Team 3

Professor Ibrahim Youssef

SBE2240 – Biostatistics

24 June 2023

Linear Regression Analysis on the Medical Cost Personal Dataset

Introduction

This project aims to conduct an association experiment on the Medical Cost Personal Dataset on the Kaggle platform using linear regression analysis. This dataset has 7 columns, 6 of which are predictors (`p`). The dataset also has 1338 rows (`n`), so the condition `n ≥ p + 5` is satisfied. The quantitative columns in this dataset are `age, bmi, children,` and `charges`. The categorical columns are `sex, smoker,` and `region`.

Methods

DataFrame. The Pandas module, a popular open-source data analysis and manipulation library for Python is imported with the alias `pd`. The CSV file containing the Medical Cost Personal Dataset is then uploaded and a Pandas DataFrame (`df`) containing the 7 columns of the uploaded CSV file is created.

Removing outliers. Outliers are removed from the dataset using the Z-score method, a common technique for identifying and removing outliers from a dataset. It works by calculating the Z-score for each data point, which measures how many standard deviations away from the mean the data point is. Data points with a Z-score above a certain threshold (usually 3 or -3) are considered outliers and can be removed from the dataset. A function named `z_score` is defined to calculate the Z-score for a given Pandas Series (i.e., a column of the DataFrame). It takes a Series as input, subtracts its mean, and then divides it by its

standard deviation to calculate the Z-score. The Z-score for each numeric column in the `df` DataFrame is calculated using the `z_score` function. It selects only the numeric columns using `df.select_dtypes(include='number')`, applies the z_score function to each column using `.apply(z_score)`, and stores the resulting Z-scores in a new DataFrame called `z_scores`. `z_scores.abs()` is a method call that returns a new Pandas DataFrame containing the absolute values of the Z-scores in the z_scores DataFrame. The `abs()` method is a built-in method of Pandas DataFrames that calculates the absolute value of each element in the DataFrame. When applied to a DataFrame containing Z-scores, it returns a new DataFrame containing the absolute values of the Z-scores. The `any(axis=1)` method call is used to check if any element along the specified axis (in this case, rows) of a Pandas DataFrame is `True`. The `any()` method is a built-in method of Pandas DataFrames that returns a boolean value indicating whether any element in the DataFrame is `True`. When called with the `axis=1` argument, it checks if any element along each row of the DataFrame is True and returns a boolean Series containing the result for each row. In the context of identifying outliers using the Z-score method, the `any(axis=1)` method call is used to check if any element along each row of the DataFrame containing the absolute values of the Z-scores is greater than the specified threshold. If any element along a row is greater than the threshold, it means that at least one data point in that row is an outlier, and the corresponding value in the boolean Series returned by the `any(axis=1)` method call will be `True`. This creates a boolean mask that is `True` for rows that contain at least one outlier and `False` otherwise. This boolean mask is then used to remove outliers from the DataFrame. It uses boolean indexing with `~outliers` to select only rows where the mask is `False` (i.e., rows that do not contain any outliers) and stores the resulting cleaned DataFrame in a new variable called `df_clean`.

Quantitative description. `stats = df_clean.describe()` calculates summary statistics for the `df_clean` DataFrame and stores the result in a new variable called `stats.` The `describe()` method is a built-in method of Pandas DataFrames that generates descriptive statistics for the columns in the DataFrame. By default, it calculates the following statistics for each numeric column: count, mean, standard deviation, minimum, 25th percentile, median (50th percentile), 75th percentile, and maximum. The interquartile range (IQR), a measure of the spread of the middle 50% of the data is also calculated. It is calculated as the difference between the 75th percentile (Q3) and the 25th percentile (Q1) of the data. quantile method. The `quantile` method takes a value between 0 and 1 as input and returns the corresponding percentile of the data. `df_clean.quantile(0.25)` calculates the 25th percentile (Q1) and `df_clean.quantile(0.75)` calculates the 75th percentile (Q3). The results are stored in two new variables called `q1` and `q3`, respectively. The interquartile range (IQR) for each numeric column by subtracting `q1` from `q3`. This is equivalent to calculating Q3 - Q1 for each column. The resulting IQR values are stored in a new variable called `iqr`.

Standardization. `df_clean_standardized = df_clean.copy()` creates a new DataFrame called `df_clean_standardized` as a copy of the `df_clean` DataFrame. `df_clean_standardized.select_dtypes(include='number')` selects only the columns in `df_clean_standardized` that have a numeric data type. This returns a new DataFrame containing only the selected columns. `.apply(z_score)` applies the `z_score` function to each column in the selected DataFrame. This calculates the Z-score for each value in each column by subtracting the mean of the column from each value and then dividing by the standard deviation of the column. The result is a new DataFrame containing the Z-scores for each numeric column.

