
Walmart
5250 Commercial St. SE
EXIT

WALMART

We decided to predict the differences in weekly sales of Walmart the american retailer

TABLES OF THE SALES AND THE STORES INFORMATION

```
[18] store_df.head()
```

	Store	Type	Size
0	1	A	151315
1	2	A	202307
2	3	B	37392
3	4	A	205863
4	5	B	34875

```
df.head()
```

	Store	Dept	Date	Weekly_Sales	IsHoliday
0	1	1	2010-02-05	24924.50	False
1	1	1	2010-02-12	46039.49	True
2	1	1	2010-02-19	41595.55	False
3	1	1	2010-02-26	19403.54	False
4	1	1	2010-03-05	21827.90	False

ANOTHER TABLE THAT HAS INFORMATION ON THE STORE IN A SPECIFIC WEEK

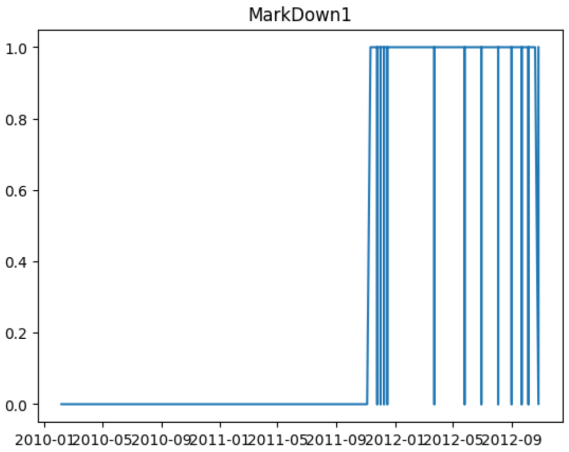
```
[17] features_df.head()
```

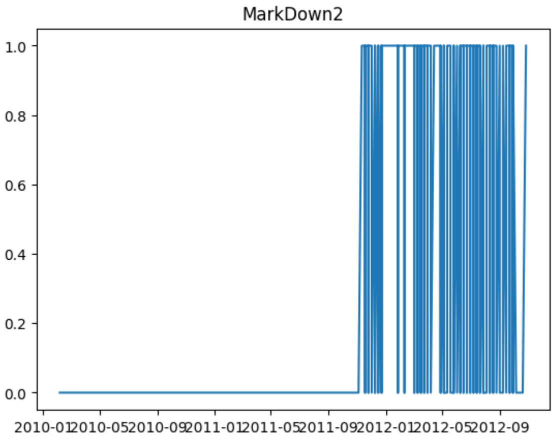
	Store	Date	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment	IsHoliday
0	1	2010-02-05	42.31	2.572	NaN	NaN	NaN	NaN	NaN	211.096358	8.106	False
1	1	2010-02-12	38.51	2.548	NaN	NaN	NaN	NaN	NaN	211.242170	8.106	True
2	1	2010-02-19	39.93	2.514	NaN	NaN	NaN	NaN	NaN	211.289143	8.106	False
3	1	2010-02-26	46.63	2.561	NaN	NaN	NaN	NaN	NaN	211.319643	8.106	False
4	1	2010-03-05	46.50	2.625	NaN	NaN	NaN	NaN	NaN	211.350143	8.106	False

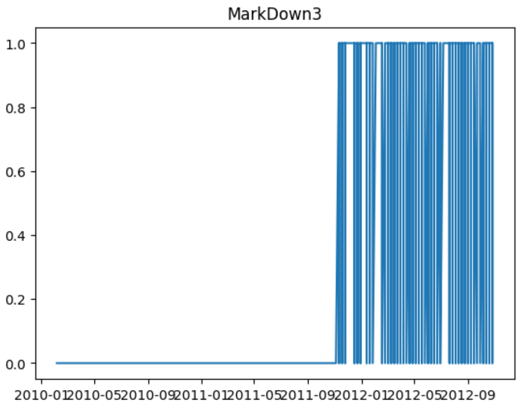
We decided to remove feautures

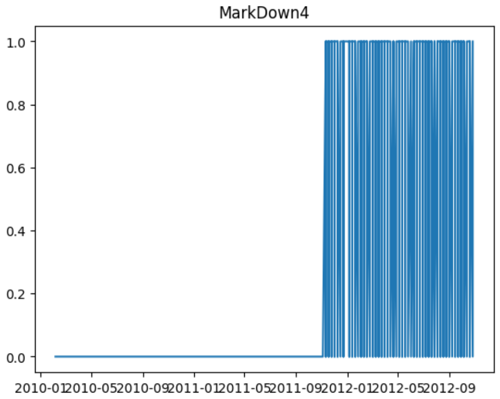
- MarkDown1
- MarkDown2
- MarkDown3
- MarkDown4
- MarkDown5

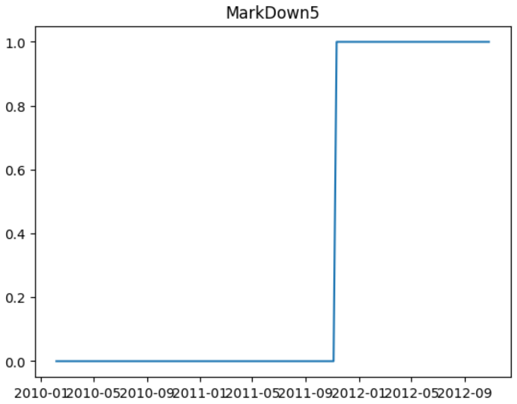
Since they exist only since 2012 and even then are lacking











JOINED TABLE

	Store	Dept	Date	Weekly_Sales	IsHoliday_x	Type	Size	Temperature	Fuel_Price	CPI	Unemployment	IsHoliday_y
0	1	1	2010-02-05	24924.50	False	A	151315	42.31	2.572	211.096358	8.106	False
1	1	2	2010-02-05	50605.27	False	A	151315	42.31	2.572	211.096358	8.106	False
2	1	3	2010-02-05	13740.12	False	A	151315	42.31	2.572	211.096358	8.106	False
3	1	4	2010-02-05	39954.04	False	A	151315	42.31	2.572	211.096358	8.106	False
4	1	5	2010-02-05	32229.38	False	A	151315	42.31	2.572	211.096358	8.106	False
...	...	...	...	...	...	...	...	...	...	...	...	...
421565	45	93	2012-10-26	2487.80	False	B	118221	58.85	3.882	192.308899	8.667	False
421566	45	94	2012-10-26	5203.31	False	B	118221	58.85	3.882	192.308899	8.667	False
421567	45	95	2012-10-26	56017.47	False	B	118221	58.85	3.882	192.308899	8.667	False
421568	45	97	2012-10-26	6817.48	False	B	118221	58.85	3.882	192.308899	8.667	False
421569	45	98	2012-10-26	1076.80	False	B	118221	58.85	3.882	192.308899	8.667	False

421570 rows × 12 columns

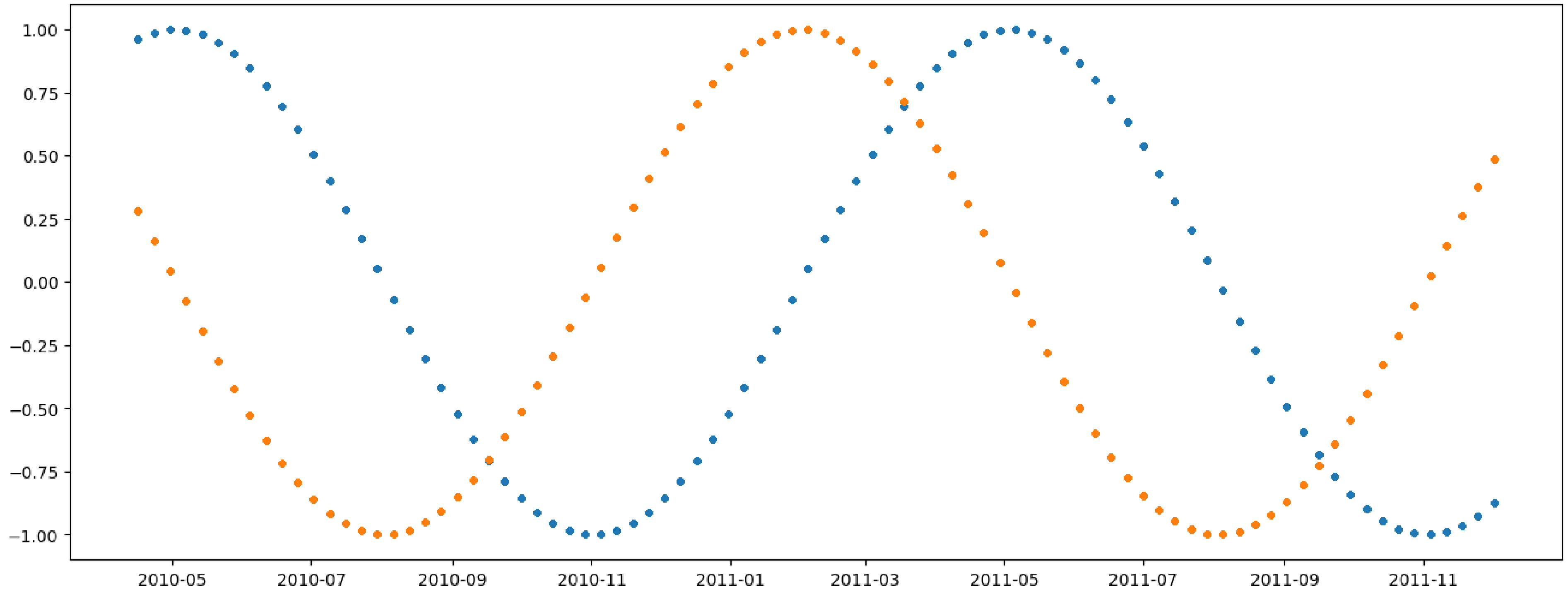
GENERAL STATISTIC INFORMATION

```
[24] df.describe()
```

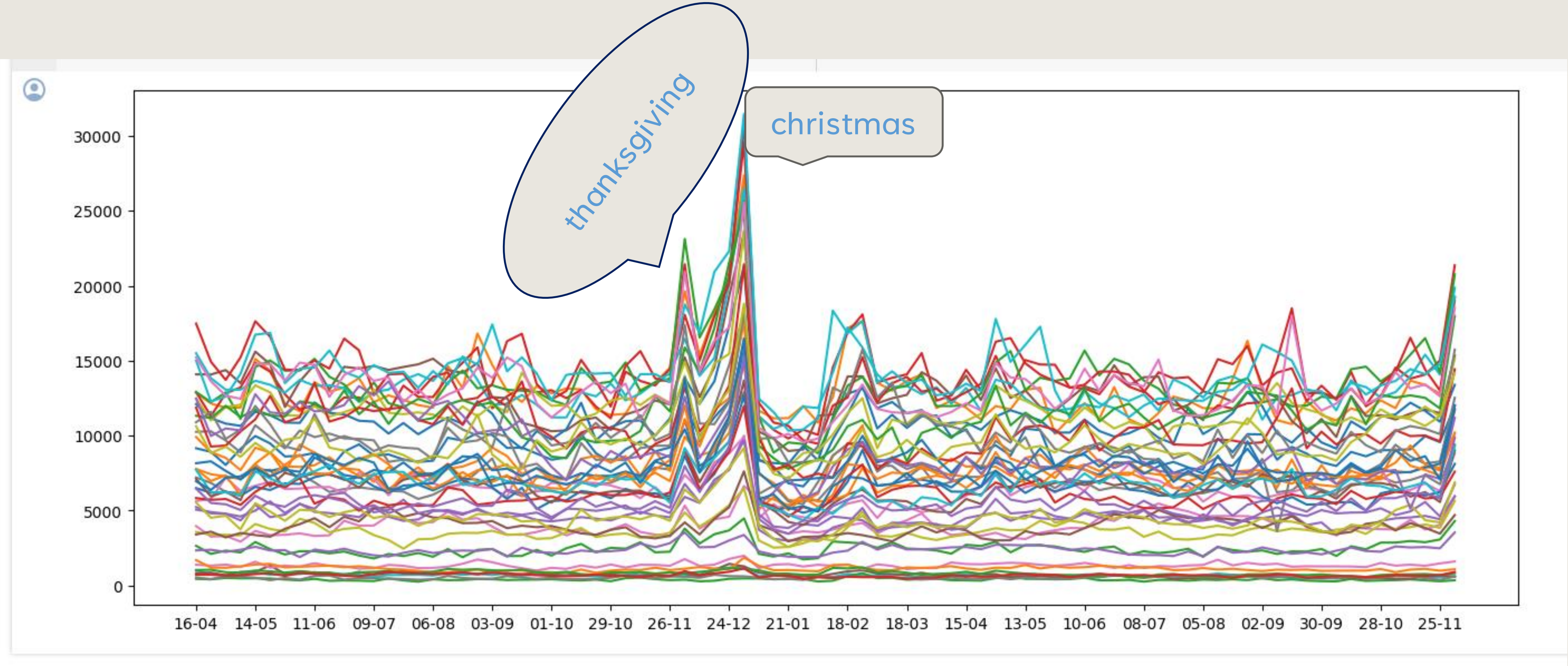
	Store	Dept	Weekly_Sales	Size	Temperature	Fuel_Price	CPI	Unemployment
count	421570.000000	421570.000000	421570.000000	421570.000000	421570.000000	421570.000000	421570.000000	421570.000000
mean	22.200546	44.260317	15981.258123	136727.915739	60.090059	3.361027	171.201947	7.960289
std	12.785297	30.492054	22711.183519	60980.583328	18.447931	0.458515	39.159276	1.863296
min	1.000000	1.000000	-4988.940000	34875.000000	-2.060000	2.472000	126.064000	3.879000
25%	11.000000	18.000000	2079.650000	93638.000000	46.680000	2.933000	132.022667	6.891000
50%	22.000000	37.000000	7612.030000	140167.000000	62.090000	3.452000	182.318780	7.866000
75%	33.000000	74.000000	20205.852500	202505.000000	74.280000	3.738000	212.416993	8.572000
max	45.000000	99.000000	693099.360000	219622.000000	100.140000	4.468000	227.232807	14.313000

