Predictive Analysis

Eric Yarger

Question proposal

The question this research proposes to answer is:

Is it possible to predict the TotalCharge of a customer and identify key contributors using Decision Tree method based on the available data?

Being able to predict the total charge that a patient will accumulate over their stay has numerous real-world applications. One example of this for our organization is that this would allow for more accurate pro-forma financial projections to base business purchases and decisions on for our executive team.

Goal

The primary goal of this data analysis is to develop a decision tree machine learning model to help the company predict its patient's estimated total charges. This model will have utility as a general tool in our toolkit for financial analysis and predicting future revenues based off of data available in this dataset.

Prediction Method

The prediction method used for this analysis is regression using Decision Trees. A decision tree is a non-parametric supervised machine learning method, that at its core, is really a series of nested if-else statements. Decision trees for regression, also referred to as a regression tree, is what this analysis will use as its prediction method. This type of model attempts to determine the relationship between a dependent variable, in this analysis TotalCharge, and a series of independent variables from the data set (Vala, 2021). Sklearn's DecisionTreeRegressor function is used for this regression analysis. This tool inputs the split training data and minimizes mean square error for a specified number of iterations over the nodes, called depth.

Expected outcome from this analysis is that the data set is sufficient to yield a model that is accurate and statistically significant in its predictive capacity of TotalCharge, without experiencing overfitting or underfitting.

Method Assumption

Assumptions of decision tree for predictive analysis are that initially the whole training data is considered as the root, and that records are distributed recursively on the basis of the attribute value (Vishalmendekarhere, 2021). The pros of decision tree include;

- Data preparation takes less time than other machine learning algorithms
- Data does not need to be normalized

- Compared to other predictive methods, decision trees are very intuitive and can be explained as if-else conditions to key decision makers.

The cons of decision tree include:

- Takes a lot of time to train models on large data sets
- Comparatively expensive to train.

Packages and libraries used

Packages and Libraries Used in this analysis and for data preparation	**Justification for how Package or Library supports the analysis**
matplotlib.pyplot	Provides an implicit way of plotting data for visualizations and analysis.
pandas	Feature rich data analysis and manipulation tool for Python. Used throughout the analysis.
numpy	Library for Python that adds support for large, multidimensional arrays.
seaborn	Visualization and graph creation tool for Python.
missingno	Library that provides graphs for visualizing missing data in a data set.
scipy	Library for scientific and technical computing in Python.
Scipy.stats import zscore	Used for calculating zscores.
platform	For printing python version
Packages and Libraries used for Decision Tree Regression	**Justification for how Package or Library supports the analysis**
Sklearn import tree	Library that contains DecisionTreeRegressor, used for regression tree analysis for our model prediction.
Sklearn.model_selection import cross_val_score, train_test_split	Cross_val_score evaluates score by cross validation. Train_test_split splits the data into respective training and testing data sets.
Sklearn import metrics import mean_square_error	This module includes score functions and functions to calculate the mean_square_error of our predictive analysis.
.score()	Used for calculating the regression model's accuracy.

Preprocessing Data

The main goal for data preprocessing that is relevant to our research question is to ensure that the data set chosen is optimally prepared for decision tree regression analysis. This ensures that the results from the model are as accurate as possible. This goal includes:

- Removal or imputation of missing data and outliers.
- Creation of dummy variables for any categorical variables.
- Scaling the chosen data set. Features need to be the same scale to output the most accurate results.

By ensuring our data preparation meets these parameters we will have an optimal starting point for answering the question posed in section A1.

Variables chosen for analysis

The table below identifies the data set variables that were used to perform the predictive analysis from part A1.

TARGET VARIABLE	**CATEGORICAL OR CONTINUOUS**	**DATA TYPE**
TotalCharge	Continuous	float64
PREDICTOR VARIABLES	**CATEGORICAL OR CONTINUOUS**	**DATA TYPE**
Initial_days	Continuous	float64
ReAdmis_0	Categorical	int8
ReAdmis_1	Categorical	int8
Initial_admin_Elective_Admission	Categorical	int8
Initial_admin_Emergency_Admission	Categorical	int8
Initial_admin_Observation_Admission	Categorical	int8
Complication_risk_High	Categorical	int8
Complication_risk_Medium	Categorical	int8

Steps for analysis

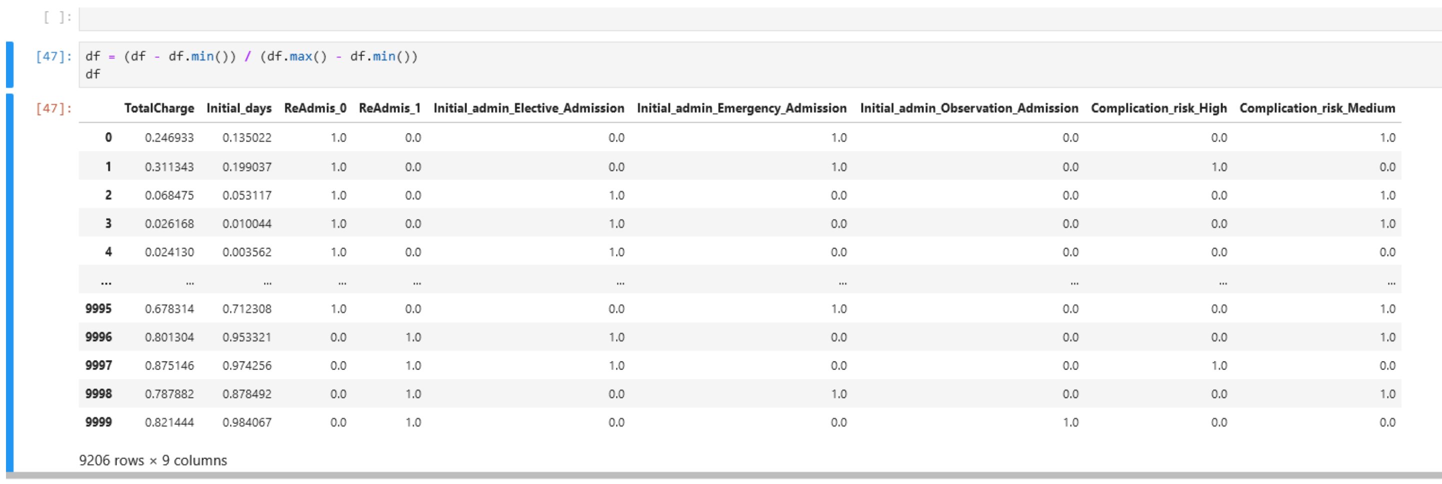
Listed and explained below are the steps to prepare the data for analysis.

