ASSIGNMENT : MODULE 3

Submitted by,

Chinmaya .A.S

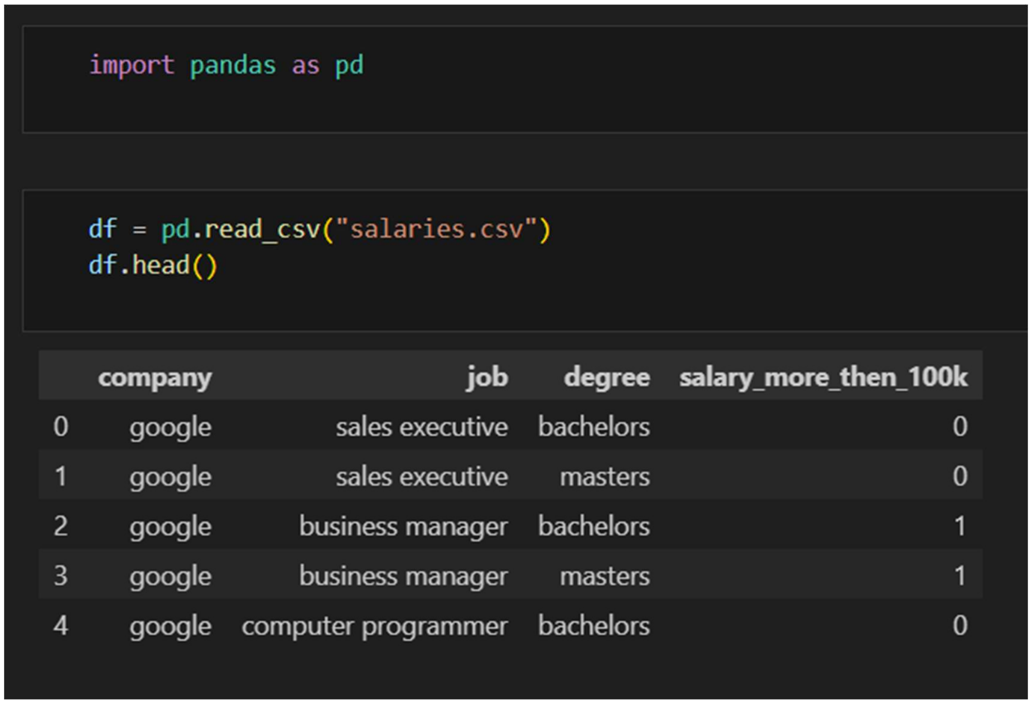
DECISION TREE : SALARY PREDICTION

DOCUMENTATION

INTRODUCTION

This project demonstrates a simple machine learning workflow to predict whether a person's salary is greater than 100,000 based on their company, job role, and educational qualification. Since the input data is categorical, it is first converted into numerical form using label encoding so that it can be processed by a machine learning algorithm.
A Decision Tree classifier is then trained on the transformed data to learn rule-based patterns that distinguish high-salary and low-salary cases. The project focuses on understanding how categorical data is prepared, how a model is trained, and how predictions are made, making it suitable for learning, revision, and basic interview preparation.

1. Import required libraries and load data set



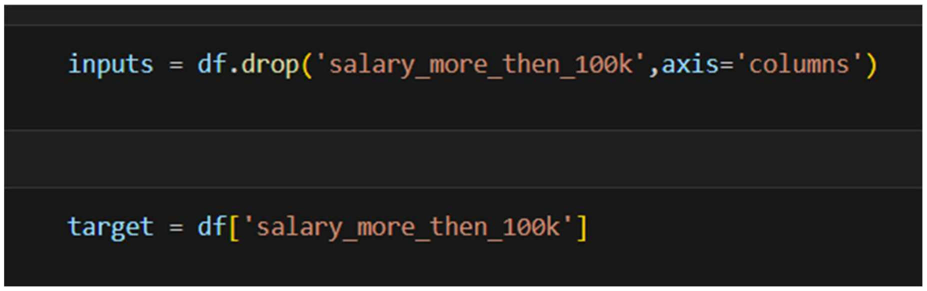
```
import pandas as pd
```

```
df = pd.read_csv("salaries.csv")
df.head()
```

```
   company                  job     degree  salary_more_then_100k
0   google      sales executive  bachelors                      0
1   google      sales executive    masters                      0
2   google     business manager  bachelors                      1
3   google     business manager    masters                      1
4   google  computer programmer  bachelors                      0
```

2. Separating input features and the target variable

Once the data is loaded, the code separates it into two logical parts: input features and the target variable. The input features consist of attributes that describe a person (company, job, degree), while the target variable represents the outcome we want the model to predict—whether the salary exceeds 100k or not. This separation is fundamental to supervised learning and ensures the model learns patterns from inputs without accidentally seeing the correct answer during prediction.

