भारतीय सूचना प्रौद्योगिकी संस्थान, नागपुर
Indian Institute of Information Technology, Nagpur
An Institution of National Importance By An Act of Parliament

Report: Predicting Medical Insurance Charges using comparison of Four Regression Models

Under the guidance of:

Dr Shishupal Kumar

Submitted by

Soumalya Roy - BT20CSE058

Mridul Gupta - BT20CSE092

I. Abstract

Insurance is a policy that helps to cover up all loss or decrease loss in terms of expenses incurred by various risks. A number of variables affect how much insurance costs. These considerations of different factors contribute to the insurance policy cost expression.

Machine Learning(ML) and Big Data in the insurance sector can make insurance more effective. In the domains of computational and applied mathematics the machine learning (ML) is a well-known research area. ML is one of the computational intelligence aspects when it comes to exploitation of historical data that may be addressed in a wide range of applications and systems. There are some limitations in ML so; Predicting medical insurance costs using ML approaches is still a problem in the healthcare industry and thus it requires few more investigation and improvement.

Using the machine learning algorithms, this study provides a computational intelligence approach for predicting healthcare insurance costs. The proposed research approach uses Linear Regression, Decision Tree Regression , Random Forest Regression and Polunomial Expansion Regression and also streamlit as a framework. We had used a medical insurance cost dataset that was acquired from the **KAGGLE** repository for the cost prediction purpose, and machine learning methods are used to show the forecasting of insurance costs by regression model comparing their accuracies.

II. Introduction

The value of insurance in the lives of individuals. That's why it becomes important for insurance companies to be sufficiently precise to measure the amount covered by this specific policy and the insurance charges which must be paid for it. Various parameters or factors play an important role in estimating the insurance charges and Each of these is important. If any factor is omitted or changed when the amounts are computed then, the overall policy cost changes. It is therefore very critical to carry out these tasks with high accuracy. So, the possibility of human mistakes are high so insurance agents also use different tools to calculate the insurance premium. And thus ML is beneficial here. ML may generalize the effort or method to formulate the policy. These ML models can be learned by themselves.

The model is trained on insurance data from the past. The model can then accurately predict insurance policy costs by using the necessary elements to measure the payments as its inputs. This decreases human effort and resources and improves the company's profitability. Thus the accuracy can be improved with ML. Our goal is to predict insurance costs. The value of insurance fees is based on different variables. As a result, insurance fees are continuous. Regression is the best choice available to fulfill our needs. We use multiple linear regression in this analysis since there are many independent variables used to calculate the dependent(target) variable. For this study, the dataset for cost of health insurance is used.

Preprocessing of the dataset done first. Then we trained regression models with training data and finally evaluated these models based on testing data. In this article, we used several models of regression, for example, Multiple linear regression, Decision Tree Regression, Random Forest Regression and Polynomial Expanison Regression. It is found that the Polynomial Expansion after removal of two attributes from the features provides the highest accuracy with an r-squared value of 84.8253. The inclusion of a novel method of insurance cost estimation is the main goal of this work.

III. Dataset

We had used a dataset from Kaggle Site for creating our prediction model. This data set includes nine attributes and the data set has splitted into two-parts : training data and testing data.For training the model, 75% of total data is used and the rest for testing.To build a predictor model of medical insurance cost the training dataset is applied and to evaluate the regression model, test set is used. The following table shows the Description of the Dataset.

The dataset used in this project is the Medical Cost Personal Datasets obtained from Kaggle. The dataset consists of information about the medical and personal of patients, as well as the charges billed by the hospital. The dataset contains 1338 observations with 7 features as described below:

1. age: age of primary beneficiary

2. sex: gender of primary beneficiary (male = 0, female = 1)

3. bmi: body mass index of primary beneficiary

4. children: number of children covered by health insurance

5. smoker: smoking status of primary beneficiary (yes = 1, no = 0)

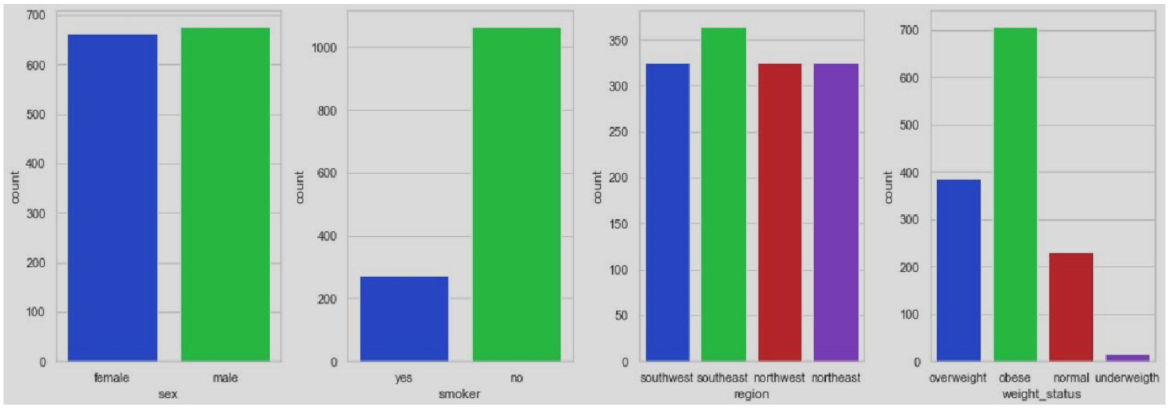
6. region : where the beneficiary resides (northeast = 0, northwest = 1, southeast = 2, southwest = 3)

7. charges: individual medical costs billed by health insurance.

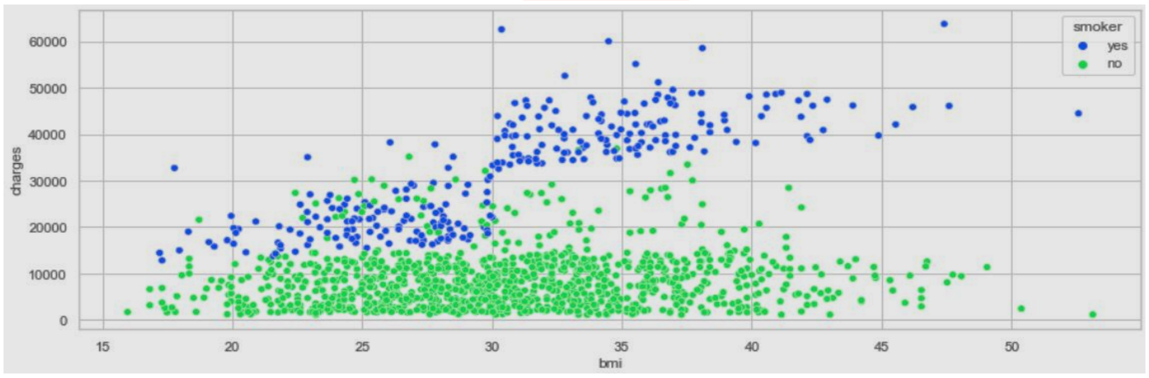
The dataset is a combination of categorical and numerical data. The 'sex', 'smoker', and 'region' features are categorical while 'age', 'bmi', 'children',and 'charges' features are numerical. This dataset is relevant for predicting medical charges of patients based on their personal and medical information. This information can be useful for insurance companies to determine the premiums for their customers based on their health risks and medical history. The dataset is relatively clean and free of missing values, making it suitable for machine learning analysis.

IV. Exploratory Data Analysis

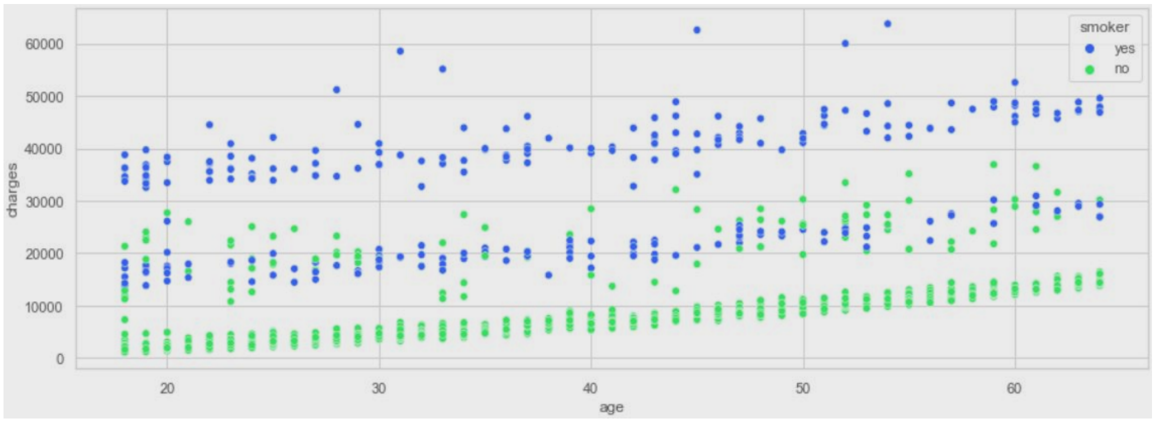
- Feature sex, region has an almost balanced amount, meanwhile most people are non smoker & obese



- A person who smoke and have BMI above 30 tends to have a higher medical cost.