Step	Action and explanation
1	Load the data and initial visualization to see shape/size/type of data features.
2	Rename survey features to provide a more usable / readable data set. Plot Pairplots X-Axis is TotalCharge, Y-axis is independent variables
3	Address missing data, duplicates, and outliers. This process makes the data more applicable for regression using Decision Trees.
4	Look at the correlation between variables using correlation table and heatmaps. This provides visual and statistically useful metrics of how each variable correlates with TotalCharge.
5	Create dummy variables - makes categorical features applicable for regression analysis. Rename any applicable variables.
6	Drop demographic features that won't be used in the analysis. These features have little correlation, many different values, and are not statistically useful for the purpose of the analysis.
7	Select features for regression statistically. Keep all variables with correlation > .04 with TotalCharge.
8	Min/Max Scale features. Scaling is necessary to give the same value limits to the features, float from 0 to 1. This makes the features more useful Decision Tree regression analysis.
9	Graphs and Summary Statistics for visualizing data stats and shapes.
10	.astype() = int8 for categorical variables. Convert categorical variables into minimal datatype for calculations.
11	Assign prepared features to y = TotalCharge, X = Prepared independent features. Prepares the data for analysis.
14	Import necessary libraries for regression analysis for prediction. This prepares the environment to have the necessary tools. Split data into Training (X_train, y_train) and Testing (X_test, y_test). This allows us to have (in the case of this analysis) 70% of the data for training and 30% for testing.

Code input and output for each step in preprocessing the data set is labeled and identified in the supplemental copy of my Jupyter Notebook used for the analysis under the section 'Data Preparation'.

Intermediate calculations and output

The analysis technique used was DecisionTreeRegressor() for regression to predict TotalCharge. The output from this method is continuous. The screenshot below details the features used for the analysis. TotalCharge is the Target feature, all other features are predictive.



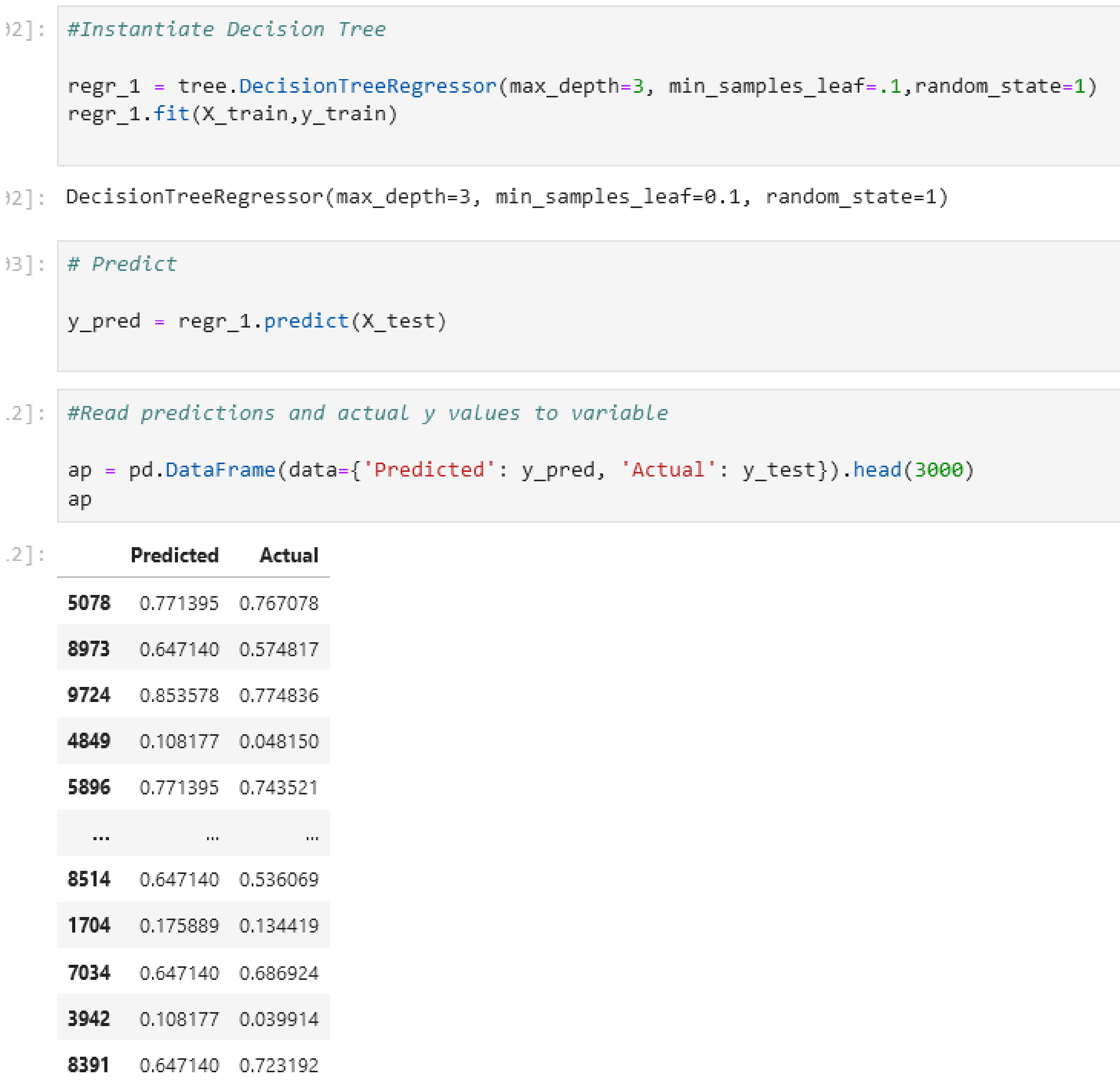
```
[ ]:

[47]: df = (df - df.min()) / (df.max() - df.min())
      df
```

