Medical Insurance Costs Prediction
Final Report (Deliverable 3)
AI Wizards

Team members: Wardan Kazzalbach, Megan Nowrangi, Mary Muang, Dexter Kong, and Mohammad Haseebuddin

---

Our Motivation/Introduction:

- Understanding and predicting medical insurance costs is crucial in today's healthcare environment. We chose to do this project to make these predictions more accurate and useful for everyone involved.
- Our goal is to provide insights that could help people make better decisions about their healthcare costs. This project could even help fill in an important need, giving individuals a better understanding of financial planning and making the healthcare system fairer and easier for everyone

Problem Statement:

The objective of this project is to predict medical insurance costs, taking into account various factors such as an individual's BMI, age, gender, and more, which may all affect the total insurance cost. The dataset used for this project contains detailed information on over 1300 individuals, specifically organized to support the prediction of medical insurance costs. This data includes many key factors that are predicted to influence the overall insurance costs, making them essential for effective predictive analysis.

Detailed Breakdown of the Dataset:

- **Age**: This attribute records the age of each individual, ranging from 18 to 64 years, which provides insights into how age might correlate with medical costs.
- **Sex**: The dataset includes gender information, categorized into 'male' and 'female'. This allows for analysis of any potential differences in insurance costs based on gender.
- **BMI (Body Mass Index)**: The BMI is a critical indicator of health, calculated as a ratio of an individual's weight and height. It varies widely in this dataset, from 15.96 to 53.13, showcasing a diverse group from underweight to obese categories.
- **Children**: This indicates the number of children or dependents an individual has, ranging from 0 to 5. Including this factor helps us understand the impact of family size on insurance costs.
- **Smoker**: This binary variable identifies if the individual is a smoker, which is a significant health risk factor and therefore crucial for predicting insurance costs.

- **Region**: The dataset categorizes individuals into one of four regions: northeast, northwest, southeast, and southwest. This regional data can be used to explore geographical trends in insurance pricing.
- **Charges**: This is the primary outcome variable and represents the medical charges that the insurance covers, which range from approximately $1,122 to $63,770. This wide range reflects the variability and complexity of insurance cost prediction.

The dataset's comprehensive structure with variables like age, BMI, smoking status, and the number of children provides a rich basis for analyzing and predicting how these factors influence the cost of medical insurance. This enables a deeper understanding of the dynamics at play in insurance pricing, which is a critical aspect of healthcare economics.

Data Processing Details:

The dataset underwent several preprocessing steps to ensure it was clean and structured for a proper analysis of it. Firstly, numerical columns were found and checked for any missing values across all columns, and if found, these were filled with the mean of the respective columns to maintain data integrity. Outliers were also another area of focus; using the z-score method, any data points with a z-score greater than 3 in numerical columns were identified as outliers and removed to prevent skewing of the results. Numerical attributes like age, BMI, and charges were also normalized using the MinMaxScaler to bring them to a uniform scale of 0 to 1. Additionally, categorical variables such as sex, smoker status, and region were encoded using the LabelEncoder function, converting them into a format suitable for machine learning algorithms, where each category is represented by a unique integer. The last column, assumed to be the label, was separated from the dataset. Both the processed attributes and labels were then prepared to be returned from the function, ensuring they were ready for numerical processing in the predictive modeling. This thorough preprocessing of the data will ensure that the dataset is optimally prepared for the machine learning algorithms that we will be using.

Implementation Details:

We implemented five programs across Python and Java, each utilizing a specific regression-formed algorithm to ensure a robust analysis across the train(70%), valid(15%), and test(15%) data sets:

- **Python**: Will have the KNearestNeighbors regression algorithm, Gradient Boosting, and Multivariate Linear Regression algorithms.
- **Java**: Will have Random Forest, and Support Vector regression algorithms.

All programs shared a common or exact preprocessing function to normalize input data, remove outliers, and encode categorical variables, ensuring consistency in data handling. This was to make sure the data results were as accurate as possible.

