main

August 5, 2024

1 Health Insurance Claim Predictor

1.1 Introduction

1.1.1 Project Overview

This project aims to identify the health insurance claims of customers based on various features. The analysis is crucial for decreasing the risk fraudulent claims and increasing overall business profitability by optimizing insurance premium.

1.1.2 Background

The dataset used in this project is a collection of customer and their health information along with their insurance claim. This dataset provides insights into various factors that might influence insurance claim, including healthiness of the customer, their geographic location, and their wealthiness.

1.1.3 Objectives

- To segment customer profiles.
- To identify what factors greatly impact the insurance claim.
- To predict the insurance claim of customers based on various features.

1.1.4 Data Description

Dataset Overview The dataset used in this project is sourced from Kaggle's health insurance dataset. It includes 15000 customer records and 13 features. #### Key Features - age : Age of the policyholder - sex: Gender of policyholder - weight: Weight of the policyholder in kg - bmi: Body mass index of the policyholder - hereditary_diseases: A policyholder has a hereditary diseases or not (no-disease=0; disease=1) - no_of_dependents: Number of dependent persons of the policyholder - smoker: Indicates policyholder is a smoker or not (non-smoker=0;smoker=1) - bloodpressure: Blood pressure reading of policyholder - diabetes: Indicates whether policyholder has diabetes or not (non-diabetic=0; diabetic=1) - regular_ex: A policyholder regularly exercises or not (no-exercise=0; exercise=1) - job_title: Job title of the policyholder - city: The city in which the policyholder resides - claim: The amount claimed by the policyholder #### Data Types - Categorical: sex, hereditary_diseases, smoker, diabetes, regular_ex, job_title, city - Numerical: age, weight, bmi, no_of_dependents, bloodpressure, claim

1.1.5 Prediction

The goal is to predict the insurance claim of customers based on various features.

1.1.6 Metrics

RMSE: The square root of the average of the squared differences between predicted and actual values.

$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$

It provides a measure of how well the model's predictions match the actual values. Lower values indicate better accuracy.

1.1.7 References

1. Dataset Source: Kaggle Health Insurance Dataset

1.2 Exploratory Data Analysis

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
data = pd.read_csv('data.csv')
```

[3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15000 entries, 0 to 14999
Data columns (total 13 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   age                  14604 non-null  float64
 1   sex                  15000 non-null  object
 2   weight               15000 non-null  int64
 3   bmi                  14044 non-null  float64
 4   hereditary_diseases  15000 non-null  object
 5   no_of_dependents     15000 non-null  int64
 6   smoker               15000 non-null  int64
 7   city                 15000 non-null  object
 8   bloodpressure        15000 non-null  int64
 9   diabetes             15000 non-null  int64
 10  regular_ex           15000 non-null  int64
 11  job_title            15000 non-null  object
 12  claim                15000 non-null  float64
```

```
dtypes: float64(3), int64(6), object(4)
memory usage: 1.5+ MB
```

[4]:
```python
data.describe()
```

[4]:
```
                age        weight           bmi  no_of_dependents  \
count  14604.000000  15000.000000  14044.000000      15000.000000
mean      39.547521     64.909600     30.266413          1.129733
std       14.015966     13.701935      6.122950          1.228469
min       18.000000     34.000000     16.000000          0.000000
25%       27.000000     54.000000     25.700000          0.000000
50%       40.000000     63.000000     29.400000          1.000000
75%       52.000000     76.000000     34.400000          2.000000
max       64.000000     95.000000     53.100000          5.000000

             smoker  bloodpressure      diabetes    regular_ex         claim
count  15000.000000   15000.000000  15000.000000  15000.000000  15000.000000
mean       0.198133      68.650133      0.777000      0.224133  13401.437620
std        0.398606      19.418515      0.416272      0.417024  12148.239619
min        0.000000       0.000000      0.000000      0.000000   1121.900000
25%        0.000000      64.000000      1.000000      0.000000   4846.900000
50%        0.000000      71.000000      1.000000      0.000000   9545.650000
75%        0.000000      80.000000      1.000000      0.000000  16519.125000
max        1.000000     122.000000      1.000000      1.000000  63770.400000
```

[5]:
```python
# imputer
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')

# impute age and bmi
imputer = imputer.fit(data[['age', 'bmi']])
data[['age', 'bmi']] = imputer.transform(data[['age', 'bmi']])
```

[6]:
```python
# winsorizer
from feature_engine.outliers import Winsorizer
winsorizer = Winsorizer(capping_method='iqr', tail='both', fold=1.5,␣
↪variables=['bloodpressure', 'claim'])

data[['bloodpressure', 'claim']] = winsorizer.
↪fit_transform(data[['bloodpressure', 'claim']])
```

[7]:
```python
data.describe()
```

[7]:
```
                age        weight           bmi  no_of_dependents  \
count  15000.000000  15000.000000  15000.000000      15000.000000
mean      39.547521     64.909600     30.266413          1.129733
std       13.829705     13.701935      5.924606          1.228469
```

```
min        18.000000     34.000000     16.000000          0.000000
25%        27.000000     54.000000     25.900000          0.000000
50%        40.000000     63.000000     29.800000          1.000000
75%        51.000000     76.000000     34.100000          2.000000
max        64.000000     95.000000     53.100000          5.000000

              smoker  bloodpressure      diabetes    regular_ex          claim
count   15000.000000   15000.000000  15000.000000  15000.000000   15000.000000
mean        0.198133      70.600800      0.777000      0.224133   12543.004248
std         0.398606      13.103514      0.416272      0.417024   10073.193516
min         0.000000      40.000000      0.000000      0.000000    1121.900000
25%         0.000000      64.000000      1.000000      0.000000    4846.900000
50%         0.000000      71.000000      1.000000      0.000000    9545.650000
75%         0.000000      80.000000      1.000000      0.000000   16519.125000
max         1.000000     104.000000      1.000000      1.000000   34027.462500
```

