MSc in Business Analytics

Module Name: Applied Statistics & Machine Learning

Module Code: B9BA102

GROUP PROJECT

Lecturer Name:	Assem Abdelhak
Group Members:	Muhammad Shaqif Syauqi Bin Mohd Syukri – 20014235 Vedant Tomer - 20015122 Shahzad Saeed - 10627692
Assignment Title	Statistical Regression Analysis on Insurance Charges
Issued Date:	20^{th} November 2023

Table of Contents

1.0 Data Preparation

In fact, a crucial step in every data analysis or machine learning project is data preparation. It involves several steps to ensure that data is relevant, clean, and transformed from raw form into a format that can be used for analysis. It is an essential step in a data analysis and machine learning pipeline. For

1.1 Importing Data

Figure 1.1 shows that, in this step first, we are using a powerful library like '**pandas**' from panda's import read_csv , Data Frame, Series, get dummies to read data in formats such as "CSV, Excel, HTML," etc., and storing the file in the environment being used, like Google Collab, Replit, etc. Without reading the data, analysing, or storing it becomes impossible to read the data for analysis. We use functions like **Data frame** and **Series** for read the csv file and handling data in tabular form. While for performing one-hot coding we use get dummies for categorical data. On the other hand, we use **Standard scaler** for feature standardizing by removing the mean and scaling to unit variance. For hyperparameter tuning by cross-validation we use the function **Grid search**. Importing "graph_objs" and ''figure_factory" from library **Plotly** is used to display interactive data that indicates a desire to create a variety of plots and interactive figures to visualize data and make model performance.

```
from pandas import  read_csv, DataFrame,Series,get_dummies
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model
#from sklearn import linear_model
from sklearn.linear_model import SGDRegressor
from plotly import graph_objs, figure_factory

# reading and metadata
insurance_df = read_csv("/content/insurance_dataset_updated.csv")
#print(data1.head(2))
print(insurance_df.shape)
print(insurance_df.info())
# # # # print(data1.describe())
```

Figure 1. 1: Importing Raw Dataset

```
Data columns (total 12 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   age                    2000 non-null   int64
 1   gender                 2000 non-null   object
 2   bmi                    2000 non-null   float64
 3   children               2000 non-null   int64
 4   smoker                 2000 non-null   object
 5   region                 2000 non-null   object
 6   medical_history        1504 non-null   object
 7   family_medical_history 1526 non-null   object
 8   exercise_frequency     2000 non-null   object
 9   occupation             2000 non-null   object
 10  coverage_level         2000 non-null   object
 11  charges                2000 non-null   float64
dtypes: float64(2), int64(2), object(8)
memory usage: 187.6+ KB
None
```

Figure 1. 2: Raw Data Info

Figure 1.2 illustrates that 'medical_history' and 'family_medical_history' contain some empty values in the dataset. To address this issue, we use the 'fillna' method to replace these empty values with 'No,' ensuring that there are no missing values in either the medical history or family medical history columns.

For the medical history column, the code used is:

```
insurance_df['medical_history'].fillna('No', inplace=True)
```

Figure 1. 3:fillna Function for 'medical_history' code snippet

Similarly, for the family medical history column, the code employed is:

```
insurance_df['family_medical_history'].fillna('No', inplace=True)
```

Figure 1. 4: fillna Function for 'family_medical_history' code snippet

These lines of code effectively replace any missing values with 'No' in the respective columns of the 'insurance_df' DataFrame.

In a nutshell, after executing these lines of code, any empty values in the 'family_medical_history' and 'medical_history' columns were replaced with the value 'No.' This ensures that no specific missing values are left in these columns within the dataset.

1.2 Encoding Data:

In this step, there are eight columns that contain categorical values that need to be encoded through performing a one-hot encoding. Among these columns, two can be encoded using map function, while 'get_dummies' can be used for the remaining columns.

```
# one hot encoding map
insurance_df['gender'] = insurance_df['gender'].map({'female': 1, 'male': 0})
insurance_df['smoker'] = insurance_df['smoker'].map({'yes': 1, 'no': 0})
insurance_df['medical_history'].fillna('No', inplace=True)
insurance_df['family_medical_history'].fillna('No', inplace=True)

#one hot encoding get dummies
insurance_final = get_dummies(insurance_df, columns =
 ['region',
  'exercise_frequency',
  'medical_history',
  'family_medical_history',
  'occupation',
  'coverage_level'],
drop_first=True) # encoding
```

Figure 1. 5: Encoding (Get_dummies & Map Fucntion)

Figure 1.5 demonstrates the process of converting categorical values in the 'gender' column. We utilize the code **insurance_df['gender'] = insurance_df['gender'].map({'female': 1, 'male': 0})** to convert string values like 'Male' and 'Female' into numerical representations; for instance, 'female' becomes 1 and 'male' becomes 0 using the **map()** function. Similarly, **insurance_df['smoker']= insurance_df['smoker'].map({'yes': 1, 'no': 0})** converts categorical values in the 'smoker' column to numerical values, labeling 'Yes' as 1 and 'No' as 0, indicating the smoker and non-smoker categories, respectively.

