Question 4

a

Let D be a $M \times n \in 0, 1$ binary matrix representing which observations were generated from which Regression Model. For example the matrix:

$$\begin{matrix} 0 & 0 & 1 \\ 1 & 1 & 0 \end{matrix}$$

Is a 2×3 that represents a dataset of 3 observations belonging to one of 2 Regression Models. Observations 1 and 2 belong to Regression Model 2 and observation 3 belongs to Regression Model 1.

Let $\hat{D}$ be the binary matrix of the above format labelling the predictions of Regression Models that each observation belongs to.

$$loss(\hat{D}, \hat{\beta}) = \sum_{j=1}^{M} [\sum_{i=1}^{n} [\hat{D}_{ji} * [(X_i^T \hat{\beta}_j - Y_i)^2 + (1 - D_{ji})]]$$

The first term $(X_i^T \hat{\beta}_j - Y_i)^2$ is a penalty for having errors in $\hat{\beta}_j$. The second term $1 - D_{j,i}$ is a penalty for prediction observation i in the incorrect D_j. Note that the observation is skipped and no loss is applied if the observation does not belong to the predicted regression model, which is why the loss for each observation is multipled by $\hat{D}_{ji}$

In words, the loss function reads: For each observation i that is predicted to be in a regression model $\hat{D}_j$, find the mean squared error of that observation and add a penalty if the observation's predicted $\hat{D}$ differed from its actual D.

When $M = 1$, this just simplifies down to:

$$loss(\hat{\beta}) = \sum_{i=1}^{n} ((X_i^T \hat{\beta}_j - Y_i)^2)$$

b

```
# First partition all observations
For each observation i:
    For each model m:
        -> calculate the absolute distance between the model (m) prediction
           on this observation (i) and the actual value of this observation

    -> assign this observation to the model that it
       had the \emph{smallest} error with

# Next find the coefficients of each model
For each partitioned dataset d:
    -> fit a linear regression model that uses MSE as its loss function
       and has the dataset (d) as its training data

# We now have values for D and \(\hat{B}\) that are minimised and the loss
# can be calculated with my function in a.
```

c

```python
def total_loss(X, y, Z, models):          You, 24 minutes ago • code for 4
    loss = 0
    M = len(models)
    n = X.shape[0]

    for j in range(M):
        model = models[j]

        for i in range(n):
            # If the observation is not partitioned to model "j", we skip
            if Z[i] != j:
                continue

            # Get penalty of the coefficients for this model on this observation
            observation = X[i]
            actual = y[i]
            prediction = model.predict(np.array([observation]))
            coef_penalty = (prediction[0] - actual)**2

            # Cannot know partition_penalty since partition predictions
            # are not provided to this function. Not really sure of the
            # point of this function. Just set penalty to 0.
            partition_penalty = 0

            loss += coef_penalty + partition_penalty

    return loss
```

```python
def q4c():
    data = import_data(Q4_TRAIN_DATA_DIR)
    Xtrain = np.array(data.drop(columns=["Y"]))
    ytrain = np.array(data["Y"])

    mod = LinearRegression().fit(Xtrain, ytrain)
    Z = np.zeros(shape=Xtrain.shape[0]) # all points would belong to a single partition.
    print(total_loss(Xtrain, ytrain, Z, [mod])) # outputs 298.328178158043
```