[8]:
```python
import sweetviz as sv
```

```
/home/hpark/Syncthing/Professional/DS_Projects/Health_Insurance_Claim/.venv/lib/
python3.10/site-packages/tqdm/auto.py:21: TqdmWarning: IProgress not found.
Please update jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

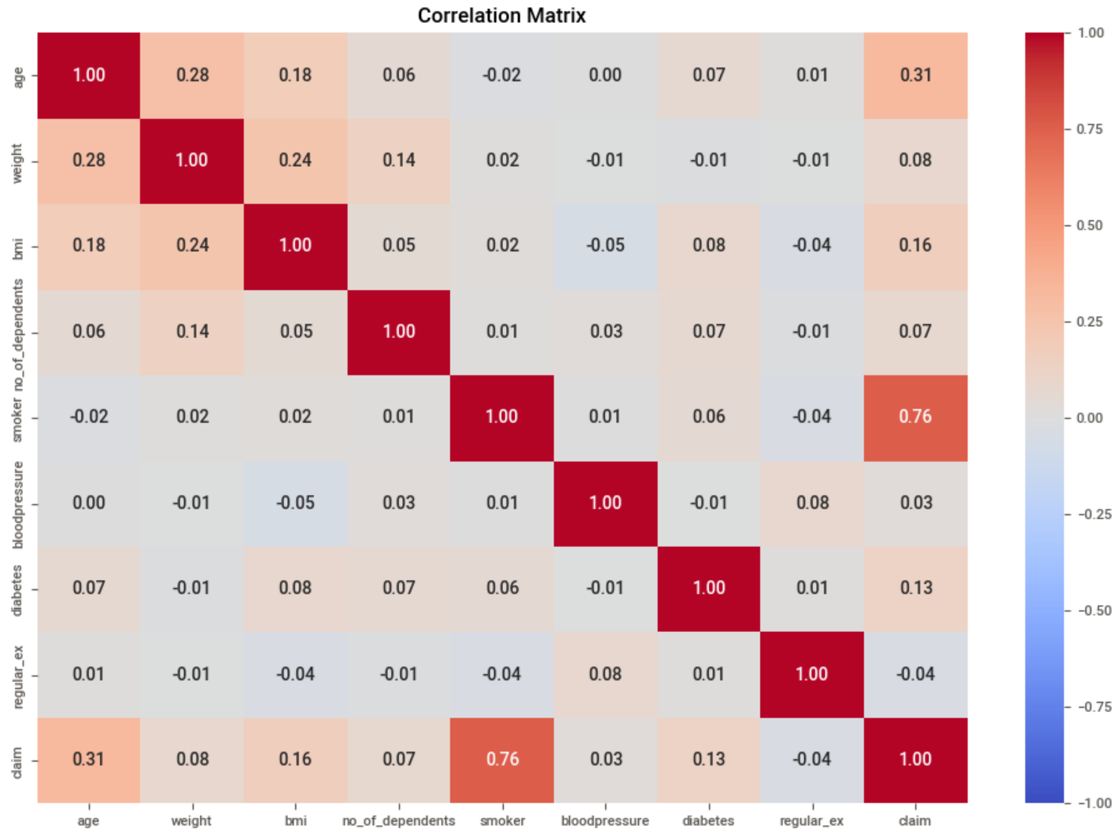
[9]:
```python
report = sv.analyze(data)

report.show_notebook()
```

```
Done! Use 'show' commands to display/save.   |      | [100%]   00:00 ->
(00:00 left)
```

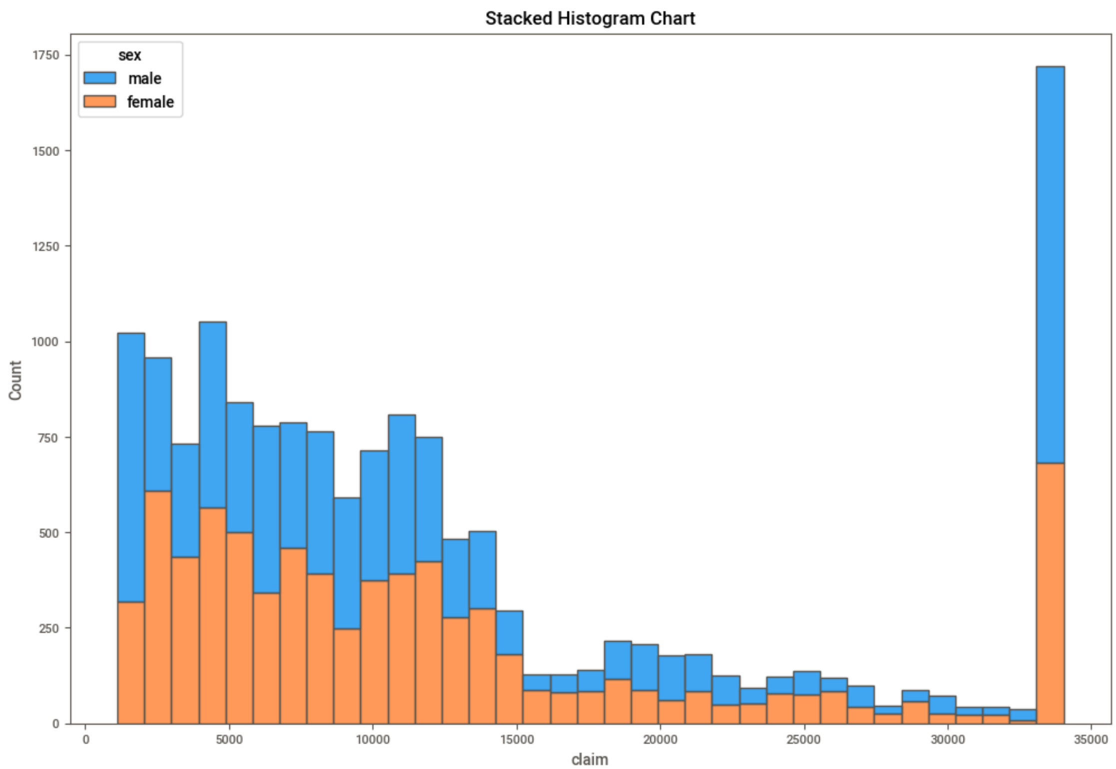
```
<IPython.core.display.HTML object>
```

[10]:
```python
numerical_columns = ['age', 'weight', 'bmi', 'no_of_dependents', 'smoker',
 ↪'bloodpressure', 'diabetes', 'regular_ex', 'claim']
categorical_columns = ['sex', 'hereditary_diseases', 'city', 'job_title']
```

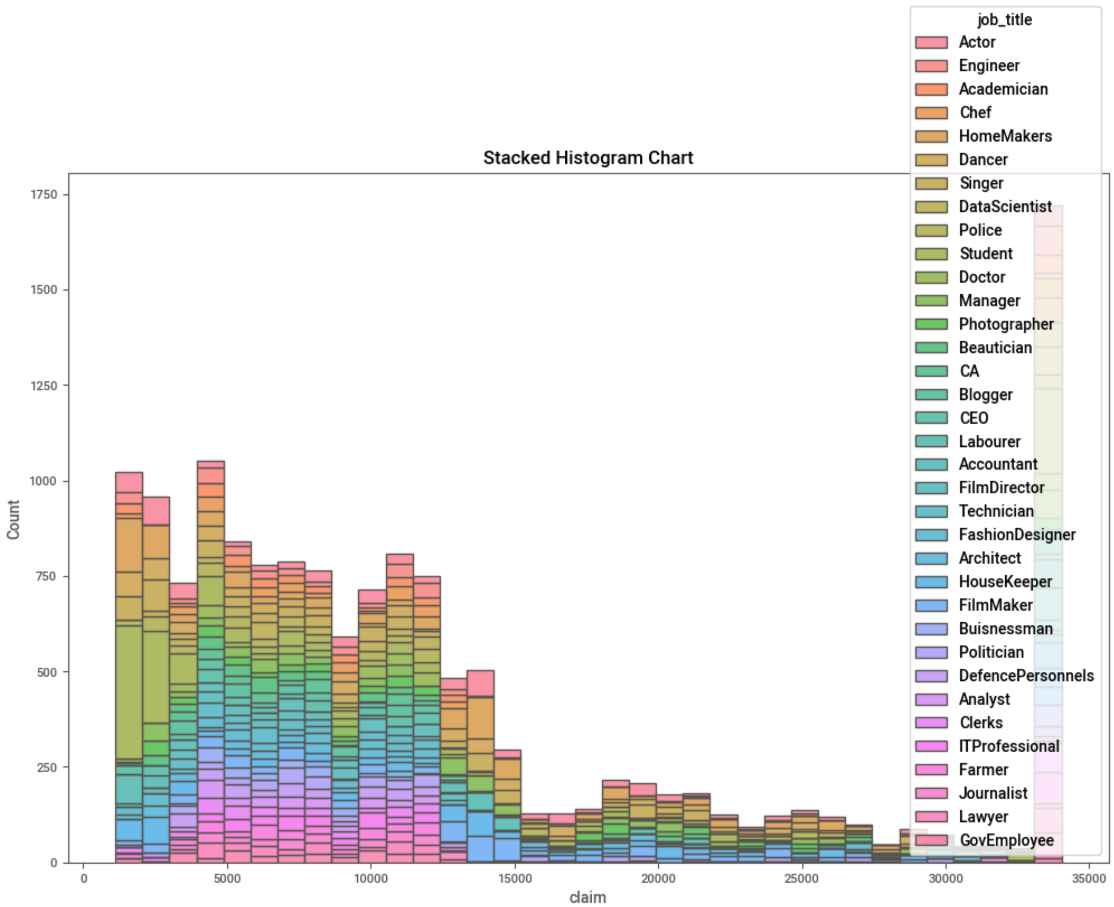
[11]:
```python
# Compute the correlation matrix
corr_matrix = data[numerical_columns].corr()

