

ypredtrain4 = xgmodel.predict(xtrain)

ypredtest4 = xgmodel.predict(xtest)

# Evaluate the model again with optimized parameters

print(r2\_score(ytrain, ypredtrain4)) # Training R² Score

print(r2\_score(ytest, ypredtest4)) # Testing R² Score

print(cross\_val\_score(xgmodel, X, Y, cv=5).mean()) # Cross-validation score

* **New Training and Testing R² Scores**: After applying the optimal parameters (n\_estimators=15, max\_depth=3, gamma=0), you should expect to see better performance on the testing set and more balanced generalization.
* **Cross-Validation Score**: The cross-validation score should improve after tuning the hyperparameters, indicating better performance across multiple folds.

### **Expected Output:**

1. **Initial Training:**
   * Training R²: Should be high (close to 1).
   * Testing R²: May be a bit lower, but ideally not too far off from the training score.
   * Cross-Validation: Should give a robust measure of how well the model is doing across different folds.
2. **Grid Search:**
   * The best parameters for n\_estimators, max\_depth, and gamma should be printed.
3. **After Hyperparameter Tuning:**
   * You should see an improvement in the testing R² score and cross-validation score, indicating better generalization and overall performance.

Let me know how these results turn out for you!

The code snippet you provided is used to save your trained XGBoost model to a file using Python's pickle module. Here’s a breakdown of what it does and how you can load the model later:

### **Code Breakdown**

**Importing dump from pickle**:  
python  
Copy code  
from pickle import dump

* + This imports the dump function, which is used to serialize and save Python objects to a file.

**Saving the Model**:  
python  
Copy code  
dump(finalmodel, open('insurancemodelf.pkl', 'wb'))

* + This line saves the finalmodel object to a file named insurancemodelf.pkl.
  + The mode 'wb' means "write binary," which is necessary for saving non-text data like models.

### **Loading the Model Later**

To load the saved model back into your Python environment, you can use the following code:

python

Copy code

from pickle import load

# Load the model from the file

with open('insurancemodelf.pkl', 'rb') as file:

loaded\_model = load(file)

# Now you can use loaded\_model to make predictions

### **Example Usage**

After loading the model, you can use it just like you would with the original finalmodel:

python

Copy code

# Example predictions

predictions = loaded\_model.predict(xtest) # Assuming xtest is defined

print(predictions)

### **Summary**

Using pickle for model serialization is a common practice in Python, allowing you to save trained models for later use without needing to retrain them. If you need any further assistance or clarification, feel free to ask!

The NameError you're encountering indicates that the variable important\_features has not been defined in your current environment or session. This can happen if the previous code that created important\_features was not run or if it encountered an error.

### **Steps to Resolve the Issue**

**Re-run the Feature Importance Calculation**: Ensure that you have the DataFrame feats defined and then filter it to create important\_features:  
python  
Copy code  
important\_features = feats[feats['Importance'] > 0.01]

**Check the Output**: After creating important\_features, print it to confirm that it contains the expected data:  
python  
Copy code  
print(important\_features)

**Plot the Important Features**: Once you confirm that important\_features is defined, you can run the plotting code:  
python  
Copy code  
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))

plt.barh(important\_features.index, important\_features['Importance'])

plt.xlabel('Importance')

plt.title('Important Features')

plt.show()

### **Complete Code Example**

Here's how you can combine everything into a single block of code:

python

Copy code

# Assuming feats DataFrame is already defined

important\_features = feats[feats['Importance'] > 0.01]

# Print important features

print(important\_features)

# Plotting important features

import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))

plt.barh(important\_features.index, important\_features['Importance'])

plt.xlabel('Importance')

plt.title('Important Features')

plt.show()

### **Troubleshooting Tips**

* If you still get an error about important\_features, double-check the previous steps to ensure they executed correctly.
* Ensure that feats was created successfully and contains valid data before filtering.

If you have any more questions or encounter further issues, feel free to ask!

