Medical Insurance Cost Prediction

August 31, 2022

1 Understanding the Context

As a data scientist hired by an insurance company trying to re-evaluate how much to charge for insurance. You've been tasked to predict how much any given individual will accumulate in medical bills based on the data that the insurance company has.

1.1 Importing the required libraries

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

# import specific components from scikit-learn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.model_selection import cross_val_score, cross_val_predict

# enhanced stats functions
from scipy import stats
```

Here, we also set the seed for numpy's random number generator such that our results are fully reproducible. This is because the other libraries (e.g. scikit-learn) use this random number generator, so if we set the seed we will always generate the same random numbers in the same sequence.

Thus, whenever we run the notebook from top-to-bottom, we will end up with the *exact* same results!

In [2]:
```
SEED = 123
np.random.seed(SEED)
```

2 Data Loading

To load in the data for this project, read in `insurance.csv` into a variable called `df` as a pandas DataFrame.

```
In [3]: # read in data
        df = pd.read_csv('insurance.csv')

In [4]: # let's have a quick look at the first few rows of the dataset
        df.head()
```

Out[4]:

	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

```
In [5]: # let's check the number of rows and columns in the dataset

        df.shape

Out[5]: (1338, 7)
```

The dataset has 1338 rows and 7 columns

3 Exploratory Data Analysis

Data dictionary

column	data definition
age	age of insured person
sex	sex of insured person, either male or female
bmi	body mass index of insured person
children	number of dependents covered by health insurance
smoker	does the insured person smoke?
region	the insured person's residential area within the US
charges	medical costs billed by health insurance; target variable

```
In [6]: # conduct any other EDA that you need to in order to get a good feel for the data
        df.describe()
```

Out[6]:

	age	bmi	children	charges
count	1338.000000	1338.000000	1338.000000	1338.000000
mean	39.207025	30.663397	1.094918	13270.422265
std	14.049960	6.098187	1.205493	12110.011237
min	18.000000	15.960000	0.000000	1121.873900
25%	27.000000	26.296250	0.000000	4740.287150
50%	39.000000	30.400000	1.000000	9382.033000
75%	51.000000	34.693750	2.000000	16639.912515
max	64.000000	53.130000	5.000000	63770.428010

```
In [7]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
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
```

```
In [8]: # check for missing values
        df.isnull().sum()
```

```
Out[8]: age         0
        sex         0
        bmi         0
        children    0
        smoker      0
        region      0
        charges     0
        dtype: int64
```

```
In [9]: # check if there are duplicated data
        df[df.duplicated(keep=False)]
```

Out[9]:

	age	sex	bmi	children	smoker	region	charges
195	19	male	30.59	0	no	northwest	1639.5631
581	19	male	30.59	0	no	northwest	1639.5631

In this case, we don't have any missing data, so we don't need to do anything about that. We have one duplicated row, but that seems to be legitimate data, so we will keep it in

3.1 More Explorations

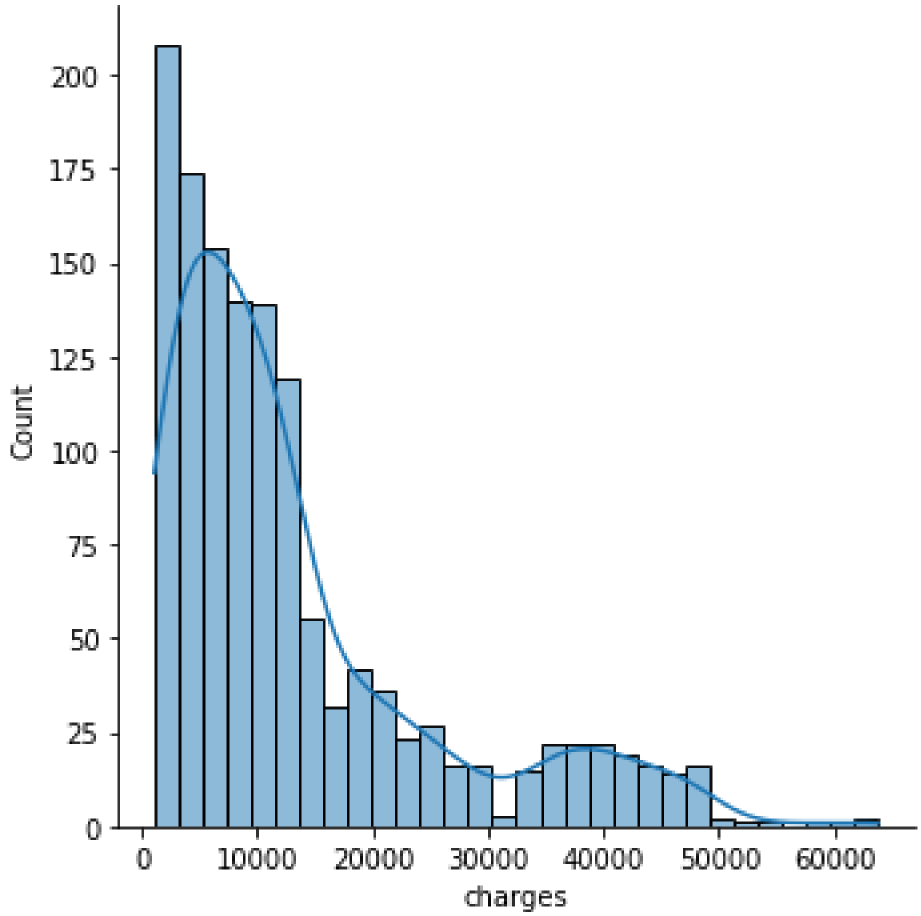
```
In [10]: df.groupby('sex').mean()
```

Out[10]:

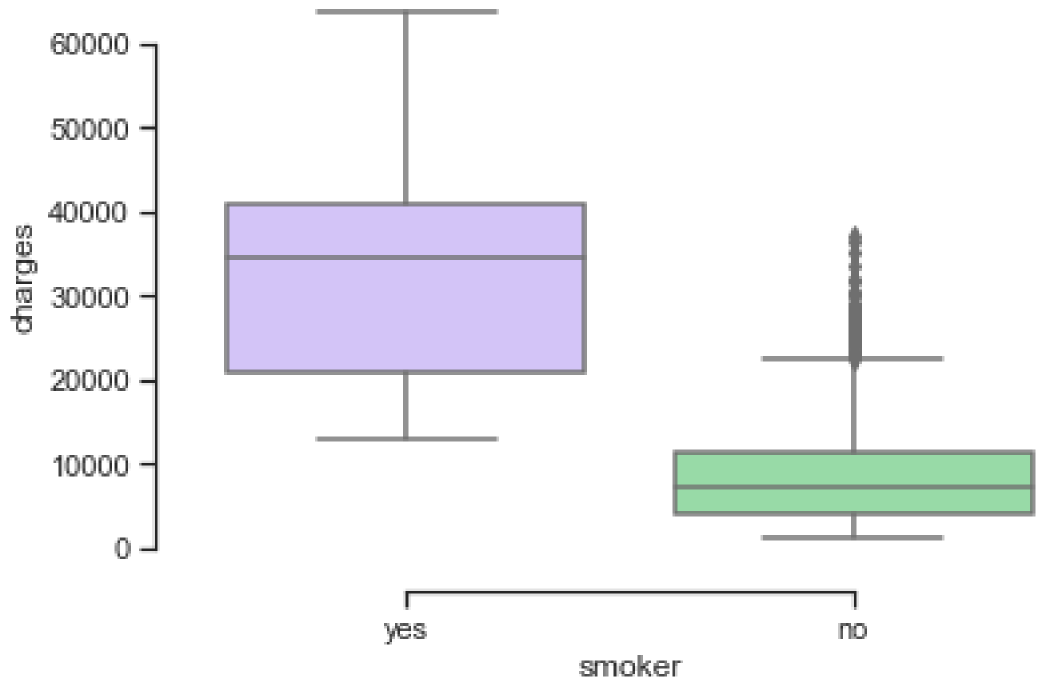
sex	age	bmi	children	charges
female	39.503021	30.377749	1.074018	12569.578844
male	38.917160	30.943129	1.115385	13956.751178

