University of Hertfordshire UH

School of Physics, Engineering and Computer Science

MSc Data Science Project

7PAM2002-0509-2023

Department of Physics, Astronomy and Mathematics

Data Science FINAL PROJECT REPORT

Project Title:

Medical Insurance Cost Prediction

Student Name and SRN:

Shubham Verma 22099668 (sv23abk@herts.ac.uk)

Supervisor: Dr Man Lai Tang

Date Submitted: 27th Aug 2024

Word Count: 7085

DECLARATION STATEMENT

This report is submitted in partial fulfilment of the requirement for the degree of Master of Science in Data Science at the University of Hertfordshire.

I have read the guidance to students on academic integrity, misconduct and plagiarism information at Assessment Offences and Academic Misconduct and understand the University process of dealing with suspected cases of academic misconduct and the possible penalties, which could include failing the project module or course.

I certify that the work submitted is my own and that any material derived or quoted from published or unpublished work of other persons has been duly acknowledged. (Ref. UPR AS/C/6.1, section 7 and UPR AS/C/5, section 3.6). I have not used ChatGPT, or any other generative AI tool, to write the report or code (other than where declared or referenced).

I did not use human participants or undertake a survey in my MSc Project.

I hereby give permission for the report to be made available on module websites provided the source is acknowledged.

Student Name printed: ***Shubham Verma***

Student Name signature: 

Student SRN number: ***22099668***

UNIVERSITY OF HERTFORDSHIRE

SCHOOL OF PHYSICS, ENGINEERING AND COMPUTER SCIENCE

Acknowledgments

I would like to express my deepest gratitude to Dr. Carolyn Devereux, my Program Leader, for her unwavering guidance and encouragement throughout this journey. Her insightful advice and dedication have been instrumental in shaping my academic path.

A special thank you to my Supervisor, Dr. Man Lai Tang, whose expertise and thoughtful feedback have been invaluable in the development of this project. Her support and constructive criticism pushed me to achieve a higher level of academic rigor.

I am also profoundly grateful to the University of Hertfordshire for providing an enriching learning environment and the resources necessary to pursue this research. The knowledge and experiences I have gained here will continue to guide me in my future endeavours.

Lastly, I would like to extend my heartfelt thanks to my parents, my sister, and my friends. Their constant love, support, and belief in me have been my greatest source of strength. This achievement would not have been possible without them.

Abstract

This MSc Data Science project focuses on developing accurate models to predict medical insurance costs, a crucial issue for both individuals and insurers. The study examines the impact of demographic, behavioural, and geographical factors, including age, sex, BMI, number of dependents, smoking status, and region, on insurance costs. Using linear regression, quantile regression, and XGBoost, the project assesses and compares the effectiveness of these models under various conditions, including feature selection and the removal of outliers. The analysis reveals the limitations of traditional regression methods and highlights the superior performance of XGBoost in capturing the complexity and variability of medical costs. The findings contribute to the development of more reliable predictive models, offering significant implications for risk management and pricing strategies in the insurance industry. This work provides a foundation for further research and practical applications in improving healthcare cost predictions.

Contents

1. Introduction

The rising cost of health care is a major problem worldwide for both the individual and the insurance provider. The cost of medical insurance needs to be predicted with high accuracy since it forms a basis for managing financial risk and ensuring the sustainability of health care systems. For individuals, unpredicted high medical costs often cause extreme financial stress, while for insurance companies, bad estimates of costs can disrupt their risk management strategies and lead to huge losses (Kaushik et al., 2022).

Therefore, the aim of this project is to come up with a reliable predictive model in estimating medical insurance costs according to diverse demographic, behavioural, and geographical variables such as age, sex, BMI, number of dependents, smoker or not, and region of residence. In fact, such variables influence the cost of insurance both in their direct capacity and in interaction with other factors in very complicated ways (Panda et al., 2022).

At present in the insurance industry, traditional linear regression models are commonly being used for the prediction of cost. This is because they are simple and interpretable. However, these models often fail to capture the nuances in the data, particularly the variability at different points in the cost distribution. However, this project goes a step further than those traditional methods by examining the effectiveness of more advanced techniques, such as quantile regression and XGBoost, in helping to model more accurately.

1.1 Research Questions, Aims, and Objectives

1.1.1 Research Questions

How can the hidden patterns and behaviours in the medical cost dataset be uncovered?

What are the effects of variable selection, interaction effects, and multicollinearity on the performance of regression models?
How does quantile regression compare to traditional regression techniques in predicting medical insurance costs?

1.1.2 Aims and Objectives:

Aim 1: To uncover hidden patterns within the medical cost data.
Objective 1.1: Perform exploratory data analysis to identify key trends and anomalies.
Objective 1.2: Apply clustering techniques to detect patterns that may influence cost predictions.

Aim 2: To analyse the impact of variable selection, interaction effects, and multicollinearity on regression models.
Objective 2.1: Employ feature selection methods to identify the most significant predictors.
Objective 2.2: Assess the impact of multicollinearity and interaction effects on model performance.

Aim 3: To compare the performance and applicability of quantile regression versus traditional regression methods.
Objective 3.1: Implement and evaluate linear regression, quantile regression, and XGBoost models.
Objective 3.2: Compare model performance across different data scenarios, including the handling of outliers and feature selection.

In summary, this project is positioned to provide a comprehensive evaluation of advanced modelling techniques in the context of medical insurance cost prediction. By addressing the limitations of traditional approaches and incorporating cutting-edge methods, this study aims to enhance predictive accuracy, offering valuable insights for the insurance industry in its ongoing efforts to manage healthcare-related financial risks.

2. Literature Review

Medical cost prediction probably stands in the forefront of all research work areas in healthcare compared with services, the complexity and costliness of which have been rising. Precise health cost prediction is of prime importance to insurance companies, healthcare providers, and policymakers for risk evaluation, efficient resource management, and design of appropriate pricing strategy. In that arena, the literature is huge, developing from basic linear models to very complicated machine learning algorithms only over the last decade. Below follow several novel research methods and approaches that, in time, have been used for medical insurance expenditure prediction: starting from linear regression to quantile regression and its recent developments under machine learning methodologies, including XGBoost. It then identifies the importance of feature selection and how high leverage points and outliers can be managed (Zou et al., 2023).

2.1 Background

Predicting medical costs is crucial for insurance companies, healthcare providers, and policymakers, as it informs risk assessment, pricing strategies, and resource allocation. Traditionally, Linear Regression has been widely used due to its simplicity and interpretability, modelling the relationship between medical costs and predictors like age, BMI, and smoking status. However, Linear Regression assumes a linear relationship and struggles with non-linear patterns, skewed distributions, and outliers commonly found in healthcare data.

To address these limitations, Quantile Regression has emerged as a more robust alternative. Unlike Linear Regression, which estimates the mean of the dependent variable, Quantile Regression provides insights across different points in the distribution, such as the median or other quantiles. This approach is particularly valuable in healthcare cost prediction, where the impact of factors like smoking or BMI can vary significantly between low-cost and high-cost patients. Quantile Regression better captures this variability, making it a more suitable method for analysing complex, skewed data (Staffa et al., 2019).

Recent advancements in machine learning, such as XGBoost, have further expanded the toolkit for medical cost prediction, offering powerful non-linear modelling capabilities. However, these methods often come with challenges related to interpretability and require careful tuning and validation (Tufail et al., 2023).

This literature review examines key studies that have applied and compared Linear Regression and Quantile Regression in predicting medical costs, providing a foundation for understanding the strengths and limitations of each approach in healthcare analytics.

2.2 Linear Regression

A statistical method to model the linear relationship between a dependent variable and one or more independent variables.

Newhouse, J. P. (1992). Medical Care Costs: How Much Welfare Loss?

Summary: Newhouse's (1992) work is a seminal paper that laid the foundation for using Linear Regression in medical cost prediction. The study demonstrated that demographic variables such as age, sex, and income were significant predictors of healthcare costs. Newhouse's application of Linear Regression provided a straightforward method to estimate medical expenditures, making it one of the earliest and most cited approaches in the literature. Despite the simplicity of the model, the paper highlighted how well linear relationships could capture the influence of demographic factors on medical costs, making it a benchmark for subsequent studies.

DeMaris, A. (2004). Regression with Social Data: Modeling Continuous and Limited Response Variables.

Summary: DeMaris (2004) provides a comprehensive overview of regression techniques, with a particular focus on their application in social sciences, including healthcare. This book critically examines the limitations of Linear Regression, particularly its assumption of linearity and its sensitivity to outliers. DeMaris emphasizes the importance of verifying model assumptions and the potential pitfalls when these assumptions do not hold, which is often the case with skewed medical

cost data. This work is crucial for understanding the limitations of Linear Regression in more complex datasets, underscoring the need for alternative models in medical cost prediction.

Dodd, S. (2006). A comparison of multivariable regression models to analyse cost data.

Summary: Dodd et al. (2006) conducted a comparative study of various regression models for predicting health costs, including Linear Regression. The study highlighted that while Linear Regression is useful for its simplicity and interpretability, it often fails in datasets characterized by high variance and non-linear relationships. The authors pointed out that Linear Regression tends to underestimate the cost predictions in cases with extreme values, leading to the exploration of more advanced models that can handle such complexities. This paper is significant as it provides empirical evidence of the limitations of Linear Regression in real-world healthcare data.

2.3 Quantile Regression

A statistical method to model the conditional quantiles of a dependent variable based on one or more independent variables.

Koenker, R., & Bassett, G. (1978). Regression Quantiles. Econometrica

Summary: Koenker and Bassett (1978) introduced Quantile Regression, a method that goes beyond the limitations of Linear Regression by modelling different points in the distribution of the dependent variable. This technique is particularly useful in the context of medical cost prediction, where costs are often skewed and include outliers. Quantile Regression allows for a more detailed analysis by estimating the effects of predictors not just on the mean but across the entire distribution of costs. This approach is critical for understanding the variability in medical costs, especially for high-cost individuals who significantly impact the healthcare system.

Buntin, M. B., & Zaslavsky, A. M. (2004). Too Much Ado about Two-Part Models and Transformation? Comparing Methods of Modeling Health Care Costs. Journal of Health Economics

Summary: Buntin and Zaslavsky (2004) applied Quantile Regression to medical cost data, emphasizing its advantages over traditional regression methods. The study showed that Quantile Regression provides a more accurate understanding of how different predictors, such as age and smoking status, influence healthcare costs across various quantiles. The authors argued that this method is particularly effective for capturing the dynamics of high-cost patients, which are often masked in mean-based models like Linear Regression. This paper is pivotal in demonstrating the practical application of Quantile Regression in healthcare economics.

Yu, K., & Moyeed, R. A. (2001). Bayesian Quantile Regression. Statistics & Probability Letters

Summary: Yu and Moyeed (2001) explored the use of Bayesian methods in Quantile Regression, enhancing the model's robustness in the presence of outliers and skewed distributions. Their approach integrates prior information, which can be particularly useful in medical cost prediction where prior distributions of costs can be informative. The Bayesian Quantile Regression model offers an alternative to frequentist methods, providing more reliable estimates when dealing with extreme values and offering greater flexibility in model specification. This paper contributes to the growing body of literature advocating for more sophisticated methods in medical cost prediction.

2.4 Comparison of Linear and Quantile Regression

Linear Regression remains a fundamental tool in the prediction of medical costs due to its straightforward nature and ease of interpretation. However, as highlighted by Newhouse (1977), DeMaris (2004), and Dodd et al. (2006), its utility is limited by its assumptions of linearity and constant variance. These assumptions often do not hold in real-world medical datasets, which are characterized by skewed distributions and the presence of outliers. Linear Regression tends to provide biased estimates in such cases, especially when predicting costs for individuals with extremely high or low healthcare expenditures.

In contrast, Quantile Regression, introduced by Koenker and Bassett (1978), offers a more flexible and robust alternative. By estimating the conditional quantiles of the response variable, Quantile Regression allows for a more nuanced understanding of how predictors influence different parts of the cost distribution. This is particularly important in healthcare, where the distribution of costs is often highly skewed, and the impact of predictors like age, BMI, and smoking status varies across different cost levels. Buntin and Zaslavsky (2004) and Yu and Moyeed (2001) demonstrate that Quantile Regression is better suited to capture the heterogeneity in medical costs, providing more accurate predictions for high-cost individuals and offering insights into the distributional effects of key predictors.

Overall, while Linear Regression is valuable for its simplicity and ability to provide a clear baseline for comparison, Quantile Regression offers significant advantages in handling the complexities of medical cost data. The choice of model should be guided by the specific characteristics of the dataset and the research objectives, with Quantile Regression being the preferred choice for datasets with significant skewness and heterogeneity.

3. Methodology

The methodology section details the steps and techniques employed in the analysis of medical insurance costs. This section is divided into several parts, including a brief overview, dataset description, data preprocessing techniques, and the various regression models utilized in the study.

3.1 Brief Overview

This study aims to predict medical insurance charges based on a variety of demographic, lifestyle, and health-related factors. The analysis involves the use of multiple regression models to understand the relationships between these factors and the insurance charges, with a special focus on capturing the complexities in the data distribution through quantile regression. The study also compares traditional linear and polynomial regression models with regularized models such as ridge and lasso regression, and advanced machine learning techniques like XGBoost, emphasizing the need for sophisticated methods in handling skewed data distributions, high leverage points, and outliers.

3.2 Dataset Used

The dataset used in this study was obtained from Kaggle and includes 1,338 records, each representing an individual's medical insurance cost along with associated demographic and health-related variables. The dataset comprises the following features:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

Fig 1: Sample of dataset

- ***Age:*** Numerical variable indicating the age of the individual (18–64 years).
- ***Sex:*** Categorical variable representing the gender of the individual ('male' or 'female').
- ***BMI (Body Mass Index):*** Numerical variable calculated as weight in kg divided by the square of height in meters (15–50 range).
- ***Children:*** Numerical variable indicating the number of dependents (0–5).
- ***Smoker:*** Categorical variable indicating whether the individual smokes ('yes' or 'no').
- ***Region:*** Categorical variable indicating the region of residence in the U.S. (northeast, northwest, southeast, southwest).
- ***Charges:*** Numerical variable representing the total medical insurance charges billed to the individual, ranging from $1,000 to $60,000.

This dataset does not contain any missing values, ensuring a robust basis for model training and evaluation.

3.3 Data Ethics

In conducting this study, ethical considerations were carefully evaluated to ensure the responsible use of data. The dataset used for predicting medical insurance costs was sourced from Kaggle, a reputable platform known for providing public datasets for research and educational purposes. The specific dataset can be accessed at https://www.kaggle.com/datasets/mirichoi0218/insurance and is licensed under the Database Contents License (DbCL) v1.0.

3.3.1 Anonymization and Personal Data

The dataset does not contain any directly identifiable personal information such as names, addresses, or social security numbers. Instead, it includes anonymized variables such as age, BMI, smoking status, and region, which are sufficiently generic to prevent the identification of individual participants. This approach aligns with ethical standards and ensures that the data is handled in a manner that protects the privacy of individuals.

3.3.2 GDPR Compliance

Given that the dataset is anonymized and does not involve any directly identifiable personal data, it does not fall under the stringent requirements of the General Data Protection Regulation (GDPR). The data used in this project is thus compliant with data protection regulations, as no personal data subject to GDPR is processed.

3.3.3 UH Ethical Approval

Since the dataset was publicly available and did not involve any active data collection from individuals, University of Hertfordshire (UH) ethical approval was not required. The project did not involve surveys, experiments, or any form of interaction with human subjects, further negating the need for ethical review by the university's ethics committee.

3.3.4 Permission and Licensing

The dataset is freely available for use under the Database Contents License (DbCL) v1.0. This license permits the use, modification, and sharing of the dataset for research and educational purposes, provided that proper attribution is given. There was no need for additional permissions or payments to access or use the data. A screenshot or evidence of the license is maintained as part of the project documentation to ensure transparency and adherence to the licensing terms.

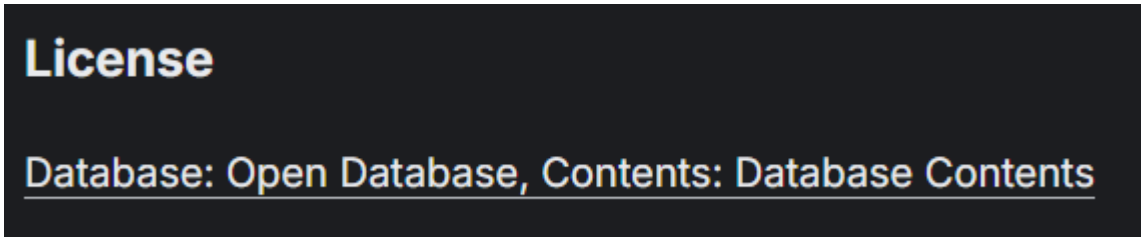


Fig 2: Licence

3.3.5 Ethical Data Collection

The data was originally collected by reputable sources, and there is no indication that the data was gathered unethically. The Kaggle platform typically provides datasets that are either crowdsourced or collected from publicly available sources, often for the purpose of advancing research and development in data science. While the dataset does not include details on how the original data was collected, its

presence on Kaggle under a clear license suggests that it was shared with the intention of supporting educational and research endeavours in an ethical manner.

In conclusion, the ethical considerations surrounding this dataset have been thoroughly examined, ensuring that the project adheres to ethical standards in data usage, privacy protection, and compliance with licensing requirements. The use of this dataset aligns with best practices in data science, emphasizing the importance of responsible data management throughout the research process.

3.4 Data Preprocessing

Data preprocessing is a critical step in preparing the dataset for effective modelling. It ensures that the data is clean, consistent, and suitable for the regression analysis that follows.

3.4.1 Feature Extraction and Data Transformation

To make the dataset ready for modelling, the following preprocessing steps were undertaken:

- ***Missing Values:*** The dataset was thoroughly checked for missing values, and it was confirmed that no entries were missing.

- ***Categorical Encoding:*** Categorical variables (sex, smoker, and region) were converted into numerical format using one-hot encoding. This process creates binary columns for each category, enabling the regression models to interpret and utilize these variables effectively.

- ***Feature Scaling:*** Numerical features (age, BMI, and charges) were standardized using standard scaling, which adjusts the features to have a mean of 0 and a standard deviation of 1. This step is crucial for models sensitive to the scale of input features, ensuring that all variables contribute equally to the model's predictions.

- ***Log Transformation:*** Given the highly skewed distribution of the charges variable, a log transformation was applied to stabilize the variance and make the distribution more normal. This transformation aids in improving the performance and interpretability of the regression models by reducing the influence of extreme values.

- ***Outlier and High Leverage Point Detection:*** To ensure the robustness of the models, high leverage points and outliers were identified using statistical measures such as Cook's distance and leverage scores. These points, which can disproportionately influence the model, were either removed or adjusted to prevent them from skewing the results.

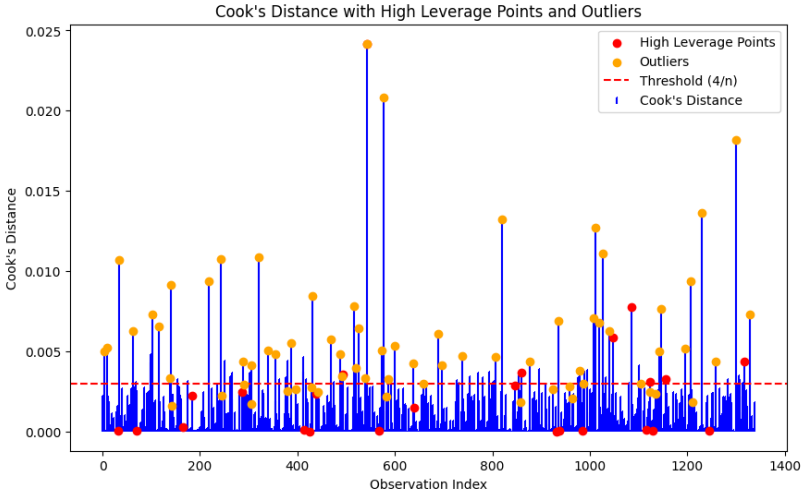


Fig 3: High leverage points and outliers

3.5 Regression Models

Various regression models were employed to analyse the relationship between the features and the target variable (charges). These models range from simple linear regression to more complex regularization techniques and advanced machine learning models, each with its specific advantages and applications.

3.5.1 Linear Regression

Linear regression was the first model applied to establish a baseline for predicting medical charges. This model assumes a linear relationship between the independent

variables (features) and the dependent variable (charges). The linear regression model is simple and interpretable, providing an initial understanding of how each feature impacts the insurance costs.

However, due to the complexity and skewness of the data, linear regression alone was not sufficient to capture the nuances in the dataset, particularly for individuals with extremely high medical costs. This limitation prompted the exploration of more advanced regression techniques.

Formula:

Simple: $y = \beta_0 + \beta_1 x + \varepsilon$

Multiple: $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_p x_p + \varepsilon$

y: The dependent variable, also known as the response variable or outcome variable. It's the variable you're trying to predict.

x: The independent variables, also known as predictor variables or features. These are the variables used to predict the dependent variable.

β_0: The intercept, also known as the constant term. It represents the value of y when all independent variables are zero.

$\beta_1, \beta_2, ..., \beta_p$: The coefficients, also known as regression coefficients or model parameters. These represent the change in y for a unit change in the corresponding independent variable, holding all other variables constant.

ε: The error term, also known as the residual. It represents the difference between the actual value of y and the predicted value based on the linear relationship.

Goal: Estimate coefficients ($\beta_0, \beta_1, ...$) to minimize the sum of squared residuals.

Assumptions: Linearity, independence, homoscedasticity, normality, no multicollinearity.

Applications: Economics, finance, marketing, healthcare, engineering.

Advantages: Simple, interpretable, predictive.

