Guide To Machine learning

I. Using the appropriate library to read CSV

In particular, reading CSV is very easy in python can be done locally without any libraries. however, when dealing with machine learning. it is best if you use some popular libraries to help you.

Pandas:	NumPy:
Being an **extremely popular** library to use when coding out machine learning project and is most likely the library you are going to use. Pandas o ers tons of functionality to read and write CSV files. It provides a Data Frame object that is a twodimensional table-like data structure that can be used to manipulate data. Pandas is widely used for data preprocessing in machine learning tasks.	NumPy is a numerical computing library in Python that provides e cient multi-dimensional array operations. NumPy provides a loadtxt() function that can be used to read CSV files into NumPy arrays.
SciPy:	**Scikit-learn:**
SciPy is a scientific computing library in Python that provides many numerical algorithms and functions. It includes a submodule called scipy.io that provides the loadmat() function that can be used to read MATLAB files into NumPy arrays. This can be useful if your CSV files were exported from MATLAB.	Scikit-learn is a popular machine learning library in Python that provides many algorithms and tools for machine learning tasks. It includes functionality to load datasets from CSV files using the load_csv() function.

II. Cleaning of Data

Check your data

Always ensure that the data makes sense to you. If you are unsure always ask your teachers if there is any concern. in particular, **always clarify** how to handle things like missing data. or special characters corrupting the data causing it to be invalid.

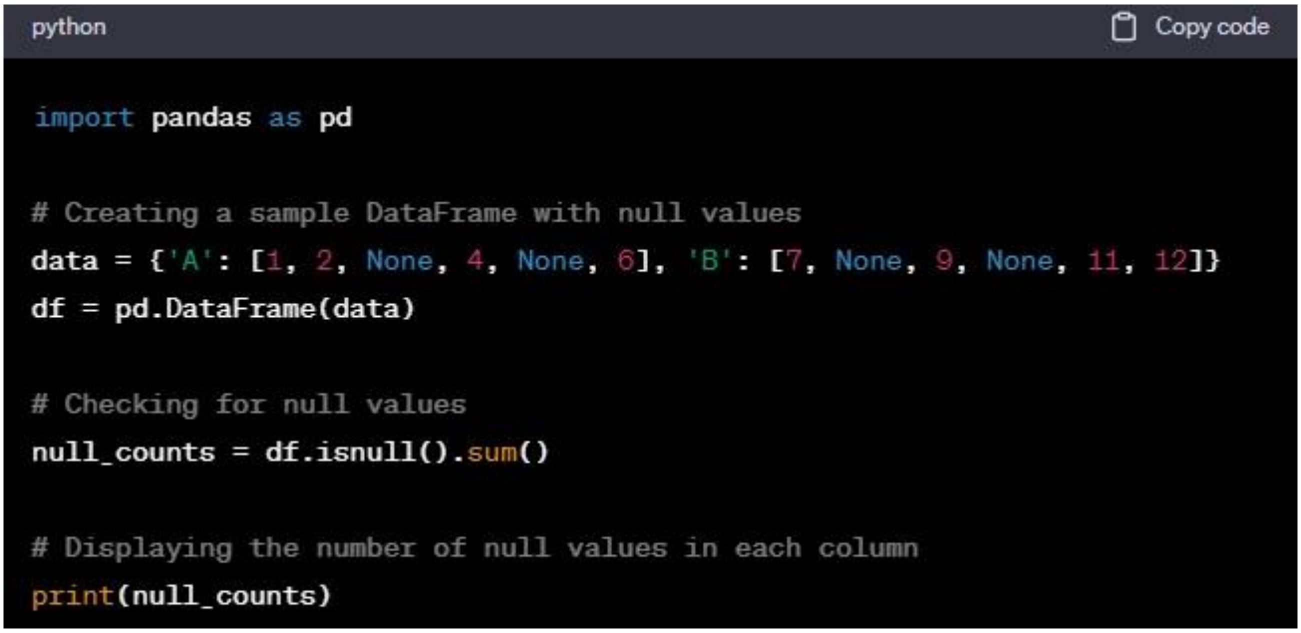
Handling Data

[redacted]	Null Values	Unknown Values
● First make sure to check with your teacher if the data truly	● Empty Data.	[redacted] idk what this means. but imma assume

is invalid		● Honestly, just check. if something doesn't make sense. **ask ur teacher** if this is a problem that requires a solution
● Do you need to replace it? or remove it. perhaps remove column? See how you might wanna handle it	● There are many solutions. therefore check with your personal logic as well as the teacher	● you may need to transform the data into a di erent format or representation
● removing outliers or erroneous data points, or imputing missing values using a statistical method. are all viable options	● check if Null first. usually done using the isna().sum() function on your dataframe	
	● replacing missing values with a 0 is **possible** solution	

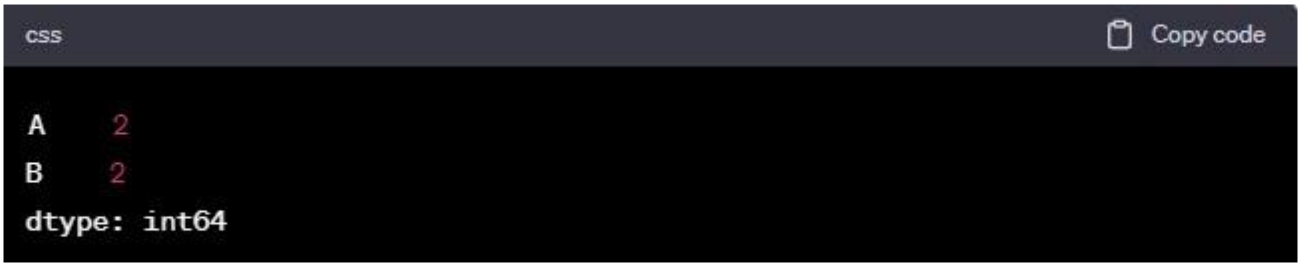
by using .fillna()