- Older people who smoke have more expensive charges.



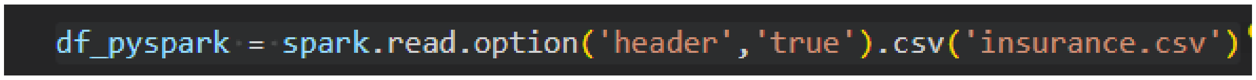
V. Tools Used:

To build the solution, we used the following tools:

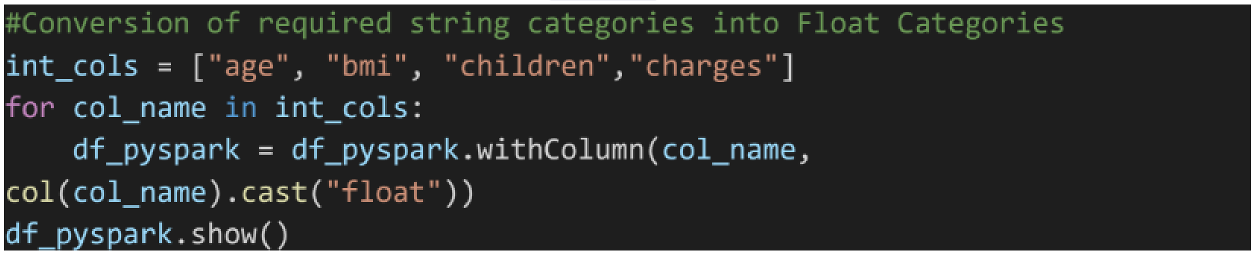
- PySpark - for data manipulation and analysis.
- Seaborn, Numpy, Matplotlib, and Pandas - for data visualization and analysis.
- CrossValidator and MulticlassClassificationEvaluator - for building and evaluating our model.
- Evaluation Metric.
- Our goal is to achieve an accuracy of 85% or higher when predicting whether a patient has heart disease or not.

VI. Method to solve the problem:

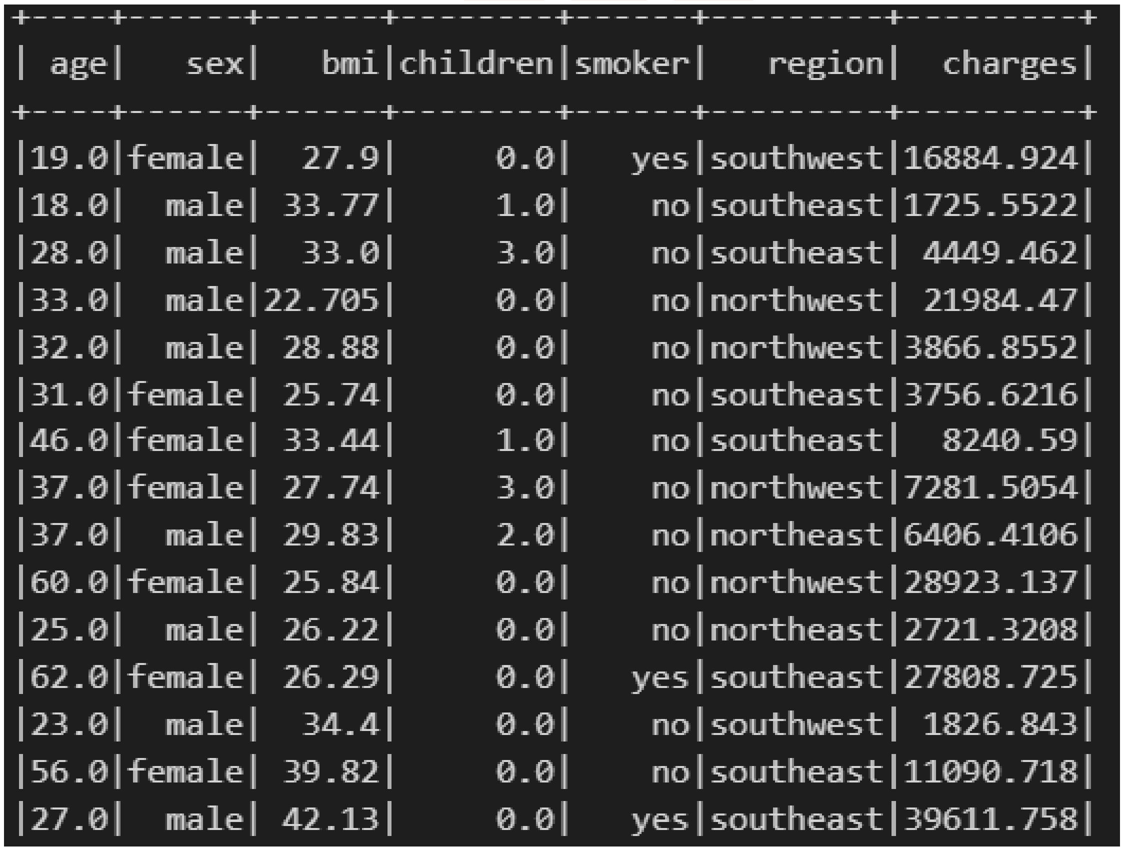
- **Data Preparation:** We will import the dataset into a PySpark dataframe and perform data cleaning, exploration, and preprocessing. This will include handling missing values, encoding categorical variables, and scaling the numerical features. Encoding sex, region, & smoker attribute using Vector Assembler in PySpark. Conversion of age,bmi,children and charges to float values.



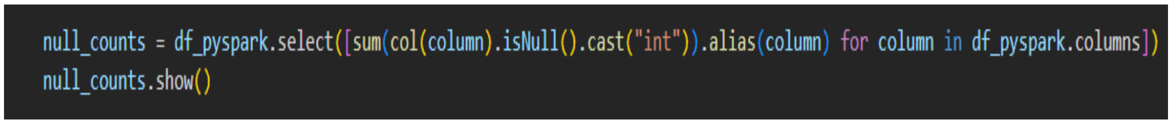
```
df_pyspark = spark.read.option('header','true').csv('insurance.csv')
```



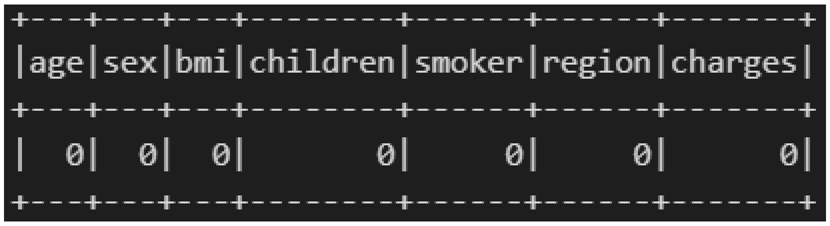
```
#Conversion of required string categories into Float Categories
int_cols = ["age", "bmi", "children","charges"]
for col_name in int_cols:
    df_pyspark = df_pyspark.withColumn(col_name,
col(col_name).cast("float"))
df_pyspark.show()
```



```
+----+------+------+--------+------+---------+---------+
| age|   sex|   bmi|children|smoker|   region|  charges|
+----+------+------+--------+------+---------+---------+
|19.0|female|  27.9|     0.0|   yes|southwest|16884.924|
|18.0|  male| 33.77|     1.0|    no|southeast|1725.5522|
|28.0|  male|  33.0|     3.0|    no|southeast| 4449.462|
|33.0|  male|22.705|     0.0|    no|northwest| 21984.47|
|32.0|  male| 28.88|     0.0|    no|northwest|3866.8552|
|31.0|female| 25.74|     0.0|    no|southeast|3756.6216|
|46.0|female| 33.44|     1.0|    no|southeast|  8240.59|
|37.0|female| 27.74|     3.0|    no|northwest|7281.5054|
|37.0|  male| 29.83|     2.0|    no|northeast|6406.4106|
|60.0|female| 25.84|     0.0|    no|northwest|28923.137|
|25.0|  male| 26.22|     0.0|    no|northeast|2721.3208|
|62.0|female| 26.29|     0.0|   yes|southeast|27808.725|
|23.0|  male|  34.4|     0.0|    no|southwest| 1826.843|
|56.0|female| 39.82|     0.0|    no|southeast|11090.718|
|27.0|  male| 42.13|     0.0|   yes|southeast|39611.758|
```

