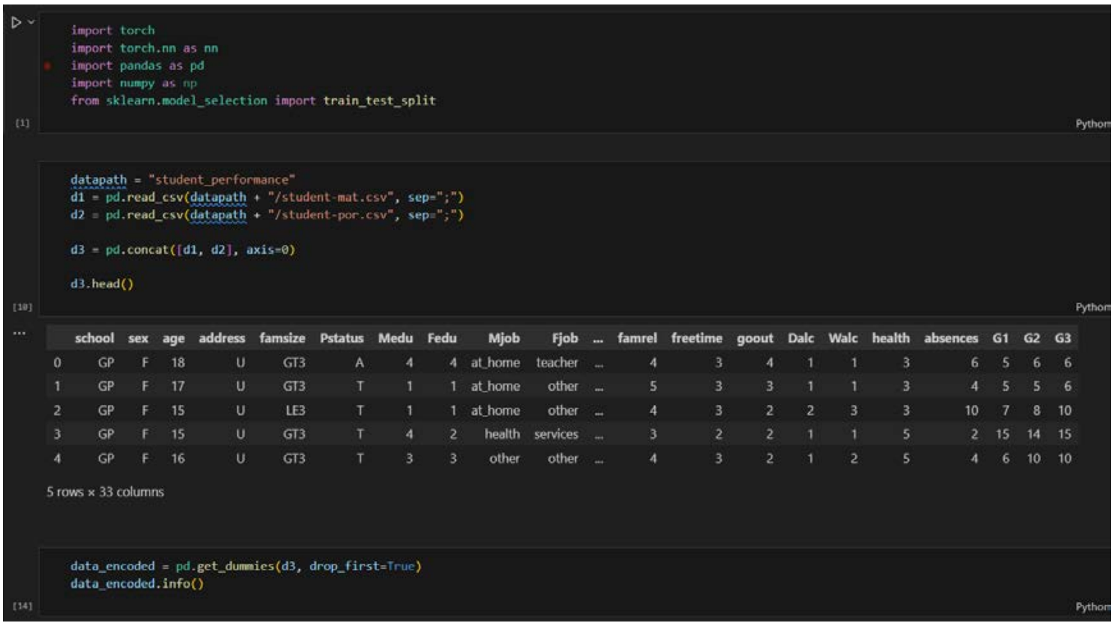
Why pdf only…



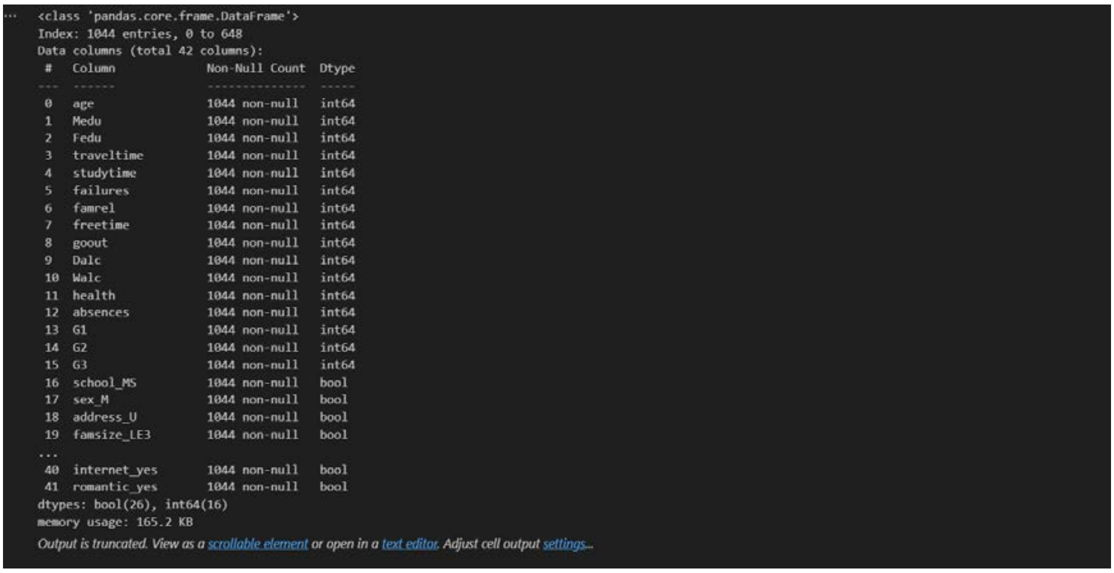
```python
import torch
import torch.nn as nn
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
```

```python
datapath = "student_performance"
d1 = pd.read_csv(datapath + "/student-mat.csv", sep=";")
d2 = pd.read_csv(datapath + "/student-por.csv", sep=";")

d3 = pd.concat([d1, d2], axis=0)

d3.head()
```

