Predicting Medical Insurance Charges using Linear Regression and Random Forest

BY GRISHMA VEMIREDDY

Introduction

Problem statement:

Predict a person’s medical insurance charges based on their demographic data and lifestyle.

Purpose:

It supports risk assessment and customer profiling.

Helps understand how insurance companies estimate premiums.

Dataset Overview

```
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1338 non-null   int64
 1   sex       1338 non-null   object
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64
 4   smoker    1338 non-null   object
 5   region    1338 non-null   object
 6   charges   1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

Source: Kaggle (Medical Cost Personal Datasets)

Data description:

1,338 total observation

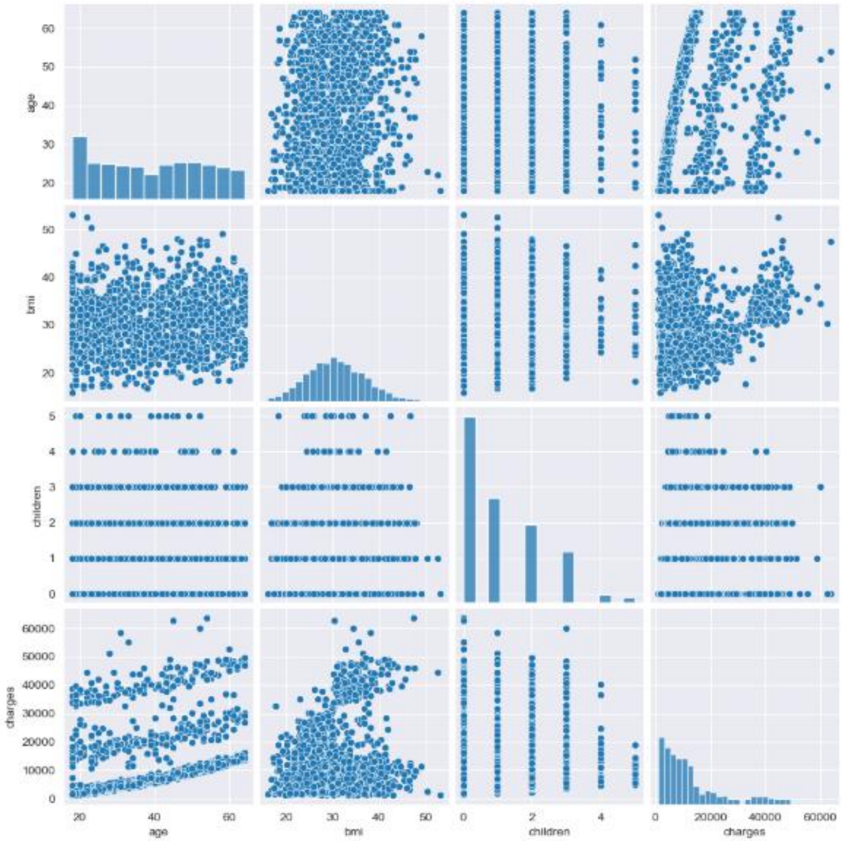
No missing values

7 features:

age, sex, bmi, children, smoker, region, charges (target variable).

Methodology

First performed data cleaning and data exploration, scaled the data, then split data to train and test models.




```
#conver categorical to numerical
df_encoded = data_df.copy()
df_encoded['sex'] = df_encoded['sex'].map({'male': 0, 'female': 1})
df_encoded['smoker'] = df_encoded['smoker'].map({'no': 0, 'yes': 1})
df_encoded['region'] = df_encoded['region'].map({'southwest': 0, 'southeast'
```

```
sns.heatmap(df_encoded.corr(), annot=True)
plt.show()
```


```
# scale data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

#Splt to train and test 80-20 split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0
```

Methodology

Two regression models:

1. Linear regression

Implemented a basic model to predict charges.

```
#Linear regression
lm = LinearRegression()
lm.fit(X_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

2. Random Forest

I initially passed in parameters of

n_estimaors=100

max_depth=10

```
# Random Forest
rfc = RandomForestRegressor(n_estimators=100,max_depth=10,random_state=42)
rfc.fit(X_train, y_train)
```

```
▾ RandomForestRegressor
RandomForestRegressor(max_depth=10, random_state=42)
```

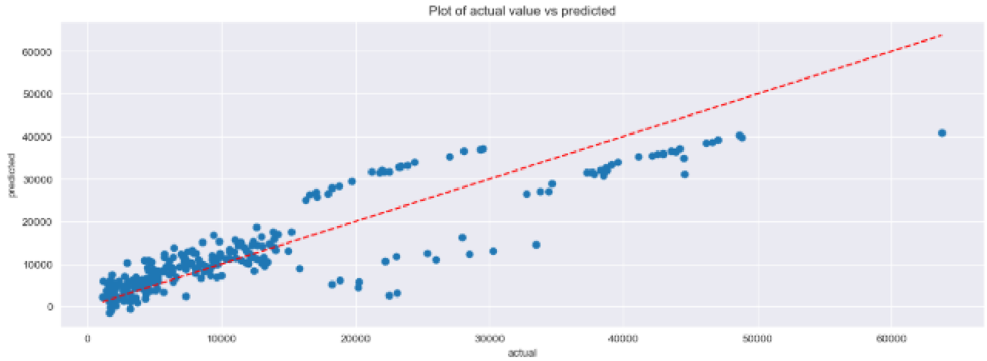
Then tuned and tested the model with different parameter values to see any performance improvement