```
In [11]: # plot distribution of charges
         x = df['charges']
         sns.displot(x,kde=True)
```

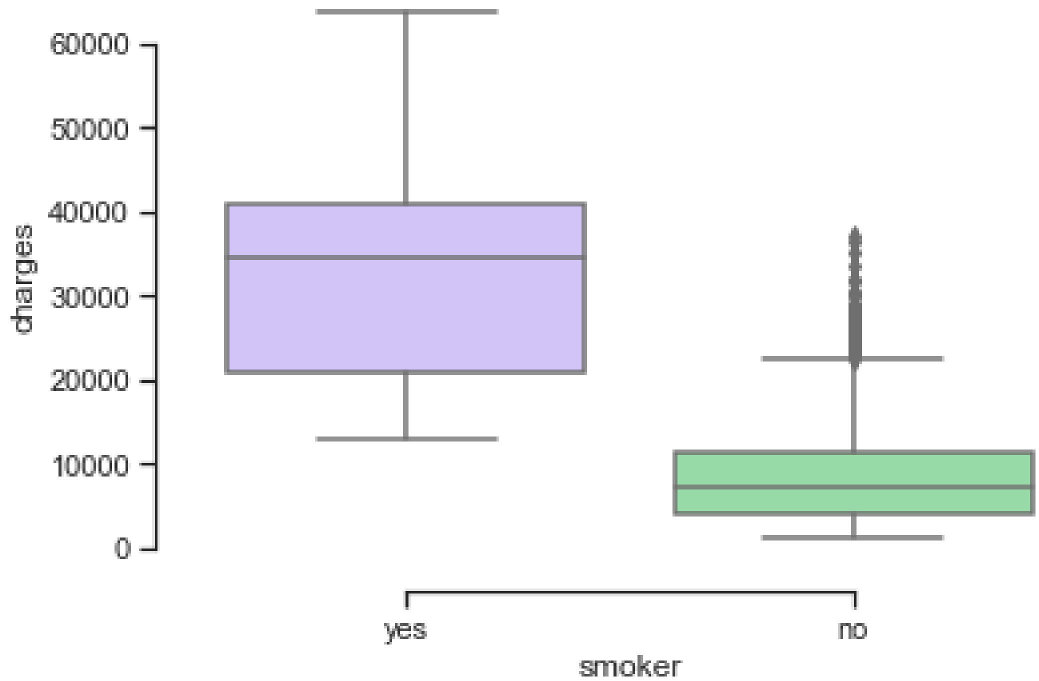
Out[11]: <seaborn.axisgrid.FacetGrid at 0x21aded576d8>



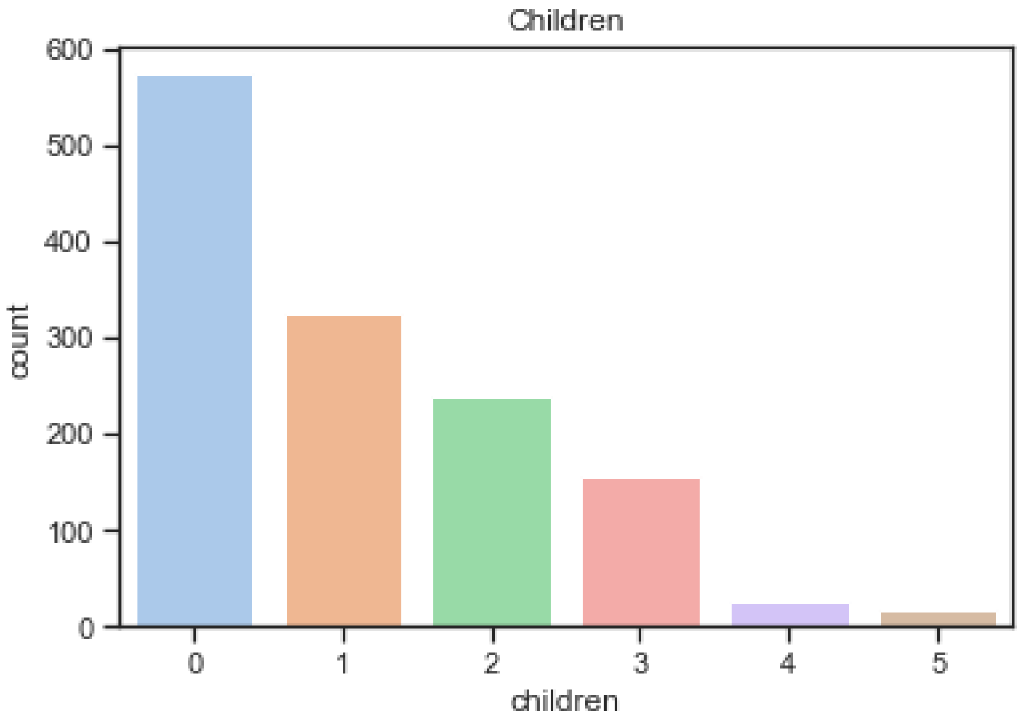
In [12]:
```
sns.set_theme(style = 'ticks', palette = 'pastel')

