In [ ]:
```
import numpy as np
import pandas as pd
```

Data preparation

In [ ]:
```
df = pd.read_csv("sales_data_2017_2018_for_tableau_with_new_date_columns.csv
df.head(5)
```

Out[ ]:

	receipt_id	date	hour	quarter	year	month_number	month_name	day_of_wee
0	14b5b35b-4155-45c5-9fa1-58e81d508a25	4/2/2018 2:16:32 PM	14	2	2018	4	April	
1	45755456-0890-450a-af1b-b10b0c197af4	1/25/2018 11:54:20 AM	11	1	2018	1	January	T
2	48910672-6e70-4c1a-8efc-e348c45d519c	4/13/2018 5:40:15 PM	17	2	2018	4	April	
3	dd2882f2-4211-4828-bccb-b53821d29559	1/11/2018 1:44:42 PM	13	1	2018	1	January	T
4	142d4d58-c63b-4fff-80c0-da43e87a2070	1/18/2018 2:28:24 PM	14	1	2018	1	January	T

5 rows × 23 columns

Filter the result to make sure that dataset only contains data between 7am-7pnm for futher predictions

In [ ]:
```
df_7_to_7 = df[(df["hour"] >= 7) & (df["hour"] <= 19)]
df_7_to_7.shape
```

Out[ ]:
```
(310466, 23)
```

In [ ]:
```
total_selling_price_by_year_month = df_7_to_7.groupby(["year", "month_number
total_selling_price_by_year_month.name = "total_selling_price"
selling_price_df = total_selling_price_by_year_month.to_frame()
selling_price_df.reset_index(inplace=True)
selling_price_df
```

Out[ ]:

	year	month_number	total_selling_price
0	2017	1	885.812220
1	2017	2	927.956955
2	2017	3	430.441765
3	2017	4	29886.365714
4	2017	5	44960.825133
5	2017	6	38619.024525
6	2017	7	43974.944089
7	2017	8	44208.190804
8	2017	9	42977.653936
9	2017	10	47247.880915
10	2017	11	46407.182879
11	2017	12	48349.830237
12	2018	1	55642.787223
13	2018	2	47742.139886
14	2018	3	48966.504335
15	2018	4	39543.946056
16	2018	5	40497.801060
17	2018	6	38813.760726
18	2018	7	41623.188615
19	2018	8	40864.303817
20	2018	9	39152.811490
21	2018	10	41046.588170
22	2018	11	40732.782385
23	2018	12	37800.686131

Seems like data of the first three months in 2017 are outliers, we extract them.

In [ ]:

```
cleaned_selling = selling_price_df.loc[3:, :]
cleaned_selling.reset_index()
```

Out[ ]:

	index	year	month_number	total_selling_price
0	3	2017	4	29886.365714
1	4	2017	5	44960.825133
2	5	2017	6	38619.024525
3	6	2017	7	43974.944089
4	7	2017	8	44208.190804
5	8	2017	9	42977.653936
6	9	2017	10	47247.880915
7	10	2017	11	46407.182879
8	11	2017	12	48349.830237
9	12	2018	1	55642.787223
10	13	2018	2	47742.139886
11	14	2018	3	48966.504335
12	15	2018	4	39543.946056
13	16	2018	5	40497.801060
14	17	2018	6	38813.760726
15	18	2018	7	41623.188615
16	19	2018	8	40864.303817
17	20	2018	9	39152.811490
18	21	2018	10	41046.588170
19	22	2018	11	40732.782385
20	23	2018	12	37800.686131

We are preparing data for training and testing a model to predict the selling price in the next three months. Our goal is to also predict quarterly prices, but the 2017 Q1 data points are outliers and do not provide enough information to train the algorithms to predict for 2019 Q1.

Linear Regression Method

In [ ]:
```
X = [i for i in range(0, 21)]
Y = cleaned_selling.loc[:, "total_selling_price"].values
```

Here, to decide how many months should be included in the training data, we tried out different combinations, we then discovered that using all data starting from 2017.03 till 2018.12 is the best choice.