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4	3	4	1	1	3	6	5	6	6
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	5	3	3	1	1	3	4	5	5	6
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	4	3	2	2	3	3	10	7	8	10
3	GP	F	15	U	GT3	T	4	2	health	services	...	3	2	2	1	1	5	2	15	14	15
4	GP	F	16	U	GT3	T	3	3	other	other	...	4	3	2	1	2	5	4	6	10	10

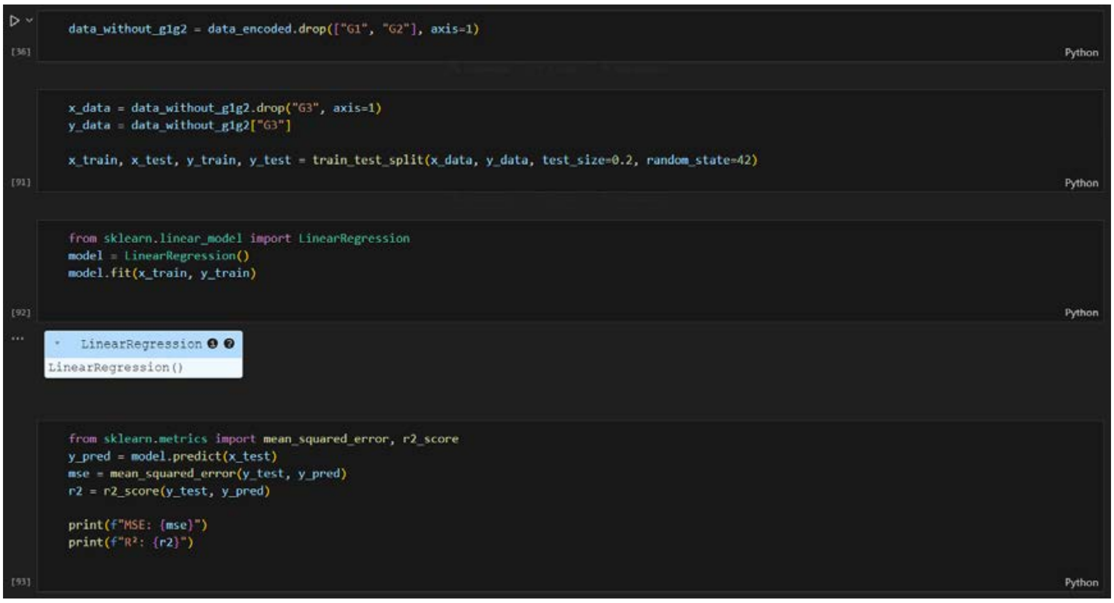
5 rows × 33 columns

```python
data_encoded = pd.get_dummies(d3, drop_first=True)
data_encoded.info()
```



```
<class 'pandas.core.frame.DataFrame'>
Index: 1044 entries, 0 to 648
Data columns (total 42 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             1044 non-null   int64
 1   Medu            1044 non-null   int64
 2   Fedu            1044 non-null   int64
 3   traveltime      1044 non-null   int64
 4   studytime       1044 non-null   int64
 5   failures        1044 non-null   int64
 6   famrel          1044 non-null   int64
 7   freetime        1044 non-null   int64
 8   goout           1044 non-null   int64
 9   Dalc            1044 non-null   int64
 10  Walc            1044 non-null   int64
 11  health          1044 non-null   int64
 12  absences        1044 non-null   int64
 13  G1              1044 non-null   int64
 14  G2              1044 non-null   int64
 15  G3              1044 non-null   int64
 16  school_MS       1044 non-null   bool
 17  sex_M           1044 non-null   bool
 18  address_U       1044 non-null   bool
 19  famsize_LE3     1044 non-null   bool
...
 40  internet_yes    1044 non-null   bool
 41  romantic_yes    1044 non-null   bool
dtypes: bool(26), int64(16)
memory usage: 165.2 KB
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...



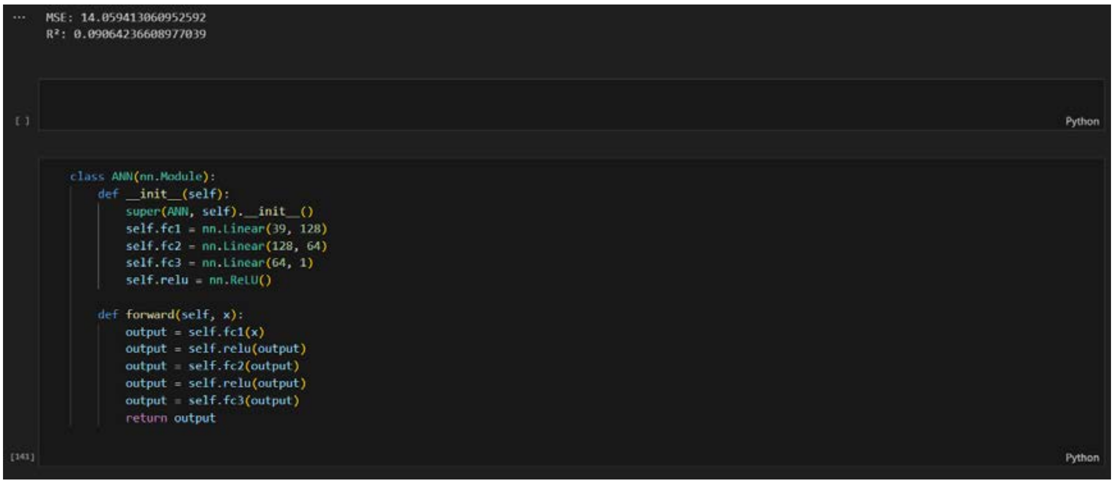
```python
data_without_g1g2 = data_encoded.drop(["G1", "G2"], axis=1)
```