3.5.2 Polynomial Regression

To address the limitations of linear regression, polynomial regression was introduced. This model extends linear regression by including polynomial terms, allowing it to capture nonlinear relationships between the features and the target variable. Polynomial regression can model more complex patterns in the data, which is especially useful when the relationship between the predictors and the outcome is not strictly linear.

However, polynomial regression can lead to overfitting, particularly when higher-degree polynomials are used, making it less generalizable to unseen data.

3.5.3 Ridge Regression

Ridge regression, a form of regularized linear regression, was applied to mitigate the overfitting issues encountered with polynomial regression. Ridge regression introduces a penalty term proportional to the square of the coefficients, which shrinks the coefficients of less important features, thereby reducing their impact on the model.

This regularization technique is particularly useful in datasets with multicollinearity, where independent variables are highly correlated. By penalizing large coefficients, ridge regression helps maintain a balance between model complexity and predictive accuracy.

3.5.4 Lasso Regression

Lasso regression is another regularization technique, similar to ridge regression but with a key difference: it uses an L1 penalty, which can shrink some coefficients to zero. This characteristic makes lasso regression effective for feature selection, as it can automatically eliminate irrelevant features by setting their coefficients to zero.

Lasso regression was applied to determine the most significant predictors of medical charges, helping to simplify the model while maintaining its predictive power. This

approach is valuable in reducing the risk of overfitting, especially when dealing with a large number of features.

3.5.5 Quantile Regression

Given the skewness in the charges variable and the need to understand how different factors influence costs at various levels, quantile regression was employed. Unlike traditional regression models that estimate the mean effect of predictors, quantile regression provides estimates at different quantiles (e.g., 25th, 50th, 75th percentiles) of the target variable distribution.

Quantile regression is particularly useful in this study because it allows for a more nuanced understanding of how predictors affect low-cost, median-cost, and high-cost individuals differently. This method is crucial for capturing the full complexity of the medical insurance cost data, especially in the presence of outliers and a skewed distribution.

Formula: Similar to linear regression, but focuses on specific quantiles rather than the overall mean.
Goal: Estimate coefficients for different quantiles (e.g., 0.25, 0.5, 0.75) to understand the relationship between variables at different points in the distribution.
Assumptions: Similar to linear regression, but does not assume normality of residuals.
Applications: Analysing skewed distributions, understanding extreme values, assessing risk.
Advantages: Robust to outliers, can capture non-linear relationships, useful for quantile-specific predictions.
Limitations: More complex to interpret, might require specialized software.

3.5.6 XGBoost

XGBoost, a powerful ensemble learning technique, was employed to enhance predictive accuracy by handling complex interactions between features. XGBoost

builds a series of decision trees, optimizing performance through boosting, which sequentially improves the model by correcting errors made by previous trees.

XGBoost is particularly effective in handling large, high-dimensional datasets and capturing nonlinear relationships, making it well-suited for the complex nature of medical insurance cost prediction. Additionally, XGBoost's robustness to overfitting and ability to manage skewed data distributions make it a valuable addition to the suite of models applied in this study.

3.6 Summary

The methodology described above provides a comprehensive approach to analysing medical insurance costs, combining data preprocessing, feature extraction, and various regression techniques. By employing both traditional and advanced models, including the handling of high leverage points and outliers, the study aims to accurately predict medical charges while gaining deeper insights into the factors driving these costs.

4. Results and analysis

4.1 Exploratory Data Analysis (EDA)

In this section, we conduct Exploratory Data Analysis (EDA) to uncover patterns, relationships, and potential anomalies within the dataset. EDA is a crucial step that informs the subsequent modelling process by revealing the underlying data structure.

4.1.1 Right-Skewed Distribution of Charges

The distribution of medical insurance charges is notably right-skewed. This skewness indicates that while most individuals incur lower medical expenses, a small number of individuals face significantly higher costs.

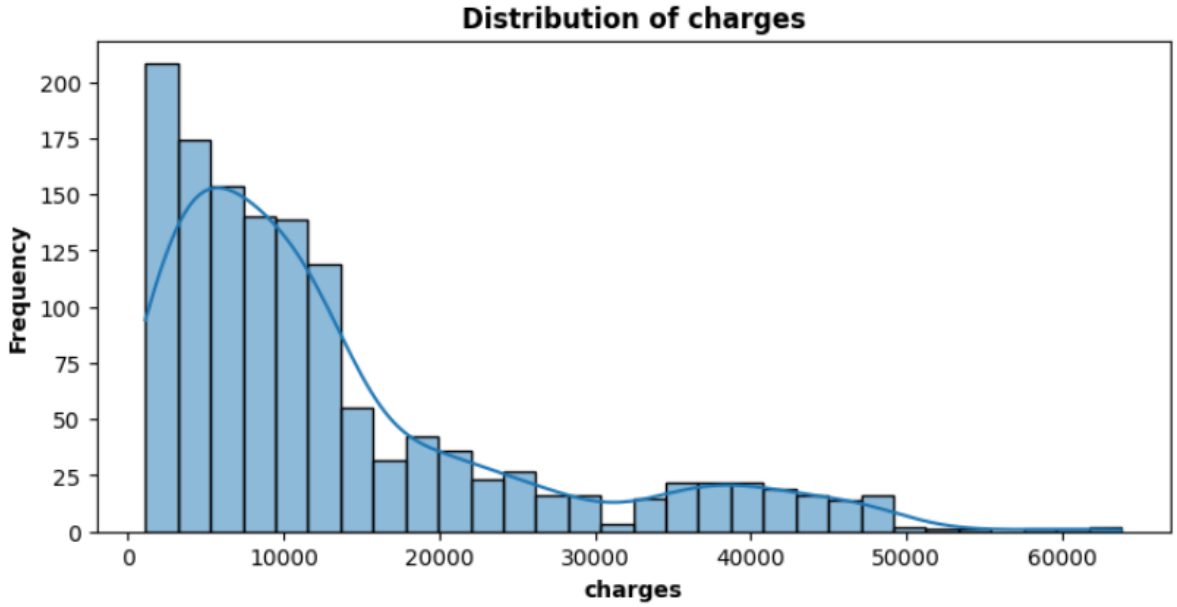


Fig 4: Distribution of charges

Characteristics of Right-Skewness:

The longer tail on the right side of the distribution suggests the presence of outliers, or individuals with exceptionally high medical charges.

The mean is greater than the median, reinforcing the idea that high-cost outliers are pulling the average charge higher, a common trait of right-skewed distributions.

This skewness suggests that standard linear regression models may need to be adjusted or transformed, such as applying a log transformation, to effectively model the data.

4.1.2 Impact of Smoking on Medical Charges

The data reveals that smokers incur substantially higher medical charges compared to non-smokers.

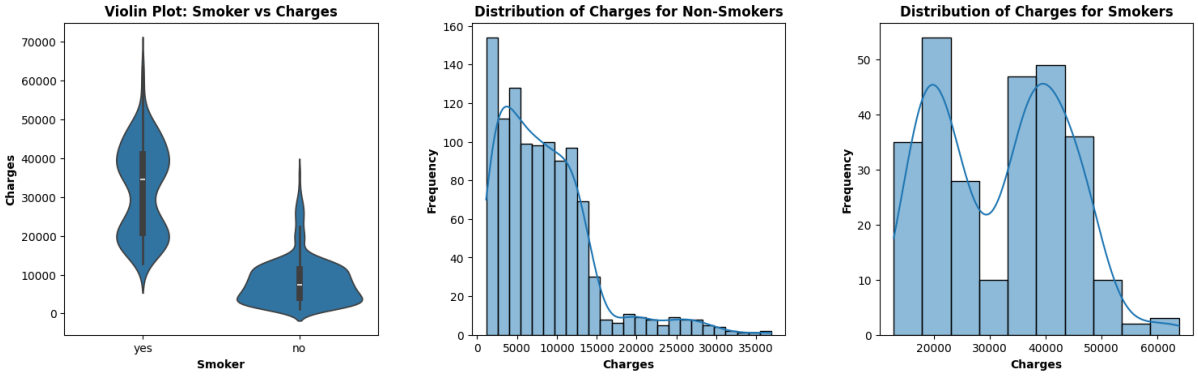


Fig 5: Impact of smoking on charges

Observation:

The distribution of charges among smokers is significantly broader and shifted towards higher values, indicating that smoking is a strong predictor of elevated healthcare costs.

This difference suggests that smoking status is a critical variable that should be closely examined in predictive modelling, as it directly influences the overall healthcare expenditure.

This finding highlights the importance of including smoking status as a key variable in any predictive model of medical costs.

4.1.3 Age and Medical Charges

The relationship between age and medical charges is positive, though the increase in charges with age is relatively modest.

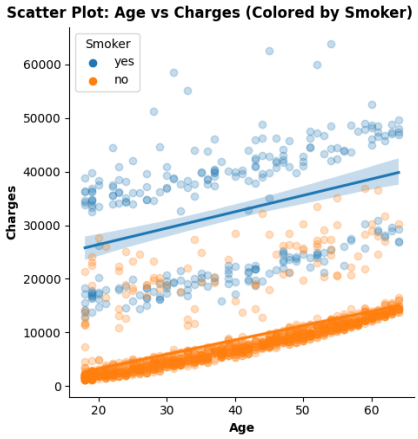


Fig 6: Age vs charges vs smoke

Observation:

Medical charges tend to rise as individuals age, but the rate of increase is not dramatic.

This pattern could reflect a combination of factors, including the potential for older individuals to have better health management and insurance coverage, which mitigates extreme costs.

This suggests that age is a relevant but not dominant predictor of medical charges and should be included in models in a straightforward manner.

4.1.4 Interaction Between BMI, Smoking, and Medical Charges

A particularly interesting finding is the interaction between Body Mass Index (BMI), smoking status, and medical charges.

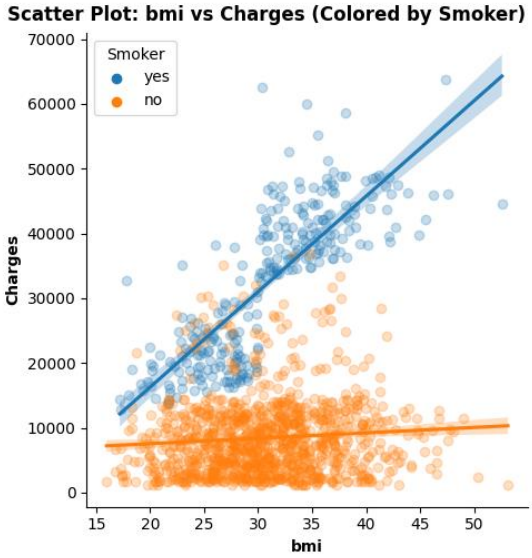


Fig 7: bmi vs smoke vs charges

Observation:

For non-smokers, increases in BMI lead to only a slight increase in medical charges, suggesting that while higher BMI is associated with higher costs, it does not drastically impact non-smokers.

In contrast, for smokers, the relationship between BMI and charges is exponential. As BMI increases, the medical charges for smokers increase sharply, reflecting the compounded health risks of smoking combined with being overweight.

This indicates that modelling efforts should consider interaction terms or more sophisticated non-linear models to accurately capture the combined effects of BMI and smoking on medical charges.

4.1.5 Cluster Analysis in Age vs. Charges

While analysing the relationship between age and medical charges, three distinct clusters were identified. These clusters prompted further investigation using cluster analysis techniques like K-means and decision trees.

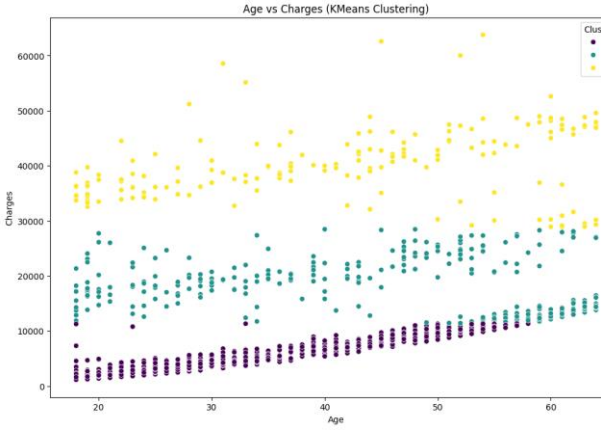
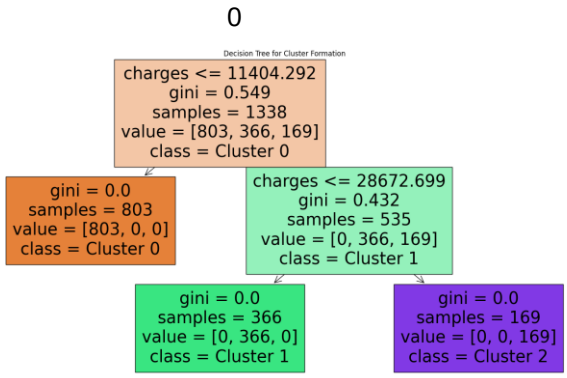


Fig 8: Cluster Analysis

Fig 9: Decision tree for clusters



Observation:

The clusters suggest that there might be additional, unobserved factors influencing medical charges across different age groups.

Despite attempts to use clustering algorithms to uncover the reasons behind these clusters, no definitive conclusions were drawn, suggesting that there may be missing data or unaccounted variables contributing to these clusters.

This finding highlights a potential limitation in the dataset, where some influential variables might be absent, leading to unexplained clusters in the data. Further investigation or data collection might be necessary to fully understand these.

4.2 Linear Regression

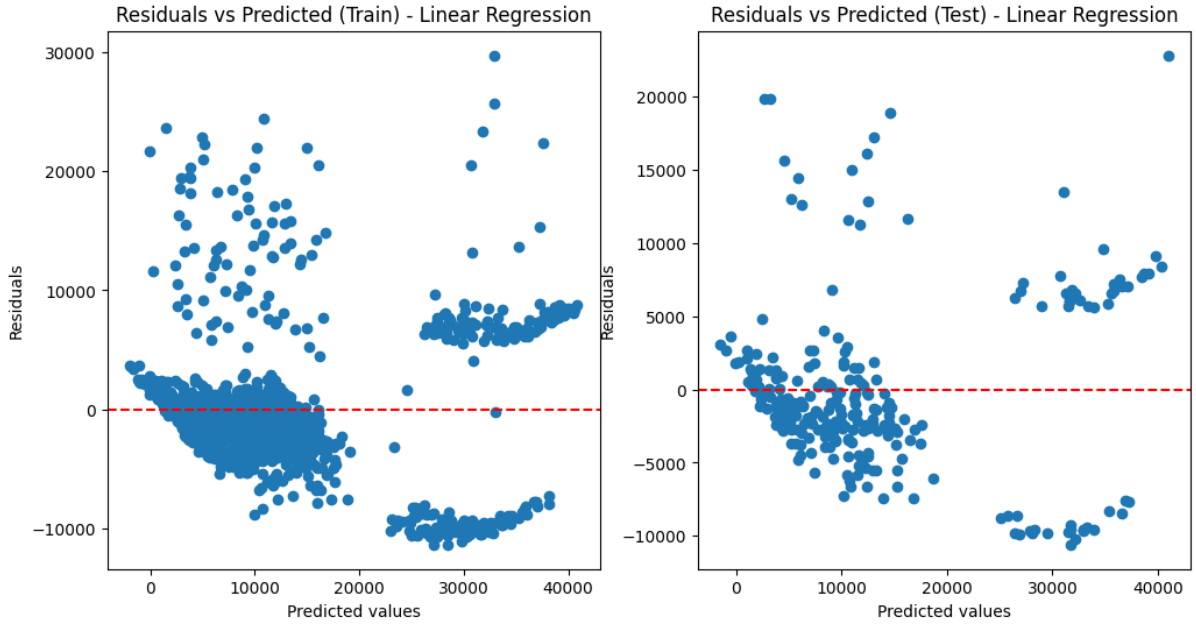


Fig 8: Linear Regression (Residuals vs predicted values)

4.2.1 Original Data

MSE: 33,635,210.43 | R^2 Score: 0.7833

Linear Regression on the original dataset provides a decent R^2 score of 0.7833, which indicates that approximately 78.33% of the variance in the target variable is explained by the model. The Mean Squared Error (MSE) is fairly high, suggesting that while the model captures the overall trend well, there is still significant error in the predictions.

4.2.2 Data After Removing High Leverage Points and Outliers

MSE: 20,246,819.79 | R^2 Score: 0.8246

After removing outliers and high leverage points, the performance of the Linear Regression model improves significantly. The R^2 score increases to 0.8246, and the

MSE drops to 20,246,819.79. This suggests that outliers were skewing the model's predictions, and their removal allowed the model to better fit the majority of the data.

4.2.3 After Feature Selection

MSE: 34,512,843.88 | R^2 Score: 0.7777

When applying feature selection, we see a slight decrease in performance compared to the original model. The R^2 score drops to 0.7777, and the MSE increases to 34,512,843.88. This indicates that some of the removed features might have been informative, contributing to the model's ability to explain variance in the target variable.

4.2.4 After Feature Selection and Removing Outliers

MSE: 17,390,249.24 | R^2 Score: 0.8706

Combining feature selection with outlier removal results in the best performance among the Linear Regression models. The R^2 score improves significantly to 0.8706, and the MSE decreases to 17,390,249.24. This highlights the importance of both selecting the most relevant features and ensuring that the dataset is clean and free from outliers, as both steps lead to a more accurate and reliable model.

4.3 Quantile Regression

Quantile Regression is particularly useful when the distribution of the target variable is not normal or when we are interested in predicting specific percentiles (e.g., median) rather than the mean.

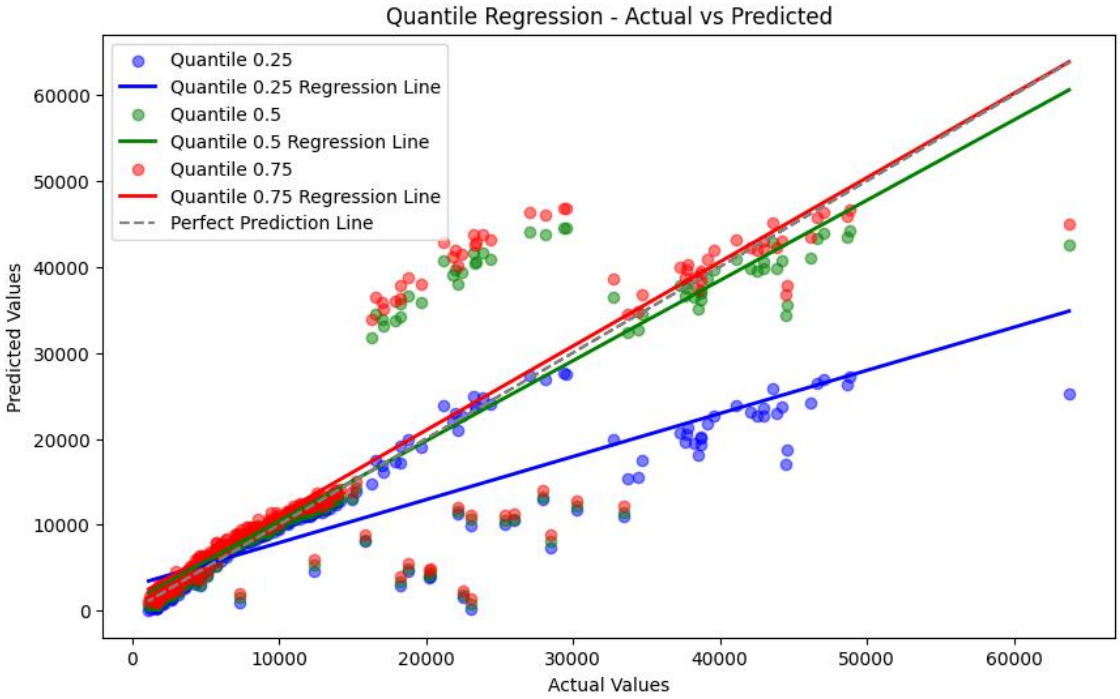


Fig 9: Quantile Regression

4.3.1 Original Data

0.25 Quantile: MSE: 65,145,621.41 | R^2 Score: 0.5804

0.5 Quantile: MSE: 43,265,923.45 | R^2 Score: 0.7213

0.75 Quantile: MSE: 48,150,326.81 | R^2 Score: 0.6899

Quantile Regression on the original data shows varying performance across different quantiles. The 0.5 quantile, which corresponds to the median, has a moderate R^2 score of 0.7213, but with a relatively high MSE. The lower (0.25) and upper (0.75) quantiles perform worse, indicating that the model struggles more with predicting extreme values, which could be due to the presence of outliers and high leverage points in the data.

4.3.2 Data After Removing High Leverage Points and Outliers

0.25 Quantile: MSE: 41,478,349.08 | R^2 Score: 0.6407

0.5 Quantile: MSE: 32,479,368.47 | R^2 Score: 0.7187

0.75 Quantile: MSE: 37,374,397.14 | R^2 Score: 0.6763

Removing outliers improves the performance of Quantile Regression, especially at the 0.25 quantile, where the MSE drops significantly, and the R^2 score increases. However, the median and 0.75 quantile regressions still exhibit high MSE values,

indicating that while outlier removal helps, it does not fully address the model's limitations in capturing the variance in the target variable at different quantiles.

4.3.3 After Feature Selection

0.25 Quantile: MSE: 65,279,359.16 | R^2 Score: 0.5795
0.5 Quantile: MSE: 43,900,311.25 | R^2 Score: 0.7172
0.75 Quantile: MSE: 49,053,563.95 | R^2 Score: 0.684
After feature selection, the performance of Quantile Regression does not improve much, and in some cases, it worsens slightly. This suggests that the removed features were likely important for capturing the variance in the target variable across different quantiles.