KEEPING TRACK OF TIME WITH SIN AND COS

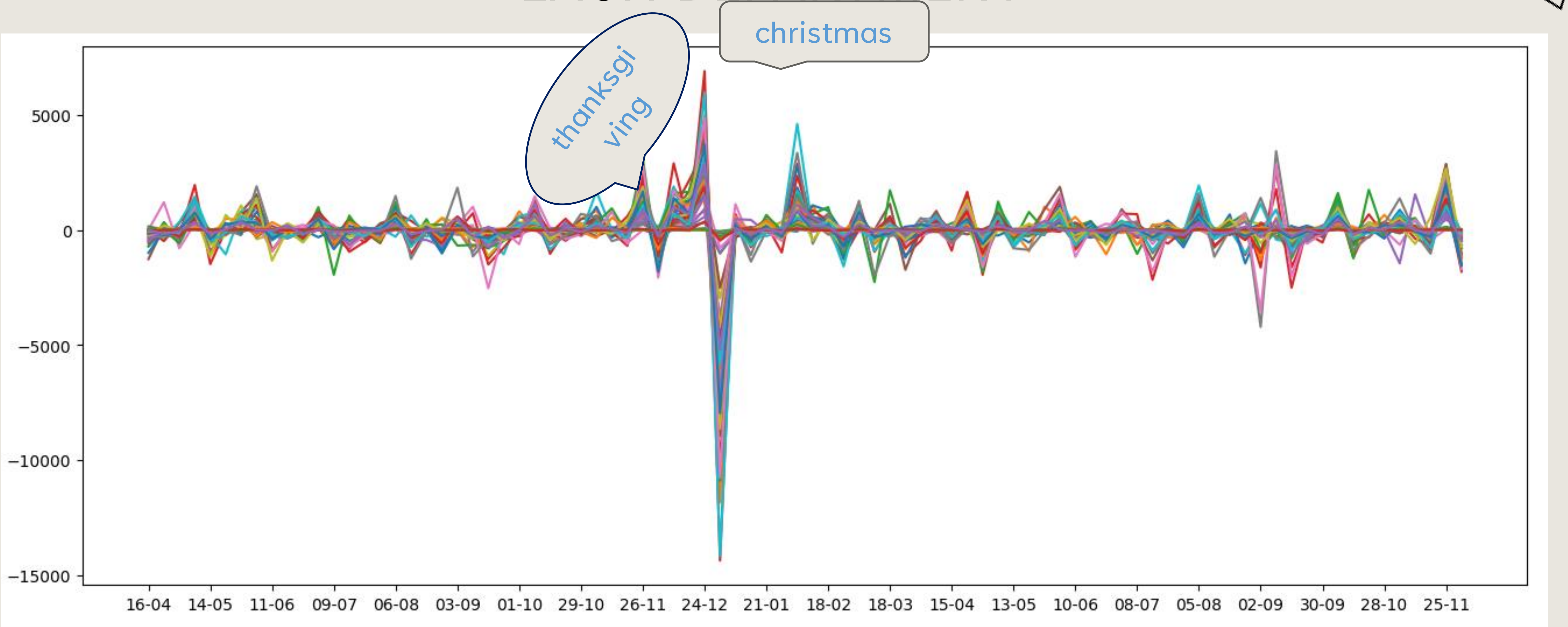
```
plt.figure(figsize=(16, 6))

plt.plot(train_df["Date"], train_df["Year-Sin"], '.')
plt.plot(train_df["Date"], train_df["Year-Cos"], '.')

plt.show()
```