sns.boxplot(x='smoker', y='charges', palette=['m','g'], data=df)
sns.despine(offset=10, trim=True)
```



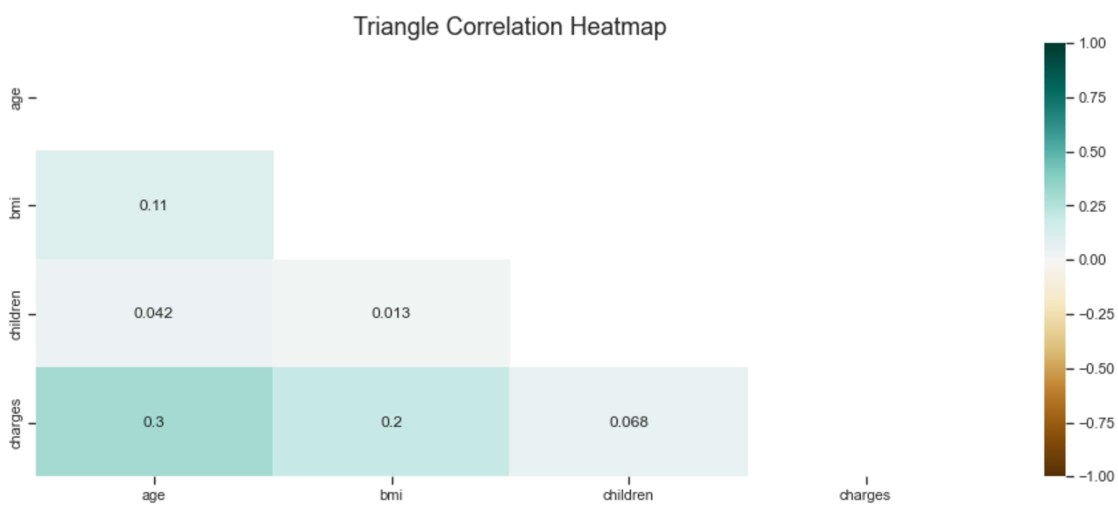
In [13]:
```
sns.set_theme(style = 'ticks', palette = 'pastel')

sns.boxplot(x='smoker', y='charges', palette=['m','g'], data=df)
sns.despine(offset=10, trim=True)
```




```
In [14]: # plot of children variable

         sns.countplot(x = 'children', data = df)
         plt.title('Children')
         plt.show()
```




```
In [15]: # plot correlation heatmap
         plt.figure(figsize=(16, 6))
         # define the mask to set the values in the upper triangle to True
         mask = np.triu(np.ones_like(df.corr(), dtype=bool))
         heatmap = sns.heatmap(df.corr(), mask=mask, vmin=-1, vmax=1, annot=True, cmap='BrBG')
         heatmap.set_title('Triangle Correlation Heatmap', fontdict={'fontsize':18});
```



4 Data Wrangling

Now that we have an idea of what our data looks like, we need to start preparing it for modelling.

From the distribution plot of `charges` above, the distribution of `charges` is not normally therefore there is need for a little transformation in some way in order to get it closer to a normal distribution.

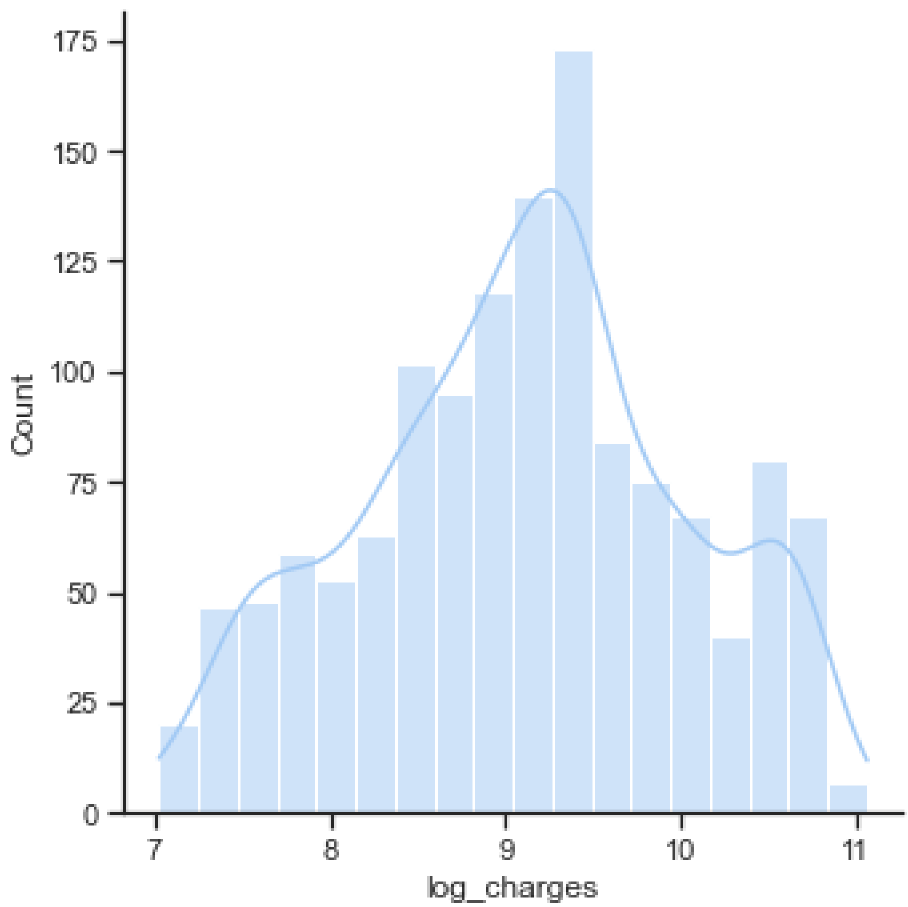
We can take the `log` of the chargers and name the new column `log_charges`. Then drop the `charge` column.

