UBC SAUDER SCHOOL OF BUSINESS

ROBERT H. LEE GRADUATE SCHOOL

BAIT 509 Final Project Group 10

#	Name	Student ID
1	Justin Lin	34209445
2	Ronghao Wu	19947225
3	Pranav Mehta	19184282
4	Baris Camli	64198302
5	Angel Zhou	22429575

Insurance Claim Prediction: A Machine Learning Approach

Instructor Name: Andy Zheng

TA: Darren Li

Submission Date: 14th February, 2025

Table of Contents

1. Introduction:

In the insurance industry, accurately predicting claim amounts is crucial for managing financial risk, optimizing premium pricing, and ensuring customer satisfaction. The ability to estimate future claims based on demographic and health-related factors can help insurance companies allocate resources more efficiently while maintaining profitability.

This project aims to build a supervised learning model to predict insurance claim amounts using a dataset that includes both demographic and health-related attributes of policyholders. The key objective is to leverage machine learning techniques to improve the accuracy of claim predictions, reducing uncertainty and financial losses for insurers.

Through this study, we will explore various feature engineering techniques, assess the impact of different regression models, and evaluate model performance based on relevant metrics. The findings from this project will provide insights into how data-driven decision-making can enhance risk assessment in the insurance sector.

2. Background, Motivation, and Business Question:

- *Background & Motivation:*
  The insurance industry faces significant challenges in estimating claim amounts due to the diverse range of factors influencing healthcare costs. Insurance companies must accurately predict claim amounts to set appropriate premiums, manage financial reserves, and reduce fraudulent claims. However, traditional actuarial models often rely on simplified assumptions and may fail to capture complex relationships in the data.

  Machine learning (ML) provides an advanced, data-driven approach to enhance prediction accuracy by leveraging a wide array of demographic and health-related variables. With the growing availability of data, insurers can use ML techniques to uncover patterns that were previously undetectable using conventional statistical methods.

- *Business Question:*
  The key business question we aim to answer is:

  "Can we develop a machine learning model that accurately predicts insurance claim amounts based on demographic and health-related factors?"

  By answering this question, we seek to provide actionable insights that can help insurance companies optimize risk assessment, improve premium pricing strategies, and enhance financial planning.

3. Data and Statistical Question:

- *Dataset Overview*
  The dataset used in this study contains demographic and health-related information about individuals, along with their corresponding insurance claim amounts. The goal is to analyze the dataset to understand how different features contribute to the final claim amount and use this knowledge to develop a predictive model.

The dataset includes the following key attributes:

- *Demographic Factors:* Age, Gender, Region, etc.
- *Health Related Factors:* BMI, Smoking Status, Number of Dependents, etc.
- *Target Variable:* Total Insurance Amount Claim

- *Statistical Question*

  To address the business question, we define the following statistical question:

  <u>"What is the expected insurance claim amount for a given individual based on their demographic and health characteristics?"</u>

  By framing this as a regression problem, we aim to predict a continuous numerical outcome (claim amount) using various independent variables (features).

```
import pandas as pd

# Load the dataset
file_path = "Insurance Claim Analysis Demographic and Health.csv"  # Update
with the actual path
df = pd.read_csv(file_path)

# Display basic information and first few rows
print("Dataset Overview:")
print(df.info())
print("\nFirst Few Rows of the Dataset:")
print(df.head())
```

```
Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1340 entries, 0 to 1339
Data columns (total 11 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   index          1340 non-null   int64
 1   PatientID      1340 non-null   int64
 2   age            1335 non-null   float64
 3   gender         1340 non-null   object
 4   bmi            1340 non-null   float64
 5   bloodpressure  1340 non-null   int64
 6   diabetic       1340 non-null   object
 7   children       1340 non-null   int64
 8   smoker         1340 non-null   object
 9   region         1337 non-null   object
 10  claim          1340 non-null   float64
dtypes: float64(3), int64(4), object(4)
memory usage: 115.3+ KB
None

First Few Rows of the Dataset:
   index  PatientID   age gender   bmi  bloodpressure diabetic  children  \
0      0          1  39.0   male  23.2             91      Yes         0
1      1          2  24.0   male  30.1             87       No         0
2      2          3   NaN   male  33.3             82      Yes         0
3      3          4   NaN   male  33.7             80       No         0
4      4          5   NaN   male  34.1            100       No         0

  smoker     region    claim
0     No  southeast  1121.87
1     No  southeast  1131.51
2     No  southeast  1135.94
3     No  northwest  1136.40
4     No  northwest  1137.01
```

4. **Exploratory Data Analysis (EDA):**

- *Summary Statistics and Missing Value Analysis:*
  To begin understanding the dataset, we calculated summary statistics for all numerical columns using the .describe() method. This includes measures such as mean, standard deviation, and percentiles to identify key characteristics of the data:

  1. *Key Insights from Summary Statistics:*
     - The age column ranges from 18 to 60.
     - The claim amount varies widely, from a minimum of 1121.87 to a maximum of 63770.43, with an average claim of approximately 13252.75.
     - BMI values range from 16 to 53.1, indicating significant diversity in the population's health profiles.