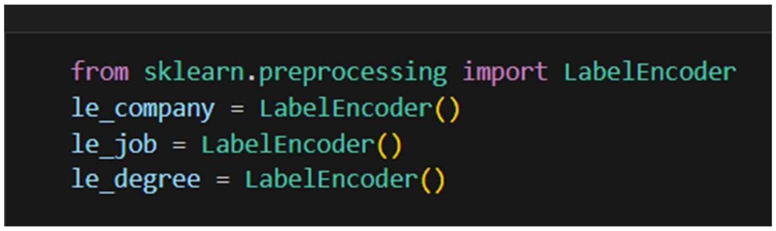


```
inputs = df.drop('salary_more_then_100k',axis='columns')
```

```
target = df['salary_more_then_100k']
```

3. Creating label encoders for each categorical feature

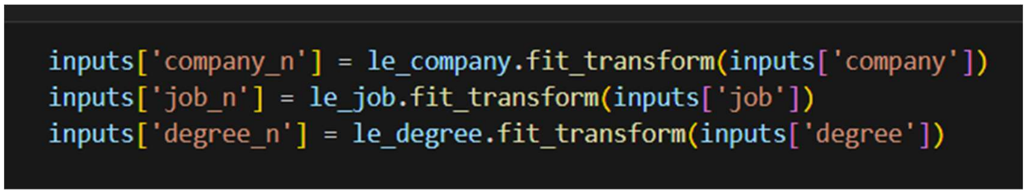
A LabelEncoder is used to convert categorical (text) values into numerical form, which is required because machine learning models cannot work directly with strings. By creating individual encoders for each column, the code ensures that each feature learns its own independent mapping between category names and numeric labels. This prevents different features from sharing or mixing category values and keeps the encoding consistent and interpretable.