```
In [16]: df["log_charges"] = np.log(df['charges'])
```

```
In [17]: df.drop(columns = 'charges', inplace = True)
```

Now plot the new distribution to see whether it's a normal distribution.

```
In [18]: # plot distribution for log_charges
         # Distribution of log_charges
         x = df['log_charges']
         sns.displot(x,kde=True)
         plt.show()
```



Note that even after log transforming our data, it is still not normally distributed. But it is now closer to a normal distribution than the data originally was, so we can continue on with our analysis using this new variable `log_charges` variable as our target variable instead!

```
In [19]: df.head()
```

```
Out[19]:    age     sex     bmi  children smoker     region  log_charges
         0   19  female  27.900         0    yes  southwest     9.734176
         1   18    male  33.770         1     no  southeast     7.453302
         2   28    male  33.000         3     no  southeast     8.400538
         3   33    male  22.705         0     no  northwest     9.998092
         4   32    male  28.880         0     no  northwest     8.260197
```

We have 3 categorical variables - sex, smoker, and region. Get dummy variables using pandas, ensuring to use the `drop_first=True` argument to mitigate possible multicollinearity issues. As a result, you should only have one dummy variable for binary values such as `sex` or `smoker`.

```
In [20]: # get dummies for sex, smoker, and region
         sex = pd.get_dummies(df['sex'], prefix = 'sex', prefix_sep = '_', drop_first=True)
         smoker = pd.get_dummies(df['smoker'], prefix = 'smoker', prefix_sep = '_', drop_first
         region = pd.get_dummies(df['region'], prefix = 'region', prefix_sep = '_', drop_first
```

```
In [21]: df = pd.concat([df,sex],axis=1)
         df = pd.concat([df,smoker],axis=1)
         df = pd.concat([df,region],axis=1)
```

```
In [22]: df.head()
```

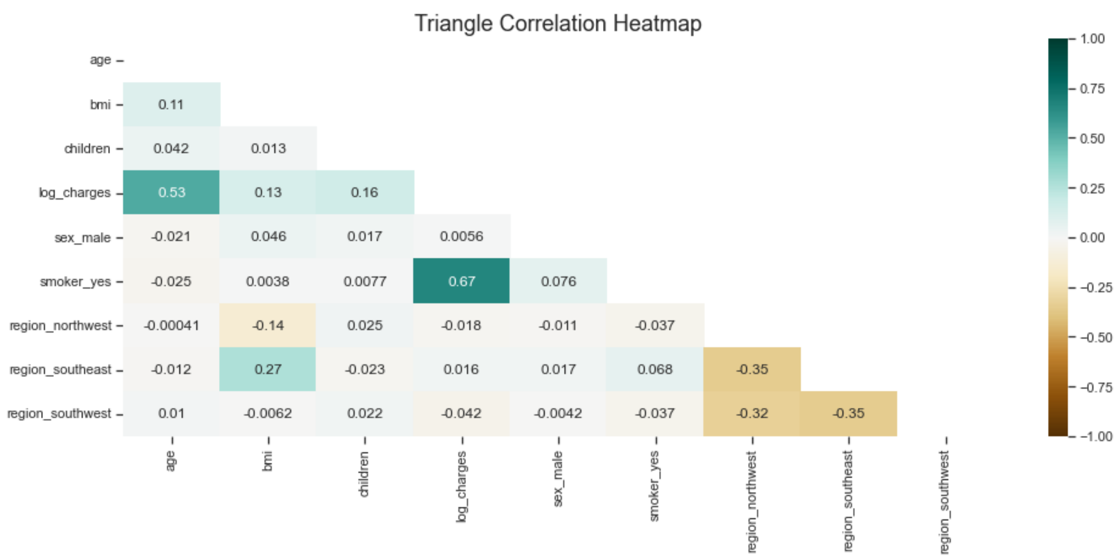
```
Out[22]:    age     sex     bmi  children smoker     region  log_charges  sex_male  \
         0   19  female  27.900         0    yes  southwest     9.734176         0
         1   18    male  33.770         1     no  southeast     7.453302         1
         2   28    male  33.000         3     no  southeast     8.400538         1
         3   33    male  22.705         0     no  northwest     9.998092         1
         4   32    male  28.880         0     no  northwest     8.260197         1

            smoker_yes  region_northwest  region_southeast  region_southwest
         0           1                 0                 0                 1
         1           0                 0                 1                 0
         2           0                 0                 1                 0
         3           0                 1                 0                 0
         4           0                 1                 0                 0
```

```
In [23]: # Now drop the categorical columns
         df.drop(columns=['sex'],inplace=True)
         df.drop(columns=['smoker'],inplace=True)
         df.drop(columns=['region'],inplace=True)
```

Now verify that we still don't have major multicollinearity issues with a heatmap. What we're looking for is that most of the correlations between independent variables should still be relatively low. There's no single cut-off value (much of this is as much an art as it is a science), but we'll say for our purposes here that we'll consider anything between -0.5 and 0.5 to be low.

Note that if a variable is strongly correlated with our dependent variable `log_charges`, that's okay. If anything, that's probably a good thing!