Additionally, **insurance_df['medical_history'].fillna('No', inplace=True)** is employed to handle missing values within the 'medical history' column, replacing them with 'No'. This line ensures that any missing value in the column is replaced with 'No'. Similarly, **insurance_df['family_medical_history'].fillna('No', inplace=True)** serves the same purpose by replacing missing values in the 'family medical history' column with 'No'.

The function **get_dummies** serve a different purpose; it encodes columns with numerous categorical values by creating new columns with dummy values for each categorical variable. In my data we use variables 'region', 'exercise_frequency', 'medical_history', 'family_medical_history', 'occupation', 'coverage_level'to make their dummies. This function expands the dataset with additional columns, representing categorical variables in a binary manner. To distinguish between the original dataset and the enhanced one with additional columns, a new variable name is assigned to these newly created columns and the previous columns.

Since algorithms typically work with numerical values, these operations were conducted to encode categorical data into numerical binary values, enabling the algorithm to process the information effectively. This type of preparation quite often used before every data to go for the analysis and modelling process.

```
Data columns (total 23 columns):
 #   Column                                  Non-Null Count  Dtype
---  ------                                  --------------  -----
 0   age                                     2000 non-null   int64
 1   gender                                  2000 non-null   int64
 2   bmi                                     2000 non-null   float64
 3   children                                2000 non-null   int64
 4   smoker                                  2000 non-null   int64
 5   charges                                 2000 non-null   float64
 6   region_northwest                        2000 non-null   uint8
 7   region_southeast                        2000 non-null   uint8
 8   region_southwest                        2000 non-null   uint8
 9   exercise_frequency_Never                2000 non-null   uint8
 10  exercise_frequency_Occasionally         2000 non-null   uint8
 11  exercise_frequency_Rarely               2000 non-null   uint8
 12  medical_history_Heart disease           2000 non-null   uint8
 13  medical_history_High blood pressure     2000 non-null   uint8
 14  medical_history_No                      2000 non-null   uint8
 15  family_medical_history_Heart disease    2000 non-null   uint8
 16  family_medical_history_High blood pressure  2000 non-null   uint8
 17  family_medical_history_No               2000 non-null   uint8
 18  occupation_Student                      2000 non-null   uint8
 19  occupation_Unemployed                   2000 non-null   uint8
 20  occupation_White collar                 2000 non-null   uint8
 21  coverage_level_Premium                  2000 non-null   uint8
 22  coverage_level_Standard                 2000 non-null   uint8
dtypes: float64(2), int64(4), uint8(17)
memory usage: 127.1 KB
```

Figure 1. 6: Prepared Data Info

After the code execution is completed, it is important to print the information of the new dataset in the terminal for verification purposes. As illustrated in figure 1.6, the new dataset comprises 22 columns, compared to the old dataset, which had only 11 columns. Moreover, there are no categorical values present in the new dataset.

1.3 Heatmap:

Heat map used as a method that helps to reduce the features or we can say the variables in the data set. To simplify the data, we aim to use the feature reduction while retaining the information important or by select the important feature. In our case, these features does not have high correlation, and the output values are not all near to **1** or **–1.** So, we are not dropping any extra features.

1.4 Scaling Data

Feature scaling is the next step after encoding. There are several ways to perform feature scaling, but we use, **'StandardScaler'**. The dataset needs to be split into two parts: features and labels. To achieve this, the drop function is used to eliminate specified columns. Features and labels are then assigned to 'X' and 'Y,' respectively, as shown in figure 1.7.

```
X = insurance_final.drop(['charges'], axis = 1) # Features
Y = insurance_final['charges'] # Labels
X.info()
print(Y.shape)
print(X.shape)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 22 columns):
 #   Column                                    Non-Null Count  Dtype
---  ------                                    --------------  -----
 0   age                                       2000 non-null   int64
 1   gender                                    2000 non-null   int64
 2   bmi                                       2000 non-null   float64
 3   children                                  2000 non-null   int64
 4   smoker                                    2000 non-null   int64
 5   region_northwest                          2000 non-null   uint8
 6   region_southeast                          2000 non-null   uint8
 7   region_southwest                          2000 non-null   uint8
 8   exercise_frequency_Never                  2000 non-null   uint8
 9   exercise_frequency_Occasionally           2000 non-null   uint8
 10  exercise_frequency_Rarely                 2000 non-null   uint8
 11  medical_history_Heart disease             2000 non-null   uint8
 12  medical_history_High blood pressure       2000 non-null   uint8
 13  medical_history_No                        2000 non-null   uint8
 14  family_medical_history_Heart disease      2000 non-null   uint8
 15  family_medical_history_High blood pressure 2000 non-null  uint8
 16  family_medical_history_No                 2000 non-null   uint8
 17  occupation_Student                        2000 non-null   uint8
 18  occupation_Unemployed                     2000 non-null   uint8
 19  occupation_White collar                   2000 non-null   uint8
 20  coverage_level_Premium                    2000 non-null   uint8
 21  coverage_level_Standard                   2000 non-null   uint8
dtypes: float64(1), int64(4), uint8(17)
memory usage: 111.5 KB
(2000,)
(2000, 22)
```

