A comparative study of R and Python for Implementing Machine Learning Algorithms

Roli Arah
Independent Research
July 2023

I. Introduction:

In today's data-driven world, Machine Learning (ML) has become a valuable tool. Machine Learning allows computers to learn from data, helping them make decisions or predictions without the need for explicit programming. This technology powers various applications, from suggesting products in online shopping to detecting fraudulent transactions.

R and Python are two tools that are widely used in the machine learning field. R, which was designed for statistical analysis, has become a favorite among researchers and statisticians due to its robust package ecosystem and flexible syntax. On the other hand, Python, known for its simplicity and readability, is broadly used in diverse areas including web development, data analysis, and machine learning. Its wide range of libraries, like NumPy, pandas, and TensorFlow, make Python a preferred language for many data scientists.

Despite their broad acceptance and important roles in machine learning, one question remains: Which of these two, R or Python, is better for implementing machine learning algorithms? Numerous debates and studies have tried to answer this question, but there hasn't been a clear consensus. The choice of programming language can significantly influence the process, efficiency, and ultimately the success of a machine learning project.

The aim of this research is to provide a comparison of R and Python in the context of machine learning. The central question guiding this study is: "How do R and Python compare when used for implementing Machine Learning algorithms?" In pursuit of this question, the research intends to explore the unique features, strengths, and limitations of both languages, particularly regarding their application in machine learning.

The study aims to provide practical insights that can act as a guide in choosing the right language for specific tasks. By evaluating the performance of common machine learning algorithms like Decision Trees, Random Forests, and Neural Networks in R and Python, this study provides a practical comparison that complements the theoretical discussions about these languages.

The significance of this research lies in the critical role of machine learning in modern data analytics and the increasing use of data for decision-making in various sectors like business, government, and healthcare. This research aims to provide an objective comparison between R and Python for implementing machine learning algorithms by examining the strengths and limitations of both programming languages.

II. Literature Review:

The discourse around the optimal programming language for machine learning is a prevalent topic, particularly the comparison between R and Python. As leading languages in the domain of data science, both R and Python have been explored in scholarly research, and their unique strengths and limitations recognized.

R originated as an open-source implementation of the S programming language, developed at Bell Laboratories in the 1970s by John Chambers and his colleagues. It was created as a tool for statistical computing and graphics, with a focus on providing a rich set of statistical functions and visualizations. R was designed to be an interactive programming language specifically tailored for data analysis and research in statistics. Its rich package ecosystem comprises of packages like ggplot2 for data visualization, caret for predictive modeling, and dplyr for data manipulation which equip users with powerful tools for data analysis (Wickham, 2016; Kuhn, 2008; Wickham et al., 2020). Furthermore, advanced features for statistical modeling and high-quality graphics make R a popular choice for statistical analysis and reporting (R Core Team, 2020). However some feel the academic orientation and statistical complexity of R can pose a steep learning curve for beginners, especially those lacking a solid foundation in statistics (Bryer & Speerschneider, 2016). There have also been concerns about R's efficiency in handling large datasets, a critical factor for machine learning applications.

On the other hand, Python was created by Guido van Rossum in the late 1980s as a successor to the ABC programming language. Guido wanted to develop a language that was easy to read, write, and understand, with a focus on code simplicity and readability. Python's straightforward syntax and readability make it a popular choice in programming and data science (Millman & Aivazis, 2011). In the context of machine learning, Python's extensive libraries, such as NumPy for numerical computation, pandas for data manipulation, and Scikit-learn, TensorFlow, and PyTorch for machine learning and deep learning applications, make it an excellent choice for diverse applications (Pedregosa et al., 2011; TensorFlow, 2015; Paszke et al., 2019). R is often favored by researchers for complex statistical analysis as it is believed to have extensive and sophisticated statistical capabilities, which are considered more advanced than those of Python (Virtanen et al., 2020).

The performance of machine learning algorithms in R and Python is a frequent subject of study. Comparative studies on the performance of machine learning algorithms in R and Python have been conducted. There have been observations that for traditional machine learning algorithms such as decision trees and random forests, the differences in performance between R and Python are generally negligible (Probst, Boulesteix & Bischl, 2019). However, when it comes to more advanced models like neural networks and deep learning, Python's comprehensive libraries such as TensorFlow and Keras have gained popularity due to their extensive support and large user communities (Chollet et al., 2015).

Comparing the interface consistency among different machine learning algorithms, Python's Scikit-learn and R's caret packages are commonly highlighted (Kuhn, 2008; Buitinck et al., 2013). However, Scikit-learn, though highly intuitive and consistent, may not offer as broad a range of algorithms as R's caret, as suggested by some researchers (Waskom et al., 2020).

In the realm of data visualization, R's ggplot2 is frequently compared to Python's matplotlib and seaborn. R's ggplot2 tends to receive favorable reviews due to its superior versatility and the aesthetic quality of its outputs (Wickham, 2016; Hunter, 2007; Waskom et al., 2020).

The ultimate choice between R and Python often hinges on project-specific needs, the user's or team's proficiency with the language, and the requirement for particular libraries or packages. The rise of "polyglot" data scientists, proficient in both R and Python, reflects the trend of selecting the most suitable language for a specific task (Ram, 2020). Tools like Rpy2 and reticulate, which allow interaction between R and Python in a single workspace, further exemplify this trend (RPy2 developers, 2020; Allaireet al., 2020).

Community support and the availability of learning resources for both languages also play crucial roles in choosing between R and Python. Python's broad application beyond data science has given rise to a larger user community and a wider variety of learning resources (Python Software Foundation, 2021). This larger community often translates into quicker support through platforms like Stack Overflow. On the other hand, the R community, though smaller, is highly specialized and academically oriented, often providing high-quality packages and resources for statistical learning (R Core Team, 2020).

The nuances in choosing between R and Python for machine learning applications are numerous and heavily influenced by various factors. This research aims to address this by offering a comparison of R and Python across different machine learning algorithms. The landscape of R and Python in the context of machine learning is continuously evolving, with new tools, packages, and updates to the core languages regularly being introduced. This fluidity keeps the decision-making process dynamic and subject to change over time.

Python's ease of integration with emerging technologies has boosted its popularity in sectors like AI and IoT, partly due to its robust libraries like TensorFlow and PyTorch that support advanced machine learning and deep learning implementations (Sweigart, 2020). Research indicates Python's popularity is on the rise, especially in the tech and start-up sectors, due to its versatility, capability for web development, automation tasks, and data analysis (Piatetsky, 2020).

As mentioned previously, R is often the preferred choice among statisticians and data scientists working with complex statistical models (Muenchen, 2018). The ongoing efforts of the R Consortium and the Comprehensive R Archive Network (CRAN) in developing new packages enhance its statistical and graphical capabilities further, solidifying R's position in the data science ecosystem.

Despite the many advantages, there are critiques for both languages. Some researchers suggest that Python's growing popularity could lead to a fragmented ecosystem, potentially causing confusion among users (Hsu, 2020). R, on the other hand, has faced criticism for its memory management and speed issues when dealing with large datasets, a significant concern in machine learning applications (Zhang, 2016).

In summary, scholarly research presents a rich debate on the comparative use of R and Python for machine learning, highlighting factors such as technical functionality, ease of use, community support, and suitability for specific use cases. This research aims to delve into some of these considerations, providing a guide to comparison.

By integrating the insights from existing literature and adding empirical analysis, this research aims to be a valuable resource for comparing of both languages.

III. Methodology:

a) Introduction to the Insurance Dataset
b) Nature of the Dataset
c) Evaluation Metrics
d) Choice and description of Algorithms
e) Implementation in R and Python

a. Dataset Source: The insurance dataset used in this study is sourced from Kaggle, a reputable online community and platform for data science and machine learning enthusiasts. The specific dataset, titled "US Health Insurance Dataset," is publicly available and can be accessed at the following URL: https://www.kaggle.com/datasets/teertha/ushealthinsurancedataset

Kaggle is known for hosting a vast collection of high-quality datasets contributed by data scientists and researchers from various domains. The availability of this insurance dataset on Kaggle ensures its reliability, quality, and suitability for exploring health insurance patterns and building predictive models.

Dataset Description: The US Health Insurance Dataset is a comprehensive collection of health insurance information for individuals in the United States. It encompasses a wide range of attributes that provide valuable insights into the factors influencing insurance coverage and associated charges.

The dataset comprises 1,338 rows, with each row representing a unique individual's insurance profile. These profiles capture diverse information, including demographic characteristics, lifestyle factors, and geographic details. By examining these attributes, we can gain a deeper understanding of the factors that contribute to differences in insurance charges among individuals.