```
In [24]: # correlation plot heatmap
         plt.figure(figsize=(16, 6))
         # define the mask to set the values in the upper triangle to True
         mask1 = np.triu(np.ones_like(df.corr(), dtype=bool))
         heatmap1 = sns.heatmap(df.corr(), mask=mask1, vmin=-1, vmax=1, annot=True, cmap='BrBG
         heatmap1.set_title('Triangle Correlation Heatmap', fontdict={'fontsize':18});
```



In [25]: `df.head()`

Out[25]:

	age	bmi	children	log_charges	sex_male	smoker_yes	region_northwest \
0	19	27.900	0	9.734176	0	1	0
1	18	33.770	1	7.453302	1	0	0
2	28	33.000	3	8.400538	1	0	0
3	33	22.705	0	9.998092	1	0	1
4	32	28.880	0	8.260197	1	0	1

	region_southeast	region_southwest
0	0	1
1	1	0
2	1	0
3	0	0
4	0	0

5 Modeling, Training and Evaluation

Now separate our independent variables into a variable called `X`, and our target variable `log_charges` into a variable called `y`.

5.0.1 Split up our variables

In [26]:
```
X = df[['age', 'bmi', 'children', 'sex_male', 'smoker_yes', 'region_northwest', 'regi
y = df['log_charges']
```

5.0.2 Train test split

Now we need to split up our data into training and test data. Using scikit-learn's `train_test_split` function, using a `test_size` of 0.3 (i.e. 30% of data in test set), and **ensure that the random state is set to our seed from above**.

In [27]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_sta
```

5.1 Modelling and Training

Now we can make our model! Instantiate a LinearRegression model in scikit-learn, then fit the training data on it.

In [28]:
```
# instantiate linear regression model and fit the training data to it
lr = LinearRegression()
```

In [29]:
```
lr.fit(X_train, y_train)
```

Out[29]:
```
LinearRegression()
```

5.1.1 Describe model

In [30]:
```
pd.DataFrame(lr.coef_,X.columns,columns=['Coefficient'])
```

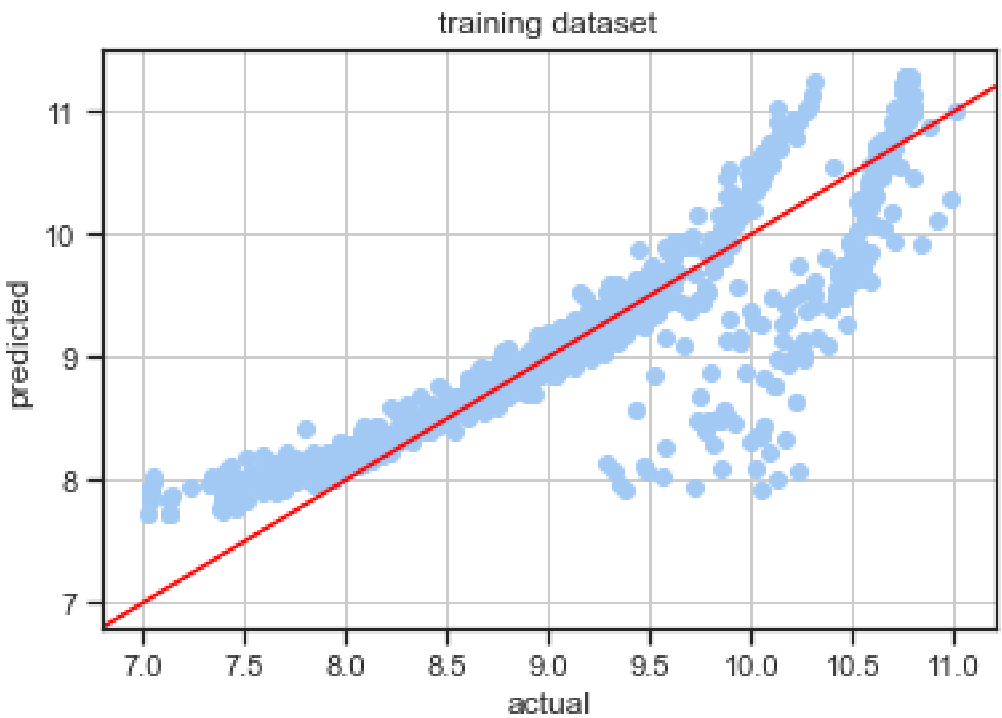
Out[30]:

	Coefficient
age	0.034708
bmi	0.015102
children	0.106982
sex_male	-0.083722
smoker_yes	1.532036
region_northwest	-0.087910
region_southeast	-0.154570
region_southwest	-0.143292