[47]:

	TotalCharge	Initial_days	ReAdmis_0	ReAdmis_1	Initial_admin_Elective_Admission	Initial_admin_Emergency_Admission	Initial_admin_Observation_Admission	Complication_risk_High	Complication_risk_Medium
0	0.246933	0.135022	1.0	0.0	0.0	1.0	0.0	0.0	1.0
1	0.311343	0.199037	1.0	0.0	0.0	1.0	0.0	1.0	0.0
2	0.068475	0.053117	1.0	0.0	1.0	0.0	0.0	0.0	1.0
3	0.026168	0.010044	1.0	0.0	1.0	0.0	0.0	0.0	1.0
4	0.024130	0.003562	1.0	0.0	1.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...
9995	0.678314	0.712308	1.0	0.0	0.0	1.0	0.0	0.0	1.0
9996	0.801304	0.953321	0.0	1.0	1.0	0.0	0.0	0.0	1.0
9997	0.875146	0.974256	0.0	1.0	1.0	0.0	0.0	1.0	0.0
9998	0.787882	0.878492	0.0	1.0	0.0	1.0	0.0	0.0	1.0
9999	0.821444	0.984067	0.0	1.0	0.0	0.0	1.0	0.0	0.0

9206 rows × 9 columns

The data set is split into training (X_train, y_train) and testing (X_test, y_test) sets provided in section D1 using train_test_split(), with 70% of the data in the training set and 30% in the test set. Next the Decision Tree is instantiated and fitted to the training data. The DecisionTreeRegressor is set to a max_depth of 3, with min_samples_leaf set to .1. The Decision Tree variable regr_1, is then used to predict on the test data, X_test, assigned to variable 'y_pred'. The predicted results and the actual y_test data are assigned to a pandas dataframe, variable 'ap'. Below is a screenshot of the code used for these intermediate calculations, with the printed table 'ap' showing predicted and actual data.



)2]:

```
#Instantiate Decision Tree

regr_1 = tree.DecisionTreeRegressor(max_depth=3, min_samples_leaf=.1,random_state=1)
regr_1.fit(X_train,y_train)
```

)2]:

```
DecisionTreeRegressor(max_depth=3, min_samples_leaf=0.1, random_state=1)
```

)3]:

```
# Predict

y_pred = regr_1.predict(X_test)
```

.2]:

```
#Read predictions and actual y values to variable

ap = pd.DataFrame(data={'Predicted': y_pred, 'Actual': y_test}).head(3000)
ap
```

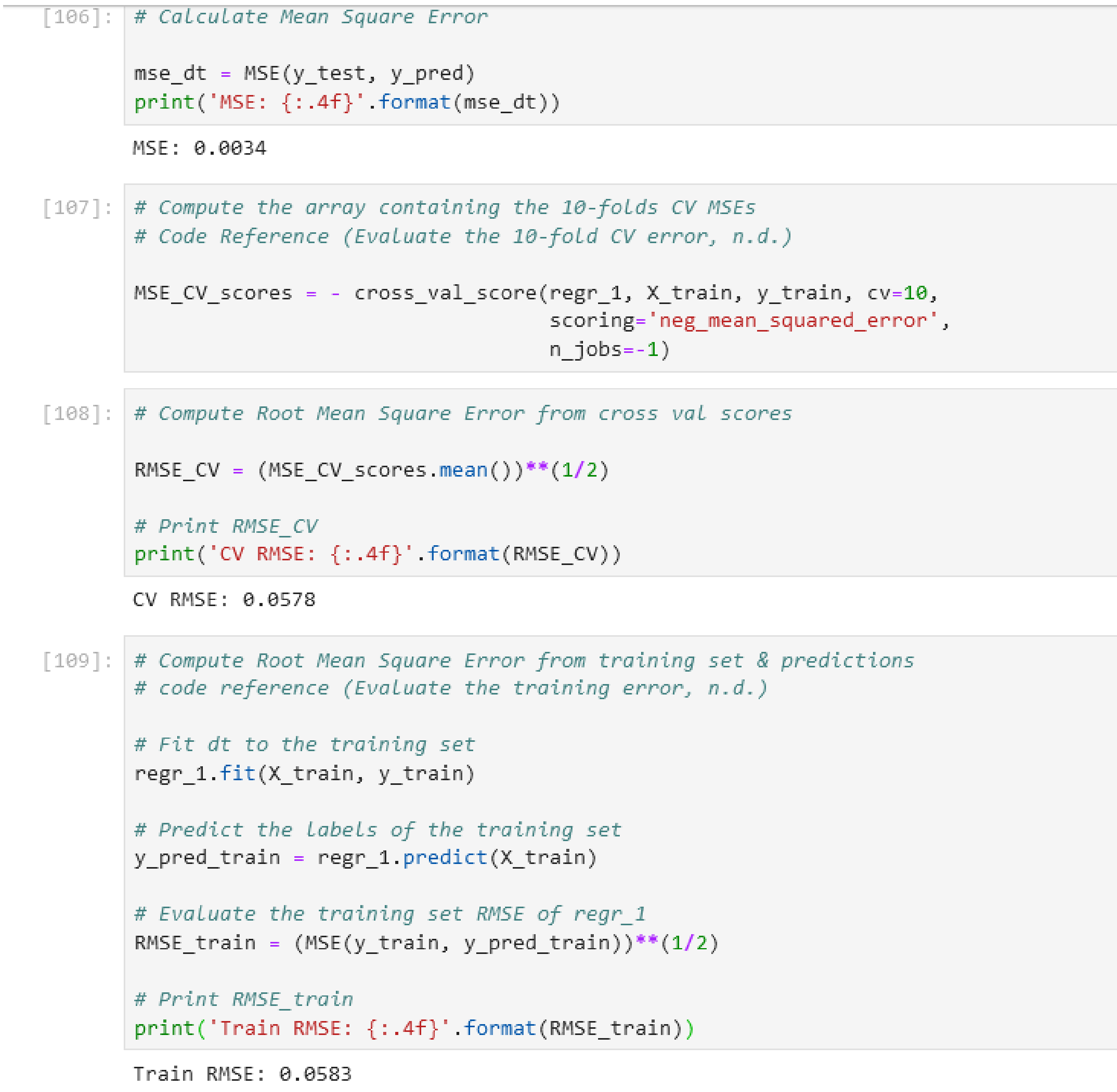
.2]:

	Predicted	Actual
5078	0.771395	0.767078
8973	0.647140	0.574817
9724	0.853578	0.774836
4849	0.108177	0.048150
5896	0.771395	0.743521
...	...	...
8514	0.647140	0.536069
1704	0.175889	0.134419
7034	0.647140	0.686924
3942	0.108177	0.039914
8391	0.647140	0.723192

Next, the Mean Square Error (MSE) is calculated using sklearn's mean_squared_error method (read in as MSE). The calculated MSE is .0034. The data is Cross Validated using sklearn's cross_val_score. The regression model from above, and the X_train, y_train data are set in the parameters with cross validation cv = 10. The cross validation output, assigned to variable 'MSE_CV_scores', is then used to calculate Root Mean Square Error (RMSE), which is .0578

Next, The Training RMSE is calculated using the regression model fitted to the training data. The training RMSE is .0583.

The coded inputs and generated outputs for the above sequences are shown in the screenshot below.

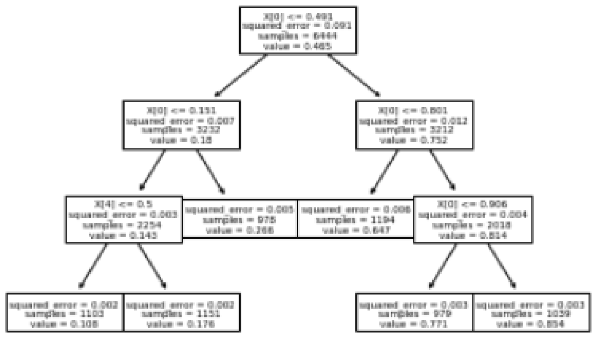


```
[106]: # Calculate Mean Square Error

       mse_dt = MSE(y_test, y_pred)
       print('MSE: {:.4f}'.format(mse_dt))

       MSE: 0.0034

[107]: # Compute the array containing the 10-folds CV MSEs
       # Code Reference (Evaluate the 10-fold CV error, n.d.)

       MSE_CV_scores = - cross_val_score(regr_1, X_train, y_train, cv=10,
                                         scoring='neg_mean_squared_error',
                                         n_jobs=-1)

[108]: # Compute Root Mean Square Error from cross val scores

       RMSE_CV = (MSE_CV_scores.mean())**(1/2)

       # Print RMSE_CV
       print('CV RMSE: {:.4f}'.format(RMSE_CV))

       CV RMSE: 0.0578

[109]: # Compute Root Mean Square Error from training set & predictions
       # code reference (Evaluate the training error, n.d.)

       # Fit dt to the training set
       regr_1.fit(X_train, y_train)

       # Predict the labels of the training set
       y_pred_train = regr_1.predict(X_train)

       # Evaluate the training set RMSE of regr_1
       RMSE_train = (MSE(y_train, y_pred_train))**(1/2)

       # Print RMSE_train
       print('Train RMSE: {:.4f}'.format(RMSE_train))

       Train RMSE: 0.0583
```

Next the Decision Tree is plotted, shown below.

```
[115]: # Graph decision tree
       # Code reference (Galarnyk, 2022)
       tree.plot_tree(regr_1)
```

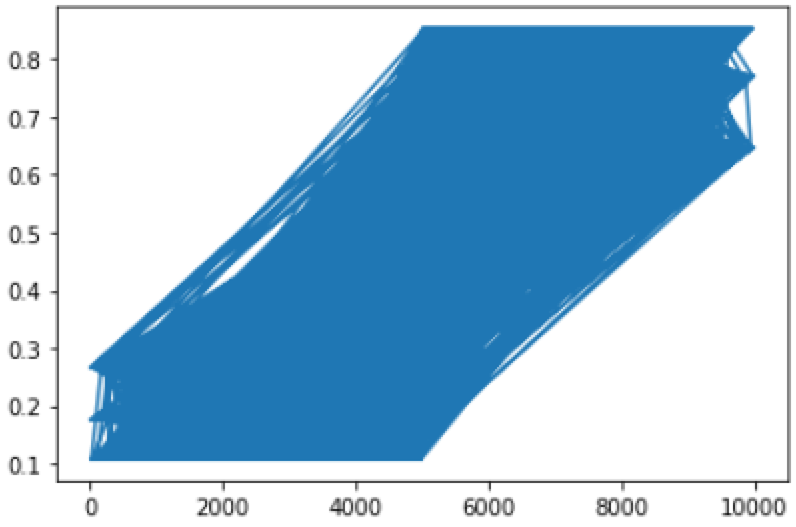
```
[115]: [Text(0.5, 0.875, 'X[0] <= 0.491\nsquared_error = 0.091\nsamples = 6444\nvalue = 0.465'),
        Text(0.3, 0.625, 'X[0] <= 0.151\nsquared_error = 0.007\nsamples = 3232\nvalue = 0.18'),
        Text(0.2, 0.375, 'X[4] <= 0.5\nsquared_error = 0.003\nsamples = 2254\nvalue = 0.143'),
        Text(0.1, 0.125, 'squared_error = 0.002\nsamples = 1103\nvalue = 0.108'),
        Text(0.3, 0.125, 'squared_error = 0.002\nsamples = 1151\nvalue = 0.176'),
        Text(0.4, 0.375, 'squared_error = 0.005\nsamples = 978\nvalue = 0.266'),
        Text(0.7, 0.625, 'X[0] <= 0.801\nsquared_error = 0.012\nsamples = 3212\nvalue = 0.752'),
        Text(0.6, 0.375, 'squared_error = 0.006\nsamples = 1194\nvalue = 0.647'),
        Text(0.8, 0.375, 'X[0] <= 0.906\nsquared_error = 0.004\nsamples = 2018\nvalue = 0.814'),
        Text(0.7, 0.125, 'squared_error = 0.003\nsamples = 979\nvalue = 0.771'),
        Text(0.9, 0.125, 'squared_error = 0.003\nsamples = 1039\nvalue = 0.854')]
```



