IEOR 142A Final Project Report
Group Members: Julia Chen, Junghyun Cheon, Michelle Guan, Audrey Ko, Jacqueline Yang

Motivation

In America, medical insurance is a necessity in order for reduced in-network care, preventative care, and protection from unexpected medical costs. However, insurance fees (i.e. premiums, deductibles, out of pocket costs, etc.) can be a significant financial burden for patients and also present a challenge for insurance companies in budgeting and risk management. Given this, there is a critical need for methods to help individuals and companies make more informed decisions regarding insurance. For individual patients, being able to predict insurance costs based on demographics enables them to select the most suitable plans, and companies can likewise optimize pricing strategies and improve on risk management. To address this we developed a predictive model for insurance charges based off of patient demographic data. Our model seeks to empower patients to make well-informed insurance choices while helping stakeholders in the healthcare industry optimize their business decisions.

Data

The data we are using contains demographic data (sex, age, BMI, smoker, & region) about 1339 individuals alongside their resulting insurance premiums. Our data is from GitHub and it was sourced using both online and offline datasets of patient information. This data provides a rich foundation for predicting insurance costs because it captures key variables that are closely tied to healthcare expenses. For example, BMI and smoking status are strong indicators of an individual's health risks, which can significantly influence insurance premiums. Likewise, regions can also contribute as the relative cost of living is different in each region, which may affect healthcare costs across different areas. This data is suitable for our use as it includes variables that are directly relevant to the factors insurance companies consider when determining premiums. Moreover, the inclusion of diverse demographic and health-related features ensures the predictions are grounded in real-world factors that impact insurance pricing.

Analytics Models

Preprocessing

The dataset `insurance.csv` comprises demographic and lifestyle information about individuals and their insurance charges. This initial understanding underscores the significance of preprocessing to transform raw data into a learning-ready format conducive to model training. Next, to ensure models treat each feature equally and avoid bias, we employed standardization using the `StandardScaler` from `scikit-learn`. Next, we applied dummy encoding via the `get_dummies()` function from `pandas`. This technique transforms each categorical variable into a set of binary features. All but one category within a variable results in a distinct binary column indicating its presence. Our model's goal is to predict the `charges` based on dependent variables or features. We therefore segregated these two parts of the dataset into a feature matrix `X` (containing all columns except `charges`) and target vector `y` (composed solely of the `charges` column). This isolation keeps model generalizations distinct from actual outcomes. Lastly, the `train_test_split` function offers a randomized approach to ensuring that 80% of the data trains the model (`X_train`, `y_train`) while the remaining 20% serves to evaluate out-of-sample performance (`X_test`, `y_test`). Choosing a fixed `random_state=42` ensures reproducibility of results across platforms and reach consistent evaluations, essential for comparing the fidelity among various models and hyperparameter configurations. These preprocessing steps prepare an optimized landscape for various models, maintaining data integrity while addressing variance, representation, and sampling concerns systematically.

Baseline Model

We began by creating a baseline model to allow for comparisons to our other models. Because our target feature is continuous, we used mean as the prediction (vs. predicting the majority class). We decided to measure the performance of the baseline model using OSR^2, RMSE, and MAE. Naturally, because no feature engineering or selection occurred, the baseline model did not perform well as indicated by the performance metrics: $R^2 = 0.0$, MSE = 1.06, and MAE = 0.79. Thus, we looked to other models for better performance and prediction.

Linear Regression

Following the baseline model, we attempted using a linear regression model using 'statsmodels.api.ols'. While this model performed significantly better than our baseline model, with $OSR^2 = 0.78$, MSE = 0.23, and MAE = 0.34, we felt it didn't adequately capture the nuances of our data. The first OLS Regression summary below also showed that 'age', 'bmi', and 'children', were statistically significant variables (using alpha level of 0.05) that should be included in models; we trained a new OLS model with these variables. In the new model, we found that the $OSR^2 = 0.16$, MSE = 0.90, and MAE = 0.76. Although age, BMI, and children have the lowest p-values and are statistically significant predictors, they do not account for the majority of the variance in insurance charges. Key features like smoker and region, which are strong drivers of charges, were excluded, reducing the model's ability to capture critical patterns and relationships in the data. Smokers typically have much higher premiums, and regional differences affect premiums due to variations in healthcare costs. Additionally, the lower R^2 value reflects that the relationship between the independent variables and the target feature is not strongly linear, further limiting the model's explanatory power.

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                charges   R-squared:                       0.742
Model:                            OLS   Adj. R-squared:                  0.739
Method:                 Least Squares   F-statistic:                     338.1
Date:                Sat, 14 Dec 2024   Prob (F-statistic):          3.02e-304
Time:                        03:58:44   Log-Likelihood:                -786.02
No. Observations:                1070   AIC:                             1592.
Df Residuals:                    1060   BIC:                             1642.
Df Model:                           9
Covariance Type:            nonrobust
====================================================================================
                       coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------
const            -2.902e+11   8.09e+11     -0.359      0.720   -1.88e+12     1.3e+12
age                  0.2978      0.016     19.007      0.000       0.267       0.329
bmi                  0.1701      0.016     10.378      0.000       0.138       0.202
children             0.0422      0.015      2.740      0.006       0.012       0.072
sex_female       -4.259e+11   1.19e+12     -0.359      0.720   -2.76e+12     1.9e+12
sex_male         -4.259e+11   1.19e+12     -0.359      0.720   -2.76e+12     1.9e+12
smoker_no         7.385e+11   2.06e+12      0.359      0.720    -3.3e+12    4.78e+12
smoker_yes        7.385e+11   2.06e+12      0.359      0.720    -3.3e+12    4.78e+12
region_northeast -2.238e+10   6.24e+10     -0.359      0.720   -1.45e+11       1e+11
region_northwest -2.238e+10   6.24e+10     -0.359      0.720   -1.45e+11       1e+11
region_southeast -2.238e+10   6.24e+10     -0.359      0.720   -1.45e+11       1e+11
region_southwest -2.238e+10   6.24e+10     -0.359      0.720   -1.45e+11       1e+11
==============================================================================
Omnibus:                      252.039   Durbin-Watson:                   2.085
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              613.556
Skew:                           1.251   Prob(JB):                    5.86e-134
Kurtosis:                       5.739   Cond. No.                     6.57e+16
==============================================================================
```