# Plot the correlation matrix
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', vmin=-1,
 ↪vmax=1)
plt.title('Correlation Matrix')
plt.show()
```



[12]:
```python
# stacked histogram chart
plt.figure(figsize=(12, 8))
sns.histplot(data=data, x='claim', hue='sex', multiple='stack')
plt.title('Stacked Histogram Chart')
plt.show()
```



[13]:
```
# stacked histogram chart
plt.figure(figsize=(12, 8))
sns.histplot(data=data, x='claim', hue='job_title', multiple='stack')
plt.title('Stacked Histogram Chart')
plt.show()
```



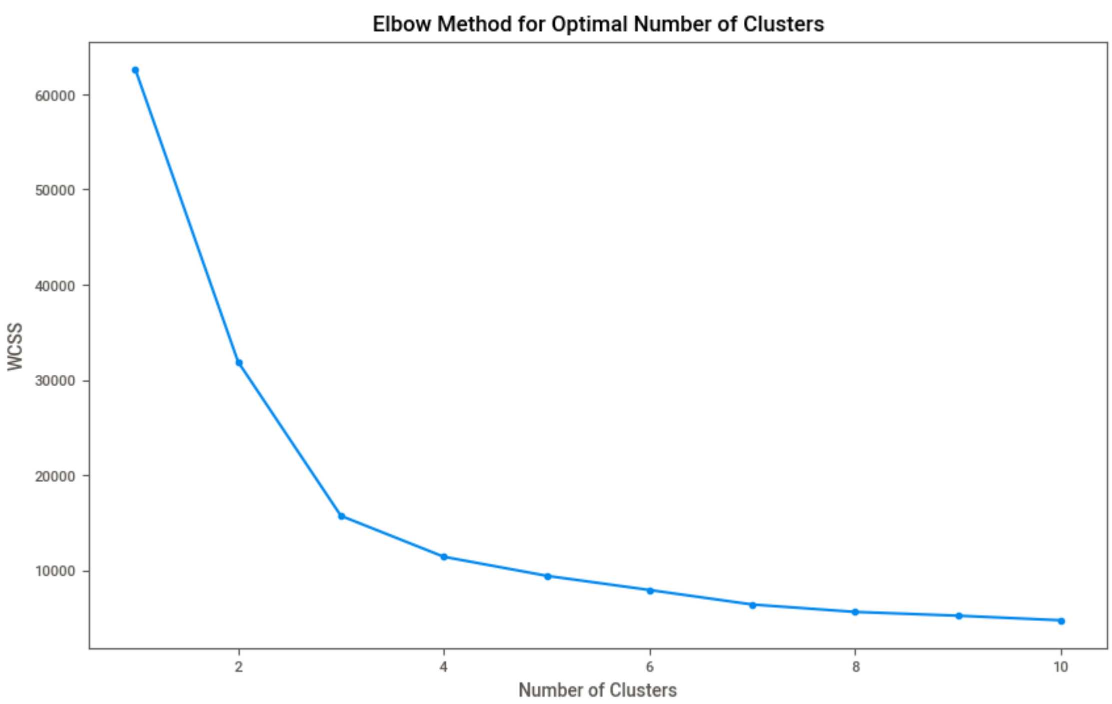
1.2.1 Cluster Analysis

[14]:
```
# cluster analysis using PCA
from sklearn.preprocessing import StandardScaler

# One-hot encode categorical features
data_encoded = pd.get_dummies(data, drop_first=True)

features = data_encoded.drop('claim', axis=1)
target = data_encoded['claim']