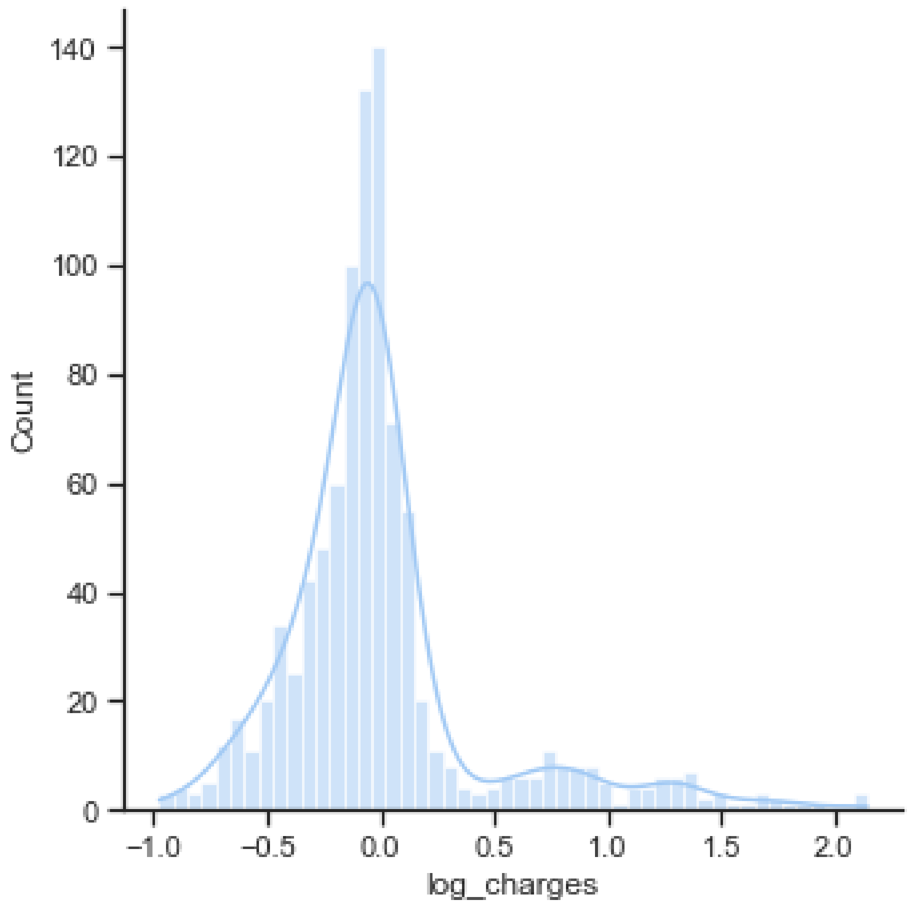
5.1.2 Predictions

In [31]:
```
# predicting training set values
predictions_train = lr.predict(X_train)
```

In [32]:
```
# plot predictions against actual values
plt.scatter(y_train, predictions_train)
plt.title("training dataset")
plt.xlabel("actual")
plt.ylabel("predicted")
plt.gca()
plt.gca().axline([7, 7], [11, 11], color="red")
plt.grid()
```

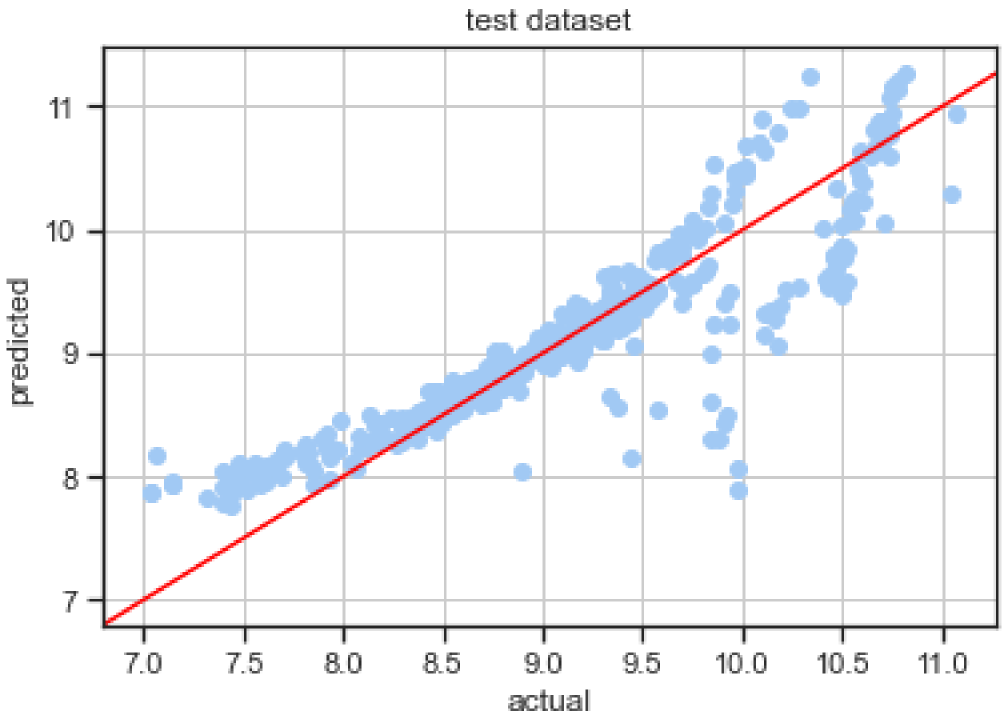


In [33]:
```
# Residual Plot
sns.displot((y_train-predictions_train),bins=50, kde=True);
```