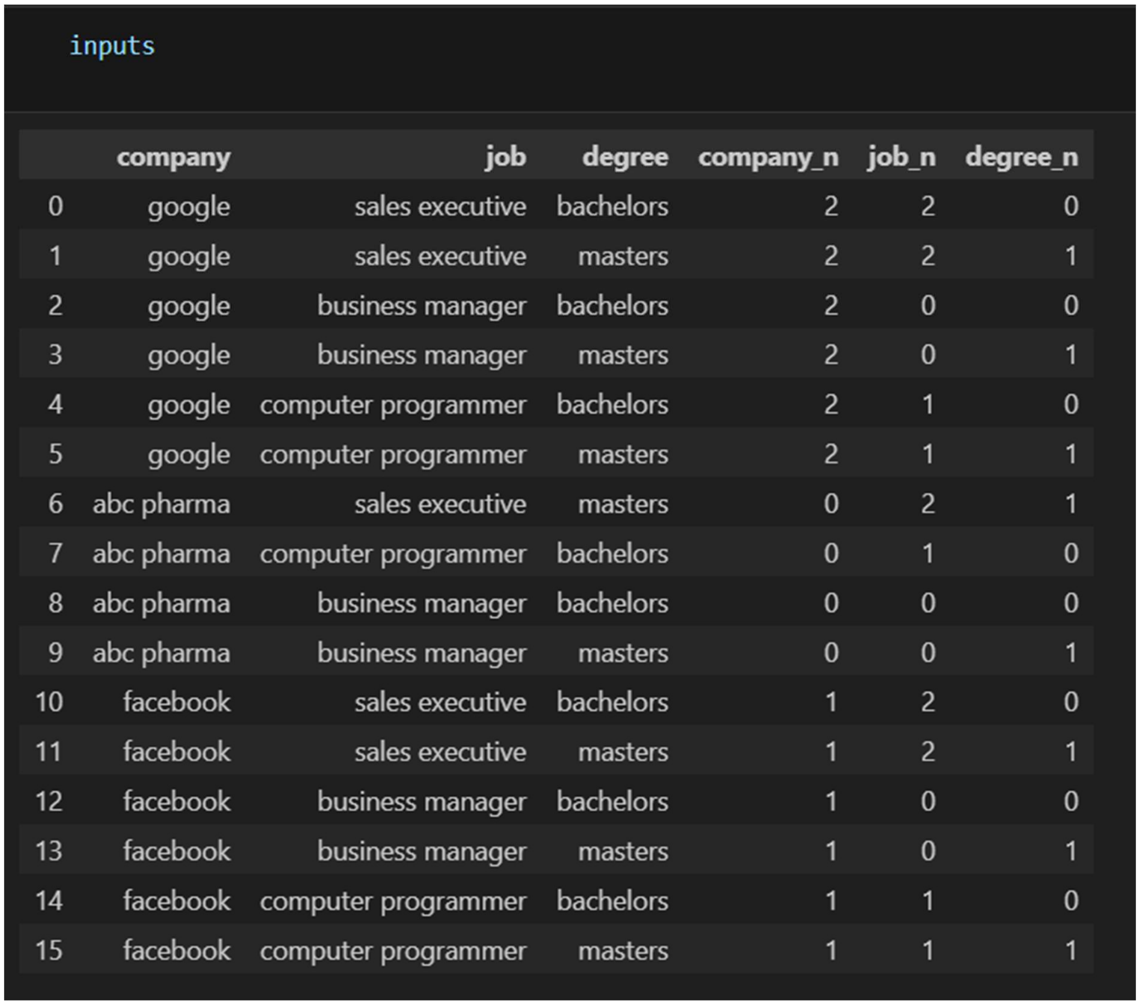
```python
from sklearn.preprocessing import LabelEncoder
le_company = LabelEncoder()
le_job = LabelEncoder()
le_degree = LabelEncoder()
```

4. Encoding Categorical Variables Using Label Encoding

Here, three separate LabelEncoder objects (le_company, le_job, and le_degree) are used—one for each column. The fit_transform() method first identifies all unique categories in a column and then assigns a unique integer value to each category. For example, company names like *Google*, *Facebook*, and *Microsoft* might be encoded as 0, 1, and 2. The resulting numerical values are then stored in new columns: company_n, job_n, and degree_n, while the original text columns remain unchanged.



```python
inputs['company_n'] = le_company.fit_transform(inputs['company'])
inputs['job_n'] = le_job.fit_transform(inputs['job'])
inputs['degree_n'] = le_degree.fit_transform(inputs['degree'])
```