Figure 1. 7: Encoded Dataset Info

We use this code to be associated with a machine learning or data analysis task related to the 'insurance_final' dataset. It involves segregating the data into features (X) and labels (Y) to prepare it for modeling purposes. The 'insurance_final' dataset follows a column-based structure. In this code snippet, the features are derived into X by excluding the 'charges' column, which likely serves as the target variable for prediction. The 'charges' column remains intact and represents the labels (Y) for the model. The '.info()' method provides insights into the features within X, including their data types and non-null counts. Lastly, the 'print(Y.shape)' command aims to display the shape of the Y variable, indicating the number of rows present in the 'charges' column.

```
X_scaled = StandardScaler().fit_transform(X)
```

Figure 1. 8: Scaling Code Snippet

2.0 Impact of L1, L2, and Elastic Net on Linear Regression

Linear Regression: The linear regression model aims to predict a target variable Y based on a set of predictor variables Xi through the following formula:

$Y = \beta0 + \beta1X1 + \beta2X2 +...+ \beta nXn + \epsilon$

Here, β0 is the intercept, βi represents the coefficients for the predictor variables, Xi, and ϵ is the error term.

L1 Regularization (Lasso): L1 regularization, often referred to as Lasso, introduces a penalty term to the linear regression objective function by adding the absolute values of the coefficients. This regularization technique encourages sparsity in the model, effectively driving some coefficients to precisely zero. The consequence is a feature selection mechanism where only a subset of variables with the most significant impact on the prediction remains. L1 regularization proves advantageous in scenarios where there is a desire to identify and emphasize the most influential features, contributing to a simpler and more interpretable model. The key parameters involved are:

- penalty:
  - Description: Specifies the type of regularization. For L1 regularization, the penalty is set to 'l1'.
- alpha:
  - Description: The regularization strength. It controls the amount of shrinkage applied to the coefficients. Higher values result in more regularization.
  - Usage in Code: Grid search is performed over a range of alpha values to find the optimal one.
- eta0:
  - Description: The initial learning rate for stochastic gradient descent. It determines the size of the steps taken during optimization.
  - Usage in Code: Grid search is performed over a range of eta0 values to find the optimal one.
- max_iter:
  - Description: Maximum number of iterations for the optimization algorithm. It represents the number of passes over the training data.

- Usage in Code: Grid search is performed over a range of max_iter values to find the optimal one.

```
# Linear Regressor L1
LR1 = linear_model.SGDRegressor(random_state = 101, penalty = 'l1') # building
LR_HP = {'eta0': [.001, .01, .1, 1], 'max_iter':[10000, 20000, 30000, 40000], 'alpha': [.0001, .001, .01]}
GS_LR = GridSearchCV(estimator = LR1, param_grid = LR_HP, scoring='r2', cv=10)
GS_LR.fit(X_scaled,Y)

# results = DataFrame.from_dict(grid_search1.cv_results_)
# print("Cross-validation results:\n", results)
best_parameters_LR = GS_LR.best_params_
print("Best parameters: ", best_parameters_LR)
best_result_LR = GS_LR.best_score_
print("Best result: ", best_result_LR)
best_model_LR = GS_LR.best_estimator_
print("Intercept β0: ", best_model_LR.intercept_)
print(DataFrame(zip(X.columns, best_model_LR.coef_), columns=['Features','Coefficients']))
print(DataFrame(zip(X.columns, best_model_LR.coef_), columns=['Features','Coefficients']).sort_values(by=['Coefficients'],as
```

Figure 2. 1: Lasso Penalty on Linear Regression Model

```
Best parameters:  {'alpha': 0.0001, 'eta0': 0.001, 'max_iter': 10000}
Best result:  0.9956331337334449
Intercept β0:  [16751.03183985]
                                   Features  Coefficients
0                                       age    284.335013
1                                    gender   -494.360741
2                                       bmi    471.402175
3                                  children    330.918114
4                                    smoker   2498.198196
5                          region_northwest   -305.495339
6                          region_southeast   -220.249476
7                          region_southwest   -350.354372
8                exercise_frequency_Never   -865.227450
9         exercise_frequency_Occasionally   -429.939280
10              exercise_frequency_Rarely   -638.621025
11          medical_history_Heart disease   1295.565030
12    medical_history_High blood pressure   -440.291127
13                     medical_history_No   -864.992069
14   family_medical_history_Heart disease   1286.432351
15 family_medical_history_High blood pressure   -438.249257
16              family_medical_history_No   -848.216381
17                     occupation_Student   -449.448311
18                  occupation_Unemployed   -660.485769
19                occupation_White collar    201.202329
20                  coverage_level_Premium   2360.075722
21                 coverage_level_Standard    943.263626
                                   Features  Coefficients
8                exercise_frequency_Never   -865.227450
13                     medical_history_No   -864.992069
16              family_medical_history_No   -848.216381
18                  occupation_Unemployed   -660.485769
10              exercise_frequency_Rarely   -638.621025
1                                    gender   -494.360741
17                     occupation_Student   -449.448311
12    medical_history_High blood pressure   -440.291127
15 family_medical_history_High blood pressure   -438.249257
9         exercise_frequency_Occasionally   -429.939280
7                          region_southwest   -350.354372
5                          region_northwest   -305.495339
6                          region_southeast   -220.249476
19                occupation_White collar    201.202329
0                                       age    284.335013
```