```
                                 OLS Regression Results
=======================================================================================
Dep. Variable:                charges   R-squared (uncentered):                   0.110
Model:                            OLS   Adj. R-squared (uncentered):              0.107
Method:                 Least Squares   F-statistic:                              43.90
Date:                Sat, 14 Dec 2024   Prob (F-statistic):                    9.37e-27
Time:                        04:08:31   Log-Likelihood:                         -1447.9
No. Observations:                1070   AIC:                                      2902.
Df Residuals:                    1067   BIC:                                      2917.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
age            0.2562      0.029      8.873      0.000       0.200       0.313
bmi            0.1672      0.029      5.739      0.000       0.110       0.224
children       0.0561      0.028      1.970      0.049       0.000       0.112
==============================================================================
Omnibus:                      263.693   Durbin-Watson:                   1.938
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              489.435
Skew:                           1.524   Prob(JB):                    5.25e-107
Kurtosis:                       4.298   Cond. No.                         1.14
==============================================================================
```

CART Model

The CART (Classification and Regression Trees) model is a non-linear, interpretable algorithm used for predictive modeling, which can handle both regression and classification tasks. For our insurance data, we employ a `DecisionTreeRegressor` to build a regression model based on cartesian splits in the feature space. The CART model was fitted to the training data after tuning the `ccp_alpha` value, which was done with a 5-fold cross validation to select the value with the highest cross validation score. The model built with the optimal `ccp_alpha` value had $OSR^2 = 0.85$, MSE = 0.16, and MAE = 0.24. The increase in model performance compared to linear regression shows how CART was able to capture a potentially nonlinear/non-parametric relationship, providing more nuanced predictions. Though linear regression provides slightly more interpretability and may be easier for customers to understand, having accurate predictions of insurance benefits is more important, and thus a CART model would be preferred in this instance. As seen by the true versus predicted insurance charges plot (see Figure 1, Appendix), predictions are generally accurate, and points are generally clustered around the line with a slope of 1, indicating that the model has no systemic over/underestimation. The tree diagram is also included (see Figure 2, Appendix), which also details how nuance is captured in this model and contributes to interpretability.

Random Forest

With multiple features included in the data, we decided to implement a Random Forest model due to its strength in capturing the potentially complex relations between variables and outcomes.

```
feature_importances = random_forest.feature_importances_
ranking = pd.DataFrame(data=feature_importances, index=X.columns, columns=['Importance'])
ranking = ranking[ranking['Importance'] > 0.01].sort_values(by='Importance', ascending=False)

print(ranking)
```

```
            Importance
smoker_yes    0.608618
bmi           0.216506
age           0.134232
children      0.019413
```

Likewise, we wished to identify key drivers in determining an individual's insurance premium and in turn removing less impactful features that would instead reduce model accuracy.

An initial Random Forest model was generated that included all features resulting in $OSR^2 = 0.86$, MSE = 0.15, MAE = 0.21 which, although, indicated fair performance, did not involve any feature selection and thus could be improved on. Using Random Forests built in Feature Importances, we were able to isolate the features with the highest mean decreases in impurity, and therefore build a more robust model by selecting only the features shown.

However, our new Random Forest model using the selected features did not improve performance, with $OSR^2 = 0.86$, MSE = 0.15, MAE = 0.21. Although when comparing our new and previous model in a scatterplot, the new model had a reduction in outliers and was able to better predict higher insurance charges, whereas the previous model struggled to do so (see Figure 3, Appendix), we hypothesize that since we have very few features in our data, removing a few will not significantly affect model performance.

A potential way to further improve predicting power and analysis within this model would be to include more demographic data and health metrics. The inclusion of additional features may help capture interactions between features and help with hyperparameter tuning allowing the model to increase in nuance and accuracy. While this model improves performance significantly over linear regression, it has similar performance to CART. This is probably due to how CART already significantly improves performance with potentially non-linear data, and that, since there aren't too many features in the data, implementing random forest with feature selection may not contribute significantly. Though the random forest model aims to reduce overfitting, the very tiny increase in performance may not be as valuable compared to the interpretability of CART.

Gradient Boosting

Next, we implemented a Gradient Boosting model due to its ability to sequentially learn from the residuals, making it effective for capturing complex relationships between the input features and target variable. Unlike Random Forest, which averages independent trees, Gradient Boosting builds trees iteratively, improving predictive accuracy at each stage. We hoped this approach would further refine insurance predictions and account for nuanced relationships in the data.

We began with a baseline GradientBoostingRegressor using default hyperparameters, trained on the preprocessed data and evaluated on the test set. The baseline model achieved an OSR^2 score of 0.88, and MSE of 0.13, and an MAE of 0.21, outperforming both Linear Regression and the Random Forest models. To further optimize the Gradient Boosting Model, we conducted hyperparameter tuning using GridSearchCV with 5-fold cross validation (parameters attached).

```
grid_search = GridSearchCV(
    estimator=gbr_model,
    param_grid=grid_values,
    cv=5,  # 5-fold cross-validation
    scoring='r2',  # Optimize for R² score
    verbose=1,
    n_jobs=-1  # Use all available processors
)
```

Using these parameters, the tuned model achieved the following performance metrics on the test set: an OSR^2 score of 0.87, an MSE of 0.13, and an MAE of 0.21.These metrics were chosen to evaluate accuracy and consistency in predicting continuous values, with OSR^2 assessing variance explained and MSE/MAE quantifying prediction error. The slight reduction in OSR^2 compared to the baseline model suggests that hyperparameter tuning reduced overfitting, improving the model's generalizability to unseen data.