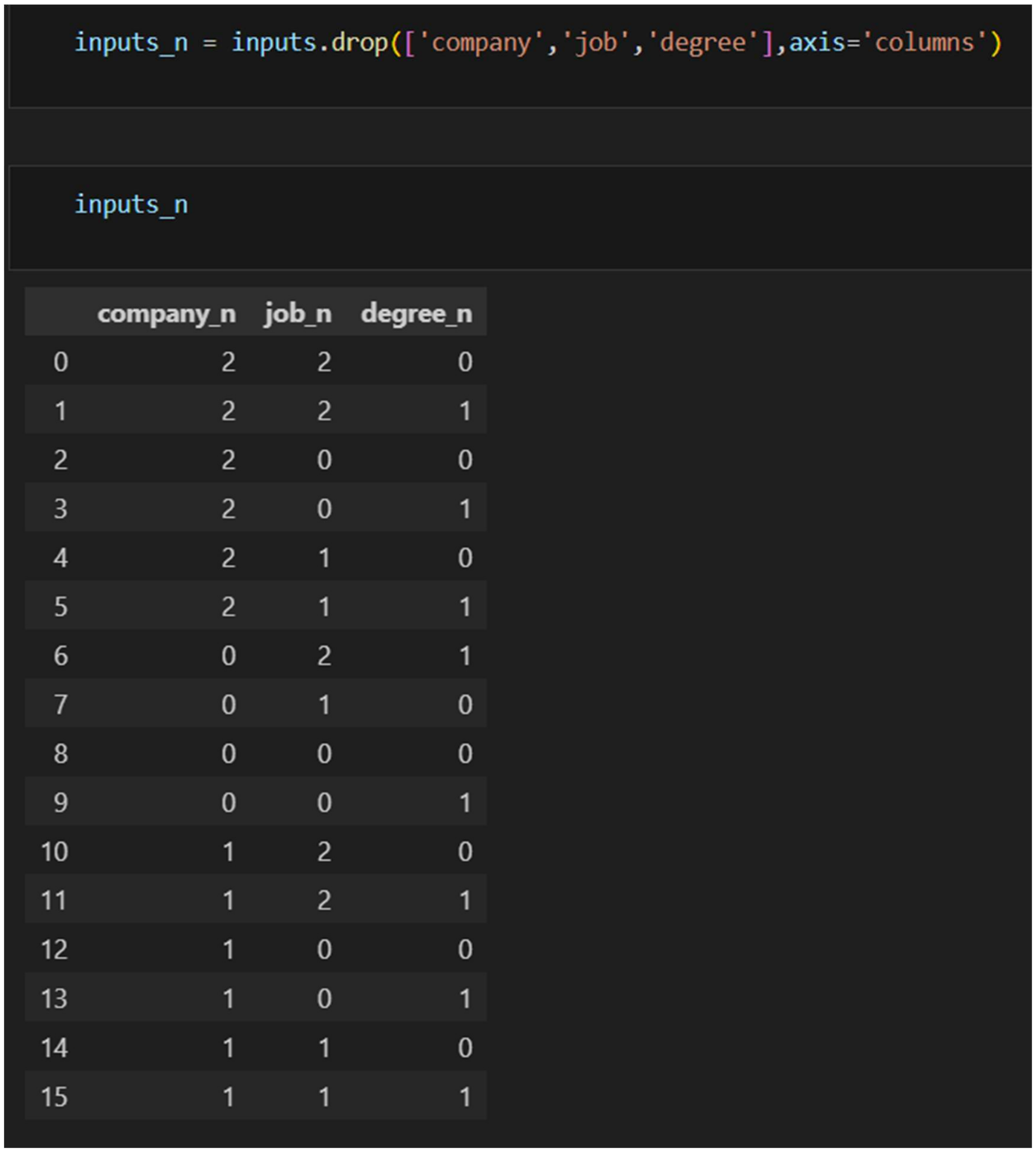


```
inputs
```

	company	job	degree	company_n	job_n	degree_n
0	google	sales executive	bachelors	2	2	0
1	google	sales executive	masters	2	2	1
2	google	business manager	bachelors	2	0	0
3	google	business manager	masters	2	0	1
4	google	computer programmer	bachelors	2	1	0
5	google	computer programmer	masters	2	1	1
6	abc pharma	sales executive	masters	0	2	1
7	abc pharma	computer programmer	bachelors	0	1	0
8	abc pharma	business manager	bachelors	0	0	0
9	abc pharma	business manager	masters	0	0	1
10	facebook	sales executive	bachelors	1	2	0
11	facebook	sales executive	masters	1	2	1
12	facebook	business manager	bachelors	1	0	0
13	facebook	business manager	masters	1	0	1
14	facebook	computer programmer	bachelors	1	1	0
15	facebook	computer programmer	masters	1	1	1

6. Removing Original Categorical Columns After Encoding

In this step, the original categorical columns company, job, and degree are removed from the dataset using the drop() method. These columns contain text values, which are no longer needed because they have already been converted into numerical form through label encoding in the previous step.The parameter axis='columns' specifies that columns (and not rows) are being removed. The resulting dataset, stored in inputs_n, now contains only numerical features, including the newly created encoded columns (company_n, job_n, degree_n). This ensures that the final input dataset is fully numeric and ready to be used by the machine learning model for training or prediction.



```
inputs_n = inputs.drop(['company','job','degree'],axis='columns')
```

```
inputs_n
```

