Predicting medical expenses using linear regression

The goal of this analysis is to use patient data to estimate the average medical care expenses. insurance.csv file contains simulated dataset containing hypothetical medical expenses for patients in the United States

First read data in the file into a dataframe object

In [1]:
```
import pandas as pd
data=pd.read_csv("insurance.csv")
```

In [2]:
```
data.head()
```

Out[2]:

	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 [3]:
```
len(data)
```

Out[3]:
```
1338
```

The insurance.csv le includes 1,338 examples of beneficiaries currently enrolled in the insurance plan, with features indicating characteristics of the patient as well as the total medical expenses charged to the plan for the calendar year. The features are: • age: An integer indicating the age of the primary beneficiary (excluding those above 64 years, since they are generally covered by the government). • sex: The policy holder's gender, either male or female. • bmi: The body mass index (BMI), which provides a sense of how over- or under-weight a person is relative to their height. BMI is equal to weight (in kilograms) divided by height (in meters) squared. An ideal BMI is within the range of 18.5 to 24.9. • children: An integer indicating the number of children/dependents covered by the insurance plan. • smoker: A yes or no categorical variable that indicates whether the insured regularly smokes tobacco. • region: The beneficiary's place of residence in the US, divided into four geographic regions: northeast, southeast, southwest, or northwest.

Exploring and preparing the data

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

Out[4]:

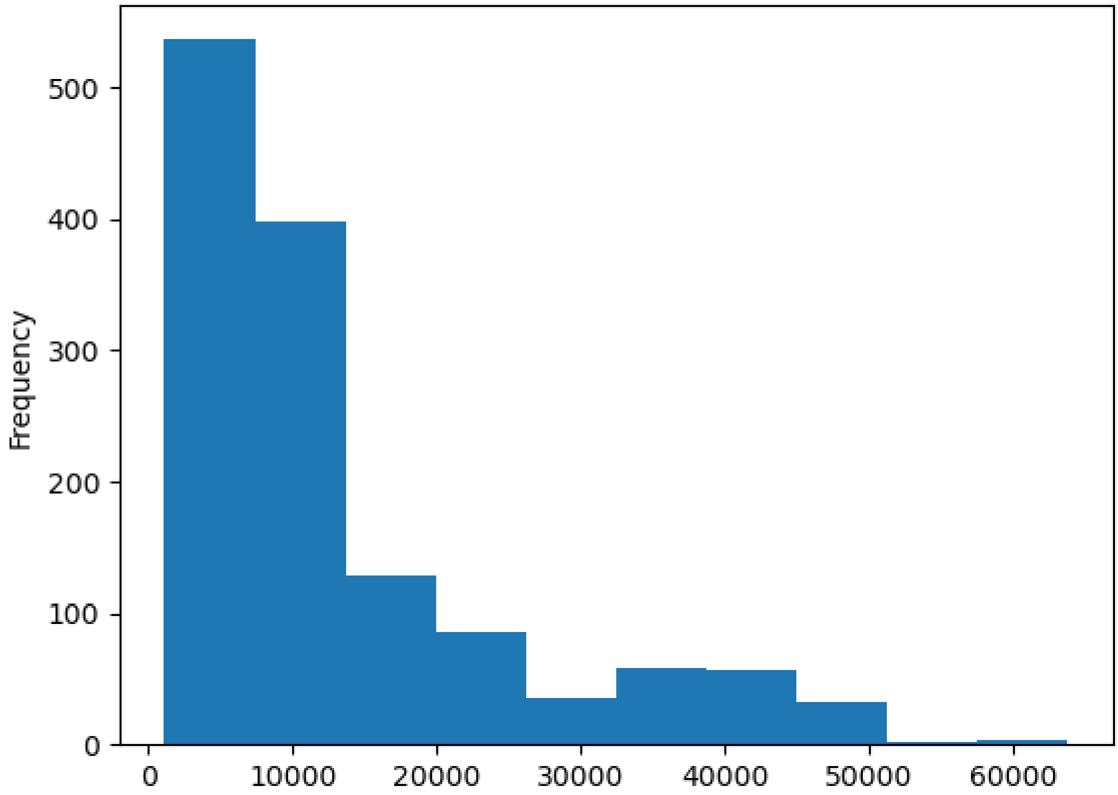
	age	bmi	children	charges
count	1338.000000	1338.000000	1338.000000	1338.000000
mean	39.207025	30.663397	1.094918	13270.422265
std	14.049960	6.098187	1.205493	12110.011237
min	18.000000	15.960000	0.000000	1121.873900
25%	27.000000	26.296250	0.000000	4740.287150
50%	39.000000	30.400000	1.000000	9382.033000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

