Machine Learning Lab (AI332L)

Class:	BS Artificial Intelligence	**Semester:**	5th (Fall 2023)
Instructor:	Dr. M. Ehatisham-ul-Haq	**Email:**	ehtisham@mail.au.edu.pk

| Lab 02 |

Building Your First Machine Learning Model in Python

Lab Objective:

This lab tutorial will guide you through the process of building your first machine learning model using Python. We will use the popular library, scikit-learn, which provides a wide range of tools for machine learning tasks.

Setting up Python with Google Colab

1. Go to Google Colab.
2. Create a new notebook by clicking on **File** > **New Notebook**.
3. You're now ready to start writing and executing Python code in the notebook!

Part (A): Building Your First Regression Model in Python

1. Loading The Data

The dataset we are going to use includes charges from patients. First, let's import the dataset using Pandas.

```
import pandas as pd
```

Pandas is an excellent library for data loading and data preprocessing. Now, we shall load the dataset with the read_csv method.

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

Now, let's take a look at the first rows of the dataset. To do this, we are going to use the head method.

```
data.head()
```

	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

As you can see, there are 7 columns: age, sex, body mass index, number of children, smoking, region, and charges.

2. Understanding The Dataset

Understanding the data is very important before building a machine learning model. For example, let's see the number of rows and columns of the dataset. I'm going to use the shape attribute to do this.

```
data.shape
```

As you can see, the dataset has 1338 rows and 7 columns. Now, I'm going to use the info method to get more information about the dataset.

```
data.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
```

There is no missing data in the dataset. You can also use the isnull method to see the missing data.

```
data.isnull()
```

	age	sex	bmi	children	smoker	region	charges
0	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...
1333	False	False	False	False	False	False	False
1334	False	False	False	False	False	False	False
1335	False	False	False	False	False	False	False
1336	False	False	False	False	False	False	False
1337	False	False	False	False	False	False	False

1338 rows × 7 columns

Let me use the sum method to see the sum of the missing data.

```
data.isnull().sum()
```

```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

As you can see, there is no missing data in the dataset. Knowing the column types is very important for building a machine learning model. Now, let's take a look at the column types.

```
data.dtypes
```

```
age           int64
sex          object
bmi         float64
children      int64
smoker       object
region       object
charges     float64
dtype: object
```

3. Data Preprocessing

Let's convert object types to category types.

```
data['sex'] = data['sex'].astype('category')
data['region'] = data['region'].astype('category')
data['smoker'] = data['smoker'].astype('category')
```

Let's see the data types again.

```
data.dtypes
```

```
age            int64
sex         category
bmi          float64
children       int64
smoker      category
region      category
charges      float64
dtype: object
```

Now, let's go ahead and take a look at the statistics of numeric variables with the described method. If we use the transpose of the dataset, you can see the statistics better.