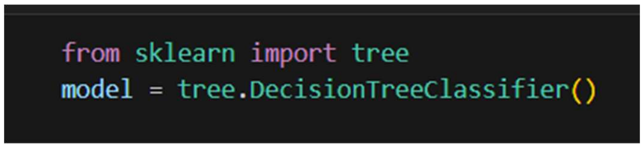
```
    company_n  job_n  degree_n
0           2      2         0
1           2      2         1
2           2      0         0
3           2      0         1
4           2      1         0
5           2      1         1
6           0      2         1
7           0      1         0
8           0      0         0
9           0      0         1
10          1      2         0
11          1      2         1
12          1      0         0
13          1      0         1
14          1      1         0
15          1      1         1
```

7. Initializing the Decision Tree Classification Model

In this step, the Decision Tree classifier from the sklearn.tree module is imported and instantiated. A Decision Tree classifier is a supervised machine learning algorithm used for classification tasks, where the target variable consists of discrete class labels.

By creating model = DecisionTreeClassifier(), an empty Decision Tree model object is initialized with default hyperparameters. This model works by learning a series of decision rules from the input features, splitting the data at each node based on conditions that best separate the target classes. Once trained, the model can classify new data by following these learned decision paths from the root to a leaf node.

This step prepares the model for the next phase, where it will be trained using labeled data.



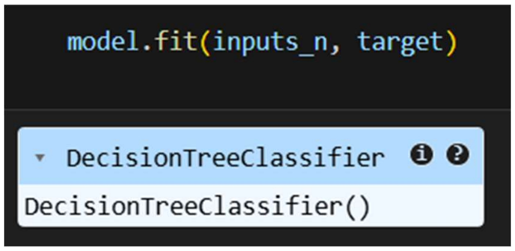
```
from sklearn import tree
model = tree.DecisionTreeClassifier()
```

8. Training the Decision Tree Model

In this step, the Decision Tree classifier is trained using the prepared dataset. The fit() method takes two inputs: inputs_n, which contains the numerical feature set, and target, which represents the output variable (class labels) that the model is expected to predict.

During training, the Decision Tree algorithm analyzes the input features and learns decision rules that best split the data to correctly classify the target values. It does this by selecting features and split points that minimize impurity (such as Gini impurity or entropy) at each node. As a result, the model builds a tree-like structure that captures patterns and relationships between the input variables and the target.

After this step, the model is fully trained and ready to be used for making predictions on new or unseen data.



```
model.fit(inputs_n, target)
```

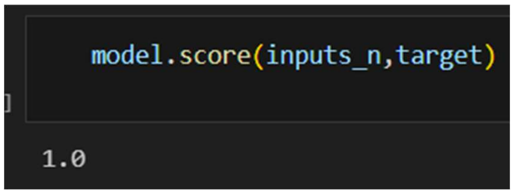
```
DecisionTreeClassifier
DecisionTreeClassifier()
```

9. Evaluating Model Accuracy on the Training Data

In this step, the score() method is used to evaluate the performance of the trained Decision Tree model on the given dataset. The method compares the model's predicted class labels with the actual values in the target variable.

For classification models like DecisionTreeClassifier, the score() method returns the accuracy, which is the proportion of correctly classified samples out of the total number of samples. The value ranges from 0 to 1, where a higher value indicates better performance.

Here, the model is being evaluated on the same data used for training (inputs_n and target), so a very high or even perfect score may indicate that the model has learned the training data very well. However, this does not necessarily mean the model will perform equally well on unseen data, as Decision Trees are prone to overfitting when evaluated only on training data.



```
model.score(inputs_n,target)
```