using isna().sum() will tell you the number of null values in each columns



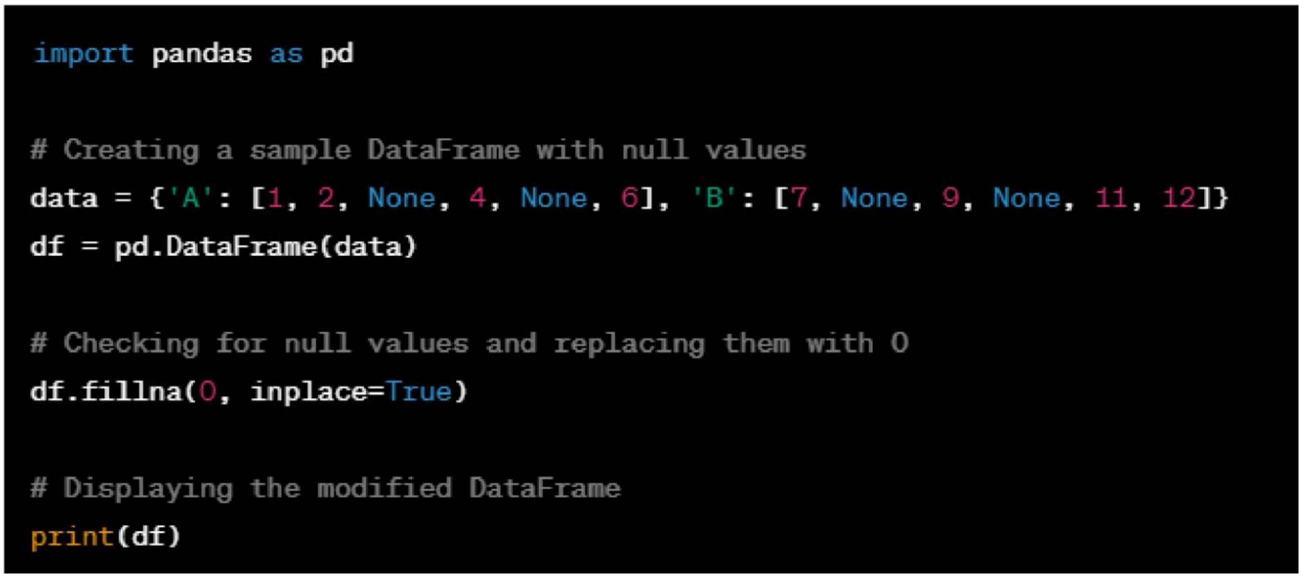
```python
import pandas as pd

# Creating a sample DataFrame with null values
data = {'A': [1, 2, None, 4, None, 6], 'B': [7, None, 9, None, 11, 12]}
df = pd.DataFrame(data)

# Checking for null values
null_counts = df.isnull().sum()

# Displaying the number of null values in each column
print(null_counts)
```

and the output will be:



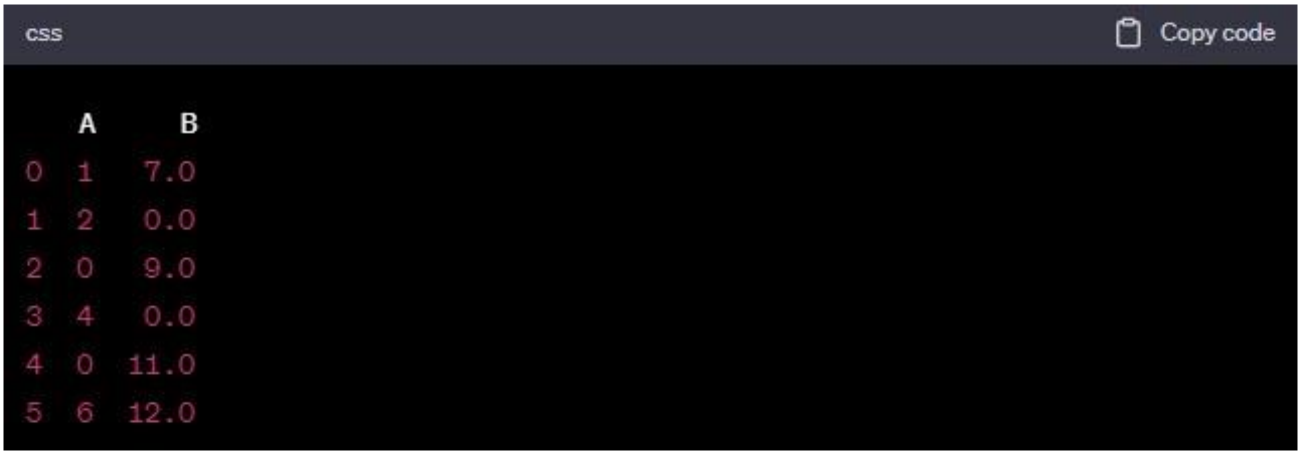
```
A    2
B    2
dtype: int64
```

One of the solution to handling missing or null values is by replacing them with 0. this is how you may handle it:



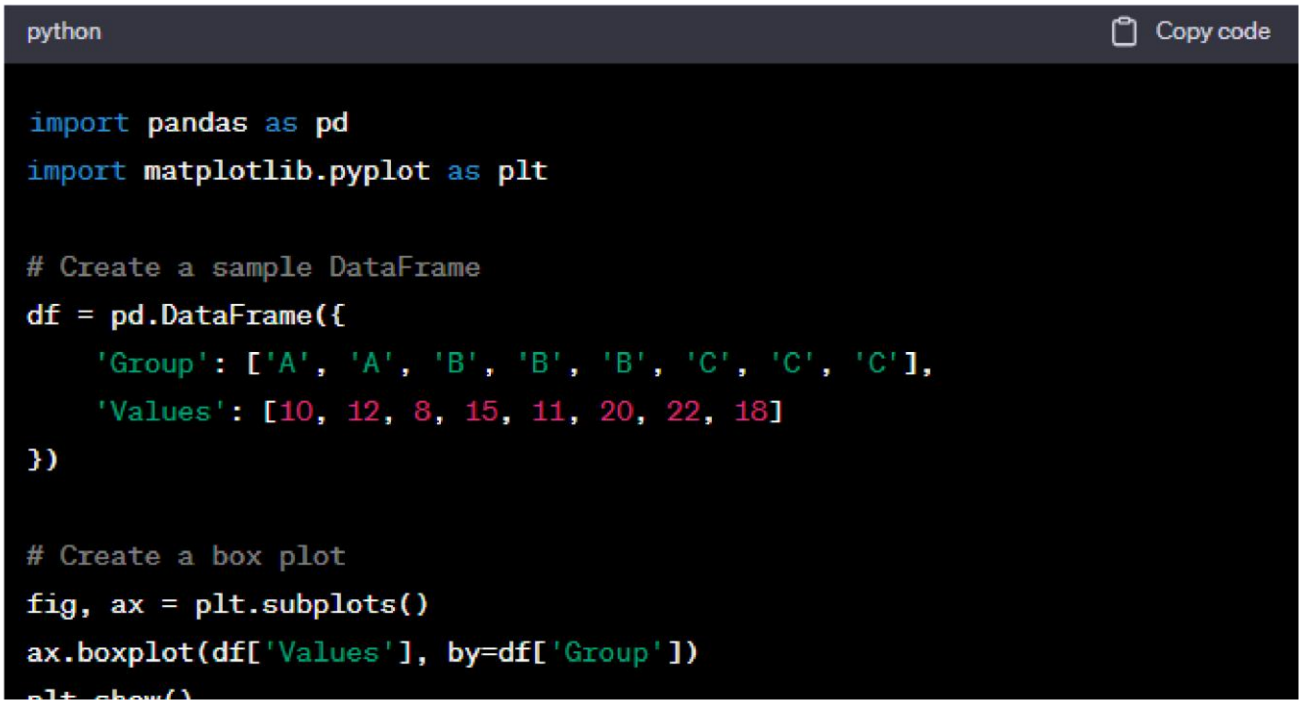
```python
import pandas as pd

# Creating a sample DataFrame with null values
data = {'A': [1, 2, None, 4, None, 6], 'B': [7, None, 9, None, 11, 12]}
df = pd.DataFrame(data)

# Checking for null values and replacing them with 0
df.fillna(0, inplace=True)

# Displaying the modified DataFrame
print(df)
```

and the output will be:



```
   A    B
0  1   7.0
1  2   0.0
2  0   9.0
3  4   0.0
4  0  11.0
5  6  12.0
```

you can use charts from matplotlib library to try to identify any other data that may cause your machine learning to go o the charts. such as using a box chart to detect outliers



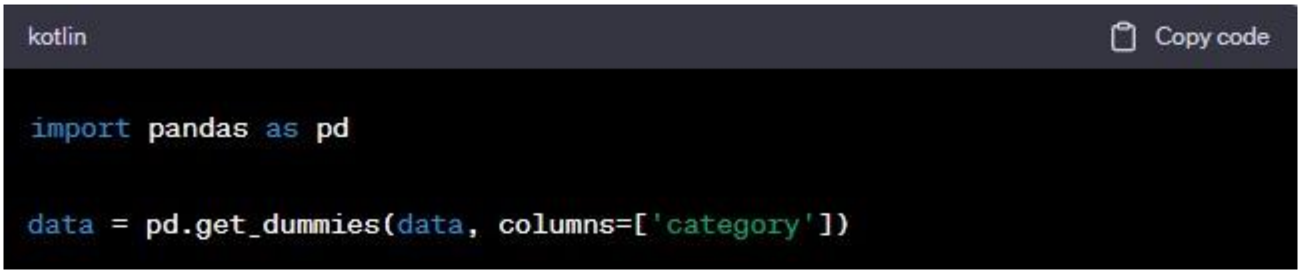
```python
import pandas as pd
import matplotlib.pyplot as plt

# Create a sample DataFrame
df = pd.DataFrame({
    'Group': ['A', 'A', 'B', 'B', 'B', 'C', 'C', 'C'],
    'Values': [10, 12, 8, 15, 11, 20, 22, 18]
})

# Create a box plot
fig, ax = plt.subplots()
ax.boxplot(df['Values'], by=df['Group'])
plt.show()
```

```
plt.show()
```

III. Convert categorical columns to numeric values

One-Hot Encoding:

This method allows you to create multiple binary columns for each category. The value 1 is assigned to the column corresponding to the category, and 0 is assigned to all other columns. The pandas library provides the get_dummies() function that can be used for this purpose.

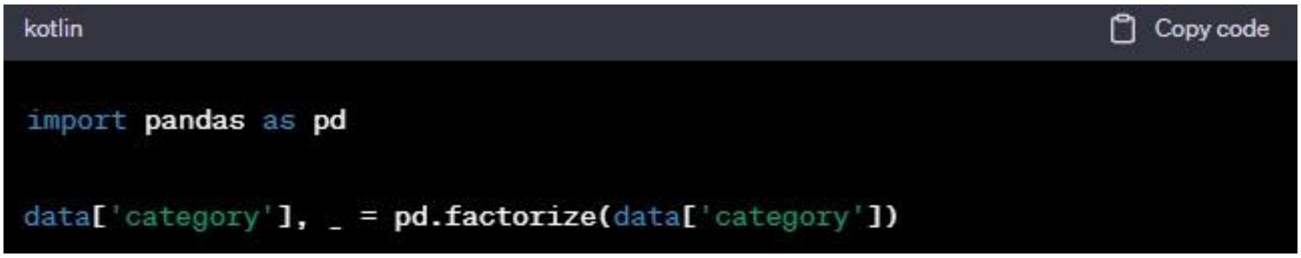


```
import pandas as pd

data = pd.get_dummies(data, columns=['category'])
```

Factorize

The pandas library provides the factorize() function for this purpose. Factorizing maps each unique category in a categorical column to a unique integer value, similar to label encoding.