In [ ]:
```
x_train, x_test = X[:18], X[18:]
y_train, y_test = Y[:18], Y[18:]
```

In [ ]:
```
x_train2 = X[9:18]
y_train2 = Y[9:18]
```

In [ ]:
```
from sklearn.linear_model import LinearRegression

m = LinearRegression()
m2 = LinearRegression()
```

In [ ]:
```
_x_train = np.array(x_train).reshape(-1, 1)
_x_test = np.array(x_test).reshape(-1, 1)
```

In [ ]:
```
_ = m.fit(_x_train, y_train)
```

In [ ]:
```
m.intercept_, m.coef_
```

Out[ ]:
```
(42946.28835289035, array([42.13039927]))
```

In [ ]:
```
m.score(_x_test, y_test)
```

Out[ ]:
```
-7.112917722788609
```

R^2 score = -7.112917722788609, which represents a really bad regression model

In [ ]:
```
LR_result = m.predict([[22], [23], [24]])
```

In [ ]:
```
d = {"YYYY.MM": ["2019.01", "2019.02", "2019.03"], "LR_total_selling_price":
LR_result_df = pd.DataFrame(data=d)
LR_result_df.set_index("YYYY.MM")
```

Out[ ]:

	LR_total_selling_price
YYYY.MM	
2019.01	43873.157137
2019.02	43915.287536
2019.03	43957.417935

In [ ]:
```
import torch
import torch.nn as nn
```

Simple neural network

In [ ]:
```
seq_model = nn.Sequential(
    nn.Linear(1, 3),
    nn.Tanh(),
    nn.Linear(3, 1)
)
seq_model
```

Out[ ]:
```
Sequential(
  (0): Linear(in_features=1, out_features=3, bias=True)
  (1): Tanh()
  (2): Linear(in_features=3, out_features=1, bias=True)
)
```

In [ ]:
```
def training_loop(n_epochs, optimiser, model, loss_fn, X_train, X_val, y_tra
    for epoch in range(1, n_epochs + 1):
        output_train = model(X_train)  # forwards pass
        loss_train = loss_fn(output_train, y_train)  # calculate loss
        output_val = model(X_val)
```

```
        loss_val = loss_fn(output_val, y_val)

        optimiser.zero_grad()  # set gradients to zero
        loss_train.backward()  # backwards pass
        optimiser.step()  # update model parameters
        if epoch == 1 or epoch % 10000 == 0:
            print(f"Epoch {epoch}, Training loss {loss_train.item():.4f},"
                  f" Validation loss {loss_val.item():.4f}")
```

In [ ]:
```
optimiser = torch.optim.SGD(seq_model.parameters(), lr=1e-3)
training_loop(
    n_epochs = 100000,
    optimiser = optimiser,
    model = seq_model,
    loss_fn = nn.MSELoss(),
    X_train = torch.from_numpy(_x_train).float(),
    X_val = torch.from_numpy(_x_test).float(),
    y_train = torch.from_numpy(y_train).float(),
    y_val = torch.from_numpy(y_test).float())
```

```
/Users/jojogong3736/opt/anaconda3/lib/python3.9/site-packages/torch/nn/modul
es/loss.py:530: UserWarning: Using a target size (torch.Size([18])) that is
different to the input size (torch.Size([18, 1])). This will likely lead to
incorrect results due to broadcasting. Please ensure they have the same siz
e.
  return F.mse_loss(input, target, reduction=self.reduction)
/Users/jojogong3736/opt/anaconda3/lib/python3.9/site-packages/torch/nn/modul
es/loss.py:530: UserWarning: Using a target size (torch.Size([3])) that is d
ifferent to the input size (torch.Size([3, 1])). This will likely lead to in
correct results due to broadcasting. Please ensure they have the same size.
  return F.mse_loss(input, target, reduction=self.reduction)
Epoch 1, Training loss 1904660608.0000, Validation loss 1590956160.0000
Epoch 10000, Training loss 29385390.0000, Validation loss 13998915.0000
Epoch 20000, Training loss 29385390.0000, Validation loss 13998915.0000
Epoch 30000, Training loss 29385390.0000, Validation loss 13998915.0000
Epoch 40000, Training loss 29385390.0000, Validation loss 13998915.0000
Epoch 50000, Training loss 29385390.0000, Validation loss 13998915.0000
Epoch 60000, Training loss 29385390.0000, Validation loss 13998915.0000
Epoch 70000, Training loss 29385390.0000, Validation loss 13998915.0000
Epoch 80000, Training loss 29385390.0000, Validation loss 13998915.0000
Epoch 90000, Training loss 29385390.0000, Validation loss 13998915.0000
Epoch 100000, Training loss 29385390.0000, Validation loss 13998915.0000
```