Because the mean value is greater than the median, this implies that the distribution of insurance expenses is right-skewed. We can con rm this visually using a histogram:

In [7]:

```
data.charges.plot.hist()
```

Out[7]: <Axes: ylabel='Frequency'>



As expected, the gure shows a right-skewed distribution. It also shows that the majority of people in our data have yearly medical expenses between zero and $15,000, in spite of the fact that the tail of the distribution extends far past these peaks. Although this distribution is not ideal for a linear regression, knowing this weakness ahead of time may help us design a better-fitting model later on.

Exploring relationships among features – the correlation matrix

Before fitting a regression model to data, it can be useful to determine how the independent variables are related to the dependent variable and each other. A correlation matrix provides a quick overview of these relationships. "Given a set of variables, it provides a correlation for each pairwise relationship

To create a correlation matrix for the four numeric variables in the insurance data, we use dataframe.corr() function

The correlation coefficients are bounded to the range -1 and 1. Two features have a perfect positive correlation if r =1, no correlation if r = 0 , and a perfect negative correlation if r = −1 , respectively. Pandas.corr() function supports different correlation algorithm, the default one is Pearson's correlation coefficient. It can be can simply be calculated as the covariance between two features x and y (numerator) divided by the product of their standard deviations (denominator).

In [10]:
```
data.corr()
```

```
C:\Users\ttazegul\AppData\Local\Temp\ipykernel_19072\2627137660.py:1: FutureWarnin
g: The default value of numeric_only in DataFrame.corr is deprecated. In a future
version, it will default to False. Select only valid columns or specify the value
of numeric_only to silence this warning.
  data.corr()
```

Out[10]:

	age	bmi	children	charges
age	1.000000	0.109272	0.042469	0.299008
bmi	0.109272	1.000000	0.012759	0.198341
children	0.042469	0.012759	1.000000	0.067998
charges	0.299008	0.198341	0.067998	1.000000

Fitting a linear model

In [12]:
```
from sklearn.linear_model import LinearRegression
```

In [13]:
```
lm=LinearRegression()
```

In [18]:
```
X=data.iloc[:,:6];X.head()
```

Out[18]:

	age	sex	bmi	children	smoker	region
0	19	female	27.900	0	yes	southwest
1	18	male	33.770	1	no	southeast
2	28	male	33.000	3	no	southeast
3	33	male	22.705	0	no	northwest
4	32	male	28.880	0	no	northwest

In [19]:
```
y=data.iloc[:,6];y.head()
```

Out[19]:
```
0    16884.92400
1     1725.55230
2     4449.46200
3    21984.47061
4     3866.85520
Name: charges, dtype: float64
```