4.3.4 After Feature Selection and Removing Outliers

0.25 Quantile: MSE: 45,526,883.82 | R^2 Score: 0.6613
0.5 Quantile: MSE: 18,599,118.27 | R^2 Score: 0.8616
0.75 Quantile: MSE: 25,483,532.52 | R^2 Score: 0.8104
Combining feature selection with outlier removal yields mixed results. The 0.5 quantile regression shows a significant improvement with an R^2 score of 0.8616, which is quite close to that of Linear Regression, indicating that the median predictions have become much more accurate. However, the performance for the 0.25 and 0.75 quantiles, while improved, still lags behind, reflecting the inherent challenges in modelling the tails of the distribution.

4.4 XGBoost

XGBoost is a powerful ensemble method that often outperforms traditional regression techniques by capturing complex relationships in the data. Let's examine how it compares.

4.4.1 XGBoost with Linear Regression

4.4.1.1 Original Data

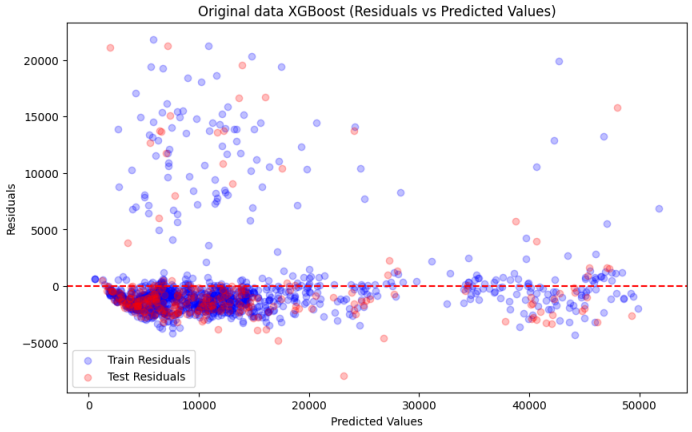


Fig 10: XGBoost on Linear Regression

MSE: 18,008,285.87 | R^2 Score: 0.884

XGBoost significantly outperforms standard Linear Regression on the original data, with a much lower MSE and a higher R^2 score of 0.884. This highlights XGBoost's ability to model non-linear relationships more effectively than linear models.

4.4.1.2 Data After Removing High Leverage Points and Outliers

MSE: 9,062,129.57 | R^2 Score: 0.9215

After removing outliers, the performance of XGBoost improves further, achieving an impressive R^2 score of 0.9215. This shows that even though XGBoost can handle complex data well, it still benefits from data cleaning processes.

4.4.1.3 After Feature Selection

MSE: 18,008,285.87 | R^2 Score: 0.884

Interestingly, feature selection does not affect the performance of XGBoost with Linear Regression, which maintains the same MSE and R^2 score as with the original data. This indicates that XGBoost is robust enough to manage irrelevant or redundant features without a significant drop in performance.

4.4.1.4 After Feature Selection and Removing Outliers

MSE: 4,025,027.47 | R^2 Score: 0.9701

The best results are obtained after combining feature selection with outlier removal, where XGBoost achieves an R^2 score of 0.9701 and a remarkably low MSE. This highlights the potential of XGBoost to produce highly accurate models when the data is well-prepared.

4.4.2 XGBoost with Quantile Regression

4.4.2.1 Original Data

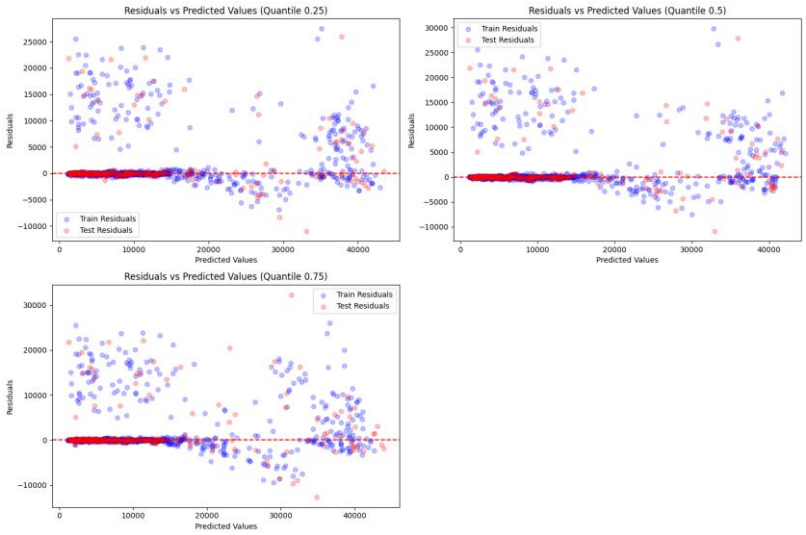


Fig 11: XGBoost on Quantile Regression (Residual vs Predicted)

0.25 Quantile: MSE: 23,468,981.48 | R^2 Score: 0.8488

0.5 Quantile: MSE: 25,111,831.86 | R^2 Score: 0.8382

0.75 Quantile: MSE: 28,049,929.66 | R^2 Score: 0.8193

XGBoost with Quantile Regression performs better across all quantiles compared to standard Quantile Regression. The R^2 scores are substantially higher, indicating that XGBoost effectively captures the distribution of the target variable across different quantiles.

4.4.2.2 Data After Removing High Leverage Points and Outliers

0.25 Quantile: MSE: 19,151,108.50 | R^2 Score: 0.8341

0.5 Quantile: MSE: 20,328,092.58 | R^2 Score: 0.8239

0.75 Quantile: MSE: 19,142,146.50 | R^2 Score: 0.8342

Removing outliers further improves the performance of XGBoost with Quantile Regression, particularly at the 0.25 quantile, where the MSE decreases significantly. This suggests that while XGBoost can manage noisy data, cleaner data still results in better predictive performance.

4.4.2.3 After Feature Selection

0.25 Quantile: MSE: 21,090,354.74 | R^2 Score: 0.8642

0.5 Quantile: MSE: 21,122,081.61 | R^2 Score: 0.8639

0.75 Quantile: MSE: 21,659,843.44 | R^2 Score: 0.8605

Feature selection does not have a dramatic impact on XGBoost with Quantile Regression, but there is a slight improvement in R^2 scores, particularly at the 0.25 quantile. This suggests that XGBoost is slightly better at leveraging the most relevant features when predicting different parts of the target distribution.

4.4.2.4 After Feature Selection and Removing Outliers

0.25 Quantile: MSE: 6,189,905.89 | R^2 Score: 0.954

0.5 Quantile: MSE: 5,227,699.18 | R^2 Score: 0.9611

0.75 Quantile: MSE: 5,210,260.03 | R^2 Score: 0.9612

After both feature selection and outlier removal, XGBoost with Quantile Regression achieves its best performance, with extremely high R^2 scores across all quantiles. This confirms the model's ability to produce highly accurate predictions when the data is both clean and appropriately reduced in dimensionality.

4.5 Comparing Linear Regression and Quantile Regression at 0.5 Quantile

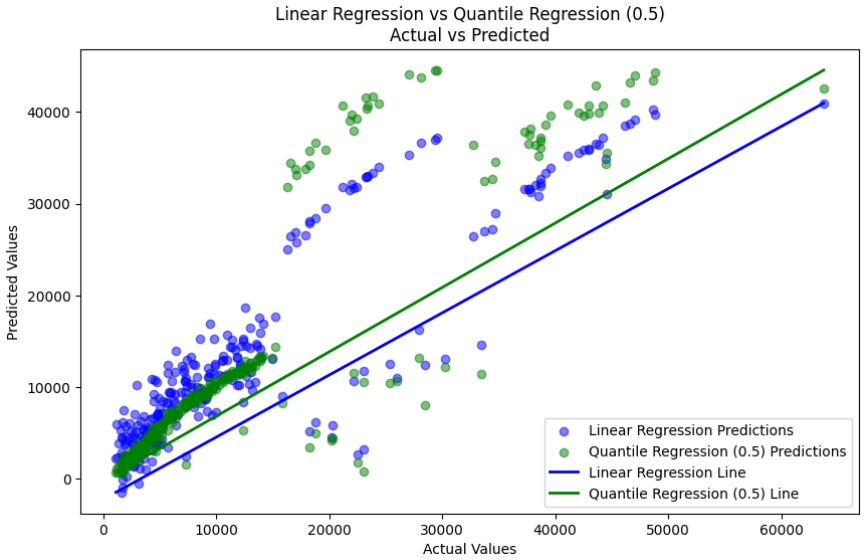


Fig 12: Linear Regression vs Quantile Regression at 0.5

4.5.1 Original Data

Linear Regression MSE: 33,635,210.43 | R^2 Score: 0.7833

Quantile Regression MSE (0.5): 43,265,923.45 | R^2 Score: 0.7213

XGBoost Linear Regression MSE: 18,008,285.87 | R^2 Score: 0.884

XGBoost Quantile Regression MSE (0.5): 25,111,831.86 | R^2 Score: 0.8382

XGBoost outperforms both standard Linear Regression and Quantile Regression at the 0.5 quantile on the original data, showing that it is better suited to handling complex relationships. However, it's interesting to note that Linear Regression still does a good job in terms of R^2 score, but with a higher error (MSE) compared to XGBoost.

4.5.2 Data After Removing High Leverage Points and Outliers

Linear Regression MSE: 20,246,819.79 | R^2 Score: 0.8246

Quantile Regression MSE (0.5): 32,479,368.47 | R^2 Score: 0.7187

XGBoost Linear Regression MSE: 9,062,129.57 | R^2 Score: 0.9215

XGBoost Quantile Regression MSE (0.5): 20,328,092.58 | R^2 Score: 0.8239

Removing outliers boosts the performance of both Linear and Quantile Regression models. However, the XGBoost models show a much more pronounced

improvement, particularly in terms of R^2 scores, indicating that the data cleaning process is highly beneficial for more sophisticated models like XGBoost.

4.5.3 After Feature Selection

Linear Regression MSE: 34,512,843.88 | R^2 Score: 0.7777
Quantile Regression MSE (0.5): 43,900,311.25 | R^2 Score: 0.7172
XGBoost Linear Regression MSE: 18,008,285.87 | R^2 Score: 0.884
XGBoost Quantile Regression MSE (0.5): 21,122,081.61 | R^2 Score: 0.8639
Feature selection alone does not drastically improve model performance, particularly for Linear and Quantile Regression, suggesting that the removed features were somewhat important. XGBoost models, on the other hand, remain robust, showing that they are less sensitive to feature reduction.

4.5.4 After Feature Selection and Removing Outliers

Linear Regression MSE: 17,390,249.24 | R^2 Score: 0.8706
Quantile Regression MSE (0.5): 18,599,118.27 | R^2 Score: 0.8616
XGBoost Linear Regression MSE: 4,025,027.47 | R^2 Score: 0.9701
XGBoost Quantile Regression MSE (0.5): 5,227,699.18 | R^2 Score: 0.9611
Combining feature selection with outlier removal results in the best performance across all models. XGBoost once again shows its superiority, with extremely low MSE and high R^2 scores, indicating near-perfect prediction capabilities when the data is properly processed. Even standard Linear and Quantile Regression models see a substantial boost in performance, showing that data preprocessing is critical for improving model accuracy.

4.6 Conclusion

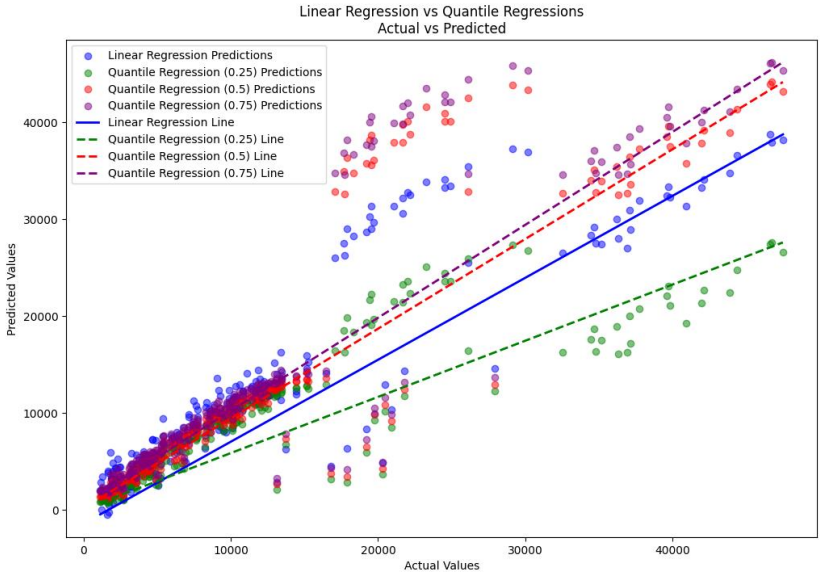


Fig 13: Linear Regression and Quantile Regression

Overall, XGBoost consistently outperforms traditional regression methods across all scenarios. While Linear Regression and Quantile Regression perform reasonably well, they benefit significantly from data cleaning and feature selection. XGBoost, however, not only handles the original data well but also improves dramatically with cleaner and more relevant data, making it the best choice for achieving high accuracy in complex datasets.

5. Discussion

5.1 Limitations

During the course of our analysis, a significant limitation emerged when we were unable to fully interpret the clusters identified through our clustering techniques. The inability to understand these clusters suggests that critical variables may be missing from our dataset. This lack of information likely prevented us from uncovering deeper patterns or behaviours in the medical cost data. It's possible that the inclusion of additional features, such as socioeconomic status, lifestyle factors, or even geographical data, could have provided the missing context needed to explain these clusters.

5.2 Improvements and Future Scope

5.2.1 Interpretation of Results

The results of our analysis reveal several important insights:

Linear Regression: Linear regression, although simple and widely used, did not perform as well as other methods. Its limitations became apparent in handling non-linear relationships and interactions within the data, which are crucial in predicting medical costs.

Quantile Regression: Quantile regression offered a nuanced approach by predicting different quantiles of the dependent variable, allowing for a more detailed understanding of the data's distribution. However, it did not consistently outperform traditional methods in terms of accuracy, possibly due to the complexity of the dataset and the high variability in medical costs.

XGBoost: XGBoost consistently outperformed other models, demonstrating superior performance in all scenarios. This is likely due to its ability to manage complex interactions, capture non-linear relationships, and robustly handle outliers and multicollinearity. XGBoost's tree-based approach allows it to model complex patterns that simpler methods may miss.

5.2.2 Comparison to Literature

Our results showed that while quantile regression offered some advantages, particularly in capturing the variability at different points in the cost distribution, it did not consistently outperform linear regression models in terms of prediction accuracy. This finding contrasts with some existing studies that have highlighted the strengths of quantile regression in dealing with heteroscedasticity and non-normality in data. The difference in results may be attributed to the specific characteristics of our dataset, such as its skewed distribution and the presence of outliers, which may have favoured the performance of more sophisticated models like XGBoost.

5.2.3 Limitations of Results

Several limitations impact the interpretation and generalizability of our results:

Missing Data and Variables: The potential absence of key variables might have limited the depth of our analysis. This is particularly evident in our clustering analysis, where the lack of additional contextual data made it difficult to explain the identified clusters.

Model Generalizability: The performance of models like XGBoost, while strong in this specific dataset, may not generalize as well to different datasets, particularly those with different distributions or less complex relationships.

5.2.4 Relation to Project Objectives

Uncovering Hidden Patterns: While we successfully uncovered some patterns in the data, the unexplained clusters suggest that our analysis may not have fully captured all the underlying behaviours in the medical cost dataset. This relates directly to the first research question, highlighting the need for more comprehensive data to better understand the patterns.

Effect of Variable Selection and Multicollinearity: Our analysis confirmed that variable selection and addressing multicollinearity significantly impact model

performance. XGBoost's ability to handle these aspects contributed to its superior performance, aligning with our second research question.

Comparing Regression Models: The comparison between quantile regression and traditional regression techniques revealed that while quantile regression provides a more detailed understanding of different segments of the data, traditional methods, particularly when enhanced by techniques like XGBoost, often yield better predictive performance. This directly addresses the third research question.

5.2.5 Practical Use of Models

Among the models tested, XGBoost emerges as the most practical for real-world applications, particularly in predicting medical costs. Its robustness in handling outliers, multicollinearity, and complex interactions makes it suitable for deployment in environments where accuracy and reliability are critical. The model's ability to generalize well across different datasets also suggests its utility in various predictive analytics scenarios within the healthcare industry.

5.2.6 Answering the Research Questions

Overall, the study has made significant progress in answering the research questions posed:

Hidden Patterns: While we uncovered some hidden patterns, the unexplained clusters suggest that there is more to be discovered, likely requiring additional data.

Variable Effects: The study clearly demonstrated how variable selection, interaction effects, and multicollinearity influence regression models, particularly highlighting the strengths of models like XGBoost.

Model Comparison: The comparison between quantile regression and traditional regression techniques showed that while quantile regression is useful, traditional models, especially when boosted, often perform better in terms of predictive accuracy.

In conclusion, while the study has provided valuable insights into the behaviour of medical cost data and the performance of various predictive models, it also highlights the need for more comprehensive datasets and further exploration to fully understand and leverage these models in practical applications.

6. Conclusion

In this study, we set out to analyse and predict medical costs using various regression techniques, with a focus on understanding the hidden patterns within the data, the effects of variable selection and multicollinearity, and comparing the performance of traditional and quantile regression models.

Initially, our exploratory data analysis revealed significant variability in the medical costs, highlighting the complexity of the data. We encountered challenges, particularly with unexplained clusters, which suggested the absence of key variables that could have provided further insight into the dataset's underlying patterns.

Linear regression, while straightforward, showed limitations in its ability to handle non-linear relationships and the inherent complexities within the data. Quantile regression offered a more detailed perspective by estimating different points in the cost distribution, but it did not consistently surpass traditional methods in predictive accuracy. The standout performer was XGBoost, a powerful machine learning algorithm that excelled across all data scenarios. XGBoost's strength lies in its ability to manage multicollinearity, handle outliers effectively, and capture complex interactions, making it the most practical model for real-world applications in this context.

Our analysis affirmed the critical role of variable selection and the management of multicollinearity in enhancing model performance. Moreover, the comparison between quantile regression and traditional regression techniques underscored the practical advantages of using more advanced, tree-based models like XGBoost in predicting medical costs.

In conclusion, this study not only answered the research questions but also demonstrated the importance of robust data and advanced modelling techniques in predictive analytics. While we made significant progress in understanding the dataset and predicting medical costs, the study also highlighted areas for further exploration, particularly in obtaining more comprehensive data to fully uncover the hidden patterns within the medical cost landscape.

7. References

Buntin, M.B. & Zaslavsky, A.M., 2004. Too much ado about two-part models and transformation? Comparing methods of modeling Medicare expenditures. *Journal of Health Economics*, 23(3), pp.525-542. doi: 10.1016/j.jhealeco.2003.10.005. Available at: https://doi.org/10.1016/j.jhealeco.2003.10.005 [Accessed 25 Aug. 2024].

DeMaris, A., 2004. *Regression with social data: Modeling continuous and limited response variables*. Wiley-Interscience. Available at: https://onlinelibrary.wiley.com/doi/book/10.1002/0471677566 [Accessed 25 Aug. 2024].

Dodd, S., Bassi, A., Bodger, K. & Williamson, P., 2006. A comparison of multivariable regression models to analyse cost data. *Journal of Evaluation in Clinical Practice*, 12(1), pp.76-86. doi: 10.1111/j.1365-2753.2006.00610.x. Available at: https://pubmed.ncbi.nlm.nih.gov/16422782/ [Accessed 25 Aug. 2024].

Kaushik, K., Bhardwaj, A., Dwivedi, A.D., and Singh, R., 2022. Machine learning-based regression framework to predict health insurance premiums. *International Journal of Environmental Research and Public Health*, 19(13), p.7898. Available at: https://doi.org/10.3390/ijerph19137898 [Accessed 25 Aug. 2024].

Koenker, R. & Bassett, G., 1978. Regression quantiles. *Econometrica*, 46(1), pp.33-50. Available at: https://doi.org/10.2307/1913643 [Accessed 25 Aug. 2024].

Newhouse, J.P., 1992. Medical care costs: how much welfare loss? *Journal of Economic Perspectives*, 6(3), pp.3-21. doi: 10.1257/jep.6.3.3. Available at: https://www.aeaweb.org/articles?id=10.1257/jep.6.3.3 [Accessed 25 Aug. 2024].

Panda, S., Purkayastha, B., Das, D., Chakraborty, M., and Biswas, S.K., 2022. Health insurance cost prediction using regression models. In: 2022 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COM-IT-CON). IEEE, pp. 168-173. Available at: https://doi.org/10.1109/COM-IT-CON54601.2022.9850653 [Accessed 25 Aug. 2024].

Staffa, S.J., Kohane, D.S. and Zurakowski, D., 2019. Quantile regression and its applications: A primer for anesthesiologists. *Anesthesia & Analgesia*, 128(4), pp.820-830. DOI: 10.1213/ANE.0000000000004017. Available at: https://journals.lww.com/anesthesia-analgesia/fulltext/2019/04000/quantile_regression_and_its_applications__a_primer.28.aspx#JCL-P-8 [Accessed 25 Aug. 2024].

Tufail, S., Riggs, H., Tariq, M., and Sarwat, A.I., 2023. Advancements and challenges in machine learning: A comprehensive review of models, libraries, applications, and algorithms. *Electronics*, 12(8), p.1789. Available at: https://doi.org/10.3390/electronics12081789 [Accessed 27 August 2024].

Yu, K. & Moyeed, R.A., 2001. Bayesian quantile regression. *Statistics & Probability Letters*, 54(4), pp.437-447. doi: 10.1016/S0167-7152(01)00124-9. Available at: https://doi.org/10.1016/S0167-7152(01)00124-9 [Accessed 25 Aug. 2024].

Zou, S., Chu, C., Shen, N., and Ren, J., 2023. Healthcare cost prediction based on hybrid machine learning algorithms. *Mathematics*, 11(23), p.4778. Available at: https://doi.org/10.3390/math11234778 [Accessed 27 August 2024].