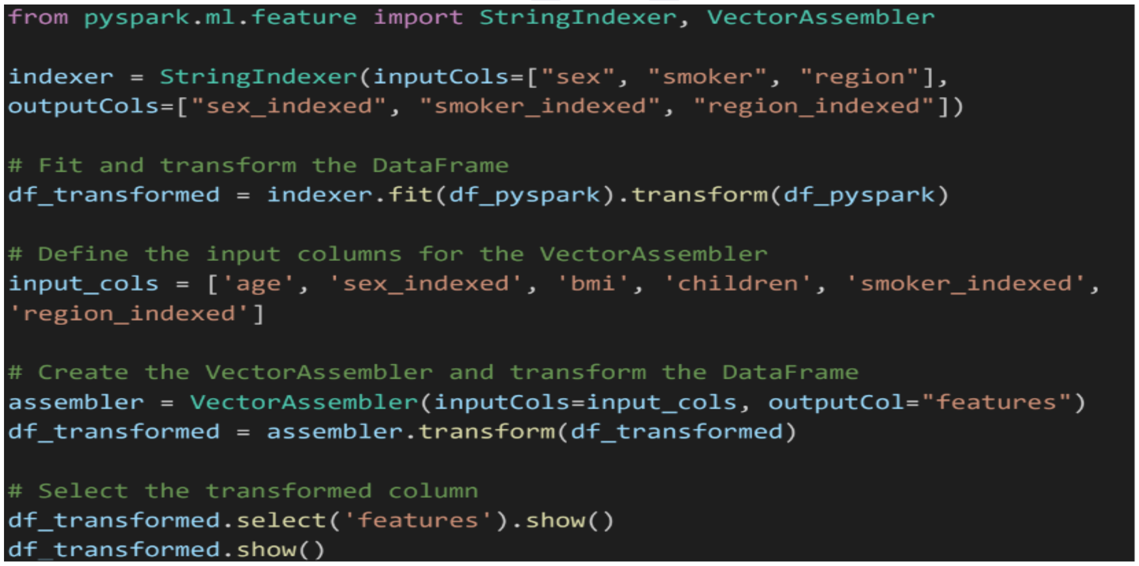


```
null_counts = df_pyspark.select([sum(col(column).isNull().cast("int")).alias(column) for column in df_pyspark.columns])
null_counts.show()
```

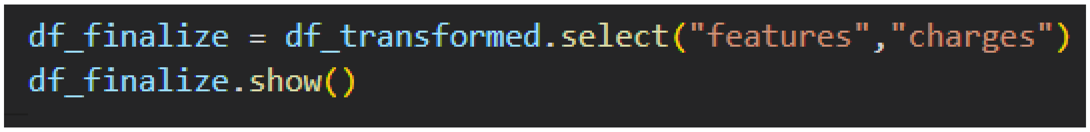


```
+---+---+---+--------+------+------+-------+
|age|sex|bmi|children|smoker|region|charges|
+---+---+---+--------+------+------+-------+
|  0|  0|  0|       0|     0|     0|      0|
+---+---+---+--------+------+------+-------+
```

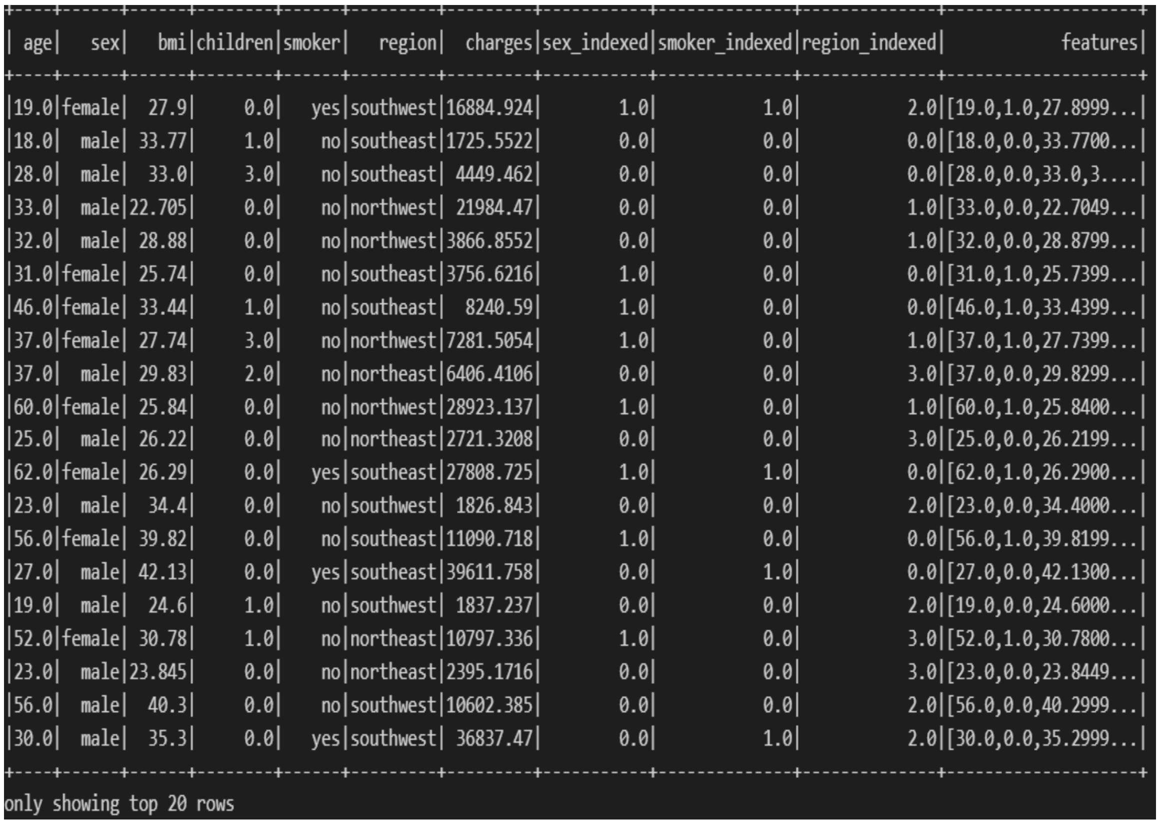
- **Feature Engineering:** Assembling features together to gather for a target (charges) attribute.



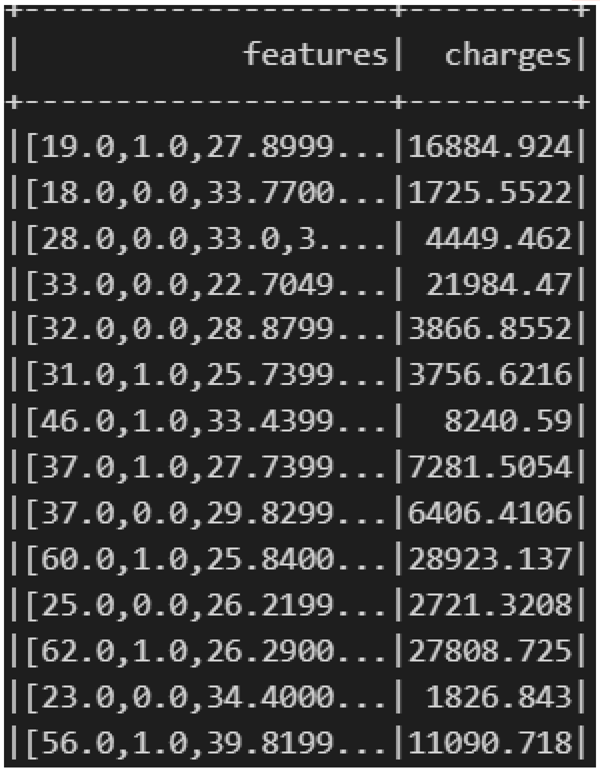
```
from pyspark.ml.feature import StringIndexer, VectorAssembler

indexer = StringIndexer(inputCols=["sex", "smoker", "region"],
outputCols=["sex_indexed", "smoker_indexed", "region_indexed"])

# Fit and transform the DataFrame
df_transformed = indexer.fit(df_pyspark).transform(df_pyspark)

# Define the input columns for the VectorAssembler
input_cols = ['age', 'sex_indexed', 'bmi', 'children', 'smoker_indexed',
'region_indexed']

# Create the VectorAssembler and transform the DataFrame
assembler = VectorAssembler(inputCols=input_cols, outputCol="features")
df_transformed = assembler.transform(df_transformed)

# Select the transformed column
df_transformed.select('features').show()
df_transformed.show()
```



```
df_finalize = df_transformed.select("features","charges")
df_finalize.show()
```