In [20]:
```
lm.fit(X,y)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[20], line 1
----> 1 lm.fit(X,y)

File ~\AppData\Local\anaconda3\Lib\site-packages\sklearn\base.py:1151, in _fit_con
text.<locals>.decorator.<locals>.wrapper(estimator, *args, **kwargs)
   1144     estimator._validate_params()
   1146 with config_context(
   1147     skip_parameter_validation=(
   1148         prefer_skip_nested_validation or global_skip_validation
   1149     )
   1150 ):
-> 1151     return fit_method(estimator, *args, **kwargs)

File ~\AppData\Local\anaconda3\Lib\site-packages\sklearn\linear_model\_base.py:67
8, in LinearRegression.fit(self, X, y, sample_weight)
    674 n_jobs_ = self.n_jobs
    676 accept_sparse = False if self.positive else ["csr", "csc", "coo"]
--> 678 X, y = self._validate_data(
    679     X, y, accept_sparse=accept_sparse, y_numeric=True, multi_output=True
    680 )
    682 has_sw = sample_weight is not None
    683 if has_sw:

File ~\AppData\Local\anaconda3\Lib\site-packages\sklearn\base.py:621, in BaseEstim
ator._validate_data(self, X, y, reset, validate_separately, cast_to_ndarray, **che
ck_params)
    619         y = check_array(y, input_name="y", **check_y_params)
    620     else:
--> 621         X, y = check_X_y(X, y, **check_params)
    622     out = X, y
    624 if not no_val_X and check_params.get("ensure_2d", True):

File ~\AppData\Local\anaconda3\Lib\site-packages\sklearn\utils\validation.py:1147,
in check_X_y(X, y, accept_sparse, accept_large_sparse, dtype, order, copy, force_a
ll_finite, ensure_2d, allow_nd, multi_output, ensure_min_samples, ensure_min_featu
res, y_numeric, estimator)
   1142         estimator_name = _check_estimator_name(estimator)
   1143     raise ValueError(
   1144         f"{estimator_name} requires y to be passed, but the target y is No
ne"
   1145     )
-> 1147 X = check_array(
   1148     X,
   1149     accept_sparse=accept_sparse,
   1150     accept_large_sparse=accept_large_sparse,
   1151     dtype=dtype,
   1152     order=order,
   1153     copy=copy,
   1154     force_all_finite=force_all_finite,
   1155     ensure_2d=ensure_2d,
   1156     allow_nd=allow_nd,
   1157     ensure_min_samples=ensure_min_samples,
   1158     ensure_min_features=ensure_min_features,
   1159     estimator=estimator,
   1160     input_name="X",
   1161 )
   1163 y = _check_y(y, multi_output=multi_output, y_numeric=y_numeric, estimator=
estimator)
   1165 check_consistent_length(X, y)

File ~\AppData\Local\anaconda3\Lib\site-packages\sklearn\utils\validation.py:917,
in check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy, forc
```

```
e_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, estima
tor, input_name)
    915         array = xp.astype(array, dtype, copy=False)
    916     else:
--> 917         array = _asarray_with_order(array, order=order, dtype=dtype, xp=x
p)
    918 except ComplexWarning as complex_warning:
    919     raise ValueError(
    920         "Complex data not supported\n{}\n".format(array)
    921     ) from complex_warning

File ~\AppData\Local\anaconda3\Lib\site-packages\sklearn\utils\_array_api.py:380,
in _asarray_with_order(array, dtype, order, copy, xp)
    378     array = numpy.array(array, order=order, dtype=dtype)
    379 else:
--> 380     array = numpy.asarray(array, order=order, dtype=dtype)
    382 # At this point array is a NumPy ndarray. We convert it to an array
    383 # container that is consistent with the input's namespace.
    384 return xp.asarray(array)

File ~\AppData\Local\anaconda3\Lib\site-packages\pandas\core\generic.py:2070, in N
DFrame.__array__(self, dtype)
   2069 def __array__(self, dtype: npt.DTypeLike | None = None) -> np.ndarray:
-> 2070     return np.asarray(self._values, dtype=dtype)

ValueError: could not convert string to float: 'female'
```

We got an error, we need to convert categorical variables into a numerical values

In [23]:
```
data.head()
```

Out[23]:

	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 [21]:
```
X=pd.get_dummies(X,drop_first=True)
```

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

Out[22]:

	age	bmi	children	sex_male	smoker_yes	region_northwest	region_southeast	region_southw
0	19	27.900	0	0	1	0	0	
1	18	33.770	1	1	0	0	1	
2	28	33.000	3	1	0	0	1	
3	33	22.705	0	1	0	1	0	
4	32	28.880	0	1	0	1	0	

In [24]:
```
lm.fit(X,y)
```

Out[24]:

```
▾ LinearRegression
LinearRegression()
```

In [26]:

```
lm.coef_
```

Out[26]:

```
array([  256.85635254,   339.19345361,   475.50054515,  -131.3143594 ,
       23848.53454191,  -352.96389942, -1035.02204939,  -960.0509913 ])
```

The beta coef cients indicate the estimated increase in expenses for an increase of one in each of the features, assuming all other values are held constant. For instance, for each additional year of age, we would expect 256.85 dollars higher medical expenses on average, assuming everything else is equal. Similarly, each additional child results in an average of 475.50 dollars in additional medical expenses each year, and each unit increase in BMI is associated with an average increase of 339.19 dollars in yearly medical expenses, all else equal.