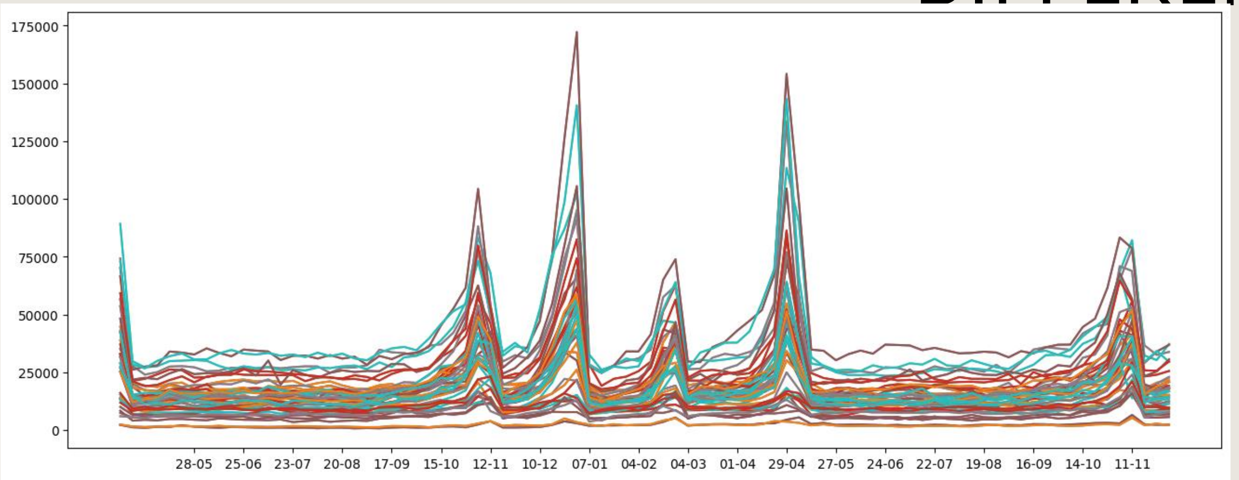
MEDIAN OF THE SALES FOR EACH DEPARTMENT

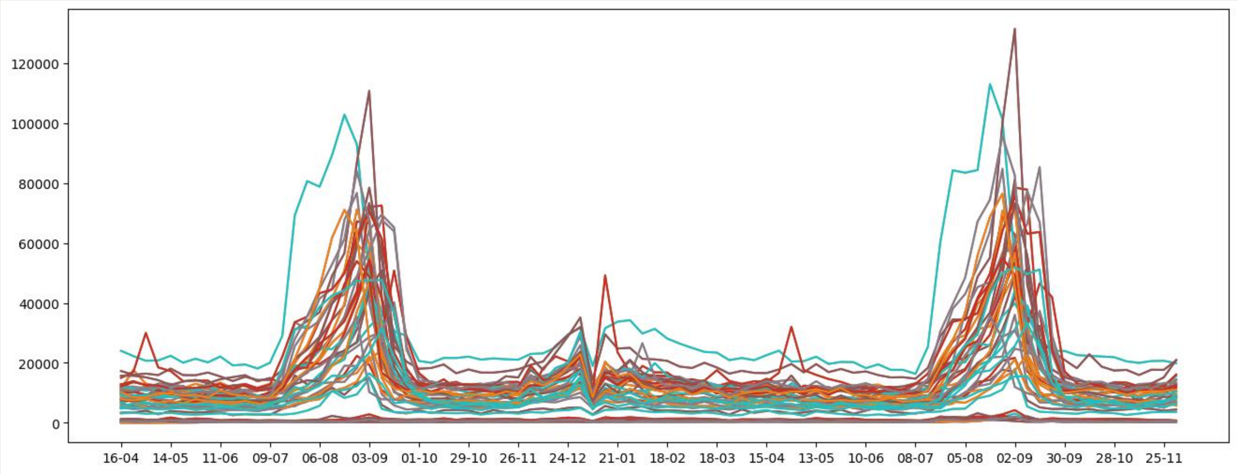


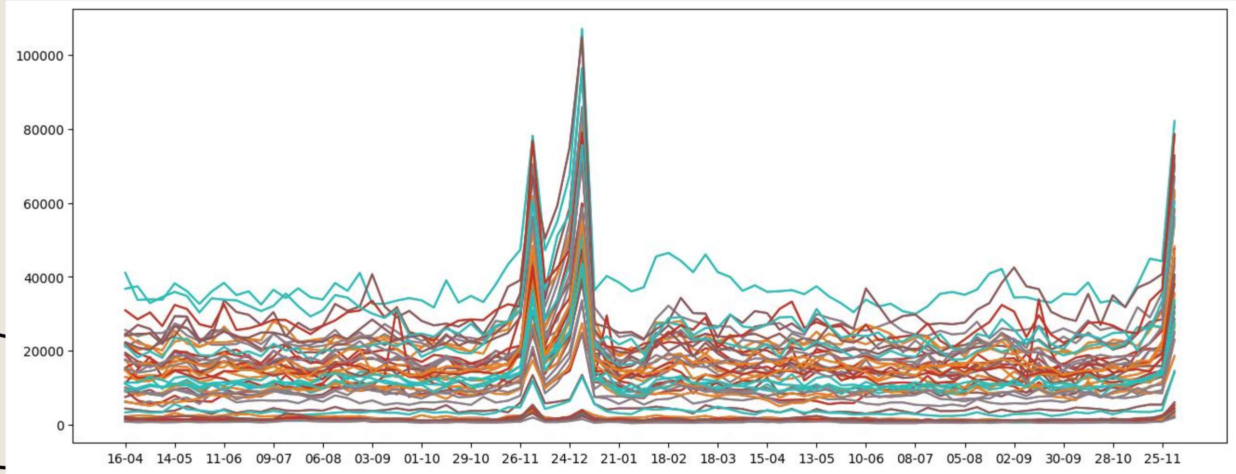
MEDIAN OF THE WEEKLY DIFFERENCE IN SALES FOR EACH DEPARTMENT

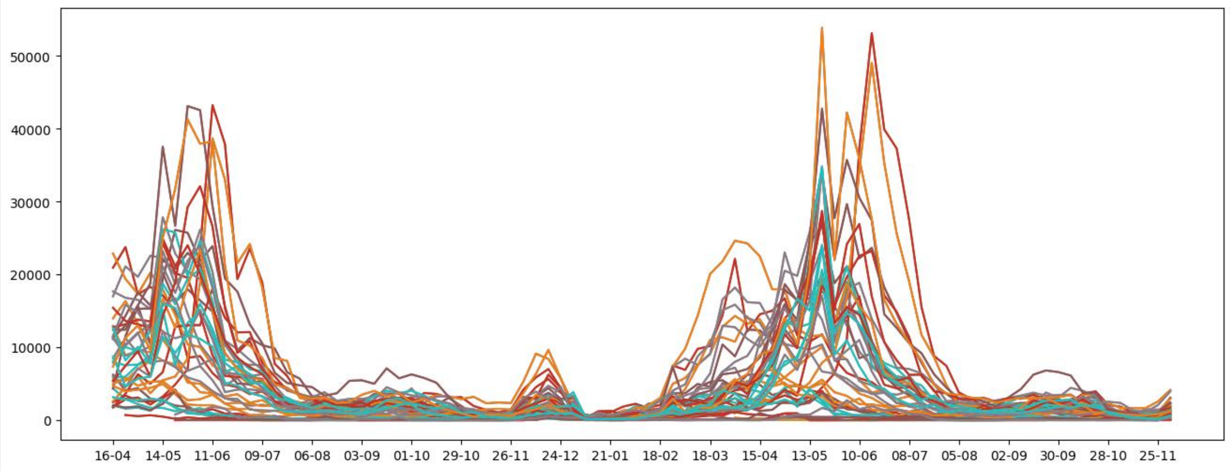


LOOKING AT SALES OF ALL DEPARTMENTS FOR DIFFERENT STORES

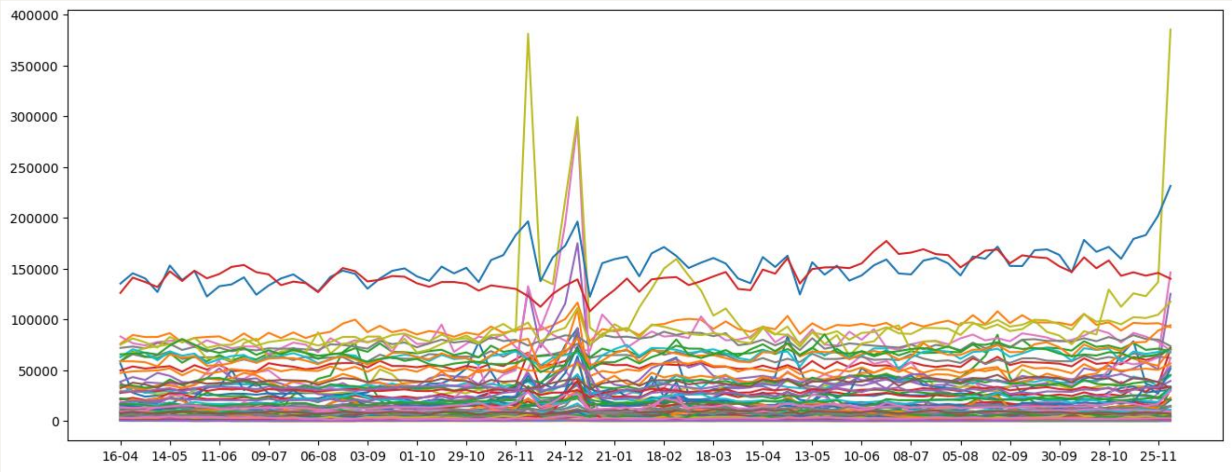


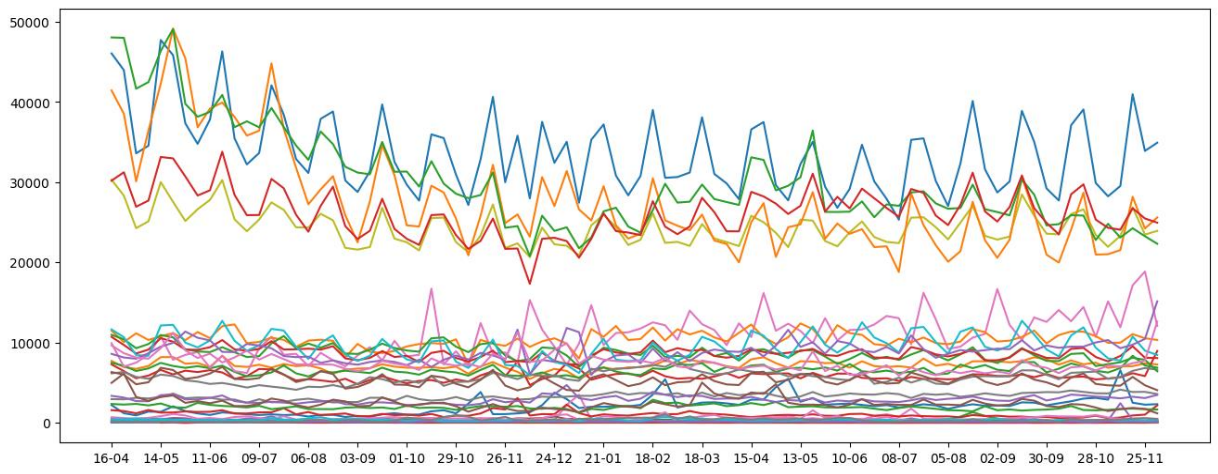


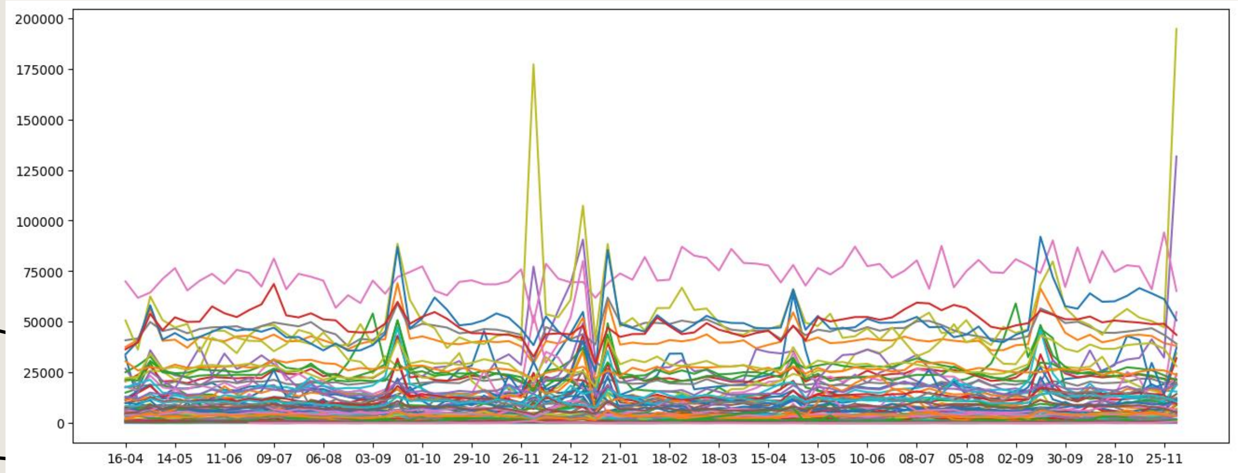


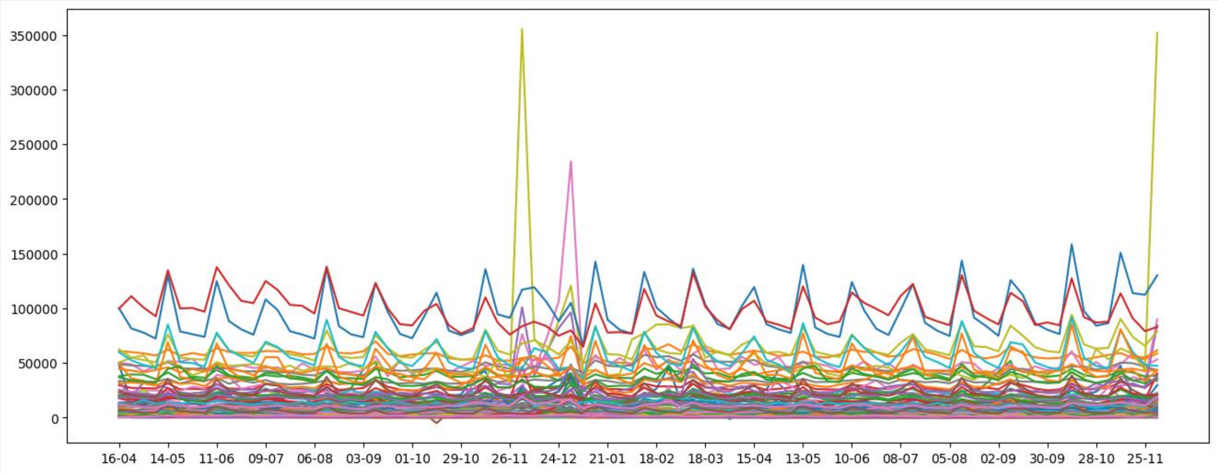


LOOKING AT ALL STORES FOR DIFFERENT DEPARTMENTS









Linear Regression data

index	Store	Dept	Date	Temperature	Fuel_Price	CPI	Unemployment	IsHoliday	Store-Type	Store-Size
0	1	1	2010-05-28	80.44	2.759	210.896761	7.808	0	1	0.233352
1	1	1	2010-06-04	80.69	2.705	211.176428	7.808	0	1	0.233352
2	1	1	2010-06-11	80.43	2.668	211.456095	7.808	0	1	0.233352
3	1	1	2010-06-18	84.11	2.637	211.453772	7.808	0	1	0.233352
4	1	1	2010-06-25	84.34	2.653	211.338653	7.808	0	1	0.233352

Weekly-Sales-last-1-weeks	Weekly-Sales-last-2-weeks	Weekly-Sales-last-3-weeks	Weekly-Sales-last-4-weeks	Weekly-Diff-last-1-weeks	Weekly-Diff-last-2-weeks	Weekly-Diff-last-3-weeks	Weekly-Diff
14773.04	18926.74	17413.94	16555.11	-4153.70	1512.80	858.83	807.39
15580.43	14773.04	18926.74	17413.94	807.39	-4153.70	1512.80	1977.66
17558.09	15580.43	14773.04	18926.74	1977.66	807.39	-4153.70	-920.47
16637.62	17558.09	15580.43	14773.04	-920.47	1977.66	807.39	-421.35
16216.27	16637.62	17558.09	15580.43	-421.35	-920.47	1977.66	112.45

We will use Stochastic Gradient Descent in order to learn linear regression

We will train with MSE as out loss function but use RMSE to evaluate

```python
def train_linear_regression_model(x_train, y_train, x_val, y_val, model, loss_fn, loss_fn_vall, num_epochs, batch_size):
    lr = 0.01

    # Define an optimizer (Stochastic Gradient Descent)
    optimizer = torch.optim.SGD(model.parameters(), lr=lr)

    train_dataset = torch.utils.data.TensorDataset(torch.tensor(x_train, dtype=torch.float32),
                                                   torch.tensor(y_train, dtype=torch.float32))
    train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

    val_dataset = torch.utils.data.TensorDataset(torch.tensor(x_val, dtype=torch.float32),
                                                 torch.tensor(y_val, dtype=torch.float32))
    val_loader = torch.utils.data.DataLoader(val_dataset, batch_size=batch_size, shuffle=True)

    losses = []
    losses_val = []

    for epoch in range(num_epochs):
        running_loss = 0

        for batch in train_loader:
            inputs, targets = batch

            # Forward pass
            outputs = model(inputs)
            loss_train = loss_fn(outputs, targets)

            # Backward and optimize
            optimizer.zero_grad()
            loss_train.backward()
            optimizer.step()

            running_loss += np.sqrt(loss_train.item())

        losses.append(running_loss / len(train_loader))

        # Validation loss
        running_loss = 0

        for batch in val_loader:
            inputs, targets = batch

            # Forward pass
            outputs = model(inputs)
            loss_val = loss_fn_vall(outputs, targets)

            running_loss += np.sqrt(loss_val.item())

        losses_val.append(running_loss / len(val_loader))

        if epoch % int(num_epochs / 10) == 0:
            print(f'Epoch [{epoch}], Running Loss: {running_loss:.4f}')

    return losses, losses_val
```