```python
x_data = data_without_g1g2.drop("G3", axis=1)
y_data = data_without_g1g2["G3"]

x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.2, random_state=42)
```

```python
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x_train, y_train)
```

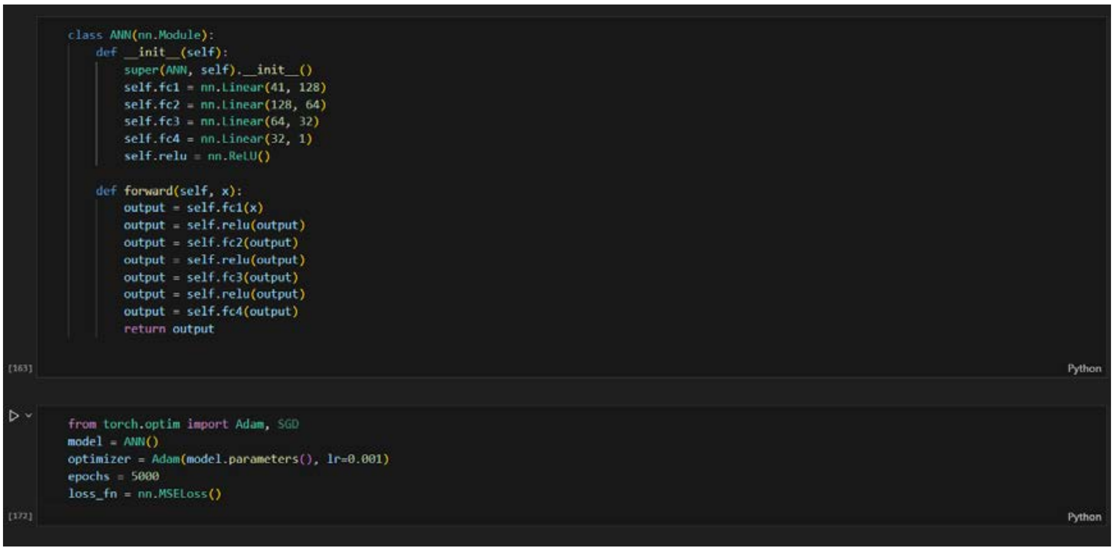
```
LinearRegression
LinearRegression()
```