The dataset has the following features:

- Age: This variable represents the age of each individual and serves as a key demographic factor that impacts insurance coverage and charges.
- Sex: The gender of the insured person is recorded as either male or female, reflecting gender-based differences in insurance patterns.
- BMI (Body Mass Index): BMI is a calculated metric based on an individual's height and weight, providing insights into their overall health and potential risks associated with insurance coverage.

- Children: This variable captures the number of children covered by each insurance policy, reflecting the impact of family size on insurance charges.
- Smoker: This binary variable indicates whether an individual is a smoker or a non-smoker, as smoking status often influences insurance rates due to associated health risks.
- Region: The dataset includes a region variable, categorizing the insured individuals into four distinct regions within the United States: northeast, southeast, southwest, and northwest.
- Charges: The insurance charges associated with each individual's policy are recorded, representing the financial aspect of insurance coverage.

These attributes collectively offer a comprehensive view of health insurance patterns, enabling us to analyze the relationships between several factors and their influence on insurance charges- the target variable.

b. Nature of the Dataset: The US Health Insurance Dataset exhibits a rich mixture of numerical and categorical variables, making it suitable for a comprehensive analysis using statistical and machine learning techniques. The inclusion of numerical variables such as age, BMI, number of children, and charges allows for quantitative analysis and modeling. Meanwhile, the categorical variables, including sex, smoker status, and region, provide qualitative information that can be encoded appropriately for analytical purposes.

With 1,338 rows, the dataset provides a substantial sample size to draw meaningful insights and develop accurate predictive models. This sample size ensures that the dataset is representative of the target population, allowing for reliable generalization and analysis of insurance patterns.

By leveraging the wealth of information contained within this extensive insurance dataset, we aim to delve deep into the factors influencing insurance charges and compare the performance of various machine learning algorithms implemented in both R and Python. The dataset's attributes and adequate sample size facilitate an exploration of the relationships between unique features, enabling us to uncover valuable insights in the field of health insurance analysis.

c. Evaluation Metrics:

To assess the performance and effectiveness of the machine learning algorithms in predicting insurance charges based on the numerical features in the dataset, we will employ several commonly used evaluation metrics. These metrics will allow us to quantitatively measure the accuracy, precision, recall, and overall predictive power of each algorithm. The following evaluation metrics are well-suited for numerical data and are commonly used in regression and numerical prediction tasks:

1. Root Mean Squared Error (RMSE): RMSE is a widely used metric that measures the average deviation between the predicted and actual insurance charges. It calculates the

square root of the average of the squared differences between predicted and actual values. A lower RMSE indicates better predictive accuracy, as it reflects smaller prediction errors.

2. R-Squared (R^2 for the linear regression): R-squared is a statistical measure that indicates the proportion of the variance in the insurance charges that can be explained by the predictive model. It represents the goodness-of-fit of the model, with higher values (closer to 1) indicating a better fit. R-squared ranges from 0 to 1, where 1 represents a perfect fit.
3. Mean Absolute Percentage Error (MAPE): MAPE is a commonly used metric for measuring the percentage deviation between the predicted and actual insurance charges. It provides a relative measure of the prediction errors, making it useful for understanding the accuracy in terms of percentage. MAPE is calculated as the average of the absolute percentage differences between predicted and actual values. Suitable Machine Learning Algorithms: Based on the nature of the numerical dataset and the task of predicting insurance charges, several machine learning algorithms are well-suited for this research.

Other non-quantitative comparisons include ease of implementation, code readability, length, availability of libraries and visualization capabilities.

d. Based on the nature of the numerical dataset and the task of predicting insurance charges, the following algorithms will be considered:

1. Linear Regression (LR):

Linear Regression is a widely used algorithm for regression tasks, particularly suited for problems where the target variable is continuous and there exists a linear relationship between the input features and the target. LR aims to estimate the coefficients of a linear equation that minimizes the sum of squared differences between the predicted and actual values. It assumes that the relationship between the features and the target variable can be approximated by a linear function.

LR offers several advantages, including interpretability, computational efficiency, and the ability to handle many features. It provides valuable insights into the importance and contribution of each feature in predicting the target variable. Furthermore, LR is simple to implement and understand, making it an excellent choice for initial exploratory analysis and baseline modeling.

2. Decision Trees:

Decision Trees are versatile and interpretable algorithms that make predictions by recursively partitioning the data based on feature values. Each internal node of the tree represents a decision based on a specific feature, and each leaf node represents a predicted value or class. Decision Trees can capture non-linear relationships, handle numerical and categorical features, and automatically manage feature selection.

One of the main advantages of Decision Trees is their ability to provide transparent and interpretable decision rules, allowing us to understand the underlying decision-making process. They are particularly useful for feature selection, as they can identify the most informative features for predicting the target variable. Decision Trees are used for both classification and regression tasks, making them versatile for different types of problems.

3. Random Forest (RF):

Random Forest is an ensemble learning algorithm that combines multiple decision trees to make predictions. It is particularly effective in handling complex relationships and interactions among features. RF works by constructing a multitude of decision trees, where each tree is trained on a random subset of the data and predictions are made by aggregating the predictions of all individual trees.

RF offers several advantages, including high predictive accuracy, robustness against overfitting, and the ability to manage high-dimensional data. It is capable of capturing non-linear relationships and interactions among features, making it suitable for complex datasets. RF also provides feature importance measures, which enable users to identify the most influential features in predicting the target variable.

4. Neural Networks (NN):

Neural Networks are powerful algorithms inspired by the structure and functioning of the human brain. They consist of interconnected layers of nodes (neurons) that learn to represent complex relationships in the data. NNs can capture intricate patterns and non-linear relationships between features and the target variable.

NNs are highly flexible and capable of handling complex and high-dimensional data. They excel in tasks where the relationships between features and the target are non-linear and require advanced modeling capabilities. However, NNs require a larger amount of training data and computational resources compared to other algorithms. They also require careful selection of architecture, activation functions, and optimization parameters to achieve optimal performance. While they can be very efficient interpretability is not simple.

By employing these machine learning algorithms in both R and Python, we can explore their strengths, weaknesses, and suitability for predicting insurance charges based on the numerical and categorical features in the dataset. We will analyze their performance using evaluation metrics such as root mean squared error (RMSE), mean absolute percentage error (MAPE), R-squared, execution time, as well as considerations of ease of implementation, code readability, length, availability of libraries and visualization capabilities.

e. Implementation of Linear regression in R and in Python

Step	**Description**	**Libraries Used**
1. Data Preprocessing	The insurance dataset is loaded, and categorical variables (sex, smoker, region) are converted to a suitable format for modeling via one-hot encoding.	R: tidyverse (read.csv, as.factor), fastDummies (dummy_cols) Python: pandas (read_csv, get_dummies)
2. Dataset Split	The dataset is divided into a training set (60% of data) and a test set (40% of data). This allows us to train our model on one portion of the data and then assess its performance on unseen data.	R: base R (set.seed, sample) Python: sklearn.model_selection (train_test_split)
3. Linear Regression Model	A linear regression model is fit to the training data, with 'charges' as the target variable and all other variables as predictors. This model learns the relationship between charges and the predictor variables.	R: base R (lm, summary) Python: sklearn.linear_model (LinearRegression)
4. Prediction and Evaluation	Predictions are made on the test set using the trained linear regression model. Evaluation metrics (RMSE and MAPE and R squared) are calculated to assess the model's performance.	R: Metrics (rmse, mape), base R (predict) Python: sklearn. metrics(mean_squared_error) ,numpy (sqrt, mean, abs)
5. Visualize results	Residuals (the difference between actual and predicted charges) are calculated and visualized against predicted charges. This helps us understand the accuracy and error distribution of our model.	R: ggplot2 (ggplot, aes, geom_point, geom_abline, geom_hline, ggtitle, xlab, ylab) Python: matplotlib.pyplot (scatter, axhline, xlabel, ylabel, title, show)
6. Execution Time	The time taken to execute the code is calculated and printed. This can be useful for benchmarking and optimization purposes.	R: base R (Sys.time) Python: time (time)

Table 1

Decision Tree implementation

Step/Description	Libraries Used
1. Data Preprocessing: Load dataset and perform one-hot encoding on categorical variables	R: tidyverse (read_csv, as.factor), fastDummies (dummy_cols) Python: pandas (read_csv, get_dummies)
2. Dataset Split: Split the data into training and testing sets	R: base R (set.seed, sample) Python: sklearn.model_selection (train_test_split)
3. Decision Tree Model: Fit the Decision Tree model to the training data	R: rpart Python: sklearn.tree (DecisionTreeRegressor)
4. Prediction and Evaluation: Make predictions on the test set and calculate evaluation metrics	R: base R (predict), Metrics (rmse, mape) Python: sklearn.metrics (mean_squared_error), numpy (sqrt, mean, abs)
5. Visualize Results: Plot the tree, scatter plot of actual vs predicted values and residuals vs predicted values	R: rpart.plot, ggplot2 (ggplot, aes, geom_point, geom_abline, geom_hline, ggtitle, xlab, ylab) Python: sklearn.tree, matplotlib.pyplot (scatter, axhline, xlabel, ylabel, title, show)
6. Execution Time: Measure the execution time	R: base R (Sys.time) Python: time

Table 2

Additional Details:

Note: Steps 1, 2, 5, and 6 are similar to the Linear Regression implementation and utilize the same libraries. The following details focus on the unique aspects of the Decision Tree implementation.