To evaluate the regression algorithms, we used the following metrics:

- **Mean Absolute Error (MAE):** Measures the average magnitude of the errors in a set of predictions, without considering their direction. It's the average over the test sample of the absolute differences between prediction and the actual observation where all individual differences have equal weight.
- **Root Mean Squared Error (RMSE):** Measures the square root of the average of the squares of the errors. RMSE is very sensitive to outliers and gives a relatively high weight to large errors.
- **R-squared (Coefficient of Determination):** Indicates goodness of fit and explains the proportion of the variance in the dependent variable that is predictable from the independent variables. It also provides a measure of how well-unseen samples are likely to be predicted by the model.

These metrics helped us in assessing the accuracy and efficacy of the models, ensuring that our predictions were as precise and reliable as possible. We used these insights to optimize the algorithm selection and refine our predictive modeling approach to accurately forecast medical insurance costs.

Method's Explanation:

For our project, we used five methods: Multivariate Linear Regression (Python), K-Nearest Neighbors (Python), Support Vector Regression (SVR) (Java), Random Forest (Java), and Gradient Boosting (Python). These methods were implemented to ensure robust analysis and accurate predictions of medical insurance costs. By using different regression algorithms, we optimized the algorithm selection and refined our predictive modeling approach.

KNearestNeighbor's

The KNearestNeighbors algorithm estimates insurance costs by averaging the costs of the 'k' closest neighbors in the data space. This proximity is determined based on similarities in age, BMI, and other features, meaning it predicts costs based on how similar cases have been charged in the past. The K-NN algorithm uses a created function called the knnClass function, which runs the K-NN Regression algorithm from the sklearn package in Python. With this algorithm, it explored many different values of 'k', specifically k = 1, k = 3, k = 5, k = 7, and k = 9, to determine the impact of the number of neighbors on the predictions. The function fits the K-NN model to the training data and then predicts values for the validation and test datasets, allowing us to evaluate the model's performance with different 'k' values. This

approach helps us understand how the choice of 'k' affects the accuracy of the predictions and ensures that our model captures the relationships in the data effectively.

Gradient Boosting

Gradient Boosting is an algorithm that combines weak learners, or trees that perform poorly into a strong tree. It starts with a single stump, usually the average value of what is being predicted in the dataset then builds additional trees based on the errors of the previous tree and scales each tree by the learning rate. This process repeats until the requested number of models are made or additional trees fail to improve the fit. A process known as Hypertuning is used to modify the parameters of the algorithm for more accurate results. The three parameters I used to try and find the best model, Max Depth, n_estimators, and learning rate. With the GridSearchCV function from sklearn, I was able to analyze the r2 scores of many parameter combinations and pick out one with the highest r2 score. For the initial run of the algorithm, I did not use hypertuning. Starting from the second run, I modified n_estimators and learning rate. Starting from the third run, I also modified max_depth. An overview of each variable is below

- n_estimators is a large integer that determines how many times the gradient boosting process repeats.
- Learning rate is a number between 0 and 1 multiplied to contributions from each tree in the model's calculation. According to Jerome Friedman, the guy that invented Gradient Boosting, empirical evidence suggests that if you find the right learning rate, you tend to get lower variance in predictions.
- Max Depth is an integer that limits the nodes of each tree.

Support Vector Regression

A support vector regression constructs a hyperplane or set of hyperplanes in a high-dimensional space to predict costs. The best hyperplane is the one that has the maximum margin from the nearest points in the training data, effectively handling outliers and ensuring that variations in data like extreme costs do not unduly influence the overall model. The main method serves as the entry point of the program, working out the entire process from data loading to model evaluation. It starts by loading the training, testing, and validation datasets using the loadData method, which leverages the CSVLoader class to read CSV files and return the data as Instances objects. Once the data is loaded, the preprocessData method is called to normalize the numeric attributes and set the class index for each dataset. This method utilizes the Normalize filter to ensure that all features are on the same scale, which is crucial for Support Vector Regression (SVR). With the preprocessed data, the program then initializes and configures an SVR model using the SMOreg class and specific options for the kernel and other parameters. The model is built with

the training dataset by calling the buildClassifier method. To evaluate the model's performance, the Evaluation class is used to assess the model on both the validation and test datasets. The results are printed to the console, providing a summary of the model's accuracy and other evaluation metrics. Overall, this process ensures that the SVR model is properly trained and evaluated, demonstrating its ability to predict costs accurately while handling variations and outliers effectively.