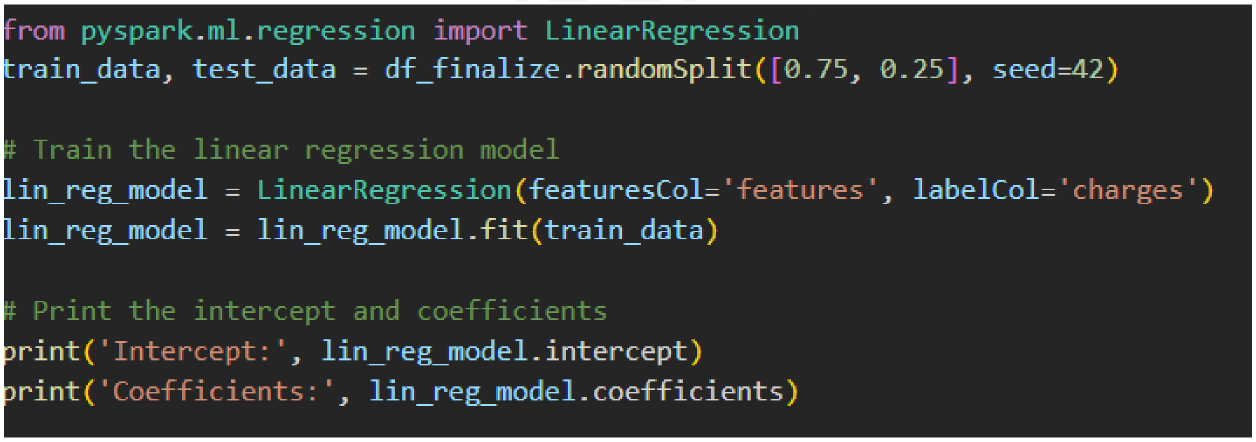
```
+----+------+------+--------+------+---------+---------+-----------+--------------+--------------+--------------------+
| age|   sex|   bmi|children|smoker|   region|  charges|sex_indexed|smoker_indexed|region_indexed|            features|
+----+------+------+--------+------+---------+---------+-----------+--------------+--------------+--------------------+
|19.0|female|  27.9|     0.0|   yes|southwest|16884.924|        1.0|           1.0|           2.0|[19.0,1.0,27.8999...|
|18.0|  male| 33.77|     1.0|    no|southeast|1725.5522|        0.0|           0.0|           0.0|[18.0,0.0,33.7700...|
|28.0|  male|  33.0|     3.0|    no|southeast| 4449.462|        0.0|           0.0|           0.0|[28.0,0.0,33.0,3....|
|33.0|  male|22.705|     0.0|    no|northwest| 21984.47|        0.0|           0.0|           1.0|[33.0,0.0,22.7049...|
|32.0|  male| 28.88|     0.0|    no|northwest|3866.8552|        0.0|           0.0|           1.0|[32.0,0.0,28.8799...|
|31.0|female| 25.74|     0.0|    no|southeast|3756.6216|        1.0|           0.0|           0.0|[31.0,1.0,25.7399...|
|46.0|female| 33.44|     1.0|    no|southeast|  8240.59|        1.0|           0.0|           0.0|[46.0,1.0,33.4399...|
|37.0|female| 27.74|     3.0|    no|northwest|7281.5054|        1.0|           0.0|           1.0|[37.0,1.0,27.7399...|
|37.0|  male| 29.83|     2.0|    no|northeast|6406.4106|        0.0|           0.0|           3.0|[37.0,0.0,29.8299...|
|60.0|female| 25.84|     0.0|    no|northwest|28923.137|        1.0|           0.0|           1.0|[60.0,1.0,25.8400...|
|25.0|  male| 26.22|     0.0|    no|northeast|2721.3208|        0.0|           0.0|           3.0|[25.0,0.0,26.2199...|
|62.0|female| 26.29|     0.0|   yes|southeast|27808.725|        1.0|           1.0|           0.0|[62.0,1.0,26.2900...|
|23.0|  male|  34.4|     0.0|    no|southwest| 1826.843|        0.0|           0.0|           2.0|[23.0,0.0,34.4000...|
|56.0|female| 39.82|     0.0|    no|southeast|11090.718|        1.0|           0.0|           0.0|[56.0,1.0,39.8199...|
|27.0|  male| 42.13|     0.0|   yes|southeast|39611.758|        0.0|           1.0|           0.0|[27.0,0.0,42.1300...|
|19.0|  male|  24.6|     1.0|    no|southwest| 1837.237|        0.0|           0.0|           2.0|[19.0,0.0,24.6000...|
|52.0|female| 30.78|     1.0|    no|northeast|10797.336|        1.0|           0.0|           3.0|[52.0,1.0,30.7800...|
|23.0|  male|23.845|     0.0|    no|northeast|2395.1716|        0.0|           0.0|           3.0|[23.0,0.0,23.8449...|
|56.0|  male|  40.3|     0.0|    no|southwest|10602.385|        0.0|           0.0|           2.0|[56.0,0.0,40.2999...|
|30.0|  male|  35.3|     0.0|   yes|southwest| 36837.47|        0.0|           1.0|           2.0|[30.0,0.0,35.2999...|
+----+------+------+--------+------+---------+---------+-----------+--------------+--------------+--------------------+
only showing top 20 rows
```



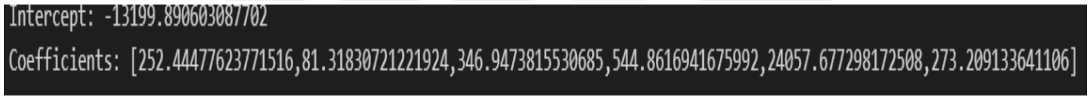
```
+--------------------+---------+
|            features|  charges|
+--------------------+---------+
|[19.0,1.0,27.8999...|16884.924|
|[18.0,0.0,33.7700...|1725.5522|
|[28.0,0.0,33.0,3....| 4449.462|
|[33.0,0.0,22.7049...| 21984.47|
|[32.0,0.0,28.8799...|3866.8552|
|[31.0,1.0,25.7399...|3756.6216|
|[46.0,1.0,33.4399...|  8240.59|
|[37.0,1.0,27.7399...|7281.5054|
|[37.0,0.0,29.8299...|6406.4106|
|[60.0,1.0,25.8400...|28923.137|
|[25.0,0.0,26.2199...|2721.3208|
|[62.0,1.0,26.2900...|27808.725|
|[23.0,0.0,34.4000...| 1826.843|
|[56.0,1.0,39.8199...|11090.718|
```

- **Modeling** - Train and evaluate several models to identify the best performing one. Further experimentation to improve our model's accuracy.

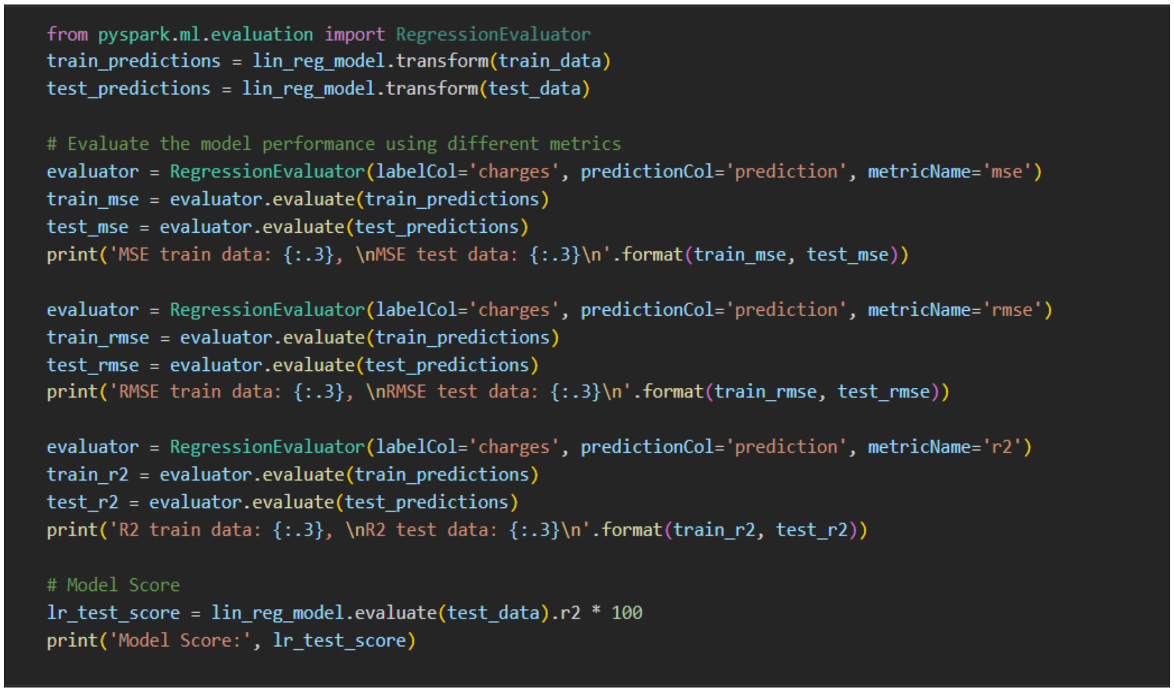
1. **Linear Regression :** Linear regression is a statistical model that is used to predict the relationship between two variables, where one variable is independent and the other is dependent. In this algorithm, a straight line is fitted to the data to create a model that can be used to predict the dependent variable based on the independent variable. The linear regression model aims to minimize the sum of the squares of the differences between the predicted values and the actual values.