`df_clean_standardized[z_scores.columns] = z_scores` assigns the standardized values in `z_scores` to the corresponding columns in `df_clean_standardized`. `z_scores.columns` returns the column labels of the `z_scores` DataFrame. These are the names of the columns that were standardized. `df_clean_standardized[z_scores.columns]` selects the corresponding columns in `df_clean_standardized` using these column labels. This returns a new DataFrame containing only the selected columns.
`df_clean_standardized[z_scores.columns] = z_scores` assigns the values in `z_scores` to the selected columns in `df_clean_standardized`. This replaces the original values in these columns with their standardized values. After these two lines of code are executed, `df_clean_standardized` will contain the standardized data of `df_clean`.

Plots. `pyplot` module is imported from the `matplotlib` library and is given the alias `plt`. This module provides functions for creating plots and visualizations. A histogram and a kernel density estimate (KDE) plot are created for the numeric columns in `df_clean`. The hist method creates a histogram for each numeric column, showing the distribution of the data. `figsize=(10, 10)` specifies the size of the figure in inches. In this case, it sets the width and height of the figure to 10 inches. The plot method with `kind='kde'` creates a KDE plot for each numeric column, showing an estimate of the probability density function of the data. `kind='kde'` specifies the type of plot to create. In this case, it sets the plot type to `'kde'`, which stands for kernel density estimate. `subplots=True` specifies whether to create a separate subplot for each column in the DataFrame. In this case, it's set to `True`, which means that a separate subplot will be created for each numeric column. `layout=(2, 2)` specifies the layout of the subplots in the figure. In this case, it's set to `(2, 2)`, which

means that the subplots will be arranged in a 2x2 grid. `sharex=False` specifies whether to share the x-axis between subplots. In this case, it's set to False, which means that each subplot will have its own x-axis. The categorical columns from a DataFrame `df_clean` are then selected using the `select_dtypes` method with `include='object'`. This creates a new variable called `categorical_columns` that contains the names of all categorical columns in `df_clean`. Bar plots for each categorical column in `df_clean` are then created. A for loop iterates over each column in `categorical_columns`, creating a new figure using `plt.figure`, and then the `plot.bar` method is used to create a bar plot showing the counts of each unique value in the column. The title of each plot is set to the name of the column using the `title=col` argument. Normalized bar plots for each categorical column in the `df_clean` are then created. Another for loop iterates over each column in `categorical_columns`, creating a new figure using `plt.figure`, and then the `plot.bar` method is used to create a bar plot showing the relative frequency of each unique value in the column (i.e., its count divided by the total number of rows in the DataFrame). The title of each plot is set to the name of the column using the `title=col` argument. Finally, after all plots have been created, `plt.show()` is used to display all figures on the screen.

To visually explore the relationships and distributions among the selected numeric variables, a scatter matrix was constructed using the `scatter_matrix` function from the `pandas.plotting` module. The `scatter_matrix` function generates scatter plots and histograms for each combination of numeric variables in the dataset.

For our research, we selected the columns 'charges', 'bmi', and 'age' from the preprocessed dataset `df_clean`. These columns were deemed relevant to our investigation. The

`df_clean` dataset was subsetted to include only these selected columns, creating a new DataFrame called `df_selected`.

The scatter matrix was created by calling `scatter_matrix(df_selected, figsize=(10, 10), diagonal='hist')`. The `figsize` parameter specified the dimensions of the resulting figure, with a width and height set to 10 inches. The `diagonal` parameter was set to `'hist'`, indicating that histograms should be plotted on the diagonal of the scatter matrix.

The resulting scatter matrix provides a visual representation of the relationships between the variables. Each scatter plot represents the correlation between two variables, while the histograms show the distribution of each variable. This allows us to gain insights into possible patterns, trends, or correlations among the variables of interest.