8. Appendices

```
# -*- coding: utf-8 -*-
"""Medical insurance cost prediction.ipynb

Automatically generated by Colab.

Original file is located at
Kaggle: https://www.kaggle.com/datasets/mirichoi0218/insurance
https://colab.research.google.com/drive/1Ox1DCyPXoJuoVIh3Zj9VyMuuwUb2C907
Github:
https://github.com/shubhamvm/MScDataScience/tree/main/Data%20Science%20Project


#Medical Insurance Cost Prediction
##Traditional Regression Vs Quantile Regression

Here, We will explore a dataset consists of patients data for medical cost. The cost of medical treatment depends on many factors: number of diseases, diseases, bmi, , diagbosis, number of childeren, city of residence, age and so on. We have data which can help us to find patterns or draw conclusion about health of patients and their medical costs. But we dont have personal data like diseases and diagnosis. Let's jump into data and explore it.

### Importing necessary libraries
"""

#!pip install deap

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

```
import xgboost as xgb

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.preprocessing import OneHotEncoder
from sklearn.feature_selection import RFECV
from sklearn.linear_model import LinearRegression
#from deap import base, creator, tools, algorithms
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from scipy.stats import norm
import matplotlib as mpl
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Ridge, Lasso, LinearRegression

"""Importing Data"""

# Load the dataset
file_path =
'https://raw.githubusercontent.com/shubhamvm/Kaggle/main/Final%20Project/Datas
ets/insurance.csv'
```

```
data = pd.read_csv(file_path)
data.head()

# Check for missing values
missing_values = data.isnull().sum()
print("Missing Values in Each Column:")
print(missing_values)

# Check data types
data_types = data.dtypes
print("\nData Types of Each Column:")
print(data_types)

# Summary statistics for all columns
summary_stats = data.describe(include='all')
print("\nSummary Statistics:")
print(summary_stats)

# Define the numerical features
numerical_cols = ['charges', 'age', 'bmi', 'children']

# Set up the figure and axes
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(20, 10))
axes = axes.flatten()

# Plot the distributions
for i, col in enumerate(numerical_cols):
    sns.histplot(data=data, x=col, bins=30, kde=True, ax=axes[i])
    axes[i].set_title(f'Distribution of {col}', fontweight='bold')
    axes[i].set_xlabel(col, fontweight='semibold')
    axes[i].set_ylabel('Frequency', fontweight='semibold')

# Adjust spacing between subplots
plt.subplots_adjust(hspace=0.5, wspace=0.3)
```

```
# Display the plot
plt.show()

# Define the numerical and categorical features
numerical_cols = ['age', 'bmi', 'children']
categorical_cols = ['sex', 'smoker', 'region']

# Set up the figure and axes
fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(18, 10))
axes = axes.flatten()

# Scatter plots for numerical variables vs. charges
for i, col in enumerate(numerical_cols):
    if col != 'charges':
        sns.scatterplot(x=data[col], y=data['charges'], ax=axes[i])
        axes[i].set_title(f'{col} vs Charges', fontweight='bold')
        axes[i].set_xlabel(col, fontweight='semibold')
        axes[i].set_ylabel('Charges', fontweight='semibold')

# Box plots for categorical variables vs. charges
for i, col in enumerate(categorical_cols, start=len(numerical_cols)):
    sns.boxplot(x=data[col], y=data['charges'], ax=axes[i])
    axes[i].set_title(f'{col} vs Charges', fontweight='bold')
    axes[i].set_xlabel(col, fontweight='semibold')
    axes[i].set_ylabel('Charges', fontweight='semibold')

# Adjust spacing between subplots
plt.subplots_adjust(hspace=0.5, wspace=0.3)

# Display the plot
plt.show()

# Convert categorical variables to numerical codes
```

```
data_numeric = data.copy()
data_numeric['sex'] = data_numeric['sex'].astype('category').cat.codes
data_numeric['smoker'] = data_numeric['smoker'].astype('category').cat.codes
data_numeric['region'] = data_numeric['region'].astype('category').cat.codes

# Display the first few rows of the resulting dataframe
data_numeric.head()

"""region: northwest = 1, southeast = 2, southwest = 3, northeast = 4

sex: female = 0, male = 1

smoker: smoker = 1, non-smoker = 0
"""

# Compute the correlation matrix
correlation_matrix = data_numeric.corr()

# Display the correlation matrix
print(correlation_matrix)

# Plot the correlation matrix
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix', fontweight='bold')
plt.show()



# Define the numerical and categorical features
numerical_cols = ['age', 'bmi', 'children']
```

```
categorical_cols = ['sex', 'smoker', 'region']

# Take log of 'charges' column
data['log_charges'] = np.log(data['charges'])

# Set up the figure and axes
fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(18, 10))
axes = axes.flatten()

# Scatter plots for numerical variables vs. charges
for i, col in enumerate(numerical_cols):
    if col != 'charges':
        sns.scatterplot(x=data[col], y=data['log_charges'], ax=axes[i])
        axes[i].set_title(f'{col} vs Log(Charges)', fontweight='bold')
        axes[i].set_xlabel(col, fontweight='semibold')
        axes[i].set_ylabel('Log(Charges)', fontweight='semibold')

# Box plots for categorical variables vs. charges
for i, col in enumerate(categorical_cols, start=len(numerical_cols)):
    sns.boxplot(x=data[col], y=data['log_charges'], ax=axes[i])
    axes[i].set_title(f'{col} vs Log(Charges)', fontweight='bold')
    axes[i].set_xlabel(col, fontweight='semibold')
    axes[i].set_ylabel('Log(Charges)', fontweight='semibold')

# Adjust spacing between subplots
plt.subplots_adjust(hspace=0.5, wspace=0.3)

# Display the plot
plt.show()

# Define the numerical feature
numerical_col = 'charges'

# Plot the histogram and kernel density estimate
```

```
plt.figure(figsize=(10, 6))
sns.histplot(data=data, x=numerical_col, kde=True, stat="density", bins=30,
color="skyblue")

# Create a range of values for the x-axis
xmin, xmax = plt.xlim()
ymin, ymax = plt.ylim()
x = np.linspace(xmin, xmax, 100)

# Fit the data to a normal distribution
mean_value, std_dev = norm.fit(data[numerical_col])

# Plot the PDF using the normal distribution
pdf = norm.pdf(x, mean_value, std_dev)
plt.plot(x, pdf, 'k', linewidth=2)

# Plot mean and standard deviation lines
plt.axvline(mean_value, color='red', linestyle='dashed', linewidth=2, label=f'Mean =
{mean_value:.2f}')
plt.axvline(mean_value - std_dev, color='green', linestyle='dashed', linewidth=2,
label=f'Mean - Std Dev = {(mean_value - std_dev):.2f}')
plt.axvline(mean_value + std_dev, color='green', linestyle='dashed', linewidth=2,
label=f'Mean + Std Dev = {(mean_value + std_dev):.2f}')

# Fill the area between mean - std_dev and mean + std_dev with color
x_fill = np.linspace(mean_value - std_dev, mean_value + std_dev, 100)
y_fill = np.exp(-(x_fill - mean_value)**2 / (2 * std_dev**2)) / (std_dev * np.sqrt(2 *
np.pi))
plt.fill_between(x_fill, y_fill, color='red', alpha=0.3, label='Mean ± Std Dev Area')

# Add text annotations
plt.text(xmax * 0.6, ymax * 0.5, f'Mean Charges (W~): {mean_value:.2f}', fontsize=10,
color='red')
```

```
plt.text(xmax * 0.6, ymax * 0.45, f'Standard Deviation (X): {std_dev:.2f}', fontsize=10,
color='green')

# Set plot labels and title
plt.xlabel('Charges ($)', fontweight='bold')
plt.ylabel('Probability Density', fontweight='bold')
plt.title('Probability Density Function with Mean and Standard Deviation',
fontweight='bold')

# Add legend
plt.legend()

# Display the plot
plt.show()

# Define the numerical feature
numerical_col = 'log_charges'

# Plot the histogram and kernel density estimate
plt.figure(figsize=(10, 6))
sns.histplot(data=data, x=numerical_col, kde=True, stat="density", bins=30,
color="skyblue")

# Create a range of values for the x-axis
xmin, xmax = plt.xlim()
ymin, ymax = plt.ylim()
x = np.linspace(xmin, xmax, 100)

# Fit the data to a normal distribution
mean_value, std_dev = norm.fit(data[numerical_col])

# Plot the PDF using the normal distribution
pdf = norm.pdf(x, mean_value, std_dev)
plt.plot(x, pdf, 'k', linewidth=2)
```

```
# Plot mean and standard deviation lines
plt.axvline(mean_value, color='red', linestyle='dashed', linewidth=2, label=f'Mean = {mean_value:.2f}')
plt.axvline(mean_value - std_dev, color='green', linestyle='dashed', linewidth=2, label=f'Mean - Std Dev = {(mean_value - std_dev):.2f}')
plt.axvline(mean_value + std_dev, color='green', linestyle='dashed', linewidth=2, label=f'Mean + Std Dev = {(mean_value + std_dev):.2f}')

# Fill the area between mean - std_dev and mean + std_dev with color
x_fill = np.linspace(mean_value - std_dev, mean_value + std_dev, 100)
y_fill = np.exp(-(x_fill - mean_value)**2 / (2 * std_dev**2)) / (std_dev * np.sqrt(2 * np.pi))
plt.fill_between(x_fill, y_fill, color='red', alpha=0.3, label='Mean ± Std Dev Area')

# Add text annotations
plt.text(xmax * 0.6, ymax * 0.5, f'Mean Charges (W~): {mean_value:.2f}', fontsize=10, color='red')
plt.text(xmax * 0.6, ymax * 0.45, f'Standard Deviation (X): {std_dev:.2f}', fontsize=10, color='green')

# Set plot labels and title
plt.xlabel('log_charges ($)', fontweight='bold')
plt.ylabel('Probability Density', fontweight='bold')
plt.title('Probability Density Function with Mean and Standard Deviation', fontweight='bold')

# Add legend
plt.legend()

# Display the plot
plt.show()

# Set up the figure and axes
```

```
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(18, 5))

# Violin plot for all smokers vs charges
sns.violinplot(x='smoker', y='charges', data=data, ax=axes[0])
axes[0].set_title('Violin Plot: Smoker vs Charges', fontweight='bold')
axes[0].set_xlabel('Smoker', fontweight='semibold')
axes[0].set_ylabel('Charges', fontweight='semibold')

# Distribution plot of charges for non-smokers
non_smokers = data[data['smoker'] == 'no']
sns.histplot(non_smokers['charges'], kde=True, ax=axes[1])
axes[1].set_title('Distribution of Charges for Non-Smokers', fontweight='bold')
axes[1].set_xlabel('Charges', fontweight='semibold')
axes[1].set_ylabel('Frequency', fontweight='semibold')

# Distribution plot of charges for smokers
smokers = data[data['smoker'] == 'yes']
sns.histplot(smokers['charges'], kde=True, ax=axes[2])
axes[2].set_title('Distribution of Charges for Smokers', fontweight='bold')
axes[2].set_xlabel('Charges', fontweight='semibold')
axes[2].set_ylabel('Frequency', fontweight='semibold')

# Adjust spacing between subplots
plt.subplots_adjust(wspace=0.3)

# Display the plot
plt.show()

"""Smoking patients spend more on health rather than non smokers. Let's see if
charges depends on more variables or not."""

# Set up the figure and axis
plt.figure(figsize=(10, 6))
```

```
# Scatter plot with regression lines for smokers and non-smokers
g = sns.lmplot(x='age', y='charges', hue='smoker', data=data, legend=False)

# Set plot labels and title
g.fig.suptitle('Scatter Plot: Age vs Charges (Colored by Smoker)', fontweight='bold',
y=1.02)
g.set_axis_labels('Age', 'Charges', fontweight='semibold')

# Make the visibility of each dot 50%
for artist in g.ax.collections:
    artist.set_alpha(0.25)

# Add legend on the left
leg = g.ax.legend(title='Smoker', title_fontsize='medium', loc='upper left')
for lh in leg.legend_handles:
    lh.set_alpha(1)  # Make legend markers opaque

# Display the plot
plt.show()

# Define the age bins
age_bins = [18, 25, 35, 45, 55, 65]
data['age_group'] = pd.cut(data['age'], bins=age_bins)

# Group by age group and smoker status and calculate average charges
grouped_data = data.groupby(['age_group',
'smoker'])['charges'].mean().reset_index()

# Plot the results
plt.figure(figsize=(12, 8))
sns.barplot(x='age_group', y='charges', hue='smoker', data=grouped_data)
plt.title('Average Charges by Age Group and Smoking Status', fontweight='bold')
plt.xlabel('Age Group', fontweight='semibold')
plt.ylabel('Average Charges', fontweight='semibold')
```

```
plt.legend(title='Smoker')
plt.xticks(rotation=45)
plt.show()

# Define the age bins
age_bins = [18, 25, 35, 45, 55, 65]
age_labels = ['18-24', '25-34', '35-44', '45-54', '55-64']
data['age_group'] = pd.cut(data['age'], bins=age_bins, labels=age_labels,
right=False)

# Create a distribution plot for the age groups
plt.figure(figsize=(10, 6))
sns.histplot(data=data, x='age_group', discrete=True)
plt.title('Distribution of Age Groups', fontweight='bold')
plt.xlabel('Age Group', fontweight='semibold')
plt.ylabel('Density', fontweight='semibold')
plt.show()

# Set up the figure and axis
plt.figure(figsize=(10, 6))

# Scatter plot with regression lines for smokers and non-smokers
g = sns.lmplot(x='bmi', y='charges', hue='smoker', data=data, legend=False)

# Set plot labels and title
g.fig.suptitle('Scatter Plot: bmi vs Charges (Colored by Smoker)', fontweight='bold',
y=1.02)
g.set_axis_labels('bmi', 'Charges', fontweight='semibold')

# Make the visibility of each dot 50%
for artist in g.ax.collections:
    artist.set_alpha(0.25)

# Add legend on the left
```

```
leg = g.ax.legend(title='Smoker', title_fontsize='medium', loc='upper left')
for lh in leg.legend_handles:
    lh.set_alpha(1)  # Make legend markers opaque

# Display the plot
plt.show()

# Set up the figure and axis
plt.figure(figsize=(10, 6))

# Scatter plot with regression lines for smokers and non-smokers
g = sns.lmplot(x='bmi', y='log_charges', hue='smoker', data=data, legend=False)

# Set plot labels and title
g.fig.suptitle('Scatter Plot: bmi vs Log(Charges) (Colored by Smoker)',
fontweight='bold', y=1.02)
g.set_axis_labels('bmi', 'Log(Charges)', fontweight='semibold')

# Make the visibility of each dot 50%
for artist in g.ax.collections:
    artist.set_alpha(0.25)

# Add legend on the left
leg = g.ax.legend(title='Smoker', title_fontsize='medium', loc='upper left')
for lh in leg.legend_handles:
    lh.set_alpha(1)  # Make legend markers opaque

# Display the plot
plt.show()

# Set up the figure and axis
plt.figure(figsize=(10, 6))

# Scatter plot with regression lines for smokers and non-smokers
```

```
g = sns.lmplot(x='age', y='bmi', hue='smoker', data=data, legend=False)

# Set plot labels and title
g.fig.suptitle('Scatter Plot: Age vs bmi (Colored by Smoker)', fontweight='bold',
y=1.02)
g.set_axis_labels('age', 'bmi', fontweight='semibold')

# Make the visibility of each dot 50%
for artist in g.ax.collections:
    artist.set_alpha(0.25)

# Add legend on the left
leg = g.ax.legend(title='Smoker', title_fontsize='medium', loc='upper left')
for lh in leg.legend_handles:
    lh.set_alpha(1)  # Make legend markers opaque

# Display the plot
plt.show()

"""Interesting findings, if bmi increases of non-smoker, charges doesn't go much
higher but for smokers, it exponentially increases."""

# Convert categorical variables to numerical codes
data_num = data.copy()
data_num['sex'] = data_num['sex'].astype('category').cat.codes
data_num['smoker'] = data_num['smoker'].astype('category').cat.codes
data_num = data_num.drop(['log_charges', 'age_group'], axis=1)
# One-hot encode the 'region' column with 0/1 values
data_encoded = pd.get_dummies(data_num, columns=['region'], dtype='uint8')

# Display the first few rows of the resulting dataframe
data_encoded.head()

"""Feature Selection
```

```
VIF
"""

X = data_encoded.drop('charges', axis=1)

# Add a constant term to the independent variable matrix
X = sm.add_constant(X)

vif = pd.DataFrame()
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['feature'] = X.columns

# Sort the VIF values in ascending order
vif = vif.sort_values('VIF', ascending=True)

print(vif)

"""ANOVA (Analysis of Variance)"""

# Assuming 'data_encoded' is your DataFrame with independent variables
X = data_encoded.drop('charges', axis=1)
y = data_encoded['charges']

anova_results = {}

for feature in X.columns:
    groups = X[feature].unique()
    data_groups = [y[X[feature] == group] for group in groups]

    f_value, p_value = f_oneway(*data_groups)
    anova_results[feature] = p_value
```

```
anova_results = pd.DataFrame.from_dict(anova_results, orient='index',
columns=['p_value'])
anova_results = anova_results.sort_values(by='p_value', ascending=True)

print(anova_results)

"""RFECV (Recursive Feature Elimination with Cross-Validation):
RFECV is a method that recursively removes features and evaluates the model
performance using cross-validation.
"""

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

model = LinearRegression()
rfecv = RFECV(estimator=model, step=1, cv=5, scoring='neg_mean_squared_error')
rfecv.fit(X, y)

print("Optimal number of features: {}".format(rfecv.n_features_))
print("Selected features: {}".format(X.columns[rfecv.support_]))
# Plot number of features vs. cross-validation scores
plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross-validation score (MSE)")
plt.plot(range(1, len(rfecv.cv_results_['mean_test_score']) + 1),
rfecv.cv_results_['mean_test_score'])
plt.show()

"""**Random Forest Importance:**
Random Forest models can provide feature importance scores, which can be used
for feature selection.
"""

from sklearn.ensemble import RandomForestRegressor
```

```
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X, y)

importances = pd.DataFrame({'feature': X.columns, 'importance':
rf.feature_importances_})
importances = importances.sort_values(by='importance', ascending=False)

print(importances)

"""Pearson Correlation Coefficient:
The Pearson Correlation Coefficient measures the linear relationship between two
variables. You can calculate the correlation between each independent variable and
the target variable to identify the most relevant features.
"""

corr_coefficients = X.corrwith(y)
corr_coefficients = corr_coefficients.sort_values(ascending=False)

print(corr_coefficients)

from sklearn.cluster import KMeans

# Select the features for clustering
features = data[['age', 'charges']]

# Perform K-means clustering
kmeans = KMeans(n_clusters=3, random_state=42)
cluster_labels = kmeans.fit_predict(features)

# Add cluster labels to the original dataframe
data['cluster'] = cluster_labels

# Create separate dataframes for each cluster
```

```
cluster_0 = data[data['cluster'] == 0]
cluster_1 = data[data['cluster'] == 1]
cluster_2 = data[data['cluster'] == 2]

# Print the size of each cluster
print("Cluster 0 size:", len(cluster_0))
print("Cluster 1 size:", len(cluster_1))
print("Cluster 2 size:", len(cluster_2))

# Display the first few rows of each cluster
print("\nCluster 0:")
print(cluster_0.head())
print("\nCluster 1:")
print(cluster_1.head())
print("\nCluster 2:")
print(cluster_2.head())
data.head()

# Select the features for clustering
cl_features = data[['age', 'charges']]

# Perform K-means clustering
kmeans = KMeans(n_clusters=3, random_state=42)
cluster_labels = kmeans.fit_predict(cl_features)

# Create a new DataFrame with the original data and cluster labels
clustered_data = data_numeric.copy()
clustered_data['cluster'] = cluster_labels

# Plot the original scatter plot with clusters
plt.figure(figsize=(12, 8))
sns.scatterplot(x='age', y='charges', hue='cluster', data=clustered_data,
palette='viridis')
plt.title('Age vs Charges (KMeans Clustering)')
```

```
plt.xlabel('Age')
plt.ylabel('Charges')
plt.legend(title='Cluster')
plt.show()

# Create separate dataframes for each cluster
cluster_0 = data[data['cluster'] == 0]
cluster_1 = data[data['cluster'] == 1]
cluster_2 = data[data['cluster'] == 2]

# Plot separate graphs for each cluster
fig, axs = plt.subplots(1, 3, figsize=(18, 6))
fig.suptitle('Age vs Charges by Cluster (KMeans)', fontsize=16)

sns.scatterplot(x='age', y='charges', data=cluster_0, ax=axs[0], color='purple')
axs[0].set_title('Cluster 0')

sns.scatterplot(x='age', y='charges', data=cluster_1, ax=axs[1], color='green')
axs[1].set_title('Cluster 1')

sns.scatterplot(x='age', y='charges', data=cluster_2, ax=axs[2], color='orange')
axs[2].set_title('Cluster 2')

for ax in axs:
    ax.set_xlabel('Age')
    ax.set_ylabel('Charges')

plt.tight_layout()
plt.show()

# Print cluster sizes
for i in range(3):
    print(f"Cluster {i} size:", len(data[data['cluster'] == i]))
```

```
data.head()