  2. *Missing Values Analysis:*
     Analyzing missing values is critical to ensure the data's quality. Using df.isnull().sum(), we identified the following:

     - The age column has 5 missing values, accounting for 0.37% of the dataset.
     - No missing values were observed in other columns.

  Next Steps: Missing values in the age column will be handled during data preprocessing, where imputation techniques such as filling with the mean, median, or mode will be considered.

```
# Summary statistics of numerical columns

print("\nSummary Statistics:")
print(df.describe())

# Check for missing values
missing_values_summary = df.isnull().sum()
missing_percentage = (missing_values_summary / len(df)) * 100
missing_df = pd.DataFrame({
    "Missing Values": missing_values_summary,
    "Percentage Missing (%)": missing_percentage
})

print("\nMissing Values Summary:")
print(missing_df)
```

Summary Statistics:

	index	PatientID	age	bmi	bloodpressure	children	claim
count	1340.000000	1340.000000	1335.000000	1340.000000	1340.000000	1340.000000	1340.000000
mean	669.500000	670.500000	38.078652	30.668955	94.157463	1.093284	13252.745642
std	386.968991	386.968991	11.102924	6.106735	11.434712	1.205334	12109.609288
min	0.000000	1.000000	18.000000	16.000000	80.000000	0.000000	1121.870000
25%	334.750000	335.750000	29.000000	26.275000	86.000000	0.000000	4719.685000
50%	669.500000	670.500000	38.000000	30.400000	92.000000	1.000000	9369.615000
75%	1004.250000	1005.250000	47.000000	34.700000	99.000000	2.000000	16604.305000
max	1339.000000	1340.000000	60.000000	53.100000	140.000000	5.000000	63770.430000

Missing Values Summary:

	Missing Values	Percentage Missing (%)
index	0	0.000000
PatientID	0	0.000000
age	5	0.373134
gender	0	0.000000
bmi	0	0.000000
bloodpressure	0	0.000000
diabetic	0	0.000000
children	0	0.000000
smoker	0	0.000000
region	3	0.223881
claim	0	0.000000

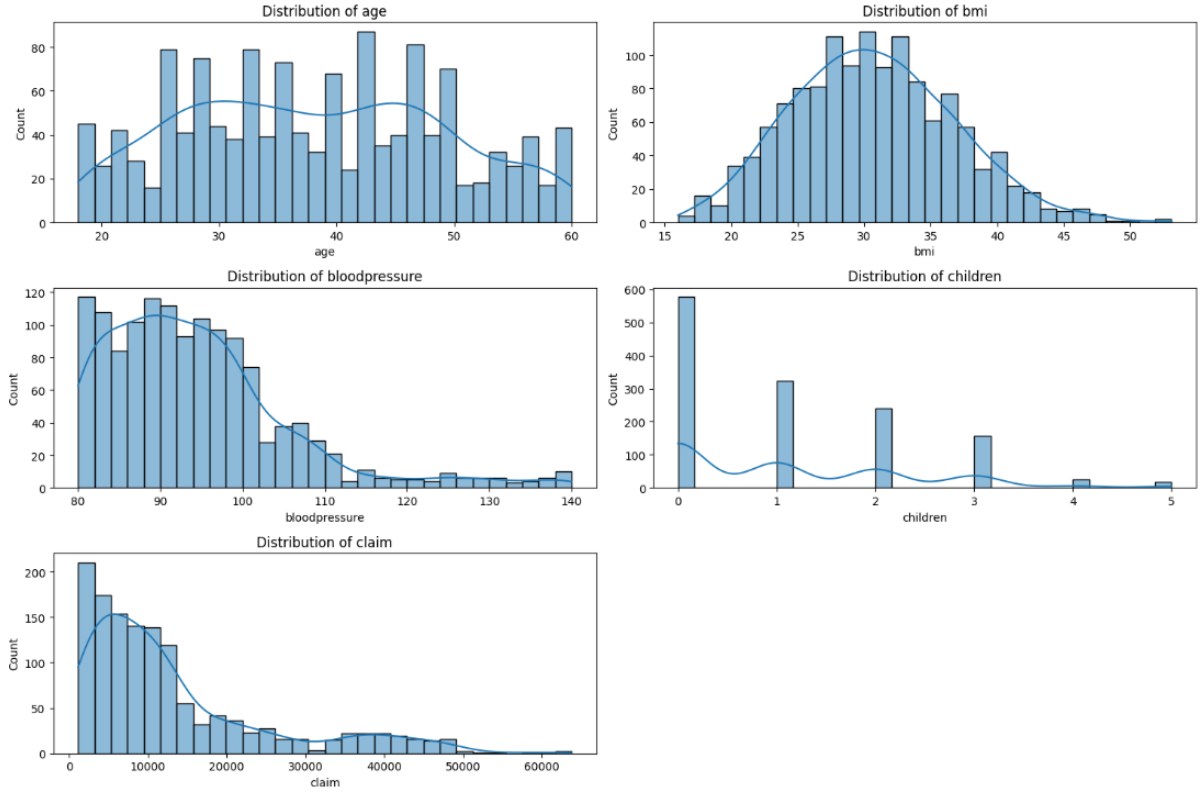
- *Distribution of Numerical Values:*
  Understanding the distribution of numerical variables is essential for identifying patterns, potential skewness, and preprocessing needs. The analysis focused on the following variables:
    - Age: The age variable is uniformly distributed across the range of 18 to 60, indicating a balanced representation of individuals from different age groups.
    - BMI: BMI exhibits a slightly right-skewed distribution, with the majority of values concentrated around 30. This suggests that the population predominantly falls within the overweight or borderline obese category.
    - Blood Pressure: The blood pressure variable demonstrates a normal-like distribution, centered around 92, which aligns with typical population norms for systolic blood pressure.
    - Children: The children variable is discrete, with most data points clustered around 0 or 1 child, reflecting common family sizes in the dataset.
    - Claim Amount: The target variable, claim amount, is heavily right-skewed, indicating that while most claims are on the lower end, a small number of outliers significantly increase variability in the data.