Multivariate Linear Regression Model

The class of linear functions is used for continuous-valued inputs. Multivariate Linear Regression predicts outcomes, y, utilizing multiple inputs, x. Weights are added to these values in order to form a linear function that best fits the data points. This function is called "linear regression." In order to determine a line of best fit, it determines the values of the weights that will minimize the empirical loss. Traditionally, the squared-error loss function, L2, is used. Regarding this model in my code, I imported the LinearRegression class from sklearn's linear_model module. I then instantiated the Linear Regression class, and fit the model with the data so that it could determine the optimal weights to minimize the loss function. Next, I used the model to make predictions and evaluated those predictions using MAE, RMSE, and R-Squared. I printed those results.

Random Forest

A random forest model builds multiple decision trees each based on a random subset of features and data points, then averages their predictions to estimate insurance costs. This method reduces overfitting and is good at capturing nonlinear relationships between features and costs without explicit modeling. The main components of the random forest code include data loading, decision tree construction, random forest aggregation, and performance evaluations. The main method, which serves as the start of the program, houses constants such as the selection of the number of trees and features for the random forest, the loading, and extraction of the datasets, and loops that evaluate and print the random forest results from the datasets. Multiple methods are created to calculate each of the metrics and impurity, predict nodes, and create and build the decision tree. Other classes such as the node and decision trees were formed to represent a node, which can be an internal node with children or a leaf node with a value, and build decision trees using recursive splitting based on impurity reduction. To reduce variance each time the program was run, a for loop (in the main method) was created to take the results of seven iterations and average them to produce one single result for each of the datasets. The final result is calculated by using the number of trees and features (which are randomized).

Results/Explanation:

We employed the five methods to analyze the train (70%), validation (15%), and test (15%) data sets, ensuring a comprehensive evaluation of the models. By using these regression algorithms, we aimed to achieve robust and accurate predictions of medical insurance costs. The evaluation metrics, including mean absolute error (MAE), root mean squared error (RMSE), and R-squared, were used to assess the accuracy and efficacy of each model, providing insights that guided the optimization and refinement of our predictive modeling approach.

KNearestNeighbor's

The results of the KNearestNeighbor's (K-NN) regression algorithm applied to our dataset revealed significant insights into the model's performance across different values of 'k'. Here are the detailed results:

For **K = 1**:

- Training MAE: 0.0587
- Training MSE: 1.396
- Training R-Squared: 0.9999
- Validation MAE: 343.074
- Validation MSE: 419.374
- Validation R-Squared: -50.7239
- Test MAE: 333.153
- Test MSE: 409.436
- Test R-Squared: -48.3016

For **K = 3:**

- Training MAE: 119.484
- Training MSE: 166.167
- Training R-Squared: 0.6226
- Validation MAE: 335.660
- Validation MSE: 379.784
- Validation R-Squared: -41.4191
- Test MAE: 336.696
- Test MSE: 384.427
- Test R-Squared: -42.4628

For **K = 5**:

- Training MAE: 137.763
- Training MSE: 184.020
- Training R-Squared: 0.5372
- Validation MAE: 334.224
- Validation MSE: 363.463
- Validation R-Squared: -37.8516
- Test MAE: 335.803

- Test MSE: 373.431
- Test R-Squared: -40.0120

For **K = 7**:

- Training MAE: 147.798
- Training MSE: 193.505
- Training R-Squared: 0.4882
- Validation MAE: 343.101
- Validation MSE: 368.469
- Validation R-Squared: -38.9293
- Test MAE: 335.862
- Test MSE: 367.296
- Test R-Squared: -38.6754

For **K = 9:**

- Training MAE: 154.728
- Training MSE: 199.370
- Training R-Squared: 0.4567
- Validation MAE: 347.299
- Validation MSE: 368.067
- Validation R-Squared: -38.8422
- Test MAE: 349.128
- Test MSE: 375.176
- Test R-Squared: -40.3961

Looking into the results, it appears that K = 7 provides the lowest errors and the least negative R-Squared, suggesting it may be the optimal choice among the tested values. However, all models exhibited poor performance with negative R-Squared values, indicating that the K-NN algorithm might not be suitable for our dataset. These results suggest that a reevaluation of the model choice, feature engineering, or additional data preprocessing might be necessary to improve the performance of this algorithm.