Step 3 - Decision Tree Model Building:

- In R, the rpart library is utilized to create a Decision Tree model. The rpart function is used, specifying the formula charges ~ . to indicate that charges is the target variable, and all other variables are predictors. The summary function is used to summarize the various statistics of the tree.

```
# Create decision tree model
model_tree <- rpart(charges ~ ., data = training, method = "anova")
summary(model_tree)
```

- In Python, the DecisionTreeRegressor class from the sklearn.tree module is used to create and fit a Decision Tree model to the training data. The model_tree object is initialized with the DecisionTreeRegressor class, and then the fit method is called, passing the training features (train_features) and target (train_target) as arguments.

```
# Perform decision tree regression
model_tree = DecisionTreeRegressor(random_state=123)
model_tree.fit(train_features, train_target)
print(model_tree.feature_importances_)
```

Step 5: Visualizing the results:

To visualize the tree the Rpart.plot library is used in R and sklearn.tree and matplotlib.pyplot are used to create the tree in python.

Random Forest implementation

Step/Description	**Libraries Used**
1. Data Preprocessing: Load dataset and perform one-hot encoding on categorical variables	R: tidyverse (read.csv, as.factor), fastDummies (dummy_cols) Python: pandas (read_csv, get_dummies)
2. Dataset Split: Split the data into training and testing sets	R: base R (set.seed, sample) Python: sklearn.model_selection (train_test_split)
3. Random Forest Model: Fit the model to the training data	R: randomForest (randomForest) Python: sklearn.ensemble (RandomForestRegressor)
4. Prediction and Evaluation: Make predictions on the test set and calculate evaluation metrics	R: base R (predict), Metrics (rmse, mape) Python: sklearn.metrics (mean_squared_error)
5. Visualize Results: Plot scatter plot of actual vs predicted values and residuals vs predicted values	R: tidyverse (read.csv, as.factor), fastDummies (dummy_cols) Python: pandas (read_csv, get_dummies)
6. Execution Time: Measure the execution time	R: base R (set.seed, sample) Python: sklearn.model_selection (train_test_split)

Table 3

Additional Details:

The following details focus on the unique aspects of the Random Forest implementation.

Step 3 - Random Forest Model Building:

- In R, the randomForest library is utilized to create a Random Forest model. The randomForest function is used, specifying the formula - charges ~ . to indicate that charges is the target variable, and all other variables are predictors. The ntree parameter is set to 500, indicating the number of trees in the random forest. The summary function is then used to provide a summary of the model.

```
# Create random forest model
model_rf <- randomForest(charges ~ ., data = training, ntree = 500)
print(summary(model_rf))
```

- In Python, the RandomForestRegressor class from the sklearn.ensemble module is used to create and fit a Random Forest model to the training data. The model_rf object is initialized with the RandomForestRegressor class, specifying the number of estimators (trees) as 500 using the n_estimators parameter. The random_state parameter is set to 123 for reproducibility. Finally, the feature_importances_ attribute is used to print the importance scores of each predictor.

```
# Create random forest model
model_rf = RandomForestRegressor(n_estimators=500, random_state=123)
model_rf.fit(train_features, train_target)
print(model_rf.feature_importances_)
```

Neural Network Implementation

Step/Description	Libraries Used
1. Data Preprocessing: Load dataset and perform one-hot encoding on categorical variables	R: tidyverse, fastDummies, scales Python: pandas, scikit-learn
2. Dataset Split: Split the data into training and testing sets	R: caTools Python: scikit-learn
3. Neural Network Model: Fit the model to the training data	R: nnet Python: TensorFlow, Keras
4. Prediction and Evaluation: Make predictions on the test set and calculate evaluation metrics	R: base R Python: TensorFlow, numpy
5. Visualize Results: Plot scatter plot of actual vs predicted values and residuals vs predicted values	R: ggplot2 Python: matplotlib.pyplot
6. Execution Time: Measure the execution time	R: base R Python: time

Table 4

Additional Details: The following details focus on the unique aspects of the Neural Network implementation.

Step 3 - Neural Network Model Building:

- In R, the nnet function from the nnet library is used to create a Neural Network model.

  The formula charges ~ . specifies that charges is the target variable, and . indicates that all other variables are predictors.

  The size argument is set to 16, indicating that the neural network should have 16 nodes in the hidden layer and maxit refers to the number of iterations around the training data which is 1000 in this instance.

  After training the model, the summary function is used to provide a summary of the model, including information about the architecture, weights, and performance metrics.

```
# Create neural network model
  model_nn <- nnet(charges ~ ., data = training, size = 16, maxit=1000)
# Print model summary
  summary(model_nn)
```

- The Neural Network model in Python utilizes the TensorFlow and Keras library.

  The model has a total of four layers with 64, 32 and16 neurons in the hidden layers and 1 in the output layer and uses the relu activation function.

  The model is compiled using the Adam optimizer which is popular with regression.

  To train the model, the fit method is called with the training features and target. The number of epochs is 1000 which is the amount of times the model iterates over the training data, allowing it to learn from the patterns.

  After training, the model summary is printed, providing information about the model's architecture, including the number of parameters and output shapes of each layer.

```
#Define the neural network architecture

model= tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation="relu",
input_shape=(train_features.shape[1],)),

    tf.keras.layers.Dense(32, activation="relu"),
    tf.keras.layers.Dense(16, activation="relu"),
    tf.keras.layers.Dense(1)])

# Compile the model
```

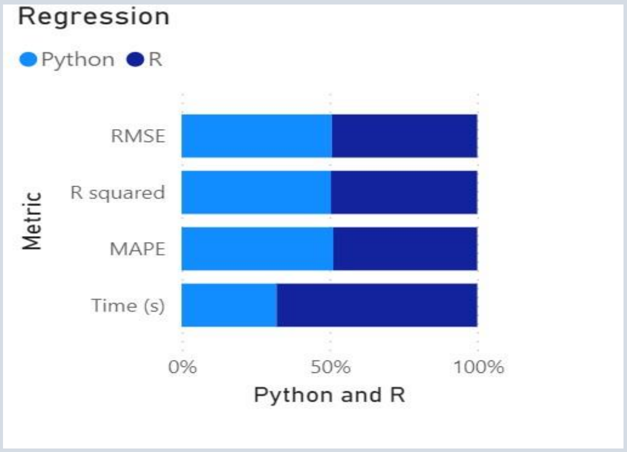
```
model.compile(optimizer="adam", loss="mean_squared_error")
# Print model summary
model.summary()

# Train the model
model.fit(train_features, train_target, epochs=1000, verbose=0)
```

IV. Results and Comparison:

Regression			Decision Tree			Random forest			Neural Network		
Metric	**R**	**Python**	**Metric**	**R**	**Python**	**Metric**	**R**	**Python**	**Metric**	**R**	**Python**
MAPE	0.4455	0.4695	MAPE	0.5219	0.47	MAPE	0.4451	0.374	MAPE	0.4455	0.3
RMSE	5726	5926	RMSE	4713	6978	RMSE	4422	4726	RMSE	5726	4638
Time (s)	0.95	0.45	Time(s)	0.55	1	Time (s)	1.1	1.2	Time(s)	0.2	50
R squared	0.7427	0.7581									

Table 5

Regression:



The regression models implemented in Python by sklearn and in R by its base had similar performance, with R showing marginally better results, yet Python was quicker. Both languages were comparable in terms of code length, ease of use, complexity, and visualization capabilities. However, they differed in identifying significant features: R highlighted 'smoker', 'BMI', 'age' and children as statistically significant while Python identified 'smoker', 'region', and 'children' as most important features. Overall, Python and R provide robust options for regression modeling, and the choice often comes down to preference.