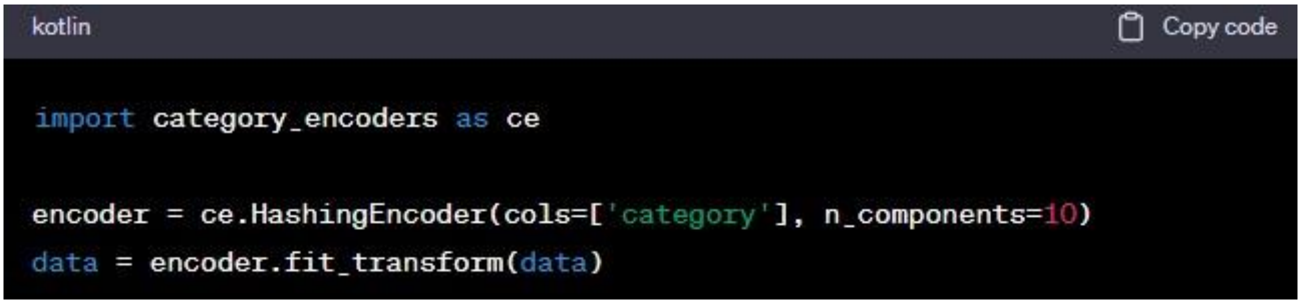
One potential drawback of factorizing is that it can result in non-contiguous integer values, which may be problematic for some machine learning algorithms. In contrast, label encoding guarantees that the integer values are contiguous.



```
import pandas as pd

data['category'], _ = pd.factorize(data['category'])
```

Hashing Encoding

This approach maps each unique category in a categorical column to a numeric value using a hashing function. The category is hashed into a fixed-length integer value, which is used as the encoded value. The category_encoders library provides the HashingEncoder class that can be used for this purpose.



```
import category_encoders as ce

encoder = ce.HashingEncoder(cols=['category'], n_components=10)
data = encoder.fit_transform(data)
```

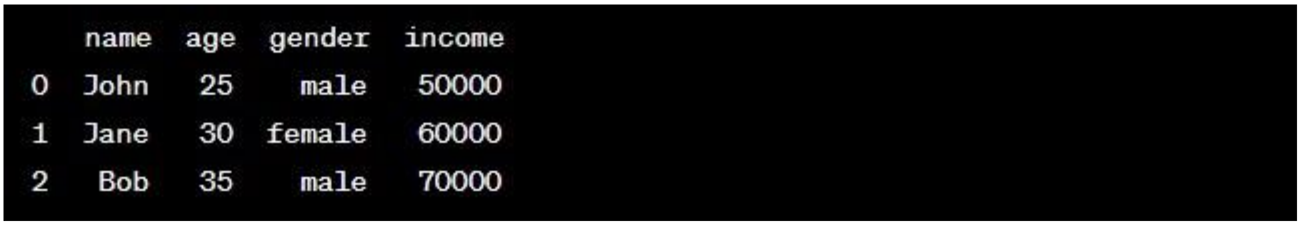
IV. Choose the appropriate columns as feature columns

Of course, this part should be relatively obvious after u cleared the steps above. in fact this should've been before step 2.

To put it simply, choose the columns that you think can be related to the answer.

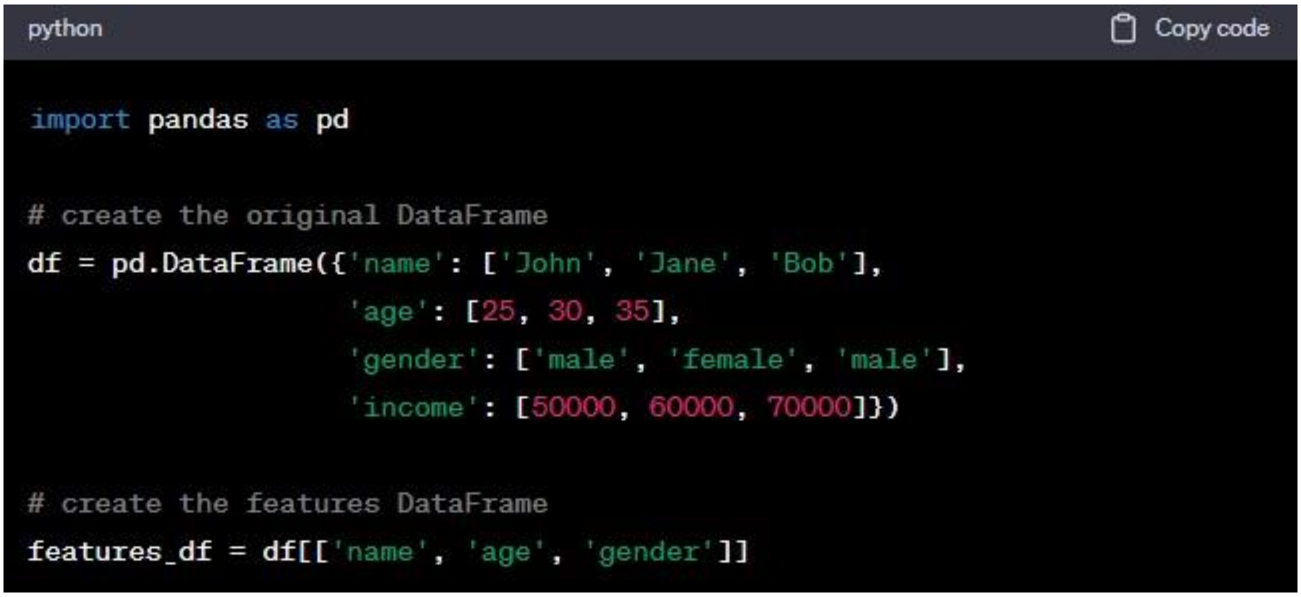
How to achieve this is:

Suppose we have a Data Frame called df that looks like this:



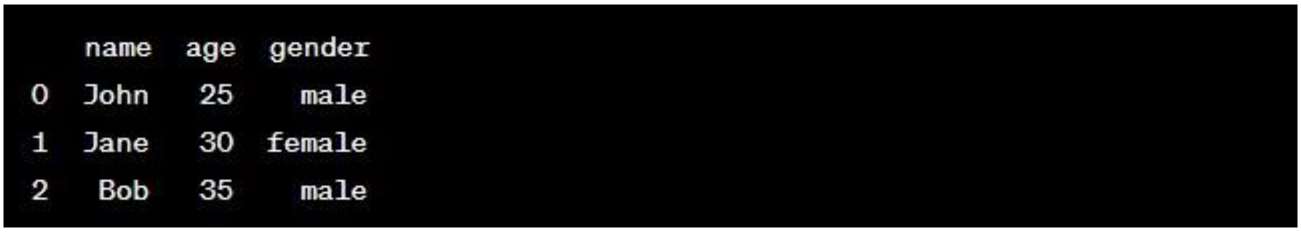
```
   name  age  gender  income
0  John   25    male   50000
1  Jane   30  female   60000
2   Bob   35    male   70000
```

We want to separate the name, age, and gender columns into a features_df DataFrame, and leave the income column out.



```python
import pandas as pd

# create the original DataFrame
df = pd.DataFrame({'name': ['John', 'Jane', 'Bob'],
                   'age': [25, 30, 35],
                   'gender': ['male', 'female', 'male'],
                   'income': [50000, 60000, 70000]})

# create the features DataFrame
features_df = df[['name', 'age', 'gender']]
```

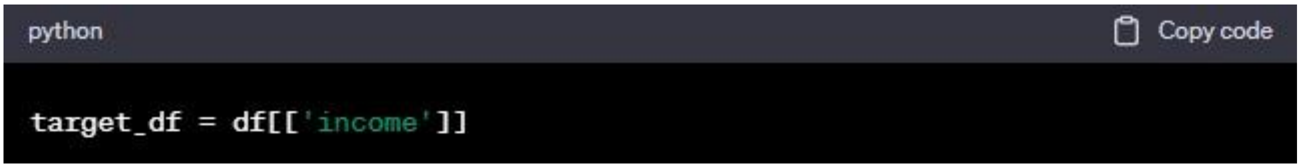
The features_df DataFrame will look like this:



```
   name  age  gender
0  John   25    male
1  Jane   30  female
2   Bob   35    male
```

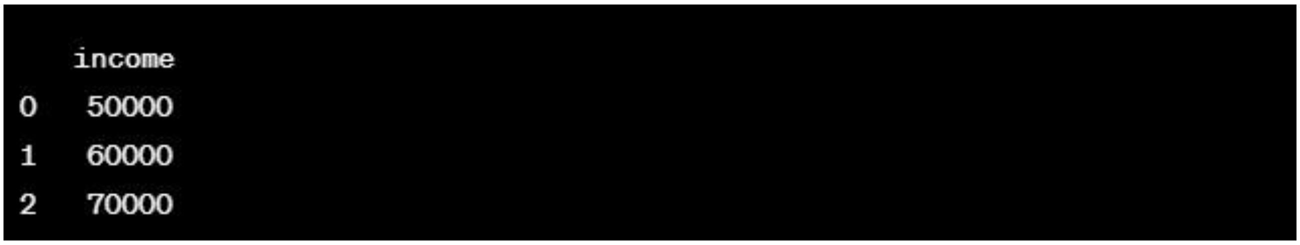
V. Choose the appropriate columns as target columns

Following the example above. Now we handle the target column. this is done very similarly. **let's say that the income column is our target column. We can create a target_df that includes only the income column like this:**



```python
target_df = df[['income']]
```

This will give us a new DataFrame target_df that looks like this



```
   income
0   50000
1   60000
2   70000
```