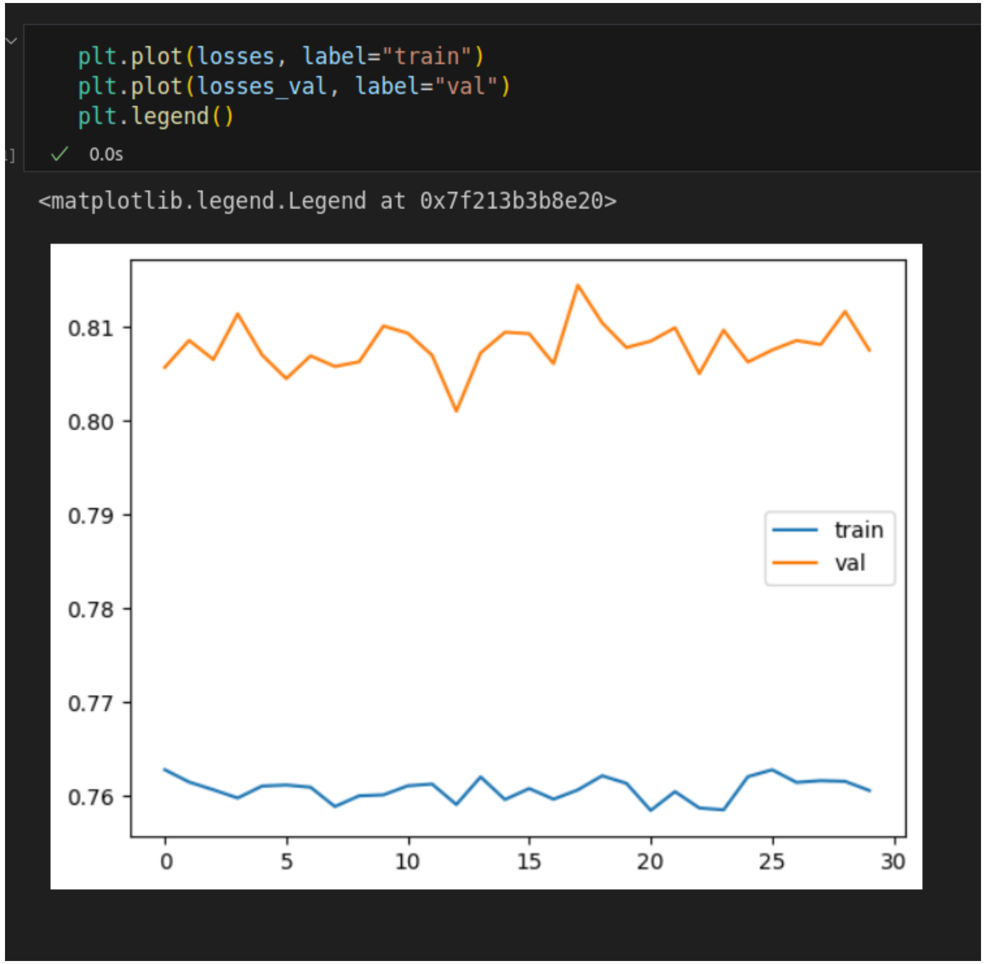
model

```python
class LinearRegression(torch.nn.Module):
    def __init__(self, input_size):
        super(LinearRegression, self).__init__()
        self.linear = torch.nn.Linear(input_size, 1)

    def forward(self, x):
        out = self.linear(x)
        return out
```

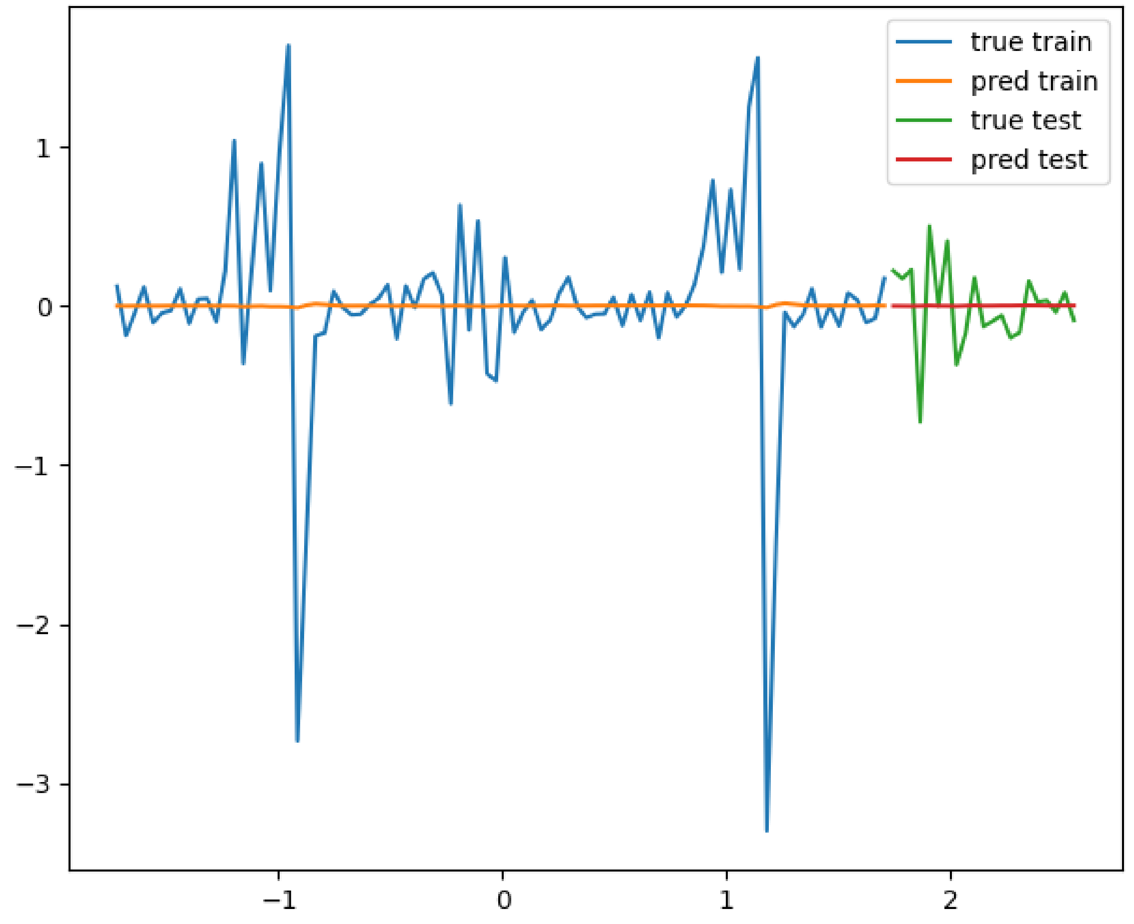
training

Loss function

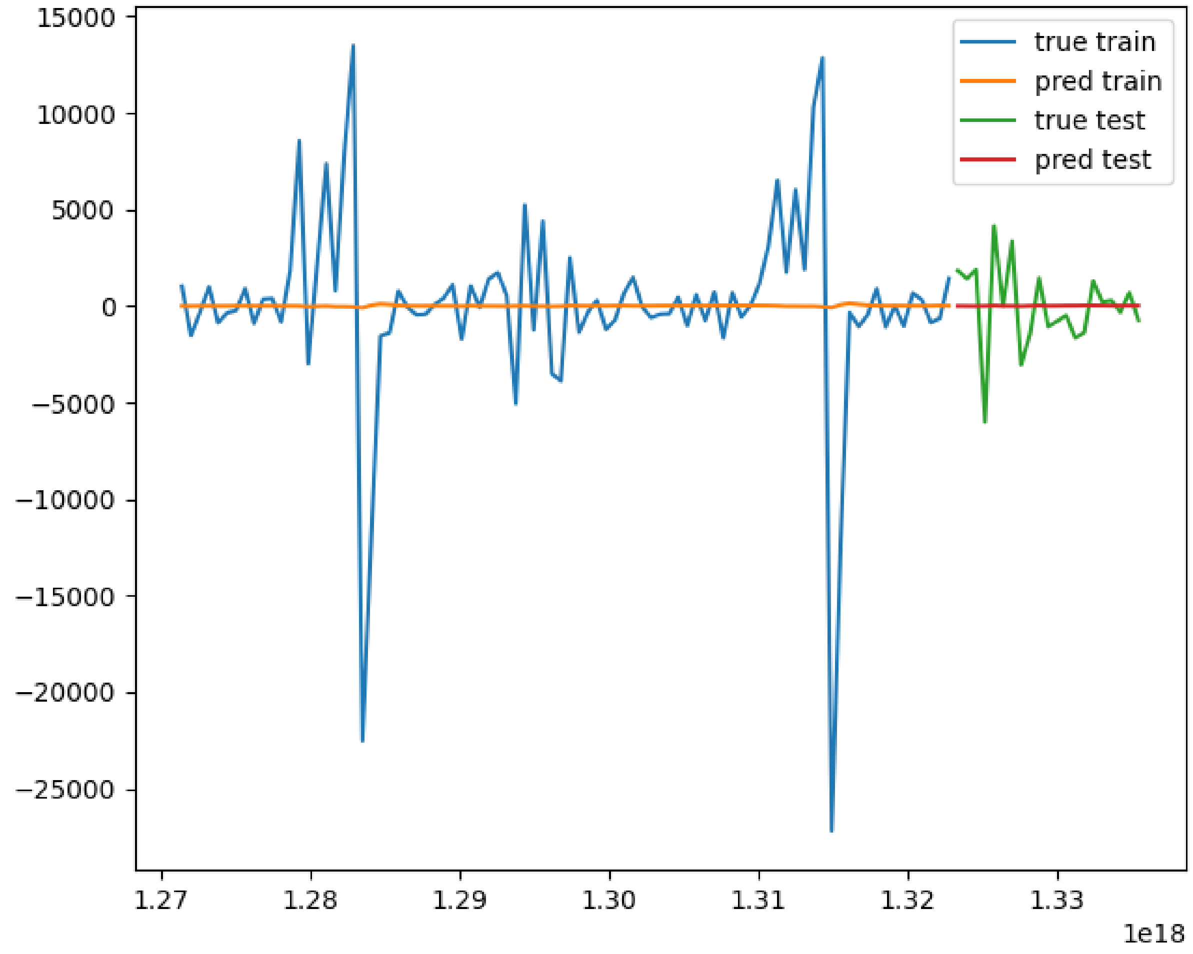
Is it really learning
?



normalized



Non normalized



```python
class NormlizerStdMean:
    def __init__(self, x, y):
        self.y_mean = y.mean()
        self.y_std = y.std()

        self.x_mean = {}
        self.x_std = {}

        for column in x.columns:
            self.x_mean[column] = x[column].mean()
            self.x_std[column] = x[column].std()

    def normalize_y(self, v):
        return (v - self.y_mean) / self.y_std

    def normalize_x(self, v, column):
        return (v - self.x_mean[column]) / self.x_std[column]

    def denormalize_y(self, v):
        return list(map(lambda v0: (v0 * self.y_std) + self.y_mean, v))

    def denormalize_x(self, v, column):
        return list(map(lambda v0: (v0 * self.x_std[column]) + self.x_mean[column], v))
```

```python
class NormlizerMinMax:
    def __init__(self, x, y):
        self.y_min = y.min()
        self.y_max = y.max()

        self.x_min = {}
        self.x_max = {}

        for column in x.columns:
            self.x_min[column] = x[column].min()
            self.x_max[column] = x[column].max()

    def normalize_y(self, v):
        return (v - self.y_min) / (self.y_max - self.y_min)

    def normalize_x(self, v, column):
        return (v - self.x_min[column]) / (self.x_max[column] - self.x_min[column])

    def denormalize_y(self, v):
        return list(map(lambda v0: (v0 * (self.y_max - self.y_min)) + self.y_min, v))

    def denormalize_x(self, v, column):
        return list(map(lambda v0: (v0 * (self.x_max[column] - self.x_min[column])) + self.x_min[column], v))
```

Model(same as before)

```
class LinearRegression(torch.nn.Module):
    def __init__(self, input_size):
        super(LinearRegression, self).__init__()
        self.linear = torch.nn.Linear(input_size, 1)

    def forward(self, x):
        out = self.linear(x)
        return out
```

Training also the same

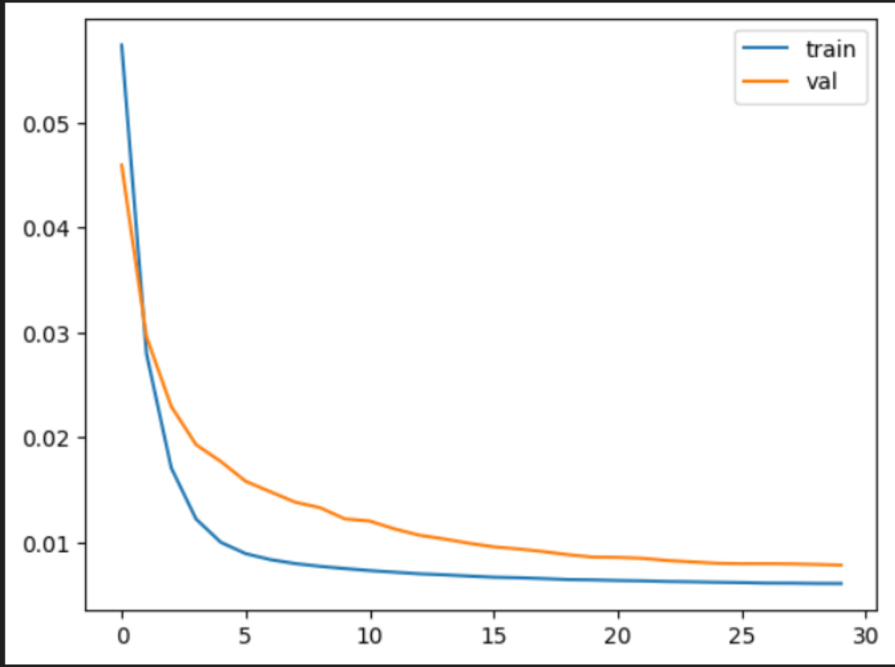
```
def train_linear_regression_model(x_train, y_train, x_val, y_val, model, loss_fn, loss_fn_vall, num_epochs, batch_size):
    lr = 0.01

    # Define an optimizer (Stochastic Gradient Descent)
    optimizer = torch.optim.SGD(model.parameters(), lr=lr)

    train_dataset = torch.utils.data.TensorDataset(torch.tensor(x_train, dtype=torch.float32),
                                                   torch.tensor(y_train, dtype=torch.float32))
    train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

    val_dataset = torch.utils.data.TensorDataset(torch.tensor(x_val, dtype=torch.float32),
                                                 torch.tensor(y_val, dtype=torch.float32))
    val_loader = torch.utils.data.DataLoader(val_dataset, batch_size=batch_size, shuffle=True)

    losses = []
    losses_val = []

    for epoch in range(num_epochs):
        running_loss = 0

        for batch in train_loader:
            inputs, targets = batch

            # Forward pass
            outputs = model(inputs)
            loss_train = loss_fn(outputs, targets)

            # Backward and optimize
            optimizer.zero_grad()
            loss_train.backward()
            optimizer.step()

            running_loss += np.sqrt(loss_train.item())

        losses.append(running_loss / len(train_loader))

        # Validation loss
        running_loss = 0

        for batch in val_loader:
            inputs, targets = batch

            # Forward pass
            outputs = model(inputs)
            loss_val = loss_fn_vall(outputs, targets)

            running_loss += np.sqrt(loss_val.item())

        losses_val.append(running_loss / len(val_loader))

        if epoch % int(num_epochs / 10) == 0:
            print(f'Epoch [{epoch}], Running Loss: {running_loss:.4f}')

    return losses, losses_val
```