```
Fitting 5 folds for each of 576 candidates, totalling 2880 fits
Best Parameters: {'learning_rate': 0.01, 'max_depth': 5, 'min_samples_leaf': 10, 'min_samples_split': 2, 'n_estimators': 300, 'random_state': 42}
```

When compared to the Random Forest model ($OSR^2 = 0.86$), Gradient Boosting offered slightly better performance by capturing nuanced relationships through its iterative learning process. It also requires fewer trees and is less computationally expensive. Compared to CART ($OSR^2 = 0.85$), Gradient Boosting captured more complexity, likely due to its sequential refinement of residuals. However, it sacrifices some interpretability compared to CART and Random Forest, as its predictions are harder to visualize or explain.

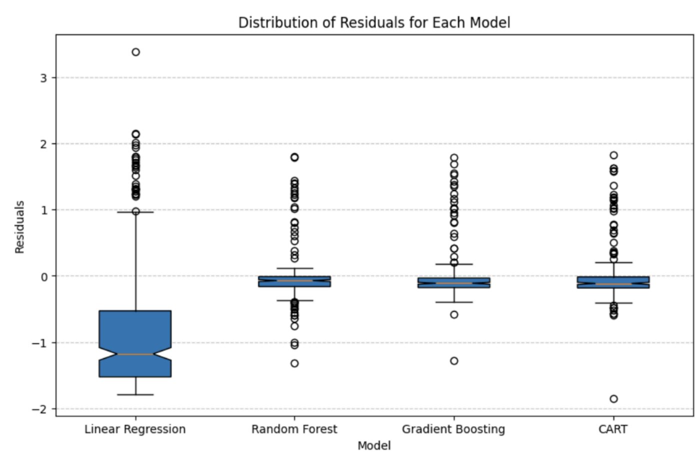
After training, we evaluated feature importance, with smoker, age, and BMI emerging as the most influential features (see Figure 4, Appendix). While these features were not explicitly used for feature selection or interaction analysis, future work could explore creating interaction terms (e.g., smoker × BMI) or building a simplified model using only these features. Incorporating additional features like income, education, or health metrics could further improve accuracy and capture more variance in insurance premiums.

Result/Impact

We reported our results using MSE, MAE, and OSR^2 instead of accuracy since these reporting metrics are more reasonable for a regression problem, as accuracy is generally used for classification. By including information about OSR^2, we are able to see if there is any overfitting and how the model would perform on a new data point. Including MAE and MSE also provided context for average error, which is important to understand from a patient's perspective, as accurate predictions help them anticipate costs and make informed financial decisions. Minimizing large errors (especially evident in MSE) is especially critical to avoid unexpected financial burdens when selecting insurance plans, and thus we consider MSE more strongly.

Model	OSR²	MSE	MAE
Baseline	-0.000919	1.060383	0.792479
Linear Regression	0.783750	0.229098	0.343515
Linear Regression With Feature Selection	0.155189	0.895001	0.755670
Random Forest	0.862540	0.145626	0.211237
Random Forest With Feature Selection	0.859342	0.149014	0.208220
Gradient Boosting	0.874407	0.133054	0.211862
CART	0.853057	0.155672	0.236722

We found that Gradient Boost had the best overall performance of the models that we compared, with the highest OSR^2 and the lowest MSE. Gradient boost did not have the lowest MAE, which may be due to how the data might have a long tail, for which MSE can capture the variability better. However, having an OSR^2 of 0.87 indicates that the model is able to predict the insurance premiums for new cases relatively well.



Based on the box plot, the median residuals for all models are around 0, indicating that the models on average are not significantly biased. Linear Regression had the widest residual spread indicating less accuracy and higher variability compared to Random Forest, Gradient Boosting, and CART. However, while Random Forest, Gradient Boosting, and CART all had narrower spreads (indicating smaller errors), the CART model had a significant amount of outliers which may affect its consistency. Overall, Random Forest and Gradient Boosting appear to have the best performance when minimizing residual spread and dealing with outliers.

This will be very helpful since patients can use this model as a metric to understand their insurance premiums, and how much it will vary based on health and demographic information. This helps inform their decision in financial planning and other healthcare planning-related needs. The scope of this analysis can be expanded to include additional demographic information about the patients (ie previous history of surgery, imaging like x-ray, etc). While the model includes location information which may help generalize the model to various subpopulations, it does not include other information that may affect insurance premiums like race or income. This may also be something to consider in future directions to make the model more generalizable to other subpopulations.

Appendix

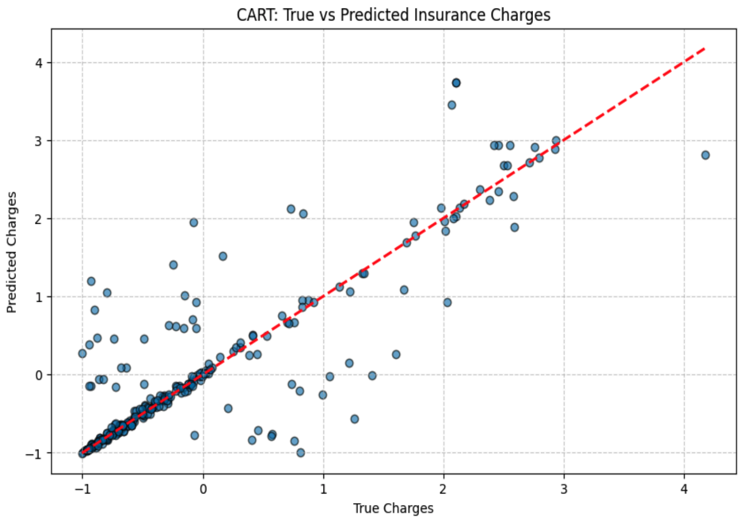


Figure 1: CART: True vs Predicted Insurance Charges

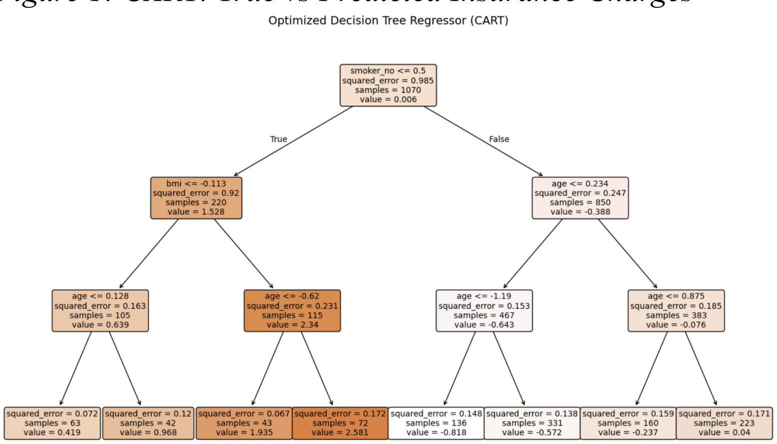


Figure 2: CART Decision Tree Diagram

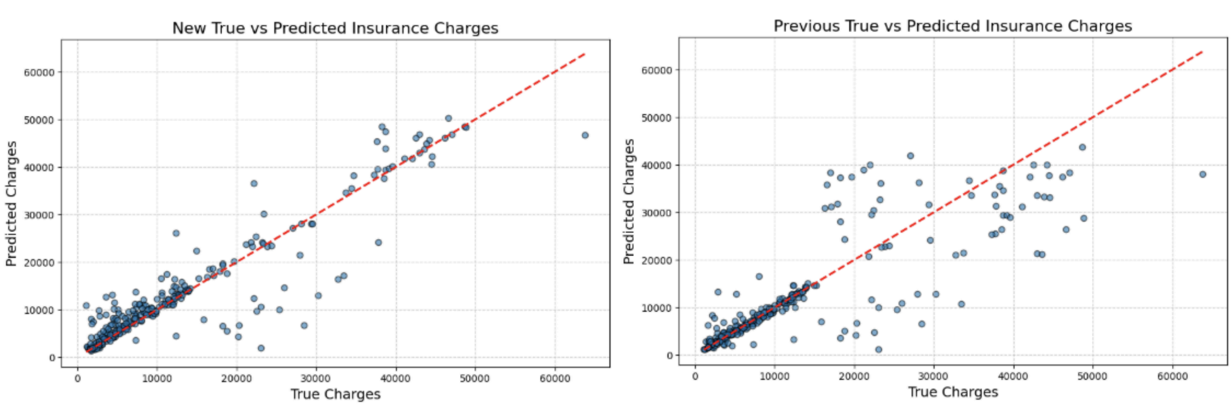


Figure 3: Random Forest Scatterplot Comparison

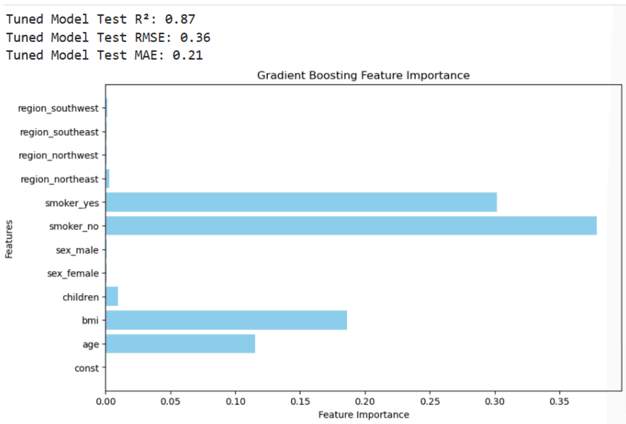


Figure 4: Gradient Boosting Results and Feature Importance

Link to Data
https://github.com/nitin-pandita/Medical-Insurance-Cost-Prediction/blob/deploy/insurance.csv

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
import statsmodels.api as sm
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.model_selection import cross_val_score
```

```
# Read in data

insurance = pd.read_csv('insurance.csv')
insurance
```