```
data.describe().T
```

	count	mean	std	min	25%	50%	7
age	1338.0	39.207025	14.049960	18.0000	27.00000	39.000	51.000(
bmi	1338.0	30.663397	6.098187	15.9600	26.29625	30.400	34.693
children	1338.0	1.094918	1.205493	0.0000	0.00000	1.000	2.000(
charges	1338.0	13270.422265	12110.011237	1121.8739	4740.28715	9382.033	16639.912

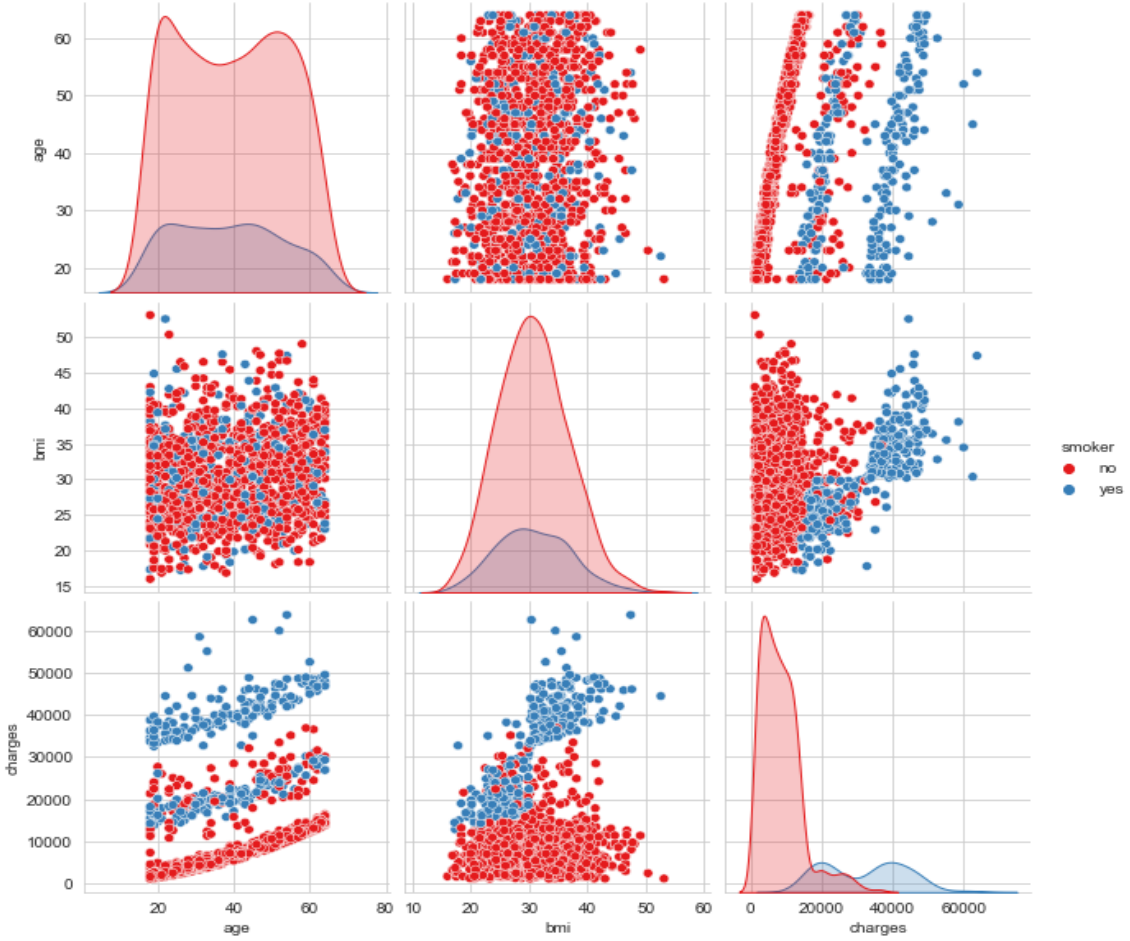
4. Data Visualization

You can understand the dataset better with data visualization. Now, let's look at the relationships of numeric variables using the seaborn. First, let me import seaborn.

```
import seaborn as sns
```

Seaborn is a library that it builds on the matplotlib, especially used for statistical plots. Now, let's choose the plot style.

```
sns.set_style("whitegrid")
sns.pairplot(
    data[["age", "bmi", "charges", "smoker"]],
    hue = "smoker",
    height = 3,
    palette = "Set1")
```



For example, smokers and non-smokers pay more when the age variable increases. Now, let's look at the correlation between the variables.

```
sns.heatmap(data.corr(), annot= True)
```

Notice that there is a relationship between charges and the other variables.

One-Hot Encoding

Now, we are going to do a one-hot encoding of the categorical variables in the dataset. This is very easy to do with Pandas. You can automatically convert categorical data into one-hot encoding using the get_dummies method in Pandas. Let's convert categorical data to one-hot encoding.

```
data = pd.get_dummies(data)
```

Thus, only categorical data were converted to one-hot encoding. Now let's look at the columns of the dataset.

```
data.columns
```

```
Index(['age', 'bmi', 'children', 'charges', 'sex_female', 'sex_male
',
       'smoker_no', 'smoker_yes', 'region_northeast', 'region_northw
est',
       'region_southeast', 'region_southwest'],
      dtype='object')
```

As you can see, new columns have been created for each subcategory. Thus, the dataset is ready to build the model. Let's go ahead and build a regression model.

5. Building a Regression Model

When building a model, you should start with the simplest model. If you don't get good accuracy, you can try more complex models. We will build a linear regression model because the output variable charges is numeric type.

Before building a machine learning model, we need to determine the input and output variables. The input variables are features. In statistics, these are called independent variables. The output variable is the target

variable. In statistics, this variable is called the dependent variable. Let's assign the target variable charges to variable **y**.

```
y = data["charges"]
```

If we drop the target variable, the remainders are the features.

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

Before the model is built, the dataset is split into training and testing. The model is built with the training data, and the model is evaluated with the test data. You can use the train_test_split method in scikit-learn to split the dataset into training and testing. With this method, you can easily split the dataset. First, let's import this method.

```
from sklearn.model_selection import train_test_split
```

Let's split the dataset into 80 percent training and 20 percent testing.

```
X_train,X_test,y_train,y_test=train_test_split(
    X,y,
    train_size = 0.80,
    random_state = 1)
```

Let's build the model after importing the linear regression class from scikit-learn.

```
from sklearn.linear_model import LinearRegression
```

Let me create an instance of the LinearRegression class.

```
lr = LinearRegression()
```

We are now going to build the model using the training data.

```
lr.fit(X_train,y_train)
```

Great!. Our model was built.

6. Model Evaluation

Let's take a look at the performance of the model. To do this, we are going to use the coefficient of determination. The closer this value is to 1, the better the model. First, let's take a look at the score of the model on the test data.