Loss function before and after test

```
plt.plot(losses, label="train")
plt.plot(losses_val, label="val")
plt.legend()
```

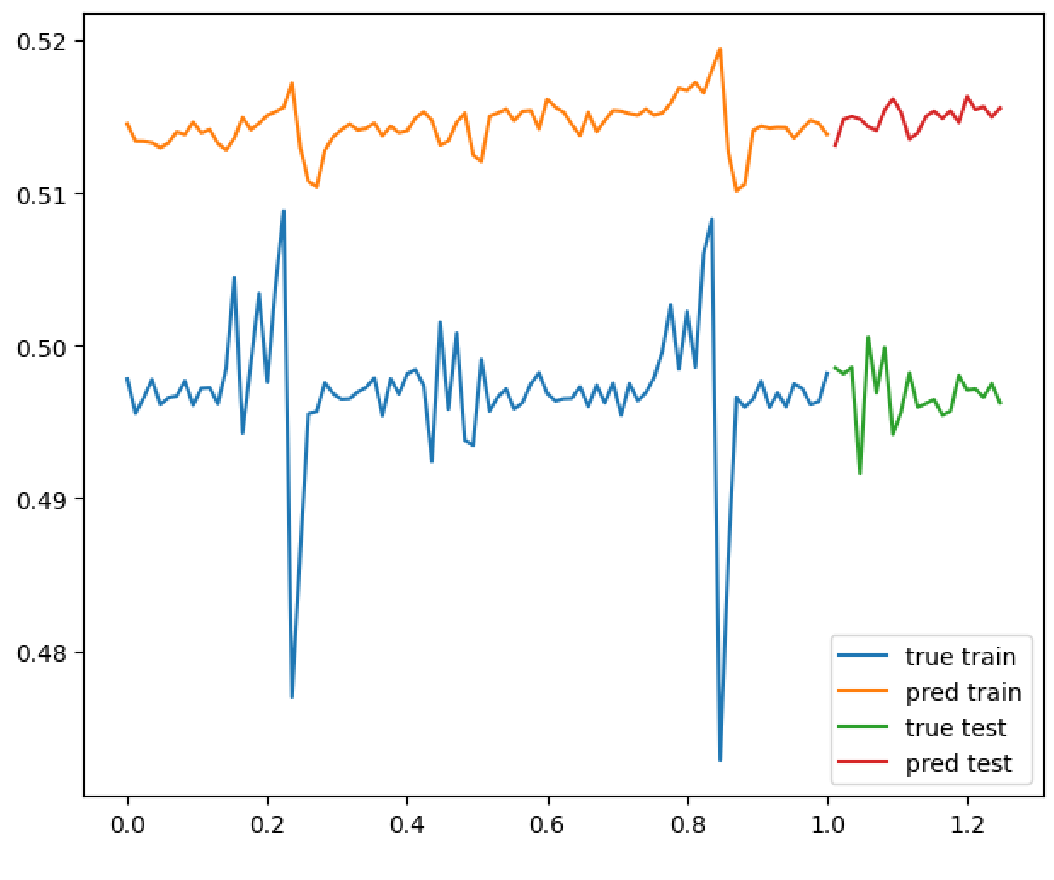
✓ 0.0s

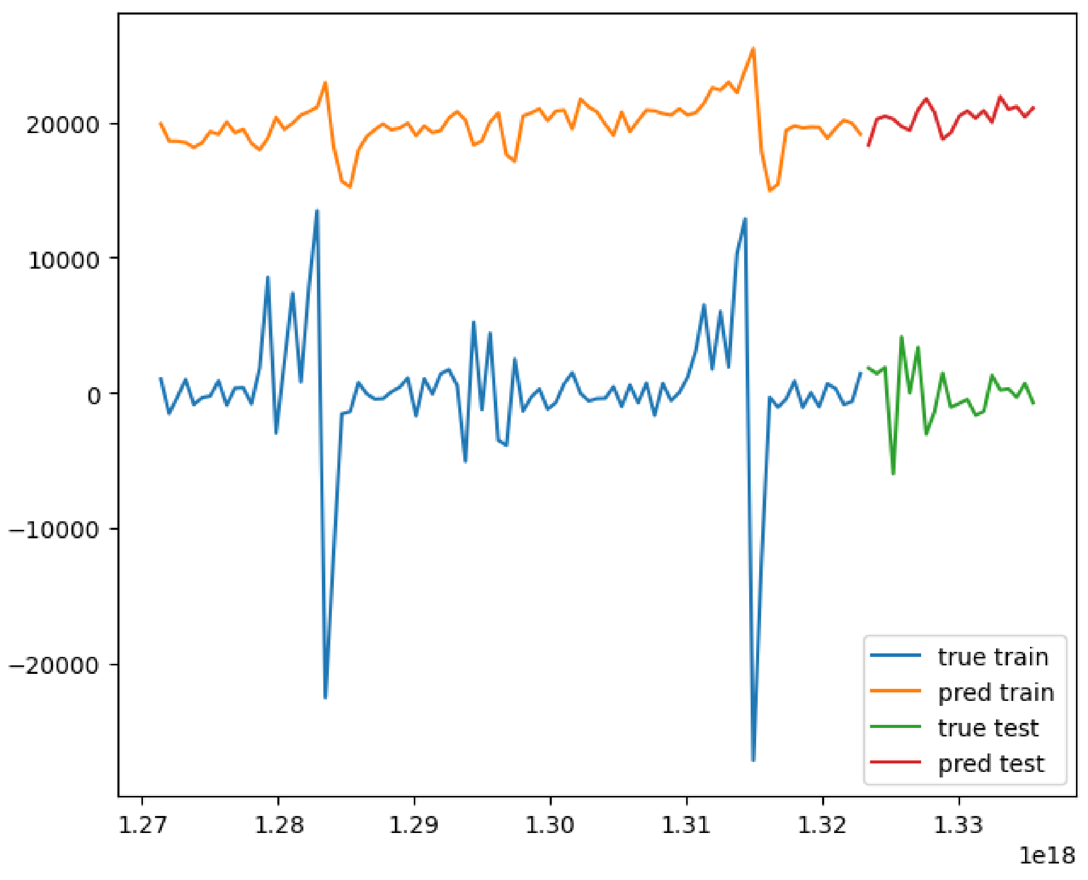
```
<matplotlib.legend.Legend at 0x7f212ce205b0>
```