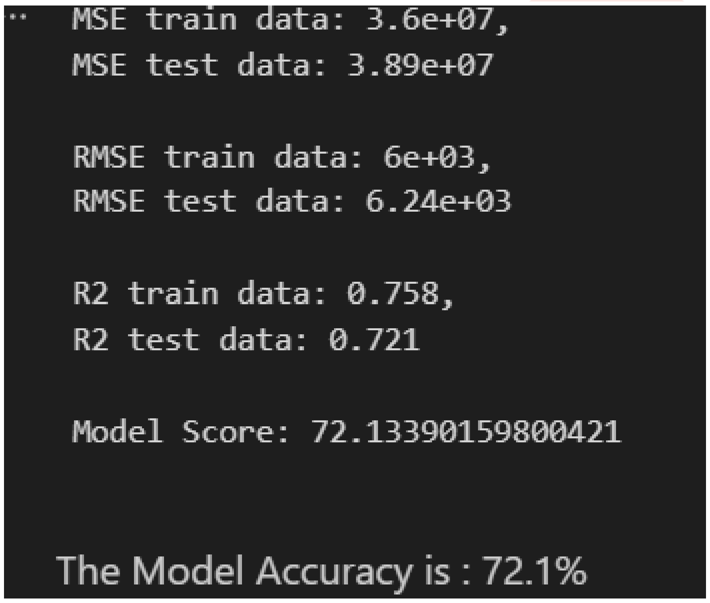
```
from pyspark.ml.regression import LinearRegression
train_data, test_data = df_finalize.randomSplit([0.75, 0.25], seed=42)

# Train the linear regression model
lin_reg_model = LinearRegression(featuresCol='features', labelCol='charges')
lin_reg_model = lin_reg_model.fit(train_data)

# Print the intercept and coefficients
print('Intercept:', lin_reg_model.intercept)
print('Coefficients:', lin_reg_model.coefficients)
```



```
Intercept: -13199.890603087702
Coefficients: [252.44477623771516,81.31830721221924,346.9473815530685,544.8616941675992,24057.677298172508,273.209133641106]
```



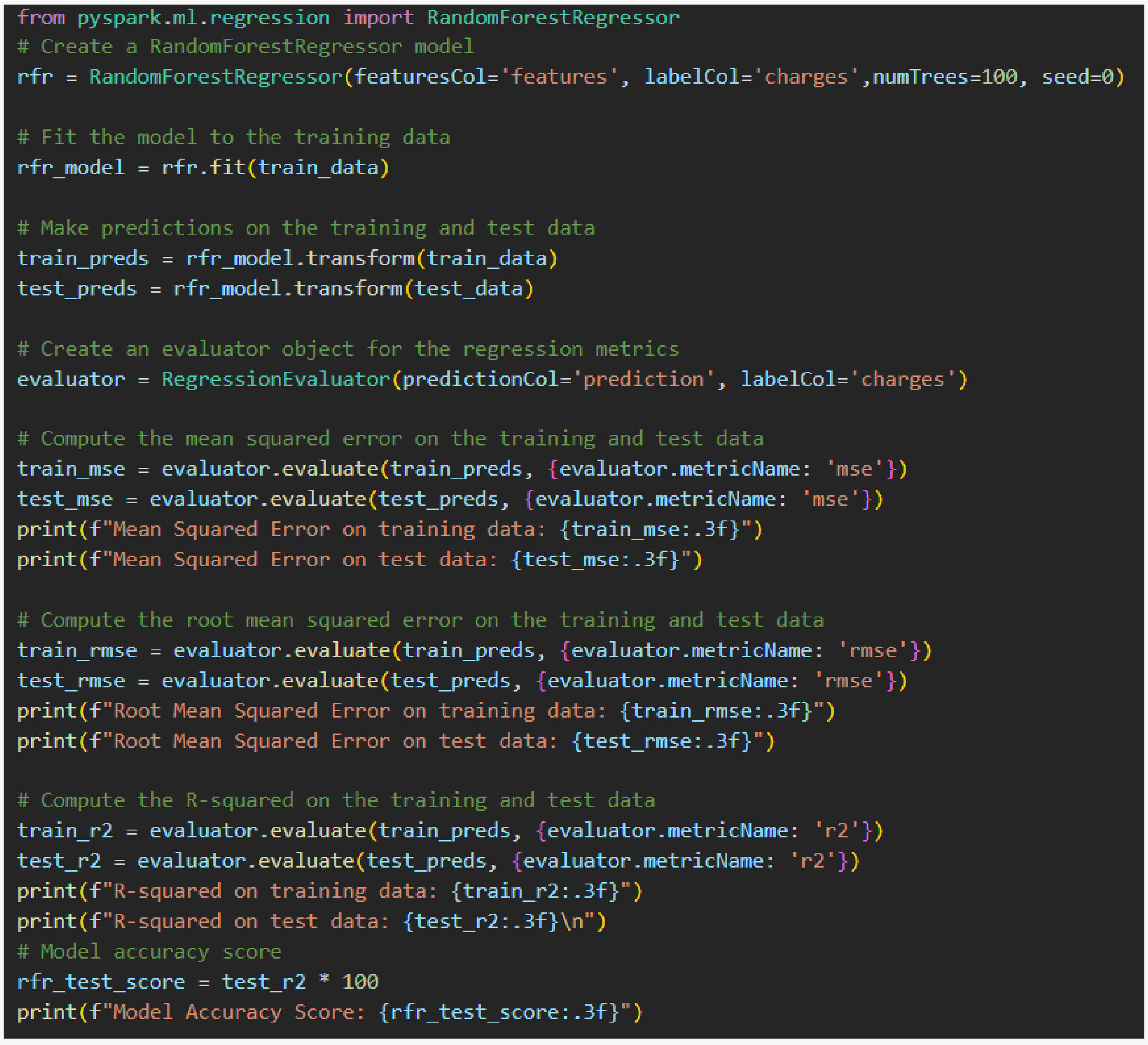
```python
from pyspark.ml.evaluation import RegressionEvaluator
train_predictions = lin_reg_model.transform(train_data)
test_predictions = lin_reg_model.transform(test_data)

# Evaluate the model performance using different metrics
evaluator = RegressionEvaluator(labelCol='charges', predictionCol='prediction', metricName='mse')
train_mse = evaluator.evaluate(train_predictions)
test_mse = evaluator.evaluate(test_predictions)
print('MSE train data: {:.3}, \nMSE test data: {:.3}\n'.format(train_mse, test_mse))

evaluator = RegressionEvaluator(labelCol='charges', predictionCol='prediction', metricName='rmse')
train_rmse = evaluator.evaluate(train_predictions)
test_rmse = evaluator.evaluate(test_predictions)
print('RMSE train data: {:.3}, \nRMSE test data: {:.3}\n'.format(train_rmse, test_rmse))

evaluator = RegressionEvaluator(labelCol='charges', predictionCol='prediction', metricName='r2')
train_r2 = evaluator.evaluate(train_predictions)
test_r2 = evaluator.evaluate(test_predictions)
print('R2 train data: {:.3}, \nR2 test data: {:.3}\n'.format(train_r2, test_r2))

# Model Score
lr_test_score = lin_reg_model.evaluate(test_data).r2 * 100
print('Model Score:', lr_test_score)
```



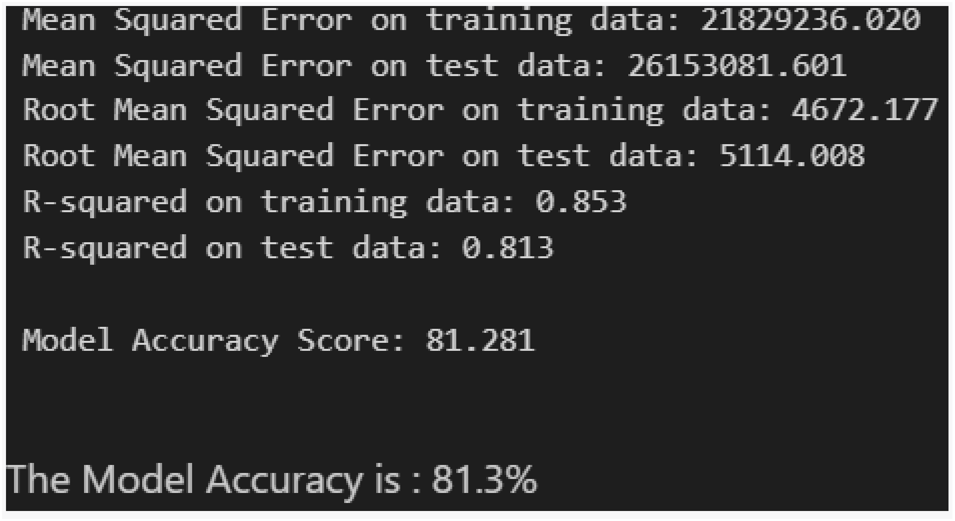
```
MSE train data: 3.6e+07,
MSE test data: 3.89e+07

RMSE train data: 6e+03,
RMSE test data: 6.24e+03

R2 train data: 0.758,
R2 test data: 0.721

Model Score: 72.13390159800421
```

The Model Accuracy is : 72.1%

2. **Random Forest Regression :** Random Forest Regression is a type of ensemble learning algorithm that combines multiple decision trees to create a more accurate model. It works by creating multiple decision trees and then combining them to get a more accurate prediction. Each tree is trained on a random subset of the data, and each node in the tree is split based on the best feature that minimizes the variance of the target variable. The final prediction is the average of all the predictions made by the individual trees.

