# Part1: Research Question

## A1: Proposal of Question

This analysis aims to answer the question, which features of a patient’s hospital admission best predict total charges for the admission? To answer the question a random forest regressor will be used.

## A2: Defined Goal

The primary goal of this analysis is to develop a random forest regression machine learning model that is capable of predicting the total charges incurred from a patients’ initial hospital admission based on other features from their initial admission. The resulting feature importance scores will provide the answer to the research question.

# Part II: Method Justification

## B1: Explanation of Prediction Method

With a random forest model, a user-specified number of decision trees are used, each on a sample of the data. The results are then aggregated to come to a single prediction. When a random forest model is used for regression, the final aggregation is the mean of the decision tree predictions. The main benefit of expanding from a decision tree regressor to a random forest regressor is that it reduces the chances the model will overfit and produces a more accurate prediction. For this analysis, the expected outcome is the predicted total cost for each patient’s initial admission.

## B2: Summary of Method Assumption

The primary assumption made by random forest regression models is that the training data is representative of the population (Laptev, 2013). This means that the distribution or shape of the training data is assumed to represent the distribution or shape of the population from which the sample came.

## B3: Packages or Libraries List

The following Python libraries and modules will be used to perform this analysis:

* **os**: The os module is used for operating system tasks. In this analysis, it is used to create the directory where all output is stored.
* **numpy**: numpy is a library used to perform operations on arrays. It is widely used for data management and analysis. For this analysis, it will be used to assist with number generation during hyperparameter tuning.
* **pandas**: pandas is a library used to work with data including, but not limited to, importing and exporting data as well as general data manipulation and simple statistical operations. In this analysis, it is used for all data preparation tasks.
* **SciPy**: SciPy is a statistics library used widely across the industry. In this analysis, it is used to aid in statistical outlier identification.
* **Matplotlib**: Matplotlib is a library used to create a wide range of visualizations. In this analysis, it is used to plot feature importance.
* **scikit-learn**: scikit-learn is a library used for various aspects of data science. From this library, the specific modules used include:
  + **feature\_selection**: From the feature\_selection module, SelectFwe and f\_regression are used. The **SelectFwe** class is used to select features most relevant to the target variable. The **f\_regression** function is the function SelectFwe uses instead of the default since the default is only applicable to classification models.
  + **model\_selection**: From the model\_selection module train\_test\_split and RandomizedSearchCV are used. The **train\_test\_split** function is used to prepare the training and testing data needed for the analysis. The **RandomizedSearchCV** class is used for hyperparameter tuning.
  + **preprocessing**: From the preprocessing module, the **StandardScaler** class is used to scale the numeric features of the data set.
  + **ensemble**: From the ensemble module, the **RandomForestRegressor** class is used to implement the random forest regression model used in the analysis.
  + **metrics**: From the metrics module, r2\_score and mean\_squared\_error are used. **r2\_score** is used to obtain r-squared. **mean\_squared\_error** is used to obtain the mean of the sum of squared error.

# Part III: Data Preparation

## C1: Data Preprocessing

One goal of preprocessing the data is encoding the features. Several of the features are categorical. String data is not well suited to machine learning algorithms. They can handle binary dummy variables indicating whether an observation fits each category. Because of this, each of the categorical features will be one-hot-encoded to transform them into dummy variables.

## C2: Data Set Variables

|  |  |
| --- | --- |
| **Variable** | **Class** |
| TotalCharge | continuous |
| Initial\_days | continuous |
| Initial\_admin\_Emergency\_Admission | categorical |
| Complication\_risk\_Medium | categorical |
| ReAdmis | categorical |
| BackPain | categorical |
| Arthritis | categorical |
| Initial\_admin\_Observation\_Admission | categorical |

## C3: Steps for Analysis

The following code was extracted from the file “task2.py” which is being included with the submission of this report.