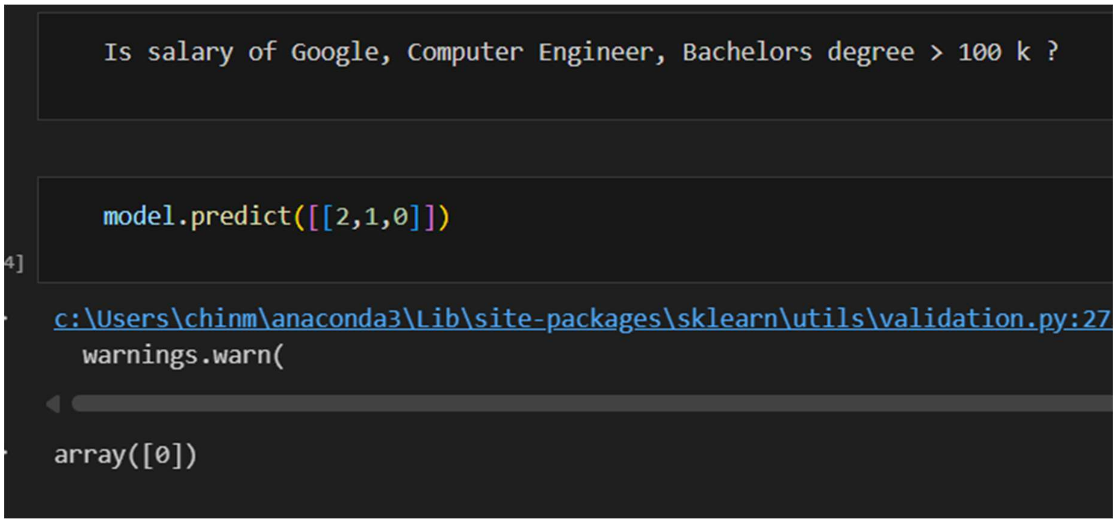
```
1.0
```

10. Making Predictions with the Trained Model

In this step, the predict() method is used to make a prediction for a new, unseen data point. The input [[2, 1, 0]] represents a single sample with three numerical features, corresponding to the encoded versions of company_n, job_n, and degree_n.

The model takes these feature values and traverses the decision tree according to the rules it learned during training, ultimately reaching a leaf node that contains a predicted class. The output of predict() is the predicted class label for this input.

This allows you to see what the model thinks the target value should be for a new combination of features, demonstrating how the trained Decision Tree can generalize to new data points.

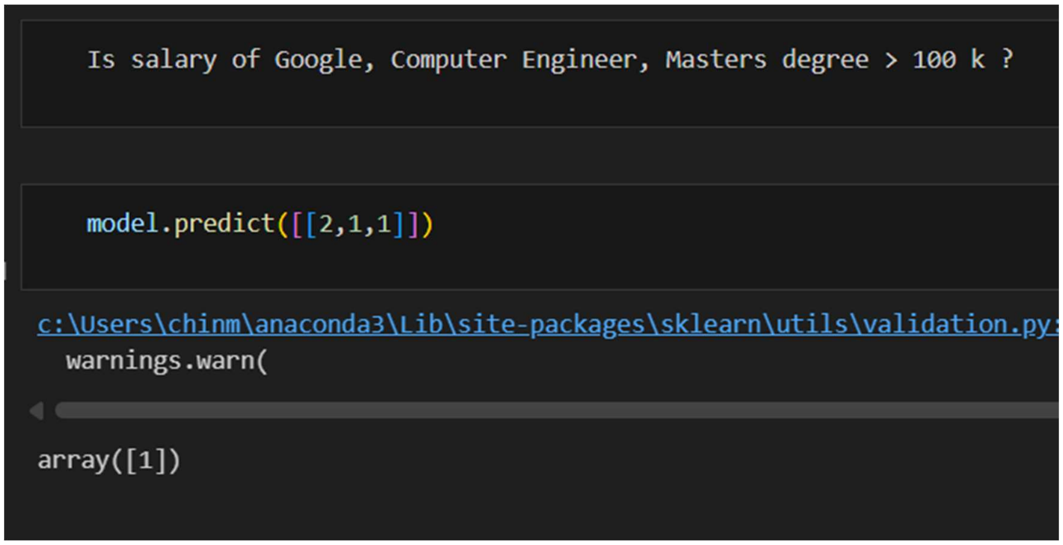


```
Is salary of Google, Computer Engineer, Bachelors degree > 100 k ?
```

```
model.predict([[2,1,0]])
```

```
4]
c:\Users\chinm\anaconda3\Lib\site-packages\sklearn\utils\validation.py:27
  warnings.warn(

array([0])
```



```
Is salary of Google, Computer Engineer, Masters degree > 100 k ?
```

```
model.predict([[2,1,1]])
```

```
c:\Users\chinm\anaconda3\Lib\site-packages\sklearn\utils\validation.py:
  warnings.warn(

array([1])
```