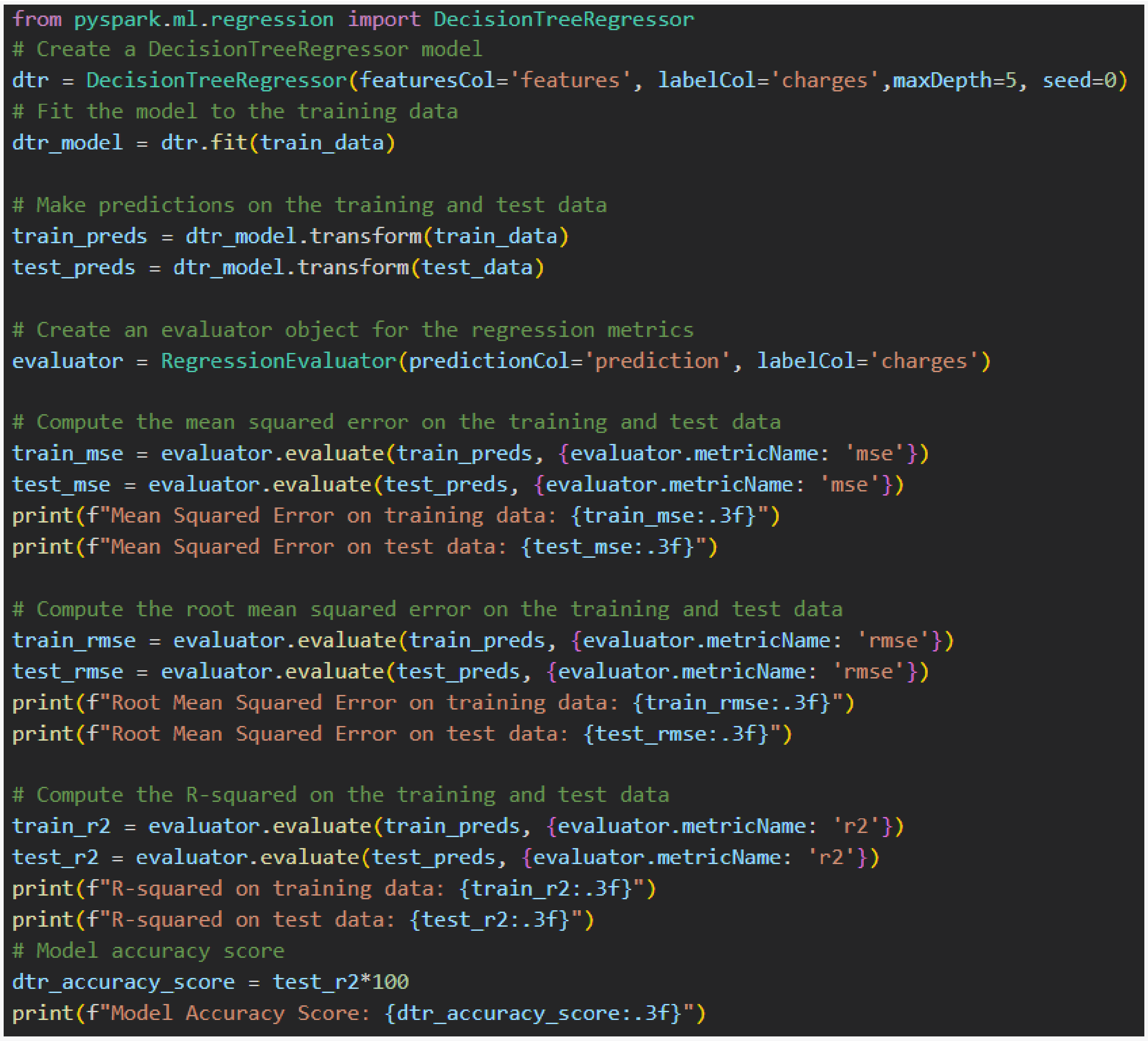


```
from pyspark.ml.regression import RandomForestRegressor
# Create a RandomForestRegressor model
rfr = RandomForestRegressor(featuresCol='features', labelCol='charges',numTrees=100, seed=0)

# Fit the model to the training data
rfr_model = rfr.fit(train_data)

# Make predictions on the training and test data
train_preds = rfr_model.transform(train_data)
test_preds = rfr_model.transform(test_data)

# Create an evaluator object for the regression metrics
evaluator = RegressionEvaluator(predictionCol='prediction', labelCol='charges')

# Compute the mean squared error on the training and test data
train_mse = evaluator.evaluate(train_preds, {evaluator.metricName: 'mse'})
test_mse = evaluator.evaluate(test_preds, {evaluator.metricName: 'mse'})
print(f"Mean Squared Error on training data: {train_mse:.3f}")
print(f"Mean Squared Error on test data: {test_mse:.3f}")

# Compute the root mean squared error on the training and test data
train_rmse = evaluator.evaluate(train_preds, {evaluator.metricName: 'rmse'})
test_rmse = evaluator.evaluate(test_preds, {evaluator.metricName: 'rmse'})
print(f"Root Mean Squared Error on training data: {train_rmse:.3f}")
print(f"Root Mean Squared Error on test data: {test_rmse:.3f}")

# Compute the R-squared on the training and test data
train_r2 = evaluator.evaluate(train_preds, {evaluator.metricName: 'r2'})
test_r2 = evaluator.evaluate(test_preds, {evaluator.metricName: 'r2'})
print(f"R-squared on training data: {train_r2:.3f}")
print(f"R-squared on test data: {test_r2:.3f}\n")
# Model accuracy score
rfr_test_score = test_r2 * 100
print(f"Model Accuracy Score: {rfr_test_score:.3f}")
```

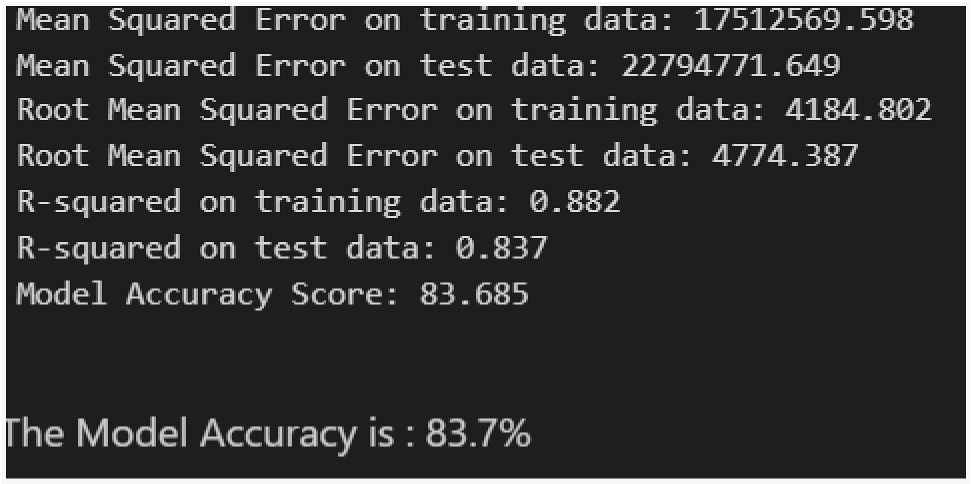


```
Mean Squared Error on training data: 21829236.020
Mean Squared Error on test data: 26153081.601
Root Mean Squared Error on training data: 4672.177
Root Mean Squared Error on test data: 5114.008
R-squared on training data: 0.853
R-squared on test data: 0.813

Model Accuracy Score: 81.281


The Model Accuracy is : 81.3%
```

3. **Decision Trees Regression :** Decision Tree Regression is a non-parametric supervised learning algorithm that is used for both classification and regression problems. It works by creating a tree-like model of decisions and their possible consequences. Each node in the tree represents a decision, and each branch represents a possible outcome. The goal of the decision tree algorithm is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.