# Summary statistics of each cluster
cluster_summary = clustered_data.groupby('cluster').mean()
print(cluster_summary)

# Plot distributions of other features for each cluster
fig, axs = plt.subplots(3, 2, figsize=(18, 18))
fig.suptitle('Feature Distributions by Cluster', fontsize=16)

sns.histplot(clustered_data[clustered_data['cluster'] == 0]['bmi'], ax=axs[0, 0],
color='purple', kde=True)
axs[0, 0].set_title('Cluster 0 - BMI')

sns.histplot(clustered_data[clustered_data['cluster'] == 1]['bmi'], ax=axs[0, 1],
color='green', kde=True)
axs[0, 1].set_title('Cluster 1 - BMI')

sns.histplot(clustered_data[clustered_data['cluster'] == 2]['bmi'], ax=axs[1, 0],
color='orange', kde=True)
axs[1, 0].set_title('Cluster 2 - BMI')

sns.histplot(clustered_data[clustered_data['cluster'] == 0]['children'], ax=axs[1, 1],
color='purple', kde=True)
axs[1, 1].set_title('Cluster 0 - Children')

sns.histplot(clustered_data[clustered_data['cluster'] == 1]['children'], ax=axs[2, 0],
color='green', kde=True)
axs[2, 0].set_title('Cluster 1 - Children')

sns.histplot(clustered_data[clustered_data['cluster'] == 2]['children'], ax=axs[2, 1],
color='orange', kde=True)
axs[2, 1].set_title('Cluster 2 - Children')
```

```
plt.tight_layout()
plt.show()

from sklearn.tree import DecisionTreeClassifier, plot_tree

# Select relevant features
features = ['age', 'charges', 'bmi', 'children', 'smoker', 'region']
X = data_numeric[features]
y = clustered_data['cluster']

# Train a decision tree classifier
clf = DecisionTreeClassifier(max_depth=3, random_state=0)
clf.fit(X, y)

# Plot the decision tree
plt.figure(figsize=(20, 10))
plot_tree(clf, feature_names=features, class_names=['Cluster 0', 'Cluster 1', 'Cluster
2'], filled=True)
plt.title('Decision Tree for Cluster Formation')
plt.show()

# Add cluster labels to the original dataframe
data['cluster'] = cluster_labels\

# Display the first few rows of each cluster
print("\nCluster 0:")
cluster_0.head()

print("\nCluster 1:")
cluster_1.head()

print("\nCluster 2:")
cluster_2.head()
```

```
# Function to create subplots for each cluster
def plot_cluster_subplots(cluster_data, cluster_num):
    fig, axs = plt.subplots(2, 3, figsize=(20, 15))
    fig.suptitle(f'Cluster {cluster_num} Relationships', fontsize=16)

    # Charges vs Age
    sns.scatterplot(x='age', y='charges', data=cluster_data, ax=axs[0, 0])
    axs[0, 0].set_title('Charges vs Age')

    # Charges vs BMI
    sns.scatterplot(x='bmi', y='charges', data=cluster_data, ax=axs[0, 1])
    axs[0, 1].set_title('Charges vs BMI')

    # Charges vs Sex
    sns.boxplot(x='sex', y='charges', data=cluster_data, ax=axs[0, 2])
    axs[0, 2].set_title('Charges vs Sex')

    # Charges vs Smoker
    sns.boxplot(x='smoker', y='charges', data=cluster_data, ax=axs[1, 0])
    axs[1, 0].set_title('Charges vs Smoker')

    # Charges vs Region
    sns.boxplot(x='region', y='charges', data=cluster_data, ax=axs[1, 1])
    axs[1, 1].set_title('Charges vs Region')

    # Charges vs Children
    sns.boxplot(x='children', y='charges', data=cluster_data, ax=axs[1, 2])
    axs[1, 2].set_title('Charges vs Children')

    plt.tight_layout()
    plt.show()

# Plot subplots for each cluster
for i in range(3):
```

```
    cluster_data = clustered_data[clustered_data['cluster'] == i]
    plot_cluster_subplots(cluster_data, i)

# Print cluster sizes
for i in range(3):
    print(f"Cluster {i} size:", len(clustered_data[clustered_data['cluster'] == i]))

"""Traditional Regression"""

data.head()

data_encoded.head()

data_numeric.head()

# Split the data
X = data_numeric[['age', 'bmi', 'sex', 'smoker', 'region', 'children']]
y = data_numeric['charges']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=42)

# Function to evaluate and print model performance
def evaluate_model(y_true, y_pred, model_name):
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_true, y_pred)
    print(f"{model_name}:")
    print(f"MSE: {mse:.2f}")
    print(f"RMSE: {rmse:.2f}")
    print(f"R2 Score: {r2:.4f}")
    print()

# Function to plot predicted vs actual values
def plot_predictions(y_true, y_pred, model_name):
```

```
    plt.figure(figsize=(10, 6))
    plt.scatter(y_true, y_pred, alpha=0.5, label='Predicted vs Actual')
    plt.plot([y_true.min(), y_true.max()], [y_true.min(), y_true.max()], '--r',
label='Prediction')
    plt.xlabel('Actual Charges')
    plt.ylabel('Predicted Charges')
    plt.title(f'{model_name} - Predicted vs Actual')
    plt.legend()  # Ensure legend is shown with labels
    plt.show()



# Create and fit the model
linear_reg = LinearRegression()
linear_reg.fit(X_train, y_train)

# Predict and evaluate
y_pred_train = linear_reg.predict(X_train)
y_pred_test = linear_reg.predict(X_test)

evaluate_model(y_train, y_pred_train, "Linear Regression (Train)")
evaluate_model(y_test, y_pred_test, "Linear Regression (Test)")

# Linear Regression
plot_predictions(y_test, linear_reg.predict(X_test), "Linear Regression")

# Calculate residuals
residuals_train_linear = y_train - y_pred_train
residuals_test_linear = y_test - y_pred_test

# Plot residuals
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
```

```
plt.scatter(y_pred_train, residuals_train_linear)
plt.axhline(y=0, color='r', linestyle='--')
plt.title('Residuals vs Predicted (Train) - Linear Regression')
plt.xlabel('Predicted values')
plt.ylabel('Residuals')

plt.subplot(1, 2, 2)
plt.scatter(y_pred_test, residuals_test_linear)
plt.axhline(y=0, color='r', linestyle='--')
plt.title('Residuals vs Predicted (Test) - Linear Regression')
plt.xlabel('Predicted values')
plt.ylabel('Residuals')

plt.show()

# Transform to polynomial features
poly = PolynomialFeatures(degree=2)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

# Create and fit the model
poly_reg = LinearRegression()
poly_reg.fit(X_train_poly, y_train)

# Predict and evaluate
y_pred_train = poly_reg.predict(X_train_poly)
y_pred_test = poly_reg.predict(X_test_poly)

evaluate_model(y_train, y_pred_train, "Polynomial Regression (Train)")
evaluate_model(y_test, y_pred_test, "Polynomial Regression (Test)")

plot_predictions(y_test, poly_reg.predict(poly.transform(X_test)), "Polynomial
Regression")
```

```
# Calculate residuals
residuals_train_poly = y_train - y_pred_train
residuals_test_poly = y_test - y_pred_test

# Plot residuals
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.scatter(y_pred_train, residuals_train_poly)
plt.axhline(y=0, color='r', linestyle='--')
plt.title('Residuals vs Predicted (Train) - Polynomial Regression')
plt.xlabel('Predicted values')
plt.ylabel('Residuals')

plt.subplot(1, 2, 2)
plt.scatter(y_pred_test, residuals_test_poly)
plt.axhline(y=0, color='r', linestyle='--')
plt.title('Residuals vs Predicted (Test) - Polynomial Regression')
plt.xlabel('Predicted values')
plt.ylabel('Residuals')

plt.show()

# Create and fit the model
ridge_reg = Ridge(alpha=1.0)
ridge_reg.fit(X_train, y_train)

# Predict and evaluate
y_pred_train = ridge_reg.predict(X_train)
y_pred_test = ridge_reg.predict(X_test)

evaluate_model(y_train, y_pred_train, "Ridge Regression (Train)")
evaluate_model(y_test, y_pred_test, "Ridge Regression (Test)")
```

```
plot_predictions(y_test, ridge_reg.predict(X_test), "Ridge Regression")

# Calculate residuals
residuals_train_ridge = y_train - y_pred_train
residuals_test_ridge = y_test - y_pred_test

# Plot residuals
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.scatter(y_pred_train, residuals_train_ridge)
plt.axhline(y=0, color='r', linestyle='--')
plt.title('Residuals vs Predicted (Train) - Ridge Regression')
plt.xlabel('Predicted values')
plt.ylabel('Residuals')

plt.subplot(1, 2, 2)
plt.scatter(y_pred_test, residuals_test_ridge)
plt.axhline(y=0, color='r', linestyle='--')
plt.title('Residuals vs Predicted (Test) - Ridge Regression')
plt.xlabel('Predicted values')
plt.ylabel('Residuals')

plt.show()

# Create and fit the model
lasso_reg = Lasso(alpha=1.0)
lasso_reg.fit(X_train, y_train)

# Predict and evaluate
y_pred_train = lasso_reg.predict(X_train)
y_pred_test = lasso_reg.predict(X_test)

evaluate_model(y_train, y_pred_train, "Lasso Regression (Train)")
```

```
evaluate_model(y_test, y_pred_test, "Lasso Regression (Test)")

plot_predictions(y_test, lasso_reg.predict(X_test), "Lasso Regression")

# Calculate residuals
residuals_train_lasso = y_train - y_pred_train
residuals_test_lasso = y_test - y_pred_test

# Plot residuals
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.scatter(y_pred_train, residuals_train_lasso)
plt.axhline(y=0, color='r', linestyle='--')
plt.title('Residuals vs Predicted (Train) - Lasso Regression')
plt.xlabel('Predicted values')
plt.ylabel('Residuals')

plt.subplot(1, 2, 2)
plt.scatter(y_pred_test, residuals_test_lasso)
plt.axhline(y=0, color='r', linestyle='--')
plt.title('Residuals vs Predicted (Test) - Lasso Regression')
plt.xlabel('Predicted values')
plt.ylabel('Residuals')

plt.show()

"""Polynomial Regression has the best performance among the models in terms of
MSE, RMSE, and R² scores for both training and testing data:

It has the lowest MSE and RMSE values, indicating it has the smallest errors.
It has the highest R² score, indicating it explains the most variance in the target
variable.
"""
```

```
# Function to plot predicted vs actual values
def plot_predictions(ax, y_true, y_pred, model_name):
    ax.scatter(y_true, y_pred, alpha=0.5, label='Actual')
    ax.plot([y_true.min(), y_true.max()], [y_true.min(), y_true.max()], '--r',
label='Predicted Trend')
    ax.set_xlabel('Actual Charges')
    ax.set_ylabel('Predicted Charges')
    ax.set_title(f'{model_name} - Predicted vs Actual')
    ax.legend()

# Create a figure and subplots
fig, axs = plt.subplots(1, 4, figsize=(20, 5))

# Plot for Linear Regression
plot_predictions(axs[0], y_test, linear_reg.predict(X_test), "Linear Regression")

# Plot for Polynomial Regressio
plot_predictions(axs[1], y_test, poly_reg.predict(poly.transform(X_test)), "Polynomial
Regression")

# Plot for Ridge Regression
plot_predictions(axs[2], y_test, ridge_reg.predict(X_test), "Ridge Regression")

# Plot for Lasso Regression
plot_predictions(axs[3], y_test, lasso_reg.predict(X_test), "Lasso Regression")

# Adjust layout
plt.tight_layout()
plt.show()

"""Using hyperparameter tuning"""
```

```
from sklearn.model_selection import GridSearchCV
# Define the pipeline with PolynomialFeatures and LinearRegression
poly_reg = Pipeline([
    ('scaler', StandardScaler()),  # Optional: Normalize your features if needed
    ('poly', PolynomialFeatures()),  # PolynomialFeatures with default settings
    ('linear_reg', LinearRegression())
])

# Define the parameter grid for GridSearchCV
param_grid = {
    'poly__degree': [2, 3, 4, 5],  # Degrees of polynomial features to try
    'poly__interaction_only': [True, False],  # Whether to include only interaction
features
}

# Instantiate GridSearchCV
grid_search = GridSearchCV(estimator=poly_reg, param_grid=param_grid,
scoring='neg_mean_squared_error', cv=5)

# Fit GridSearchCV
grid_search.fit(X_train, y_train)

# Get the best model
best_poly_reg = grid_search.best_estimator_

# Predict with best model
y_pred_train = best_poly_reg.predict(X_train)
y_pred_test = best_poly_reg.predict(X_test)

evaluate_model(y_train, y_pred_train, "Polynomial Regression (Train)")
evaluate_model(y_test, y_pred_test, "Polynomial Regression (Test)")

# Print the best parameters found by GridSearchCV
```

```
print("Best parameters found by GridSearchCV:")
print(grid_search.best_params_)

# Plotting actual vs predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_test, color='blue', alpha=0.5, label='Actual')
plt.plot(y_test, y_test, color='red', label='Prediction Trend')
plt.title('Actual vs Predicted')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()

# Define the pipeline with StandardScaler and Ridge regression
ridge_reg = Pipeline([
    ('scaler', StandardScaler()),
    ('ridge', Ridge())
])

# Define the parameter grid for GridSearchCV
param_grid = {
    'ridge__alpha': [0.001, 0.01, 0.1, 1.0, 10.0],  # Regularization strength
    'ridge__fit_intercept': [True, False],  # Whether to fit intercept
    'ridge__solver': ['auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga'],  # Solver
type
    'ridge__max_iter': [None, 100, 1000, 10000],  # Maximum number of iterations
}

# Instantiate GridSearchCV
grid_search = GridSearchCV(estimator=ridge_reg, param_grid=param_grid,
scoring='neg_mean_squared_error', cv=5)

# Fit GridSearchCV
grid_search.fit(X_train, y_train)
```

```
# Get the best model
best_ridge_reg = grid_search.best_estimator_

# Predict with best model
y_pred_train = best_ridge_reg.predict(X_train)
y_pred_test = best_ridge_reg.predict(X_test)

# Evaluate best model
def evaluate_model(y_true, y_pred, model_name):
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_true, y_pred)
    print(f"{model_name}:")
    print(f"MSE: {mse:.2f}")
    print(f"RMSE: {rmse:.2f}")
    print(f"R2 Score: {r2:.4f}")
    print()

evaluate_model(y_train, y_pred_train, "Ridge Regression (Train)")
evaluate_model(y_test, y_pred_test, "Ridge Regression (Test)")

# Plotting actual vs predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_test, color='blue', alpha=0.5, label='Actual vs Predicted')
plt.plot(y_test, y_test, color='red', label='Prediction Trend')
plt.title('Actual vs Predicted')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()

# Print the best parameters found by GridSearchCV
print("Best parameters found by GridSearchCV:")
```

```
print(grid_search.best_params_)

# Define the pipeline with StandardScaler and Lasso regression
lasso_reg = Pipeline([
    ('scaler', StandardScaler()),
    ('lasso', Lasso())
])

# Define the parameter grid for GridSearchCV
param_grid = {
    'lasso__alpha': [0.001, 0.01, 0.1, 1.0, 10.0],  # Regularization strength
    'lasso__fit_intercept': [True, False],  # Whether to fit intercept
    'lasso__max_iter': [1000, 2000, 3000],  # Maximum number of iterations
    'lasso__selection': ['cyclic', 'random'],  # Method used to select features
}

# Instantiate GridSearchCV
grid_search = GridSearchCV(estimator=lasso_reg, param_grid=param_grid,
scoring='neg_mean_squared_error', cv=5)

# Fit GridSearchCV
grid_search.fit(X_train, y_train)

# Get the best model
best_lasso_reg = grid_search.best_estimator_

# Predict with best model
y_pred_train = best_lasso_reg.predict(X_train)
y_pred_test = best_lasso_reg.predict(X_test)

evaluate_model(y_train, y_pred_train, "Lasso Regression (Train)")
evaluate_model(y_test, y_pred_test, "Lasso Regression (Test)")

# Plotting actual vs predicted values
```

```
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_test, color='blue', alpha=0.5, label='Actual vs Predicted')
plt.plot(y_test, y_test, color='red', label='Prediction Trend')
plt.title('Actual vs Predicted')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()

# Print the best parameters found by GridSearchCV
print("Best parameters found by GridSearchCV:")
print(grid_search.best_params_)

"""From the results, it appears that the metrics (MSE, RMSE, R2 Score) for both
training and test sets remain almost identical before and after using GridSearchCV.
This suggests that the hyperparameters chosen by GridSearchCV did not
significantly alter the model's performance metrics in this particular case.

### Quantile Regression
"""

# Separate predictors (X) and response variable (y)
# Split the data
X = data_numeric[['age', 'bmi', 'sex', 'smoker', 'region', 'children']]
y = data_numeric['charges']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=42)

# Add constant to X for the intercept term in the model
X_train = sm.add_constant(X_train)
X_test = sm.add_constant(X_test)

# Quantile levels to estimate
quantiles = [0.25, 0.50, 0.75]
```

```
max_iter = 10000  # Maximum number of iterations for optimization
# Fit quantile regression models
models = {}
for quantile in quantiles:
    model = sm.QuantReg(y_train, X_train).fit(q=quantile, max_iter=max_iter)
    models[quantile] = model

# Print summary of each model
for quantile, model in models.items():
    print(f"Quantile Regression Results for Quantile {quantile}:")
    print(model.summary())
    print()

# Predictions at different quantiles
quantile_predictions = {}
for quantile, model in models.items():
    y_pred = model.predict(X_test)
    quantile_predictions[quantile] = y_pred

# Plotting actual vs predicted values
plt.figure(figsize=(10, 6))

# Plot each quantile
for quantile, color in zip(quantiles, ['blue', 'green', 'red', 'blue', 'purple']):
    plt.scatter(y_test, quantile_predictions[quantile], color=color, alpha=0.5,
label=f'Quantile {quantile}')

plt.plot(y_test, y_test, color='gray', label='Prediction Trend')
plt.title('Quantile Regression - Actual vs Predicted')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()
```

"""

Quantile 0.25 (25th percentile):

Pseudo R-squared: 0.6019

Significant Predictors: Age, BMI, Sex, Smoker, Region, Children

Interpretation: Similar significant predictors as quantile 0.1, with slightly different coefficient values indicating their impact on charges at the 25th percentile of the distribution.

Quantile 0.5 (50th percentile - Median):

Pseudo R-squared: 0.5828

Significant Predictors: Age, BMI, Sex, Smoker, Region, Children

Interpretation: Median quantile shows that similar predictors impact charges, but with different coefficient magnitudes compared to lower quantiles, reflecting the middle of the charge distribution.

Quantile 0.75 (75th percentile):

Pseudo R-squared: 0.6465

Significant Predictors: Age, BMI, Sex, Smoker, Region, Children

Interpretation: At the 75th percentile, predictors continue to have significant effects on charges, with higher coefficients for some predictors compared to lower quantiles, indicating a higher charge distribution.

Quantile 0.9 (90th percentile):

Pseudo R-squared: 0.5600

Significant Predictors: Age, BMI, Smoker, Children (Sex and Region are not significant)

Interpretation: At the 90th percentile, age, BMI, smoking status, and number of children remain significant predictors, while sex and region do not significantly affect charges. The model suggests larger coefficients for BMI and smoker status at this higher quantile."""

```
# Separate predictors (X) and response variable (y)
# Split the data
X = data_numeric[['age', 'bmi', 'sex', 'smoker', 'region', 'children']]
y = data_numeric['charges']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=42)

# Add constant to X for the intercept term in the model
X_train = sm.add_constant(X_train)
X_test = sm.add_constant(X_test)

# Quantile levels to estimate
quantiles = [0.1, 0.2, 0.3, 0.4, 0.50, 0.6, 0.7, 0.8, 0.9]
max_iter = 10000  # Maximum number of iterations for optimization
# Fit quantile regression models
models = {}
for quantile in quantiles:
    model = sm.QuantReg(y_train, X_train).fit(q=quantile, max_iter=max_iter)
    models[quantile] = model

# Print summary of each model
for quantile, model in models.items():
    print(f"Quantile Regression Results for Quantile {quantile}:")
    print(model.summary())
    print()
```

```
# Predictions at different quantiles
quantile_predictions = {}
for quantile, model in models.items():
    y_pred = model.predict(X_test)
    quantile_predictions[quantile] = y_pred

# Plotting actual vs predicted values
plt.figure(figsize=(10, 6))

# Plot each quantile
for quantile, color in zip(quantiles, ['blue', 'green', 'orange', 'red', 'purple', 'cyan',
'magenta', 'yellow', 'brown', 'pink']):
    plt.scatter(y_test, quantile_predictions[quantile], color=color, alpha=0.5,
label=f'Quantile {quantile}')

plt.plot(y_test, y_test, color='gray', label='Prediction Trend')
plt.title('Quantile Regression - Actual vs Predicted')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()

"""The pseudo R-squared values range between approximately 0.56 to 0.64 across
quantiles, indicating that the models explain a substantial portion of the variability in
medical charges.

As quantiles increase (moving from lower to higher percentiles), the coefficients
generally increase in magnitude, indicating that the effects of age, BMI, smoking,
and number of children on medical charges become more pronounced at higher
charges.