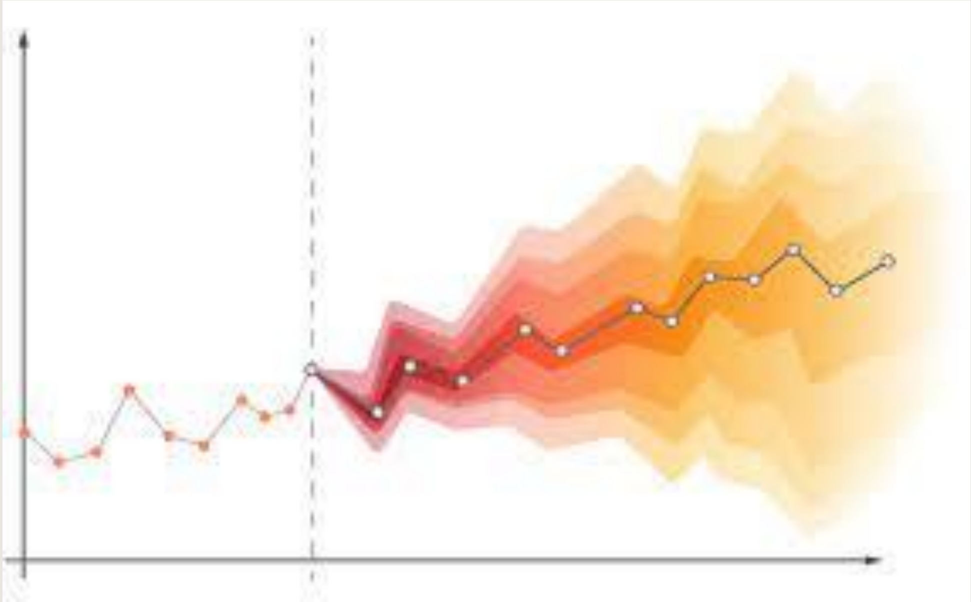
normalized

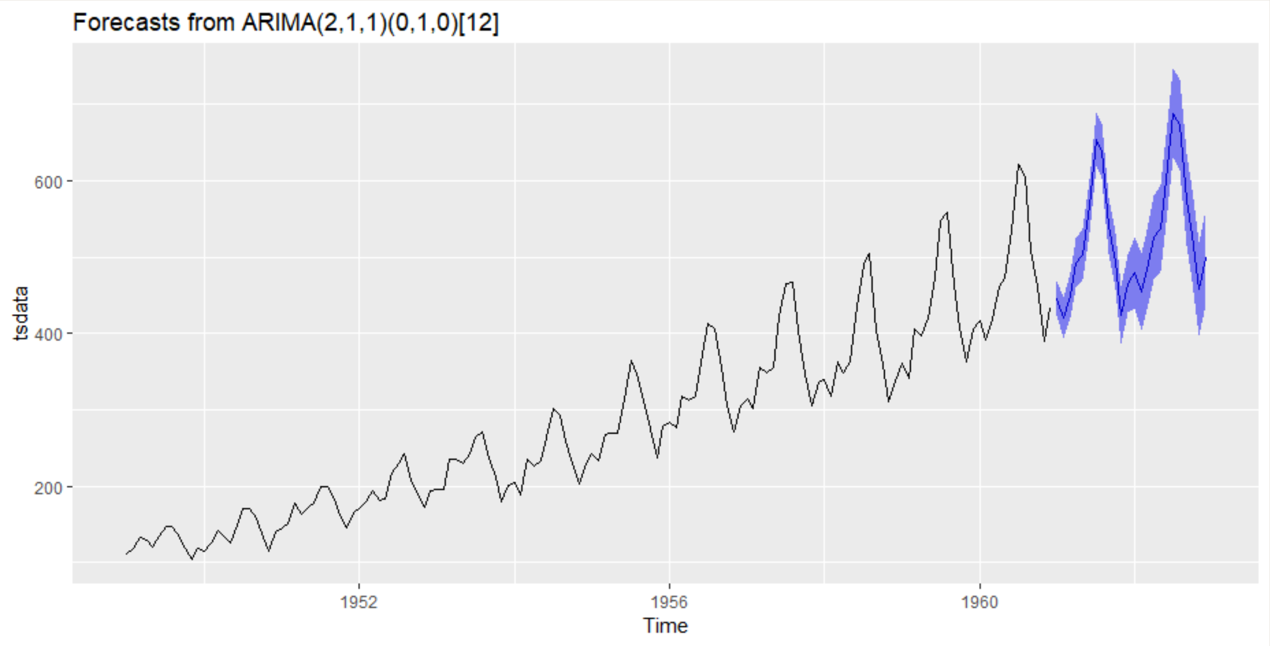
Non normalized





TIME PREDICTION IS NOT EASY



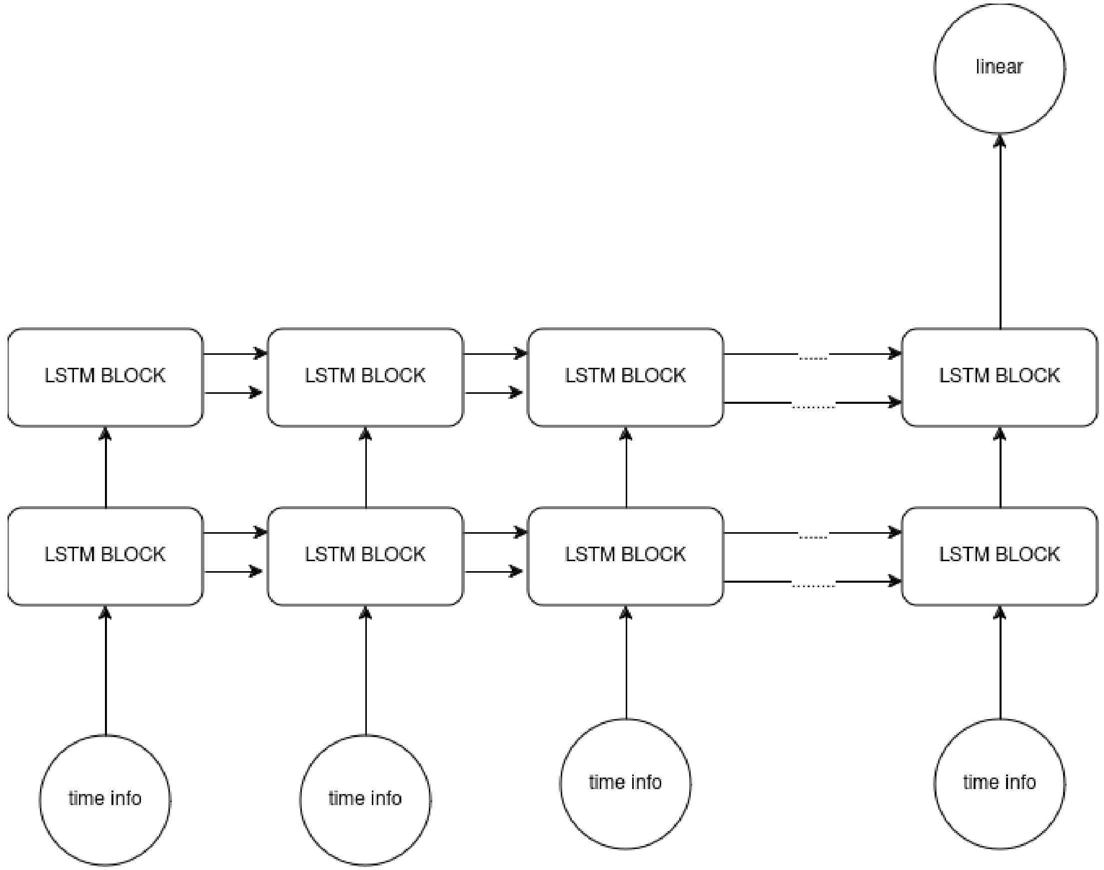


RNN(SPECIFICALLY LSTM) TO THE RESCUE

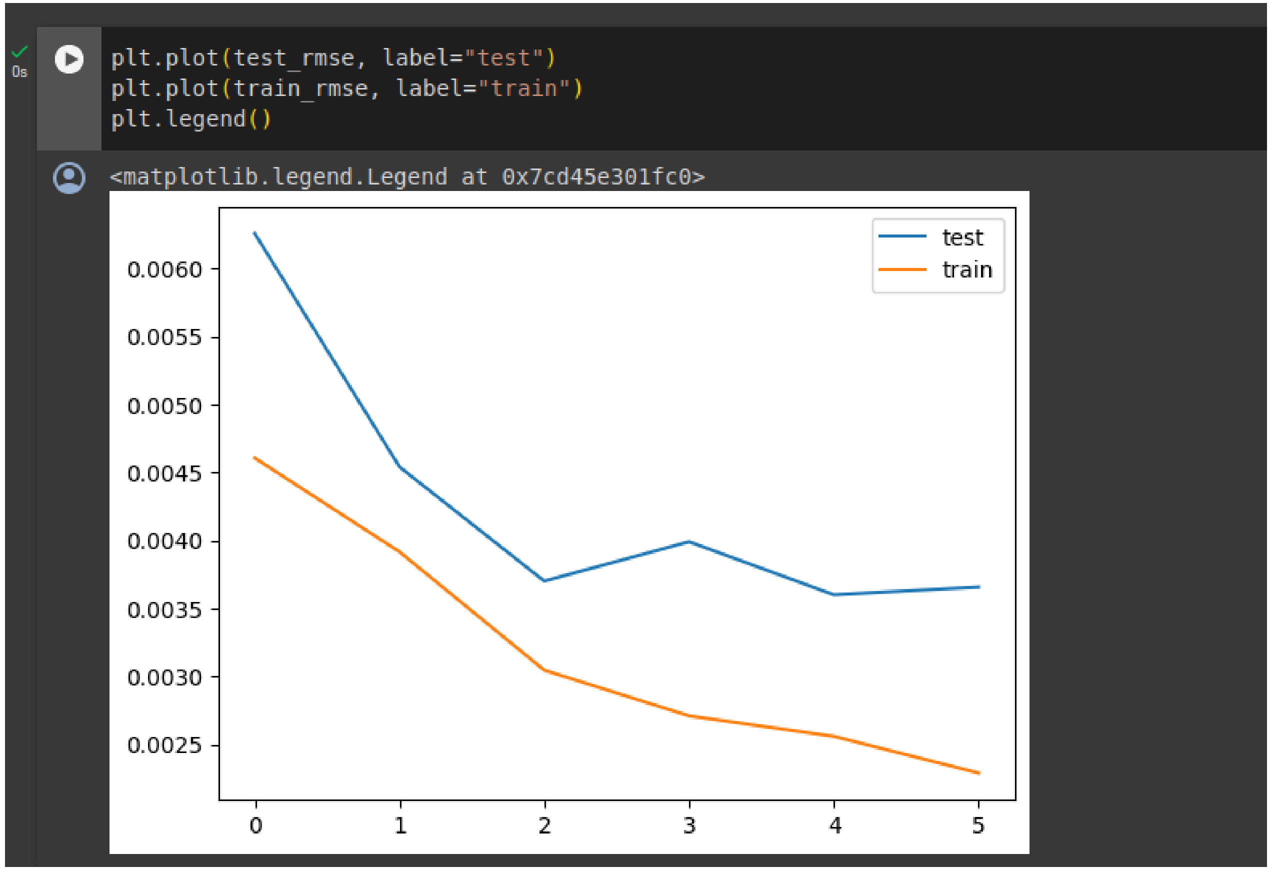
- Has built in time flow
- Is proven to be effective
- Already solved the vanishing gradient problem
- Has comparitively low number of weights for the amount of data

OUR ARCHITECTURE

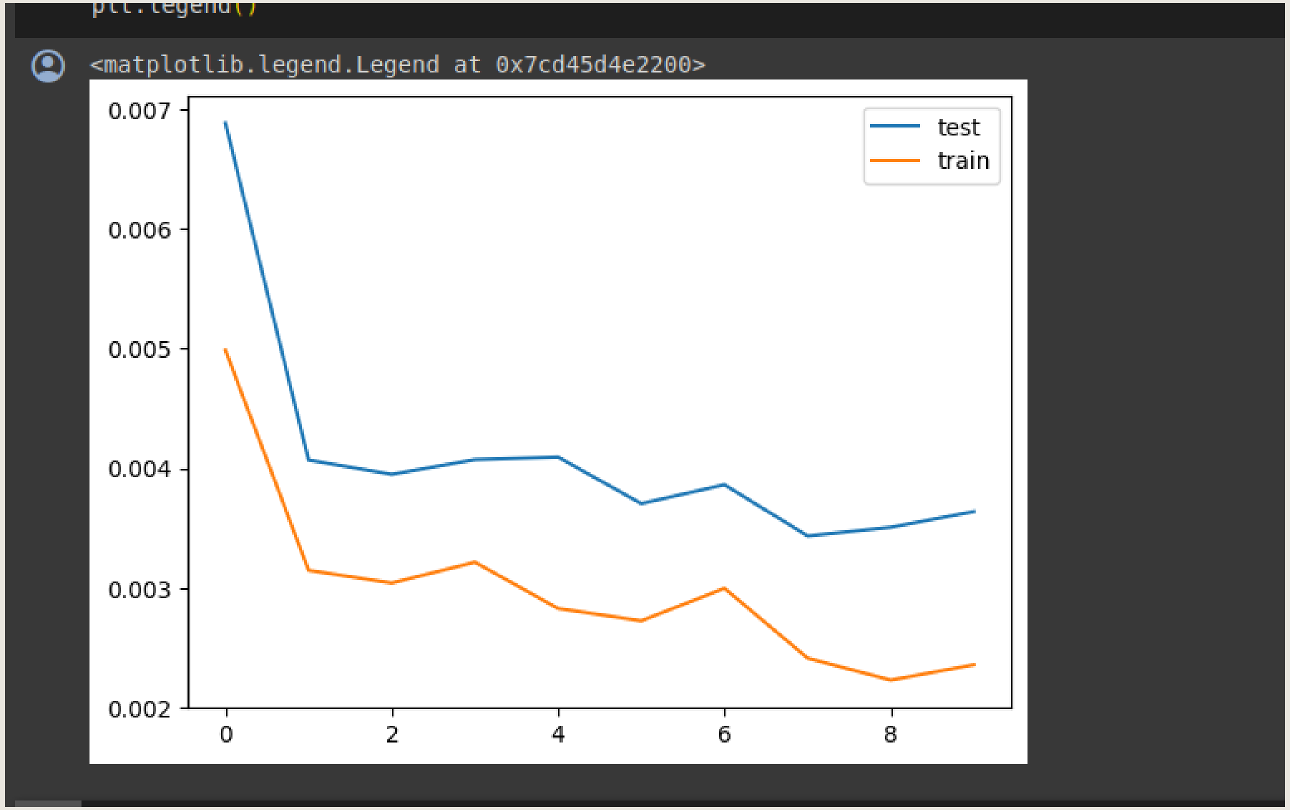
- **Two layered LSTM**
- **Hidden state passed through each block =50**
- **At each time period we get the all the relveant data from today(including sin cos of the date instead of the date)**
- **Additionally we add the last week's week-diff**



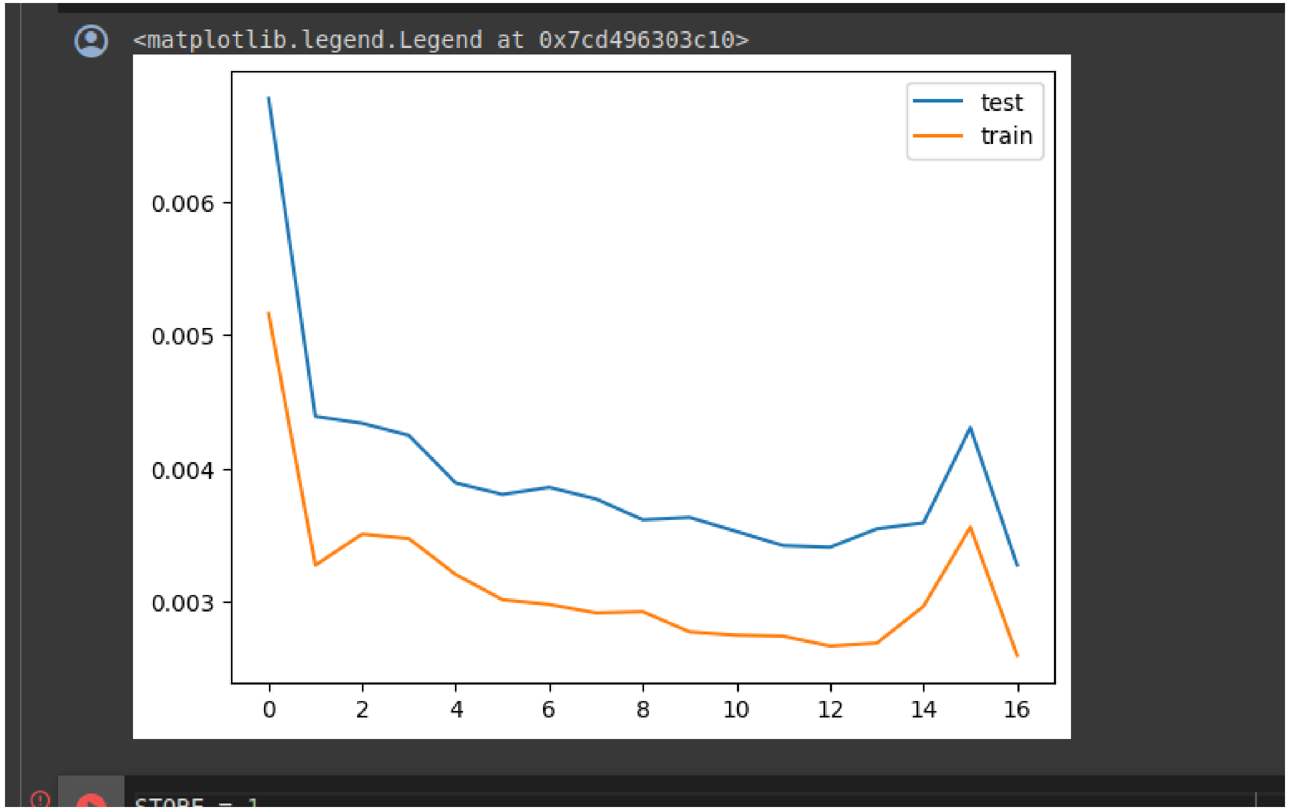
RESULT AFTER 5 EPOCHS WITH 5 LSTM LAYERS (SEEMS TO BE OVERFITTING BUT CANT BE SURE)