Figure 2. 2: Best R2 Score and Coefficient of Lasso Penalty

L2 Regularization (Ridge)

In contrast, L2 regularization, commonly known as Ridge, mitigates multicollinearity and controls the magnitude of coefficients by adding the squared values of the coefficients as a penalty. Unlike L1 regularization, Ridge does not force coefficients to zero but rather reduces their overall impact. Ridge regularization is particularly beneficial in situations where there are highly correlated features. It strikes a balance by limiting the influence of each feature while retaining all variables in the model, resulting in a more interpretable regression model compared to non-regularized linear regression.

The key parameters for L2 are same as L1 and have the same effect. The regularization parameters for L2 (Ridge) in the code are identical to those for L1 (Lasso), emphasizing the consistent structure of the scikit-learn library for regularization techniques. Both L1 and L2 regularization share common hyperparameters such as 'alpha,' 'eta0,' and 'max_iter'. 'Alpha' controls the regularization strength, influencing the degree of shrinkage applied to the coefficients. Additionally, 'eta0' determines the initial learning rate for stochastic gradient descent, and 'max_iter' sets the maximum number of iterations for the optimization algorithm. This uniformity in parameter names simplifies the comparison and application of different regularization techniques, contributing to the ease of implementation and interpretation in machine learning workflows.

```
# Linear Regressor L2
LR2 = linear_model.SGDRegressor(random_state = 101, penalty = 'l2') # building
LR_HP2 = {'eta0': [.001, .01, .1, 1], 'max_iter':[10000, 20000, 30000, 40000], 'alpha': [.0001, .001, .01]}
GS_LR2 = GridSearchCV(estimator = LR2, param_grid = LR_HP2, scoring='r2', cv=10)
GS_LR2.fit(X_scaled,Y)

# results = DataFrame.from_dict(grid_search1.cv_results_)
# print("Cross-validation results:\n", results)
best_parameters_LR2 = GS_LR2.best_params_
print("Best parameters: ", best_parameters_LR2)
best_result_LR2 = GS_LR2.best_score_
print("Best result: ", best_result_LR2)
best_model_LR2 = GS_LR2.best_estimator_
print("Intercept β0: ", best_model_LR2.intercept_)
print(DataFrame(zip(X.columns, best_model_LR2.coef_), columns=['Features','Coefficients']).sort_values(by=['Coefficients'],
```

Figure 2. 3: Lasso Penalty on Linear Regression Model

```
Best parameters:  {'alpha': 0.0001, 'eta0': 0.001, 'max_iter': 10000}
Best result:  0.9956330264253426
Intercept β0:  [16751.00207053]
                                    Features  Coefficients
13                        medical_history_No   -864.875192
8                   exercise_frequency_Never   -864.783930
16                 family_medical_history_No   -848.203962
18                     occupation_Unemployed   -660.333902
10                 exercise_frequency_Rarely   -638.162156
1                                     gender   -494.340547
17                        occupation_Student   -449.288605
12          medical_history_High blood pressure   -440.259256
15   family_medical_history_High blood pressure   -438.289069
9            exercise_frequency_Occasionally   -429.597729
7                           region_southwest   -350.249812
5                           region_northwest   -305.447764
6                           region_southeast   -220.183873
19                   occupation_White collar    201.298701
0                                        age    284.356309
3                                   children    330.776807
2                                        bmi    471.359052
21                   coverage_level_Standard    942.927282
14       family_medical_history_Heart disease   1286.320136
11              medical_history_Heart disease   1295.483947
20                    coverage_level_Premium   2359.583461
4                                     smoker   2497.960348
```

Figure 2. 4: Best R2 Result and Coefficient for Lasso Penalty

Elastic Net Regularization

Elastic net regularization combines the strengths of both L1 and L2 regularization by introducing a linear combination of their penalties. This hybrid approach allows for a flexible adjustment between feature selection and regularization, controlled by a parameter alpha. Elastic net is especially useful when faced with datasets containing many features, as it provides a robust solution that considers both sparsity and magnitude control. The code employs GridSearchCV to explore different alpha values, enabling a nuanced understanding of the trade-off between L1 and L2 penalties and emphasizing the versatility of elastic net regularization in achieving model interpretability without sacrificing predictive performance. The one additional parameter that we add for Elastic Net is L1_Ratio.