The scatter matrix provides a comprehensive overview of the relationships and distributions within the dataset, aiding in the exploration and analysis of the data. By visually examining the scatter plots and histograms, we can identify potential associations and patterns that may guide our research findings and further analysis

Distribution tests. The Shapiro-Wilk test from the `scipy` library is used to test whether the age, bmi, children, and charges columns in the `df_clean` DataFrame follow a Gaussian (normal) distribution. The `shapiro` function is imported from the `scipy.stats` module. This function performs the Shapiro-Wilk test, which is a statistical test for normality. The age column in the `df_clean` DataFrame is tested whether it follows a Gaussian distribution. The age column is extracted from the DataFrame using `df_clean['age']`, the Shapiro-Wilk test is performed on the data using `shapiro(data)`, and then the test statistic and p-value are printed out. The results of the test are also interpreted by comparing the p-value to a

significance level of 0.05. If the p-value is greater than 0.05, it is concluded that the data follows a Gaussian distribution (i.e., fails to reject the null hypothesis). Otherwise, it is concluded that the data does not follow a Gaussian distribution (i.e., rejects the null hypothesis). The same thing is done for the three other columns in the `df_clean` DataFrame: bmi, children, and charges. They are extracted from the DataFrame, the Shapiro-Wilk test is performed on the data, the test statistic and p-value are printed out, and the results are interpreted by comparing the p-value to a significance level of 0.05.

The Anderson-Darling test is used to test whether the numeric columns of the `df_clean` DataFrame follow an exponential distribution. The `anderson` function is imported from the `scipy.stats` module. The 'age' column from the df_clean DataFrame is extracted using `data = df_clean['age']` and is stored in the variable `data`. The Anderson-Darling test on the data in the 'age' column is performed using `result = anderson(data, dist='expon')`. The first argument to the `anderson` function is the data to be tested, and the second argument specifies the distribution to test against, in this case, an exponential distribution. The result of the test is stored in the variable result. `print(f'Statistic: {result.statistic:.4f}')` prints the test statistic calculated from the data. A for loop iterates over the critical values for different significance levels returned by the Anderson-Darling test. For each significance level, the code compares the test statistic to the critical value. If the test statistic is less than the critical value, then we fail to reject the null hypothesis that the data comes from an exponential distribution. Otherwise, we reject the null hypothesis and conclude that the data does not come from an exponential distribution. The same thing is repeated for three more columns: 'bmi', 'children', and 'charges'. For each column, the code extracts the data, performs the Anderson-Darling test, and prints the results.

The Kolmogorov-Smirnov (K-S) test is used to test whether the numeric columns of the `df_clean` DataFrame follow a uniform distribution. The `stats` module is imported from the `scipy` library. This module contains statistical functions, including the `kstest` function used to perform the Kolmogorov-Smirnov test. The ‘age’ column from the `df_clean` DataFrame is extracted using `data = df_clean['age']` and is stored in the variable `data`. `stat, p = stats.kstest(data, 'uniform')` performs the Kolmogorov-Smirnov test on the data in the ‘age’ column. The first argument to the `kstest` function is the data to be tested, and the second argument specifies the distribution to test against, in this case, a uniform distribution. The result of the test is a tuple containing the test statistic and p-value, which are stored in the variables `stat` and `p`, respectively. `print(f'Statistic: {stat:.4f}, p-value: {p:.4f}')` prints the test statistic and p-value calculated from the data. The code sets a significance level of 0.05 and compares the p-value to this significance level. If the p-value is greater than 0.05, then we fail to reject the null hypothesis that the data comes from a uniform distribution. Otherwise, we reject the null hypothesis and conclude that the data does not come from a uniform distribution. The same thing is repeated for three more columns: ‘bmi’, ‘children’, and ‘charges’. For each column, the code extracts the data, performs the Kolmogorov-Smirnov test, prints the results, and interprets them.

Correlation coefficients. `data_encoded = pd.get_dummies(df_clean)` one-hot encodes the categorical variables in the `df_clean` DataFrame. One-hot encoding is a process of converting categorical data variables into a numerical form that can be used in machine learning models. The `get_dummies` function from the `pandas` library is used to perform the one-hot encoding. The resulting DataFrame with encoded categorical variables is stored in the variable `data_encoded`. `corr_matrix = data_encoded.corr()` computes the correlation matrix of the encoded data. The correlation matrix is a table

showing the correlation coefficients between different variables in the data. The `corr` function from the `pandas` library is used to compute the correlation matrix. The resulting correlation matrix is stored in the variable `corr_matrix`. `response_variable = 'charges'` sets the response variable to be 'charges'. This is the variable for which we want to compute the correlations with other variables. `correlations = corr_matrix[response_variable]` extracts the correlation coefficients for the response variable from the correlation matrix. This gives us a series of correlation coefficients between the response variable and each of the other variables in the data. `print(correlations)` prints the series of correlation coefficients. This code computes and prints the correlations between a response variable ('charges') and other variables in the DataFrame `df_clean`. It first one-hot encodes any categorical variables in the data, then computes the correlation matrix, and finally extracts and prints the correlations for the response variable.