# Standardize features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)
```

[15]:
```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)  # Use 2 components for 2D visualization
```

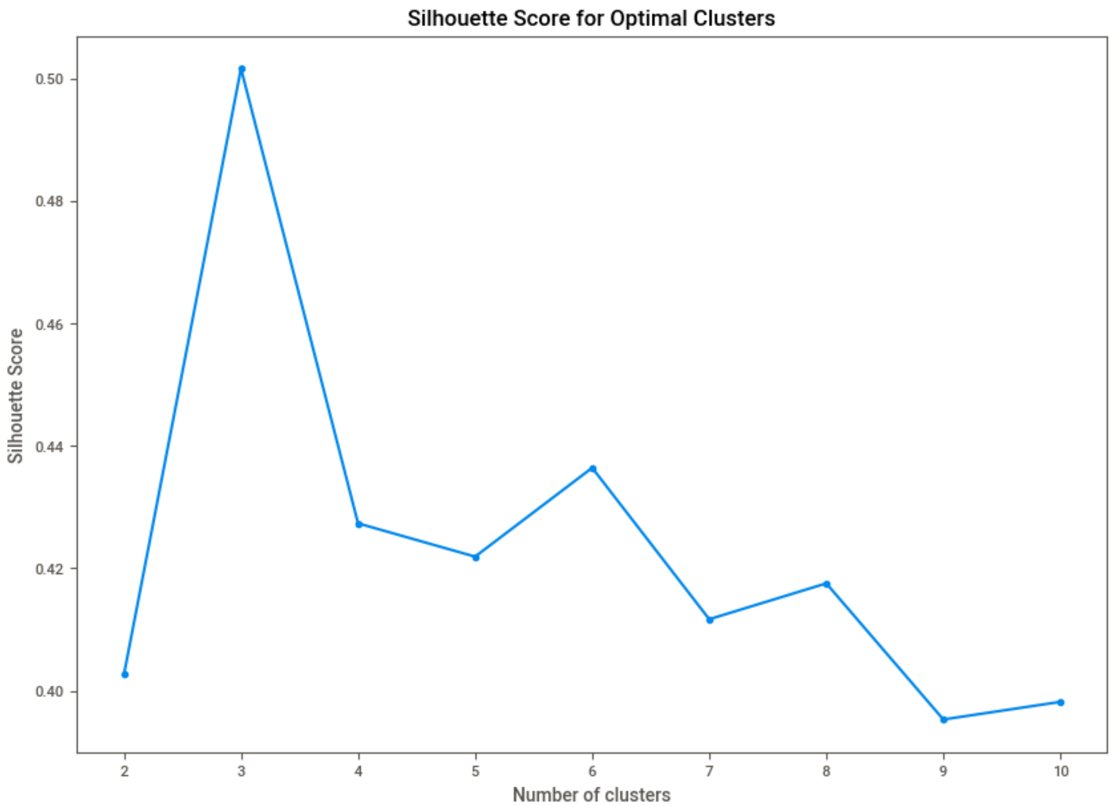
```
pca_result = pca.fit_transform(scaled_features)
```

[16]:
```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

wcss = []

# Testing from 1 to 10 clusters
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, random_state=42)
    kmeans.fit(pca_result)
    wcss.append(kmeans.inertia_)

plt.figure(figsize=(10, 6))
plt.plot(range(1, 11), wcss, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.title('Elbow Method for Optimal Number of Clusters')
plt.show()
```



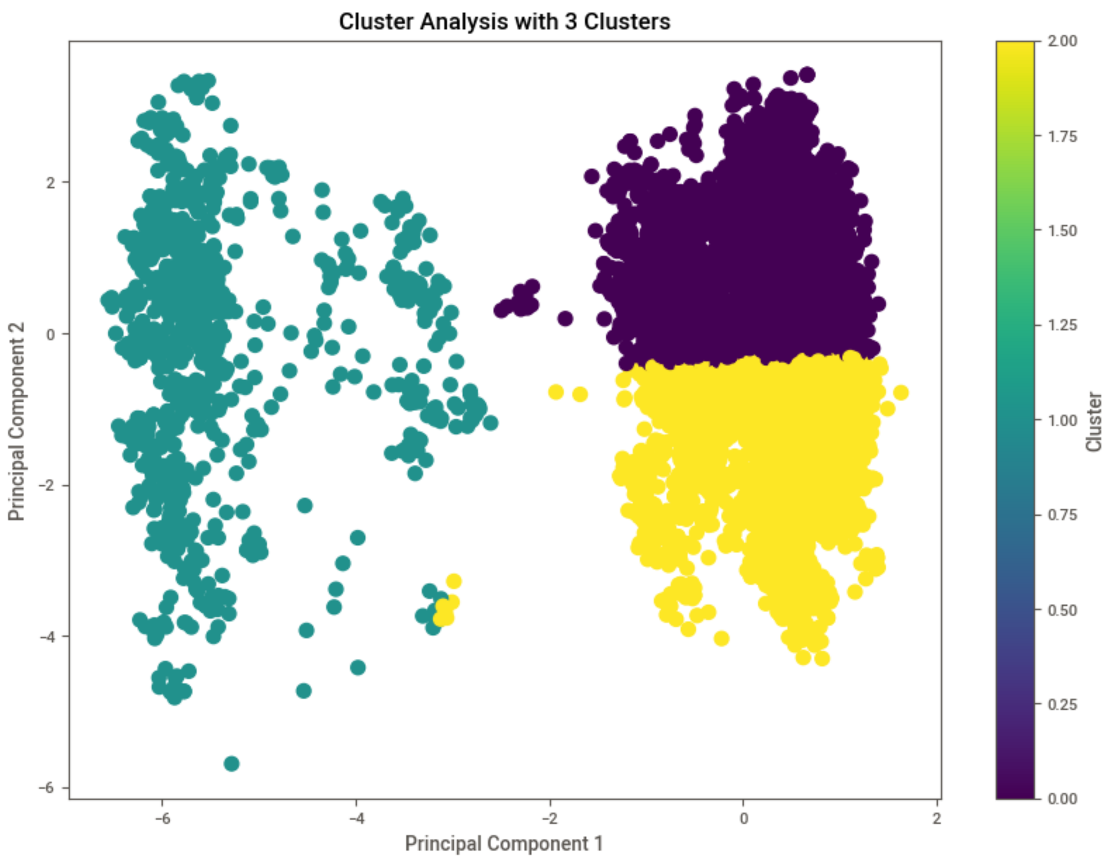
[17]:
```
from sklearn.metrics import silhouette_score

# Calculate Silhouette Scores for a range of cluster numbers
silhouette_scores = []
```

```
for n_clusters in range(2, 11):  # At least 2 clusters needed for silhouette␣
↪score
    kmeans = KMeans(n_clusters=n_clusters, random_state=0)
    clusters = kmeans.fit_predict(pca_result)
    silhouette_avg = silhouette_score(pca_result, clusters)
    silhouette_scores.append(silhouette_avg)