4o mini

##### **You said:**

df.drop(df[['sex','region']],axis=1,inplace=True) Xf=df.drop(df[['charges']],axis=1) X=df.drop(df[['charges']],axis=1) xtrain,xtest,ytrain,ytest=train\_test\_split(Xf,Y,test\_size=0.2,random\_state=42) finalmodel=XGBRegressor(n\_estimators=15,max\_depth=3,gamma=0) finalmodel.fit(xtrain,ytrain) ypredtrain4=finalmodel.predict(xtrain) ypredtest4=finalmodel.predict(xtest) print(r2\_score(ytrain,ypredtrain4)) print(r2\_score(ytest,ypredtest4)) print(cross\_val\_score(finalmodel,X,Y,cv=5,).mean())

###### **ChatGPT said:**

ChatGPT

It looks like you're trying to build and evaluate an XGBoost regression model using your dataset. Here's a breakdown of the code you provided along with some suggestions and improvements.

### **Code Breakdown**

**Dropping Columns**:  
python  
Copy code  
df.drop(df[['sex','region']], axis=1, inplace=True)

1. This line removes the 'sex' and 'region' columns from the DataFrame df in place.

**Defining Features and Target**:  
python  
Copy code  
Xf = df.drop(df[['charges']], axis=1)

X = df.drop(df[['charges']], axis=1)

1. Here, you're defining Xf and X as the features by dropping the 'charges' column, which is presumably your target variable.

**Splitting Data**:  
python  
Copy code  
xtrain, xtest, ytrain, ytest = train\_test\_split(Xf, Y, test\_size=0.2, random\_state=42)

1. This splits your features and target into training and testing sets with 20% of the data reserved for testing.

**Training the Model**:  
python  
Copy code  
finalmodel = XGBRegressor(n\_estimators=15, max\_depth=3, gamma=0)

finalmodel.fit(xtrain, ytrain)

1. An XGBoost regressor is created and fitted to the training data.

**Making Predictions**:  
python  
Copy code  
ypredtrain4 = finalmodel.predict(xtrain)

ypredtest4 = finalmodel.predict(xtest)

**Evaluating the Model**:  
python  
Copy code  
print(r2\_score(ytrain, ypredtrain4))

print(r2\_score(ytest, ypredtest4))

print(cross\_val\_score(finalmodel, X, Y, cv=5).mean())

### **Improvements and Suggestions**

1. **Ensure Y is Defined**: Make sure that Y (the target variable) is defined before this code runs. If it's not, you'll get a NameError.

**Cross-Validation on Training Data**: It's better to perform cross-validation on the training data instead of the full dataset:  
python  
Copy code  
print(cross\_val\_score(finalmodel, xtrain, ytrain, cv=5).mean())

1. **Hyperparameter Tuning**: You might consider tuning the hyperparameters of your model for better performance.
2. **Feature Importance**: After training, you can also look into feature importance to understand which features are most influential in your model's predictions.

### **Example Code After Improvements**

Here's how you can improve your code:

python

Copy code

# Assuming Y is defined as the target variable

df.drop(['sex', 'region'], axis=1, inplace=True)

# Defining features and target

Xf = df.drop(['charges'], axis=1)

Y = df['charges'] # Ensure Y is defined correctly

# Splitting the data

xtrain, xtest, ytrain, ytest = train\_test\_split(Xf, Y, test\_size=0.2, random\_state=42)

# Training the model

finalmodel = XGBRegressor(n\_estimators=15, max\_depth=3, gamma=0)

finalmodel.fit(xtrain, ytrain)

# Making predictions

ypredtrain4 = finalmodel.predict(xtrain)

ypredtest4 = finalmodel.predict(xtest)

# Evaluating the model

print("R^2 Score on training data:", r2\_score(ytrain, ypredtrain4))

print("R^2 Score on testing data:", r2\_score(ytest, ypredtest4))

print("Cross-validated R^2 Score:", cross\_val\_score(finalmodel, xtrain, ytrain, cv=5).mean())

Make sure to run this code in a context where all the necessary libraries are imported and the data is correctly loaded. If you have any further questions or run into issues, feel free to ask!