R diagrams

Python diagrams

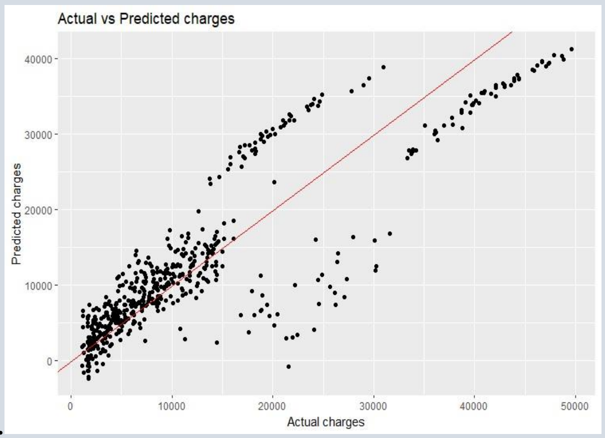


Fig. 1a

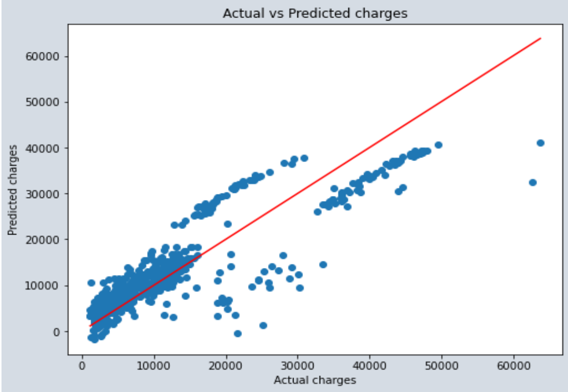


Fig. 1c

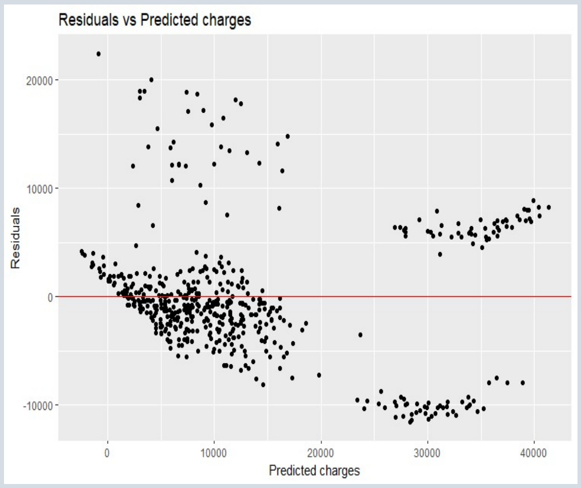


Fig. 1b

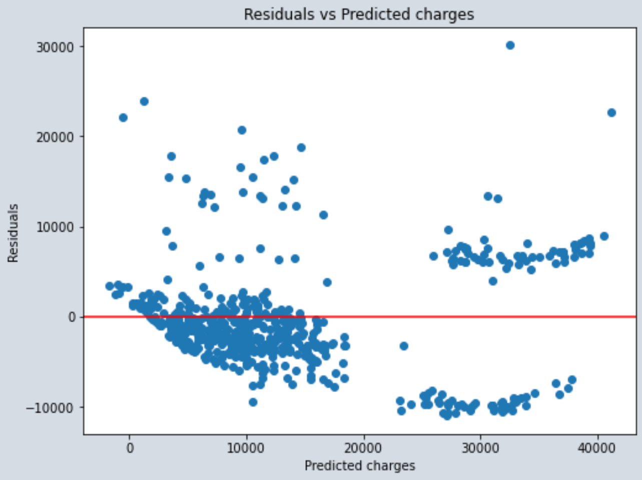


Fig. 1d

Decision Tree:

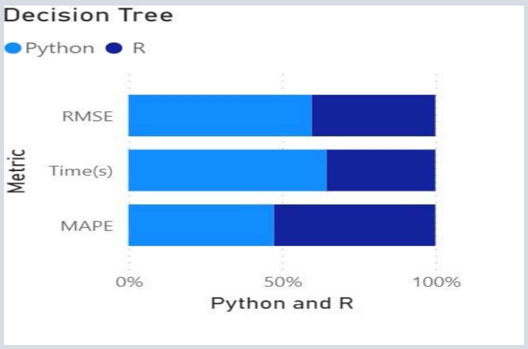


Fig 2.

Python and R exhibited unique strengths in their decision tree models. Python's model displayed superior relative accuracy, reflected by a lower Mean Absolute Percentage Error (MAPE). Conversely, R's model was better at handling larger errors, demonstrated by a lower Root Mean Square Error (RMSE). Additionally, R's model was computationally more efficient, executing faster than Python's. This difference can be significant when dealing with larger datasets or when frequent model runs are necessary.

Regarding interpretability, Python's decision tree was larger, making it slightly more complex to understand. In contrast, R's model, which focused on what it considered the most important features 'Smokers', 'BMI', and 'Age', was simpler and more interpretable.

Despite these differences, both models identified the same key features. Overall R slightly outperformed Python.