# Plot the Silhouette Scores
plt.figure(figsize=(10, 7))
plt.plot(range(2, 11), silhouette_scores, marker='o')
plt.title('Silhouette Score for Optimal Clusters')
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Score')
plt.show()
```

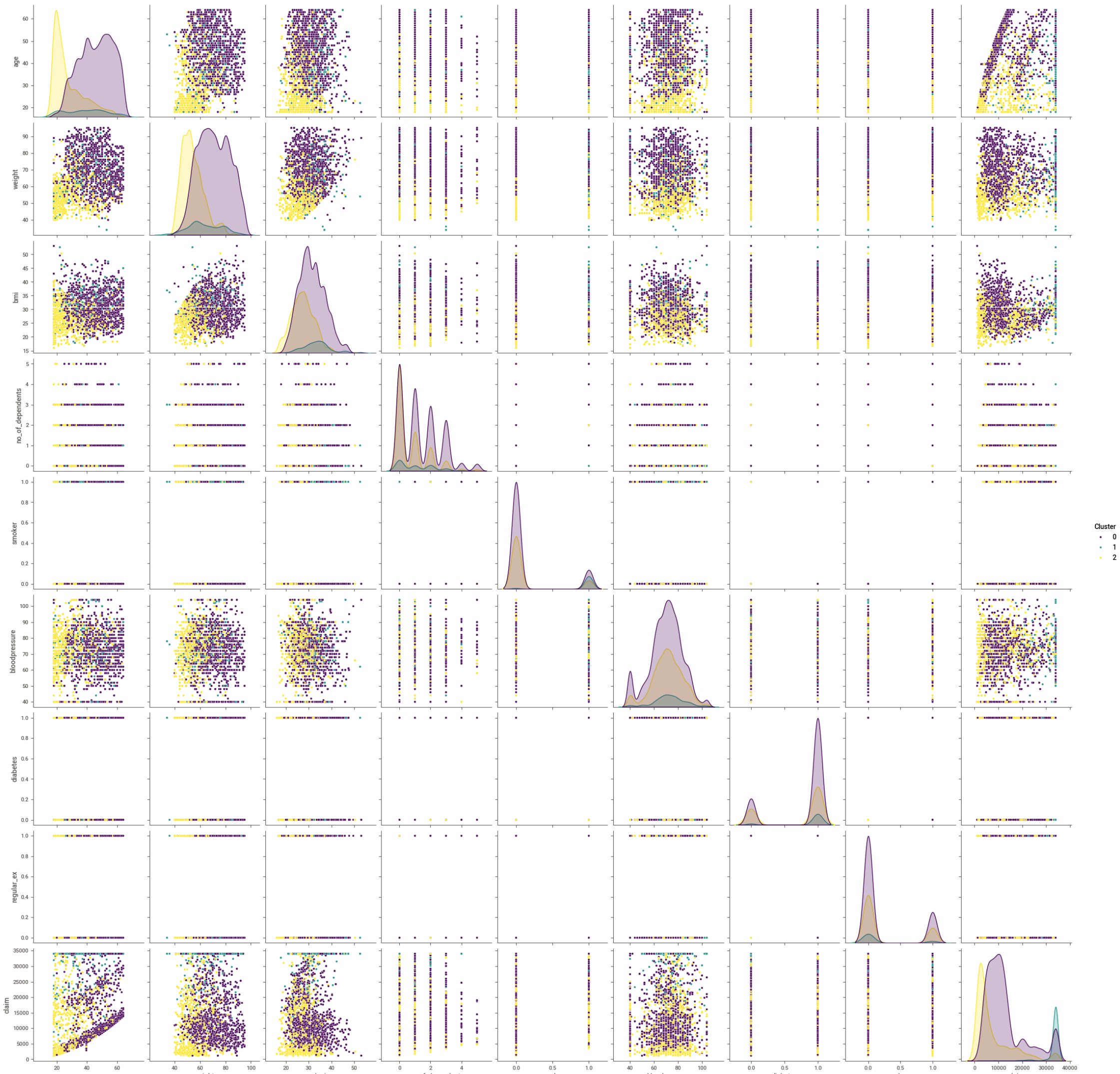


[18]:
```
optimal_clusters = 3  # Replace with the number you determined from the elbow␣
↪plot
kmeans = KMeans(n_clusters=optimal_clusters, random_state=42)
clusters = kmeans.fit_predict(pca_result)
```

[19]:
```
plt.figure(figsize=(10, 7))
plt.scatter(pca_result[:, 0], pca_result[:, 1], c=clusters, cmap='viridis',␣
↪s=50)
plt.colorbar(label='Cluster')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title(f'Cluster Analysis with {optimal_clusters} Clusters')
plt.show()
```



[20]:
```
# Add cluster labels to DataFrame
data['Cluster'] = clusters

# Plot pairwise relationships
sns.pairplot(data, hue='Cluster', palette='viridis')
plt.show()
```

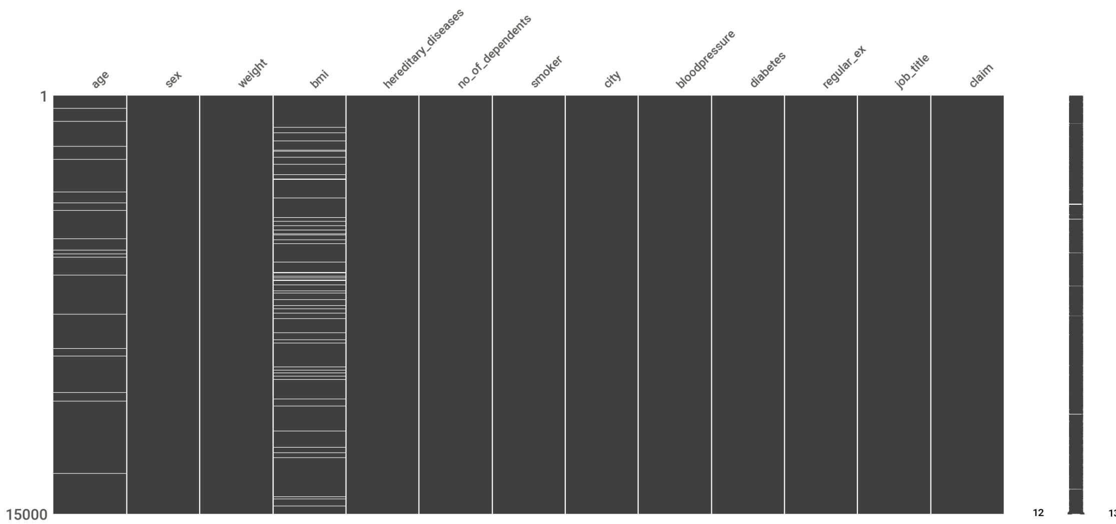


1.3 Preprocessing

[21]:
```
df = pd.read_csv('data.csv')
```

[22]:
```
# missing values
import missingno as msno

msno.matrix(df)
```

[22]: <Axes: >



[23]:
```
# count missing values
df.isna().sum()
```

[23]:
```
age                    396
sex                      0
weight                   0
bmi                    956
hereditary_diseases      0
no_of_dependents         0
smoker                   0
city                     0
bloodpressure            0
diabetes                 0
regular_ex               0
job_title                0
claim                    0
dtype: int64
```

[24]:
```
# drop claim column
numerical_columns.remove('claim')
```

[25]:
```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import KNNImputer
from sklearn.preprocessing import OneHotEncoder
from feature_engine.outliers import Winsorizer
from sklearn.preprocessing import FunctionTransformer

log_transform_columns = []
```

```
scale_columns = ['bloodpressure']

def log_transform(x):
    return np.log1p(x)