Age and BMI: Older age and higher BMI consistently lead to higher medical charges.
```

Smoking: Smoking is a significant predictor of higher medical charges across all quantiles.

Sex and Region: The impact of sex and region on medical charges varies but generally shows smaller effects compared to age, BMI, and smoking.
Children: Having more children is associated with higher medical charges.
"""

```
from sklearn.tree import DecisionTreeClassifier, plot_tree

# Select features for clustering
features = ['age', 'charges']
X = data_numeric[features]

# Standardize the data
scaler = StandardScaler()
scaled_X = scaler.fit_transform(X)

# Apply K-means clustering
kmeans = KMeans(n_clusters=3, random_state=0)
data_numeric['cluster'] = kmeans.fit_predict(scaled_X)

# Plot age vs. charges colored by cluster
plt.figure(figsize=(10, 6))
sns.scatterplot(x='age', y='charges', hue='cluster', palette='Set1',
data=data_numeric)
plt.title('Age vs Charges (colored by Clusters)')
plt.xlabel('Age')
plt.ylabel('Charges')
plt.show()

# Train a Decision Tree Classifier
clf = DecisionTreeClassifier(max_depth=3, random_state=0)
clf.fit(X, data_numeric['cluster'])
```

```
# Create a mesh grid
x_min, x_max = X['age'].min() - 1, X['age'].max() + 1
y_min, y_max = X['charges'].min() - 1, X['charges'].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                     np.arange(y_min, y_max, 100))

# Predict clusters for each point in the mesh grid
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot the decision boundaries
plt.figure(figsize=(10, 6))
plt.contourf(xx, yy, Z, alpha=0.4, cmap='Set1')
sns.scatterplot(x='age', y='charges', hue='cluster', data=data_numeric,
palette='Set1', edgecolor='k')
plt.title('Decision Boundaries with Decision Tree')
plt.xlabel('Age')
plt.ylabel('Charges')
plt.show()

# Plot the decision tree
plt.figure(figsize=(20,10))
plot_tree(clf, feature_names=features, class_names=['Cluster 0', 'Cluster 1', 'Cluster
2'], filled=True)
plt.show()

"""well we have observed that there are 3 cluster, also we;ve tried to see if there are
another variables helped it to make these clusters

there might be missing variable, these would be in limitations

for regression:
r square method
```

check the residual after fitting the model
normality assumption
constant variable assumption

For variable selection:
Inference oberservation
cook distance

Refit the Model

If we decide to remove these points, we'll refit the model and compare the results to see if the model's performance improves.

Finding high leverage points and outliers
"""

```
target_column = 'charges'
X = data_numeric.drop(columns=[target_column])
y = data_numeric[target_column]

# Add a constant term for the intercept
X = sm.add_constant(X)

# Fit the model
model = sm.OLS(y, X).fit()

# Calculate influence measures
influence = model.get_influence()

# Leverage (hat values)
leverage = influence.hat_matrix_diag
```

```
# Standardized residuals
standardized_residuals = influence.resid_studentized_internal #if >3 then standard value, how we defined the outlier, is it absolute value or what

# Cook's distance
cooks_d = influence.cooks_distance[0]

# Identify high leverage points
high_leverage_threshold = 2 * (X.shape[1] / X.shape[0])  # Common threshold
high_leverage_points = np.where(leverage > high_leverage_threshold)[0]

# Identify outliers
outlier_threshold = 2  # Common threshold for standardized residuals
outliers = np.where(np.abs(standardized_residuals) > outlier_threshold)[0]

# Combine influential points, high leverage points, and outliers
combined_indices = np.unique(np.concatenate((high_leverage_points, outliers)))

# Identify points with high Cook's distance
cooks_d_threshold = 4 / len(X)
high_cooks_d_points = np.where(cooks_d > cooks_d_threshold)[0]

# Combine influential points that are also high leverage points or outliers above Cook's distance threshold
combined_indices = np.unique(np.concatenate((
    np.intersect1d(high_leverage_points, high_cooks_d_points),
    np.intersect1d(outliers, high_cooks_d_points)
)))

# Plot Cook's distance
plt.figure(figsize=(10, 6))
plt.stem(np.arange(len(cooks_d)), cooks_d, markerfmt=",", linefmt='b-', basefmt=' ', label='Cook\'s Distance')
```

```
# Highlight high leverage points
plt.scatter(high_leverage_points, cooks_d[high_leverage_points], color='red',
label='High Leverage Points', zorder=3)

# Highlight outliers
plt.scatter(outliers, cooks_d[outliers], color='orange', label='Outliers', zorder=3)

plt.title("Cook's Distance with High Leverage Points and Outliers")
plt.xlabel("Observation Index")
plt.ylabel("Cook's Distance")
plt.axhline(y=4 / len(X), color='r', linestyle='--', label='Threshold (4/n)')
plt.legend()
plt.show()

# Display influential points summary
influential_points_summary = pd.DataFrame({
    'Index': combined_indices,
    'Leverage': leverage[combined_indices],
    'Standardized Residuals': standardized_residuals[combined_indices],
    'Cook\'s Distance': cooks_d[combined_indices]
})

influential_points_summary

# Split the cleaned data into training and testing sets
X_train_wo, X_test_wo, y_train_wo, y_test_wo = train_test_split(X, y, test_size=0.2,
random_state=42)

# Fit the model again
linear_reg_wo = LinearRegression()
linear_reg_wo.fit(X_train_wo, y_train_wo)

# Predict and evaluate
y_pred_train_wo = linear_reg_wo.predict(X_train_wo)
```

```
y_pred_test_wo = linear_reg_wo.predict(X_test_wo)

# Define the evaluate_model function to calculate evaluation metrics
def evaluate_model(y_true, y_pred, model_name):
    from sklearn.metrics import mean_squared_error, r2_score
    mse = mean_squared_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    print(f'{model_name} - Mean Squared Error: {mse:.4f}, R2 Score: {r2:.4f}')

evaluate_model(y_train_wo, y_pred_train_wo, "Linear Regression (Train) - Original")
evaluate_model(y_test_wo, y_pred_test_wo, "Linear Regression (Test) - Original")

# Calculate residuals for the training set
residuals_train_wo = y_train_wo - y_pred_train_wo

# Calculate residuals for the test set
residuals_test_wo = y_test_wo - y_pred_test_wo

# Plot residuals for the training and test sets together
plt.figure(figsize=(10, 6))
plt.scatter(y_pred_train_wo, residuals_train_wo, color='blue', alpha=0.25,
label='Train Residuals')
plt.scatter(y_pred_test_wo, residuals_test_wo, color='red', alpha=0.25, label='Test
Residuals')
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Original Residuals vs Predicted Values (Training and Test Sets)')
plt.legend()
plt.show()

"""remving"""

# Remove high leverage points and outliers from the dataset
```

```
data_cleaned = data_numeric.drop(index=combined_indices)

# Split the cleaned data into features and target
X_cleaned = data_cleaned.drop(columns=[target_column])
y_cleaned = data_cleaned[target_column]

# Add a constant term for the intercept
X_cleaned = sm.add_constant(X_cleaned)

# Split the cleaned data into training and testing sets
X_train_cleaned, X_test_cleaned, y_train_cleaned, y_test_cleaned =
train_test_split(X_cleaned, y_cleaned, test_size=0.2, random_state=42)

# Fit the model again
linear_reg_cleaned = LinearRegression()
linear_reg_cleaned.fit(X_train_cleaned, y_train_cleaned)

# Predict and evaluate
y_pred_train_cleaned = linear_reg_cleaned.predict(X_train_cleaned)
y_pred_test_cleaned = linear_reg_cleaned.predict(X_test_cleaned)

# Define the evaluate_model function to calculate evaluation metrics
def evaluate_model(y_true, y_pred, model_name):
    from sklearn.metrics import mean_squared_error, r2_score
    mse = mean_squared_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    print(f'{model_name} - Mean Squared Error: {mse:.4f}, R2 Score: {r2:.4f}')

evaluate_model(y_train_cleaned, y_pred_train_cleaned, "Linear Regression (Train) -
Cleaned")
evaluate_model(y_test_cleaned, y_pred_test_cleaned, "Linear Regression (Test) -
Cleaned")

"""Calculating and visualize the residuals"""
```

```
# Calculate residuals for the training set
residuals_train_cleaned = y_train_cleaned - y_pred_train_cleaned

# Calculate residuals for the test set
residuals_test_cleaned = y_test_cleaned - y_pred_test_cleaned

# Plot residuals for the training and test sets together
plt.figure(figsize=(10, 6))
plt.scatter(y_pred_train_cleaned, residuals_train_cleaned, color='blue', alpha=0.25,
label='Train Residuals')
plt.scatter(y_pred_test_cleaned, residuals_test_cleaned, color='red', alpha=0.25,
label='Test Residuals')
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Cleaned Residuals vs Predicted Values (Training and Test Sets)')
plt.legend()
plt.show()

"""XGBoost (Extreme Gradient Boosting)"""

# Split the Original data into training and testing sets
X_train_wo, X_test_wo, y_train_wo, y_test_wo = train_test_split(X, y, test_size=0.2,
random_state=42)

# Initialize the XGBRegressor
xgb_reg = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=100,
learning_rate=0.1, max_depth=3, random_state=42)

# Fit the model
xgb_reg.fit(X_train, y_train)

# Predict
```

```
y_pred_train = xgb_reg.predict(X_train)
y_pred_test = xgb_reg.predict(X_test)

def evaluate_model(y_true, y_pred, model_name):
    mse = mean_squared_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    print(f'{model_name} - Mean Squared Error: {mse:.4f}, R2 Score: {r2:.4f}')

evaluate_model(y_train, y_pred_train, "XGBoost Regression (Train)")
evaluate_model(y_test, y_pred_test, "XGBoost Regression (Test)")

# Calculate residuals for the training set
residuals_train = y_train - y_pred_train

# Calculate residuals for the test set
residuals_test = y_test - y_pred_test

# Plot residuals for the training and test sets together
plt.figure(figsize=(10, 6))
plt.scatter(y_pred_train, residuals_train, color='blue', alpha=0.25, label='Train
Residuals')
plt.scatter(y_pred_test, residuals_test, color='red', alpha=0.25, label='Test
Residuals')
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Original data XGBoost (Residuals vs Predicted Values)')
plt.legend()
plt.show()

# Initialize the XGBRegressor
xgb_reg2 = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=100,
learning_rate=0.1, max_depth=3, random_state=42)
```

```
# Fit the model
xgb_reg2.fit(X_train_cleaned, y_train_cleaned)

# Predict
y_pred_train_cleaned_xg = xgb_reg2.predict(X_train_cleaned)
y_pred_test_cleaned_xg = xgb_reg2.predict(X_test_cleaned)

def evaluate_model(y_true, y_pred, model_name):
    mse = mean_squared_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    print(f'{model_name} - Mean Squared Error: {mse:.4f}, R2 Score: {r2:.4f}')

evaluate_model(y_train_cleaned, y_pred_train_cleaned_xg, "XGBoost Regression
(Train)")
evaluate_model(y_test_cleaned, y_pred_test_cleaned_xg, "XGBoost Regression
(Test)")

# Calculate residuals for the training set
residuals_train_xg = y_train_cleaned - y_pred_train_cleaned_xg

# Calculate residuals for the test set
residuals_test_xg = y_test_cleaned - y_pred_test_cleaned_xg

# Plot residuals for the training and test sets together
plt.figure(figsize=(10, 6))
plt.scatter(y_pred_train_cleaned_xg, residuals_train_xg, color='blue', alpha=0.25,
label='Train Residuals')
plt.scatter(y_pred_test_cleaned_xg, residuals_test_xg, color='red', alpha=0.25,
label='Test Residuals')
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Original data XGBoost (Residuals vs Predicted Values)')
plt.legend()
```

```
plt.show()

def quantile_regression(q, X_train, y_train, X_test, y_test, max_iter=10000):
    # Fit the model with increased maximum iterations
    mod = sm.QuantReg(y_train, X_train)
    res = mod.fit(q=q, max_iter=max_iter)

    # Predict
    y_pred_train = res.predict(X_train)
    y_pred_test = res.predict(X_test)

    # Evaluate
    mse_train = mean_squared_error(y_train, y_pred_train)
    mse_test = mean_squared_error(y_test, y_pred_test)
    r2_train = r2_score(y_train, y_pred_train)
    r2_test = r2_score(y_test, y_pred_test)

    print(f'Quantile {q} - Train MSE: {mse_train:.4f}, R2: {r2_train:.4f}')
    print(f'Quantile {q} - Test MSE: {mse_test:.4f}, R2: {r2_test:.4f}')

    return res, y_pred_train, y_pred_test

quantiles = [0.25, 0.5, 0.75]
results = {}

for q in quantiles:
    res, y_pred_train_quant, y_pred_test_quant = quantile_regression(q, X_train_wo,
y_train_wo, X_test_wo, y_test_wo)
    results[q] = (res, y_pred_train_quant, y_pred_test_quant)

plt.figure(figsize=(15, 10))

for i, q in enumerate(quantiles):
    res, y_pred_train_quant, y_pred_test_quant = results[q]
```

```
    # Calculate residuals for the training set
    residuals_train_quant = y_train_wo - y_pred_train_quant

    # Calculate residuals for the test set
    residuals_test_quant = y_test_wo - y_pred_test_quant

    # Plot residuals for the training and test sets together
    plt.subplot(2, 2, i+1)
    plt.scatter(y_pred_train_quant, residuals_train_quant, color='blue', alpha=0.25,
label='Train Residuals')
    plt.scatter(y_pred_test_quant, residuals_test_quant, color='red', alpha=0.25,
label='Test Residuals')
    plt.axhline(y=0, color='red', linestyle='--')
    plt.xlabel('Predicted Values')
    plt.ylabel('Residuals')
    plt.title(f'Residuals vs Predicted Values (Quantile {q})')
    plt.legend()

plt.tight_layout()
plt.show()

# Plot residuals for each quantile in separate subplots for training and testing data
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

colors = ['blue', 'green', 'red']
labels = ['Quantile 0.25', 'Quantile 0.5', 'Quantile 0.75']

for i, q in enumerate(quantiles):
    res, y_pred_train_quant, y_pred_test_quant = results[q]

    # Calculate residuals for the training set
    residuals_train_quant = y_train_wo - y_pred_train_quant
```

```
    # Calculate residuals for the test set
    residuals_test_quant = y_test_wo - y_pred_test_quant

    # Plot residuals for the training set
    ax1.scatter(y_pred_train_quant, residuals_train_quant, color=colors[i],
alpha=0.25, label=f'Train Residuals ({labels[i]})')

    # Plot residuals for the test set
    ax2.scatter(y_pred_test_quant, residuals_test_quant, color=colors[i], alpha=0.25,
label=f'Test Residuals ({labels[i]})', marker='x')

# Customize the training residuals plot
ax1.axhline(y=0, color='black', linestyle='--')
ax1.set_xlabel('Predicted Values')
ax1.set_ylabel('Residuals')
ax1.set_title('Training Residuals vs Predicted Values for Different Quantiles')
ax1.legend()

# Customize the testing residuals plot
ax2.axhline(y=0, color='black', linestyle='--')
ax2.set_xlabel('Predicted Values')
ax2.set_ylabel('Residuals')
ax2.set_title('Testing Residuals vs Predicted Values for Different Quantiles')
ax2.legend()

plt.tight_layout()
plt.show()

"""For cleaned data"""

quantiles = [0.25, 0.5, 0.75]
results = {}

for q in quantiles:
```

```
    res, y_pred_train_cq, y_pred_test_cq = quantile_regression(q, X_train_cleaned,
y_train_cleaned, X_test_cleaned, y_test_cleaned)
    results[q] = (res, y_pred_train_cq, y_pred_test_cq)

plt.figure(figsize=(15, 10))

for i, q in enumerate(quantiles):
    res, y_pred_train_cq, y_pred_test_cq = results[q]

    # Calculate residuals for the training set
    residuals_train_cq = y_train_cleaned - y_pred_train_cq

    # Calculate residuals for the test set
    residuals_test_cq = y_test_cleaned - y_pred_test_cq

    # Plot residuals for the training and test sets together
    plt.subplot(2, 2, i+1)
    plt.scatter(y_pred_train_cq, residuals_train_cq, color='blue', alpha=0.25,
label='Train Residuals')
    plt.scatter(y_pred_test_cq, residuals_test_cq, color='red', alpha=0.25, label='Test
Residuals')
    plt.axhline(y=0, color='red', linestyle='--')
    plt.xlabel('Predicted Values')
    plt.ylabel('Residuals')
    plt.title(f'Cleaned Residuals vs Predicted Values (Quantile {q})')
    plt.legend()

plt.tight_layout()
plt.show()



# Plot residuals for each quantile in separate subplots for training and testing data
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))
```

```
colors = ['blue', 'green', 'red']
labels = ['Quantile 0.25', 'Quantile 0.5', 'Quantile 0.75']

for i, q in enumerate(quantiles):
    res, y_pred_train_cq, y_pred_test_cq = results[q]

    # Calculate residuals for the training set
    residuals_train_cq = y_train_cleaned - y_pred_train_cq

    # Calculate residuals for the test set
    residuals_test_cq= y_test_cleaned - y_pred_test_cq

    # Plot residuals for the training set
    ax1.scatter(y_pred_train_cq, residuals_train_cq, color=colors[i], alpha=0.25,
label=f'Train Residuals ({labels[i]})')

    # Plot residuals for the test set
    ax2.scatter(y_pred_test_cq, residuals_test_cq, color=colors[i], alpha=0.25,
label=f'Test Residuals ({labels[i]})', marker='x')

# Customize the training residuals plot
ax1.axhline(y=0, color='black', linestyle='--')
ax1.set_xlabel('Predicted Values')
ax1.set_ylabel('Residuals')
ax1.set_title('Training Residuals vs Predicted Values for Different Quantiles')
ax1.legend()

# Customize the testing residuals plot
ax2.axhline(y=0, color='black', linestyle='--')
ax2.set_xlabel('Predicted Values')
ax2.set_ylabel('Residuals')
ax2.set_title('Testing Residuals vs Predicted Values for Different Quantiles')
ax2.legend()
```

```
plt.tight_layout()
plt.show()

"""Using XGBoost (Extreme Gradient Boosting)"""

# Define a function to fit and predict using XGBoost for quantile regression
def xgboost_quantile_regression(q, X_train, y_train, X_test, y_test):
    # Custom objective function for quantile regression
    def quantile_obj(y_true, y_pred):
        e = y_true - y_pred
        grad = np.where(e > 0, -q, 1 - q)
        hess = np.ones_like(y_true)
        return grad, hess

    dtrain = xgb.DMatrix(X_train, label=y_train)
    dtest = xgb.DMatrix(X_test, label=y_test)

    params = {
        'eta': 0.1,
        'max_depth': 3,
        'objective': quantile_obj
    }

    num_boost_round = 100
    model = xgb.train(params, dtrain, num_boost_round)

    y_pred_train = model.predict(dtrain)
    y_pred_test = model.predict(dtest)

    # Evaluate
    mse_train = mean_squared_error(y_train, y_pred_train)
    mse_test = mean_squared_error(y_test, y_pred_test)
    r2_train = r2_score(y_train, y_pred_train)
```

```
    r2_test = r2_score(y_test, y_pred_test)

    print(f'Quantile {q} - Train MSE: {mse_train:.4f}, R2: {r2_train:.4f}')
    print(f'Quantile {q} - Test MSE: {mse_test:.4f}, R2: {r2_test:.4f}')

    return model, y_pred_train, y_pred_test

# Define a function for the quantile loss
def quantile_loss(q, y_true, y_pred):
    e = y_true - y_pred
    return np.mean(np.maximum(q * e, (q - 1) * e))

# Define a function to fit and predict using XGBoost for quantile regression
def xgboost_quantile_regression(q, X_train, y_train, X_test, y_test):
    dtrain = xgb.DMatrix(X_train, label=y_train)
    dtest = xgb.DMatrix(X_test, label=y_test)

    params = {
        'eta': 0.1,
        'max_depth': 3,
        'objective': 'reg:absoluteerror',
        'alpha': q
    }

    num_boost_round = 100
    model = xgb.train(params, dtrain, num_boost_round)

    y_pred_train = model.predict(dtrain)
    y_pred_test = model.predict(dtest)

    # Evaluate
    mse_train = mean_squared_error(y_train, y_pred_train)
    mse_test = mean_squared_error(y_test, y_pred_test)
    r2_train = r2_score(y_train, y_pred_train)
```

```
    r2_test = r2_score(y_test, y_pred_test)

    print(f'Quantile {q} - Train MSE: {mse_train:.4f}, R2: {r2_train:.4f}')
    print(f'Quantile {q} - Test MSE: {mse_test:.4f}, R2: {r2_test:.4f}')

    return model, y_pred_train, y_pred_test

# Perform XGBoost quantile regression for 0.25, 0.5, and 0.75 quantiles
quantiles = [0.25, 0.5, 0.75]
results = {}

for q in quantiles:
    model, y_pred_train_quant, y_pred_test_quant = xgboost_quantile_regression(q,
X_train_wo, y_train_wo, X_test_wo, y_test_wo)
    results[q] = (model, y_pred_train_quant, y_pred_test_quant)

plt.figure(figsize=(15, 10))

for i, q in enumerate(quantiles):
    res, y_pred_train_qxg, y_pred_test_qxg = results[q]

    # Calculate residuals for the training set
    residuals_train_qxg = y_train_wo - y_pred_train_qxg

    # Calculate residuals for the test set
    residuals_test_qxg = y_test_wo - y_pred_test_qxg

    # Plot residuals for the training and test sets together
    plt.subplot(2, 2, i+1)
    plt.scatter(y_pred_train_qxg, residuals_train_qxg, color='blue', alpha=0.25,
label='Train Residuals')
    plt.scatter(y_pred_test_qxg, residuals_test_qxg, color='red', alpha=0.25,
label='Test Residuals')
    plt.axhline(y=0, color='red', linestyle='--')
```

```
    plt.xlabel('Predicted Values')
    plt.ylabel('Residuals')
    plt.title(f'Residuals vs Predicted Values (Quantile {q})')
    plt.legend()

plt.tight_layout()
plt.show()

# Plot residuals for each quantile in separate subplots for training and testing data
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

colors = ['blue', 'green', 'red']
labels = ['Quantile 0.25', 'Quantile 0.5', 'Quantile 0.75']

for i, q in enumerate(quantiles):
    res, y_pred_train_qxg, y_pred_test_qxg = results[q]

    # Calculate residuals for the training set
    residuals_train_qxg = y_train_wo - y_pred_train_qxg

    # Calculate residuals for the test set
    residuals_test_qxg = y_test_wo - y_pred_test_qxg

    # Plot residuals for the training set
    ax1.scatter(y_pred_train_qxg, residuals_train_qxg, color=colors[i], alpha=0.25,
label=f'Train Residuals ({labels[i]})')