The images below plot the predicted and actual values from the Decision Tree Regression analysis.

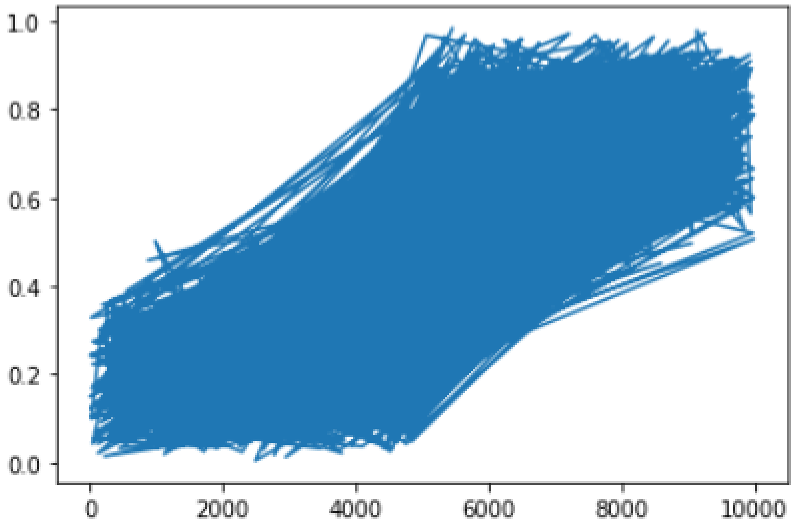
```
[119]: plt.plot(ap['Predicted'])
```

```
[119]: [<matplotlib.lines.Line2D at 0x1cb1ef1a5c8>]
```



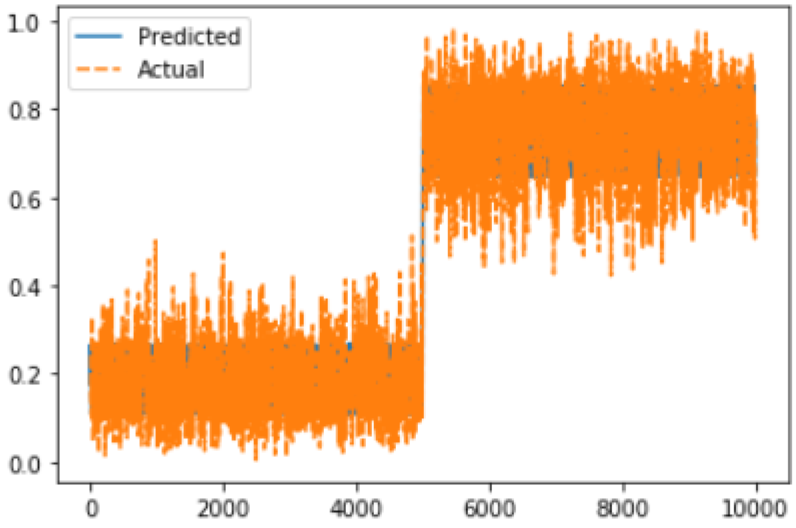
```
[132]: plt.plot(ap['Actual'])
```

```
[132]: [<matplotlib.lines.Line2D at 0x1cb1fc115c8>]
```



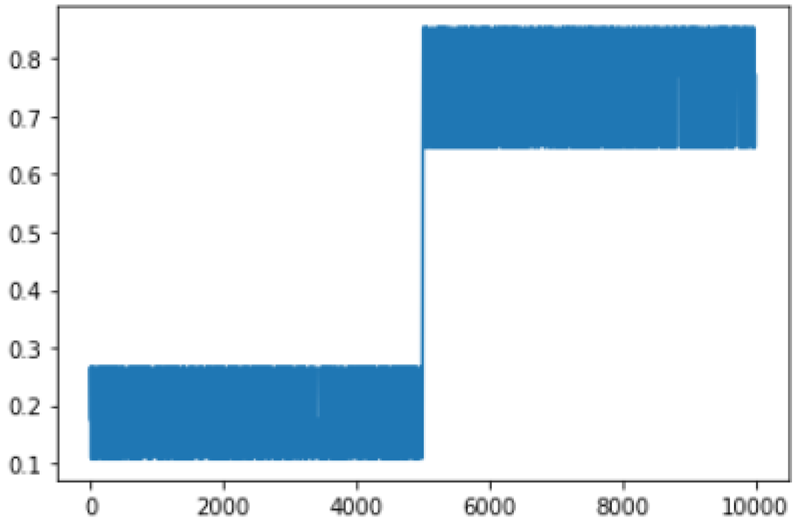
```
[125]: sns.lineplot(data=ap)
```

```
[125]: <matplotlib.axes._subplots.AxesSubplot at 0x1cb1efcd408>
```