Load the packages

os.makedirs(**'./output2'**, exist\_ok=**True**)

**import** os  
**import** numpy **as** np  
**import** pandas **as** pd  
**import** scipy.stats **as** stats  
**import** matplotlib.pyplot **as** plt  
**from** sklearn.feature\_selection **import** SelectFwe, f\_regression  
**from** sklearn.model\_selection **import** train\_test\_split, RandomizedSearchCV  
**from** sklearn.preprocessing **import** StandardScaler  
**from** sklearn.ensemble **import** RandomForestRegressor  
**from** sklearn.metrics **import** r2\_score, mean\_squared\_error **as** MSE

Create the output directory

medical\_clean = pd.read\_csv(**'./medical\_clean.csv'**)

Load the data

print(**f'\n{**medical\_clean.head().to\_string()**}\n'**)

View the data

print(**f'\ncontinuous data:\n{**medical\_clean.describe().T.to\_string()**}'**)  
cat\_data = medical\_clean[  
 [col **for** col **in** medical\_clean.columns  
 **if** medical\_clean[col].dtype **not in** [**'float64'**, **'int64'**]]  
]  
print(**f'\ncategorical data:\n{**cat\_data.describe().T.to\_string()**}\n'**)

Evaluate the data structures

medical\_clean.info()

Evaluate data types and check for nulls

print(**f'\nduplicates:\n{**medical\_clean.duplicated().sum()**}'**)

Check for duplicates

num\_cols = [col **for** col **in** medical\_clean.columns  
 **if** medical\_clean[col].dtype **in** [**'float64'**, **'int64'**]]  
df\_num = medical\_clean[num\_cols]  
df\_zscores = df\_num.apply(stats.zscore)  
df\_outliers = df\_zscores.apply(**lambda** x: (x > 3) | (x < -3))  
print(**f'\noutliers:\n{**df\_outliers.sum().to\_string()**}'**)

Check for outliers

corr = df\_num.corr()  
print(**f'\ncorrelation:\n{**corr.to\_string()**}'**)

Look for correlation between features

Encode categorical features

df\_encoded.columns = [col.replace(**' '**, **'\_'**) **for** col **in** df\_encoded.columns]  
print(**f'\nencoded data:\n{**df\_encoded.head().to\_string()**}'**)

excluded = [**'CaseOrder'**, **'Customer\_id'**, **'Interaction'**, **'UID'**,  
 **'TimeZone'**, **'City'**, **'State'**, **'County'**, **'Job'**]  
y\_n\_cols = [col **for** col **in** medical\_clean.columns  
 **if** medical\_clean[col].nunique() == 2  
 **and** col **not in** excluded]  
num\_cols = [col **for** col **in** medical\_clean.columns  
 **if** medical\_clean[col].dtype **in** [**'int64'**, **'float64'**]  
 **and** col **not in** y\_n\_cols  
 **and** col **not in** excluded]  
cat\_cols = [col **for** col **in** medical\_clean.columns  
 **if** col **not in** num\_cols  
 **and** col **not in** y\_n\_cols  
 **and** col **not in** excluded]  
df\_encoded = medical\_clean[[col **for** col **in** medical\_clean.columns  
 **if** col **not in** excluded]]  
df\_encoded = pd.get\_dummies(df\_encoded, columns=cat\_cols, drop\_first=**True**)  
df\_encoded.replace({**'Yes'**: 1, **'No'**: 0}, inplace=**True**)

Fix column names

df\_scaled = df\_encoded.copy()  
scaler = StandardScaler()  
df\_scaled.loc[:, num\_cols] = \  
 scaler.fit\_transform(df\_scaled.loc[:, num\_cols])  
print(**f'\nscaled data:\n{**df\_scaled.head().to\_string()**}'**)

Scale numeric features

selector = SelectFwe(f\_regression, alpha=0.05)  
X = df\_scaled[[col **for** col **in** df\_scaled.columns **if** col != **'TotalCharge'**]]  
y = df\_scaled[**'TotalCharge'**]  
X = pd.DataFrame(selector.fit\_transform(X, y),  
 columns=selector.get\_feature\_names\_out(selector.feature\_names\_in\_))  
feature\_data = pd.DataFrame({**'feature'**: selector.feature\_names\_in\_,  
 **'score'**: selector.scores\_,  
 **'pvalue'**: selector.pvalues\_})  
top\_features = feature\_data.iloc[selector.get\_support(indices=**True**)].\  
 sort\_values(by=**'score'**, ascending=**False**)  
print(**f'\nfeature scores:\n{**top\_features.to\_string()**}'**)

Select features

df = pd.concat([y, X], axis=1)  
print(**f'\nfinal data:\n{**df.head().to\_string()**}'**)  
df.to\_csv(**'./output2/cleaned\_data\_set.csv'**)

Save cleaned data set

## C4: Cleaned Data Set

The cleaned data is being submitted with this report as “cleaned\_data\_set.csv”.