VI. Choose at least 2 appropriate machine learning algorithm

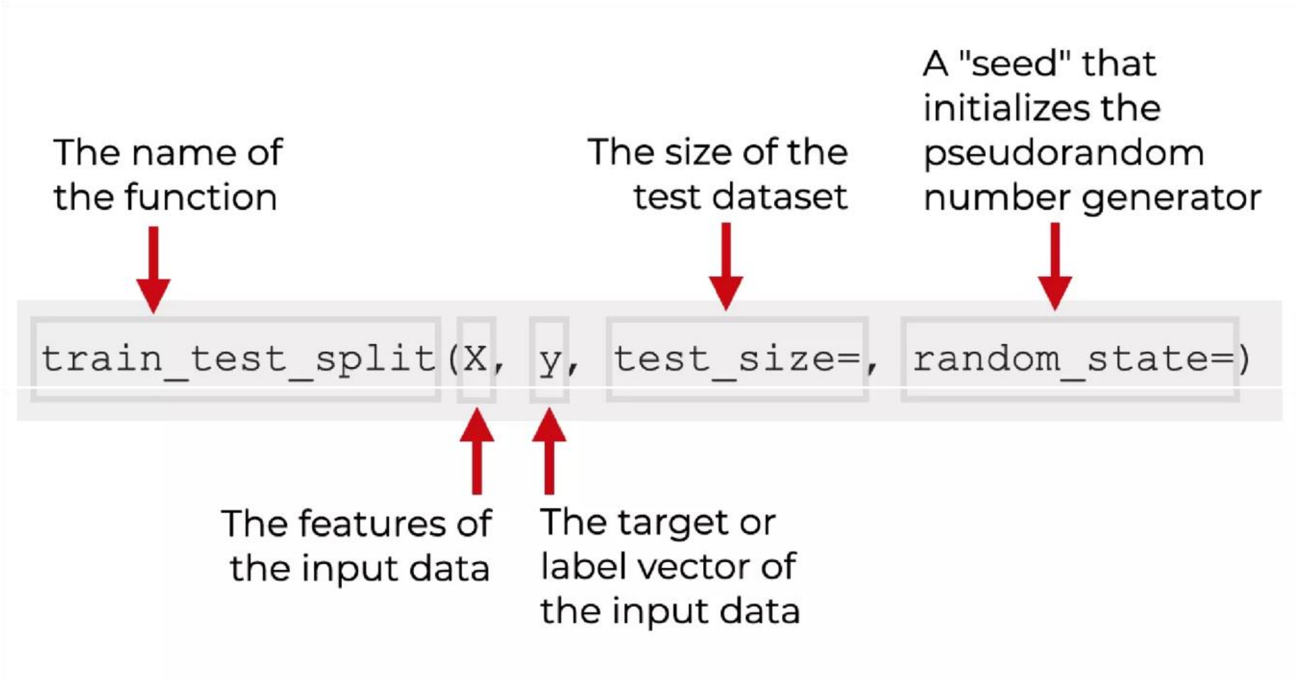
If the target variable is **categorical**, you need to use classification algorithms such as logistic regression, decision trees, random forests, or support vector machines (SVMs). If the target variable is continuous, you need to use **regression** algorithms such as linear regression, polynomial regression,

The size and complexity of the dataset can also influence your choice of machine learning algorithm. you have a small dataset, you may want to use simpler algorithms such as logistic regression or decision trees.

The performance metrics that you want to optimize can also influence your choice of machine learning algorithm. if you want to optimize accuracy, you may want to use algorithms such as logistic regression, decision trees, or random forests.

So find the best models for your training from your lab sheet. or maybe be **adventurous** if your teacher allows it

VII. Select training data and test data from dataset


The name of the function
The size of the test dataset
A "seed" that initializes the pseudorandom number generator
train_test_split(X, y, test_size=, random_state=)
The features of the input data
The target or label vector of the input data

If you have done machine learning the function above should look familiar. however it is up to you to determine the test_size. common answers are 80:20, 70:30, 60:40. it is not unusual to go for 70:30 if your data size is lower.

According to Pareto principle. roughly 80% of consequences come from 20% of causes.

which is why 80:20 is so popular. however if you have a smaller dataset. 70:30 is okay to use.

The random state you may set if you wish to find the most optimized model and refining the hyperparameters of the models

VIII. Pre-processing on the training and test data

Grid searching is a hyperparameter tuning technique used to optimize machine learning algorithms. In machine learning, hyperparameters are parameters that are not learned from data, but rather set before training the model. Examples of hyperparameters include learning rate, regularization strength, and number of hidden layers.

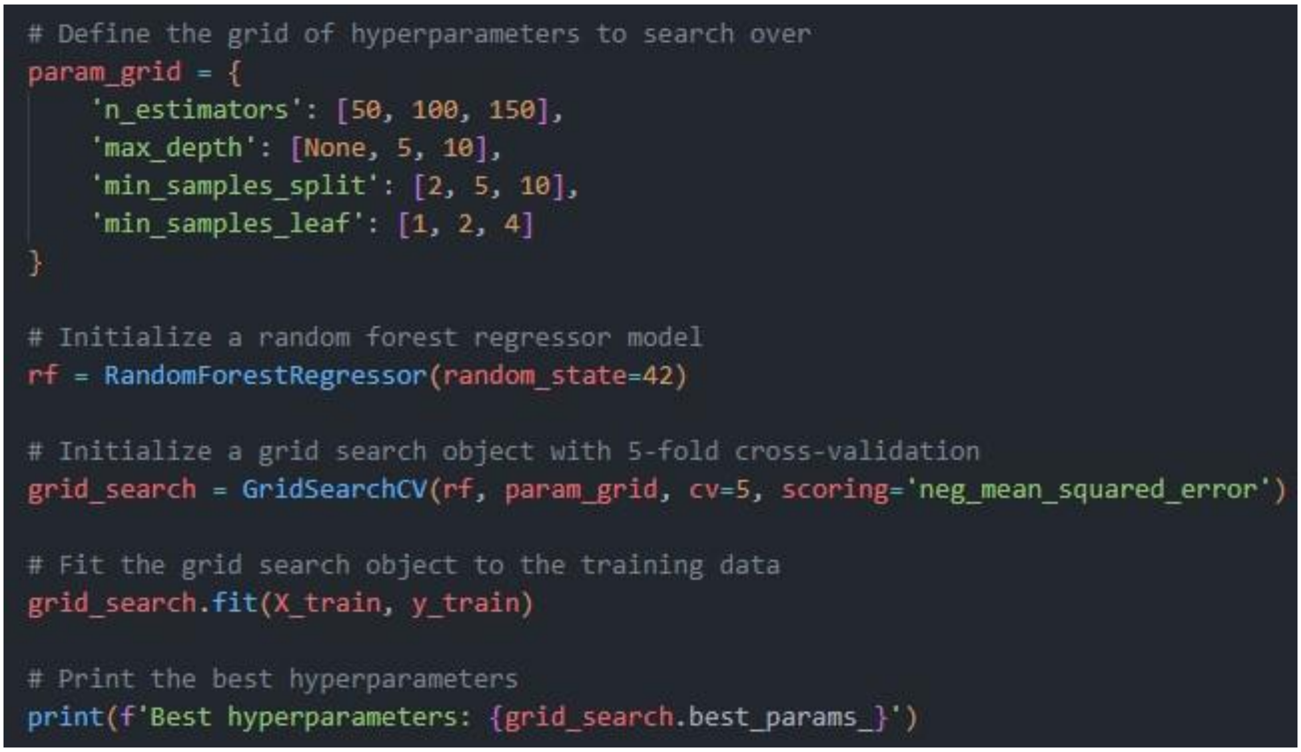
Grid searching involves selecting a set of hyperparameters and then systematically evaluating the performance of a machine learning model trained with these hyperparameters. Specifically, the set of hyperparameters is defined as a grid of possible values, and the algorithm is trained and evaluated for each combination of hyperparameters on this grid. The result is a table of evaluation metrics that allows the selection of the optimal combination of hyperparameters that maximizes the performance of the model.