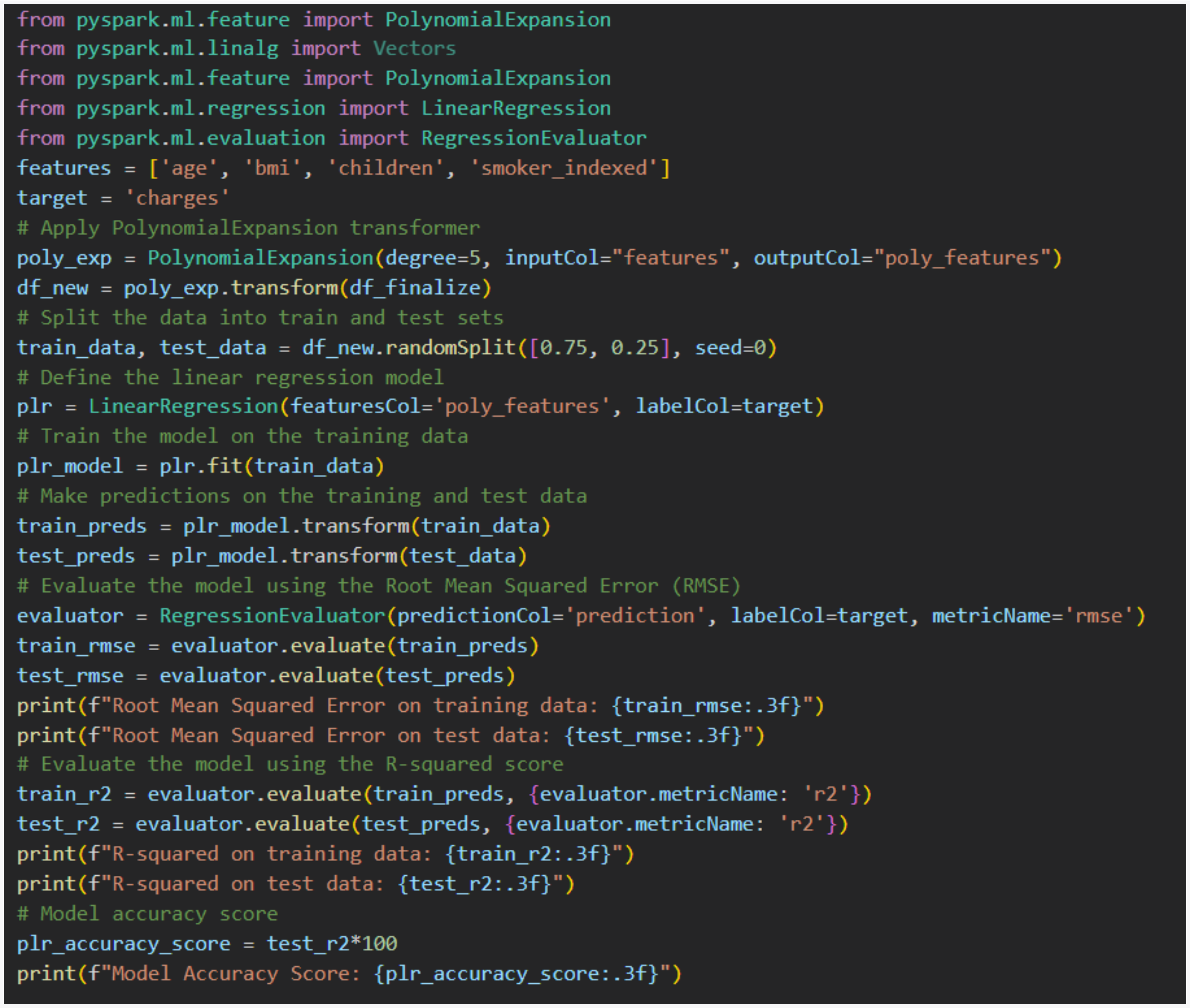
```python
from pyspark.ml.regression import DecisionTreeRegressor
# Create a DecisionTreeRegressor model
dtr = DecisionTreeRegressor(featuresCol='features', labelCol='charges',maxDepth=5, seed=0)
# Fit the model to the training data
dtr_model = dtr.fit(train_data)

# Make predictions on the training and test data
train_preds = dtr_model.transform(train_data)
test_preds = dtr_model.transform(test_data)

# Create an evaluator object for the regression metrics
evaluator = RegressionEvaluator(predictionCol='prediction', labelCol='charges')

# Compute the mean squared error on the training and test data
train_mse = evaluator.evaluate(train_preds, {evaluator.metricName: 'mse'})
test_mse = evaluator.evaluate(test_preds, {evaluator.metricName: 'mse'})
print(f"Mean Squared Error on training data: {train_mse:.3f}")
print(f"Mean Squared Error on test data: {test_mse:.3f}")

# Compute the root mean squared error on the training and test data
train_rmse = evaluator.evaluate(train_preds, {evaluator.metricName: 'rmse'})
test_rmse = evaluator.evaluate(test_preds, {evaluator.metricName: 'rmse'})
print(f"Root Mean Squared Error on training data: {train_rmse:.3f}")
print(f"Root Mean Squared Error on test data: {test_rmse:.3f}")

# Compute the R-squared on the training and test data
train_r2 = evaluator.evaluate(train_preds, {evaluator.metricName: 'r2'})
test_r2 = evaluator.evaluate(test_preds, {evaluator.metricName: 'r2'})
print(f"R-squared on training data: {train_r2:.3f}")
print(f"R-squared on test data: {test_r2:.3f}")
# Model accuracy score
dtr_accuracy_score = test_r2*100
print(f"Model Accuracy Score: {dtr_accuracy_score:.3f}")
```



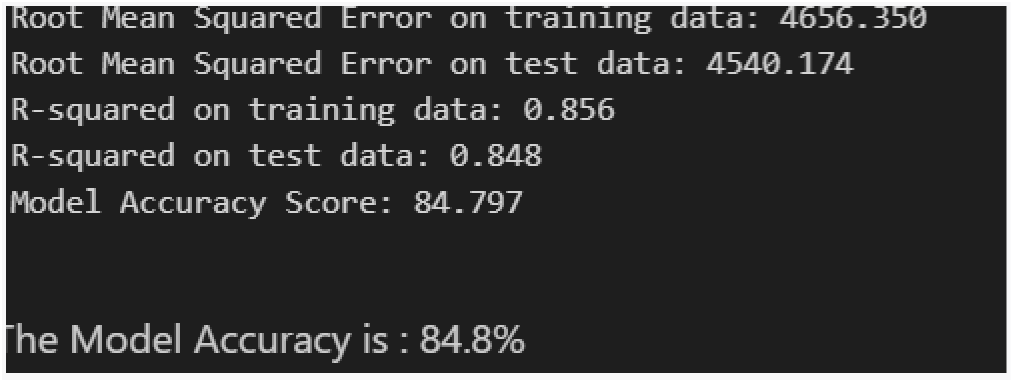
```
Mean Squared Error on training data: 17512569.598
Mean Squared Error on test data: 22794771.649
Root Mean Squared Error on training data: 4184.802
Root Mean Squared Error on test data: 4774.387
R-squared on training data: 0.882
R-squared on test data: 0.837
Model Accuracy Score: 83.685
```

The Model Accuracy is : 83.7%

4. **Polynomial Expansion Regression:** Polynomial Regression is a regression algorithm that models the relationship between the independent and dependent variables as an nth-degree polynomial. The polynomial regression model aims to fit a curve to the data that is not a straight line. It is useful in situations where a linear model is not adequate to model the relationship between the variables. Polynomial regression works by adding polynomial terms of the independent variable to the model, which increases the complexity of the model but also makes it more accurate.



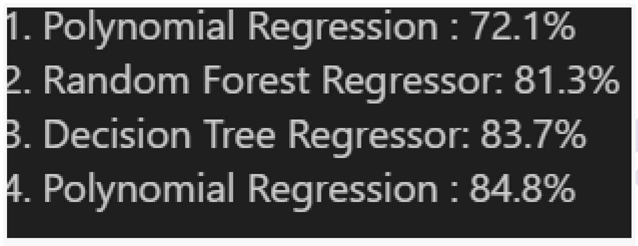
```
from pyspark.ml.feature import PolynomialExpansion
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import PolynomialExpansion
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator
features = ['age', 'bmi', 'children', 'smoker_indexed']
target = 'charges'
# Apply PolynomialExpansion transformer
poly_exp = PolynomialExpansion(degree=5, inputCol="features", outputCol="poly_features")
df_new = poly_exp.transform(df_finalize)
# Split the data into train and test sets
train_data, test_data = df_new.randomSplit([0.75, 0.25], seed=0)
# Define the linear regression model
plr = LinearRegression(featuresCol='poly_features', labelCol=target)
# Train the model on the training data
plr_model = plr.fit(train_data)
# Make predictions on the training and test data
train_preds = plr_model.transform(train_data)
test_preds = plr_model.transform(test_data)
# Evaluate the model using the Root Mean Squared Error (RMSE)
evaluator = RegressionEvaluator(predictionCol='prediction', labelCol=target, metricName='rmse')
train_rmse = evaluator.evaluate(train_preds)
test_rmse = evaluator.evaluate(test_preds)
print(f"Root Mean Squared Error on training data: {train_rmse:.3f}")
print(f"Root Mean Squared Error on test data: {test_rmse:.3f}")
# Evaluate the model using the R-squared score
train_r2 = evaluator.evaluate(train_preds, {evaluator.metricName: 'r2'})
test_r2 = evaluator.evaluate(test_preds, {evaluator.metricName: 'r2'})
print(f"R-squared on training data: {train_r2:.3f}")
print(f"R-squared on test data: {test_r2:.3f}")
# Model accuracy score
plr_accuracy_score = test_r2*100
print(f"Model Accuracy Score: {plr_accuracy_score:.3f}")
```