In [ ]:
```
data = np.array([[22], [23], [24]])
tensor = torch.from_numpy(data).float()
```

In [ ]:
```
nn_prediction = seq_model(tensor)
```

In [ ]:
```
nn_result = nn_prediction.detach().numpy().flatten()
```

In [ ]:
```
d = {"YYYY.MM": ["2019.01", "2019.02", "2019.03"], "nn_total_selling_price":
nn_result_df = pd.DataFrame(data=d)
nn_result_df.set_index("YYYY.MM")
```

Out[ ]:

	nn_total_selling_price
YYYY.MM	
2019.01	43304.15625
2019.02	43304.15625
2019.03	43304.15625

Long Short Term Memory network

In [ ]:
```
class LSTM(nn.Module):
    def __init__(self, hidden_layers=64):
        super(LSTM, self).__init__()
        self.hidden_layers = hidden_layers
        # lstm1, lstm2, linear are all layers in the network
        self.lstm1 = nn.LSTMCell(1, self.hidden_layers)
        self.lstm2 = nn.LSTMCell(self.hidden_layers, self.hidden_layers)
        self.linear = nn.Linear(self.hidden_layers, 1)

    def forward(self, y, future=0):
        outputs, num_samples = [], y.size(0)
        h_t = torch.zeros(num_samples, self.hidden_layers, dtype=torch.float
        c_t = torch.zeros(num_samples, self.hidden_layers, dtype=torch.float
        h_t2 = torch.zeros(num_samples, self.hidden_layers, dtype=torch.floa
        c_t2 = torch.zeros(num_samples, self.hidden_layers, dtype=torch.floa

        for time_step in y.split(1, dim=1):
            # N, 1
            h_t, c_t = self.lstm1(time_step, (h_t, c_t)) # initial hidden ar
            h_t2, c_t2 = self.lstm2(h_t, (h_t2, c_t2)) # new hidden and cell
            output = self.linear(h_t2) # output from the last FC layer
            outputs.append(output)

        for i in range(future):
            # this only generates future predictions if we pass in future_pr
            # mirrors the code above, using last output/prediction as input
            h_t, c_t = self.lstm1(output, (h_t, c_t))
            h_t2, c_t2 = self.lstm2(h_t, (h_t2, c_t2))
            output = self.linear(h_t2)
            outputs.append(output)
        # transform list to tensor
        outputs = torch.cat(outputs, dim=1)
        return outputs
```

In [ ]:
```
train_input = torch.from_numpy(_x_train).float()
train_target = torch.from_numpy(y_train).float()
```

In [ ]:
```
test_input = torch.from_numpy(_x_test).float()
test_target = torch.from_numpy(y_test).float()
```

In [ ]:
```
model = LSTM()
criterion = nn.MSELoss()
optimiser = torch.optim.LBFGS(model.parameters(), lr=0.08)
```