Multivariable linear regression analysis. The `LinearRegression` class from `sklearn.linear_model` module from the `scikit-learn` library that fits a linear model to the data by minimizing the sum of squared residuals is used to create a linear regression model. `model_multiple_regression = LinearRegression()` creates an instance of the `LinearRegression` class and assigns it to the variable `model_multiple_regression`. This instance is used to fit the multiple regression model. `X = data_encoded.drop( ['charges', 'smoker_no', 'sex_female'], axis=1)` selects the predictor variables/features by dropping the response variable and any additional columns that are redundant for our analysis. The `drop` function from the `pandas` library is used to remove specified columns from the DataFrame . `X` is a new DataFrame that only contains the `data_encoded` DataFrame after dropping

`['charges', 'smoker_no', 'sex_female']` which is a list of columns to be dropped containing `charges` since it is considered as the response variable, `smoker_no` because for a single data point (one person), that person may be a smoker or may not be one (`smoker_yes` or not `smoker_yes`) and not both, then for all data points it is enough to consider only one of these two columns, the same applies to `sex_female` and `sex_male`. `axis=1` indicates that columns should be dropped. `y = data_encoded['charges']` Selects the response variable, which is the `charges` column from the `data_encoded` DataFrame, the result is a pandas Series containing the response values assigned to `y`. `model_multiple_regression.fit(X, y)` fits the linear regression model to the data. The `fit` function takes the predictor variables and response variable and when it is called it internally performs calculations to estimate the regression coefficients using mathematical techniques such as gradient descent and aims to minimize the sum of squared residuals between the predicted and the actual values of the response variable by adjusting the coefficients iteratively. `model_multiple_regression.coef_[i]` After fitting the model the `coef_` attribute of the LinearRegression object named `model_multiple_regression` provides an array of coefficient values each corresponding to a predictor variable in the same order provided during fitting. The coefficient values are accessed by index `i`. `model_multiple_regression.intercept_` retrieves the intercept term (or bias term) of the multiple linear regression model.

Assessment of the multivariable regression quality in terms of the regression model as a whole. R-squared (Coefficient of Determination): R-squared is a statistical measure that quantifies the proportion of the response variable's variance that is explained by the predictors in the model and indicates how well the model fits the data. R-squared value

ranges from 0 to 1. A higher R-squared value (closer to 1) suggests that a larger proportion of the response variable's variability is accounted for by the predictors, indicating a better fit. `R_squared = model_multiple_regression.score(X, y)` calculates the Coefficient of Determination for the multivariable regression model using the `score` method that takes input data `X` and the corresponding target(actual) values y. it internally computes the predicted target values $\hat{y}$ using the fitted model and the input data, then calculates R_squared that has the following mathematical formula: $R^2 = 1 - \frac{SSR}{SST}$,where SSR represents the sum of squared residuals, and SST represents the total sum of squares or the sum of the distance the data is away from the mean all squared. The calculated R_squared value is then assigned to the variable `R_squared`. `Adjusted R_squared` is a modified version of R_squared that is more precise by considering the impact of additional (redundant) independent variables. It is calculated using the following formula $1 - [(1 - R\_squared) * (n - 1) / (n - p - 1)]$ where n is number of observations (the number of data points in the dataset) and p is the number of predictors. The value of the adjusted R_squared is assigned to the variable `adjusted_R_squared` and is less than or equal to the value of R_squared. Root Mean Square Error (RMSE) for the Multiple Linear Regression Model: `y_pred = model_multiple_regression.predict(X)` calculates the predicted values of the response variable using the predict method that takes the predictor variables as an input and uses the fitted model (`model_multiple_regression`). The predicted values are assigned to `y_pred` variable. The mean square error is calculated between the actual response variable y and the predicted response variable `y_pred` using `mean_squared_error` function from the `scikit-learn` library that also takes the `squared=False` parameter to specify that the root mean squared error (RMSE) instead of the mean squared error (MSE). Root Mean Square Error (RMSE) for the Simple Linear Regression: the function `def rmse_calc(y,`

`x, slope, intercept)`: is defined that takes the actual response value `y`, a specific predictor variable `x`, the slope and the line intercept for each line model. `y_pred = intercept + np.dot(x, slope)` calculates the predicted response value `y_pred` by using `.dot()` function in NumPy to perform dot product operation on the predictor variables x and the slope coefficients slope, and then adding the intercept to it. `residuals = y - y_pred` calculates the residual which is the difference between the actual and the predicted response values. `rmse = (residuals**2).mean() ** 0.5` calculates the RMSE by first squaring the residuals (`residuals**2`), then taking the mean of the squared residuals using the `.mean()` function from NumPy, and finally taking the square root of the mean squared.