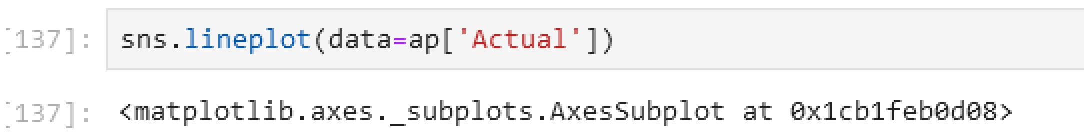


```
[136]: sns.lineplot(data=ap['Predicted'])
```

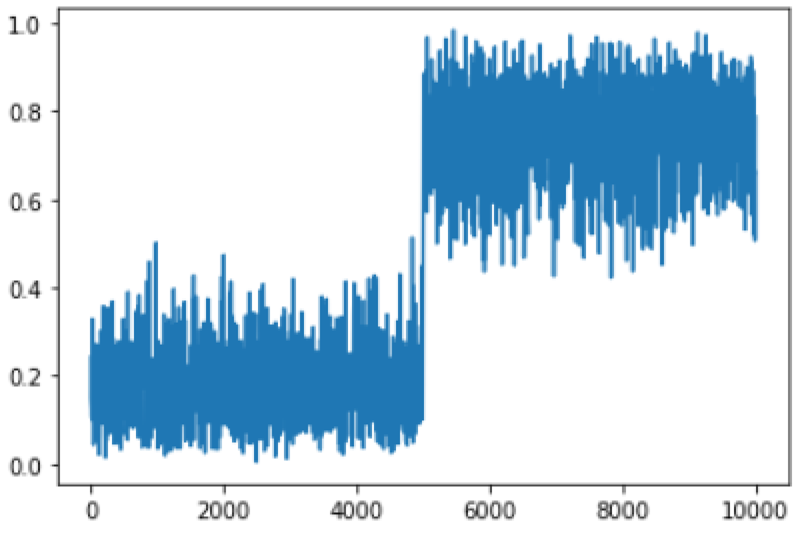
```
[136]: <matplotlib.axes._subplots.AxesSubplot at 0x1cb1fe31788>
```

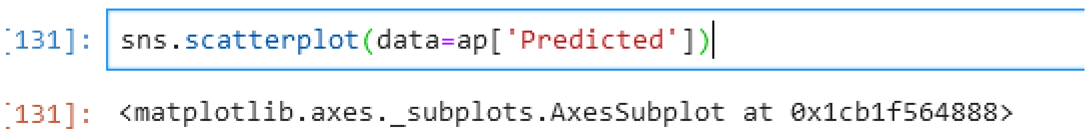


```
[137]: sns.lineplot(data=ap['Actual'])
```

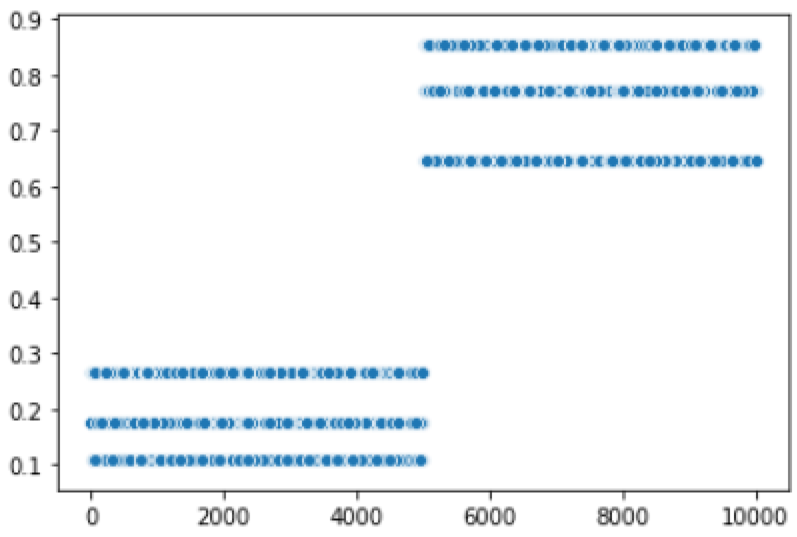


```
[137]: sns.lineplot(data=ap['Actual'])

[137]: <matplotlib.axes._subplots.AxesSubplot at 0x1cb1feb0d08>
```



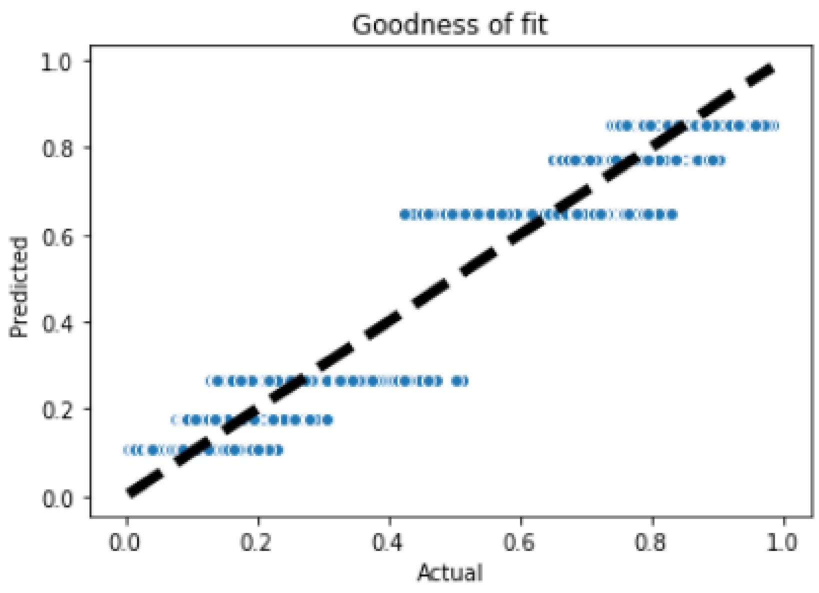


```
[131]: sns.scatterplot(data=ap['Predicted'])

[131]: <matplotlib.axes._subplots.AxesSubplot at 0x1cb1f564888>
```