In [ ]:
```
import matplotlib.pyplot as plt
```

In [ ]:
```
def training_loop(n_epochs, model, optimiser, loss_fn,
                  train_input, train_target, test_input, test_target):
    for i in range(n_epochs):
```

```
        def closure():
            optimiser.zero_grad()
            out = model(train_input)
            loss = loss_fn(out, train_target)
            loss.backward()
            return loss
        optimiser.step(closure)
        with torch.no_grad():
            future = 3
            pred = model(test_input, future=future)
            # use all pred samples, but only go to 999
            loss = loss_fn(pred[:, :-future], test_target)
            y = pred.detach().numpy()
        if (i % 10 == 0):
            # draw figures
            plt.figure(figsize=(12, 6))
            plt.title(f"Step {i+1}")
            plt.xlabel("x")
            plt.ylabel("y")
            plt.xticks(fontsize=20)
            plt.yticks(fontsize=20)
            n = train_input.shape[1]  # 999
            def draw(yi, colour):
                plt.plot(np.arange(n), yi[:n], colour, linewidth=2.0)
                plt.plot(np.arange(n, n+future), yi[n:], colour+":", linewid
            draw(y[0], 'r')
            draw(y[1], 'b')
            draw(y[2], 'g')
            plt.savefig("predict%d.png" % i, dpi=200)
            plt.close()
            # print the loss
            # out = model(train_input)
            # loss_print = loss_fn(out, train_target)
            print("Step: {}, Loss: {}".format(i, loss))
```

In [ ]:

```
training_loop(100, model, optimiser, nn.MSELoss(),
                  train_input, train_target, test_input, test_target)
```

```
/Users/jojogong3736/opt/anaconda3/lib/python3.9/site-packages/torch/nn/modul
es/loss.py:530: UserWarning: Using a target size (torch.Size([18])) that is
different to the input size (torch.Size([18, 1])). This will likely lead to
incorrect results due to broadcasting. Please ensure they have the same siz
e.
  return F.mse_loss(input, target, reduction=self.reduction)
/Users/jojogong3736/opt/anaconda3/lib/python3.9/site-packages/torch/nn/modul
es/loss.py:530: UserWarning: Using a target size (torch.Size([3])) that is d
ifferent to the input size (torch.Size([3, 1])). This will likely lead to in
correct results due to broadcasting. Please ensure they have the same size.
  return F.mse_loss(input, target, reduction=self.reduction)
Step: 0, Loss: 223763216.0
Step: 10, Loss: 14006988.0
Step: 20, Loss: 14001472.0
Step: 30, Loss: 14000960.0
Step: 40, Loss: 14000960.0
Step: 50, Loss: 14000960.0
Step: 60, Loss: 14000960.0
Step: 70, Loss: 14000960.0
Step: 80, Loss: 14000960.0
Step: 90, Loss: 14000960.0
```

In [ ]:

```
input = torch.from_numpy(np.array(X).reshape(-1, 1)).float()

pred = model(input, future = 2)
```

```
y = pred.detach().numpy()
```

In [ ]:
```
LMST_result = y[y.shape[0]-1]
```

In [ ]:
```
d = {"YYYY.MM": ["2019.01", "2019.02", "2019.03"], "LMST_total_selling_price
LMST_result_df = pd.DataFrame(data=d)
LMST_result_df.set_index("YYYY.MM")
```

Out[ ]:

YYYY.MM	LMST_total_selling_price
2019.01	43304.453125
2019.02	49826.804688
2019.03	51395.523438

In [ ]:
```
result = pd.concat([LR_result_df, nn_result_df, LMST_result_df], axis=1)
result
```

Out[ ]:

	YYYY.MM	LR_total_selling_price	YYYY.MM	nn_total_selling_price	YYYY.MM	LMST_tota
0	2019.01	43873.157137	2019.01	43304.15625	2019.01	
1	2019.02	43915.287536	2019.02	43304.15625	2019.02	
2	2019.03	43957.417935	2019.03	43304.15625	2019.03	

In [ ]:
```
result = result.loc[:,~result.columns.duplicated()].copy()
result.set_index("YYYY.MM")
```

Out[ ]:

YYYY.MM	LR_total_selling_price	nn_total_selling_price	LMST_total_selling_price
2019.01	43873.157137	43304.15625	43304.453125
2019.02	43915.287536	43304.15625	49826.804688
2019.03	43957.417935	43304.15625	51395.523438

Machine learning model:

- Linear Regression model R^2 score: -7.112917722788609

Deep Learning Model:

- Neural Network Model loss: 13998915
- Long Short Term Memory Model loss: 14000476.0