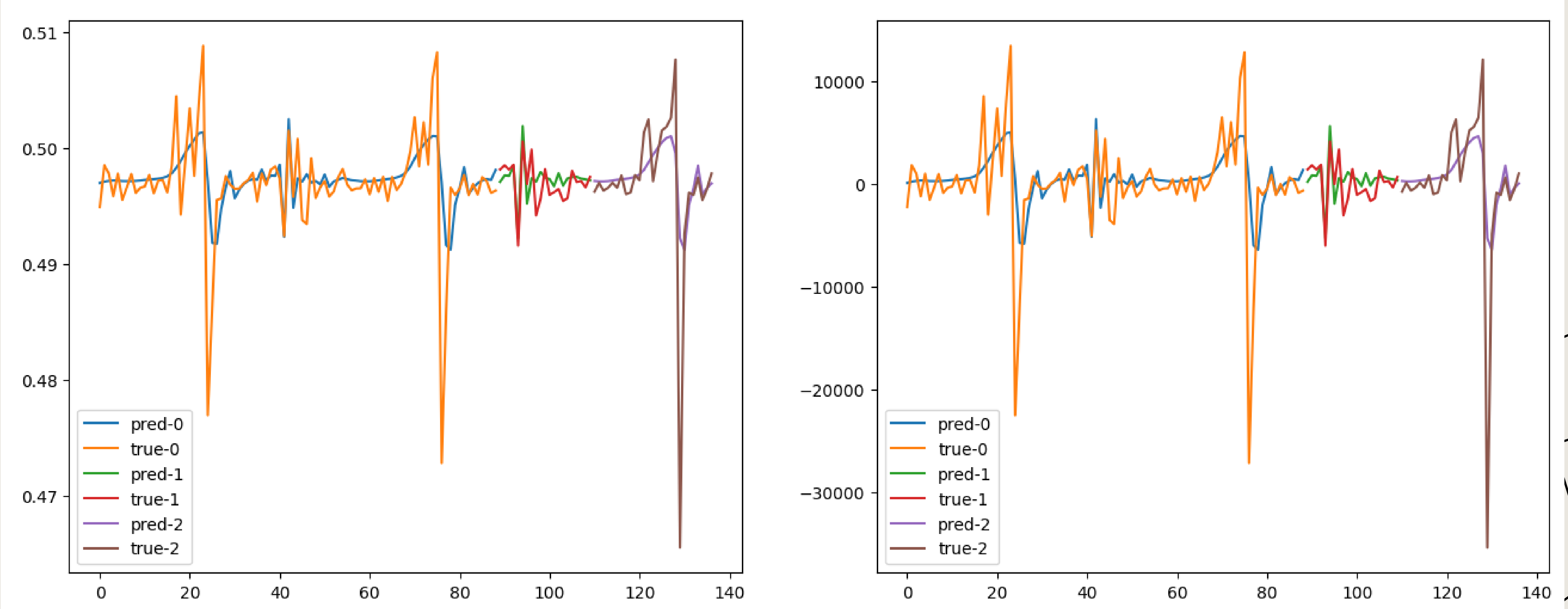
RESULTS AFTER 30 EPOCHS WITH ONLY 2 LSTM LAYERS(OBSERVED EVERY 3 EPOCHS)



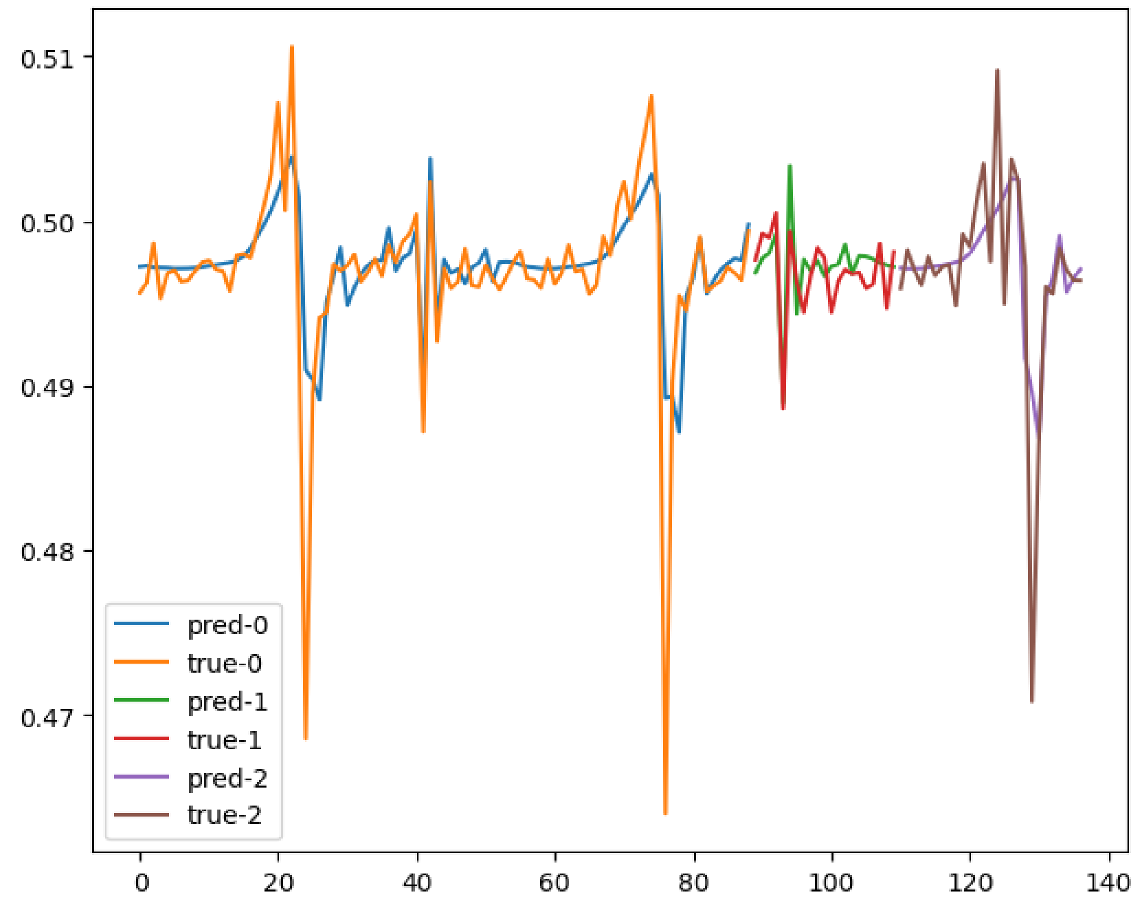
NEED TO SOLVE OVERFITTING PROBLEM....
DROPOUT!!!(0.2) AND 50 EPOCHS(OBSERVED EVERY 3 EPOCHS)

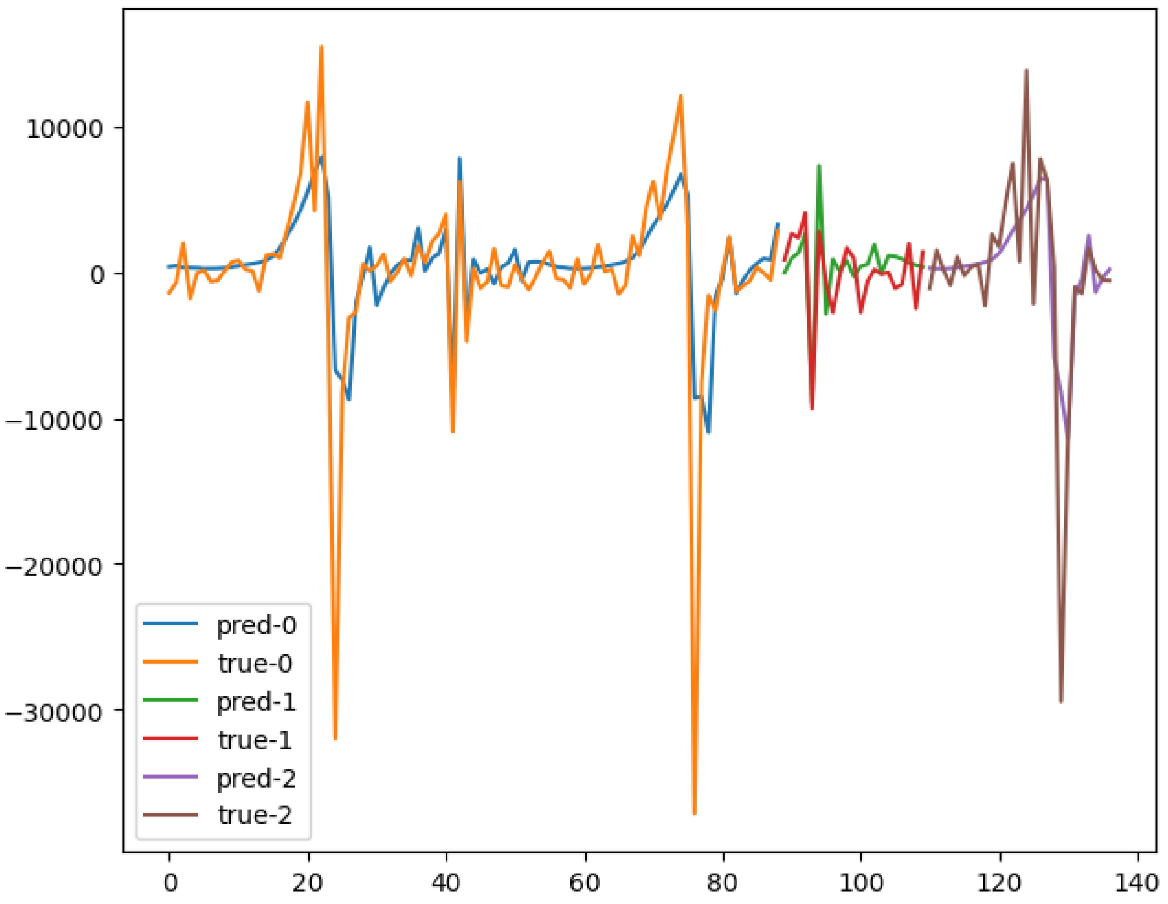


OBSERVED RESULTS IN ARBITRARY STORE AND SPECIFIC DEPARTMENT (DEPT=3)