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520
...	...	...	...	...	...	...	...
1333	50	male	30.970	3	no	northwest	10600.54830
1334	18	female	31.920	0	no	northeast	2205.98080
1335	18	female	36.850	0	no	southeast	1629.83350
1336	21	female	25.800	0	no	southwest	2007.94500
1337	61	female	29.070	0	yes	northwest	29141.36030

1338 rows × 7 columns

Next steps: Generate code with insurance | View recommended plots | New interactive sheet

```
# Exploratory Data Analysis

df = insurance.copy()
df = sm.add_constant(df)

scaler = StandardScaler()
numerical_cols = ['age', 'bmi', 'children', 'charges']
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])
df = pd.get_dummies(df).astype(float)

X = df.drop('charges', axis=1)
y = df['charges']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Baseline Model

baseline_pred = np.mean(y_train)

print(f"Mean Charge (standardized): {baseline_pred}")

baseline_array = np.full(len(y_test), baseline_pred)

baseline_mae = mean_absolute_error(y_test, baseline_array)
baseline_mse = mean_squared_error(y_test, baseline_array)
baseline_osr2 = r2_score(y_test, baseline_array)

print(f"Linear Regression Mean Squared Error: {baseline_mse:.2f}")
print(f"Linear Regression OSR² Score: {baseline_osr2:.2f}")
print(f"Linear Regression Mean Absolute Error: {baseline_mae:.2f}")
```

```
Mean Charge (standardized): 0.006250676563226175
Linear Regression Mean Squared Error: 1.06
Linear Regression OSR² Score: -0.00
Linear Regression Mean Absolute Error: 0.79
```

```
# Linear Regression With All Features

linear_model_all = sm.OLS(y_train, X_train).fit()

print(linear_model_all.summary())

linear_all_y_pred = linear_model_all.predict(X_test)
linear_all_mse = mean_squared_error(y_test, linear_all_y_pred)
linear_all_osr2 = r2_score(y_test, linear_all_y_pred)
linear_all_mae = mean_absolute_error(y_test, linear_all_y_pred)

print()
print(f"Linear Regression Mean Squared Error: {linear_all_mse:.2f}")
print(f"Linear Regression OSR² Score: {linear_all_osr2:.2f}")
print(f"Linear Regression Mean Absolute Error: {linear_all_mae:.2f}")
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                charges   R-squared:                       0.742
Model:                            OLS   Adj. R-squared:                  0.739
Method:                 Least Squares   F-statistic:                     338.1
Date:                Sat, 14 Dec 2024   Prob (F-statistic):          3.02e-304
Time:                        05:49:03   Log-Likelihood:                -786.02
No. Observations:                1070   AIC:                             1592.
Df Residuals:                    1060   BIC:                             1642.
Df Model:                           9
Covariance Type:            nonrobust
====================================================================================
                       coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------
const            -2.902e+11   8.09e+11     -0.359      0.720   -1.88e+12     1.3e+12
age                  0.2978      0.016     19.007      0.000       0.267       0.329
bmi                  0.1701      0.016     10.378      0.000       0.138       0.202
children             0.0422      0.015      2.740      0.006       0.012       0.072
sex_female       -4.259e+11   1.19e+12     -0.359      0.720   -2.76e+12     1.9e+12
sex_male         -4.259e+11   1.19e+12     -0.359      0.720   -2.76e+12     1.9e+12
```