The process of grid searching involves the following steps:

1. Define the grid: Define a set of hyperparameters and their possible values to create a grid of all possible hyperparameter combinations.
2. Train and evaluate: Train and evaluate the model for each combination of hyperparameters in the grid.
3. Select the best hyperparameters: Select the combination of hyperparameters that yield the best performance on the evaluation metric.

By performing grid searching, the optimal set of hyperparameters can be found that maximizes the performance of the machine learning algorithm. This can lead to a significant improvement in the accuracy of the model, and can ultimately help to solve the problem at hand more e ectively.

For example:



```python
# Define the grid of hyperparameters to search over
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initialize a random forest regressor model
rf = RandomForestRegressor(random_state=42)

# Initialize a grid search object with 5-fold cross-validation
grid_search = GridSearchCV(rf, param_grid, cv=5, scoring='neg_mean_squared_error')

# Fit the grid search object to the training data
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print(f'Best hyperparameters: {grid_search.best_params_}')
```

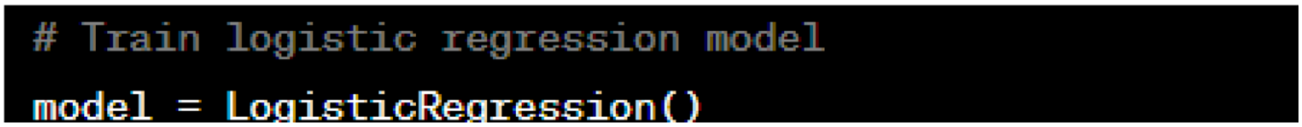
IX. Train the Machine learning models

run the training code by now you should have everything and usually only two lines to run the model. might take awhile depending how much data you have

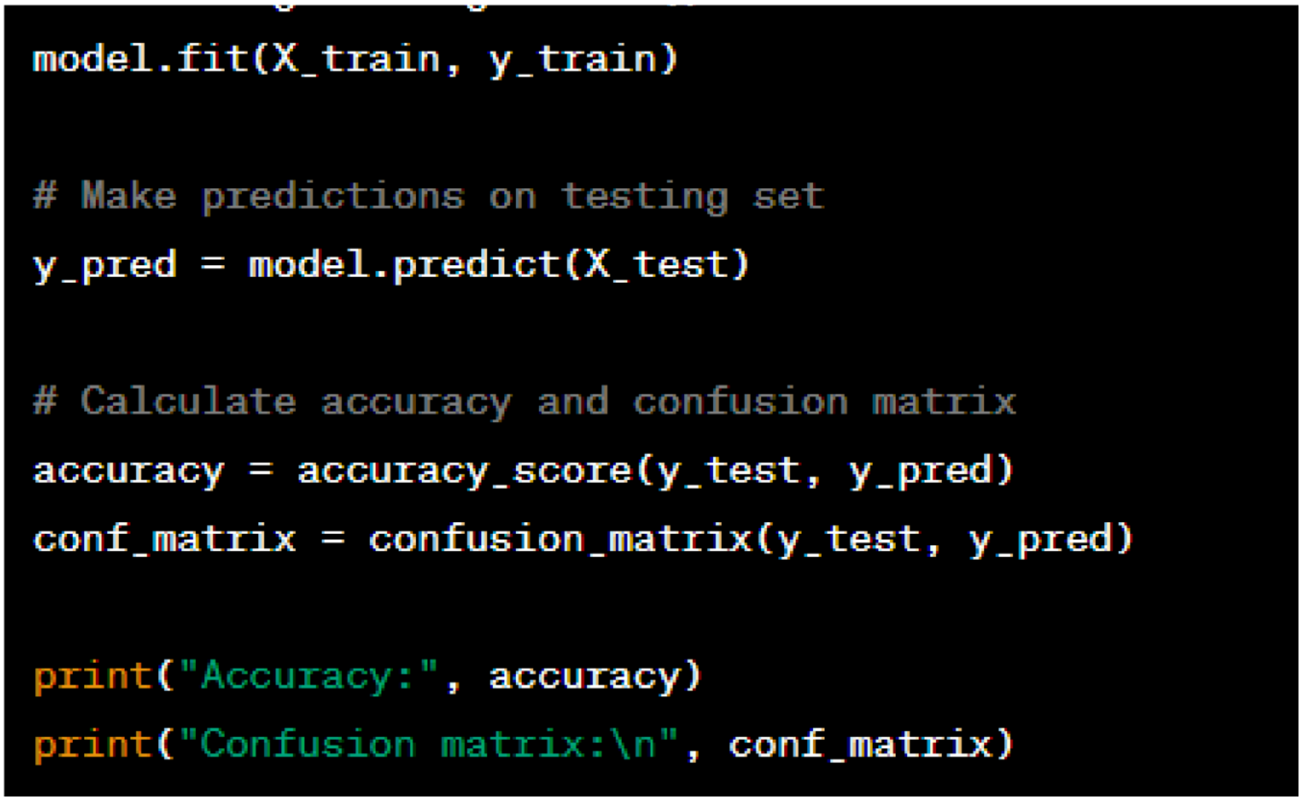
X. & XI. Use Test Data to evaluate the models, Select appropriate metrics to evaluate the accuracy of the models.