```python
from sklearn.metrics import mean_squared_error, r2_score
y_pred = model.predict(x_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"MSE: {mse}")
print(f"R²: {r2}")
```



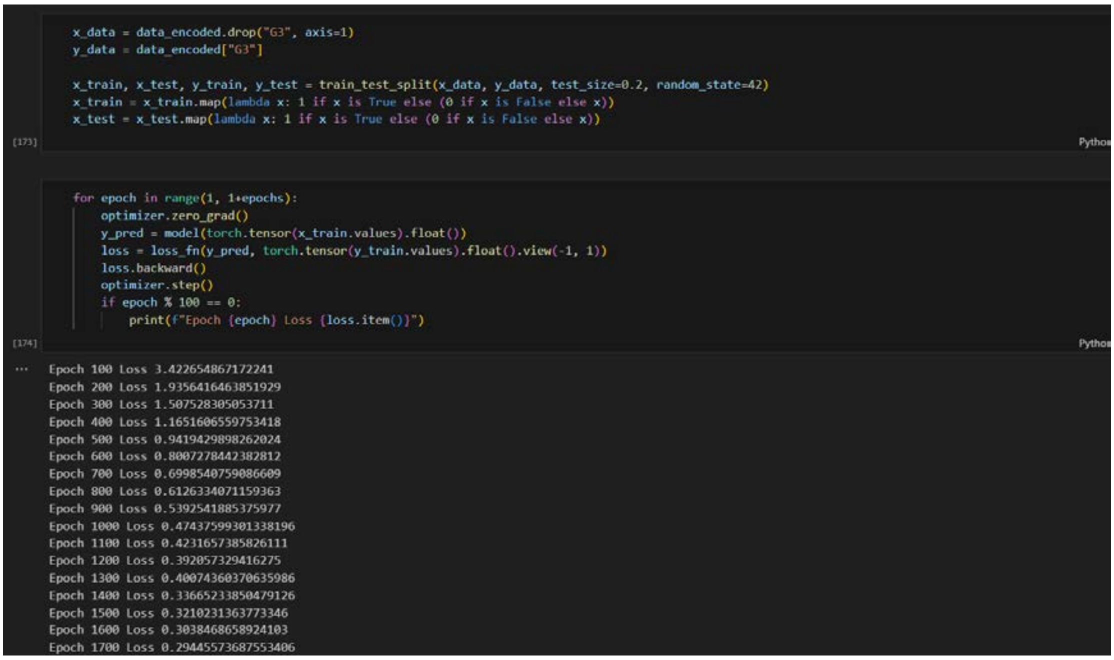
```
MSE: 14.059413060952592
R²: 0.09064236608977039
```

```python
class ANN(nn.Module):
    def __init__(self):
        super(ANN, self).__init__()
        self.fc1 = nn.Linear(39, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 1)
        self.relu = nn.ReLU()

    def forward(self, x):
        output = self.fc1(x)
        output = self.relu(output)
        output = self.fc2(output)
        output = self.relu(output)
        output = self.fc3(output)
        return output
```



```python
class ANN(nn.Module):
    def __init__(self):
        super(ANN, self).__init__()
        self.fc1 = nn.Linear(41, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 32)
        self.fc4 = nn.Linear(32, 1)
        self.relu = nn.ReLU()