Fig. 3a Decision Tree- R

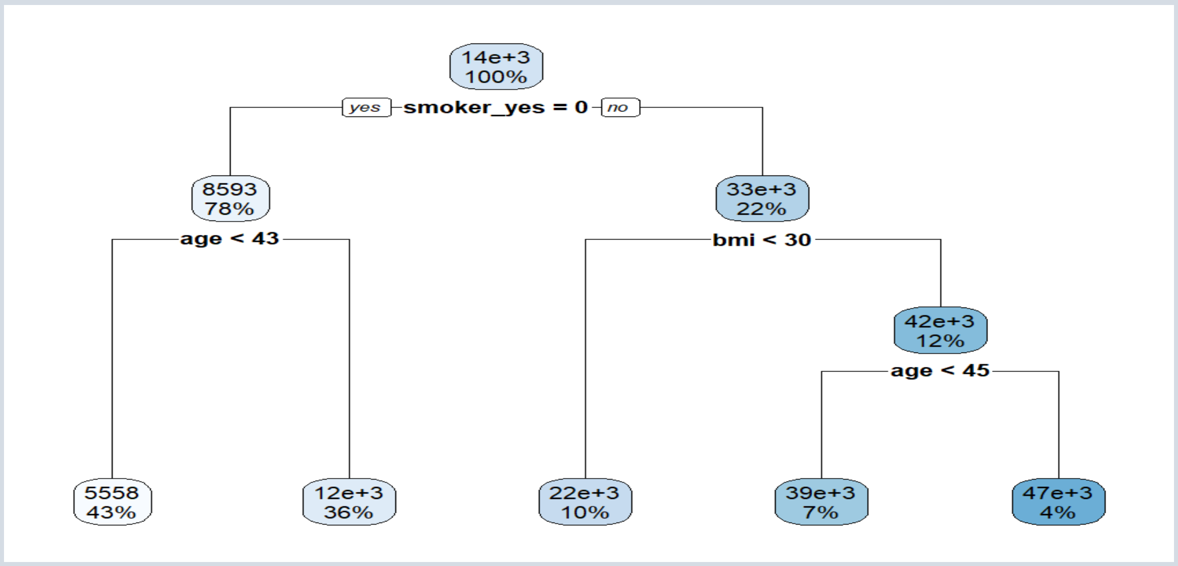


Fig 3b Decision tree -Python

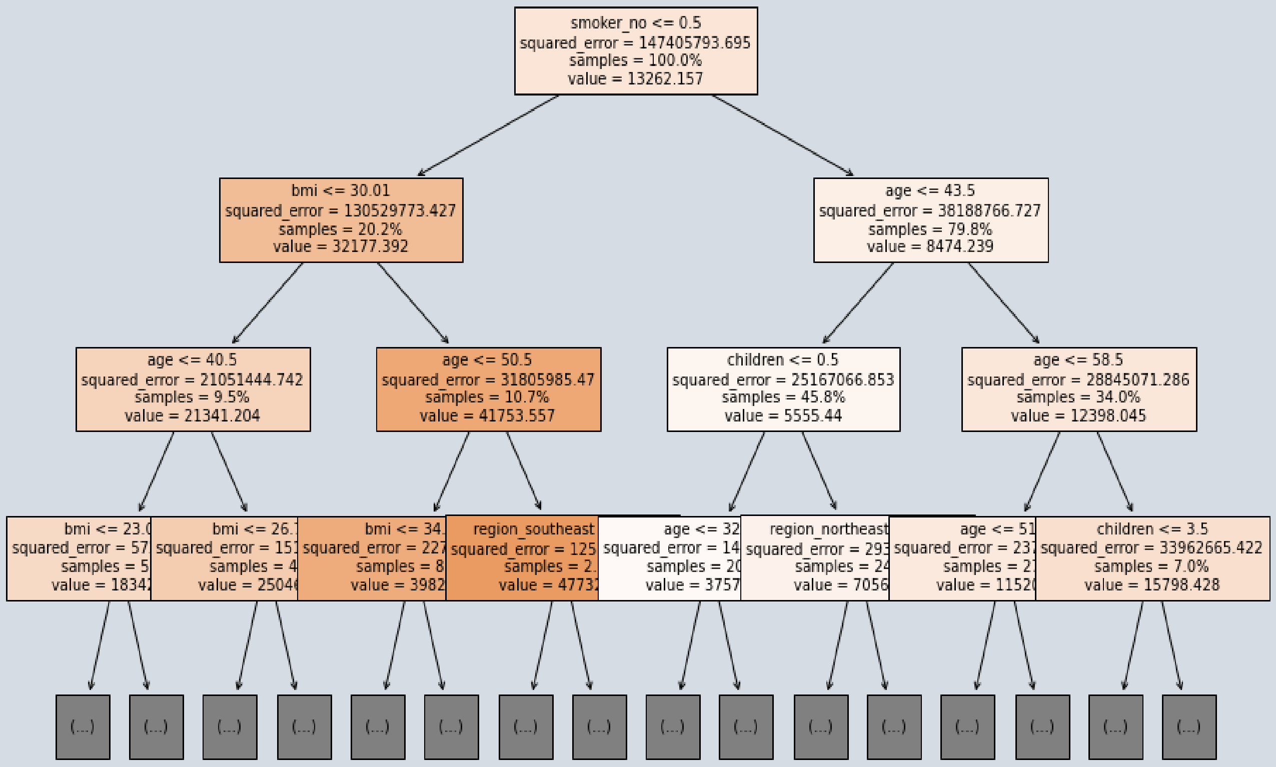
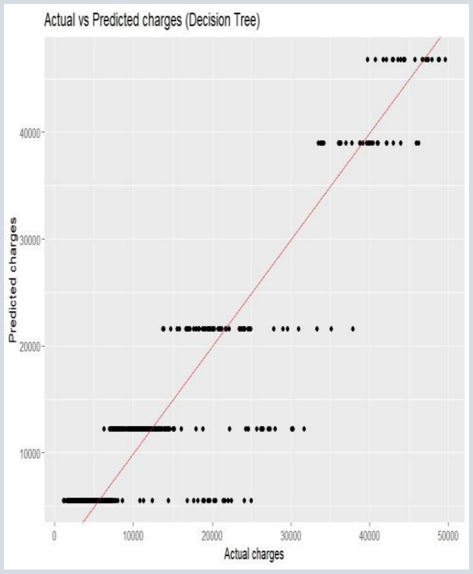
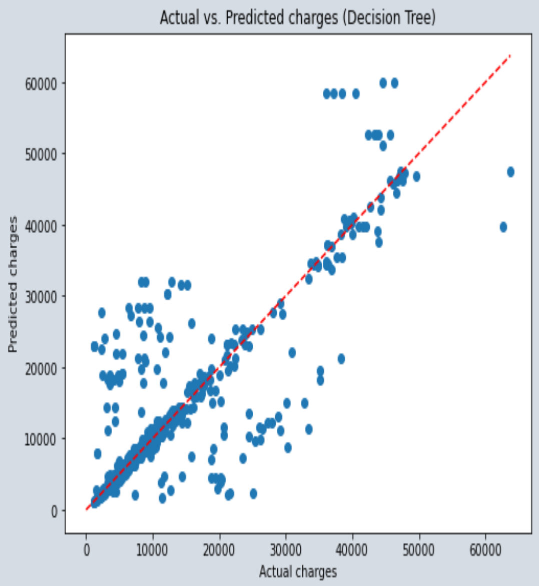


Fig 3b



R

Fig 3c



Python

In the "Actual vs Predicted" plot for a decision tree model, you will see a series of horizontal lines or steps. Each horizontal line represents a different leaf node in the decision tree. The predicted value for a given data point will be the average target value of the leaf node to which the data point belongs. We see this in the R visualization however the plot in Python appears like a regular scatter plot.

Random forest

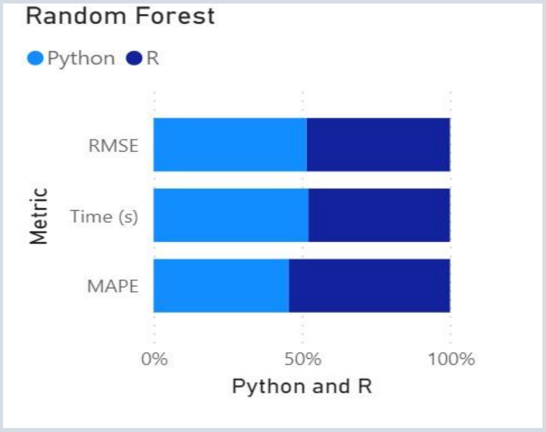
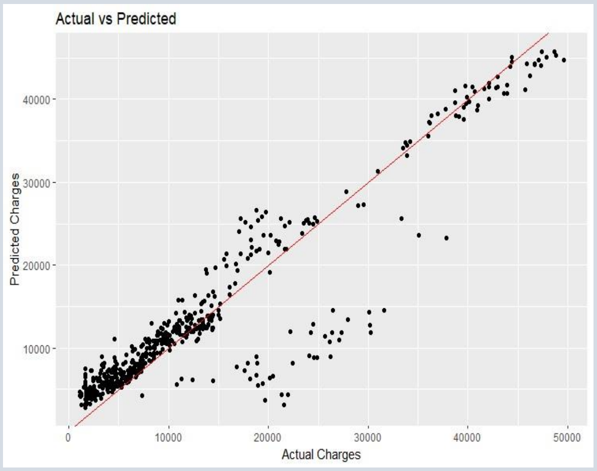


Fig 4

The performance of Random Forest models in both Python and R was quite similar as seen in Figure 4. Python's model had a slightly lower Mean Absolute Percentage Error (MAPE), indicating better relative accuracy, while R's model showed a lower Root Mean Square Error (RMSE), suggesting better handling of larger errors. Execution time was similar, with Python marginally slower. Both produced similar visualizations and identified similar key features.



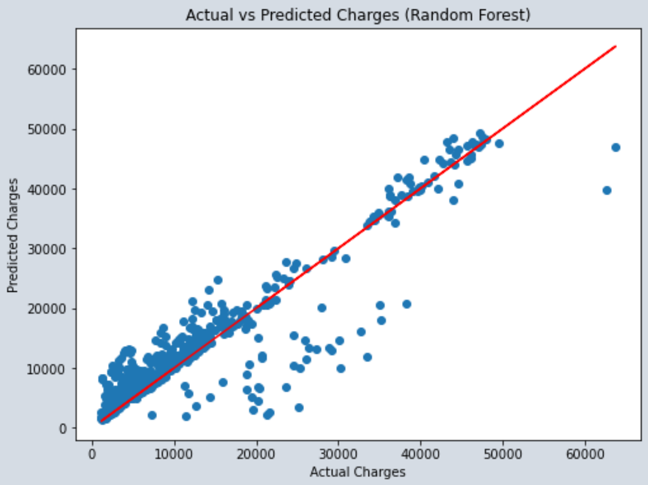


Fig.5a. R

Fig.5b. Python

Neural Network

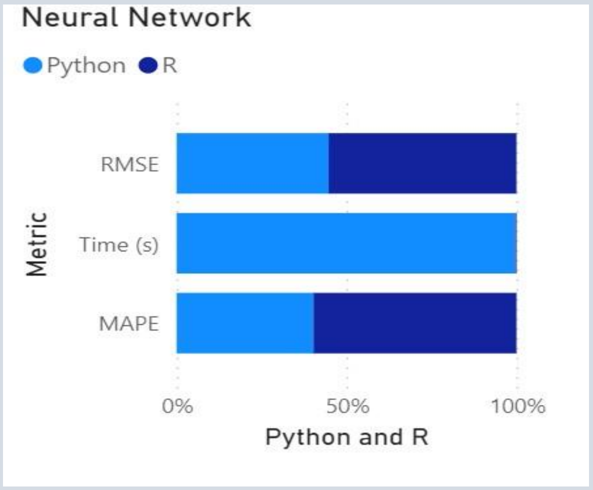


Fig. 6

In my experience, the TensorFlow library within Python offered more configuration options for neural network modeling, enabling the construction of complex models with multiple hidden layers and the use of advanced optimization algorithms like Adam. However, this came with significantly increased computational demand and processing time.