The screenshot below provides the code used for the goodness of fit graph, showing the predicted vs. actual fit.

```
[135]: # Goodness of fit test, predicted vs actual

       fig, ax = plt.subplots()

       ax.scatter(y_test, y_pred, edgecolors=(1, 1, 1))
       ax.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=5)

       ax.set_xlabel('Actual')
       ax.set_ylabel('Predicted')
       ax.set_title("Goodness of fit")
       plt.show()
```



E1: Accuracy and MSE

Mean squared error (MSE) is the measurement of the amount of error in the model. It works by assessing the average squared difference between the actual values and the predicted values from our model. A lower MSE means that the model has less error, and likewise as MSE increases the model has more error. In regression the MSE represents the average squared residual (Frost, 2021). The formula for Mean Squared Error is:

$$MSE = \frac{\sum(y_i - \hat{y}_i)^2}{n}$$

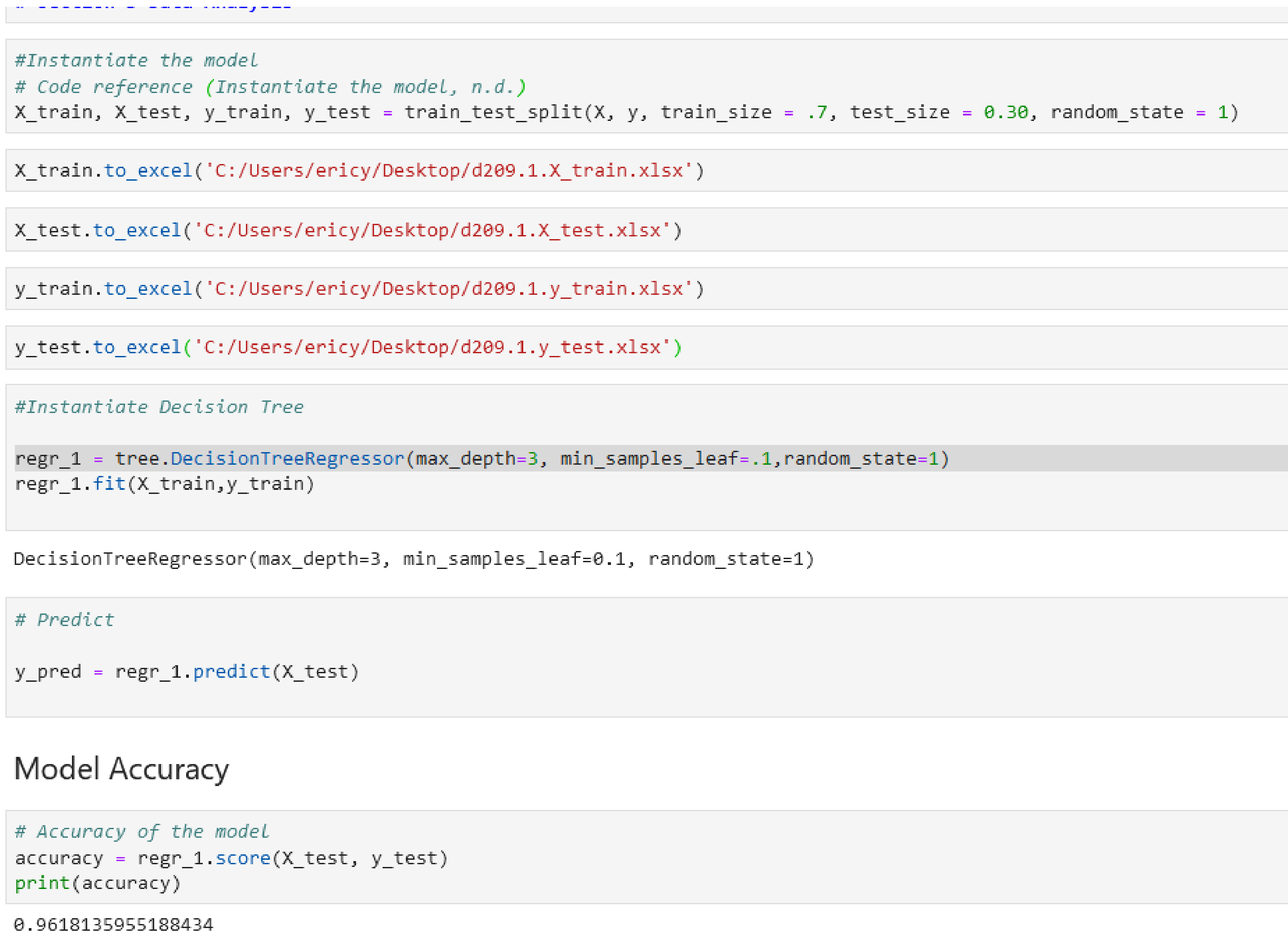
Where :

- y(sub) i is the ith observed value.

- y hat(sub) i is the corresponding predicted value from the model.
- N = the number of observations

The calculated Mean Squared Error for the model is .0034. This can be explained as the average of the square of the difference between the actual and estimated residual. This metric is used to assess how well our model fits the actual data for model evaluation. With a MSE of .0034 that means that at each point on the data set the predicted values are, on average, very close to the actual value. The calculated MSE points to a high level of model relevance, and good accuracy in the models capacity in being able to predict TotalCharge.