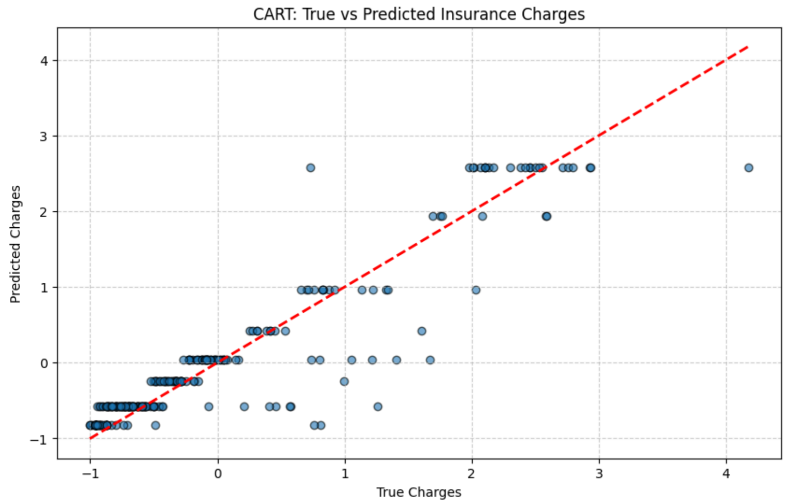
```
    smoker_no         7.385e+11   2.06e+12      0.359      0.720    -3.3e+12    4.78e+12
    smoker_yes        7.385e+11   2.06e+12      0.359      0.720    -3.3e+12    4.78e+12
    region_northeast -2.238e+10   6.24e+10     -0.359      0.720   -1.45e+11       1e+11
    region_northwest -2.238e+10   6.24e+10     -0.359      0.720   -1.45e+11       1e+11
    region_southeast -2.238e+10   6.24e+10     -0.359      0.720   -1.45e+11       1e+11
    region_southwest -2.238e+10   6.24e+10     -0.359      0.720   -1.45e+11       1e+11
    ==============================================================================
    Omnibus:                      252.039   Durbin-Watson:                   2.085
    Prob(Omnibus):                  0.000   Jarque-Bera (JB):              613.556
    Skew:                           1.251   Prob(JB):                    5.86e-134
    Kurtosis:                       5.739   Cond. No.                     6.57e+16
    ==============================================================================

    Notes:
    [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
    [2] The smallest eigenvalue is 6.07e-31. This might indicate that there are
    strong multicollinearity problems or that the design matrix is singular.

    Linear Regression Mean Squared Error: 0.23
    Linear Regression OSR² Score: 0.78
    Linear Regression Mean Absolute Error: 0.34
```

```
# Linear Regression With Feature Selection

X_train_select = X_train[['age', 'bmi', 'children']]
X_test_select = X_test[['age', 'bmi', 'children']]

linear_model_select = sm.OLS(y_train, X_train_select).fit()

print(linear_model_select.summary())

linear_select_y_pred = linear_model_select.predict(X_test_select)
linear_select_mse = mean_squared_error(y_test, linear_select_y_pred)
linear_select_osr2 = r2_score(y_test, linear_select_y_pred)
linear_select_mae = mean_absolute_error(y_test, linear_select_y_pred)

print()
print(f"Linear Regression Feature Selection Mean Squared Error: {linear_select_mse:.2f}")
print(f"Linear Regression Feature Selection OSR² Score: {linear_select_osr2:.2f}")
print(f"Linear Regression Feature Selection Mean Absolute Error: {linear_select_mae:.2f}")
```