# Preprocessing for numerical data
numerical_transformer = Pipeline(steps=[
    ('imputer', KNNImputer(n_neighbors=5)),
        # ('winsorizer', Winsorizer(capping_method='iqr', tail='both', fold=1.
 ↪5, variables=list(numerical_columns)))
])
# Preprocessing for categorical data
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# Bundle preprocessing for numerical and categorical data
preprocessor = ColumnTransformer(
    transformers=[
        ('log', FunctionTransformer(log_transform), log_transform_columns),
        ('scale', StandardScaler(), scale_columns),
        ('num', numerical_transformer, numerical_columns),
        ('cat', categorical_transformer, categorical_columns)
    ],
    remainder='passthrough')
```

1.4 Model

[26]:
```
X = df.drop('claim', axis=1)
y = np.log1p(df['claim'])
```

[27]:
```
from sklearn.model_selection import train_test_split

# Assuming X is your feature matrix and y is your target variable (labels)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[28]:
```
# models
# linear, ridge, lasso, polynomial, dt, rf, svr, catboost
```

[29]:
```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.preprocessing import PolynomialFeatures
```

```python
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from catboost import CatBoostRegressor
```

[30]:
```python
models = [
    LinearRegression(),
    Ridge(),
    Lasso(),
    DecisionTreeRegressor(random_state=42),
    RandomForestRegressor(random_state=42),
    SVR(),
    CatBoostRegressor(random_seed=42, logging_level='Silent')
]
```

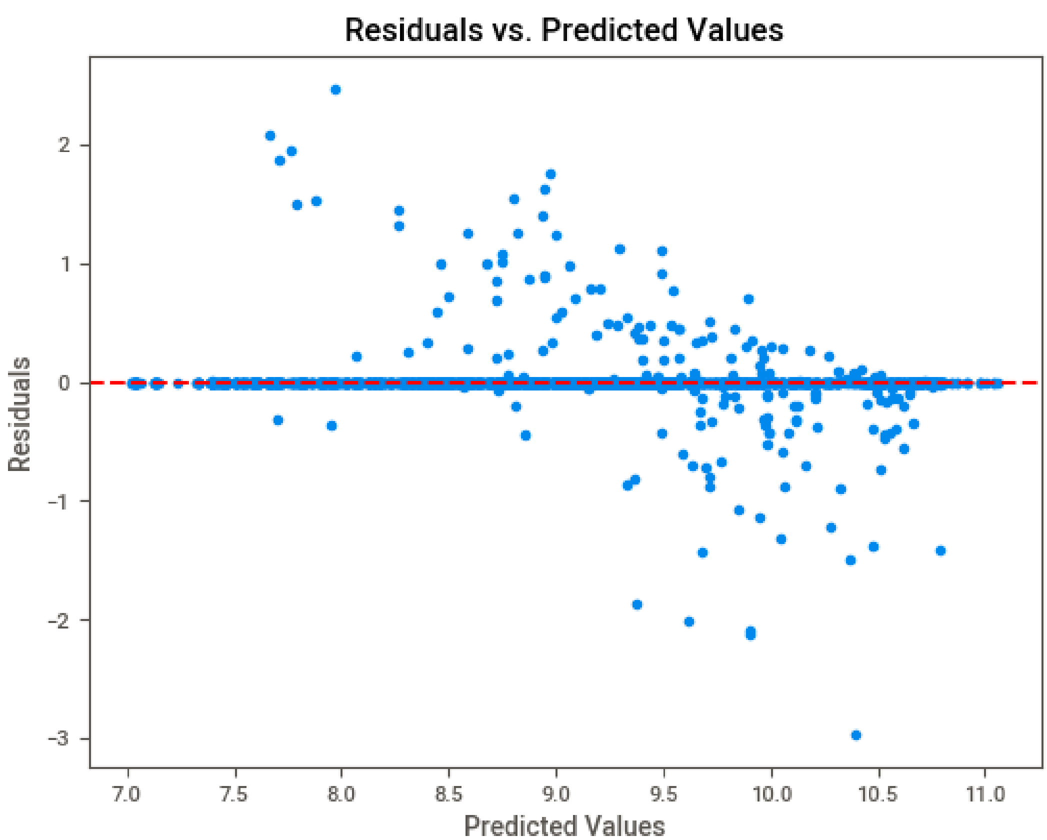
[31]:
```python
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_predict
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error

for model in models:
    pipeline = Pipeline(steps=[('preprocessor', preprocessor), ('model',␣
 ↪model)])
    kfold = KFold(n_splits=5, shuffle=True, random_state=42)
    y_pred = cross_val_predict(pipeline, X_train, y_train, cv=kfold)

    y_pred_exp1m = np.expm1(y_pred)
    y_train_exp1m = np.expm1(y_train)

    print(f"{model.__class__.__name__}: RMSE = {np.
 ↪sqrt(mean_squared_error(y_train_exp1m, y_pred_exp1m)):.2f}")
```

```
LinearRegression: RMSE = 8288.09
Ridge: RMSE = 8286.63
Lasso: RMSE = 12123.63
DecisionTreeRegressor: RMSE = 2840.08
RandomForestRegressor: RMSE = 2419.83
SVR: RMSE = 11799.18
CatBoostRegressor: RMSE = 3073.95
```

[32]:
```python
best_models = [
    DecisionTreeRegressor(random_state=42),
    RandomForestRegressor(random_state=42),
    CatBoostRegressor(random_seed=42, logging_level='Silent')
]
```