4o mini

##### **You said:**

KeyError Traceback (most recent call last) Cell In[116], line 2 1 # Assuming Y is defined as the target variable ----> 2 df.drop(['sex', 'region'], axis=1, inplace=True) 4 # Defining features and target 5 Xf = df.drop(['charges'], axis=1) File /lib/python3.12/site-packages/pandas/core/frame.py:5568, in DataFrame.drop(self, labels, axis, index, columns, level, inplace, errors) 5420 def drop( 5421 self, 5422 labels: IndexLabel | None = None, (...) 5429 errors: IgnoreRaise = "raise", 5430 ) -> DataFrame | None: 5431 """ 5432 Drop specified labels from rows or columns. 5433 (...) 5566 weight 1.0 0.8 5567 """ -> 5568 return super().drop( 5569 labels=labels, 5570 axis=axis, 5571 index=index, 5572 columns=columns, 5573 level=level, 5574 inplace=inplace, 5575 errors=errors, 5576 ) File /lib/python3.12/site-packages/pandas/core/generic.py:4782, in NDFrame.drop(self, labels, axis, index, columns, level, inplace, errors) 4780 for axis, labels in axes.items(): 4781 if labels is not None: -> 4782 obj = obj.\_drop\_axis(labels, axis, level=level, errors=errors) 4784 if inplace: 4785 self.\_update\_inplace(obj) File /lib/python3.12/site-packages/pandas/core/generic.py:4824, in NDFrame.\_drop\_axis(self, labels, axis, level, errors, only\_slice) 4822 new\_axis = axis.drop(labels, level=level, errors=errors) 4823 else: -> 4824 new\_axis = axis.drop(labels, errors=errors) 4825 indexer = axis.get\_indexer(new\_axis) 4827 # Case for non-unique axis 4828 else: File /lib/python3.12/site-packages/pandas/core/indexes/base.py:7069, in Index.drop(self, labels, errors) 7067 if mask.any(): 7068 if errors != "ignore": -> 7069 raise KeyError(f"{labels[mask].tolist()} not found in axis") 7070 indexer = indexer[~mask] 7071 return self.delete(indexer) KeyError: "['sex', 'region'] not found in axis"

###### **ChatGPT said:**

ChatGPT

The KeyError you're encountering indicates that the columns 'sex' and 'region' do not exist in the DataFrame df at the time you're trying to drop them. This can happen for several reasons, such as the columns being removed in a previous operation or being incorrectly named.

### **Steps to Troubleshoot**

**Check Existing Columns**: First, you can check what columns are currently in your DataFrame by using:  
python  
Copy code  
print(df.columns)

1. **Confirm Column Names**: Ensure that the column names you want to drop match exactly with the existing column names, including case sensitivity and any extra spaces.

**Handle Missing Columns Gracefully**: You can use the errors='ignore' parameter in the drop() method to avoid raising an error if the columns are not found:  
python  
Copy code  
df.drop(['sex', 'region'], axis=1, inplace=True, errors='ignore')

### **Revised Code**

Here’s how you can implement these steps in your code:

python

Copy code

# Check existing columns

print("Current DataFrame columns:", df.columns)

# Drop the specified columns safely

df.drop(['sex', 'region'], axis=1, inplace=True, errors='ignore')

# Defining features and target

Xf = df.drop(['charges'], axis=1)

Y = df['charges'] # Ensure Y is defined correctly

# Splitting the data

xtrain, xtest, ytrain, ytest = train\_test\_split(Xf, Y, test\_size=0.2, random\_state=42)

# Training the model

finalmodel = XGBRegressor(n\_estimators=15, max\_depth=3, gamma=0)

finalmodel.fit(xtrain, ytrain)

# Making predictions

ypredtrain4 = finalmodel.predict(xtrain)

ypredtest4 = finalmodel.predict(xtest)

# Evaluating the model

print("R^2 Score on training data:", r2\_score(ytrain, ypredtrain4))

print("R^2 Score on testing data:", r2\_score(ytest, ypredtest4))

print("Cross-validated R^2 Score:", cross\_val\_score(finalmodel, xtrain, ytrain, cv=5).mean())

### **Additional Suggestions**

* **Data Cleaning**: If you're performing operations on your DataFrame, ensure that your data cleaning steps (like dropping columns) are executed in the correct order.
* **Data Inspection**: It can be helpful to inspect your DataFrame at various points in your code using .head() or .info() to understand its structure and content.

If you continue to experience issues, feel free to share the output of print(df.columns) for further assistance!

4o mini

##### **You said:**

Final Model: Train accuracy : 0.870691899927822 Test accuracy : 0.904151903449132 CV Score : 0.8600710679082143

###### **ChatGPT said:**

ChatGPT

It looks like your final model using XGBoost has achieved strong performance metrics:

* **Train Accuracy**: 0.8707 (approximately 87.07%)
* **Test Accuracy**: 0.9042 (approximately 90.42%)
* **Cross-Validated Score**: 0.8601 (approximately 86.01%)

### **Interpretation of Results**

1. **Training Accuracy**: The model performs well on the training data, indicating it has learned the patterns present in that dataset. However, a significant difference between training and testing accuracy can sometimes indicate overfitting.
2. **Testing Accuracy**: The high test accuracy suggests that the model generalizes well to unseen data, which is a positive sign of its predictive capability.
3. **Cross-Validated Score**: A CV score close to the test accuracy further indicates that the model is robust and not just memorizing the training data.