The accuracy of the model was calculated with DecisionTreeRegressor().score function, which returns the coefficient of determination R-Squared of the predictive model. First, the regression model is fitted to the training datasets. The .score() function is then used with the X and Y parameters being the testing datasets. This returns the accuracy score for the model, which is .9618. The closer the accuracy score is to 1, the more accurate the model is in its predicted vs actual values for TotalCharge. Using this metric we can deduce the accuracy of the model is high. Below is a screenshot showing the code input and output used to find model accuracy.



```
#Instantiate the model
# Code reference (Instantiate the model, n.d.)
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = .7, test_size = 0.30, random_state = 1)
```

```
X_train.to_excel('C:/Users/ericy/Desktop/d209.1.X_train.xlsx')
```

```
X_test.to_excel('C:/Users/ericy/Desktop/d209.1.X_test.xlsx')
```

```
y_train.to_excel('C:/Users/ericy/Desktop/d209.1.y_train.xlsx')
```

```
y_test.to_excel('C:/Users/ericy/Desktop/d209.1.y_test.xlsx')
```

```
#Instantiate Decision Tree

regr_1 = tree.DecisionTreeRegressor(max_depth=3, min_samples_leaf=.1,random_state=1)
regr_1.fit(X_train,y_train)
```

```
DecisionTreeRegressor(max_depth=3, min_samples_leaf=0.1, random_state=1)
```

```
# Predict

y_pred = regr_1.predict(X_test)
```

Model Accuracy

```
# Accuracy of the model
accuracy = regr_1.score(X_test, y_test)
print(accuracy)
```

```
0.9618135955188434
```

The accuracy of the model can also be assessed by calculating the Root Mean Square Error (RMSE). The RMSE represents the square root of the variance of residuals (Zach, 2021). The RMSE tells us the difference between the predicted values from the model and the actual values in the dataset. The lower the RMSE, the more accurately a model fits a dataset. The formula for calculating RMSE is

RMSE = $\sqrt{}\Sigma(P_i – O_i)^2 / n$

- Where P(sub)i is the predicted value for the i(th) observation
- O(sub)i is the actual value for the i(th) observation
- N is the sample size

The RMSE for our trained model is .0583. This is the measurement of how far from the regression line data points are. The target variable, TotalCharge, as well as the whole dataset, were scaled before regression analysis from 0 to 1 for all variables. To give perspective to the RMSE value the median of TotalCharge is

.4365, the mean of TotalCharge is .4650. The RMSE of .0583 in this context indicates that there is residual standard error in the model, but that the model is practically significant in its output range of accuracy as a tool for prediction.

Implications and results

The result of the regression analysis is that the model is .9618 accurate in predicting TotalCharge using the DecisionTreeRegressor(). The MSE of .0034 and training set RMSE of .0583 indicate that the model accurately predicts TotalCharge with minimal error in the model. Cross validation is implemented using sklearn's cross_val_score() with 10-folds. The Cross validation RMSE is .0578, a .005 improvement over the training RMSE. This metric can be used to validate the training model's RMSE, showing there is very little overfitting of the model. This points to an effective and overall useful model that can be used with other data from the data set, and isn't overfitted to the training data specifically.

The implication of this Decision Trees regression analysis results is that the model can be used to predict TotalCharge from the medical_clean dataset visit with high accuracy.

Limitations of this analysis

One limitation of the data analysis lies in the data set's suitableness for regression analysis using Decision Trees. When looking at correlation and feasibility of each feature for prediction selection, Initial_days and ReAdmis appeared to be highly correlated with TotalCharge, while all other features were limited in correlation. Correlation doesn't define regression accuracy, but it should be noted that the prediction ability of the variables is not distributed evenly. When a model is so dependent on two predictive features for the majority of its prediction ability, this is a limitation in the overall utility of the model if the data gathered from patients is ever altered.

Recommended course of action

The course of action recommended for our organization, in answer to the question posed in section A1, is to use this model for effectively predicting TotalCharge for patients. The model provides a simple, accurate, and effective tool for initial prediction of total charges accrued. It should be said that this model should not be the only tool used by the organization for this action. Accurate predictions of patient charges is an important metric to track and forecast. This model should be used as only one predictive tool in a collective toolkit for managing and predicting total charges as a business performance objective.

Third-party sources or code

Bushmanov, Sergey. (2019, January 28). *How to remove Outliers in Python?.* Stackoverflow.com. https://stackoverflow.com/questions/54398554/how-to-remove-outliers-in-python

Seaborn.heatmap. (n.d.). Seaborn.pydata.org. Retrieved from

https://seaborn.pydata.org/generated/seaborn.heatmap.html

Pandas.get_dummies. (N.d.). Pandas.pydata.org. Retreived from https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.get_dummies.html

Instantiate the model. (n.d.). DataCamp.com. Retrieved from https://campus.datacamp.com/courses/machine-learning-with-tree-based-models-in-python/the-bias-variance-tradeoff?ex=5

Evaluate the 10-fold CV error. (n.d.). DataCamp.com. Retrieved from https://campus.datacamp.com/courses/machine-learning-with-tree-based-models-in-python/the-bias-variance-tradeoff?ex=6

Evaluate the training error. (n.d.). DataCamp.com. Retrieved from https://campus.datacamp.com/courses/machine-learning-with-tree-based-models-in-python/the-bias-variance-tradeoff?ex=7

Sources

Vala, K. (2021, December 6). How to get started with regression trees. Builtin.com. Retrieved from https://builtin.com/data-science/regression-tree

Vishalmendekarhere. (2021, January 22). It's all about assumptions, pros & cons. Medium. Retrieved from https://medium.com/swlh/its-all-about-assumptions-pros-cons-497783cfed2d

Frost, J. (2021, November 14). Mean squared error (MSE). Statisticsbyjim.com. Retrieved from https://statisticsbyjim.com/regression/mean-squared-error-mse/

Zach. (2021, May 10). How to interpret root mean Square error (RMSE). Statology.org. Retrieved from https://www.statology.org/how-to-interpret-rmse/