# Part IV: Analysis

## D1: Splitting the Data

The data was split so that 30% was reserved for model evaluation and 70% was used for model training. The resulting data sets are being submitted with this report as “X\_train.csv”, “X\_test.csv”, “y\_train.csv”, and “y\_test.csv”.

## D2: Output and Intermediate Calculations

The random forest regression analysis was implemented using scikit-learn’s RandomForestRegressor class. To tune the hyperparameters, RandomizedSearchCV was used with 10-fold cross validation. The model was repeatedly fit on the training data, randomly varying the number of trees in the forest, the maximum depth of each tree, the minimum samples required in each leaf node, and the number of samples drawn to train each tree to obtain the best mean squared error.

Text

Description automatically generated

Once the model with the best score was obtained, predictions were made using the test data. Then, the feature importance scores were extracted and plotted.

Graphical user interface, text, application

Description automatically generated with medium confidence

Chart, bar chart

Description automatically generated

Finally, the r-squared and mean squared error were calculated on the true and predicted y values using scikit-learn’s r2\_score and mean\_squared\_error functions respectively. Root mean squared error was calculated by taking the square root of the mean squared error.

Text

Description automatically generated

## D3: Code Execution

The following code was extracted from the file “task2.py” which is being included with the submission of this report.

*# Split into train and test data*X\_train, X\_test, y\_train, y\_test = \  
 train\_test\_split(X, y, test\_size=0.3, random\_state=42)  
X\_train.to\_csv(**'./output2/X\_train.csv'**)  
X\_test.to\_csv(**'./output2/X\_test.csv'**)  
y\_train.to\_csv(**'./output2/y\_train.csv'**)  
y\_test.to\_csv(**'./output2/y\_test.csv'**)  
  
*# Hyperparameter tuning*model = RandomForestRegressor(n\_jobs=-1, random\_state=42)  
param\_grid = {**'n\_estimators'**: np.arange(10, 110, 10),  
 **'max\_depth'**: np.arange(1, 12, 2),  
 **'min\_samples\_leaf'**: np.arange(0.001, 0.01, 0.005),  
 **'max\_samples'**: np.arange(0.1, 1, 0.2)}  
search = RandomizedSearchCV(model, param\_grid, cv=10, n\_jobs=-1,   
 random\_state=42)

search.fit(X\_train, y\_train)  
print(**f'\nhyperparameter tuning:'  
 f'\n\tbest parameters:'**)  
**for** k, v **in** search.best\_params\_.items():  
 print(**f'\t\t{**k**}: {**v**}'**)  
print(**f'\tbest score: {**search.best\_score\_**}'**)  
  
*# Make predictions*y\_pred = search.predict(X\_test)  
  
*# Metrics*features = pd.DataFrame({

**'feature'**: search.feature\_names\_in\_,

**'importance'**: search.best\_estimator\_.feature\_importances\_

})

features\_sorted = features.sort\_values(by=**'importance'**, ascending=**False**)  
print(**f'\nfeature importance:\n{**features\_sorted.to\_string()**}'**)  
  
fig, ax = plt.subplots()  
features\_sorted.sort\_values(by=**'importance'**, inplace=**True**)  
features\_sorted.plot(x=**'feature'**, y=**'importance'**, kind=**'barh'**, ax=ax,  
 logx=**True**, rot=20, legend=**False**)  
ax.set\_xlabel(**'importance (log scale)'**)  
fig.suptitle(**'Feature Importance'**)  
fig.set\_tight\_layout(**True**)  
plt.savefig(**'./output2/feature\_importance.png'**)  
plt.show()  
  
r\_squared = r2\_score(y\_test, y\_pred)  
mse = MSE(y\_test, y\_pred)  
rmse = mse \*\* (1/2)  
print(**f'\nr-squared: {**r\_squared**}'  
 f'\nmean squared error: {**mse**}'  
 f'\nroot mean squared error: {**rmse**}'**)

# Part V: Data Summary and Implications

## E1: Accuracy and MSE

To assess model accuracy, r-squared and the root mean squared error were calculated. R-squared was about 0.99, meaning the model explains 99% of the variance in the dependent variable, TotalCharge. Root mean squared error was about 0.09, meaning the predictions differed from the trues value by 0.09 units on average.