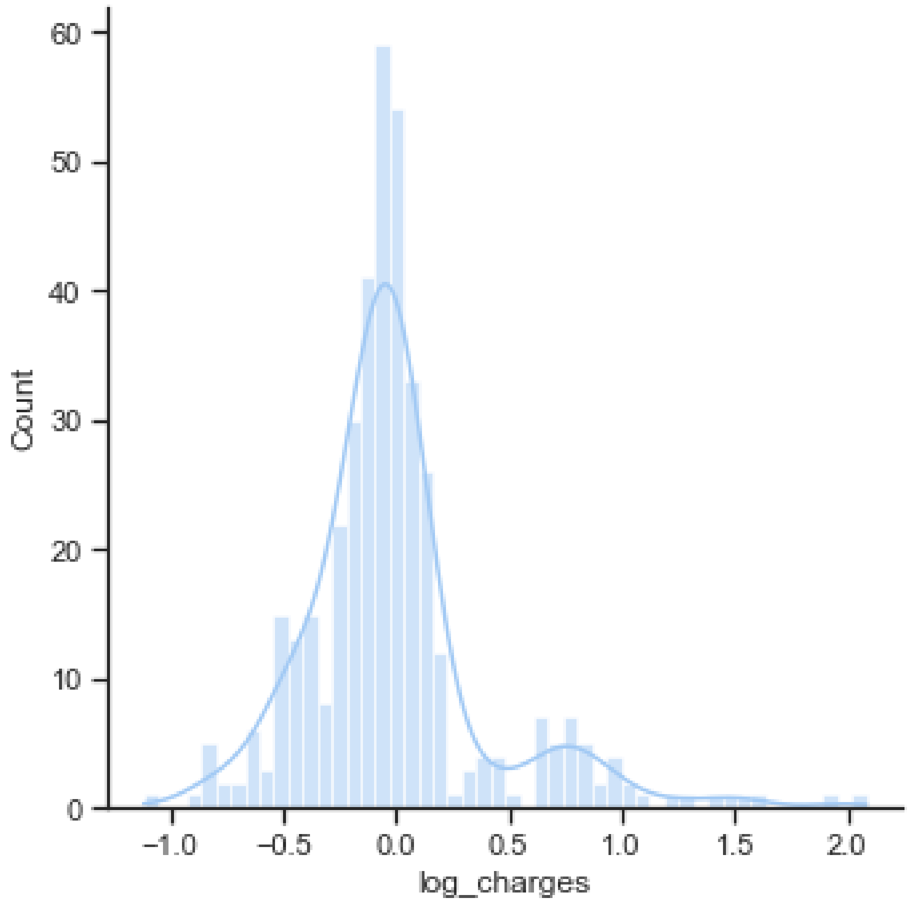



```
In [34]: # make predictions on test set
         predictions_test = lr.predict(X_test)
```

```
In [35]: # plot predictions against actual values
         plt.scatter(y_test, predictions_test)
         plt.title("test dataset")
         plt.xlabel("actual")
         plt.ylabel("predicted")
         plt.gca()
         plt.gca().axline([7, 7], [11, 11], color="red")
         plt.grid()
```




```
In [36]: # Residual Plot
         sns.displot((y_test-predictions_test),bins=50, kde=True);
```



5.2 Evaluation

5.2.1 Metrics

Mean Absolute Error (MAE) is the mean of the absolute value of the errors:

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Mean Squared Error (MSE) is the mean of the squared errors:

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

Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors:

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

R Square or the coefficient of determinant is the amount of variation that can be explained by the model:

Comparing these metrics:

- **MAE** is the easiest to understand, because it's the average error. Good for ignoring outlier point in dataset
- **MSE** is more popular than MAE, because MSE "punishes" larger errors.
- **RMSE** is even more popular than MSE, because RMSE is interpretable in the "y" units.

All of these are **loss functions**, because we want to minimize them.

Metrics for training data

```
In [37]: r2_train = r2_score(y_train, predictions_train)
         mae_train = mean_absolute_error(y_train, predictions_train)
         mse_train = mean_squared_error(y_train, predictions_train)
         rmse_train = np.sqrt(mean_squared_error(y_train, predictions_train))
```

```
In [38]: print(f'MAE: {mae_train:.4f}')
         print(f'MSE: {mse_train:.4f}')
         print(f'RMSE: {rmse_train:.4f}')
         print(f'R_squared: {r2_train:.4f}')
```

```
MAE: 0.2880
MSE: 0.2076
RMSE: 0.4557
R_squared: 0.7608
```

Clearly, while our model has clearly learned something about the data, there is still room for improvement.

Now we're going to evaluate our model's performance on the test set.