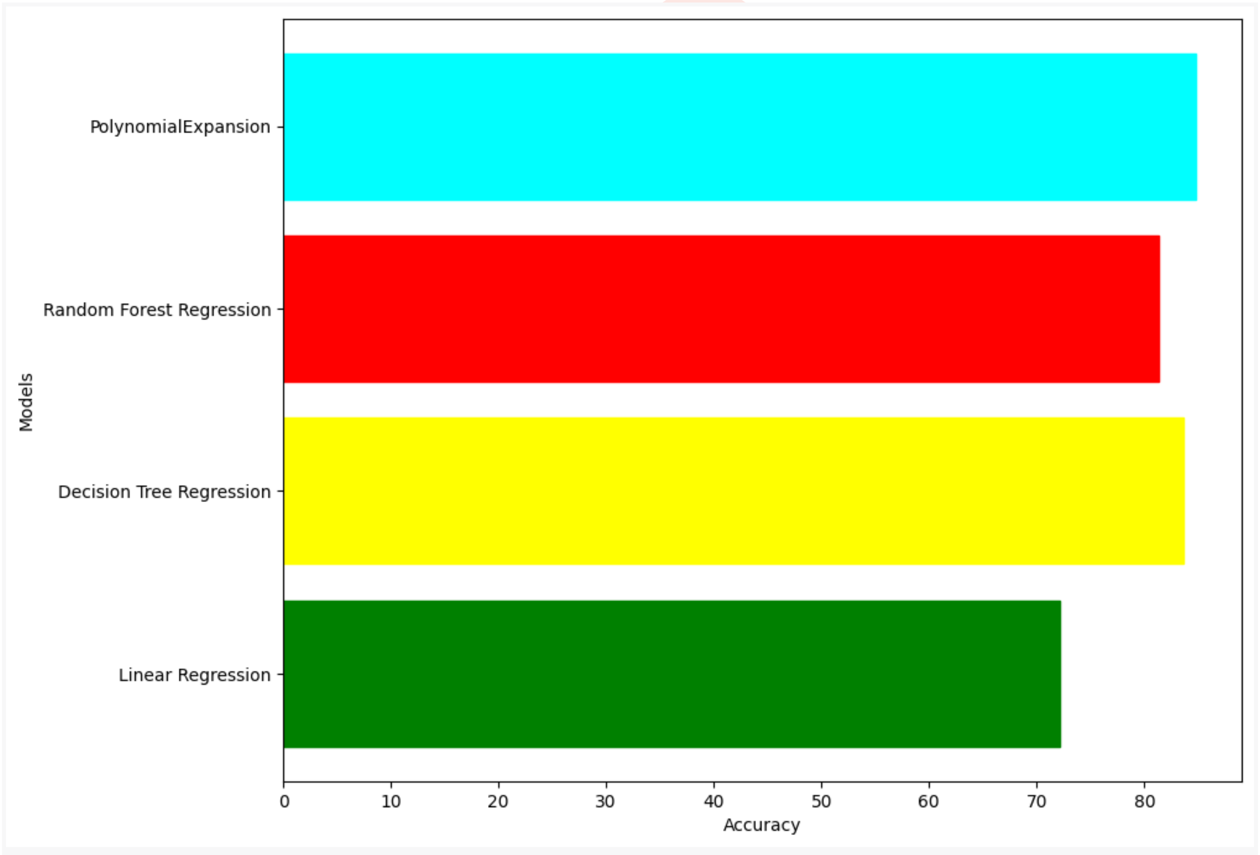
```
Root Mean Squared Error on training data: 4656.350
Root Mean Squared Error on test data: 4540.174
R-squared on training data: 0.856
R-squared on test data: 0.848
Model Accuracy Score: 84.797


The Model Accuracy is : 84.8%
```

- **Model Evaluation:** We will evaluate the performance of the trained model on the testing set using evaluation metrics such as Mean Squared Error , Root mean squared error (RMSE) and R-squared.

Based on the perfomed machine learning algorithms, it seems that the Polynomial Regression after eliminating two unimportant independent variables produces the most accurate method and will be used to predict the insurance cost for an individual.

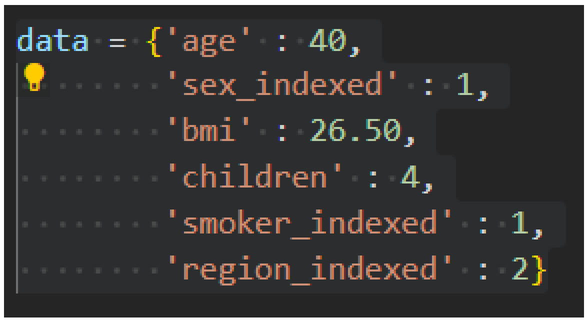


```
1. Polynomial Regression : 72.1%
2. Random Forest Regressor: 81.3%
3. Decision Tree Regressor: 83.7%
4. Polynomial Regression : 84.8%
```

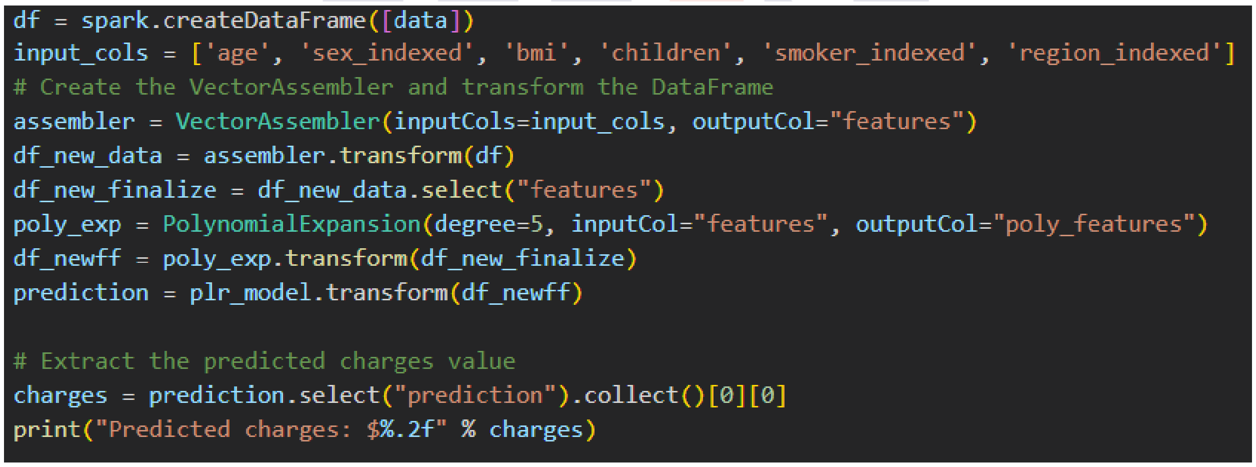


VII. Conclusion

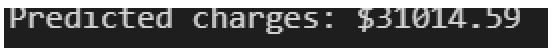
In this project, we used a linear regression model with polynomial expansion to predict medical insurance charges. We used PySpark's MLlib library to perform the data preprocessing, feature engineering, and model training. We achieved a reasonable RMSE value of 4540.874 and R-squared value of 0.848 on the testing set, which indicates that the model is able to capture the underlying relationships between the features and the target(charges) variable. This model can be used by insurance companies to estimate the charges for their customers based on their personal and medical information.



```
data = {'age' : 40,
        'sex_indexed' : 1,
        'bmi' : 26.50,
        'children' : 4,
        'smoker_indexed' : 1,
        'region_indexed' : 2}
```



```
df = spark.createDataFrame([data])
input_cols = ['age', 'sex_indexed', 'bmi', 'children', 'smoker_indexed', 'region_indexed']
# Create the VectorAssembler and transform the DataFrame
assembler = VectorAssembler(inputCols=input_cols, outputCol="features")
df_new_data = assembler.transform(df)
df_new_finalize = df_new_data.select("features")
poly_exp = PolynomialExpansion(degree=5, inputCol="features", outputCol="poly_features")
df_newff = poly_exp.transform(df_new_finalize)
prediction = plr_model.transform(df_newff)

# Extract the predicted charges value
charges = prediction.select("prediction").collect()[0][0]
print("Predicted charges: $%.2f" % charges)
```



```
Predicted charges: $31014.59
```

VIII. Future Work

Some future work that can be done to improve the performance of the model are:

- Collect more data on other factors that can influence medical insurance charges such as lifestyle habits, family history, and pre-existing medical conditions.

- Based on the above collected data we are assuming that insurance premium to be paid is of the same amount but in reality different insurance premiums are paid for different insured amounts.

- Moreover with past data , eating habbits, pre-existing medical conditions we can determine whether the person is likely to fall for any disease quickly and the amount insured to be is required high thus determining from food habits about the pricing based on certain such factors.

IX. References

- The Medical Cost Personal Datasets available on Kaggle: https://www.kaggle.com/mirichoi0218/insurance

- PySpark's MLlib library documentation: https://spark.apache.org/docs/latest/ml-guide.html

- PySpark Tutorial on Youtube : https://www.youtube.com/watch?v=_C8kWso4ne4&t=2s

- Report Refrence : https://www.ijraset.com/best-journal/medical-insurance-cost-prediction-using-machine-learning