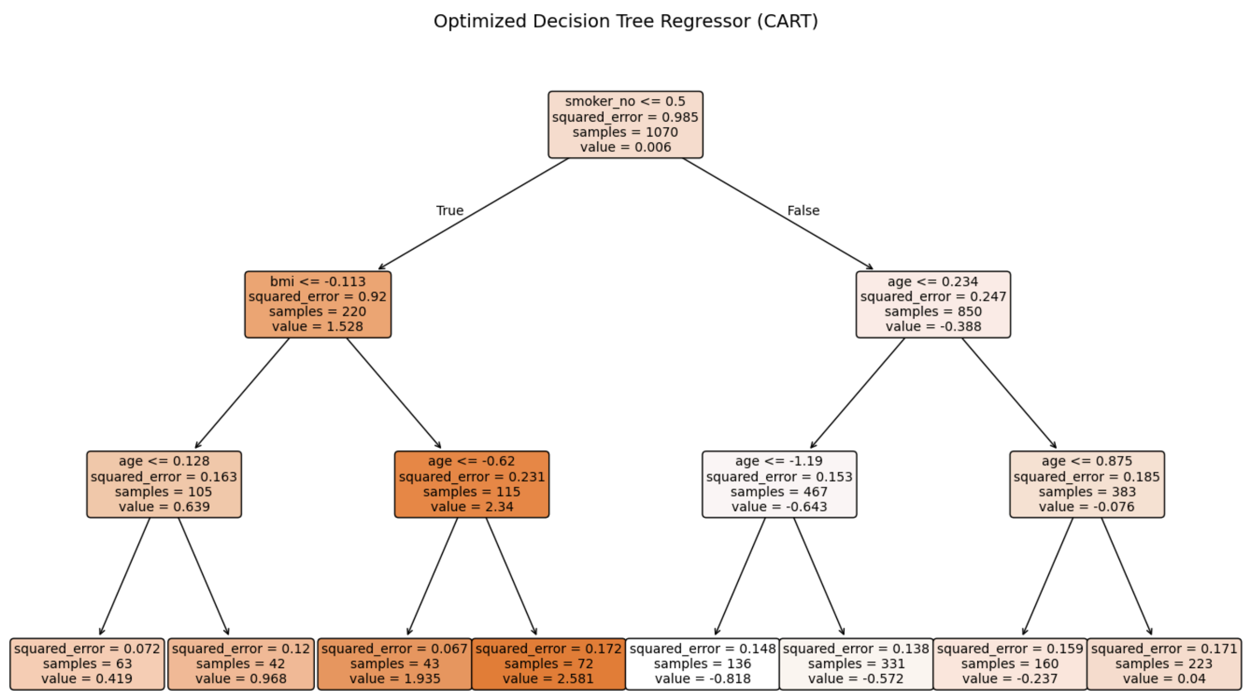
```
                                 OLS Regression Results
=======================================================================================
Dep. Variable:                charges   R-squared (uncentered):                   0.110
Model:                            OLS   Adj. R-squared (uncentered):              0.107
Method:                 Least Squares   F-statistic:                              43.90
Date:                Sat, 14 Dec 2024   Prob (F-statistic):                    9.37e-27
Time:                        05:49:03   Log-Likelihood:                         -1447.9
No. Observations:                1070   AIC:                                      2902.
Df Residuals:                    1067   BIC:                                      2917.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
age            0.2562      0.029      8.873      0.000       0.200       0.313
bmi            0.1672      0.029      5.739      0.000       0.110       0.224
children       0.0561      0.028      1.970      0.049       0.000       0.112
==============================================================================
Omnibus:                      263.693   Durbin-Watson:                   1.938
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              489.435
Skew:                           1.524   Prob(JB):                    5.25e-107
Kurtosis:                       4.298   Cond. No.                         1.14
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Linear Regression Feature Selection Mean Squared Error: 0.90
Linear Regression Feature Selection OSR² Score: 0.16
Linear Regression Feature Selection Mean Absolute Error: 0.76
```

```
# CART model building and cross validation

clf = DecisionTreeRegressor(random_state=42)

path = clf.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas = path.ccp_alphas

clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeRegressor(random_state=42, ccp_alpha=ccp_alpha)
    clf.fit(X_train, y_train)
    clfs.append(clf)

cv_scores = [cross_val_score(clf, X_train, y_train, cv=5).mean() for clf in clfs]

best_alpha_idx = np.argmax(cv_scores)
best_alpha = ccp_alphas[best_alpha_idx]

# Build best model
cart_model = DecisionTreeRegressor(random_state=42, ccp_alpha=best_alpha)
cart_model.fit(X_train, y_train)

# Evaluate on the test set
cart_y_pred = cart_model.predict(X_test)

cart_mae = mean_absolute_error(y_test, cart_y_pred)
cart_mse = mean_squared_error(y_test, cart_y_pred)
cart_osr2 = r2_score(y_test, cart_y_pred)

print(f"CART Mean Squared Error: {cart_mse:.2f}")
print(f"CART Mean Absolute Error: {cart_mae:.2f}")
print(f"CART OSR² Score: {cart_osr2:.2f}")

# Plot True vs Predicted for CART
plt.figure(figsize=(10, 6))
plt.scatter(y_test, cart_y_pred, alpha=0.6, edgecolors="k")
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--', linewidth=2)
plt.title("CART: True vs Predicted Insurance Charges")
plt.xlabel("True Charges")
plt.ylabel("Predicted Charges")
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```

```
CART Mean Squared Error: 0.16
CART Mean Absolute Error: 0.24
CART OSR² Score: 0.85
```



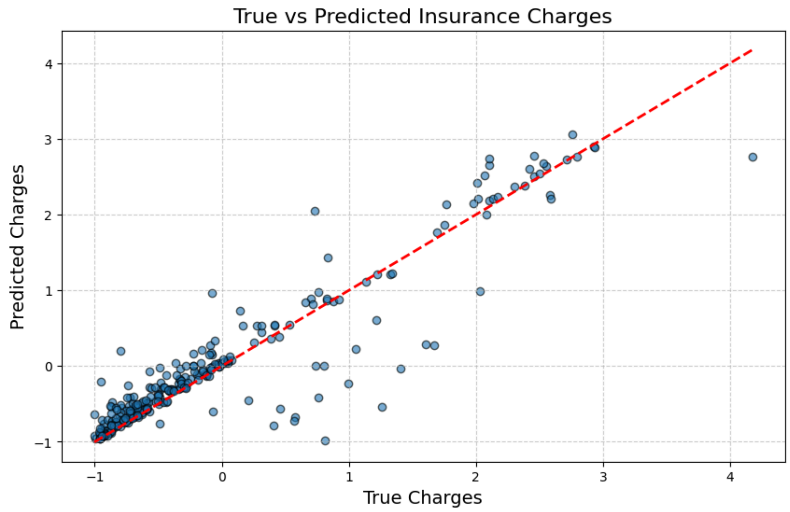