Multiple Simple Linear Regression (Calculating Regression Coefficients from Scratch). First, we define the `simple_linear_regression` function which takes two arguments: `x` and `y`, which represent the independent (predictor) and dependent (response) variables, respectively. The function calculates the slope and intercept of the regression line using the formula for simple linear regression. The formula for the slope of the regression line is `slope = xy_cov / x_var`, where `xy_cov` is the covariance between `x` and `y`, and `x_var` is the variance of `x`. The formula for the intercept of the regression line is `intercept = y_mean - slope * x_mean`, where `y_mean` is the mean of `y` and `x_mean` is the mean of `x`.

The function starts by calculating the length of `x`, which is also the length of `y` since they must have the same length. It then calculates the means of `x` and `y` using the formula `mean = sum(x) / n`, where `n` is the length of `x`.

Next, the function calculates the variance of `x` using the formula `variance = sum((xi - x_mean) ** 2 for xi in x)`. This formula calculates the sum of squared

differences between each value in `x` and the mean of `x`. The function also calculates the covariance between `x` and `y` using the formula `covariance = sum((xi - x_mean) * (yi - y_mean) for xi, yi in zip(x, y))`. This formula calculates the sum of products between corresponding values in `x` and `y`, after subtracting their respective means.

Finally, the function uses these values to calculate the slope and intercept of the regression line using the formulas mentioned above. It returns these values as a tuple.

Afterwards, this function is used to calculate the regression coefficients for several different independent variables in the dataset called `df_clean`. These independent variables include `bmi`, `children`, `age`, `sex`, `region`, and `smoker`. For each independent variable, the slope and intercept of the regression line are calculated and then printed to the console.

Also, before the intercepts and slopes are calculated, one-hot encoding is used to convert categorical variables such as `sex`, `region`, and `smoker` into numerical values. This is done using the `pd.get_dummies` function from the `pandas` library.

Calculating Individual Regression Coefficients Using Python Standard Library. In our analysis, the first part of the code calculates the regression coefficients for the numerical independent variables `age`, `bmi`, and `children`. For each numerical independent variable, the code performs the following steps:

1. Remove missing values from the feature: The `df_clean` DataFrame is used to extract the values for the current independent variable and remove any missing values.
2. Add a constant term to the predictor variable: The `sm.add_constant` function is used from the `statsmodels` library to add a column of ones to the predictor

variable. This is necessary because the simple linear regression model includes an intercept term.

3. Fit a simple linear regression model using `statsmodels`: The `sm.OLS` function from the `statsmodels` library is used to fit a simple linear regression model. This function takes two arguments: the dependent (response) variable `y` and the predictor variable `x`, which includes a constant term.
4. Get the intercept and slope coefficients from the `statsmodels` model: The code uses the `params` attribute of the fitted model to extract the intercept and slope coefficients. These coefficients represent the intercept and slope of the regression line.

Then, the second part of the code calculates the regression coefficients for several categorical independent variables, including `smoker`, `region`, and `sex`. For each categorical independent variable, one-hot encoding is performed to convert it into binary variables. This is done using the `pd.get_dummies` function from the `pandas` library. For example, the code uses one-hot encoding to convert the `smoker` categorical variable into two binary variables: `x_smoker_no_sm` and `x_smoker_yes_sm`. These binary variables represent whether or not an individual is a smoker.

For each binary variable created by one-hot encoding, similar steps are performed as for the numerical independent variable. Meaning that for each binary variable, it is converted to a float data type, a constant term is added to the predictor variable, and a simple linear regression model is fitted using the `sm.OLS` function. The code then extracts the intercept and slope coefficients from the fitted model and prints them to the console. These coefficients represent the effect of each binary variable on the dependent variable `charges`.

It is to be noted that when you compare the resulting individual regression coefficients, both implemented methods of calculation produce similar results, as they both calculate the

regression coefficients for a simple linear regression model. However, the second set of code uses a library to fit the model, which can make it easier to use and more flexible.
The `statsmodels` library provides many additional features and options for fitting and analyzing regression models.