```
lr.score(X_test, y_test).round(3)#Output:
0.762
```

The coefficient of determination on the test data is greater than 0.7. Our model is not bad. Of course, it would be better if it was closer to 1. Now, let's see the score of the model on the training data.

```
lr.score(X_train, y_train).round(3)#Output:
0.748
```

As you can see, the performance of the model on the training data is close to the performance of the test data. If the performance of the model on the training data were high, it would mean that there is an overfitting problem.

Now let's take a look at another metric, mean squared error, to evaluate the model. For this, let's first predict the test data with the predict method.

```
y_pred = lr.predict(X_test)
```

Now, let's import the mean_squared_error metric.

```
from sklearn.metrics import mean_squared_error
```

We are going to use this metric now. First, let's import the math module to calculate the square root of this metric.

```
import math
```

Let's take a look at the square root of the mean squares error.

```
math.sqrt(mean_squared_error(y_test, y_pred))#Output:
5956.45
```

This value means that the model predicts with a standard deviation of 5956.45.

7. Model Prediction

Now, I'm going to predict the first row as an example. First, let's select the first row of the training data.

```
data_new = X_train[:1]
```

Let us predict the data with our model.

```
lr.predict(data_new)#Output:
10508. 42
```

Let's take a look at the real value.

```
y_train[:1]#Output:
10355.64
```

As you can see, our model predicted close to the real value.

Part (B): Building Your First Classification Model in Python

First, you need to import the needed libraries for the present tutorial.

```
from sklearn import datasets, model_selection, metrics, tree

import pandas as pd
```

As a side note, if you are trying to configure a computer for ML, you need to install all these dependencies on top of configuring Python.

Step 1. The Working Dataset:

We will use the red wine quality dataset for the current tutorial. The wine dataset is often used as an example for showing ML models, and thus, it is conveniently available in the sklearn and can be loaded easily.

wine_data = datasets.load_wine()

```
df = pd.DataFrame(wine_data["data"], columns=wine_data["feature_names"])
df.head()
```

The above code shows you the features of the data. In ML, we use “features” to study what factors may be important for the correct prediction.

You can see that there are 12 features available which are potentially important for the quality of the red wine, such as alcohol and malic acid.

One specific ML is concerned with classification. Each data record has a label showing its class, and the classes of all the records are known as the “target” of the dataset. In the red wine data set, there are three classes for the labels, and we can check the labels, as shown below:

```
target = wine_data["target"]
```

Step 2. Training the Model:

The next step is train the ML model. The purpose of training an ML model is more or less about making predictions on things that they’ve never seen. The model is about how to make good predictions. The way to create a model is called training — using existing data to identify a proper way to make predictions.

To test the model’s performance, we will split the dataset into two parts, one for training and the other for testing. We can simply use the train_test_split method, as shown below.

```
X_train, X_test, y_train, y_test
= model_selection.train_test_split(df, target, test_size=0.2, random_state=0)
print("Training Dataset:", X_train.shape)
print("Test Dataset:", X_test.shape)
```

The training dataset has 142 records, while the test dataset has 36 records, approximately in a ratio of 4:1 (note that test_size=0.2 meaning 20% (with round-ups if needed) of the original dataset are used for testing). You can use the split size of your own choice as well.

There are many ways to build a model, such as Naïve bayes, decision trees, K-nearest neighbors, SVM, random forest, and gradient boosting, just to name a few. For the purpose of the present tutorial showing you how to build an ML model using Google Colab, we will just use a model that is readily available in sklearn — the decision tree classifier.

```
classifier = tree.DecisionTreeClassifier()
model = classifier.fit(X_train, y_train)
model
```

For predicting the test data, you can use the following:

```
y_pred = model.predict(X_test)
report = metrics.classification_report(y_test, y_pred)
print(report)

model.score(X_test, y_test)
```

Lab Task:

Problem 01: You have to develop an ML-based model for predicting a student's exam score (0 to 100) in the Machine Learning course. What are the factors you would consider for training such a model? How can those factors be extracted as features? Propose how you will predict the **student's score** based on the above-mentioned factors/features.

Problem 02: We have a problem of predicting a Mobile SIM Card price category based on the contact number. The format of an 11-digit SIM card contact number is as follows: **03XXXXXXXXX.** Suppose you have a large list of SIM Card numbers and their prices, and you need to train an ML algorithm that can predict the SIM price category as **low**, **moderate**, or **high** based on the SIM Card contact number.

Considering the above-stated problems, develop two datasets of your own choice (one for regression and the other for classification problem), preferably in .csv format. The datasets should be in the form of features. You can pre-process the dataset files manually if needed. Load these datasets into Google Colab using the Python code and apply the above-described tutorials to the chosen datasets.

Prepare a one-page report on the output results and analysis.