  Insights:
    - The right-skewed distributions of BMI and claim amount suggest that these variables may require logarithmic transformation to normalize the data for predictive modeling
    - The discrete nature of the children variable suggests it could be treated as a categorical feature in specific contexts, depending on the modeling approach.

These insights will guide data preprocessing steps, including feature scaling and transformations, to enhance model performance.

```
import matplotlib.pyplot as plt
import seaborn as sns

# Plot distributions of numerical variables
numerical_cols = ['age', 'bmi', 'bloodpressure', 'children', 'claim']
plt.figure(figsize=(15, 10))
for i, col in enumerate(numerical_cols):
    plt.subplot(3, 2, i + 1)
    sns.histplot(df[col], kde=True, bins=30)
    plt.title(f"Distribution of {col}")
plt.tight_layout()
plt.show()
```



- *Correlation Analysis:*
  To understand the relationships between numerical variables in the dataset, a correlation heatmap was generated. This analysis identifies features that are closely related to each other and to the target variable (claim amount), guiding feature selection and preprocessing for machine learning models.

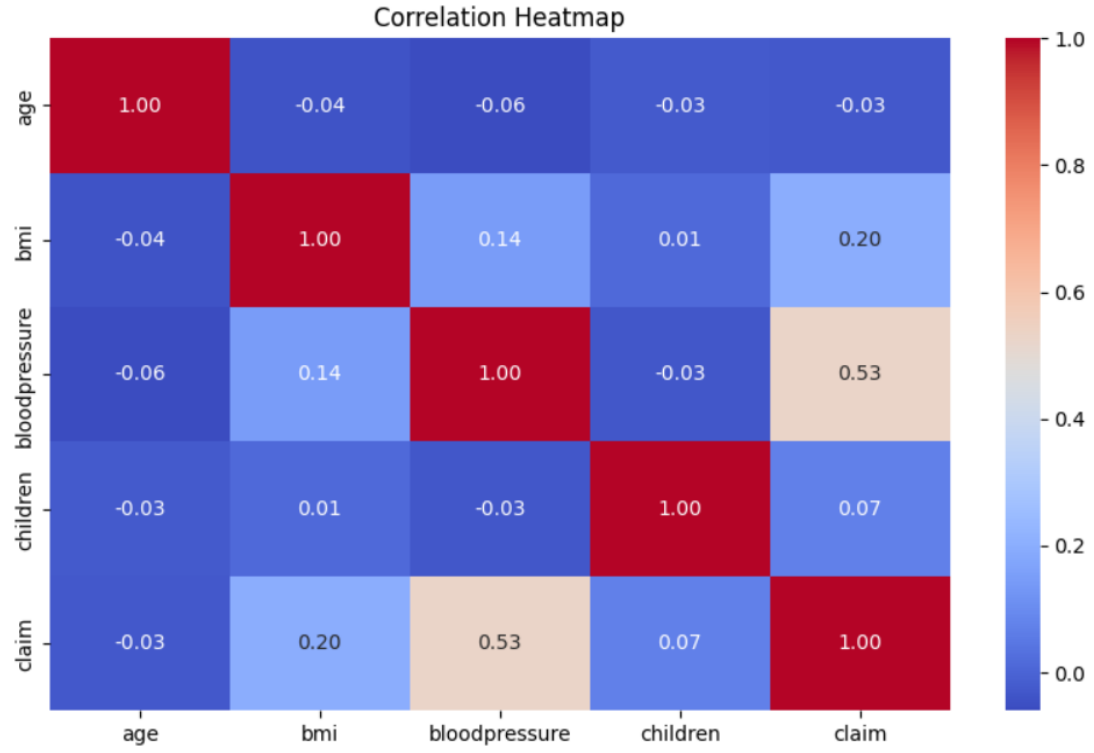
  1. Claim Amount:
  - Exhibits a moderate positive correlation with blood pressure (0.53), indicating that higher blood pressure levels might be associated with increased insurance claims.
  - Displays a weak positive correlation with BMI (0.20) and Children (0.07), suggesting a smaller, yet observable, impact of these variables on claim amounts.
  - Shows a negligible correlation with Age (-0.03), implying age may not directly influence claim amounts in this dataset.