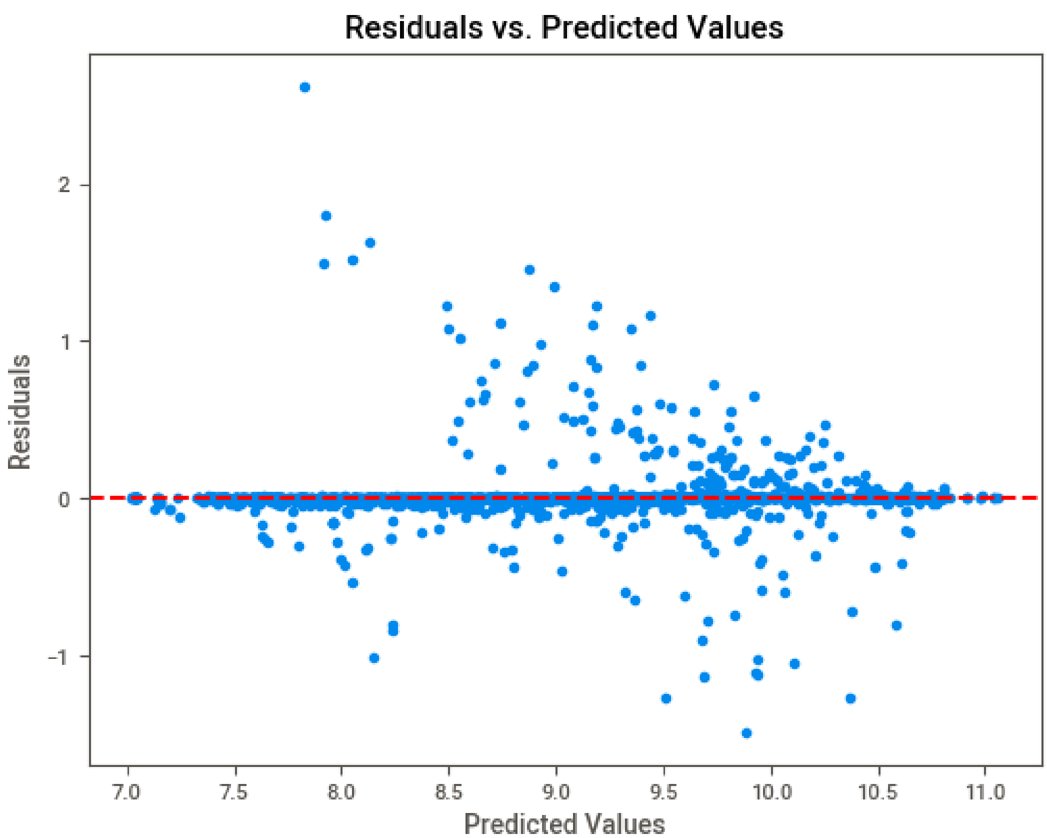
1.4.1 Error Analysis

[33]:
```
pipeline_dt = Pipeline(steps=[('preprocessor', preprocessor), ('model',␣
↪best_models[0])])

pipeline_dt.fit(X_train, y_train)

y_pred_dt = pipeline_dt.predict(X_test)

residuals = y_test - y_pred_dt

plt.scatter(y_pred_dt, residuals)
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Residuals vs. Predicted Values')
plt.show()
```



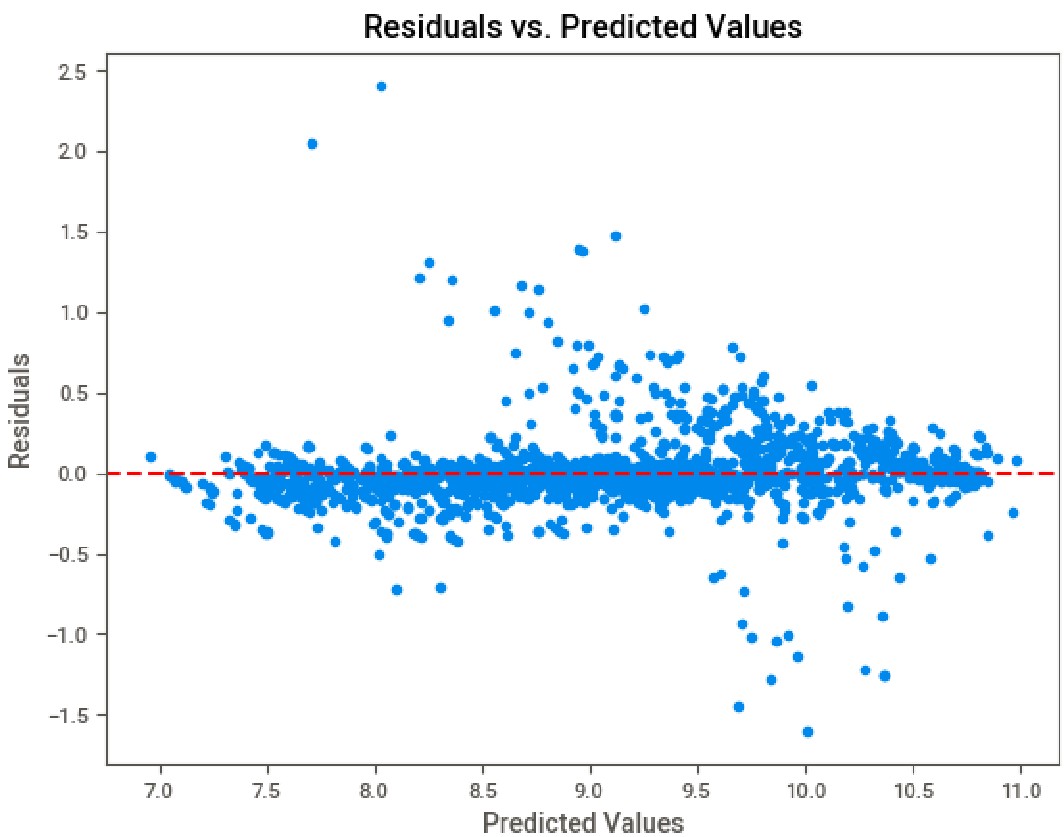
[34]:
```
pipeline_rf = Pipeline(steps=[('preprocessor', preprocessor), ('model',␣
↪best_models[1])])
```

```
pipeline_rf.fit(X_train, y_train)

y_pred_rf = pipeline_rf.predict(X_test)

residuals = y_test - y_pred_rf

plt.scatter(y_pred_rf, residuals)
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Residuals vs. Predicted Values')
plt.show()
```



[35]:
```
pipeline_cb = Pipeline(steps=[('preprocessor', preprocessor), ('model',␣
↪best_models[2])])

pipeline_cb.fit(X_train, y_train)

y_pred_cb = pipeline_cb.predict(X_test)
```

```
residuals = y_test - y_pred_cb

plt.scatter(y_pred_cb, residuals)
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Residuals vs. Predicted Values')
plt.show()
```



1.4.2 Ensemble Methods

[36]:
```
# bagging
from sklearn.ensemble import BaggingRegressor

for model in best_models:
    # Initialize the base model
    base_model = model

    # Initialize the bagging model
```

```
    bagging_model = BaggingRegressor(base_model, n_estimators=3,␣
↪random_state=42)

    pipeline = Pipeline(steps=[('preprocessor', preprocessor), ('model',␣
↪bagging_model)])
    kfold = KFold(n_splits=5, shuffle=True, random_state=42)
    y_pred = cross_val_predict(pipeline, X_train, y_train, cv=kfold)

    y_pred_exp1m = np.expm1(y_pred)
    y_train_exp1m = np.expm1(y_train)

    print(f"{model.__class__.__name__}: RMSE = {np.
↪sqrt(mean_squared_error(y_train_exp1m, y_pred_exp1m)):.2f}")
```

```
DecisionTreeRegressor: RMSE = 2715.16
RandomForestRegressor: RMSE = 2759.49
CatBoostRegressor: RMSE = 3299.14
```

