Machine Learning Methods and Applications (IST438)

HOMEWORK-1

In this homework, we will build and train a regression model for predicting insurance charges.

IMPORTING LIBRARIES

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
from statsmodels.graphics.gofplots import qqplot
from scipy.stats import shapiro
from sklearn.model_selection import train_test_split, cross_val_score, cross_val_predict
from scipy import stats as st
from sklearn.metrics import mean_absolute_error
import warnings
from sklearn.model_selection import learning_curve
import sklearn
from sklearn.metrics import mean_squared_error
```

In [2]:

```
warnings.filterwarnings("ignore")
```

READING THE DATA

In [3]:

```
df=pd.read_csv("insurance.csv")
```

In [4]:

```
df
```

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
...	...	...	...	...	...	...	...
1333	50	male	30.970	3	no	northwest	10600.54830
1334	18	female	31.920	0	no	northeast	2205.98080
1335	18	female	36.850	0	no	southeast	1629.83350
1336	21	female	25.800	0	no	southwest	2007.94500
1337	61	female	29.070	0	yes	northwest	29141.36030

1338 rows × 7 columns

INFORMATION ABOUT DATA

This dataset provides information on people's age, gender, body mass index, number of children, whether they smoke, where they live, and insurance charges.

DIMENSION and VARIABLE TYPE

In [5]:

```
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
```

Sex, smoking and living region are categorical variables. Others are numerical variables.

It consists of 1338 rows and 7 columns.

To insert categorical data into the model.

The inclusion of categorical variables in regression models is achieved by coding the variables correctly. This encoding involves converting categorical variables to a numeric value.

For nominal variables, "dummy" variables are often used.

Using Dummy Variables

In [6]:

```
encoded_df=pd.get_dummies(df,columns=['sex',"smoker","region"],drop_first=True)
```

With this code, I numericalized sex, smoker and living area.

I used "drop_first=True" to avoid falling into the dummy trap.

NEW DATA

In [7]:

```
encoded_df
```

Out[7]:

	age	bmi	children	charges	sex_male	smoker_yes	region_northwest	region_sc
0	19	27.900	0	16884.92400	0	1	0	
1	18	33.770	1	1725.55230	1	0	0	
2	28	33.000	3	4449.46200	1	0	0	
3	33	22.705	0	21984.47061	1	0	1	
4	32	28.880	0	3866.85520	1	0	1	
...	...	...	...	...	...	...	...	
1333	50	30.970	3	10600.54830	1	0	1	
1334	18	31.920	0	2205.98080	0	0	0	
1335	18	36.850	0	1629.83350	0	0	0	
1336	21	25.800	0	2007.94500	0	0	0	
1337	61	29.070	0	29141.36030	0	1	1	

1338 rows × 9 columns

NEW DIMENSION and VARIABLE TYPE

In [8]:

```
encoded_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   age               1338 non-null   int64  
 1   bmi               1338 non-null   float64
 2   children          1338 non-null   int64  
 3   charges           1338 non-null   float64
 4   sex_male          1338 non-null   uint8  
 5   smoker_yes        1338 non-null   uint8  
 6   region_northwest  1338 non-null   uint8  
 7   region_southeast  1338 non-null   uint8  
 8   region_southwest  1338 non-null   uint8  
dtypes: float64(2), int64(2), uint8(5)
memory usage: 48.5 KB
```

All columns are numerical.

It consists of 1338 rows and 9 columns.

In [9]:

```
encoded_df.isna().sum()
```

Out[9]:

```
age               0
bmi               0
children          0
charges           0
sex_male          0
smoker_yes        0
region_northwest  0
region_southeast  0
region_southwest  0
dtype: int64
```

There are no missing variables.

INFORMATION ABOUT DATA

In [10]:

```
encoded_df.describe().T
```

Out[10]:

	count	mean	std	min	25%	50%	
age	1338.0	39.207025	14.049960	18.0000	27.00000	39.000	51
bmi	1338.0	30.663397	6.098187	15.9600	26.29625	30.400	34
children	1338.0	1.094918	1.205493	0.0000	0.00000	1.000	2
charges	1338.0	13270.422265	12110.011237	1121.8739	4740.28715	9382.033	16639
sex_male	1338.0	0.505232	0.500160	0.0000	0.00000	1.000	1
smoker_yes	1338.0	0.204783	0.403694	0.0000	0.00000	0.000	0
region_northwest	1338.0	0.242900	0.428995	0.0000	0.00000	0.000	0
region_southeast	1338.0	0.272048	0.445181	0.0000	0.00000	0.000	1
region_southwest	1338.0	0.242900	0.428995	0.0000	0.00000	0.000	0

Create a Multiple Linear Regression Model

First, let's introduce the dependent and independent variables.

In [11]:

```
X=encoded_df.drop("charges",axis=1)
```

In [12]:

```
y=encoded_df[["charges"]]
```

In [13]:

```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.20, random_state=
```

Introduce Beta0 .

In [14]:

```
B0=sm.add_constant(X_train)
```

In [15]:

```
model=sm.OLS(y_train,B0).fit()
```

INFORMATION ABOUT REGRESSION MODEL

In [16]:

```
model.summary()
```

Out[16]:

OLS Regression Results

Dep. Variable:	charges	**R-squared:**	0.764
Model:	OLS	**Adj. R-squared:**	0.762
Method:	Least Squares	**F-statistic:**	428.4
Date:	Fri, 24 Mar 2023	**Prob (F-statistic):**	0.00
Time:	10:02:33	**Log-Likelihood:**	-10811.
No. Observations:	1070	**AIC:**	2.164e+04
Df Residuals:	1061	**BIC:**	2.168e+04
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-1.181e+04	1063.195	-11.107	0.000	-1.39e+04	-9722.451
age	255.4742	13.044	19.586	0.000	229.880	281.068
bmi	324.8861	30.736	10.570	0.000	264.576	385.197
children	495.5632	150.163	3.300	0.001	200.914	790.213
sex_male	125.0759	364.295	0.343	0.731	-589.745	839.897
smoker_yes	2.396e+04	452.899	52.905	0.000	2.31e+04	2.48e+04
region_northwest	-330.1951	518.472	-0.637	0.524	-1347.542	687.151
region_southeast	-811.9291	519.188	-1.564	0.118	-1830.680	206.822
region_southwest	-1093.3016	527.987	-2.071	0.039	-2129.319	-57.284

Omnibus:	251.678	**Durbin-Watson:**	1.758
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	655.012
Skew:	1.220	**Prob(JB):**	5.83e-143
Kurtosis:	5.955	**Cond. No.**	307.

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Interpretation of The Model Summary

The R-square value of the model is 0.742, showing that 74.2 percent of the independent variables are effective in explaining the variance in the dependent variable.

The F-statistic value of the model is quite high and the probability value (Prob(F-statistic)) is zero , indicating that the model is significant and at least one of the independent variables is important in explaining the variations in the dependent variable.

The Durbin-Watson value of the model was found to be 2.085, which means that the model does not exhibit autocorrelation. However, the Jarque-Bera test has shown that the error terms are not normally distributed.

When the coefficients of the independent variables are also examined, it is observed that smokers have much higher premiums than others. The variables of age and BMI also have a positive effect. However, the variables of sex and region were found to be insignificant.

As a result, this model is highly effective in estimating health insurance premiums, but the assumption of normal distribution is not provided. Therefore, work should be done to further develop and refine the model.

Editting Model

I need the transformation to correct the normality assumptions.

I start by doing a logarithmic transformation to the dependent variable.

In [17]:

```
encoded_df["charges_log"]= np.log(encoded_df["charges"]+1)
```

In [18]:

```
df=encoded_df.drop("charges",axis=1)
```

In [19]:

```
df
```

Out[19]:

	age	bmi	children	sex_male	smoker_yes	region_northwest	region_southeast	regi
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	
...	...	...	...	...	...	...	...	
1333	50	30.970	3	1	0	1	0	
1334	18	31.920	0	0	0	0	0	
1335	18	36.850	0	0	0	0	1	
1336	21	25.800	0	0	0	0	0	
1337	61	29.070	0	0	1	1	0	

1338 rows × 9 columns

In [20]:

```
independent =df.drop("charges_log",axis=1)
depend=df[["charges_log"]]
```

In [21]:

```
X_train_new, X_test_new, y_train_new, y_test_new = train_test_split(independent,depend, t
```

In [22]:

```
intercept=sm.add_constant(X_train_new)
mod=sm.OLS(y_train_new,intercept).fit()
```

In [23]:

```
mod.summary()
```

Out[23]:

OLS Regression Results

Dep. Variable:	charges_log	**R-squared:**	0.785
Model:	OLS	**Adj. R-squared:**	0.784
Method:	Least Squares	**F-statistic:**	485.0
Date:	Fri, 24 Mar 2023	**Prob (F-statistic):**	0.00
Time:	10:02:33	**Log-Likelihood:**	-608.74
No. Observations:	1070	**AIC:**	1235.
Df Residuals:	1061	**BIC:**	1280.
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	7.0011	0.077	91.090	0.000	6.850	7.152
age	0.0350	0.001	37.073	0.000	0.033	0.037
bmi	0.0131	0.002	5.917	0.000	0.009	0.018
children	0.1046	0.011	9.636	0.000	0.083	0.126
sex_male	-0.0610	0.026	-2.318	0.021	-0.113	-0.009
smoker_yes	1.5441	0.033	47.162	0.000	1.480	1.608
region_northwest	-0.0705	0.037	-1.880	0.060	-0.144	0.003
region_southeast	-0.1599	0.038	-4.260	0.000	-0.234	-0.086
region_southwest	-0.1423	0.038	-3.727	0.000	-0.217	-0.067

Omnibus:	391.968	**Durbin-Watson:**	1.970
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1574.704
Skew:	1.720	**Prob(JB):**	0.00
Kurtosis:	7.847	**Cond. No.**	307.

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

prob(JB)=0, the assumption of normality is now provided.

After the transformation, the AIC and BIC values decreased, the R square value increased.

I did not drop the "region_northwest" variable from the model because P>|t| value is very close to the 95 percent confidence interval.

Performance of The Trained Model

We can use metrics such as MSE, RMSE or MAE to measure the performance of the model.

The MAE is the average of the absolute values of the errors and has less impact on outliers.

Therefore, I find the MAE choice more appropriate as there are outliers in the dataset.

In [24]:

```
lm = LinearRegression(fit_intercept=True)
model1 = lm.fit(X_train_new, y_train_new)
```

In [25]:

```
MAE_train= mean_absolute_error(y_train_new,model1.predict(X_train_new))
MAE_test= mean_absolute_error(y_test_new,model1.predict(X_test_new))
```

In [26]:

```
MAE_train
```

Out[26]:

```
0.26501376462970866
```

In [27]:

```
MAE_test
```

Out[27]:

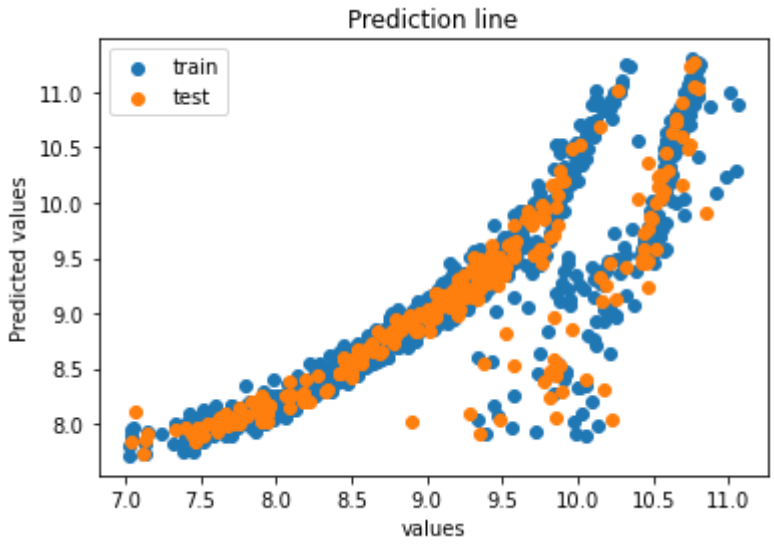
```
0.3073768584183657
```

it means that on average, the predictions are off by 0.26501376462970866 and 0.3073768584183657 units from the actual values. The lower the MAE value, the better the model's performance.

Check The Overfitting

In [28]:

```
plt.scatter(y_train_new, model1.predict(X_train_new),label='train')
plt.scatter(y_test_new, model1.predict(X_test_new),label='test')
plt.legend()
plt.xlabel("values")
plt.ylabel("Predicted values")
plt.title("Prediction line")
plt.show()
```



The model seems to be overfitting

I will test it numerically.

In [29]:

```
train_mse = mean_squared_error(y_train_new, model1.predict(X_train_new))
test_mse = mean_squared_error(y_test_new, model1.predict(X_test_new))
```

In [30]:

```
train_mse
```

Out[30]:

```
0.1826748817313811
```

In [31]:

```
test_mse
```

Out[31]:

```
0.2516120845001844
```

OVERFITTING!!!

If the training error (train_mse) is very low but the test error (test_mse) is high, we may suspect that the model is overfitting.

train_mse < test_mse

The model is overfitting.

Predict

Features I want to predict

age=24

BMI=30.2

children=0

smoker_yes=1

region_northwest=1

region_southeast=0

region_southwest=0

In [32]:

```
predict_data = [[1.0],[24], [30.2],[0],[1], [1],[1],[0], [0]]
predict_data = pd.DataFrame(predict_data).T
```

In [33]:

```
predict_data
```

Out[33]:

	0	1	2	3	4	5	6	7	8
0	1.0	24.0	30.2	0.0	1.0	1.0	1.0	0.0	0.0

In [34]:

```
predict= mod.predict(predict_data)
```

In [35]:

```
predict
```

Out[35]:

```
0    9.649783
dtype: float64
```

Since we have transformed the data before, this predict value does not reflect the truth.

For the line, we must apply the inverse of the logarithmic transformation, that is, the exponential transformation.

In [36]:

```
np.exp(predict)
```

Out[36]:

```
0    15518.413884
dtype: float64
```

According to the entered data, our predict value is 15518.413884.