- L1_ratio:
    - Description: The mixing parameter for L1 and L2 penalties. It controls the balance between L1 and L2 regularization.
    - Usage in Code: Grid search is performed over a range of l1_ratio values to find the optimal one.

```
#  Lienar Regressor Elastic Net and Regularization
LR_EN = linear_model.SGDRegressor(random_state = 101, penalty = 'elasticnet')
HP_EN = {'eta0': [.0001, .001, .01], 'max_iter':[10000, 20000, 30000],'alpha': [.0001, .001, .01], 'l1_ratio': [.4, .5, .6]}
GS_EN = GridSearchCV(estimator=LR_EN, param_grid=HP_EN, scoring='r2', cv=5)
GS_EN.fit(X_scaled, Y)

# results = DataFrame.from_dict(grid_search2.cv_results_)
# print("Cross-validation results:\n", results)
best_parameters_EN = GS_EN.best_params_
print("Best parameters: ", best_parameters_EN)
best_result_EN = GS_EN.best_score_
print("Best result: ", best_result_EN)
best_model_EN = GS_EN.best_estimator_
print("Intercept β0: ", best_model_EN.intercept_)
print(DataFrame(zip(X.columns, best_model_EN.coef_), columns=['Features','Coefficients']).sort_values(by=['Coefficients'],ascend
```

Figure 2. 5: Elastic Net Penalty on Linear Regression Model

```
Best parameters:  {'alpha': 0.0001, 'eta0': 0.001, 'l1_ratio': 0.5, 'max_iter': 10000}
Best result:  0.9956727637793593
Intercept β0:  [16751.00205168]
                                 Features  Coefficients
4                                  smoker   2498.085659
20                   coverage_level_Premium   2359.781773
11              medical_history_Heart disease   1295.529119
14       family_medical_history_Heart disease   1286.373583
21                  coverage_level_Standard    943.083546
2                                     bmi    471.389935
3                                children    330.793413
0                                     age    284.376461
19                occupation_White collar    201.274450
6                        region_southeast   -220.207240
5                        region_northwest   -305.481445
7                        region_southwest   -350.286016
9           exercise_frequency_Occasionally   -429.677295
15  family_medical_history_High blood pressure   -438.298192
12         medical_history_High blood pressure   -440.280356
17                     occupation_Student   -449.338289
1                                  gender   -494.363544
10               exercise_frequency_Rarely   -638.262735
18                  occupation_Unemployed   -660.383712
16               family_medical_history_No   -848.238965
8                 exercise_frequency_Never   -864.888543
13                      medical_history_No   -864.911749
```

Figure 2. 6: Best R2 Result and Coefficient for Elastic Net Penalty

Impact on Coefficients, Performance, and Interpretability

The impact of these regularization techniques on linear regression coefficients is substantial. L1 regularization tends to produce sparse models, enhancing interpretability by selecting a subset of the most relevant features. In contrast, L2 regularization maintains all features but reduces their impact, improving overall stability. Elastic Net strikes a compromise, offering a model that is both interpretable and stable.

Performance-wise, all three regularization techniques play a crucial role in preventing overfitting and improving model generalization but here in our code the score for all three is the same with elastic net being a little better. They contribute to enhanced predictive accuracy and robustness.

In terms of interpretability, L1 regularization stands out by providing a clear selection of influential features. L2 regularization improves stability but does not offer sparsity. Elastic Net, with its balance between feature selection and stability, strikes a chord that is particularly beneficial in real-world scenarios. The choice between these regularization techniques depends on the characteristics of the dataset and the desired trade-off between sparsity and stability in the linear regression model.

3.0 Impact of L2 on Support Vector Regression

3.1 Support Vector Regression without L2 Regularisation

Support Vector Regression (SVR) stands out as a robust tool chosen for its adeptness in handling the mixed, non-linear, and non-continuous patterns inherent in such datasets. Its application within the code signifies a strategic choice to tackle the complexity of insurance charges prediction. The addition of hyperparameter selection, which includes kernel type and epsilon optimisation via a careful grid search, demonstrates the commitment to improving the reliability of the SVR model.

The Significance of SVR Hyperparameters

Within SVR, hyperparameters serve as pivotal settings shaping the model's performance and adaptability. The choice of the kernel type, be it 'linear', 'radial basis function (rbf)', polynomial, or other kernel describes the model's ability to capture intricate relationships. Simultaneously, epsilon governs the margin around the regression line, striking a solid balance between model accuracy and complexity. The code's attention to hyperparameter tuning not only highlights technical aspect but also emphasizes the strategic importance of configuring SVR for optimal accuracy in the nuanced domain of insurance charge predictions. Since the data has a linear relationship between features and label, the suitable kernel is 'linear' and epsilon is chosen by the GridsearchCV function.

```
SVRegressor = SVR()
# Tuning kernel and epsilon
SVR_HP = {'kernel': ['linear', 'rbf'], 'epsilon': [.001, .01, .1, 1]}
grid_search_SVR = GridSearchCV(SVR(), SVR_HP, cv=5, scoring='r2')
grid_search_SVR.fit(X_scaled, Y)

best_params_SVR = grid_search_SVR.best_params_
best_model_SVR = grid_search_SVR.best_estimator_

print("Best parameters: ", best_params_SVR)
best_result_SVR = grid_search_SVR.best_score_
print("Best result: ", best_result_SVR)
print("Intercept β0: ", best_model_SVR.intercept_)

# Displaying the coeficients
# The model is train using linear kernel
coefficients = best_model_SVR.coef_[0]  # Extracting the coefficients

# Creating a DataFrame with features and their coefficients
coefficients_df = DataFrame({'Features': X.columns, 'Coefficients': coefficients})

# Sorting the DataFrame by coefficients in descending order
coefficients_df = coefficients_df.sort_values(by='Coefficients', ascending=False)
print(coefficients_df)
```