In contrast, R's **'nnet'** package offered a simpler and quicker implementation, but with limited flexibility, as it supports only one hidden layer. I also tried to use the '**neuralnet'** R packages which could offer more complexity, but it proved challenging due to initial errors and unending processing times. This could also be due to my limited expertise in its implementation.

Overall, Python provided more advanced neural network configurations at the cost of longer processing times and at such gave better results (as seen in table 5), while R offered ease of use but with restrictions on model complexity.

Other Considerations in comparing R and Python:

1. Code Length: In this research, the average length of code written for similar tasks is slightly longer for Python than R. This difference in code length is possibly due to Python's emphasis on code readability and its syntax, which can result in more explicit code compared to R's concise nature.

2. Support and Libraries: Python outperforms R in terms of the activity and support of their respective communities for machine learning, due to a wide array of libraries, packages, and online resources. Python's community engagement, as evidenced by the number of active forums and resource contributions, significantly surpasses R's.

3. Bridges Between Languages: In terms of interoperability between the two languages, R's 'reticulate' package stands out. It facilitates a more seamless bridge to Python, superior to Python's bridging solutions.

4. Learning Resources: Both R and Python are backed by extensive learning resources, including tutorials, documentation, books, and online courses. However, Python, with its wider general usage, offers a slightly more diverse range of learning materials.

5. Integration with Other Tools: R and Python both provide easy integration with common tools and platforms used in data analysis and machine learning workflows, including databases, visualization tools, cloud services, and big data frameworks.

6. Development Speed and Ease: Python might have a slight edge over R in terms of development speed, owing to its simpler syntax and a greater choice of IDEs. Nonetheless, R still maintains a competitive position with its intuitive code readability and availability of templates.

7. Visualization Capabilities: Both R and Python excel in data visualization, offering a robust array of libraries to generate high-quality graphics with customization options.

8. Industry Adoption: Python holds a slightly larger share in industry adoption due to its wide-ranging applications. However, R continues to be a preferred choice in research and academic settings.

9. Statistical Capabilities: In the field of statistics, R is often preferred due to its rich library of statistical packages and its inherent design as a statistical language, making it a powerful tool for complex statistical analyses.

10. Scalability and Performance: Python typically has the upper hand for large-scale data processing and complex machine learning models. This is attributed to its efficient memory usage and seamless integration with distributed computing frameworks.

11. Platform Compatibility: Both R and Python display strong compatibility with different operating systems, ensuring easy deployment, portability, and harmony with cloud platforms or containerization technologies.

The comparison between R and Python goes beyond quantitative performance metrics. Practical aspects, usability, and industry relevance equally matter in choosing the appropriate tool for machine learning tasks. Each language has its own strengths and trade-offs, often contingent upon the specific needs and context of a project.

V. **Limitations:**

1. Dataset Type and Size: The study was performed with a numerical target and mixed predictors, not extending to other types of data. Furthermore, the size of the dataset used was not very large, limiting our ability to make inferences about the performance of R and Python with big data.

2. Expertise Limitations: The study was conducted considering my current expertise in using complex machine learning algorithms. A more experienced data scientist might achieve different results or use other advanced techniques that were not considered in this comparison.

3. Preprocessing and Data Cleanliness: The dataset used was relatively clean, requiring minimal preprocessing. In real-world scenarios, datasets often require substantial cleaning and preprocessing, which can significantly impact the efficiency and effectiveness of the analysis.

4. Generalizability: The findings may be specific to the datasets, algorithms, and versions of R and Python used in this study. They may not be representative of all scenarios and applications of machine learning.

5. Implementation Bias: The results may have been influenced by the researcher's familiarity with R and Python. The level of expertise with each language could impact the quality of implementation and interpretation of results.

6. Performance Variations: The performance of ML algorithms can be affected by hardware configurations, optimization techniques, and library versions. Differences in system settings may introduce variability in the results.

7. Algorithm Selection: This study focused on a specific set of ML algorithms, not encompassing the entire range of algorithms available in R and Python. This selection may not fully represent the strengths and weaknesses of each language.

8. Versioning and Updates: The findings may not reflect the advancements and improvements made in subsequent versions of R and Python, as these languages are constantly evolving.

9. Data-specific Considerations: The results may vary depending on the characteristics of the datasets. Different datasets may require tailored approaches for optimal performance, which may not be fully captured in the comparison.

10. Sample Size: A moderate number of observations were used, which could affect the statistical power and reliability of the results.

11. Data Preprocessing and Feature Engineering: The impact of different data preprocessing techniques or feature engineering methods was not extensively explored in this study.

12. Human Factors: The role of human factors, such as user experience, cognitive load, and ease of use, were not explicitly considered in this study.

13. Bias in Metrics: Depending solely on a few metrics may not provide a comprehensive assessment of algorithm performance.

14. Time and Resource Constraints: The study may have overlooked or not explored certain aspects in depth due to time and resource constraints.

Recognizing the limitations of this study is crucial, not only for transparency, but also for guiding future research. Addressing these constraints, such as leveraging larger datasets, exploring diverse machine learning algorithms, and refining the performance metrics, could significantly enhance the robustness and relevance of the findings. Thus, the limitations present a pathway for furthering the understanding and comparison of machine learning capabilities in R and Python.

VI. Conclusion:

Based on the analysis of quantitative metrics and other non-quantitative considerations, Python emerges as a slightly more advantageous choice for implementing machine learning algorithms. While the observed differences in performance metrics, such as MAPE and RMSE, between Python and R were relatively small, Python's overall superiority across various aspects makes it a preferred option for many practical scenarios.

One significant factor contributing to Python's favorable position is its extensive library ecosystem and the robust support from its active community. Python offers a diverse range of machine learning libraries and frameworks, including TensorFlow and scikit-learn, which provide advanced functionalities and incorporate impressive algorithms. This wealth of resources empowers data scientists and developers to effortlessly implement complex machine learning models and leverage advanced techniques.

Furthermore, Python's wider industry adoption and prevalence across diverse domains enhance its compatibility with existing workflows and tools. The seamless integration of Python with other technologies, such as databases, visualization tools, and cloud services, facilitates streamlined data processing and analysis.

Additionally, Python's code simplicity and readability contribute to its popularity. The language's intuitive syntax and expressive nature enable data scientists and developers to write and maintain machine learning code with ease. Furthermore, the extensive learning resources further augments Python's usability and expedites the development process.

While R excels in statistical capabilities and remains prominent in research and academic circles, Python's versatility and broader industry support confer additional advantages. Python's flexibility in handling large-scale datasets, scalability, and compatibility with distributed computing frameworks make it particularly suitable for projects involving big data and intricate machine learning models.

It is however vital to emphasize that the choice between Python and R should not rely solely on performance metrics but necessitates a holistic evaluation of project requirements, available resources, team expertise, and overall objectives. The decision should align with the specific needs of the project, considering factors such as code efficiency, ease of use, visualization capabilities, community support, and future scalability.

References:

Ushey K, Allaire J, Tang Y (2023). *reticulate: Interface to 'Python'*. https://rstudio.github.io/reticulate/, https://github.com/rstudio/reticulate.

Bryer, J., & Speerschneider, K. (2016). R Programming for Data Science.

Buitinck, L. et. Al (2013). API design for machine learning software: experiences from the scikit-learn project. In ECML PKDD Workshop: Languages for Data Mining and Machine Learning

Chang, W., Cheng, J., Allaire, J., Xie, Y., & McPherson, J. (2021). shiny: Web Application Framework for R. R package version 1.6.0.

Chollet, F., & others. (2015). Keras.

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), 90-95.

Kuhn, M. Caret package. Journal of Statistical Software

McKinney, W. (2010). Data Structures for Statistical Computing in Python. Proceedings of the 9th Python in Science Conference.

Millman, K. J., & Aivazis, M. (2011). Python for scientists and engineers. Computing in Science & Engineering.

Paszke, A., Chanan et Al (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library.