    # Plot residuals for the test set
    ax2.scatter(y_pred_test_qxg, residuals_test_qxg, color=colors[i], alpha=0.25,
label=f'Test Residuals ({labels[i]})', marker='x')

# Customize the training residuals plot
ax1.axhline(y=0, color='black', linestyle='--')
ax1.set_xlabel('Predicted Values')
```

```
ax1.set_ylabel('Residuals')
ax1.set_title('Training Residuals vs Predicted Values for Different Quantiles')
ax1.legend()

# Customize the testing residuals plot
ax2.axhline(y=0, color='black', linestyle='--')
ax2.set_xlabel('Predicted Values')
ax2.set_ylabel('Residuals')
ax2.set_title('Testing Residuals vs Predicted Values for Different Quantiles')
ax2.legend()

plt.tight_layout()
plt.show()

"""On cleaned data"""

# Perform XGBoost quantile regression for 0.25, 0.5, and 0.75 quantiles
quantiles = [0.25, 0.5, 0.75]
results = {}

for q in quantiles:
    model, y_pred_train_qxgc, y_pred_test_qxgc = xgboost_quantile_regression(q,
X_train_cleaned, y_train_cleaned, X_test_cleaned, y_test_cleaned)
    results[q] = (model, y_pred_train_qxgc, y_pred_test_qxgc)

plt.figure(figsize=(15, 10))

for i, q in enumerate(quantiles):
    res, y_pred_train_qxgc, y_pred_test_qxgc = results[q]

    # Calculate residuals for the training set
    residuals_train_qxgc = y_train_cleaned - y_pred_train_qxgc

    # Calculate residuals for the test set
```

```
    residuals_test_qxgc = y_test_cleaned - y_pred_test_qxgc

    # Plot residuals for the training and test sets together
    plt.subplot(2, 2, i+1)
    plt.scatter(y_pred_train_qxgc, residuals_train_qxgc, color='blue', alpha=0.25,
label='Train Residuals')
    plt.scatter(y_pred_test_qxgc, residuals_test_qxgc, color='red', alpha=0.25,
label='Test Residuals')
    plt.axhline(y=0, color='red', linestyle='--')
    plt.xlabel('Predicted Values')
    plt.ylabel('Residuals')
    plt.title(f'Residuals vs Predicted Values (Quantile {q})')
    plt.legend()

plt.tight_layout()
plt.show()

# Plot residuals for each quantile in separate subplots for training and testing data
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

colors = ['blue', 'green', 'red']
labels = ['Quantile 0.25', 'Quantile 0.5', 'Quantile 0.75']

for i, q in enumerate(quantiles):
    res, y_pred_train_qxgc, y_pred_test_qxgc = results[q]

    # Calculate residuals for the training set
    residuals_train_qxgc = y_train_cleaned - y_pred_train_qxgc

    # Calculate residuals for the test set
    residuals_test_qxgc = y_test_cleaned - y_pred_test_qxgc

    # Plot residuals for the training set
```

```
    ax1.scatter(y_pred_train_qxgc, residuals_train_qxgc, color=colors[i], alpha=0.25, label=f'Train Residuals ({labels[i]})')

    # Plot residuals for the test set
    ax2.scatter(y_pred_test_qxgc, residuals_test_qxgc, color=colors[i], alpha=0.25, label=f'Test Residuals ({labels[i]})', marker='x')

# Customize the training residuals plot
ax1.axhline(y=0, color='black', linestyle='--')
ax1.set_xlabel('Predicted Values')
ax1.set_ylabel('Residuals')
ax1.set_title('Training Residuals vs Predicted Values for Different Quantiles')
ax1.legend()

# Customize the testing residuals plot
ax2.axhline(y=0, color='black', linestyle='--')
ax2.set_xlabel('Predicted Values')
ax2.set_ylabel('Residuals')
ax2.set_title('Testing Residuals vs Predicted Values for Different Quantiles')
ax2.legend()

plt.tight_layout()
plt.show()

"""After Feature selection
Most significant features are:

1. Smoker

2. BMI

3. Age

Lets select only these feature and find outliers and high leverege points and again
```

"""

```
target_column = 'charges'
selected_features = ['smoker', 'bmi', 'age']
X = data_numeric[selected_features]
y = data_numeric[target_column]

# Add a constant term for the intercept
X = sm.add_constant(X)

# Fit the model
model = sm.OLS(y, X).fit()

# Calculate influence measures
influence = model.get_influence()

# Leverage (hat values)
leverage = influence.hat_matrix_diag

# Standardized residuals
standardized_residuals = influence.resid_studentized_internal

# Cook's distance
cooks_d = influence.cooks_distance[0]

# Identify high leverage points
high_leverage_threshold = 2 * (X.shape[1] / X.shape[0])  # Common threshold
high_leverage_points = np.where(leverage > high_leverage_threshold)[0]

# Identify outliers
outlier_threshold = 2  # Common threshold for standardized residuals
outliers = np.where(np.abs(standardized_residuals) > outlier_threshold)[0]

# Combine influential points, high leverage points, and outliers
```

```
combined_indices = np.unique(np.concatenate((high_leverage_points, outliers)))

# Plot Cook's distance
plt.figure(figsize=(10, 6))
plt.stem(np.arange(len(cooks_d)), cooks_d, markerfmt=",", linefmt='b-', basefmt=' ',
label='Cook\'s Distance')

# Highlight high leverage points
plt.scatter(high_leverage_points, cooks_d[high_leverage_points], color='red',
label='High Leverage Points', zorder=3)

# Highlight outliers
plt.scatter(outliers, cooks_d[outliers], color='orange', label='Outliers', zorder=3)

plt.title("Cook's Distance with High Leverage Points and Outliers")
plt.xlabel("Observation Index")
plt.ylabel("Cook's Distance")
plt.axhline(y=4 / len(X), color='r', linestyle='--', label='Threshold (4/n)')
plt.legend()
plt.show()

# Display influential points summary
influential_points_summary = pd.DataFrame({
    'Index': combined_indices,
    'Leverage': leverage[combined_indices],
    'Standardized Residuals': standardized_residuals[combined_indices],
    'Cook\'s Distance': cooks_d[combined_indices]
})

influential_points_summary

X

y
```

```
"""Fitting linear regression model without removing these outliers and high levereage poings"""

# Split the cleaned data into training and testing sets
X_train_wo, X_test_wo, y_train_wo, y_test_wo = train_test_split(X, y, test_size=0.2, random_state=42)

# Fit the model again
linear_reg_wo = LinearRegression()
linear_reg_wo.fit(X_train_wo, y_train_wo)

# Predict and evaluate
y_pred_train_wo = linear_reg_wo.predict(X_train_wo)
y_pred_test_wo = linear_reg_wo.predict(X_test_wo)

# Define the evaluate_model function to calculate evaluation metrics
def evaluate_model(y_true, y_pred, model_name):
    from sklearn.metrics import mean_squared_error, r2_score
    mse = mean_squared_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    print(f'{model_name} - Mean Squared Error: {mse:.4f}, R2 Score: {r2:.4f}')

evaluate_model(y_train_wo, y_pred_train_wo, "Linear Regression (Train) - Original")
evaluate_model(y_test_wo, y_pred_test_wo, "Linear Regression (Test) - Original")

# Calculate residuals for the training set
residuals_train_wo = y_train_wo - y_pred_train_wo

# Calculate residuals for the test set
residuals_test_wo = y_test_wo - y_pred_test_wo

# Plot residuals for the training and test sets together
plt.figure(figsize=(10, 6))
```

```
plt.scatter(y_pred_train_wo, residuals_train_wo, color='blue', alpha=0.25,
label='Train Residuals')
plt.scatter(y_pred_test_wo, residuals_test_wo, color='red', alpha=0.25, label='Test
Residuals')
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Original Residuals vs Predicted Values (Training and Test Sets)')
plt.legend()
plt.show()



"""After removing high leverage points and outliers"""

# Remove high leverage points and outliers from the dataset
data_cleaned = data_numeric.drop(index=combined_indices)

# Split the cleaned data into features and target
X_cleaned = data_cleaned[selected_features]
y_cleaned = data_cleaned[target_column]

# Add a constant term for the intercept
X_cleaned = sm.add_constant(X_cleaned)

# Split the cleaned data into training and testing sets
X_train_cleaned, X_test_cleaned, y_train_cleaned, y_test_cleaned =
train_test_split(X_cleaned, y_cleaned, test_size=0.2, random_state=42)

# Fit the model again
linear_reg_cleaned = LinearRegression()
linear_reg_cleaned.fit(X_train_cleaned, y_train_cleaned)

# Predict and evaluate
```

```
y_pred_train_cleaned = linear_reg_cleaned.predict(X_train_cleaned)
y_pred_test_cleaned = linear_reg_cleaned.predict(X_test_cleaned)

# Define the evaluate_model function to calculate evaluation metrics
def evaluate_model(y_true, y_pred, model_name):
    from sklearn.metrics import mean_squared_error, r2_score
    mse = mean_squared_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    print(f'{model_name} - Mean Squared Error: {mse:.4f}, R2 Score: {r2:.4f}')

evaluate_model(y_train_cleaned, y_pred_train_cleaned, "Linear Regression (Train) - Cleaned")
evaluate_model(y_test_cleaned, y_pred_test_cleaned, "Linear Regression (Test) - Cleaned")

# Calculate residuals for the training set
residuals_train_cleaned = y_train_cleaned - y_pred_train_cleaned

# Calculate residuals for the test set
residuals_test_cleaned = y_test_cleaned - y_pred_test_cleaned

# Plot residuals for the training and test sets together
plt.figure(figsize=(10, 6))
plt.scatter(y_pred_train_cleaned, residuals_train_cleaned, color='blue', alpha=0.25, label='Train Residuals')
plt.scatter(y_pred_test_cleaned, residuals_test_cleaned, color='red', alpha=0.25, label='Test Residuals')
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Cleaned Residuals vs Predicted Values (Training and Test Sets)')
plt.legend()
plt.show()
```

```
# Split the Original data into training and testing sets
X_train_wo, X_test_wo, y_train_wo, y_test_wo = train_test_split(X, y, test_size=0.2,
random_state=42)

# Initialize the XGBRegressor
xgb_reg = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=100,
learning_rate=0.1, max_depth=3, random_state=42)

# Fit the model
xgb_reg.fit(X_train, y_train)

# Predict
y_pred_train = xgb_reg.predict(X_train)
y_pred_test = xgb_reg.predict(X_test)

def evaluate_model(y_true, y_pred, model_name):
    mse = mean_squared_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    print(f'{model_name} - Mean Squared Error: {mse:.4f}, R2 Score: {r2:.4f}')

evaluate_model(y_train, y_pred_train, "XGBoost Regression (Train)")
evaluate_model(y_test, y_pred_test, "XGBoost Regression (Test)")

# Calculate residuals for the training set
residuals_train = y_train - y_pred_train

# Calculate residuals for the test set
residuals_test = y_test - y_pred_test

# Plot residuals for the training and test sets together
plt.figure(figsize=(10, 6))
plt.scatter(y_pred_train, residuals_train, color='blue', alpha=0.25, label='Train
Residuals')
```

```
plt.scatter(y_pred_test, residuals_test, color='red', alpha=0.25, label='Test
Residuals')
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Original data XGBoost (Residuals vs Predicted Values)')
plt.legend()
plt.show()

# Initialize the XGBRegressor
xgb_reg2 = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=100,
learning_rate=0.1, max_depth=3, random_state=42)

# Fit the model
xgb_reg2.fit(X_train_cleaned, y_train_cleaned)

# Predict
y_pred_train_cleaned_xg = xgb_reg2.predict(X_train_cleaned)
y_pred_test_cleaned_xg = xgb_reg2.predict(X_test_cleaned)

def evaluate_model(y_true, y_pred, model_name):
    mse = mean_squared_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    print(f'{model_name} - Mean Squared Error: {mse:.4f}, R2 Score: {r2:.4f}')

evaluate_model(y_train_cleaned, y_pred_train_cleaned_xg, "XGBoost Regression
(Train)")
evaluate_model(y_test_cleaned, y_pred_test_cleaned_xg, "XGBoost Regression
(Test)")

# Calculate residuals for the training set
residuals_train_xg = y_train_cleaned - y_pred_train_cleaned_xg

# Calculate residuals for the test set
```

```
residuals_test_xg = y_test_cleaned - y_pred_test_cleaned_xg

# Plot residuals for the training and test sets together
plt.figure(figsize=(10, 6))
plt.scatter(y_pred_train_cleaned_xg, residuals_train_xg, color='blue', alpha=0.25,
label='Train Residuals')
plt.scatter(y_pred_test_cleaned_xg, residuals_test_xg, color='red', alpha=0.25,
label='Test Residuals')
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Original data XGBoost (Residuals vs Predicted Values)')
plt.legend()
plt.show()

"""Quantile regression after feature selection"""

quantiles = [0.25, 0.5, 0.75]
results = {}

for q in quantiles:
    res, y_pred_train_quant, y_pred_test_quant = quantile_regression(q, X_train_wo,
y_train_wo, X_test_wo, y_test_wo)
    results[q] = (res, y_pred_train_quant, y_pred_test_quant)

plt.figure(figsize=(15, 10))

for i, q in enumerate(quantiles):
    res, y_pred_train_quant, y_pred_test_quant = results[q]

    # Calculate residuals for the training set
    residuals_train_quant = y_train_wo - y_pred_train_quant

    # Calculate residuals for the test set
```

```
    residuals_test_quant = y_test_wo - y_pred_test_quant

    # Plot residuals for the training and test sets together
    plt.subplot(2, 2, i+1)
    plt.scatter(y_pred_train_quant, residuals_train_quant, color='blue', alpha=0.25,
label='Train Residuals')
    plt.scatter(y_pred_test_quant, residuals_test_quant, color='red', alpha=0.25,
label='Test Residuals')
    plt.axhline(y=0, color='red', linestyle='--')
    plt.xlabel('Predicted Values')
    plt.ylabel('Residuals')
    plt.title(f'Residuals vs Predicted Values (Quantile {q})')
    plt.legend()

plt.tight_layout()
plt.show()

# Plot residuals for each quantile in separate subplots for training and testing data
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

colors = ['blue', 'green', 'red']
labels = ['Quantile 0.25', 'Quantile 0.5', 'Quantile 0.75']

for i, q in enumerate(quantiles):
    res, y_pred_train_quant, y_pred_test_quant = results[q]

    # Calculate residuals for the training set
    residuals_train_quant = y_train_wo - y_pred_train_quant

    # Calculate residuals for the test set
    residuals_test_quant = y_test_wo - y_pred_test_quant

    # Plot residuals for the training set
```

```
    ax1.scatter(y_pred_train_quant, residuals_train_quant, color=colors[i],
alpha=0.25, label=f'Train Residuals ({labels[i]})')

    # Plot residuals for the test set
    ax2.scatter(y_pred_test_quant, residuals_test_quant, color=colors[i], alpha=0.25,
label=f'Test Residuals ({labels[i]})', marker='x')

# Customize the training residuals plot
ax1.axhline(y=0, color='black', linestyle='--')
ax1.set_xlabel('Predicted Values')
ax1.set_ylabel('Residuals')
ax1.set_title('Training Residuals vs Predicted Values for Different Quantiles')
ax1.legend()

# Customize the testing residuals plot
ax2.axhline(y=0, color='black', linestyle='--')
ax2.set_xlabel('Predicted Values')
ax2.set_ylabel('Residuals')
ax2.set_title('Testing Residuals vs Predicted Values for Different Quantiles')
ax2.legend()

plt.tight_layout()
plt.show()

"""For cleaned data"""

quantiles = [0.25, 0.5, 0.75]
results = {}

for q in quantiles:
    res, y_pred_train_cq, y_pred_test_cq = quantile_regression(q, X_train_cleaned,
y_train_cleaned, X_test_cleaned, y_test_cleaned)
    results[q] = (res, y_pred_train_cq, y_pred_test_cq)
```

```
plt.figure(figsize=(15, 10))

for i, q in enumerate(quantiles):
    res, y_pred_train_cq, y_pred_test_cq = results[q]

    # Calculate residuals for the training set
    residuals_train_cq = y_train_cleaned - y_pred_train_cq

    # Calculate residuals for the test set
    residuals_test_cq = y_test_cleaned - y_pred_test_cq

    # Plot residuals for the training and test sets together
    plt.subplot(2, 2, i+1)
    plt.scatter(y_pred_train_cq, residuals_train_cq, color='blue', alpha=0.25,
label='Train Residuals')
    plt.scatter(y_pred_test_cq, residuals_test_cq, color='red', alpha=0.25, label='Test
Residuals')
    plt.axhline(y=0, color='red', linestyle='--')
    plt.xlabel('Predicted Values')
    plt.ylabel('Residuals')
    plt.title(f'Cleaned Residuals vs Predicted Values (Quantile {q})')
    plt.legend()

plt.tight_layout()
plt.show()

# Plot residuals for each quantile in separate subplots for training and testing data
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

colors = ['blue', 'green', 'red']
labels = ['Quantile 0.25', 'Quantile 0.5', 'Quantile 0.75']

for i, q in enumerate(quantiles):
    res, y_pred_train_cq, y_pred_test_cq = results[q]
```

```
    # Calculate residuals for the training set
    residuals_train_cq = y_train_cleaned - y_pred_train_cq

    # Calculate residuals for the test set
    residuals_test_cq= y_test_cleaned - y_pred_test_cq

    # Plot residuals for the training set
    ax1.scatter(y_pred_train_cq, residuals_train_cq, color=colors[i], alpha=0.25,
label=f'Train Residuals ({labels[i]})')

    # Plot residuals for the test set
    ax2.scatter(y_pred_test_cq, residuals_test_cq, color=colors[i], alpha=0.25,
label=f'Test Residuals ({labels[i]})', marker='x')

# Customize the training residuals plot
ax1.axhline(y=0, color='black', linestyle='--')
ax1.set_xlabel('Predicted Values')
ax1.set_ylabel('Residuals')
ax1.set_title('Training Residuals vs Predicted Values for Different Quantiles')
ax1.legend()

# Customize the testing residuals plot
ax2.axhline(y=0, color='black', linestyle='--')
ax2.set_xlabel('Predicted Values')
ax2.set_ylabel('Residuals')
ax2.set_title('Testing Residuals vs Predicted Values for Different Quantiles')
ax2.legend()

plt.tight_layout()
plt.show()

"""XGboost"""
```

```
# Perform XGBoost quantile regression for 0.25, 0.5, and 0.75 quantiles
quantiles = [0.25, 0.5, 0.75]
results = {}

for q in quantiles:
    model, y_pred_train_quant, y_pred_test_quant = xgboost_quantile_regression(q, X_train_wo, y_train_wo, X_test_wo, y_test_wo)
    results[q] = (model, y_pred_train_quant, y_pred_test_quant)

plt.figure(figsize=(15, 10))

for i, q in enumerate(quantiles):
    res, y_pred_train_qxg, y_pred_test_qxg = results[q]

    # Calculate residuals for the training set
    residuals_train_qxg = y_train_wo - y_pred_train_qxg

    # Calculate residuals for the test set
    residuals_test_qxg = y_test_wo - y_pred_test_qxg

    # Plot residuals for the training and test sets together
    plt.subplot(2, 2, i+1)
    plt.scatter(y_pred_train_qxg, residuals_train_qxg, color='blue', alpha=0.25, label='Train Residuals')
    plt.scatter(y_pred_test_qxg, residuals_test_qxg, color='red', alpha=0.25, label='Test Residuals')
    plt.axhline(y=0, color='red', linestyle='--')
    plt.xlabel('Predicted Values')
    plt.ylabel('Residuals')
    plt.title(f'Residuals vs Predicted Values (Quantile {q})')
    plt.legend()

plt.tight_layout()
plt.show()
```

```
# Plot residuals for each quantile in separate subplots for training and testing data
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

colors = ['blue', 'green', 'red']
labels = ['Quantile 0.25', 'Quantile 0.5', 'Quantile 0.75']

for i, q in enumerate(quantiles):
    res, y_pred_train_qxg, y_pred_test_qxg = results[q]

    # Calculate residuals for the training set
    residuals_train_qxg = y_train_wo - y_pred_train_qxg

    # Calculate residuals for the test set
    residuals_test_qxg = y_test_wo - y_pred_test_qxg

    # Plot residuals for the training set
    ax1.scatter(y_pred_train_qxg, residuals_train_qxg, color=colors[i], alpha=0.25,
label=f'Train Residuals ({labels[i]})')

    # Plot residuals for the test set
    ax2.scatter(y_pred_test_qxg, residuals_test_qxg, color=colors[i], alpha=0.25,
label=f'Test Residuals ({labels[i]})', marker='x')

# Customize the training residuals plot
ax1.axhline(y=0, color='black', linestyle='--')
ax1.set_xlabel('Predicted Values')
ax1.set_ylabel('Residuals')
ax1.set_title('Training Residuals vs Predicted Values for Different Quantiles')
ax1.legend()

# Customize the testing residuals plot
ax2.axhline(y=0, color='black', linestyle='--')
ax2.set_xlabel('Predicted Values')
```

```
ax2.set_ylabel('Residuals')
ax2.set_title('Testing Residuals vs Predicted Values for Different Quantiles')
ax2.legend()

plt.tight_layout()
plt.show()

"""XGboost on clean data"""

# Perform XGBoost quantile regression for 0.25, 0.5, and 0.75 quantiles
quantiles = [0.25, 0.5, 0.75]
results = {}

for q in quantiles:
    model, y_pred_train_qxgc, y_pred_test_qxgc = xgboost_quantile_regression(q,
X_train_cleaned, y_train_cleaned, X_test_cleaned, y_test_cleaned)
    results[q] = (model, y_pred_train_qxgc, y_pred_test_qxgc)

plt.figure(figsize=(15, 10))

for i, q in enumerate(quantiles):
    res, y_pred_train_qxgc, y_pred_test_qxgc = results[q]