n_estimaors=300

max_depth=20

Results and Visualization

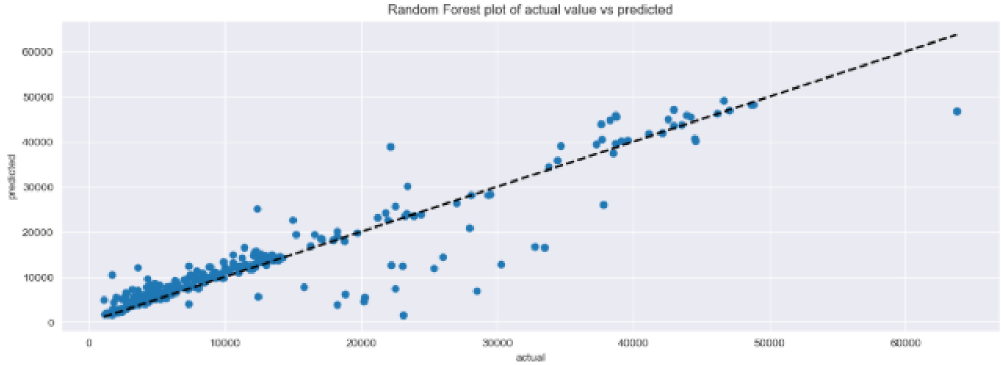
Linear Regression




```
# RMSE and R^2
print('RMSE:', np.sqrt(mean_squared_error(y_test, predictions)))
print('R2 Score: ', r2_score(y_test, predictions))
```

```
RMSE: 5799.5870914383595
R2 Score:  0.7833463107364536
```

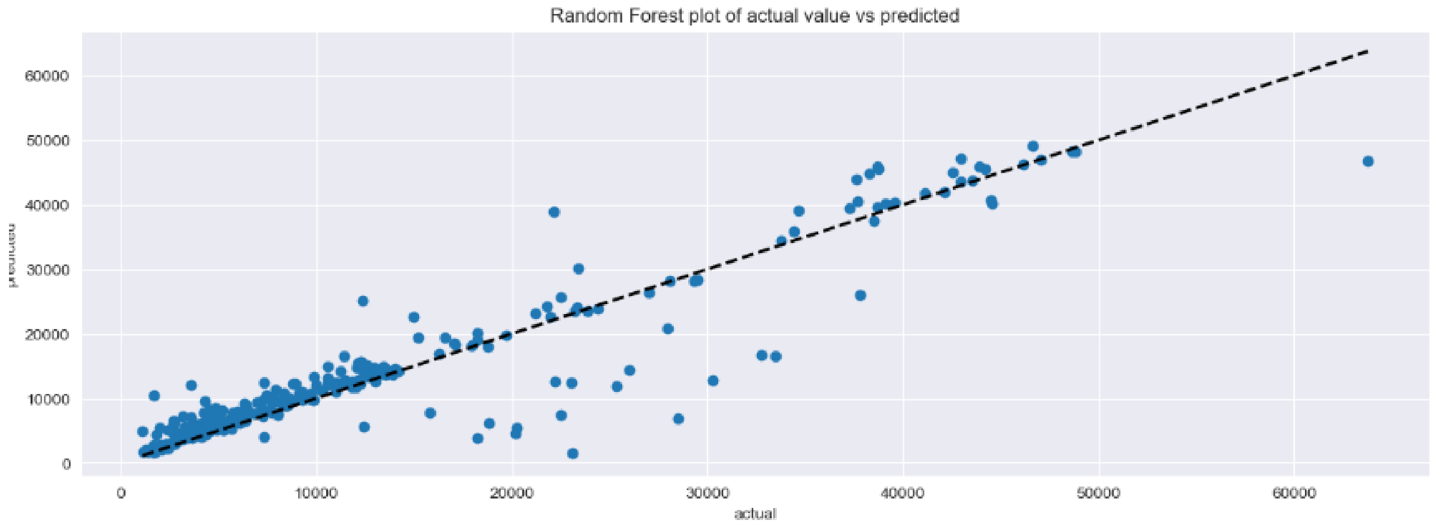
Random Forest




```
print('RMSE:', np.sqrt(mean_squared_error(y_test, rfc_pred)))
print('R2 Score: ', r2_score(y_test, rfc_pred))
```

```
RMSE: 4572.110965856546
R2 Score:  0.8653502770474271
```

Conclusion



Overall, the random forest regression model provided with the best predictions performance with high r-squared score and lower RMSE compare with the linear regression model.

While the Linear Regression model gives a good baseline, the Random Forest model is much more effective at capturing complex relationships within the data.

Thank You!