The mean squared error (MSE) was about 0.01. This means on average, the sum of the difference between the true and predicted values squared was about 0.01.

## E2: Results and Implications

This analysis used a random forest regression machine learning model to predict the total charges for inpatient hospital admissions given other features of the admissions. The final model had an r-squared value of 99% and a root mean squared error value of 0.09. These metrics are excellent and indicate that the model is expected to perform reliably in a real-world setting. The features used in the model were selected using SelectFwe from scikit-learn, which evaluates the p-values of the family-wise error rate and selects only the features whose p-value falls below the specified value (Vieira, 2018), in this case 0.05. The hyperparameters of the model were tuned using scikit-learn’s RandomizedSearchCV with 10-fold cross validation. The best identified model consisted of 100 regression trees. Each of the trees had a maximum depth of 9 nodes, leaves no smaller than 0.1% of the data, and used no more than 70% of the features. With additional data and more computing resources, the quality of the model could still be strengthened. Because this analysis used a randomized rather than a complete search for hyperparameter tuning, it is possible that a better model still exists.

The feature importance values were also evaluated to answer the proposed research question. In order of descending importance, the features were Initial\_days – length of stay, Initial\_admin\_Emergency\_Admission – whether they were admitted from the Emergency Department (1) or for an elective procedure (0), Complication\_risk\_Medium – whether the patients risk of complication was medium (1) or not (0), ReAdmis – whether the patient was eventually readmitted (1) or not (0), BackPain – whether the patient had back pain documented in their chart (1) or not (0), Arthritis – whether the patient had arthritis documented in their chart (1) or not (0), and Initial\_admin\_Observation\_Admission – whether the patient was admitted for observation (1) or for an elective procedure (0). Initial\_days accounted for 98% of the model, which makes sense since charges are incurred as the length of stay increases. The remaining features account for less than 2% of the model combined, with being admitted from the Emergency Department taking up over half of it. See section D2 for the feature importance bar chart. Note that the importance values are displayed on a log scale. Without scaling the axis, the last feature was not visible on the chart.

## E3: Limitation

The largest limitation of the data analysis is that random forests can not extrapolate beyond the range of data it has seen. If the sample used to create the model does not represent every facet of its population, a phenomenon known as covariate shift can occur. This happens when the independent variables’ ranges in the sample differ from the population, forcing it to make decisions about things it has not yet learned (Thompson, 2019). This tends to lead to a poor prediction in these situations.

## E4: Course of Action

Based on the findings of this analysis, there are two courses of action which should be followed. First, the model should be implemented in a pilot capacity so that hospital administrators can start to evaluate how the model performs on real world data. This will allow them to determine whether the covariate shift occurred due to the selected sample. If not, it should further be implemented in a full capacity where it can be used to for financial forcasting. The second course of action is a more in-depth analysis using additional observations and a complete search of an expanded hyperparameter grid. This type of analysis would likely require a dedicated server to allow the process to continue running for an extended period of time.

# Part VI: Demonstration

## F: Panopto Recording

The Panopto video recording is being included with the submission of this report as “task2\_recording”.

## G: Sources for Third-Party Code

Only library documentation was used to support the coding of this analysis. See section B3 for the list of libraries and modules used.

## H: Sources

Laptev, D. (2013, May 16). *Random forest assumptions*. Retrieved from Cross Validated: https://stats.stackexchange.com/q/59182

Thompson, B. (2019, December 17). *A limitation of Random Forest Regression*. Retrieved from Towards Data Science: https://towardsdatascience.com/a-limitation-of-random-forest-regression-db8ed7419e9f

Vieira, B. H. (2018, February 13). *fpr, fdr and fwe for feature selection*. Retrieved from Cross Validated: https://stats.stackexchange.com/q/328366