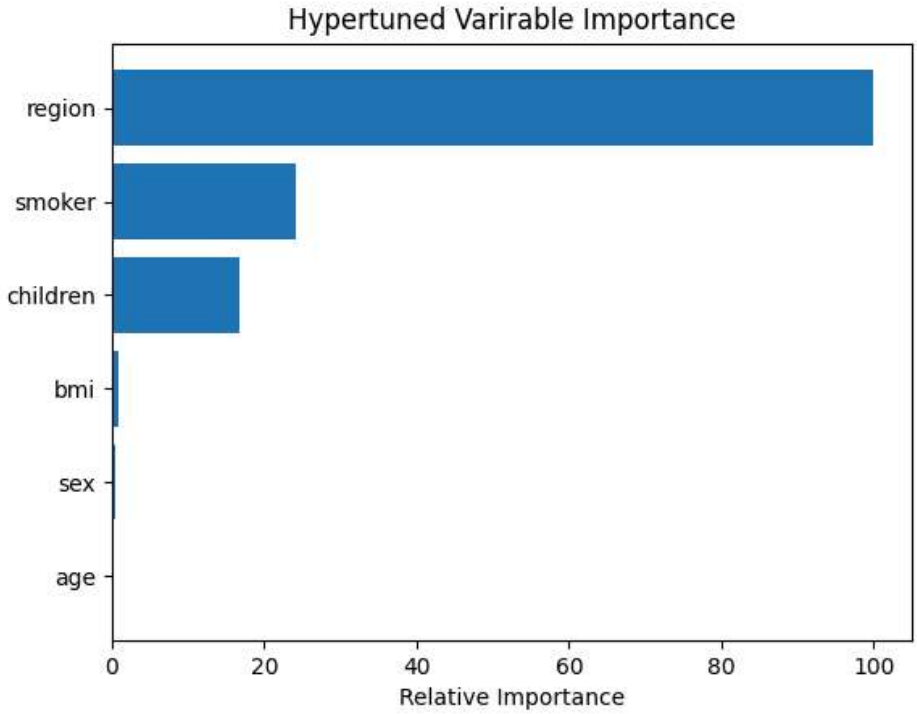
Gradient Boosting

Because my research suggested that I should attempt to hypertune with more values, I manually ran the algorithm with a variety of different parameters. A table of my findings is below.

Run ID	learning_rate	n_estimators	max_depth	Optimal learning_rate	Optimal n_estimators	Optimal max_depth	R2 score
Initial	3	3	2	N/A	N/A	N/A	0.8184
2	0.15,0.1,0.10,0.05	100,150,200,250	N/A	0.05	100	N/A	0.8491
3	0.15,0.1,0.25,0.05,0.75	1000,1500,2000,2500,3000	2, 4, 6, 8, 10	0.5	1000	2	0.8272

4	0.3,0.4,0.45,0.50,0.55	40,60,80,100,120	2, 4, 6, 8, 10	0.3	40	2	0.8464
5	0.2,0.25,0.3,0.35,0.4	20,30,40,50,60	2, 4, 6, 8, 10	0.3	20	2	0.8498
6	0.2,0.25,0.3,0.35,0.4	1,5,10,15,20	2, 4, 6, 8, 10	0.4	15	2	0.8498
7	0.2,0.25,0.3,0.35,0.4	13,14,15,16,17	2, 4, 6, 8, 10	0.4	15	2	0.8498

The initial and second runs are test runs for baseline values. It should be noted that max_depth was static for the initial run and not hypertuned for the second run for the purposes of saving time. The third run was a test to see if the suggestion from my research that larger values may result in a better r2 score was correct. For each run after the third run, I attempted to raise the r2 score by inputting a range of values where the median is the optimal value from the previous run. At the end of my runs, I found optimal parameters that led to an accuracy of 85%, which I believe to be an acceptable degree of accuracy. Using the optimal parameters, I used feature_importance_ from sklearn to find which variable from the dataset had the most impact on predicting health insurance prices. I made a bar graph of importances based on max importance. As you can see from the graph below, I found that the patient's region has the most impact on predicting health insurance costs. If they were a smoker or had children it would have some impact. BMI and sex have negligible impact. Age would have no impact.