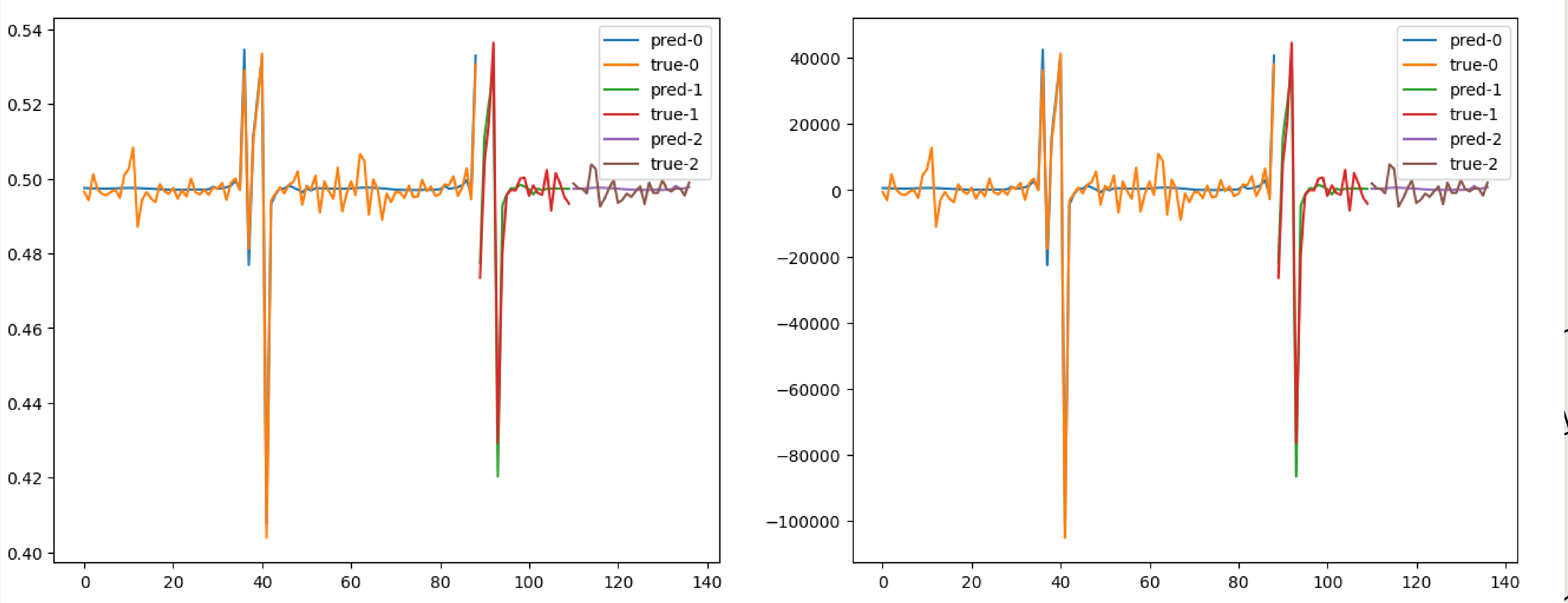


OBSERVED RESULTS IN THE SAME DEPARTMENT BUT AT A DIFFERENT STORE

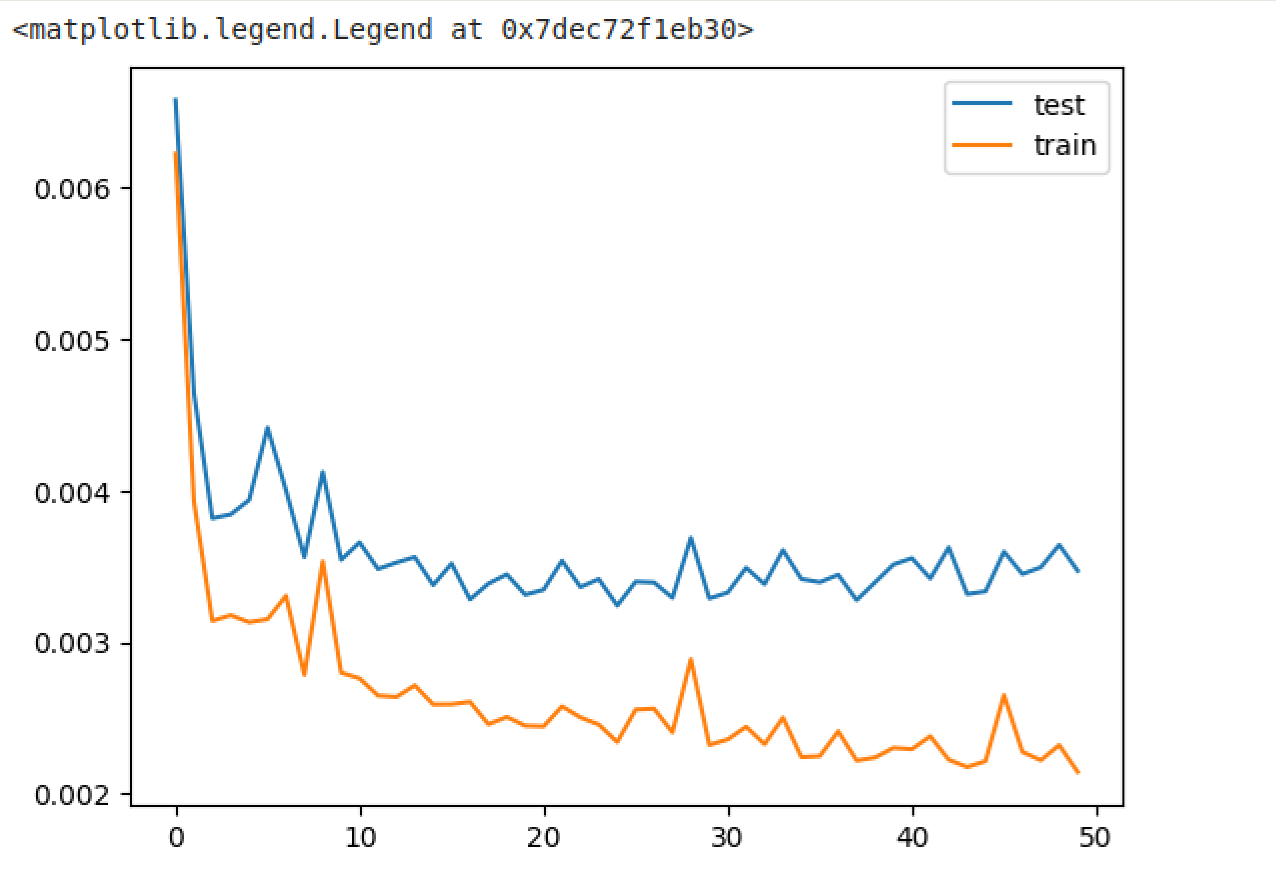




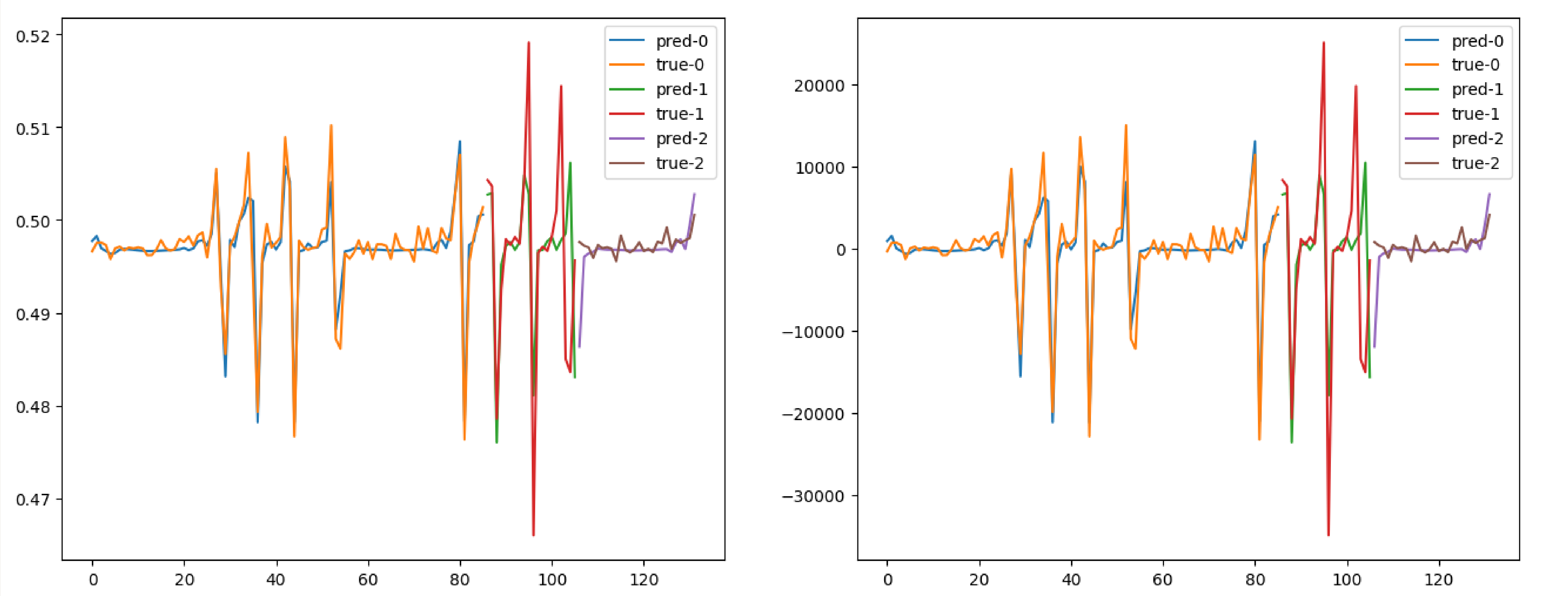
RESULTS IN A DIFFERENT DEPARTMENT



USING LOOKBACK = 10 AND 150 EPOCHS

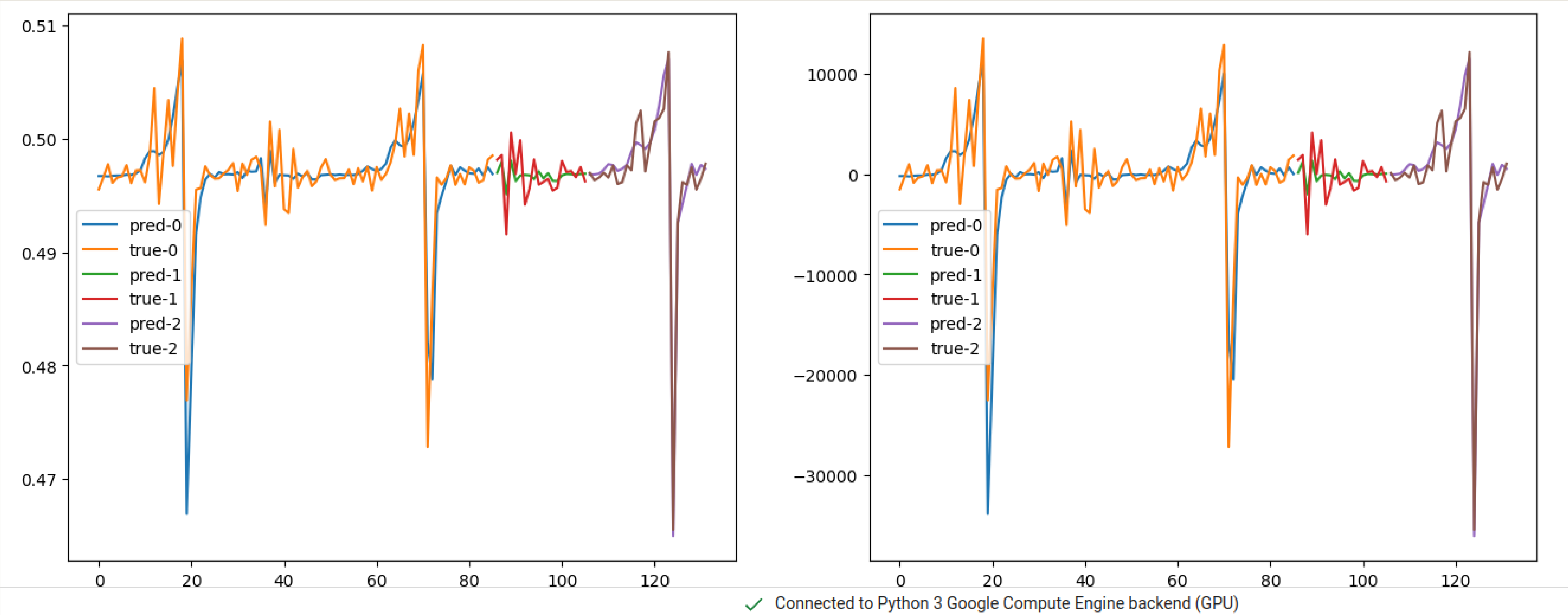


RESULTS



0.52
0.51
0.50
0.49
0.48
0.47
0
20
40
60
80
100
120
pred-0
true-0
pred-1
true-1
pred-2
true-2
20000
10000
0
−10000
−20000
−30000
0
20
40
60
80
100
120
pred-0
true-0
pred-1
true-1
pred-2
true-2

RESULTS


0.51
0.50
0.49
0.48
0.47
0
20
40
60
80
100
120
pred-0
true-0
pred-1
true-1
pred-2
true-2
10000
0
−10000
−20000
−30000
Connected to Python 3 Google Compute Engine backend (GPU)

AND NOW CLAP FOR THE FINAL SCORE

THANK YOU

We got Loss of 0.0026

Lior Shiboli and Omer Priel