### **Next Steps**

1. **Feature Importance**: Consider evaluating the importance of different features to understand which ones contribute most to the model's predictions. This could help in feature selection and improving interpretability.
2. **Hyperparameter Tuning**: If you haven't already, you might want to explore further tuning of hyperparameters (e.g., increasing the number of estimators, adjusting learning rates, etc.) to optimize performance.
3. **Model Evaluation**: Use other metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), or R² Score to evaluate model performance further.
4. **Visualizations**: Plotting residuals, learning curves, or feature importances can provide additional insights into the model's performance.
5. **Model Comparison**: If you have implemented other models, comparing their performance can help identify the best approach for your problem.

If you have any specific questions or need assistance with the next steps, feel free to ask!

4o mini

##### **You said:**

Final Model: Train accuracy : 0.870691899927822 Test accuracy : 0.904151903449132 CV Score : 0.8600710679082143

###### **ChatGPT said:**

ChatGPT

Memory updated

4o mini

##### **You said:**

df.drop(df[['sex','region']],axis=1,inplace=True) Xf=df.drop(df[['charges']],axis=1) X=df.drop(df[['charges']],axis=1) xtrain,xtest,ytrain,ytest=train\_test\_split(Xf,Y,test\_size=0.2,random\_state=42) finalmodel=XGBRegressor(n\_estimators=15,max\_depth=3,gamma=0) finalmodel.fit(xtrain,ytrain) ypredtrain4=finalmodel.predict(xtrain) ypredtest4=finalmodel.predict(xtest) print(r2\_score(ytrain,ypredtrain4)) print(r2\_score(ytest,ypredtest4)) print(cross\_val\_score(finalmodel,X,Y,cv=5,).mean())

###### **ChatGPT said:**

ChatGPT

Your final model has achieved the following performance metrics:

* **Train Accuracy**: 0.8707
* **Test Accuracy**: 0.9042
* **Cross-Validation Score**: 0.8601

These results indicate that your model performs well both on the training and test datasets, suggesting a good balance and generalization. If you have further questions or need assistance with the next steps, feel free to ask!

4o mini

##### **You said:**

I want the code to make output for the same result

###### **ChatGPT said:**

ChatGPT

Sure! Here’s the code to fit your final XGBoost model and obtain the specified performance metrics:

python

Copy code

import pandas as pd

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.metrics import r2\_score

from xgboost import XGBRegressor

# Assuming df is your DataFrame and 'charges' is your target variable

# Drop 'sex' and 'region' columns

df.drop(['sex', 'region'], axis=1, inplace=True)

# Define features and target variable

Xf = df.drop(['charges'], axis=1) # Features without target

Y = df['charges'] # Target variable

# Split the data into training and test sets

xtrain, xtest, ytrain, ytest = train\_test\_split(Xf, Y, test\_size=0.2, random\_state=42)

# Initialize the XGBoost model

finalmodel = XGBRegressor(n\_estimators=15, max\_depth=3, gamma=0)

# Fit the model on training data

finalmodel.fit(xtrain, ytrain)

# Predict on training and test data

ypredtrain = finalmodel.predict(xtrain)

ypredtest = finalmodel.predict(xtest)

# Calculate R² scores for training and test predictions

train\_r2 = r2\_score(ytrain, ypredtrain)

test\_r2 = r2\_score(ytest, ypredtest)

# Perform cross-validation and calculate the mean score

cv\_score = cross\_val\_score(finalmodel, Xf, Y, cv=5).mean()

# Print the results

print(f"Train accuracy : {train\_r2:.10f}")

print(f"Test accuracy : {test\_r2:.10f}")

print(f"CV Score : {cv\_score:.10f}")

This code snippet will output the train accuracy, test accuracy, and cross-validation score formatted to ten decimal places, similar to the results you've provided earlier. Adjust any DataFrame names or features as necessary based on your dataset.