    def forward(self, x):
        output = self.fc1(x)
        output = self.relu(output)
        output = self.fc2(output)
        output = self.relu(output)
        output = self.fc3(output)
        output = self.relu(output)
        output = self.fc4(output)
        return output
```

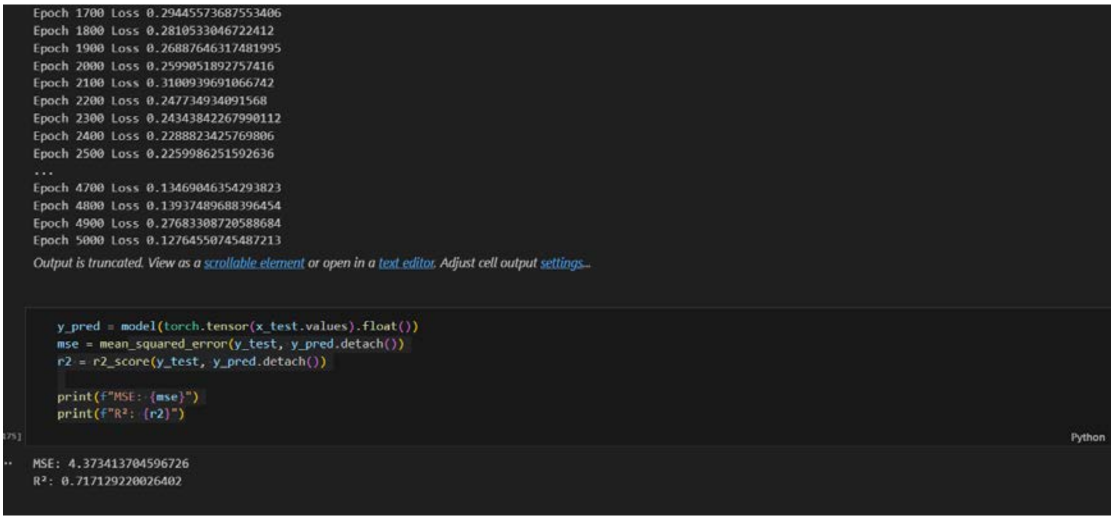
```python
from torch.optim import Adam, SGD
model = ANN()
optimizer = Adam(model.parameters(), lr=0.001)
epochs = 5000
loss_fn = nn.MSELoss()
```



```python
x_data = data_encoded.drop("G3", axis=1)
y_data = data_encoded["G3"]

x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.2, random_state=42)
x_train = x_train.map(lambda x: 1 if x is True else (0 if x is False else x))
x_test = x_test.map(lambda x: 1 if x is True else (0 if x is False else x))
```

```python
for epoch in range(1, 1+epochs):
    optimizer.zero_grad()
    y_pred = model(torch.tensor(x_train.values).float())
    loss = loss_fn(y_pred, torch.tensor(y_train.values).float().view(-1, 1))
    loss.backward()
    optimizer.step()
    if epoch % 100 == 0:
        print(f"Epoch {epoch} Loss {loss.item()}")
```

```
Epoch 100 Loss 3.422654867172241
Epoch 200 Loss 1.9356416463851929
Epoch 300 Loss 1.507528305053711
Epoch 400 Loss 1.1651606559753418
Epoch 500 Loss 0.9419429898262024
Epoch 600 Loss 0.8007278442382812
Epoch 700 Loss 0.6998540759086609
Epoch 800 Loss 0.6126334071159363
Epoch 900 Loss 0.5392541885375977
Epoch 1000 Loss 0.47437599301338196
Epoch 1100 Loss 0.4231657385826111
Epoch 1200 Loss 0.392057329416275
Epoch 1300 Loss 0.40074360370635986
Epoch 1400 Loss 0.33665233850479126
Epoch 1500 Loss 0.3210231363773346
Epoch 1600 Loss 0.3038468658924103
Epoch 1700 Loss 0.29445573687553406
```



```
Epoch 1700 Loss 0.29445573687553406
Epoch 1800 Loss 0.2810533046722412
Epoch 1900 Loss 0.26887646317481995
Epoch 2000 Loss 0.2599051892757416
Epoch 2100 Loss 0.3100939691066742
Epoch 2200 Loss 0.247734934091568
Epoch 2300 Loss 0.24343842267990112
Epoch 2400 Loss 0.2288823425769806
Epoch 2500 Loss 0.2259986251592636
...
Epoch 4700 Loss 0.13469046354293823
Epoch 4800 Loss 0.13937489688396454
Epoch 4900 Loss 0.27683308720588684
Epoch 5000 Loss 0.12764550745487213
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```python
y_pred = model(torch.tensor(x_test.values).float())
mse = mean_squared_error(y_test, y_pred.detach())
r2 = r2_score(y_test, y_pred.detach())

print(f"MSE: {mse}")
print(f"R²: {r2}")
```

```
MSE: 4.373413704596726
R²: 0.717129220026402
```