Pedregosa et Al (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research.

Probst, P., Boulesteix, A. L., & Bischl, B. (2019). Tunability: Importance of hyperparameters of machine learning algorithms. Journal of Machine Learning Research

Python Software Foundation. (2021). The Python Community. Python.org.

R Core Team. (2020). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria.

TensorFlow. (2015). Large-scale machine learning on heterogeneous distributed systems.

Virtanen, P. et Al. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods

Walt, S. V. D., Colbert, S. C., & Varoquaux, G. (2011). The NumPy Array: A Structure for Efficient Numerical Computation. Computing in Science & Engineering

Waskom, M et Al. (2020). mwaskom/seaborn: v0.12.2 (December 2022).

Wickham, H. (2016). ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York.

Wickham, H., & Grolemund, G. (2017). R for Data Science: Import, Tidy, Transform, Visualize, and Model Data. O'Reilly Media, Inc.

Chat GPT

Appendix

R files: https://github.com/RoliAr/IR-files/blob/main/Indep%20Research%20insurance.R

Python files:https://github.com/RoliAr/IRfiles/blob/main/Ind%20Research%20insurance.ipynb

Regression Codes

R

```
#Import Libraries
library(tidyverse)
library(caret)
library(Metrics)
library(ggplot2)

# Start the timer
start_time <- Sys.time()

# Read data
data                <-                  read.csv
("https://raw.githubusercontent.com/RoliAr/IR-
files/main/insurance.csv")
# Convert categorical variables to factor
data$sex <- as.factor(data$sex)
data$smoker <- as.factor(data$smoker)
data$region <- as.factor(data$region)

# One-hot encoding
data     <-     fastDummies::dummy_cols(data,
select_columns = c("sex", "smoker", "region"))

N <- nrow(data)
set.seed(123)

# Set training size
training_size <- round(N * 0.6)

# Set training rows
training_cases <- sample(N, training_size)

# Make training df
training <- data[training_cases,]

# Make test df = not in training cases
test <- data[-training_cases,]

model <- lm(charges ~ ., data = training)
summary(model)

# Predict
predictions <- predict(model, test)

# Evaluate
# Find true values
observations <- test$charges

# Compute the errors
errors <- observations - predictions

# Compute KPIS = rmse = root mean squared error
rmse <- sqrt(mean(errors^2))

# Mean absolute % error
mape <- mean(abs(errors / observations))

# Create residuals
```

Python

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error,
 r2_score
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import OneHotEncoder
import time

# Start the timer
start_time = time.time()

# Read the data
#data = #pd.read_csv("C:/Users/ROEYE/Downloads/insurance.csv")
data = pd.read_csv("https://raw.githubusercontent.com/RoliAr/
IR-files/main/insurance.csv")

# One-hot encoding
data = pd.get_dummies(data, columns=["sex", "smoker", "region"])

# Split the data into training and testing sets
train, test = train_test_split(data, test_size=0.4, random_state=123)

# Separate the features from the target
train_features = train.drop(columns="charges")
train_target = train["charges"]

test_features = test.drop(columns="charges")
test_target = test["charges"]
##################################Regression##############################
# Perform linear regression
model = LinearRegression()
model.fit(train_features, train_target)

# Model summary
print(f"R-squared: {r2_score(test_target, model.predict(test_features))}")

# Get the feature names from train_features after one-hot encoding
feature_names = train_features.columns

# Print feature importances
print("Feature Importances:")
print(feature_importance_df)

# Predict the charges on the test data and evaluate the model
predictions = model.predict(test_features)

# Compute errors
errors = test_target - predictions

# Compute RMSE and MAPE
rmse = np.sqrt(mean_squared_error(test_target, predictions))
mape = np.mean(np.abs(errors/test_target))

# Print results
```

```r
residuals <- test$charges - predictions

# Create data frame for plotting residuals
residuals_data   <-   data.frame(Predicted   =
predictions, Residuals = residuals)

# Plot Actual vs Predicted charges
ggplot(residuals_data, aes(x = observations, y
= predictions)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, color
= "red") +
  ggtitle("Actual vs Predicted charges") +
  xlab("Actual charges") +
  ylab("Predicted charges")

# Plot residuals vs predicted values
ggplot(residuals_data, aes(x = Predicted, y =
Residuals)) +
  geom_point() +
  geom_hline(yintercept = 0, color = "red") +
  ggtitle("Residuals vs Predicted charges") +
  xlab("Predicted charges") +
  ylab("Residuals")

# End the timer
end_time <- Sys.time()

# Print the time taken
print(end_time - start_time)
```

```python
print(f"RMSE: {rmse}, MAPE: {mape}")

# Scatter plot of actual vs predicted values
plt.figure(figsize=(8, 6))
plt.scatter(test_target, predictions)
plt.xlabel('Actual charges')
plt.ylabel('Predicted charges')
plt.title('Actual vs Predicted charges')

# Plotting the line for perfect prediction
plt.plot([min(test_target), max(test_target)], [min(test_target),


max(test_target)], color='red')
plt.show()

# Residuals vs Predicted values
residuals = test_target - predictions

plt.figure(figsize=(8, 6))
plt.scatter(predictions, residuals)
plt.xlabel('Predicted charges')
plt.ylabel('Residuals')
plt.title('Residuals vs Predicted charges')
plt.axhline(0, color='red') # adding a horizontal line at y=0 for
reference
plt.show()

# Print the time taken
end_time = time.time()
print("Time taken: ", end_time - start_time, "seconds")
```

Decision Tree Codes

R

```r
#install.packages("rpart")
library(rpart)
library(rpart.plot)

# Start the timer
start_time_tree <- Sys.time()


# Create decision tree model
model_tree <- rpart(charges ~ ., data = training, method =
"anova")
printcp(model_tree)
summary(model_tree)

# Predict
predictions_tree <- predict(model_tree, test)

# Compute the errors for decision tree model
errors_tree <- test$charges - predictions_tree

# Compute RMSE and MAPE for decision tree model
rmse_tree <- sqrt(mean(errors_tree^2))
mape_tree <- mean(abs(errors_tree / test$charges))

# Print metrics for decision tree model
print(paste("Decision Tree - RMSE: ", rmse_tree))
print(paste("Decision Tree - MAPE: ", mape_tree))

# Plot the tree
rpart.plot(model_tree)

# Plot actual vs predicted values
```

Python

```python
# Import necessary libraries- Decision Tree
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import numpy as np
import time

# Start the timer
start_time = time.time()

# Read the data
data                                                     =
pd.read_csv("https://raw.githubusercontent.com/RoliAr/IR-
files/main/insurance.csv")

# One-hot encoding
data   =   pd.get_dummies(data,   columns=["sex",   "smoker",
"region"])

# Split the data into training and testing sets
train,   test   =   train_test_split(data,   test_size=0.4,
random_state=123)

# Separate the features from the target
train_features = train.drop(columns="charges")
train_target = train["charges"]

test_features = test.drop(columns="charges")
test_target = test["charges"]
```

```r
ggplot(data.frame(Actual = test$charges, Predicted =
predictions_tree), aes(x = Actual, y = Predicted)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, color = "red") +
  ggtitle("Actual vs Predicted charges (Decision Tree)") +
  xlab("Actual charges") +
  ylab("Predicted charges")

# Calculate residuals
residuals_tree <- test$charges - predictions_tree

# Create data frame for plotting
residuals_data_tree <- data.frame(Predicted =
predictions_tree, Residuals = residuals_tree)

# Plot residuals vs predicted values
ggplot(residuals_data_tree, aes(x = Predicted, y =
Residuals)) +
  geom_point() +
  geom_hline(yintercept = 0, color = "red") +
  ggtitle("Residuals vs Predicted charges (Decision
Tree)") +
  xlab("Predicted charges") +
  ylab("Residuals")

# End the timer
end_time_tree <- Sys.time()

# Print the time taken
print(end_time_tree - start_time_tree)
```

```python
# Perform decision tree regression
model_tree = DecisionTreeRegressor(random_state=123)
model_tree.fit(train_features, train_target)

# Predict the charges on the test data and evaluate the
model
predictions_tree = model_tree.predict(test_features)

# Compute errors
errors_tree = test_target - predictions_tree

# Compute RMSE and MAPE for decision tree
rmse_tree = np.sqrt(mean_squared_error(test_target,
predictions_tree))
mape_tree = np.mean(np.abs(errors_tree / test_target))