```
plt.figure(figsize=(17, 10))
plot_tree(
    cart_model,
    feature_names=X_train.columns,
    filled=True,
    rounded=True,
    fontsize=10,
    max_depth=3  # Limit depth for better visualization
)
plt.title("Optimized Decision Tree Regressor (CART)", fontsize=14)
plt.show()
```




```
# Random Forest

rf_model = RandomForestRegressor(random_state=42)
rf_model.fit(X_train, y_train)

rf_y_pred = rf_model.predict(X_test)

rf_r2 = r2_score(y_test, rf_y_pred)
rf_mse = mean_squared_error(y_test, rf_y_pred)
rf_mae = mean_absolute_error(y_test, rf_y_pred)

print(f"Random Forest Mean Squared Error: {rf_mse:.2f}")
print(f"Random Forest OSR² Score: {rf_r2:.2f}")
print(f"Random Forest Mean Absolute Error: {rf_mae:.2f}")
```

```
Random Forest Mean Squared Error: 0.15
Random Forest OSR² Score: 0.86
Random Forest Mean Absolute Error: 0.21
```

```
plt.figure(figsize=(10, 6))
plt.scatter(y_test, rf_y_pred, alpha=0.6, edgecolors="k")
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--', linewidth=2)
plt.title("True vs Predicted Insurance Charges", fontsize=16)
plt.xlabel("True Charges", fontsize=14)
plt.ylabel("Predicted Charges", fontsize=14)
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```



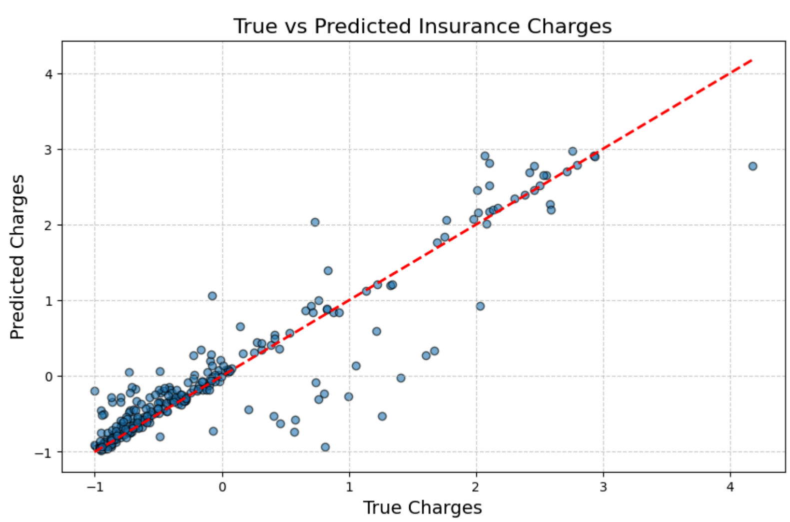

```
# these features have strongest impact on model predictions using threshold of > 0.01

feature_importances = rf_model.feature_importances_
ranking = pd.DataFrame(data=feature_importances, index=X.columns, columns=['Importance'])
ranking = ranking[ranking['Importance'] > 0.01].sort_values(by='Importance', ascending=False)

print(ranking)
```

```
            Importance
smoker_yes    0.328406
smoker_no     0.280213
bmi           0.212681
age           0.134473
children      0.019444
```

```
new_features = ['smoker_no', 'bmi', 'age', 'children']
X_train_new = X_train[new_features]
X_test_new = X_test[new_features]

rf_nmodel = RandomForestRegressor(random_state=42)
rf_nmodel.fit(X_train_new, y_train)

y_npred = rf_nmodel.predict(X_test_new)

rf_selected_osr2 = r2_score(y_test, y_npred)
rf_selected_mse = mean_squared_error(y_test, y_npred)
rf_selected_mae = mean_absolute_error(y_test, y_npred)

print(f"Random Forest Feature Selection Mean Squared Error: {rf_selected_mse:.2f}")
print(f"Random Forest Feature Selection OSR² Score: {rf_selected_osr2:.2f}")
print(f"Random Forest Feature Selection Mean Absolute Error: {rf_selected_mae:.2f}")
```

```
Random Forest Feature Selection Mean Squared Error: 0.15
Random Forest Feature Selection OSR² Score: 0.86
Random Forest Feature Selection Mean Absolute Error: 0.21
```

```
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_npred, alpha=0.6, edgecolors="k")
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--', linewidth=2)
plt.title("True vs Predicted Insurance Charges", fontsize=16)
plt.xlabel("True Charges", fontsize=14)
plt.ylabel("Predicted Charges", fontsize=14)
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```



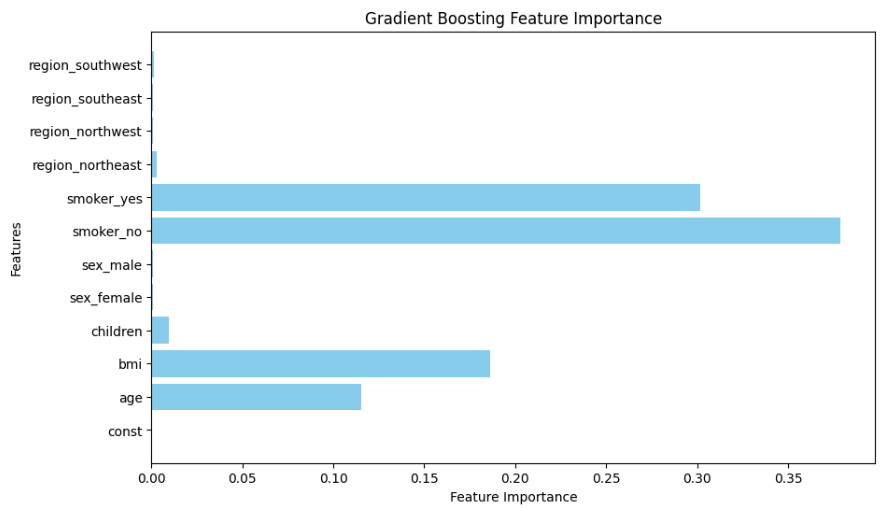