I'd also like to say from my research and experience working with this algorithm that gradient boosting has its advantages and disadvantages. The advantage is that because it corrects mistakes, it tends to have good accuracy. However, It is space and time complex because the process of correcting trees requires the creation of a lot of trees for best results. On top of that, hypertuning is required for best results, which means going through the process of making/correcting trees for every permutation of parameters. Hypertuning took upwards of 20 minutes when calculating the optimal values for dataset 3. I do believe though that the advantage of consistently good accuracy is worth the time required to hypertune. Effort can be reduced as well by automating the manual process of selecting better variables that I used.

Support Vector Regression

The results of our Support Vector Regression (SVR) model across different datasets reveal important insights into its performance. For the test dataset, the model achieved a Mean Absolute Error (MAE) of 2934.13, a Root Mean Squared Error (RMSE) of 4214.26, and an R-squared value of 0.925. This indicates a strong predictive capability, with the model explaining 92.5% of the variance in the insurance costs. For the validation dataset, the MAE was 4565.77, the RMSE was 5645.38, and the R-squared remained at 0.888, suggesting that the model somewhat maintains its predictive accuracy even on unseen data. However, the higher errors compared to the test set imply a slightly lower precision when generalizing to new data. On the training dataset, the MAE was 7535.58, the RMSE was 5385.68, and the R-squared was also .675. The higher errors here could be attributed to overfitting, where the model fits the training data too closely, capturing noise along with the underlying patterns. Interpretation and Relation Between Metrics The MAE and RMSE values indicate the average error magnitude in our predictions, with the RMSE being more sensitive to larger errors due to its squaring of differences. The lower MAE and RMSE in the test set compared to the validation set suggest that while the model is generally accurate, it may face challenges with more varied or unseen data, as evidenced by the increased errors in the validation set. The consistent drop R-squared value across all datasets demonstrates that the model is not being consistent explains a significant proportion of the variance in insurance costs. An MAE of less than $1,500 and an RMSE less than $2,500 might be more ideal targets, ensuring that the model is highly accurate but also robust enough to handle occasional outliers effectively. These metrics collectively highlight the model's strength in predictive accuracy, while also pointing to areas for potential improvement, such as addressing overfitting and enhancing generalization to more diverse datasets.

Multivariate Linear Regression Model

Printed below are the results for the Multivariate Linear Regression Model:

Training:

Mean Absolute Error: 0.08498246334957187

Root Mean Squared Error: 0.34919807958475174

R-Squared: 0.7474907741319463

Validation:

Mean Absolute Error: 0.08894104546596247
Root Mean Squared Error: 0.36645401462522825
R-Squared: 0.6819371333641144

Testing:

Mean Absolute Error: 0.08077760166003213
Root Mean Squared Error: 0.3355333099448997
R-Squared: 0.8136396878372192

The MAE and RMSE are expressed in the same units as the original data (dollars), whereas the R-Squared is a proportion value.

Across all of the sets, the average MAE is about 0.0849, ranging from 0.081 to 0.089, meaning that the predicted healthcare costs deviate from the actual healthcare costs by about $84.90 on average. This is a surprisingly low deviation, considering that the healthcare charges from the given data range from $1,120 to $63,800. This suggests the model is fairly accurate and reliable.

The RMSE penalizes errors more harshly, and averages at about 0.35039, or about $350.39, across all data sets. It ranges from about $349.20 to $366.45. This is also relatively small, considering the original range of the healthcare charges, though not as small as the MAE. This larger RMSE indicates that although the model is generally reasonably accurate, there are occasionally predictions that contain more significant errors.

The R-Squared values across the sets range from 0.682 to 0.814, with an average of about 0.748. This means that, on average, about 74.8% of the healthcare costs are captured by the model. The low range in values indicates that the model can perform consistently. This value indicates that the model is generally a good fit, reliable, and accurate.