Results and Discussion

Before removing outliers, the dataset had 1338 rows. After removing outliers, the number of rows decreased to 1309. We performed several statistical tests on the ‘age’, ‘bmi’, ‘children’, and ‘charges’ columns, including the Shapiro-Wilk test for normality, the Anderson-Darling test for exponential distribution, and the Kolmogorov-Smirnov test for uniform distribution. The results of these tests showed that none of these columns follow a normal, exponential or uniform distribution.

For multiple linear regression the use of the `LinearRegression` class from `sklearn.linear_model` module from the scikit-learn library helped fitting the model to the data and provided the resulted coefficients and the intercept term.

1- Assessment of the two models

1.1- Correlation coefficients, R-squared score and adjusted R-squared

When calculating the correlation coefficients for the features/predictors we got these results:

age	0.305263
bmi	0.191453
children	0.100438
charges	1.000000
sex_female	-0.059455
sex_male	0.059455
smoker_no	-0.785129
smoker_yes	0.785129

region_northeast 0.011435

region_northwest -0.045847

region_southeast 0.074829

region_southwest -0.042922

This shows that for the first model that predicts the response: charges based on these predictors, the feature smoker_yes and smoker_no are the best for prediction. The method of calculating R-squared (Coefficient of Determination) for the assessment of the multivariable regression quality in terms of the regression model as a whole resulted R-squared equals approximately 0.7532, this means that 75.32% of the dependent variable's variability is explained by the independent variables. The modified version of R_squared (Adjusted R_squared) approximately equals 0.75211 which as expected is less than the R_squared value.

1.2- Comparing Root Mean Squared Error metric.

The Root Mean Squared Error calculations for the Simple Linear Regression resulted the following:

RMSE for bmi: 11516.631994484027

RMSE for children: 11674.350758069299

RMSE for age: 11173.612368442391

RMSE for sex (male): 11712.926761158675

RMSE for sex (female): 11712.926761158675

RMSE for region (northeast): 11732.916833026438

RMSE for region (northwest): 11721.34592811692

RMSE for region (southeast): 11700.787209399117

RMSE for region (southwest): 11722.87046882455

RMSE for smoker (yes): 7267.04695304955

RMSE for smoker (no): 7267.04695304955

And RMSE for the Multiple Linear Regression Model equals 5828.598068041572.

Comparing this value with all the other values calculated for each feature it is smaller, and this implies that the multiple linear regression model is more accurate in predicting the response variable.