    # Calculate residuals for the training set
    residuals_train_qxgc = y_train_cleaned - y_pred_train_qxgc

    # Calculate residuals for the test set
    residuals_test_qxgc = y_test_cleaned - y_pred_test_qxgc

    # Plot residuals for the training and test sets together
    plt.subplot(2, 2, i+1)
    plt.scatter(y_pred_train_qxgc, residuals_train_qxgc, color='blue', alpha=0.25,
label='Train Residuals')
```

```
    plt.scatter(y_pred_test_qxgc, residuals_test_qxgc, color='red', alpha=0.25,
label='Test Residuals')
    plt.axhline(y=0, color='red', linestyle='--')
    plt.xlabel('Predicted Values')
    plt.ylabel('Residuals')
    plt.title(f'Residuals vs Predicted Values (Quantile {q})')
    plt.legend()

plt.tight_layout()
plt.show()

# Plot residuals for each quantile in separate subplots for training and testing data
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

colors = ['blue', 'green', 'red']
labels = ['Quantile 0.25', 'Quantile 0.5', 'Quantile 0.75']

for i, q in enumerate(quantiles):
    res, y_pred_train_qxgc, y_pred_test_qxgc = results[q]

    # Calculate residuals for the training set
    residuals_train_qxgc = y_train_cleaned - y_pred_train_qxgc

    # Calculate residuals for the test set
    residuals_test_qxgc = y_test_cleaned - y_pred_test_qxgc

    # Plot residuals for the training set
    ax1.scatter(y_pred_train_qxgc, residuals_train_qxgc, color=colors[i], alpha=0.25,
label=f'Train Residuals ({labels[i]})')

    # Plot residuals for the test set
    ax2.scatter(y_pred_test_qxgc, residuals_test_qxgc, color=colors[i], alpha=0.25,
label=f'Test Residuals ({labels[i]})', marker='x')
```

```
# Customize the training residuals plot
ax1.axhline(y=0, color='black', linestyle='--')
ax1.set_xlabel('Predicted Values')
ax1.set_ylabel('Residuals')
ax1.set_title('Training Residuals vs Predicted Values for Different Quantiles')
ax1.legend()

# Customize the testing residuals plot
ax2.axhline(y=0, color='black', linestyle='--')
ax2.set_xlabel('Predicted Values')
ax2.set_ylabel('Residuals')
ax2.set_title('Testing Residuals vs Predicted Values for Different Quantiles')
ax2.legend()

plt.tight_layout()
plt.show()

"""Doing same on Log prices after feature selection"""

data.head()

data_log_fs = pd.concat([data_numeric[selected_features], data[['charges',
'log_charges']]], axis=1)
data_log_fs.head()

# Assuming 'data_numeric' is your initial DataFrame and 'target_column' is defined
as 'charges'
target_column = 'log_charges'
X = data_log_fs.drop(columns=[target_column, 'charges'])
y = data_log_fs[target_column]

# Add a constant term for the intercept
X = sm.add_constant(X)
```

```
# Fit the model
model = sm.OLS(y, X).fit()

# Calculate influence measures
influence = model.get_influence()

# Leverage (hat values)
leverage = influence.hat_matrix_diag

# Standardized residuals
standardized_residuals = influence.resid_studentized_internal

# Cook's distance
cooks_d = influence.cooks_distance[0]

# Identify high leverage points
high_leverage_threshold = 2 * (X.shape[1] / X.shape[0])  # Common threshold
high_leverage_points = np.where(leverage > high_leverage_threshold)[0]

# Identify outliers
outlier_threshold = 2  # Common threshold for standardized residuals
outliers = np.where(np.abs(standardized_residuals) > outlier_threshold)[0]

# Identify points with high Cook's distance
cooks_d_threshold = 4 / len(X)
high_cooks_d_points = np.where(cooks_d > cooks_d_threshold)[0]

# Combine influential points that are also high leverage points or outliers above
Cook's distance threshold
combined_indices = np.unique(np.concatenate((
    np.intersect1d(high_leverage_points, high_cooks_d_points),
    np.intersect1d(outliers, high_cooks_d_points)
)))
```

```
# Plot Cook's distance
plt.figure(figsize=(10, 6))
plt.stem(np.arange(len(cooks_d)), cooks_d, markerfmt=",", linefmt='b-', basefmt=' ',
label='Cook\'s Distance')

# Highlight high leverage points
plt.scatter(high_leverage_points, cooks_d[high_leverage_points], color='red',
label='High Leverage Points', zorder=3)

# Highlight outliers
plt.scatter(outliers, cooks_d[outliers], color='orange', label='Outliers', zorder=3)

plt.title("Cook's Distance with High Leverage Points and Outliers")
plt.xlabel("Observation Index")
plt.ylabel("Cook's Distance")
plt.axhline(y=cooks_d_threshold, color='r', linestyle='--', label='Threshold (4/n)')
plt.legend()
plt.show()

# Display influential points summary
influential_points_summary = pd.DataFrame({
    'Index': combined_indices,
    'Leverage': leverage[combined_indices],
    'Standardized Residuals': standardized_residuals[combined_indices],
    'Cook\'s Distance': cooks_d[combined_indices]
})

# Print influential points summary
print(influential_points_summary)

"""Analysing high leverage points and outliers"""

# Extract high leverage points and outliers from the original dataset
high_leverage_outliers = data_numeric.loc[combined_indices]
```

```
# Compare high leverage points and outliers to the rest of the dataset
non_high_leverage_outliers = data_numeric.drop(index=combined_indices)

# Summary statistics for high leverage points and outliers
high_leverage_outliers_summary = high_leverage_outliers.describe()

# Summary statistics for the rest of the dataset
non_high_leverage_outliers_summary = non_high_leverage_outliers.describe()

# Display the summaries
print("Summary Statistics for High Leverage Points and Outliers:")
print(high_leverage_outliers_summary)
print("\nSummary Statistics for the Rest of the Dataset:")
print(non_high_leverage_outliers_summary)

# Visualize the high leverage points and outliers in the context of the dataset
# Pairplot can help to see the distribution and relationships between variables

# Combine the datasets with a new column to indicate if they are outliers/high
leverage
high_leverage_outliers['Type'] = 'High Leverage/Outlier'
non_high_leverage_outliers['Type'] = 'Normal'

combined_data = pd.concat([high_leverage_outliers, non_high_leverage_outliers])

# Create pairplot
sns.pairplot(combined_data, hue='Type', diag_kind='kde')
plt.suptitle("Pairplot Comparing High Leverage/Outliers and Normal Points", y=1.02)
plt.show()

# Visualize distribution of each variable with boxplots
plt.figure(figsize=(15, 10))
for i, column in enumerate(data_numeric.columns):
```

```
    plt.subplot(len(data_numeric.columns) // 2 + 1, 2, i + 1)
    sns.boxplot(x='Type', y=column, data=combined_data)
    plt.title(f'Boxplot of {column}')
plt.tight_layout()
plt.show()

# Remove outliers and high leverage points above the Cook's distance threshold
data_log_fs_cleaned = data_log_fs.drop(index=combined_indices)

# reset the index of the cleaned DataFrame
data_log_fs_cleaned = data_log_fs_cleaned.reset_index(drop=True)

data_log_fs_cleaned.head()

# 'data_log_fs_cleaned' is cleaned DataFrame and 'target_column' is still 'charges'
target_column = 'log_charges'
X_cleaned = data_log_fs_cleaned.drop(columns=[target_column, 'charges'])
y_cleaned = data_log_fs_cleaned[target_column]

# Add a constant term for the intercept
X_cleaned = sm.add_constant(X_cleaned)

# Fit the model on the cleaned data
model_cleaned = sm.OLS(y_cleaned, X_cleaned).fit()

# Calculate influence measures
influence_cleaned = model_cleaned.get_influence()

# Standardized residuals for QQ plot
standardized_residuals_cleaned = influence_cleaned.resid_studentized_internal

# Cook's distance for Cook's distance plot
cooks_d_cleaned = influence_cleaned.cooks_distance[0]
```

```
# Plot QQ plot
sm.qqplot(standardized_residuals_cleaned, line='45')
plt.title("QQ Plot of Standardized Residuals (Cleaned Data)")
plt.show()

# Plot Cook's distance
plt.figure(figsize=(10, 6))
plt.stem(np.arange(len(cooks_d_cleaned)), cooks_d_cleaned, markerfmt=",",
linefmt='b-', basefmt=' ', label='Cook\'s Distance')
plt.title("Cook's Distance (Cleaned Data)")
plt.xlabel("Observation Index")
plt.ylabel("Cook's Distance")
plt.axhline(y=4 / len(X_cleaned), color='r', linestyle='--', label='Threshold (4/n)')
plt.legend()
plt.show()

"""Linear Regression"""

# Add a constant term for the intercept
X_cleaned = sm.add_constant(X_cleaned)

# Split the cleaned data into training and testing sets
X_train_cleaned, X_test_cleaned, y_train_cleaned, y_test_cleaned =
train_test_split(X_cleaned, y_cleaned, test_size=0.2, random_state=42)

# Fit the model again
linear_reg_cleaned = LinearRegression()
linear_reg_cleaned.fit(X_train_cleaned, y_train_cleaned)

# Predict and evaluate
y_pred_train_cleaned = linear_reg_cleaned.predict(X_train_cleaned)
y_pred_test_cleaned = linear_reg_cleaned.predict(X_test_cleaned)
```

```
evaluate_model(y_train_cleaned, y_pred_train_cleaned, "Linear Regression (Train) - Cleaned")
evaluate_model(y_test_cleaned, y_pred_test_cleaned, "Linear Regression (Test) - Cleaned")

# Calculate residuals for the training set
residuals_train_cleaned = y_train_cleaned - y_pred_train_cleaned

# Calculate residuals for the test set
residuals_test_cleaned = y_test_cleaned - y_pred_test_cleaned

# Plot residuals for the training and test sets together
plt.figure(figsize=(10, 6))
plt.scatter(y_pred_train_cleaned, residuals_train_cleaned, color='blue', alpha=0.25, label='Train Residuals')
plt.scatter(y_pred_test_cleaned, residuals_test_cleaned, color='red', alpha=0.25, label='Test Residuals')
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Cleaned Residuals vs Predicted Values (Training and Test Sets)')
plt.legend()
plt.show()

# Create subplots
fig, axes = plt.subplots(1, 2, figsize=(15, 6))

# Plot for the training set
sns.regplot(ax=axes[0],
            x=y_train_cleaned,
            y=y_pred_train_cleaned,
            scatter_kws={'color': 'blue', 'alpha': 0.5},
            line_kws={'color': 'blue'},
            ci=95)
```

```
axes[0].plot([y_train_cleaned.min(), y_train_cleaned.max()],
         [y_train_cleaned.min(), y_train_cleaned.max()],
         '--', color='red', label='Regression Line')
axes[0].set_xlabel('Actual log Charges')
axes[0].set_ylabel('Predicted log Charges')
axes[0].set_title('Actual vs Predicted log Charges (Training Data)')
axes[0].legend()

# Plot for the test set
sns.regplot(ax=axes[1],
        x=y_test_cleaned,
        y=y_pred_test_cleaned,
        scatter_kws={'color': 'green', 'alpha': 0.5},
        line_kws={'color': 'green'},
        ci=95)
axes[1].plot([y_test_cleaned.min(), y_test_cleaned.max()],
         [y_test_cleaned.min(), y_test_cleaned.max()],
         '--', color='red', label='Regression Line')
axes[1].set_xlabel('Actual log Charges')
axes[1].set_ylabel('Predicted log Charges')
axes[1].set_title('Actual vs Predicted log Charges (Test Data)')
axes[1].legend()

# Adjust layout
plt.tight_layout()
plt.show()

# Create a combined DataFrame for easier plotting
combined_df = pd.DataFrame({
   'Actual': pd.concat([pd.Series(y_train_cleaned).reset_index(drop=True),
                pd.Series(y_test_cleaned).reset_index(drop=True)]),
   'Predicted': pd.concat([pd.Series(y_pred_train_cleaned).reset_index(drop=True),
                   pd.Series(y_pred_test_cleaned).reset_index(drop=True)]),
   'Dataset': ['Train'] * len(y_train_cleaned) + ['Test'] * len(y_test_cleaned)
```

```
})

# Create the plot
plt.figure(figsize=(10, 6))

# Plot the training data with regression line and confidence interval
sns.regplot(data=combined_df[combined_df['Dataset'] == 'Train'],
            x='Actual',
            y='Predicted',
            scatter_kws={'color': 'blue', 'alpha': 0.5},
            line_kws={'color': 'blue'},
            label='Training Data',
            ci=95)

# Plot the test data with regression line and confidence interval
sns.regplot(data=combined_df[combined_df['Dataset'] == 'Test'],
            x='Actual',
            y='Predicted',
            scatter_kws={'color': 'green', 'alpha': 0.5},
            line_kws={'color': 'green'},
            label='Test Data',
            ci=95)

# Add a line for perfect prediction
plt.plot([combined_df['Actual'].min(), combined_df['Actual'].max()],
         [combined_df['Actual'].min(), combined_df['Actual'].max()],
         '--', color='red', label='Regression Line')

# Customize the plot
plt.xlabel('Actual Charges')
plt.ylabel('Predicted Charges')
plt.title('Actual vs Predicted log Charges (Train and Test Data)')
plt.legend()
plt.grid()
```

```
plt.show()

X_cleaned = data_log_fs_cleaned.drop(columns=[target_column, 'charges'])
y_cleaned = data_log_fs_cleaned[target_column]

# Add a constant term for the intercept
X_cleaned = sm.add_constant(X_cleaned)

# Split the cleaned data into training and testing sets
X_train_cleaned, X_test_cleaned, y_train_cleaned, y_test_cleaned =
train_test_split(X_cleaned, y_cleaned, test_size=0.2, random_state=42)

quantiles = [0.25, 0.5, 0.75]
results = {}

for q in quantiles:
    res, y_pred_train_cq, y_pred_test_cq = quantile_regression(q, X_train_cleaned,
y_train_cleaned, X_test_cleaned, y_test_cleaned)
    results[q] = (res, y_pred_train_cq, y_pred_test_cq)

plt.figure(figsize=(15, 10))

for i, q in enumerate(quantiles):
    res, y_pred_train_cq, y_pred_test_cq = results[q]

    # Calculate residuals for the training set
    residuals_train_cq = y_train_cleaned - y_pred_train_cq

    # Calculate residuals for the test set
    residuals_test_cq = y_test_cleaned - y_pred_test_cq

    # Plot residuals for the training and test sets together
    plt.subplot(2, 2, i+1)
```

```
    plt.scatter(y_pred_train_cq, residuals_train_cq, color='blue', alpha=0.25,
label='Train Residuals')
    plt.scatter(y_pred_test_cq, residuals_test_cq, color='red', alpha=0.25, label='Test
Residuals')
    plt.axhline(y=0, color='red', linestyle='--')
    plt.xlabel('Predicted Values')
    plt.ylabel('Residuals')
    plt.title(f'Cleaned Residuals vs Predicted Values (Quantile {q})')
    plt.legend()

plt.tight_layout()
plt.show()

# Create subplots for each quantile
fig, axes = plt.subplots(1, len(quantiles), figsize=(18, 6))

for i, q in enumerate(quantiles):
    # Unpack the results
    res, y_pred_train_cq, y_pred_test_cq = results[q]

    # Create a combined DataFrame for easier plotting
    combined_df = pd.DataFrame({
        'Actual': pd.concat([y_train_cleaned.reset_index(drop=True),
y_test_cleaned.reset_index(drop=True)]),
        'Predicted': pd.concat([pd.Series(y_pred_train_cq).reset_index(drop=True),
                                pd.Series(y_pred_test_cq).reset_index(drop=True)]),
        'Dataset': ['Train'] * len(y_train_cleaned) + ['Test'] * len(y_test_cleaned)
    })

    # Plot for the current quantile
    sns.regplot(ax=axes[i],
                data=combined_df,
                x='Actual',
                y='Predicted',
```

```
            scatter_kws={'alpha': 0.5},
            line_kws={'color': 'blue'},
            ci=None)

    # Add a line for perfect prediction
    axes[i].plot([combined_df['Actual'].min(), combined_df['Actual'].max()],
                 [combined_df['Actual'].min(), combined_df['Actual'].max()],
                 '--', color='red', label='Perfect Prediction Line')

    # Customize the plot
    axes[i].set_title(f'Actual vs Predicted Charges (Quantile: {q})')
    axes[i].set_xlabel('Actual Charges')
    axes[i].set_ylabel('Predicted Charges')
    axes[i].legend()

# Adjust layout
plt.tight_layout()
plt.show()

# Create subplots: 1 row, 2 columns
fig, axes = plt.subplots(1, 2, figsize=(15, 6))

# Linear Regression Plot
# Combine actual and predicted values for linear regression
actual_linear = pd.concat([y_train_cleaned, y_test_cleaned]).reset_index(drop=True)
predicted_linear = pd.concat([pd.Series(y_pred_train_cleaned),
pd.Series(y_pred_test_cleaned)]).reset_index(drop=True)

sns.regplot(ax=axes[0],
            x=actual_linear,
            y=predicted_linear,
            scatter_kws={'color': 'blue', 'alpha': 0.5},
            line_kws={'color': 'blue'},
            ci=None)
```

```
axes[0].plot([actual_linear.min(), actual_linear.max()],
         [actual_linear.min(), actual_linear.max()],
         '--', color='red', label='Perfect Prediction Line')
axes[0].set_xlabel('Actual Charges')
axes[0].set_ylabel('Predicted Charges')
axes[0].set_title('Linear Regression: Actual vs Predicted Charges')
axes[0].legend()

# Quantile Regression Plot for 0.5 Quantile
q = 0.5
res, y_pred_train_cq, y_pred_test_cq = results[q]

# Create a combined DataFrame for the quantile regression plot
combined_df_q5 = pd.DataFrame({
   'Actual': pd.concat([y_train_cleaned.reset_index(drop=True),
y_test_cleaned.reset_index(drop=True)]),
   'Predicted': pd.concat([pd.Series(y_pred_train_cq).reset_index(drop=True),
                    pd.Series(y_pred_test_cq).reset_index(drop=True)]),
   'Dataset': ['Train'] * len(y_train_cleaned) + ['Test'] * len(y_test_cleaned)
})

sns.regplot(ax=axes[1],
         data=combined_df_q5,
         x='Actual',
         y='Predicted',
         scatter_kws={'color': 'green', 'alpha': 0.5},
         line_kws={'color': 'green'},
         ci=None)
axes[1].plot([combined_df_q5['Actual'].min(), combined_df_q5['Actual'].max()],
          [combined_df_q5['Actual'].min(), combined_df_q5['Actual'].max()],
          '--', color='red', label='Regression Line')
axes[1].set_xlabel('Actual Charges')
axes[1].set_ylabel('Predicted Charges')
axes[1].set_title('Quantile Regression (0.5): Actual vs Predicted Charges')
```

```
axes[1].legend()

# Adjust layout
plt.tight_layout()
plt.show()

"""XG Boost

Linear Regression
"""

# Split the Original data into training and testing sets
X_train_wo, X_test_wo, y_train_wo, y_test_wo = train_test_split(X, y, test_size=0.2,
random_state=42)

# Initialize the XGBRegressor
xgb_reg = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=100,
learning_rate=0.1, max_depth=3, random_state=42)

# Fit the model
xgb_reg.fit(X_train, y_train)

# Predict
y_pred_train = xgb_reg.predict(X_train)
y_pred_test = xgb_reg.predict(X_test)

def evaluate_model(y_true, y_pred, model_name):
    mse = mean_squared_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    print(f'{model_name} - Mean Squared Error: {mse:.4f}, R2 Score: {r2:.4f}')

evaluate_model(y_train, y_pred_train, "XGBoost Regression (Train)")
evaluate_model(y_test, y_pred_test, "XGBoost Regression (Test)")
```

```
X_cleaned = data_log_fs_cleaned.drop(columns=[target_column, 'charges'])
y_cleaned = data_log_fs_cleaned[target_column]

# Add a constant term for the intercept
X_cleaned = sm.add_constant(X_cleaned)

# Split the cleaned data into training and testing sets
X_train_cleaned, X_test_cleaned, y_train_cleaned, y_test_cleaned =
train_test_split(X_cleaned, y_cleaned, test_size=0.2, random_state=42)

quantiles = [0.25, 0.5, 0.75]
results = {}

for q in quantiles:
    #res, y_pred_train_cq, y_pred_test_cq = quantile_regression(q, X_train_cleaned,
y_train_cleaned, X_test_cleaned, y_test_cleaned)
    res, y_pred_train_cq, y_pred_test_cq = xgboost_quantile_regression(q,
X_train_cleaned, y_train_cleaned, X_test_cleaned, y_test_cleaned)
    results[q] = (res, y_pred_train_cq, y_pred_test_cq)

plt.figure(figsize=(15, 10))

for i, q in enumerate(quantiles):
    res, y_pred_train_cq, y_pred_test_cq = results[q]

    # Calculate residuals for the training set
    residuals_train_cq = y_train_cleaned - y_pred_train_cq

    # Calculate residuals for the test set
    residuals_test_cq = y_test_cleaned - y_pred_test_cq

    # Plot residuals for the training and test sets together
    plt.subplot(2, 2, i+1)
```

```
    plt.scatter(y_pred_train_cq, residuals_train_cq, color='blue', alpha=0.25,
label='Train Residuals')
    plt.scatter(y_pred_test_cq, residuals_test_cq, color='red', alpha=0.25, label='Test
Residuals')
    plt.axhline(y=0, color='red', linestyle='--')
    plt.xlabel('Predicted Values')
    plt.ylabel('Residuals')
    plt.title(f'Cleaned Residuals vs Predicted Values (Quantile {q})')
    plt.legend()

plt.tight_layout()
plt.show()
```