Depending on what kind of models that you have. there are di erent kinds of validation of the models. example of what metrics you would use:

- Classification:**accuracy,**precision, recall, F1-score, ROC curve, confusion matrix, etc.

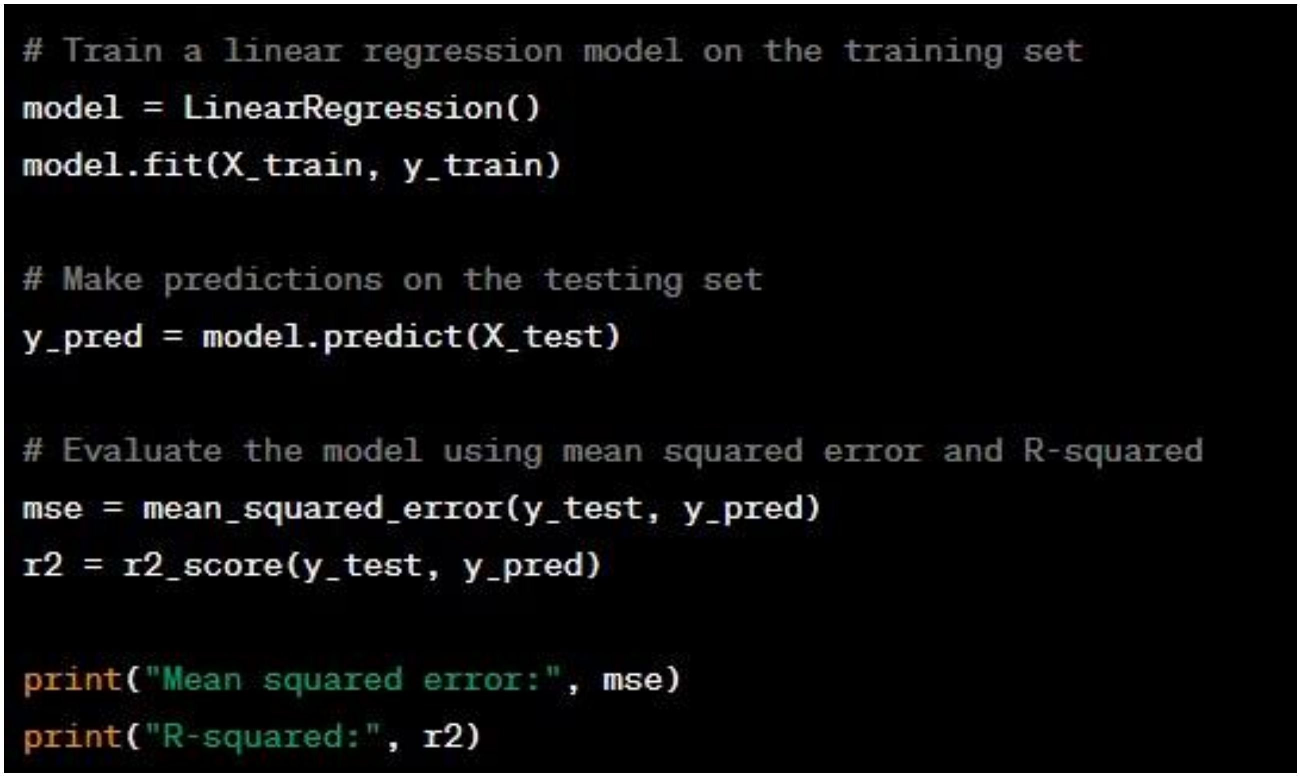


```python
# Train logistic regression model
model = LogisticRegression()
```



```
model.fit(X_train, y_train)

# Make predictions on testing set
y_pred = model.predict(X_test)

# Calculate accuracy and confusion matrix
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print("Accuracy:", accuracy)
print("Confusion matrix:\n", conf_matrix)
```

- Regression: **mean squared error (MSE), root mean squared error (RMSE)**, mean absolute error (MAE), R-squared, etc.



```
# Train a linear regression model on the training set
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = model.predict(X_test)

# Evaluate the model using mean squared error and R-squared
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean squared error:", mse)
print("R-squared:", r2)
```

Classification Metrics:

Classification metrics are used to evaluate models that predict a categorical outcome. Some common classification metrics include:

- Accuracy: the proportion of correct predictions made by the model
- Precision: the proportion of true positives among all positive predictions made by the model
- Recall: the proportion of true positives among all actual positive instances in the data F1 score: a
- weighted average of precision and recall that balances the trade-o between the two
-

You can use the accuracy_score, precision_score, recall_score, and f1_score functions from the scikit-learn library to calculate these metrics in Python.

Regression Metrics:

Regression metrics are used to evaluate models that predict a continuous outcome. Some common regression metrics include:

- Mean Squared Error (MSE): the average of the squared di erences between the predicted and actual values
- Mean Absolute Error (MAE): the average of the absolute di erences between the predicted and actual values
- R-squared: a measure of how well the model fits the data relative to a simple average of the target variable

You can use the mean_squared_error, mean_absolute_error, and r2_score functions from the scikit-learn library to calculate these metrics in Python.

XII. Demonstrate the accuracy of each model using the metrics.

Run the metrics and compare the two models. see which one better. Document your answers.