2. Blood Pressure:
   - Positively correlated with BMI (0.14), reflecting a weak association between these health-related factors.
   - Has negligible or no correlation with other variables.

Insights:

- The moderate correlation between blood pressure and claim amount suggests that blood pressure is an important predictor for the target variable and should be prioritized during feature selection.
- Variables like BMI and Children, although weakly correlated, may still contribute to the model's performance in combination with other features.
- Age may require further exploration or be dropped depending on its impact during modeling.

```
# Correlation heatmap for numerical variables
plt.figure(figsize=(10, 6))
sns.heatmap(df[numerical_cols].corr(), annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Heatmap")
plt.show()
```



5. **Data Splitting:**

To prepare the data for model training and evaluation, the dataset was split into training (80%) and test sets (20%) using train_test_split from sklearn.model_selection. The split ensures a separate test set for unbiased performance evaluation, with random_state=42 for reproducibility.

Key Details:

- Features (X): All columns except the target (claim).
- Target (y): Claim amount.
- Training Set: 1072 samples, 10 columns.
- Test Set: 268 samples, 10 columns.

This approach balances feature representation and preserves data integrity for evaluation.

```
from sklearn.model_selection import train_test_split

# Define features (X) and target (y)
X = df.drop(['claim'], axis=1)
y = df['claim']

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Display sizes of the splits
print(f"Training Set: {X_train.shape}, {y_train.shape}")
print(f"Test Set: {X_test.shape}, {y_test.shape}")
```

6. **Handling Missing Values:**

To address missing values in the dataset, the following strategies were implemented:

- *Missingness Indicators:*
  - Created binary columns to capture missingness:
    - age_missing: Indicates missing values in age (1 for missing, 0 otherwise).
    - region_missing: Indicates missing values in region (1 for missing, 0 otherwise).
- *Imputation:*
  - Missing age values were imputed with 0.
  - Missing region values were replaced with "Unknown".
- *Verification:*
  - Ensured no missing values remained using isnull().sum().

Results:

- Training and test datasets are now free of missing values.
- Additional columns (age_missing and region_missing) provide information on imputed data.

These preprocessing steps ensure data integrity and retain information about missing patterns for modeling.

```
# Step 1: Handling Missing Values with Indicators

# Add missingness indicator for 'age'
X_train['age_missing'] = X_train['age'].isnull().astype(int)
X_test['age_missing'] = X_test['age'].isnull().astype(int)

# Impute missing values in 'age' with 0
X_train['age'].fillna(0, inplace=True)
X_test['age'].fillna(0, inplace=True)

# Add missingness indicator for 'region'
X_train['region_missing'] = X_train['region'].isnull().astype(int)
X_test['region_missing'] = X_test['region'].isnull().astype(int)

# Impute missing values in 'region' with 'Unknown'
X_train['region'].fillna('Unknown', inplace=True)
X_test['region'].fillna('Unknown', inplace=True)

# Verify no missing values remain
print("\nMissing Values After Imputation (Training Data):")
print(X_train.isnull().sum())

print("\nMissing Values After Imputation (Test Data):")
print(X_test.isnull().sum())
```

```
Missing Values After Imputation (Training Data):
index             0
PatientID         0
age               0
gender            0
bmi               0
bloodpressure     0
diabetic          0
children          0
smoker            0
region            0
age_missing       0
region_missing    0
dtype: int64

Missing Values After Imputation (Test Data):
index             0
PatientID         0
age               0
gender            0
bmi               0
bloodpressure     0
diabetic          0
children          0
smoker            0
region            0
age_missing       0
region_missing    0
dtype: int64
```

7. **Encoding Categorical Variables:**

To prepare categorical variables for machine learning models, we applied one-hot encoding to convert them into numerical representations.