4o mini

##### **You said:**

# Assuming df is your DataFrame and 'charges' is your target variable # Drop 'sex' and 'region' columns df.drop(['sex', 'region'], axis=1, inplace=True)--------------------------------------------------------------------------- KeyError Traceback (most recent call last) Cell In[135], line 3 1 # Assuming df is your DataFrame and 'charges' is your target variable 2 # Drop 'sex' and 'region' columns ----> 3 df.drop(['sex', 'region'], axis=1, inplace=True) File /lib/python3.12/site-packages/pandas/core/frame.py:5568, in DataFrame.drop(self, labels, axis, index, columns, level, inplace, errors) 5420 def drop( 5421 self, 5422 labels: IndexLabel | None = None, (...) 5429 errors: IgnoreRaise = "raise", 5430 ) -> DataFrame | None: 5431 """ 5432 Drop specified labels from rows or columns. 5433 (...) 5566 weight 1.0 0.8 5567 """ -> 5568 return super().drop( 5569 labels=labels, 5570 axis=axis, 5571 index=index, 5572 columns=columns, 5573 level=level, 5574 inplace=inplace, 5575 errors=errors, 5576 ) File /lib/python3.12/site-packages/pandas/core/generic.py:4782, in NDFrame.drop(self, labels, axis, index, columns, level, inplace, errors) 4780 for axis, labels in axes.items(): 4781 if labels is not None: -> 4782 obj = obj.\_drop\_axis(labels, axis, level=level, errors=errors) 4784 if inplace: 4785 self.\_update\_inplace(obj) File /lib/python3.12/site-packages/pandas/core/generic.py:4824, in NDFrame.\_drop\_axis(self, labels, axis, level, errors, only\_slice) 4822 new\_axis = axis.drop(labels, level=level, errors=errors) 4823 else: -> 4824 new\_axis = axis.drop(labels, errors=errors) 4825 indexer = axis.get\_indexer(new\_axis) 4827 # Case for non-unique axis 4828 else: File /lib/python3.12/site-packages/pandas/core/indexes/base.py:7069, in Index.drop(self, labels, errors) 7067 if mask.any(): 7068 if errors != "ignore": -> 7069 raise KeyError(f"{labels[mask].tolist()} not found in axis") 7070 indexer = indexer[~mask] 7071 return self.delete(indexer) KeyError: "['sex', 'region'] not found in axis"

###### **ChatGPT said:**

ChatGPT

The KeyError indicates that the columns 'sex' and 'region' are not present in your DataFrame df. You can check the columns in your DataFrame using df.columns to confirm their names. Here's how to proceed:

1. **Check Current Columns**: First, let's print the columns in your DataFrame to see what is available.

python

Copy code

print(df.columns)

1. **Drop Existing Columns**: Modify the drop command based on the columns that are actually in your DataFrame. If you find that the names are different or if those columns don't exist, you can adjust your code accordingly. Here's how to safely drop columns if they exist:

python

Copy code

# Drop 'sex' and 'region' columns only if they are present

columns\_to\_drop = ['sex', 'region']

df.drop(columns=[col for col in columns\_to\_drop if col in df.columns], axis=1, inplace=True)

1. **Continue with Your Analysis**: Once you've successfully dropped the columns, you can proceed with defining your features and target variable as before:

python

Copy code

# Define features and target variable

Xf = df.drop(['charges'], axis=1) # Features without target

Y = df['charges'] # Target variable

# Split the data into training and test sets

xtrain, xtest, ytrain, ytest = train\_test\_split(Xf, Y, test\_size=0.2, random\_state=42)

# Initialize the XGBoost model

finalmodel = XGBRegressor(n\_estimators=15, max\_depth=3, gamma=0)

# Fit the model on training data

finalmodel.fit(xtrain, ytrain)

# Predict on training and test data

ypredtrain = finalmodel.predict(xtrain)

ypredtest = finalmodel.predict(xtest)

# Calculate R² scores for training and test predictions

train\_r2 = r2\_score(ytrain, ypredtrain)

test\_r2 = r2\_score(ytest, ypredtest)

# Perform cross-validation and calculate the mean score

cv\_score = cross\_val\_score(finalmodel, Xf, Y, cv=5).mean()

# Print the results

print(f"Train accuracy : {train\_r2:.10f}")

print(f"Test accuracy : {test\_r2:.10f}")

print(f"CV Score : {cv\_score:.10f}")

This approach will help avoid errors related to non-existent columns while allowing you to continue with your analysis smoothly.