Overall, while not perfect, this model still generally fits the data and generalizes well to unseen data. Points to improve upon include reducing the RMSE and closing the gap between the Training and Validation evaluations. Fixing these would improve the model's accuracy and reduce its prediction errors. Nevertheless, this model can still reasonably be relied upon to predict healthcare costs based on the input factors, though with about a $350 margin of error in mind during more sensitive calculations.

Random Forest

Here are the detailed results of the random forest model at the optimal parameters of five trees and three features:

Test Data - Mean Absolute Error: 0.8690442010068166
Test Data - Root Mean Squared Error: 1.0489960034565642
Test Data - R-squared: -15.650212683499083

Train Data - Mean Absolute Error: 0.8305344340136813
Train Data - Root Mean Squared Error: 1.0252544813801474
Train Data - R-squared: -17.24816779986565

Valid Data - Mean Absolute Error: 0.7921530642044535

Valid Data - Root Mean Squared Error: 0.9743773676734236
Valid Data - R-squared: -16.219285380817812

In comparison to the other models, the MAE and RSME are very low and suggest that the model's predictions are close to the actual costs. Taking into consideration the range of insurance charges, the average MAE for all of the datasets is 0.83 which means that the model's predictions are off by about $830. The average RSME for all of the datasets is 1.01 which means that the model is off by about $1,010. The calculations of the MAE and RSME reveal that the model performed well. However, the same cannot be said for the R-squared values. The R-squared values are big negative numbers which indicates that the model performed worse than just using the average cost (a simple horizontal line). One reason why the values are negative is random noise in which the features (which are randomized) used to calculate the results might not have a strong relationship with insurance cost and in turn, capture only the random noises from the data leading to predictions that are far from the actual cost. Another reason why the R-squared values are negative may be due to overfitting. The model may have memorized specific patterns in the training set that don't generalize well to unseen data like the test dataset. This may have led to inaccurate predictions causing the R-squared values to become negative. The parameters that performed the best and most accurate predictions were five trees and three features. An interesting finding is that the MAE and RMSE values increased as the number of trees and features also increased; however, the R-squared values decreased resulting in larger negative values. In conclusion, the random forest model performed well in terms of the MAE and RSME but performed poorly in terms of R-squared. The model may not be ideal for predicting insurance costs since the negative R-squared values are not effective in capturing relationships between features and costs.

Conclusion:

In this project, we aimed to predict medical costs based on variables like age, sex, BMI, and more, to assist in healthcare decision-making. Our goal was to provide insights that could help individuals better plan their finances and navigate the healthcare system more effectively.

Key Findings:

1. Gradient Boosting:
    a. **Performance**: This method had the highest R-squared value, averaging 0.849, indicating it was the best at predicting healthcare costs.
    b. **Insights**: The feature importance analysis revealed that a patient's region was the most significant factor, followed by smoking status and the number of children. BMI and sex had minimal impact, while age had none.
    c. **Advantages and Disadvantages**: Gradient Boosting corrects its mistakes, leading to

good accuracy but is time and space-intensive due to the creation of numerous trees. Hypertuning for optimal results also requires substantial time.

2. Support Vector Regression (SVR):
    a. **Performance**: SVR had a high average R-squared value of 0.829, but it had the highest MAE and RMSE, indicating larger prediction errors compared to other models.
    b. **Insights**: The model showed strong predictive capability but struggled with generalizing to new data, suggesting overfitting issues.
3. Multivariate Linear Regression:
    a. **Performance**: This method had an average R-squared value of 0.747, indicating it was reliable and consistent in predicting healthcare costs.
    b. **Insights**: The average deviation in predicted costs was about $84.90, and the model captured about 74.8% of the variance in healthcare costs.
4. KNearestNeighbors (K-NN):
    a. **Performance**: K-NN performed poorly with negative R-squared values, indicating it was not suitable for our dataset.
    b. **Insights**: Different values of 'k' were tested, but none provided satisfactory results, suggesting the need for reevaluation of the model choice or additional data preprocessing.
5. Random Forest:
    a. **Performance**: Random Forest had low MAE and RMSE values but negative R-squared values, indicating poor model fit.
    b. **Insights**: The model captured random noise rather than meaningful patterns, leading to inaccurate predictions. Overfitting was also a significant issue where the model performed well on training data but poorly on the other datasets.