Key Steps:

- *Selected Categorical Columns:*
  - Gender
  - Smoker
  - Diabetic
  - Region
- *One-Hot Encoding:*
  - Used pd.get_dummies to generate binary columns for each category, dropping the first category to avoid multicollinearity.
  - Encoding was applied separately to the training (X_train) and test (X_test) datasets.
- *Column Alignment:*
  - Ensured that the encoded test dataset (X_test_encoded) had the same columns as the training dataset (X_train_encoded) by reindexing with fill_value=0 for missing columns.

Results:

- Encoded Training Data: New binary columns were created for the categorical variables.
- Encoded Test Data: Fully aligned with the training data, maintaining column consistency.

This step ensures that categorical data is transformed into a format suitable for machine learning algorithms, improving compatibility and model performance.

```
# Step 2: Encoding Categorical Variables

# List of categorical columns to encode
categorical_cols = ['gender', 'smoker', 'diabetic', 'region']

# Apply one-hot encoding to both training and test datasets
X_train_encoded = pd.get_dummies(X_train, columns=categorical_cols,
drop_first=True)
X_test_encoded = pd.get_dummies(X_test, columns=categorical_cols,
drop_first=True)

# Ensure both datasets have the same columns after encoding
X_test_encoded = X_test_encoded.reindex(columns=X_train_encoded.columns,
fill_value=0)

# Display encoded datasets
print("\nEncoded Training Data:")
print(X_train_encoded.head())

print("\nEncoded Test Data:")
print(X_test_encoded.head())
```

Encoded Training Data:

	index	PatientID	age	bmi	bloodpressure	children	age_missing	\
1148	1148	1149	32.0	26.7	115	1	0	
807	807	808	51.0	25.7	83	0	0	
1287	1287	1288	32.0	39.0	96	0	0	
590	590	591	38.0	23.4	96	3	0	
1188	1188	1189	37.0	28.3	88	3	0	

	region_missing	gender_male	smoker_Yes	diabetic_Yes	region_northeast	\
1148	0	True	True	True	False	
807	0	False	False	True	False	
1287	0	False	True	True	False	
590	0	False	False	True	True	
1188	0	True	True	False	False	

	region_northwest	region_southeast	region_southwest
1148	True	False	False
807	True	False	False
1287	True	False	False
590	False	False	False
1188	True	False	False

Encoded Test Data:

	index	PatientID	age	bmi	bloodpressure	children	age_missing	\
394	394	395	29.0	40.2	85	0	0	
881	881	882	36.0	26.8	88	1	0	
358	358	359	25.0	25.8	97	2	0	
367	367	368	19.0	17.9	90	1	0	
259	259	260	18.0	30.9	92	2	0	

	region_missing	gender_male	smoker_Yes	diabetic_Yes	region_northeast	\
394	0	False	False	True	0	
881	0	True	False	True	0	
358	0	False	False	True	0	
367	0	True	False	False	0	
259	0	True	False	False	0	

	region_northwest	region_southeast	region_southwest
394	False	True	False
881	True	False	False
358	False	False	True
367	True	False	False
259	True	False	False

8. Feature Engineering: Adding New Features:

To enhance the dataset and improve the predictive power of the model, new features were engineered based on domain knowledge. These features capture important health risk indicators and enrich the dataset for machine learning.

Key Steps:

- *Risk Score:*
  - A composite health risk score was created by summing points assigned to key risk factors:
    1. Smoking: +3 points for smoker_Yes.
    2. Diabetes: +2 points for diabetic_Yes.
    3. Obesity: +2 points for bmi > 30.
    4. High Blood Pressure: +2 points for bloodpressure > 120.
  - The new column, risk_score, quantifies the overall health risk for each individual.
- *BMI Categories:*
  - BMI values were grouped into clinical categories:
    1. Underweight: bmi < 18.5.
    2. Normal: 18.5 ≤ bmi < 25.
    3. Overweight: 25 ≤ bmi < 30.
    4. Obese: bmi ≥ 30.

- One-hot encoding was applied to these categories, creating columns like bmi_cat_Overweight and bmi_cat_Obese.

Results:

- *New Features:*
  - risk_score: Composite score indicating health risks.
  - Encoded BMI categories: bmi_cat_Overweight and bmi_cat_Obese.
- *Dataset Update:*
  - Training data now includes 19 columns, with the new features added.
  - A preview of the new features shows:
    - risk_score: A numeric indicator of cumulative health risk.
    - bmi_cat_Overweight and bmi_cat_Obese: Encoded BMI categories.

```
# Step 3: Feature Engineering

# 1. Risk Score - Composite health risk indicator
def calculate_risk_score(row):
    score = 0
    # Add points for key risk factors
    if row['smoker_Yes']:
        score += 3  # Smoking is a major risk factor
    if row['diabetic_Yes']:
        score += 2  # Diabetes is significant
    if row['bmi'] > 30:
        score += 2  # Obesity impact
    if row['bloodpressure'] > 120:
        score += 2  # High blood pressure
    return score

# Add risk score to both datasets
X_train_encoded['risk_score'] = X_train_encoded.apply(calculate_risk_score,
axis=1)
X_test_encoded['risk_score'] = X_test_encoded.apply(calculate_risk_score,
axis=1)

# 2. BMI Categories - Clinical groupings
def create_bmi_category(bmi):
    if bmi < 18.5:
        return 'Underweight'
    elif 18.5 <= bmi < 25:
        return 'Normal'
    elif 25 <= bmi < 30:
        return 'Overweight'
    else:
        return 'Obese'

# Add BMI categories and encode
X_train_encoded['bmi_category'] =
X_train_encoded['bmi'].apply(create_bmi_category)
```

```
X_test_encoded['bmi_category'] =
X_test_encoded['bmi'].apply(create_bmi_category)

# One-hot encode BMI categories
bmi_dummies_train = pd.get_dummies(X_train_encoded['bmi_category'],
prefix='bmi_cat', drop_first=True)
bmi_dummies_test = pd.get_dummies(X_test_encoded['bmi_category'],
prefix='bmi_cat', drop_first=True)

# Add encoded BMI categories to datasets
X_train_encoded = pd.concat([X_train_encoded.drop('bmi_category', axis=1),
bmi_dummies_train], axis=1)
X_test_encoded = pd.concat([X_test_encoded.drop('bmi_category', axis=1),
bmi_dummies_test], axis=1)

# Verify new features
print("\nNew features added. Updated shape:", X_train_encoded.shape)
print("\nNew features preview:")
print(X_train_encoded[['risk_score', 'bmi_cat_Overweight',
'bmi_cat_Obese']].head())
```

```
New features added. Updated shape: (1072, 19)

New features preview:
      risk_score  bmi_cat_Overweight  bmi_cat_Obese
1148           5                True          False
807            2                True          False
1287           7               False           True
590            2               False          False
1188           3                True          False
```

9. **<u>Model Selection and Evaluation:</u>**

Model Selection:

To effectively predict insurance claim amounts, we explored multiple supervised learning models to determine the best-performing approach. Given the continuous nature of the target variable, this problem is framed as a regression task. The models considered include:

- *Random Forest Regression* – An ensemble method that improves upon decision trees by reducing overfitting.
- *Gradient Boosting Machines (GBM)* – A powerful boosting technique known for high predictive accuracy.
- *Ridge Regression* – A regularized linear model to handle multicollinearity and improve generalization.
- *Support Vector Regression (SVR)* – A flexible model capable of handling non-linearity.
- *K-Nearest Neighbors (KNN) Regression* – A non-parametric model useful for capturing local relationships.

Each model was trained on the preprocessed dataset, and hyperparameter tuning was conducted using grid search and cross-validation.

Model Training and Hyperparameter Tuning:

To ensure optimal performance, we implemented hyperparameter tuning using GridSearchCV for models with hyperparameters requiring optimization. Below are the key tuning strategies for each model:

- *Random Forest Regression:* Optimized n_estimators and max_depth for better generalization.
- *Gradient Boosting Machines:* Tuned learning_rate and n_estimators for best performance.
- *Ridge Regression:* Adjusted alpha for regularization strength.
- *Support Vector Regression:* Tuned C, gamma, and kernel type.
- *K-Nearest Neighbors Regression:* Adjusted n_neighbors and weights for optimal performance.

Each model was trained using an 80-20 train-test split with 5-fold cross-validation to ensure stability in performance evaluation.

Model Evaluation Metrics:

To compare model performance, we used the following evaluation metrics:

- *Root Mean Squared Logarithmic Error (RMSLE):* A metric that reduces sensitivity to outliers and focuses on relative errors.
- *R-squared (R^2):* Measures how well the model explains the variance in the target variable.

Key Insights:

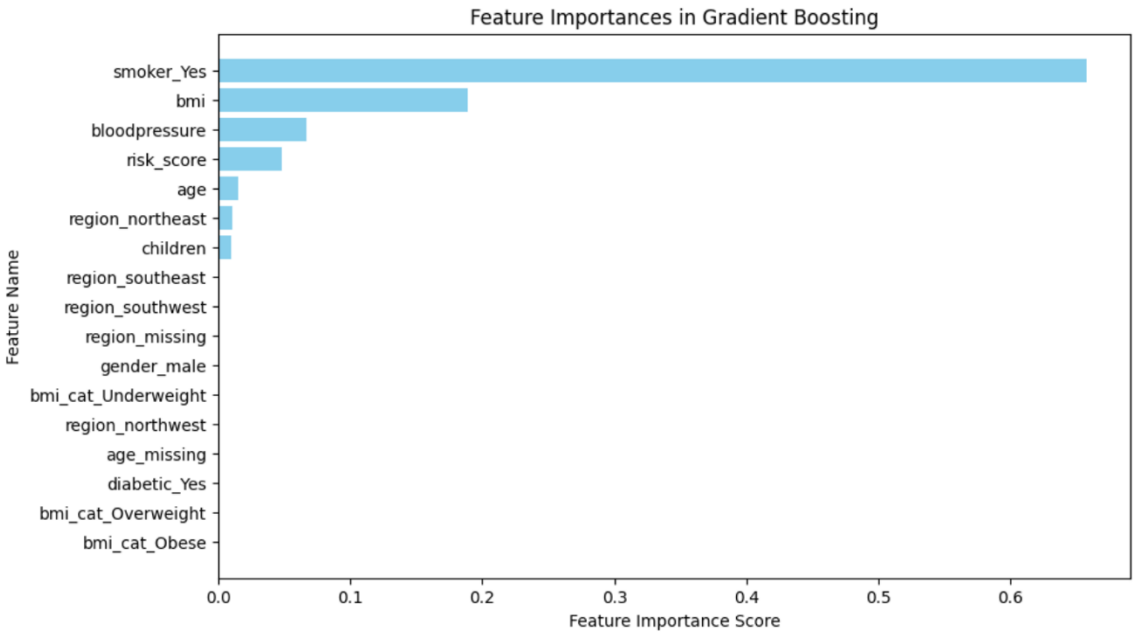
- *Gradient Boosting Machines* outperformed all models, making it the best-performing model.
- *Random Forest Regression* followed closely behind.
- *Ridge Regression* provided moderate performance but was outperformed by ensemble methods.

10. <u>Feature Importance Analysis:</u>

To understand which features contribute the most to prediction accuracy, we extracted feature importances from the best-performing Gradient Boosting Model**.** The most influential features were:

- *Smoker Status* – Major determinant of claim amounts.
- *BMI* – Strong predictor of insurance claims.
- *Blood Pressure* – Significant health risk indicator.
- *Risk Score* – Composite metric incorporating multiple health risks.
- *Age* – Moderate impact on claim prediction.

The feature importance plot visualized these rankings, confirming their relative influence on the model's decisions.



11. **<u>Final Model Selection and Justification:</u>**

    Based on the performance metrics, we selected Gradient Boosting Machines as the final model due to its:

    - *Superior predictive accuracy:* Lowest RMSLE and highest R^2.
    - *Robustness to overfitting:* Effective regularization techniques.
    - *Efficiency:* Strong balance between bias and variance.
    - *Flexibility:* Handles feature interactions and non-linearity effectively.

```
from sklearn.ensemble import  GradientBoostingRegressor, RandomForestRegressor
from sklearn.metrics import mean_squared_log_error
from sklearn.metrics import mean_squared_log_error, r2_score
from sklearn.linear_model import Ridge
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
import numpy as np

# Define Models
rf_model = RandomForestRegressor(random_state=42)
gb_model = GradientBoostingRegressor(random_state=42)
ridge_model = Ridge(random_state=42)
svr_model = SVR()
knn_model = KNeighborsRegressor()

# Train Models
rf_model.fit(X_train_encoded, y_train)
gb_model.fit(X_train_encoded, y_train)
ridge_model.fit(X_train_encoded, y_train)
svr_model.fit(X_train_encoded, y_train)
```

```
knn_model.fit(X_train_encoded, y_train)

# Evaluate Models
def rmsle(y_true, y_pred):
    return np.sqrt(mean_squared_log_error(y_true, y_pred))


# Root Mean Squared Logarithmic Error (RMSLE) is a good initial choice. It's
less sensitive to outliers and focuses on relative errors, which are often
more meaningful when dealing with a wide range of values.
print("Default Model Performance:")
print("Random Forest RMSLE:", rmsle(y_test, rf_model.predict(X_test_encoded)))
print("Gradient Boosting RMSLE:", rmsle(y_test,
gb_model.predict(X_test_encoded)))
print("Ridge Regression RMSLE:", rmsle(y_test,
abs(ridge_model.predict(X_test_encoded))))
print("SVR RMSLE:", rmsle(y_test, svr_model.predict(X_test_encoded)))
print("KNN RMSLE:", rmsle(y_test, knn_model.predict(X_test_encoded)))


from sklearn.model_selection import GridSearchCV

# Define hyperparameter grids
param_grids = {
    "RandomForest": {
        'n_estimators': [100, 200, 300],
        'max_depth': [None, 10, 20, 30]
    },
    "GradientBoosting": {
        'n_estimators': [100, 200, 300],
        'learning_rate': [0.001, 0.01, 0.1]
    },
    "RidgeRegression": {
        'alpha': [0.1, 1.0, 10.0, 100.0]
    },
    "SVR": {
        'C': [0.1, 1, 10],
        'gamma': ['scale', 'auto'],
        'kernel': ['rbf', 'linear']
    },
    "KNN": {
        'n_neighbors': [3, 5, 7, 9],
        'weights': ['uniform', 'distance']
    }
}

# Define models
models = {
    "RandomForest": rf_model,
```

```
    "GradientBoosting": gb_model,
    "RidgeRegression": ridge_model,
    "SVR": svr_model,
    "KNN": knn_model
}

# Perform Grid Search for each model
grid_search_results = {}

for model_name, model in models.items():
    grid_search = GridSearchCV(model, param_grids[model_name], cv=5,
scoring='neg_mean_squared_log_error')
    grid_search.fit(X_train_encoded, y_train)
    grid_search_results[model_name] = grid_search

    print(f"Best parameters for {model_name}: {grid_search.best_params_}")
    print(f"Best score for {model_name}: {grid_search.best_score_}\n")


# Define the tuned models dictionary
tuned_models = grid_search_results

# Rank models based on RMSLE (lower is better)
model_names = list(tuned_models.keys())
sorted_by_rmsle = sorted(
    model_names,
    key=lambda name: np.sqrt(mean_squared_log_error(y_test,
tuned_models[name].best_estimator_.predict(X_test_encoded)))
)
rmsle_ranks = {name: rank for rank, name in enumerate(sorted_by_rmsle,
start=1)}

# Rank models based on R² (higher is better)
sorted_by_r2 = sorted(
    model_names,
    key=lambda name: r2_score(y_test,
tuned_models[name].best_estimator_.predict(X_test_encoded)),
    reverse=True
)
r2_ranks = {name: rank for rank, name in enumerate(sorted_by_r2, start=1)}

# Combine the rankings (lower combined rank is better)
combined_ranks = {name: rmsle_ranks[name] + r2_ranks[name] for name in
model_names}

# Sort models by combined rank and select the top 3
top_3 = sorted(combined_ranks.items(), key=lambda x: x[1])[:3]

# Display top 3 models
```

```
print("\nTop 3 Models Based on Combined Rankings (Lower RMSLE & Higher R²):")
for model_name, total_rank in top_3:
    model = tuned_models[model_name]
    prediction = model.best_estimator_.predict(X_test_encoded)
    rmsle = np.sqrt(mean_squared_log_error(y_test,prediction ))
    r2 = r2_score(y_test, prediction)
    print("Model: {:<20} Combined Rank: {}  RMSLE: {:.4f}  R²: {:.4f}  Best
Params: {}".format(
        model_name, total_rank, rmsle, r2, model.best_params_
    ))
```

12. Business Insights:

Model Performance Summary:

The Gradient Boosting model demonstrated the best overall performance, achieving a Combined Rank of 2, an RMSLE of 0.5952, and an R^2 score of 0.8461. The optimal hyperparameters identified were learning_rate=0.1 and n_estimators=100, allowing the model to strike a balance between accuracy and generalizability.

The Random Forest model ranked second with a Combined Rank of 4, an RMSLE of 0.6070, and an R^2 score of 0.8241. The best-performing parameters were max_depth=10 and n_estimators=100, which provided a strong predictive performance while controlling complexity.

Ridge Regression, though less competitive, ranked third with a Combined Rank of 6, an RMSLE of 0.6543, and an R^2 score of 0.7237. The optimal tuning parameter alpha=10.0 contributed to improving its generalization capabilities.

Feature Importance Insights:

Analysis of feature importance indicated that smoker status had the most significant impact on insurance claim predictions, followed closely by BMI and blood pressure. Other factors, such as the risk score and age**,** also played minor yet noteworthy roles in shaping claim amounts.

The dominance of these features aligns with established medical and actuarial findings, where smoking, obesity, and high blood pressure are key risk indicators for high medical expenditures. The inclusion of a composite risk score further enhanced predictive capabilities by integrating multiple health-related factors into a single metric.

Future Improvements:

While Gradient Boosting proved to be the most effective model, there are several ways to further refine the prediction framework:

- *Hyperparameter Fine-Tuning:* Conduct more extensive tuning using Bayesian optimization or randomized search to identify even better parameter configurations.
- *Ensemble Approaches:* Explore stacking models by combining Gradient Boosting with Random Forest or Ridge Regression to further improve predictive robustness.

- *Feature Engineering:* Introduce additional health and behavioral data points, such as medical history and lifestyle factors, to improve model accuracy.
- *Explainability Enhancement:* Implement SHAP (Shapley Additive Explanations) to gain deeper insights into feature contributions at an individual level.

By incorporating these enhancements, we can continue to improve model accuracy, making insurance claim predictions even more reliable for risk assessment and financial planning in the insurance industry.