Figure 3.1: Support Vector Regression without L2 Regularization

```
Best parameters:  {'epsilon': 1, 'kernel': 'linear'}
Best result:  0.5089467324031268
Intercept β0:  [16711.42944185]
                               Features  Coefficients
4                                smoker    989.851279
20                coverage_level_Premium    671.735722
11         medical_history_Heart disease    586.628474
14  family_medical_history_Heart disease    517.257671
3                              children    167.823413
19              occupation_White collar    163.835992
2                                   bmi    123.618487
0                                   age     91.867439
21               coverage_level_Standard     12.095723
9        exercise_frequency_Occasionally    -43.331125
7                      region_southwest    -58.376070
10            exercise_frequency_Rarely    -68.582582
5                      region_northwest    -70.299577
6                      region_southeast    -75.831510
17                   occupation_Student    -96.073941
8              exercise_frequency_Never   -177.226068
1                                gender   -180.704922
12     medical_history_High blood pressure   -181.453347
15  family_medical_history_High blood pressure   -197.722568
18                occupation_Unemployed   -206.329118
16             family_medical_history_No   -345.970993
13                    medical_history_No   -349.753652
```

Figure 3.2: Coefficients and result of Support Vector Regression

3.2 Support Vector Regression with L2 Regularization

Based on figure 3.2, the R-squared result is low, indicating underfitting. To avoid underfitting or overfitting, L2 or ridge regularization called C is optimised in the model. L2 regularization helps by penalising large coefficient values, this penalty will be added into the loss function while training the model. Figure 3.3 demonstrates how L2 regularization can be tuned and used to penalise large coefficient and consequently provide a better R-squared result.

As shown in figure 3.4, the model has a significant improvement where R-squared result is 0.99558 rather than 0.50894. This shows that L2 regularization has a huge impact on the SVR because it penalises larger coefficient illustrates in figure 3.4. After applying L2 regularization, coefficient for each feature has substantially increases because large penalty was added into the loss function. This means now for each X3 (smoker) will increase the target variable by 2486.06 because of L2 regularization. However, for each X3 (smoker) will only increase the target variable by 989.85 when L2 regularization is not utilised.

```
#Support Vector Regression with L2 Regularization
from sklearn.svm import SVR

SVRegressor = SVR()
# Penalise the coefficient by adding Ridge or L2 regularization (C)
SVR_HP1 = {'kernel': ['linear', 'rbf'], 'C': [1, 100, 1000], 'epsilon': [.001, .01, .1, 1]}
grid_search_SVR1 = GridSearchCV(SVR(), SVR_HP1, cv=5, scoring='r2')
grid_search_SVR1.fit(X_scaled, Y)

best_params_SVR1 = grid_search_SVR1.best_params_
best_model_SVR1 = grid_search_SVR1.best_estimator_

print("Best parameters: ", best_params_SVR1)
best_result_SVR1 = grid_search_SVR1.best_score_
print("Best result: ", best_result_SVR1)
print("Intercept β0: ", best_model_SVR1.intercept_)

# Displaying the coeficients
# Since the model is train using linear kernel we can get the coefficients
coefficients = best_model_SVR1.coef_[0]  # Extracting the coefficients

# Creating a DataFrame with features and their coefficients
coefficients_df = DataFrame({'Features': X.columns, 'Coefficients': coefficients})

# Sorting the DataFrame by coefficients in descending order
coefficients_df = coefficients_df.sort_values(by='Coefficients', ascending=False)
print(coefficients_df)
```

Figure 3. 3: Support Vector Regression with L2 Regularization

```
Best parameters:  {'C': 100, 'epsilon': 0.01, 'kernel': 'linear'}
Best result:  0.9955809382373306
Intercept β0:  [16742.11735619]
                                  Features  Coefficients
4                                   smoker   2486.060290
20                  coverage_level_Premium   2355.544001
11           medical_history_Heart disease   1293.893638
14    family_medical_history_Heart disease   1272.698854
21                 coverage_level_Standard    933.370717
2                                      bmi    472.108185
3                                 children    333.738255
0                                      age    288.140732
19                 occupation_White collar    186.195650
6                         region_southeast   -230.873883
5                         region_northwest   -309.189343
7                         region_southwest   -346.854452
9         exercise_frequency_Occasionally   -418.409916
12     medical_history_High blood pressure   -427.178700
15 family_medical_history_High blood pressure -446.502401
17                      occupation_Student   -456.104660
1                                   gender   -495.011256
10            exercise_frequency_Rarely   -623.826024
18                   occupation_Unemployed   -668.798564
16               family_medical_history_No   -854.835667
8               exercise_frequency_Never   -859.634498
13                      medical_history_No   -861.793806
```

Figure 3. 4: Output for Support Vector Regression with L2 Regularization

Table 3.1 shows the comparison of coefficient for features that have positive impact on the target variable. This table illustrates that the L2 regularization has penalised each feature with larger coefficient to prevent overfitting and generalise the model better.