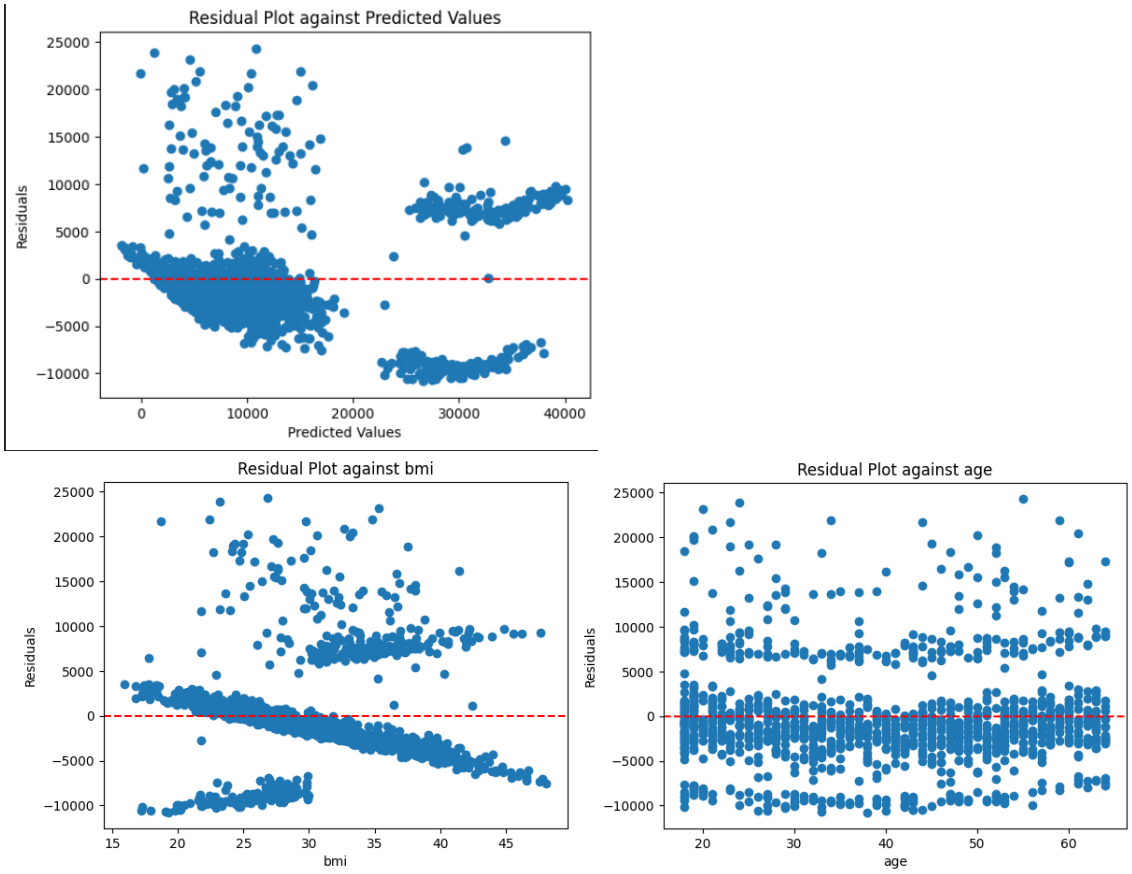
2- Assessment of the best model

2.1- Performing residual analysis

Residual analysis is one of the methods used to assess the quality of the model. It can show the residual (which is the difference between the predicted value and the real value) against multiple parameters. It starts with calculating the predicted value for y, we used the equation for y we got for the model as whole. It uses the function predict in `LinearRegression` class in `scikit-learn` popular python package. The second step is calculating the Residuals: The residuals are computed by subtracting the predicted values from the actual observed values of the response variable (y). Then a scatter plot is created with the predicted values on the x-axis and the corresponding residuals on the y-axis. This plot helps visualize the relationship between the predicted values and the residuals. The horizontal red dashed line at y=0 represents the ideal condition where the residuals are centered around the zero, indicating unbiased predictions. It shows that most of the predictions have negative residuals, meaning that they are larger than the real Y. Also, most of them are around the

horizontal red dashed line, meaning that the residual is small, especially for small predicted values.

Additional scatter plots are created to assess the relationship between the residuals and two specific predictor variables, 'age' and 'bmi'. It shows that there's a better distribution for age and a kind of linear distribution for 'bmi', which behaves like a predicted error that can be compensated.



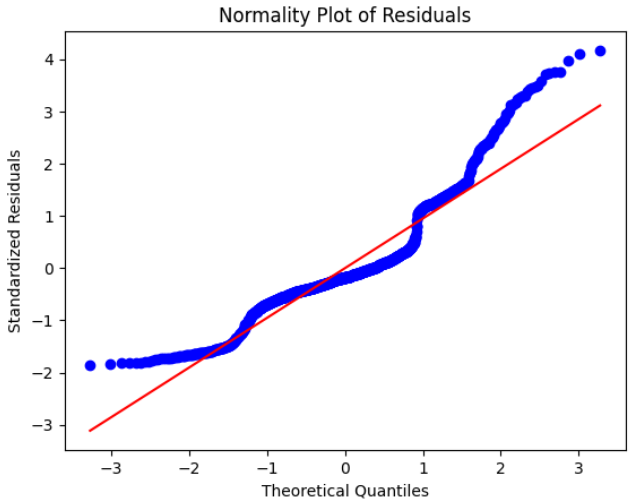
Also, the Breusch-Pagan test is performed to assess the assumption of homoscedasticity, which assumes constant variance of residuals across different values of predictor variables. The `het_breuschpagan()` function is used to calculate the test statistics and the associated p-value. A lower p-value indicates a violation of the homoscedasticity assumption,

suggesting the presence of heteroscedasticity. The p-value for our model is 3.15312532942e-17 (very small, indicating heteroscedasticity as shown in the scatter plot)

2.2- QQ Plot.

The QQ plot, or quantile-quantile plot, is a graphical tool to help us assess if a set of data plausibly came from some theoretical distribution such as a normal or exponential. The quantiles are values dividing a probability distribution into equal intervals, with every interval having the same fraction of the total population.