# Print results
print(f"Decision Tree - RMSE: {rmse_tree}, MAPE:
{mape_tree}")

# Plot the tree with percentage samples
plt.figure(figsize=(15, 10))
plot_tree(model_tree, max_depth=3, filled=True,
feature_names=train_features.columns, fontsize=10,
proportion=True)
plt.show()

# Plot actual vs. predicted charges
plt.figure(figsize=(8, 6))
plt.scatter(test_target, predictions_tree)
plt.xlabel('Actual charges')
plt.ylabel('Predicted charges')
plt.title('Actual vs. Predicted charges (Decision Tree)')
plt.plot([0, np.max(test_target)], [0,
np.max(test_target)], color='red', linestyle='--') #
Adding diagonal reference line
plt.show()

# Plot residuals vs. predicted charges
plt.figure(figsize=(8, 6))
plt.scatter(predictions_tree, errors_tree)
plt.xlabel('Predicted charges')
plt.ylabel('Residuals')
plt.title('Residuals vs. Predicted charges (Decision
Tree)')
plt.axhline(0, color='red') # Adding a horizontal line at
y=0 for reference
plt.show()

# Calculate elapsed time
end_time = time.time()
elapsed_time = end_time - start_time
print(f"Elapsed time: {elapsed_time} seconds")
```

Random Forest Codes

R

```r
# Load the package
library(randomForest)

# Start the timer
start_time_rf <- Sys.time()

# Create random forest model
model_rf <- randomForest(charges ~ ., data = training,
ntree=500)
```

Python

```python
# Random Forest- Import necessary libraries
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import numpy as np
import matplotlib.pyplot as plt

# Start the timer
start_time = time.time()

# Perform random forest regression
```

```r
# Print model summary
print(summary(model_rf))

# Predict
predictions_rf = predict(model_rf, test)

# Compute the errors for random forest model
errors_rf = test$charges - predictions_rf

# Compute RMSE and MAPE for random forest model
rmse_rf = sqrt(mean(errors_rf^2))
mape_rf = mean(abs(errors_rf/test$charges))

# Print metrics for random forest model
print(paste("Random Forest - RMSE: ", rmse_rf))
print(paste("Random Forest - MAPE: ", mape_rf))
# Create a data frame to hold the actual and predicted values
df <- data.frame(Actual = test$charges, Predicted =
predictions_rf)

# Plot actual vs predicted values
library(ggplot2)
ggplot(df, aes(x = Actual, y = Predicted)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, color = "red") +
  ggtitle("Actual vs Predicted") +
  xlab("Actual Charges") +
  ylab("Predicted Charges")

# Calculate residuals
residuals_rf <- test$charges - predictions_rf

# Create data frame for plotting
residuals_data_rf <- data.frame(Predicted = predictions_rf,
Residuals = residuals_rf)

# Plot residuals vs predicted values
ggplot(residuals_data_rf, aes(x = Predicted, y = Residuals)) +
  geom_point() +
  geom_hline(yintercept = 0, color = "red") +
  ggtitle("Residuals vs Predicted charges (Random Forest)") +
  xlab("Predicted charges") +
  ylab("Residuals")

# End the timer
end_time_rf <- Sys.time()

# Print the time taken
print(end_time_rf - start_time_rf)
```

```python
model_rf = RandomForestRegressor(n_estimators=500,
random_state=123)
model_rf.fit(train_features, train_target)

# print summary of model
print(model_rf.feature_importances_)

# Predict the charges on the test data and evaluate
the model
predictions_rf = model_rf.predict(test_features)

# Compute errors
errors_rf = test_target - predictions_rf

# Compute RMSE and MAPE for random forest
rmse_rf = np.sqrt(mean_squared_error(test_target,
predictions_rf))
mape_rf = np.mean(np.abs(errors_rf/test_target))

# Print results
print(f"Random Forest - RMSE: {rmse_rf}, MAPE:
{mape_rf}")

# Calculate residuals
residuals_rf = test_target - predictions_rf


# Create a scatter plot of actual vs predicted values
plt.figure(figsize=(8, 6))
plt.scatter(test_target, predictions_rf)
plt.xlabel('Actual Charges')
plt.ylabel('Predicted Charges')
plt.title('Actual vs Predicted Charges (Random
Forest)')

# Adding a diagonal line (y=x) for reference
plt.plot(test_target, test_target, color='red')

plt.show()
# Residuals vs Predicted values
plt.figure(figsize=(8, 6))
plt.scatter(predictions_rf, residuals_rf)
plt.xlabel('Predicted charges')
plt.ylabel('Residuals')
plt.title('Residuals vs Predicted charges (Random
Forest)')
plt.axhline(0, color='red') # adding a horizontal line
at y=0 for reference
plt.show()

# Calculate elapsed time
end_time = time.time()
elapsed_time = end_time - start_time
print(f"Elapsed time: {elapsed_time} seconds")
```

Neural Network Codes

R

```r
library(nnet)
# Start the t
imer
start_time_nn <- Sys.time()

model_nn <- nnet(charges ~ ., data = training, size = 16,
maxit = 1000)
summary(model_nn)
```

Python

```python
#Neural Network
import tensorflow as tf
# Get the feature names
feature_names = train_features.columns

# Convert features and target to arrays
train_features_array = train_features.values
train_target_array = train_target.values
test_features_array = test_features.values
```

```r
# Predict
predictions_nn <- predict(model, test, type = "response")

# Evaluate
# Find true car values
observations <- test$charges

# Compute the errors
errors_nn <- observations - predictions

# Compute KPIS
rmse_nn <- sqrt(mean(errors^2))
mape_nn <- mean(abs(errors / observations))

# Print evaluation metrics
print(paste("Neural Network Model - RMSE:", rmse_nn))
print(paste("Neural Network Model - MAPE:", mape_nn))

# Plot actual vs predicted
plot_data <- data.frame(Actual = observations, Predicted =
predictions_nn)
plot(plot_data$Actual, plot_data$Predicted, xlab = "Actual
charges", ylab = "Predicted charges",
     main = "Actual vs Predicted charges (Neural Network)",
col = "blue", pch = 16)
abline(a = 0, b = 1, col = "red")  # Adding a diagonal
reference line

# Plot residuals vs predicted
residuals_nn <- errors_nn
plot(predictions_nn, residuals_nn, xlab = "Predicted
charges", ylab = "Residuals",
     main = "Residuals vs Predicted charges (Neural
Network)", col = "blue", pch = 16)
abline(h = 0, col = "red")  # Adding a horizontal line at y
= 0

# End the timer
end_time_nn <- Sys.time()

# Print the time taken
print(end_time_nn - start_time_nn)
```

```python
# Start the timer
start_time = time.time()


# Define the neural network architecture
model_nn = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation="relu",
input_shape=(train_features_array.shape[1],),
name="input_layer"),
    tf.keras.layers.Dense(32, activation="relu",
name="hidden_layer1"),
    tf.keras.layers.Dense(16, activation="relu",
name="hidden_layer2"),
    tf.keras.layers.Dense(1, name="output_layer")
])

# Compile the model
model_nn.compile(optimizer="adam",
loss="mean_squared_error")

# Print model summary
#model_nn.summary()

# Train the model
model_nn.fit(train_features_array, train_target_array,
epochs=1000, verbose=0)

# Evaluate the model
predictions =
model_nn.predict(test_features_array).flatten()

# Compute the errors
errors = test_target - predictions

# Compute RMSE and MAPE
rmse = np.sqrt(mean_squared_error(test_target,
predictions))
mape = np.mean(np.abs(errors / test_target))

# Print evaluation metrics
print("Neural Network Model - RMSE:", rmse)
print("Neural Network Model - MAPE:", mape)

# Plot actual vs predicted
plt.figure(figsize=(8, 6))
plt.scatter(test_target, predictions)
plt.xlabel("Actual charges")
plt.ylabel("Predicted charges")
plt.title("Actual vs Predicted charges (Neural Network)")
plt.plot([0, np.max(test_target)], [0,
np.max(test_target)], color="red", linestyle="--")  #
Adding diagonal reference line
plt.show()

# Plot residuals vs predicted
plt.figure(figsize=(8, 6))
plt.scatter(predictions, errors)
plt.xlabel("Predicted charges")
plt.ylabel("Residuals")
plt.title("Residuals vs Predicted charges (Neural
Network)")
plt.axhline(0, color="red")  # Adding a horizontal line
at y=0 for reference
plt.show()

# Calculate elapsed time
end_time = time.time()
elapsed_time = end_time - start_time
print(f"Elapsed time: {elapsed_time} seconds")
```