In [27]:

```
pd.concat([pd.Series(X.columns),pd.Series(lm.coef_)],axis=1)
```

Out[27]:

	0	1
0	age	256.856353
1	bmi	339.193454
2	children	475.500545
3	sex_male	-131.314359
4	smoker_yes	23848.534542
5	region_northwest	-352.963899
6	region_southeast	-1035.022049
7	region_southwest	-960.050991

People who smoke have 23K more expense compared to people who do not

Evaluating model performance

In [29]:

```
lm.score(X,y)
```

Out[29]:

```
0.7509130345985207
```

R-squared value is 0.75 for our model

Adding non-linear relationships

In [30]:

```
X["Age2"]=X["age"]*X["age"]
```

In [31]:

```
X.head()
```

Out[31]:

	age	bmi	children	sex_male	smoker_yes	region_northwest	region_southeast	region_southw
0	19	27.900	0	0	1	0	0	
1	18	33.770	1	1	0	0	1	
2	28	33.000	3	1	0	0	1	
3	33	22.705	0	1	0	1	0	
4	32	28.880	0	1	0	1	0	

Transformation – converting a numeric variable to a binary indicator

Suppose we have a hunch that the effect of a feature is not cumulative, rather it has an effect only after a specific threshold has been reached. For instance, BMI may have zero impact on medical expenditures for individuals in the normal weight range, but it may be strongly related to higher costs for the obese (that is, BMI of 30 or above). We can model this relationship by creating a binary obesity indicator variable that is 1 if the BMI is at least 30, and 0 if less. The estimated beta for this binary feature would then indicate the average net impact on medical expenses for individuals with BMI of 30 or above, relative to those with BMI less than 30.

In [32]:
```
X["bmi"]=X["bmi"].map(lambda x:1 if x>30 else 0)
```

In [33]:
```
X.head()
```

Out[33]:

	age	bmi	children	sex_male	smoker_yes	region_northwest	region_southeast	region_southwe
0	19	0	0	0	1	0	0	
1	18	1	1	1	0	0	1	
2	28	1	3	1	0	0	1	
3	33	0	0	1	0	1	0	
4	32	0	0	1	0	1	0	

Model specification – adding interaction effects

So far, we have only considered each feature's individual contribution to the outcome. What if certain features have a combined impact on the dependent variable? For instance, smoking and obesity may have harmful effects separately, but it is reasonable to assume that their combined effect may be worse than the sum of each one alone. When two features have a combined effect, this is known as an interaction. If we suspect that two variables interact, we can test this hypothesis by adding their interaction to the model we would write a formula in the form expenses ~ bmi30*smoker.

In [35]:
```
X["interaction"]=X["bmi"]*X["smoker_yes"]
```

In [38]:
```
X.head(20)
```

Out[38]:

	age	bmi	children	sex_male	smoker_yes	region_northwest	region_southeast	region_southw
0	19	0	0	0	1	0	0	
1	18	1	1	1	0	0	1	
2	28	1	3	1	0	0	1	
3	33	0	0	1	0	1	0	
4	32	0	0	1	0	1	0	
5	31	0	0	0	0	0	1	
6	46	1	1	0	0	0	1	
7	37	0	3	0	0	1	0	
8	37	0	2	1	0	0	0	
9	60	0	0	0	0	1	0	
10	25	0	0	1	0	0	0	
11	62	0	0	0	1	0	1	
12	23	1	0	1	0	0	0	
13	56	1	0	0	0	0	1	
14	27	1	0	1	1	0	1	
15	19	0	1	1	0	0	0	
16	52	1	1	0	0	0	0	
17	23	0	0	1	0	0	0	
18	56	1	0	1	0	0	0	
19	30	1	0	1	1	0	0	

In [39]:
```
lm.fit(X,y)
```

Out[39]:

▾ LinearRegression

LinearRegression()

In [40]:
```
lm.score(X,y)
```

Out[40]:

0.8668011847459627

Let's calculate score using different metrics

In [41]:
```
from sklearn.model_selection import cross_val_score
```

In [43]:
```
cross_val_score(lm,X,y,cv=5,scoring="neg_mean_absolute_error")
```

Out[43]:
```
array([-2283.83211007, -2737.34862364, -2292.97631079, -2532.97528535,
       -2512.36598001])
```

In [70]:
```
df=pd.DataFrame([cross_val_score(lm,X,y,cv=5,scoring="neg_mean_absolute_error")])
```

In [71]:
```
df
```

Out[71]:

	0	1	2	3	4
0	-2283.83211	-2737.348624	-2292.976311	-2532.975285	-2512.36598

Let's use cross_validate function

The cross_validate function differs from cross_val_score in two ways:

It allows specifying multiple metrics for evaluation. It returns a dict containing fit-times, score-times (and optionally training scores as well as fitted estimators) in addition to the test score.