Metrics for training data

```
In [39]: r2_test = r2_score(y_test, predictions_test)
         mae_test = mean_absolute_error(y_test, predictions_test)
         mse_test = mean_squared_error(y_test, predictions_test)
         rmse_test =np.sqrt(mean_squared_error(y_test, predictions_test))
```

```
In [40]: print(f'MAE: {mae_test:.4f}')
         print(f'MSE: {mse_test:.4f}')
         print(f'RMSE: {rmse_test:.4f}')
         print(f'R_squared: {r2_test:.4f}')
```

```
MAE: 0.2697
MSE: 0.1705
RMSE: 0.4129
R_squared: 0.7841
```

```
In [41]: lr.score(X_test, y_test)
```

```
Out[41]: 0.7841289883812209

In [42]: cross_val_score(lr, X_test, y_test,cv=3)

Out[42]: array([0.76999112, 0.83921275, 0.73168705])

In [43]: cross_val_score(lr, X_test, y_test).mean()

Out[43]: 0.7767290629022574

In [44]: cross_val_predict(lr, X_test, y_test,cv=3)

Out[44]: array([ 9.25762024,  8.59503226,  9.8706121 ,  8.52803096,  9.12827551,
                 9.19307058,  8.3685776 ,  8.37082845,  7.92763913,  8.94966325,
                 8.69150676,  9.28394259,  9.21576437, 10.03414854,  9.43126867,
                10.23613942,  9.27821832,  7.94820684, 10.44984919, 10.61998214,
                10.70957233, 10.92780854,  8.15667981,  8.8641612 ,  8.0230341 ,
                 9.05302627,  9.63270103,  8.48107512,  9.52429789,  7.82853021,
                 9.01922211,  9.39861379,  8.16972548,  8.03650976,  8.37818918,
                 9.36046081,  9.58364176,  8.15707052,  8.90138909,  9.37776241,
                 8.61030091,  8.68566455,  8.14206513, 10.55402211,  8.91787337,
                 7.828012  ,  9.2483868 ,  9.35124851,  9.50698069,  8.67673824,
                 9.24236354,  9.19481301,  9.73276746,  9.05656426,  9.47165372,
                 8.68952473, 10.52295467,  9.54401203, 10.92627561,  8.49604174,
                 9.85699506,  8.68268405, 11.22551629, 10.0858944 ,  8.17984559,
                 8.40031648,  9.57478992,  8.51769743,  9.56020044,  9.35964695,
                 9.31532972,  9.23673223,  9.3008768 ,  9.3896158 ,  9.27525175,
                10.49441279,  8.26137847,  9.30503823,  8.60112825, 10.23983862,
                 7.94371948, 10.3788445 , 10.08977553,  9.26430246,  8.15078865,
                 9.80390038,  9.85689215,  9.45642782,  9.2253282 , 11.27610126,
                 8.97757491,  8.68985187,  8.8257367 ,  8.3850913 ,  7.94318706,
                 7.98685316,  8.58747854,  8.67362301,  8.4809094 ,  8.04612952,
                 9.29425439,  9.08670907,  9.29781284,  8.13757379,  8.92431443,
                 8.1850281 ,  9.73116457,  8.68299221,  9.38163003,  8.95411879,
                 8.0609954 , 10.68454503, 10.3263494 ,  8.86735372, 10.10794962,
                 9.33682432,  8.74652915,  8.77774481,  9.23659933,  8.66469181,
                10.73122933,  9.89624086,  8.5951409 ,  9.26808528,  8.63495855,
                 8.41324717, 10.45709205,  8.79264045, 10.63066147, 10.99607446,
                 8.41844564,  8.32256656,  8.02065126,  9.1260135 ,  8.87950233,
                 8.39326305,  8.4533114 ,  8.46611348,  8.48188562,  9.25194789,
                 8.25757248,  9.5217882 , 10.09421505,  8.88971105,  9.56121431,
                 8.84673784,  9.81483528,  8.60678423,  9.1520586 ,  8.2557396 ,
                 7.89766439,  9.45727732,  9.61402746,  9.09075161,  8.27411444,
                 8.63617002, 10.8295892 ,  9.24224041,  9.57747187,  8.26447818,
                 8.67529245,  9.07481476,  8.58980315, 10.74158049,  8.53644102,
                 8.00530035,  9.08755487,  9.48568035,  9.8659674 ,  8.67046857,
                 8.20786547, 10.20126657,  9.3378093 ,  9.69670686,  9.51691522,
                 8.23151818,  8.0790724 ,  8.98972672,  9.43648701,  8.30172345,
                 8.6571094 ,  9.18738629,  8.59599145,  8.3077993 ,  9.20074963,
```

```
 8.56871643,  9.58132338,  8.3461323 ,  9.02756082,  9.70353602,
10.60471883,  9.55553227, 10.82844466,  8.0896102 ,  8.38157534,
 8.60284923,  9.48497457,  9.09487165,  9.83189935,  9.28667238,
11.03649469,  9.45924296,  7.93728445,  8.28629052,  8.2677288 ,
 9.637571  , 10.57860876, 10.92859611,  8.12250803,  8.12564021,
 8.76891874,  8.02011429,  9.19261055,  8.91389763,  9.04255624,
11.08306043,  9.45362343,  8.8092711 ,  9.97596541,  8.96464738,
 9.79616586,  8.45226609,  9.11322543,  9.41114753,  9.91814409,
 8.00586865,  8.59854638,  7.89800294,  9.28029759,  9.50198739,
 8.60592586,  7.99220354,  8.8408036 ,  9.68784178,  9.0600776 ,
 8.68817565,  8.39274966,  9.49163706, 10.58901981, 10.82102774,
 8.84468754,  8.66727391,  9.10460823,  9.06209144,  9.17739682,
 9.4794666 ,  9.30921655,  8.29390086,  8.35113901, 10.26036028,
 9.3688486 ,  9.1831299 ,  8.96625599,  9.67429803,  9.1369569 ,
 8.73573936,  9.99979062,  9.42074107,  9.07065023,  9.0311594 ,
10.50090111,  8.61073144, 11.18963193,  8.84839188,  8.58843337,
 9.17742255,  8.33572133,  9.49705009,  8.73821665, 10.22503094,
 8.96722508,  9.14386975,  8.06510491,  9.18913239,  8.98253884,
 8.52284322,  9.69980241,  8.86462243, 10.61450032,  9.09413957,
 9.21155958,  8.47961406, 10.57724532,  8.9564592 ,  8.41814279,
 9.38175834, 10.81711655,  9.09915453,  9.45654685,  8.44059238,
10.18571071, 10.92073134,  8.63885126,  8.89227798,  9.45331018,
11.07124866,  9.65192945,  7.98337793,  9.46481065,  8.55159591,
 9.7543068 ,  7.85674687,  8.88498243,  9.98059342,  9.11804105,
 8.67830738,  9.49061362, 10.93946769, 10.37291491,  8.44710882,
 9.67715017,  7.90373295,  9.03617327, 10.67468599,  9.29123484,
 9.74919804,  9.02680592,  9.67708855,  8.91231723,  9.06775788,
 9.40589891,  8.36753845,  8.98920504,  9.27528818,  8.57536389,
 9.31654049,  9.18081958,  9.22958472,  9.13874619,  9.22264245,
 9.20911049,  9.31305984,  9.07303559,  8.05571499,  8.73779211,
10.33733176,  7.94841618, 10.48570414,  8.16551868,  9.78523279,
10.86120656,  9.15473815,  8.47999173,  8.381986  ,  9.32361088,
 7.98239604,  9.26844248,  8.96137831,  8.23988145,  9.32409796,
 8.90058468, 10.40750025,  9.26657726,  8.78705591,  8.42260027,
 9.23474218,  8.42361478,  8.28869058,  8.01962775,  9.84479679,
10.09101172,  8.46484426,  8.64286961,  9.26830072,  8.02679749,
 8.18465532,  8.54403501,  9.21622142,  8.7593879 ,  8.78243598,
 8.78417372,  8.29630744,  9.41071295,  7.84145469, 10.19138396,
11.08697982,  9.85236943,  9.03187226,  9.60486942,  8.29425336,
 8.63102156,  7.88988287,  8.7598466 ,  9.32755355,  7.9308429 ,
 9.77375189,  8.56824865,  9.23257553, 11.24755923,  9.40413296,
 8.33096531,  9.4514972 ,  9.0923175 ,  9.44454548,  8.50675329,
 7.96509294,  8.04121039,  8.70712797,  8.07642802,  9.79598528,
 8.60671154,  8.0669617 ])
```