Feature	SVR Without L2 Regularization	SVR With L2 Regularization
smoker	989.85	2486.08
coverage_level_premium	671.74	2355.54
medical_history_heart_disease	586.63	1293.89
family_medical_history_heart_disease	517.26	1272.70
coverage_level_standard	167.82	933.37
children	163.84	472.11
bmi	123.62	333.74
age	91.87	288.14

Table 3. 1: Comparison of Coefficient Between SVR Model with or without L2 Regularization

In conclusion, L2 regularization penalises the coefficients for this model to encourage higher coefficient values. This penalty seeks a balance between fitting the training data well and preventing overfitting or underfitting by adding the penalty to the loss function. By penalizing large coefficients, L2 regularization helps create a more generalized model that not only performs adequately on the training data but also generalises better to new or unseen data. This balancing act ensures a model that captures essential patterns while avoiding excessive sensitivity to noise in the training data, ultimately leading to better overall performance. Hence why the L2 regularization help this model to predict the target variable accurately.

4.0 Random Forest Regression

Apart from linear and support vector regression, there is another machine learning model that can perform regression tasks called random forest regression model. It is a great algorithm for continuous data, same as the other two models. However, there are differences between these three models. Linear regression is only good when the relationship between features and target variable is linear, as opposed to support vector and random forest regression, both can handle a non-linear data. Support vector and linear regression are used when there is a need for interpretability such as understanding the feature coefficient which random forest model does not provide.

These coefficients show the impact of each feature with the target variable (label). If the coefficient for a feature called 'credit limit' is positive and it is a huge value, that means that it has a positive and enormous impact on the target variable. For instance, the coefficient for 'credit limit' is 28000.89, the target variable will increase a lot in a positive way. Contrasting to this if the coefficient is –28000.89, the target variable will decrease significantly. These coefficients are great to have as it is interpretable and can be explained for non-technical individuals. Figure 2.2, 2.4, 2.6 shows the coefficients of each feature.

For this project, random forest regression was implemented to get a better insight into which model is the best model to predict the target variable (charges). As shown in figure 2.2, 2.4, and 2.6, linear regression with lasso, ridge, and elastic net penalty has a result of 0.9956331, 0.9956330, and 0.9956727, respectively. This shows that there is no significant performance difference between these three model with different penalty. For support vector regression model, the best r2 result is 0.99558 the best kernel is 'linear' so we can find the coefficient of each feature.

```
Best parameters:  {'n_estimators': 780}
best_score:  0.9150873999645217
smoker                                      0.269964
coverage_level_Premium                      0.143703
medical_history_Heart disease               0.115869
family_medical_history_Heart disease        0.098901
bmi                                         0.054600
age                                         0.046223
medical_history_No                          0.039024
family_medical_history_No                   0.036401
children                                    0.029013
coverage_level_Standard                     0.017700
occupation_Unemployed                       0.017400
gender                                      0.016896
family_medical_history_High blood pressure  0.016599
medical_history_High blood pressure         0.015442
exercise_frequency_Never                    0.014879
occupation_White collar                     0.014690
occupation_Student                          0.009581
exercise_frequency_Rarely                   0.009309
region_southwest                            0.009024
region_northwest                            0.008503
region_southeast                            0.008442
exercise_frequency_Occasionally             0.007838
dtype: float64
```

Figure 4. 1: Output for Random Forest Regressor

Random forest model does not provide a better performance between linear and support vector regression, but it has slightly lower performance than support vector regression and linear regression which is 0.91383. This shows that random forest algorithm is good but in this case, interpretability is important so that we can see how each feature impacts the target variable. As illustrates in figure 4.2 above, random forest model does not provide the coefficient of each feature, it only provides the best features. Best features are used to reduce the complexity of the model and at the same time avoid overfitting. These features and best result are obtained after hyperparameter tuning is done which is 'n_estimators'.

Model	**R2 Result**	**Interpretability**
Support vector regression	0.99558	Yes
Linear regression	0.99567	Yes
Random forest regression	0.91383	No

Table 4. 1: Comparison of SVR, Linear Regression, and Random Forest Model

To summarise, the random forest regression algorithm lacks interpretability where support vector and linear regression, allow interpretability since these two models provide coefficients (relationship between features and target variable). Table 4.1 shows a comparison between linear regression, support vector regression, and random forest regression. In the context of an insurance dataset, ensuring interpretability is crucial to comprehend the relationships between features and the target label by leveraging coefficients. These coefficients serve as valuable indicators, aiding in understanding how each feature impacts the predicted outcomes, which is particularly valuable in explaining the factors influencing insurance-related predictions.