In [73]:
```
from sklearn.model_selection import cross_validate
```

In [74]:
```
result=cross_validate(lm,X,y,cv=5,scoring="neg_mean_absolute_error",return_train_sc
```

In [75]:
```
result
```

Out[75]:
```
{'fit_time': array([0.00561023, 0.00307274, 0.00373101, 0.00050163, 0.        ]),
 'score_time': array([0.00212455, 0.0005281 , 0.        , 0.00153446, 0.0046129
2]),
 'test_score': array([-2283.83211007, -2737.34862364, -2292.97631079, -2532.975285
35,
        -2512.36598001]),
 'train_score': array([-2551.40112178, -2306.9530822 , -2552.97741481, -2419.71641
303,
        -2430.37535542])}
```

In [86]:
```
result2=pd.DataFrame([cross_validate(lm,X,y,cv=5,scoring="neg_mean_absolute_error",
```

In [87]:
```
result2
```

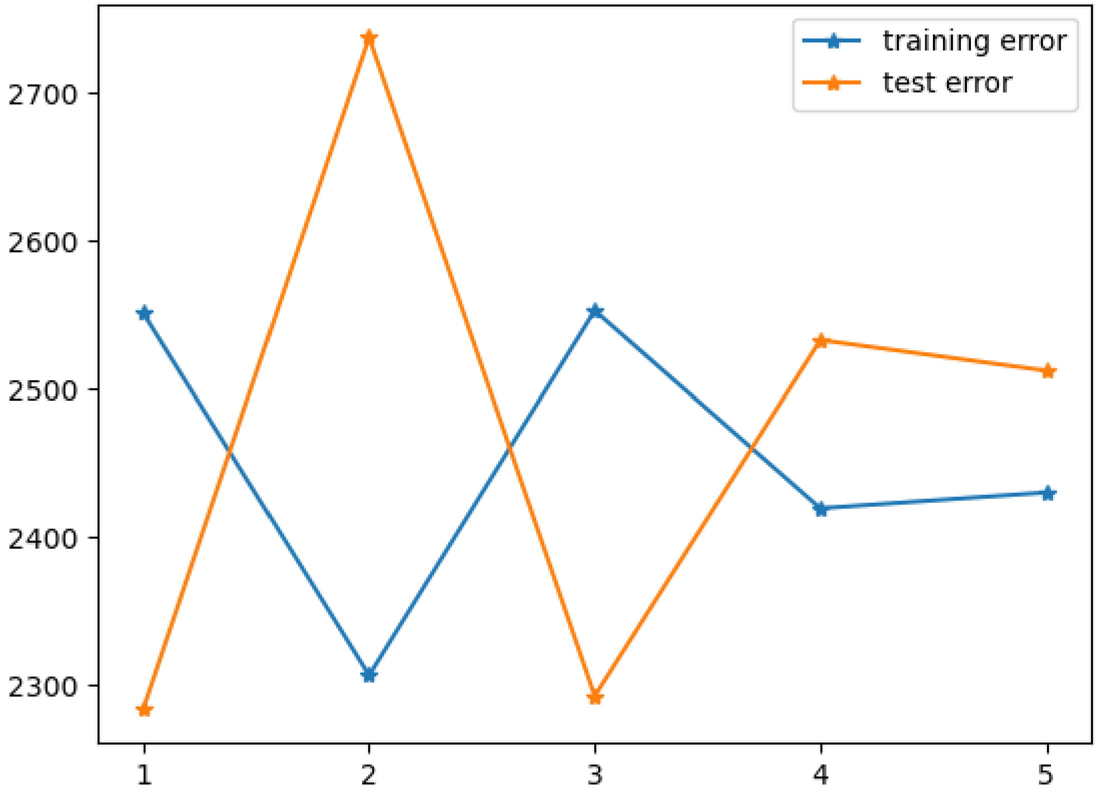
Out[87]:

	fit_time	score_time	test_score	train_score
0	[0.003833293914794922, 0.002000570297241211, 0...	[0.002069234848022461, 0.0010232925415039062, ...	[-2283.8321100715984, -2737.348623641074, -229...	[-2551.401121781797, -2306.9530822046495, -255...

In [76]:
```
import matplotlib.pyplot as plt
import numpy as np
```

In [89]:
```
plt.plot(np.arange(1,6),-result["train_score"],label="training error",ls="-",marker
plt.plot(np.arange(1,6),-result["test_score"],label="test error",ls="-",marker="*")
plt.xticks(np.arange(1,6))
plt.legend()
```

Out[89]: <matplotlib.legend.Legend at 0x23d06fef450>



In [90]:
```
result["train_score"].mean()
```

Out[90]: -2452.28467744981

In [91]:
```
result["test_score"].mean()
```

Out[91]: -2471.8996619723025