```
#Gradient Boosting
gbr_model = GradientBoostingRegressor(random_state=42)

# Define the parameter grid (adjusted for Gradient Boosting)
grid_values = {
    'n_estimators': [100, 200, 300, 500],
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
    'max_depth': [5, 10, 15, 20],
    'min_samples_leaf': [1, 5, 10],
    'min_samples_split': [2, 10, 20],
    'random_state': [42]
}
```

```
# Perform grid search with cross-validation
grid_search = GridSearchCV(
    estimator=gbr_model,
    param_grid=grid_values,
    cv=5,  # 5-fold cross-validation
    scoring='r2',  # Optimize for R² score
    verbose=1,
    n_jobs=-1  # Use all available processors
)

# Fit the model with the training data
grid_search.fit(X_train, y_train)

# Extract the results
mean_test_scores = grid_search.cv_results_['mean_test_score']
param_combinations = grid_search.cv_results_['params']
best_params = grid_search.best_params_
best_score = grid_search.best_score_

# Print the results
print(f"Best Parameters: {best_params}")
print(f"Best R² Score: {best_score:.2f}")
```

```
Fitting 5 folds for each of 576 candidates, totalling 2880 fits
Best Parameters: {'learning_rate': 0.01, 'max_depth': 5, 'min_samples_leaf': 10, 'min_samples_split': 2, 'n_estimators': 300, 'random_state': 42}
Best R² Score: 0.84
```

```
# Best model from GridSearchCV
best_model = grid_search.best_estimator_

# Evaluate on the test set
gb_y_pred = best_model.predict(X_test)
gb_osr2 = r2_score(y_test, gb_y_pred)
gb_mse = mean_squared_error(y_test, gb_y_pred)
gb_mae = mean_absolute_error(y_test, gb_y_pred)

# Print metrics
print(f"Gradient Boost Test OSR²: {gb_osr2:.2f}")
print(f"Gradient Boost Test MSE: {gb_mse:.2f}")
print(f"Gradient Boost Test MAE: {gb_mae:.2f}")

# Feature Importance
feature_importance = best_model.feature_importances_
features = X_train.columns

# Plot Feature Importance
plt.figure(figsize=(10, 6))
plt.barh(features, feature_importance, color='skyblue')
plt.xlabel('Feature Importance')
plt.ylabel('Features')
plt.title('Gradient Boosting Feature Importance')
plt.show()
```

```
Gradient Boost Test OSR²: 0.87
Gradient Boost Test MSE: 0.13
Gradient Boost Test MAE: 0.21
```

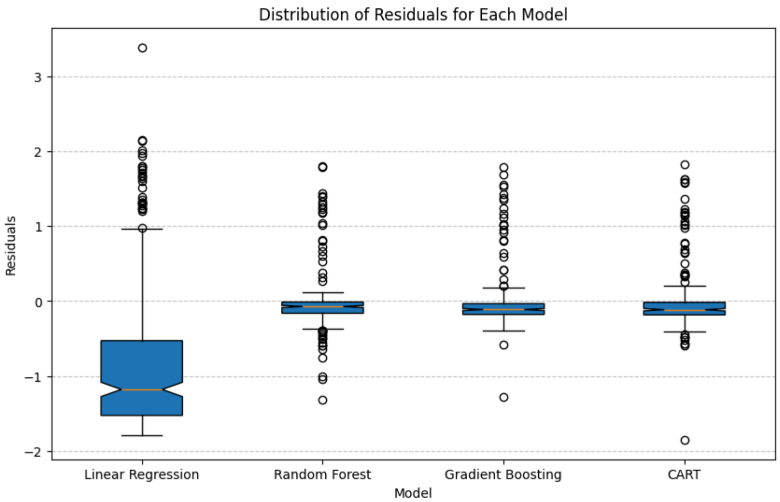



```
# Creates a df with the model and our chosen performance metrics

results = {
    'Model': ['Baseline', 'Linear Regression', 'Linear Regression With Feature Selection', 'Random Forest', 'Random Forest With Feature Selection', 'Gradient Boosting', 'CART'],
    'OSR²': [baseline_osr2, linear_all_osr2, linear_select_osr2, rf_r2, rf_selected_osr2, gb_osr2, cart_osr2],
    'MSE': [baseline_mse, linear_all_mse, linear_select_mse, rf_mse, rf_selected_mse, gb_mse, cart_mse],
    'MAE': [baseline_mae, linear_all_mae, linear_select_mae, rf_mae, rf_selected_mae, gb_mae, cart_mae]
}

results_df = pd.DataFrame(results)

results_df.set_index('Model', inplace=True)

results_df['MSE'] = results_df['MSE'].round(6)

print(results_df)
```

```
                                              OSR²       MSE       MAE
Model
Baseline                                 -0.000919  1.060383  0.792479
Linear Regression                         0.783750  0.229098  0.343515
Linear Regression With Feature Selection  0.155189  0.895001  0.755670
Random Forest                             0.862540  0.145626  0.211237
Random Forest With Feature Selection      0.859342  0.149014  0.208220
Gradient Boosting                         0.874407  0.133054  0.211862
CART                                      0.853057  0.155672  0.236722
```

```
# Creates a box and whisker plot for the distribution of residuals for each model

residuals = {
    'Linear Regression': y_test - linear_all_osr2,
```

```python
    'Random Forest': y_test - rf_y_pred,
    'Gradient Boosting': y_test - gb_y_pred,
    'CART': y_test - cart_y_pred
}

# Create a box plot for residuals
plt.figure(figsize=(10, 6))
plt.boxplot(residuals.values(), labels=residuals.keys(), patch_artist=True, notch=True)
plt.title('Distribution of Residuals for Each Model')
plt.xlabel('Model')
plt.ylabel('Residuals')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



Start coding or generate with AI.