[37]:

```
# stacking
from mlxtend.regressor import StackingRegressor

# Initialize the base models
model0 = best_models[0]
model1 = best_models[1]
model2 = best_models[2]

# Initialize the meta model
meta_model = LinearRegression()

# Initialize the stacking model
stacking_model = StackingRegressor(regressors=[model1, model2],␣
↪meta_regressor=meta_model)

pipeline = Pipeline(steps=[('preprocessor', preprocessor), ('model',␣
↪stacking_model)])
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
y_pred = cross_val_predict(pipeline, X_train, y_train, cv=kfold)

y_pred_exp1m = np.expm1(y_pred)
y_train_exp1m = np.expm1(y_train)

print(f"Stacking: RMSE = {np.sqrt(mean_squared_error(y_train_exp1m,␣
↪y_pred_exp1m)):.2f}")
```

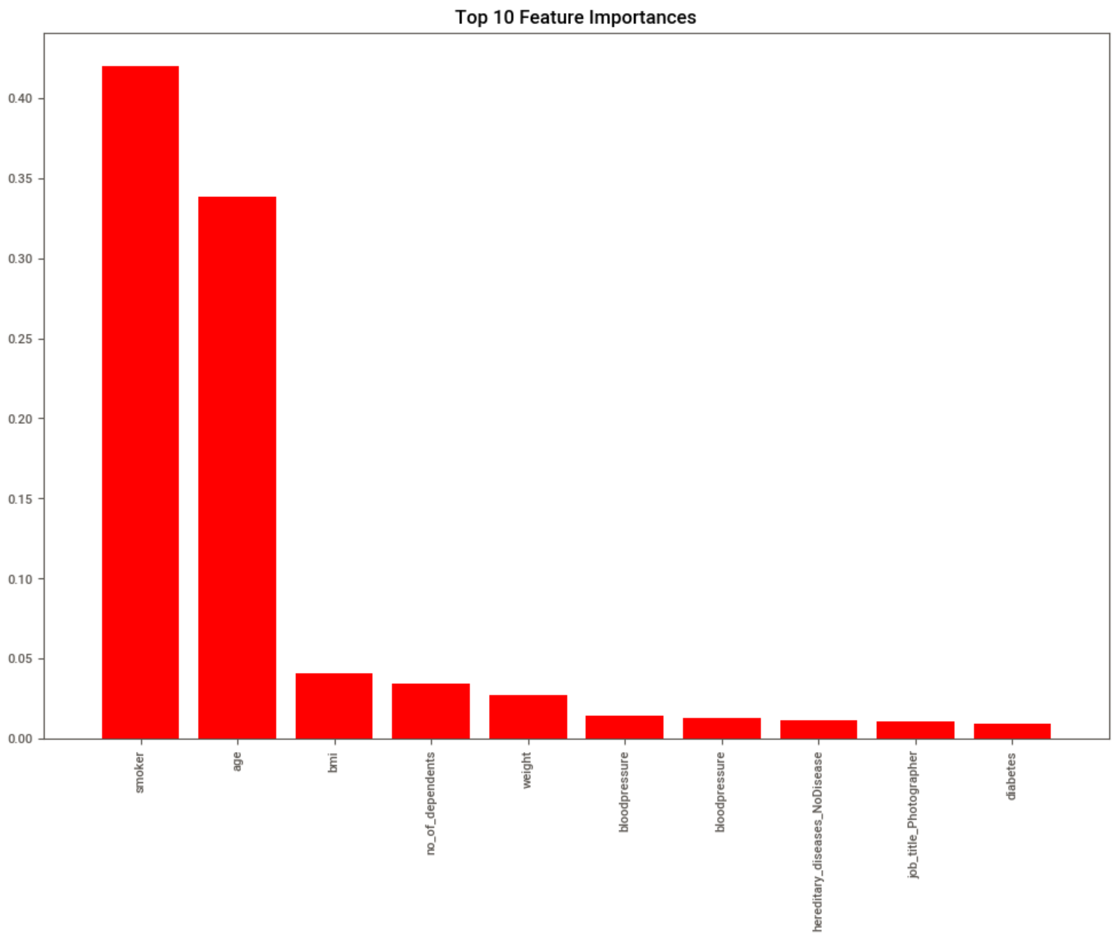
```
Stacking: RMSE = 2488.24
```

1.4.3 Feature Importance

[38]:
```
best_model = best_models[1]

pipeline = Pipeline(steps=[('preprocessor', preprocessor), ('model',␣
↪best_model)])

p = pipeline.fit(X_train, y_train)
```

[39]:
```
fitted_model = p.named_steps['model']

# Ensure the model has feature_importances_ attribute
if hasattr(fitted_model, 'feature_importances_'):
    importances = fitted_model.feature_importances_
    indices = np.argsort(importances)[::-1]

    # Check feature names from the transformed feature set
    if hasattr(pipeline.named_steps['preprocessor'], 'transformers_'):
        preprocessor = pipeline.named_steps['preprocessor']
        feature_names = []
        for name, transformer, columns in preprocessor.transformers_:
            if hasattr(transformer, 'get_feature_names_out'):
                feature_names.extend(transformer.get_feature_names_out(columns))
            else:
                feature_names.extend(columns)

        # Handle mismatch in feature names and importances
        if len(feature_names) != len(importances):
            print("Mismatch between number of features and importances.")
            feature_names = feature_names[:len(importances)]  # Adjust if needed

        # Select top 10 features
        top_n = 10
        top_indices = indices[:top_n]
        top_importances = importances[top_indices]
        top_feature_names = np.array(feature_names)[top_indices]

        # Plot the top 10 feature importances
        plt.figure(figsize=(12, 8))
        plt.title(f"Top {top_n} Feature Importances")
        plt.bar(range(top_n), top_importances, color="r", align="center")
        plt.xticks(range(top_n), top_feature_names, rotation=90)
        plt.xlim([-1, top_n])
        plt.show()
    else:
        print("Preprocessor does not have 'transformers_' attribute.")
else:
```

```
    raise AttributeError("The fitted model does not have 'feature_importances_'␣
↪attribute.")
```



1.5 Conclusion

1.5.1 Summary of Findings

- Identified there exists 3 clusters in the data
- Identified smoking and customer age as the biggest factors contributing to claim amount

1.5.2 Implications

It was found that despite common sense, there is no significant impact to claim amount from most of the features. Therefore, using this information, it may be better to drop the insurance premium for customers who are younger and who do not smoke for a higher profit. Additionally, since there are 3 significant clusters in the data, it may be better to market the insurance product separately for each cluster.

1.5.3 Limitations

Data is lacking a time series feature. Therefore, it may not capture seasonality or long term trends in the data if there exists one.

1.5.4 Future Work

Further analysis could explore how often and how much a customer claims. This would be useful to calculate life time value of the customer and therefore charging the optimal insurance premium. Expanding the dataset to include more diverse samples could also enhance the model's generalizability.

1.5.5 Final Thoughts

This project has enhanced the understanding of health insurance claim. Key insights include the importance of smoking and age as the only signing factors. The analysis demonstrates how effective model selection can drive actionable business strategies.