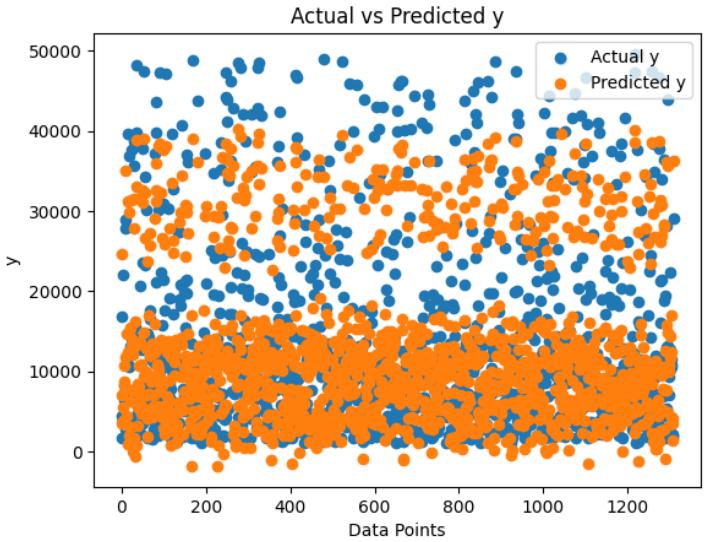
To check the quality of our linear regression model, we put an assumption that the residuals are normally distributed. When looking at the QQ plot, we see the points don't match up along a straight line which shows that the quantiles don't match that much. Although the line plotted is not a necessary component of the QQ plot, it allows the reader to visualize where the points should line up and match the base distribution.



2.3- Y predict Vs Y

Finally and most importantly, as the model as a whole is more accurate according to Root Mean Squared Error metric comparison, the resulting model can't be plotted as line (it is a function of six parameters, which means it can be plotted in a 6-D space), so using `scatter` class from `matplotlib.pyplot,` we can scatter the predicted value against the real

value and visually compare the results that show as expected, around 70% of the response values can be predicted using the model.



3- The Assessment results

The three used methods to assess the model show that it can predict the charges a person may pay knowing their BMI, age, sex, number of children, whether they are smoking or not and their region according to the dataset. However, the model's accuracy might be low according to the context of use.

Conclusion

We had a dataset that contains 7 features, 3 of which are categorical:

Smoker: Whether the person is smoking or not.

Region: The beneficiary's residential area in the US, northeast, southeast, southwest, northwest.

Sex: The insurance contractor's gender: female or male.

And 4 of which are numerical:

Age: The age of primary beneficiary

BMI: Body mass index, providing an understanding of body, weights that are relatively high or low relative to height.

Children: Number of children covered by health insurance / Number of dependents.

Charges: Individual medical costs billed by health insurance.

After exploring the data, removing the outliers, and testing the distributions of each feature, we decided that the response is the ‘charges’, thus we calculated the correlation coefficient for each feature to get insights of which feature the response is mostly dependent on it. Then to build the models, we had two choices; the first being multiple linear equations, where the slope and intercept of each feature is calculated using simple linear regression method and the second is using python standard packages to obtain the coefficients of the features in the multivariable regression model. When comparing the root mean squared metric of both models, it appears that describing the complete dataset in one model as a whole performs better. Then we performed residual analysis and comparison between the predicted value and real value to test the best model, it appears that our model can describe the dataset correctly by about 75.3%. This can be a good percentage or a bad percentage according to the context of using the model.

Members Contribution

1- Nouran Mahmoud

- In dataset analysis methods: scatter matrix.
- For the comparison between multivariable regression model and simple linear regression:

- Correlation coefficient comparison.
- Calculating mean squared error for each feature.

- The Regression model Validation tests:

- Residual analysis, Breusch-Pagen test, QQ plot and Y predicted vs Y scatter plot.

- Making conclusions based on the assessment results.

2- Bassmalla Tarek

- Applied the linear regression analysis on the response against each predictor individually:
  - Implemented from scratch the method of obtaining the regression coefficient (RC).
  - Compared the results to the case of getting the RC from standard Python packages.

3- Salma Ashraf

- Applied a multivariable linear regression analysis on the response against all the predictors simultaneously.
- Assessed statistically the multivariable regression quality in terms of The regression model as a whole using R-squared and adjusted R-squared.
- Assessed statistically the multivariable regression quality by comparing Root Mean Squared Error metric.

4- Yasmin Tarek

- Mentioned which columns are quantitative and which ones are categorical.
- Used the Z-score method to remove outliers from the data.
- Calculated a set of descriptive statistics to quantitatively describe the data.
- Standardized the features of the data
- Plotted each feature/column in the training data.
- Statistically tested if a feature/column is normally distributed.
- Computed correlation coefficients between the response variable and each predictor