Overall Assessment:

Gradient Boosting and Multivariate Linear Regression emerged as the best models for predicting healthcare costs. SVR, while having high predictive accuracy, struggled with large prediction errors. K-NN and Random Forest were not well-suited for our dataset.

Future Work:

Our work successfully predicted healthcare costs to a decent degree, but there is room for improvement. Future work could involve further refining the models, exploring other algorithms, or enhancing data preprocessing techniques. By improving these aspects, we hope to provide even more accurate cost predictions, helping individuals make better-informed healthcare decisions and contributing to a fairer and more transparent healthcare system.

Limitations/Challenges:

- Five members are using separate coding programs for five different algorithms
- Three members used Python; two used Java
- There were some integration issues between Python and Java. (We were not sure how to midgate our work between each other)
- We had issues with importing packages into our program, like pandas (python) and weka (java)
- Megan faced many issues when using Java to code the Multivariate Linear Regression Model. After many troubleshooting steps, they ultimately had to switch to coding in Python

Student roles and contributions:

Project Manager: Wardan Kazzalbach (KNearestNeighbors)

- Scheduled weekly team meetings and had check-ins to track the progress in the project
- In Python, splitted the data properly into (training (70%), valid (15%), test (15%)) data sets
- Reviewed and researched on the KNearestNeighbors regression model in the python sklearn library
- Worked on the KNearestNeighbors regression algorithm in python
- Presented the slides for the KNearestNeighbors regression algorithm

Lead Configuration Manager: Dexter Kong (Gradient Boosting)

- Reviewed and researched Gradient Boosting
- Worked on the Gradient Boosting Regression algorithm in python
- Helped with project material organization
- Provided insight into answering the problem statement
- Presented the slides for Gradient Boosting in Python
- Wrote the sections of the report pertaining to Gradient Boosting

Deployment Engineer: Mohammad Haseeb (Support Vector Regression)

- Reviewed and research Support Vector regression
- Coded the vector regression in Java using weka
- Researched on how to create a hyperplane and some graphs to understand the Vector Regression
- Presented the slides for SVR and Multivariate Linear Regression Model

Data Preprocessing Specialist: Megan Nowrangi (Multivariate Linear Regression Model)

- Reviewed and researched coding for Data Preprocessing of the dataset's variables
- Researched other libraries that could be used with Java to carry out all the tasks
- Researched on Multivariate Linear Regression Model in Weka library
- Reviewed Python
- Researched and coded Multivariate Linear Regression Model with Python

Domain Expert: Mary Muang (Random Forest)

- Coded the algorithm for the random forest in Java
- Presented slides for the random forest sections of the presentations
- Researched how certain attributes can affect potential prediction
- Investigated methodologies for achieving precise results due to the large variance in metric values

Acknowledgments:

This project utilizes the data from the 'insurance.csv' dataset. Special thanks to the original creators and providers of this dataset. The dataset has been invaluable in demonstrating various regression algorithms, including Multivariate Linear Regression, K-Nearest Neighbors, Support Vector Regression (SVR), Random Forest, and Gradient Boosting. The successful completion of this project is attributed to the use of this great dataset and the effective implementation of these algorithms.

References:

https://www.kaggle.com/datasets/joebeachcapital/medical-insurance-costs/data
https://spark.apache.org/docs/latest/api/java/org/apache/spark/ml/feature/MinMaxScaler.html
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
https://www.geeksforgeeks.org/how-to-split-a-dataset-into-train-and-test-sets-using-python/
https://www.kdnuggets.com/2019/07/data-pre-processing-optimizing-regression-model-performance.html
https://ai.plainenglish.io/understanding-common-regression-evaluation-metrics-mae-mse-rmse-r2-and-adjusted-r2-6c5709e614c4
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html#sklearn.neighbors.KNeighborsRegressor.kneighbors
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html
https://machinelearningmastery.com/use-regression-machine-learning-algorithms-weka/
https://weka.sourceforge.io/doc.dev/weka/classifiers/functions/SMOreg.html
https://www.codeproject.com/articles/566326/multi-linear-regression-in-java
https://griddb.net/en/blog/how-to-implement-a-random-forest-algorithm-in-java/
https://likegeeks.com/downloading-files-using-python/