5.0 Prediction Using New Dataset

Using 'pickle' we can predict new or unseen data. As demonstrated in figure 5.1, joblib.dump is a function for storing an algorithm model into a pickle. This function is used to export the model into a pickle file so that no codes are needed. Figure 5.1 demonstrates how the pickle is saved using joblib.dump function and how to load the model.

```
import joblib
import csv

# Assuming X_scaled is your scaled feature matrix and Y is your target variable
# Save the model to a file
joblib.dump(best_model_SVR, "SVR.pkl")

# Load the model from file
model = joblib.load("SVR.pkl")

# Fit the model (if needed)
model.fit(X_scaled, Y)  # Replace 'Y' with your target variable
```

Figure 5. 1: Dump Pickle, Load Pickle, and Fit Pickle

A new or unseen dataset is used to make a prediction, this dataset was split prior to data preparation, model training, model testing. Initially, the dataset for training and testing has 3002 values and after splitting the dataset, updated insurance dataset has 2000 values while the test dataset has 1002 values. After that, the test dataset called 'new_data' gone through the same data preparation such as encoding and scaling as shown in figure 5.2. This is due to the model will expect 22 features and 'new_data' has 11 features prior to data preparation. We predicted 1002 'charges' based on the unseen data and some of the results are as indicated in figure 5.3.

```python
# Assuming 'new_data.csv' contains your new data
new_data = read_csv("/content/newdata.csv")
# one hot encoding map
new_data['gender'] = new_data['gender'].map({'female': 1, 'male': 0})
new_data['smoker'] = new_data['smoker'].map({'yes': 1, 'no': 0})
new_data['medical_history'].fillna('No', inplace=True)
new_data['family_medical_history'].fillna('No', inplace=True)

#one hot encoding get dummies
new_data = get_dummies(new_data, columns =
 ['region',
  'exercise_frequency',
  'medical_history',
  'family_medical_history',
  'occupation',
  'coverage_level'],
drop_first=True) # encoding

new_data_final = new_data.drop(['charges'], axis = 1) # Features
new_data_scaled = StandardScaler().fit_transform(new_data_final)
# Make predictions
predictions = svr_model.predict(new_data_scaled)
formatted_predictions = ["{:.2f}".format(pred) for pred in predictions]

# Display the formatted predictions
print(formatted_predictions)
```

Figure 5. 2: Predict New or Unseen Data and Data Preparation Code Snippet

```
['18873.81',
 '19276.50',
 '14715.13',
 '22367.50',
 '14435.14',
 '18855.20',
 '18539.70',
 '24581.72',
 '19347.40',
 '18909.56',
 '18816.96',
 '23493.02',
 '14077.06',
 '17567.88',
 '25783.01',
 '15715.73',
 '13119.27',
 '14086.44',
 '12134.21',
 '22505.11',
 '5589.10',
 '17801.27',
 '15303.11',
```

Figure 5. 3: Output of Data Prediction

Figure 5.4 illustrates the comparison between the actual charges and predicted charges and how to concatenate the target variable from 'new_data' to the predicted data. This will allow us to get a better insight on the predicted data. The prediction is remarkably close to the actual 'charges' for each row, this is due to the model has 0.99 or 99%.

```
prediction_charges = read_csv('/content/predictions_charges.csv')

# Assuming 'column_to_add' is the specific column from new_data.csv to add
prediction_charges['charges'] = new_data['charges']

prediction_charges.head(10)
```

	Predicted Charges	charges
0	18873.81	18313.123831
1	19276.50	19532.656450
2	14715.13	14398.700361
3	22367.50	22351.083539
4	14435.14	14124.244544
5	18855.20	18622.329145
6	18539.70	18320.988319
7	24581.72	24350.724613
8	19347.40	19425.883307
9	18909.56	19120.630257

Figure 5. 4: Comparison between Predicted Charges and Actual Charges

Overall, among the various models assessed for predicting insurance charges, the linear regression model using an 'elastic net penalty' achieved the highest R-squared score. This score measures how well the model fits the data, and in our case, it indicates that this specific linear model captured the relationships in the data more accurately compared to other models like random forest and support vector regression.

The reason behind its slightly better performance lies in the data, which exhibits linear patterns. Linear regression excels precisely in handling these types of relationships, making it a highly suitable choice for our dataset.

Additionally, the provision of coefficients by this linear model offers a clear understanding of how each variable impacts the predicted insurance charges. These coefficients serve as numerical guides, enabling straightforward explanations of the factors influencing predictions, a crucial aspect in comprehensible and transparent modelling.

Upon meticulous comparison considering R-squared scores and interpretability, the linear regression model with the elastic net penalty emerged as the optimal choice for predicting insurance charges in our dataset, owing to its superior fit and explanatory power.

6.0 Contribution

Assignment Question	Contributed By
Data Preparation (Question 1)	Shahzad Saeed - 10627692
Impact of L1, L2, and Elastic Net on linear regression coefficients (Question 2)	Vedant Tomer - 20015122
Impact of L2 regularization on SVR performance and interpretability (Question 3)	Muhammad Shaqif Syauqi Bin Mohd Syukri – 20014235
Random forest regression (Question 4)	Muhammad Shaqif Syauqi Bin Mohd Syukri – 20014235 Vedant Tomer - 20015122 Shahzad Saeed - 10627692
Performing prediction on new data (Question 5)	Muhammad Shaqif Syauqi Bin Mohd Syukri – 20014235 Vedant Tomer - 20015122 Shahzad Saeed - 10627692