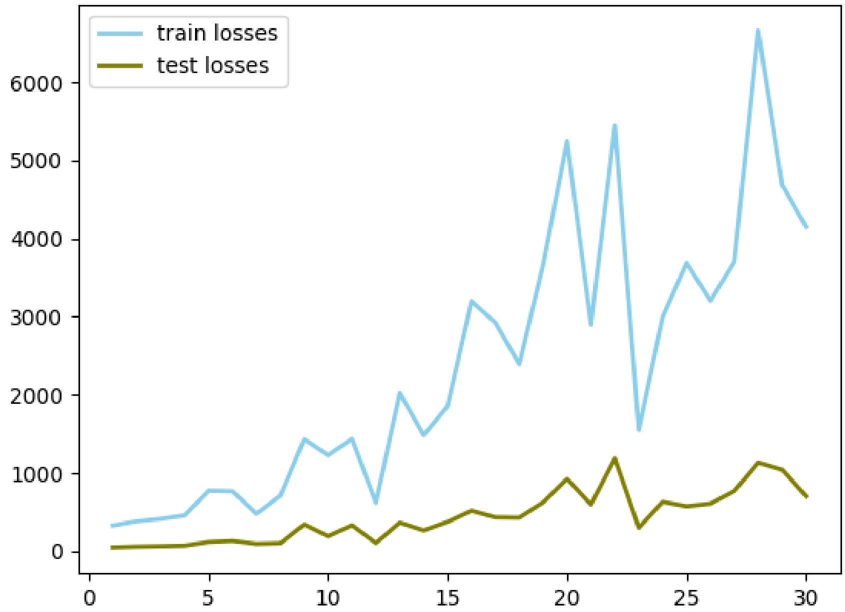
d

```python
def find_partitions(X, y, models):        You, 25 minutes ago • code for
    M = len(models)
    n = X.shape[0]

    Z = []
    for i in range(n):
        observation = X[i]

        # predict this observation on each model. Whichever model
        # predicted the closest to the actual we assume this
        # datapoint belongs to this model.
        pred_diffs = []
        for model in models:
            pred = model.predict(np.array([observation]))
            actual = y[i]

            # Could also square this to get a positive number
            pred_diffs.append(abs(pred - actual))

        # Append index of smallest value to list
        Z.append(pred_diffs.index(min(pred_diffs)))

    return np.array(Z)

def q4d():
    data = import_data(Q4_TEST_DATA_DIR)
    Xtest = np.array(data.drop(columns=["Y"]))
    ytest = np.array(data["Y"])

    # Run with one model
    mod = LinearRegression().fit(Xtest, ytest)

    Z = find_partitions(Xtest, ytest, [mod])
    print(total_loss(Xtest, ytest, Z, [mod]))
```

e

See code for this question here 0.1

```
M  |         train         |         test          |
5  |  773.2139301322158    |  116.60294714118255   |
10 |  1226.975556530428    |  190.77503856677967   |
15 |  1854.7758161852978 |  374.3318184723467    |
20 |  5245.153194657786    |  924.8003322754047    |
25 |  3687.1371589765313 |  572.1619888958278    |
30 |  4150.443300036157    |  703.5325244593158    |
```



Based on the results, I would guess that the most reasonable value for M is around 6 or 8. Even though the train losses were lower when M < 5, the test losses were approximately the same for M from 0 to 8 and a lower value for M would be favoured since each of the models gets trained on more data.

Appendix

0.1 q4e

```
# So that we don't have to train thousands of models
def generate_Z(M, n):       You, seconds ago • code for 4

    # np.tile is such a misdirect, you have to find how many
    # times you want to repeat and then append two lists together.
    #
    # Sorry, I've been going at this exam for years and am getting
    # to the end of a tether.
    Z = []
    for i in range(n):
        Z.append(i % M)

    return np.array(Z)
```

```
def q4e():       You, a minute ago • code for 4
    # import train and test data
    data_train = import_data(Q4_TRAIN_DATA_DIR)
    Xtrain = np.array(data_train.drop(columns=["Y"]))
    ytrain = np.array(data_train["Y"])

    data_test = import_data(Q4_TEST_DATA_DIR)
    Xtest = np.array(data_test.drop(columns=["Y"]))
    ytest = np.array(data_test["Y"])

    # model
    model = LinearRegression()

    # Iterate over values of M
    train_losses = []
    test_losses = []

    print("M  |         train         |         test          |")
    MAX_M = 30
    for M in range(1,MAX_M + 1):
        # Partition data (done in a mechanical way for exam testing)
        Ztrain = generate_Z(M, Xtrain.shape[0])
        Ztest = generate_Z(M, Xtest.shape[0])

        # Train models on their own data
        train_models = []
        for i in range(M):
            indices = np.where(Ztrain == i)
            fitted = model.fit(Xtrain[indices], ytrain[indices])
            train_models.append(fitted)
```

```python

        test_models = []
        for i in range(M):
            indices = np.where(Ztest == i)
            fitted = model.fit(Xtest[indices], ytest[indices])
            test_models.append(fitted)

        # Calculate loss
        train_loss = total_loss(Xtrain, ytrain, Ztrain, train_models)
        test_loss = total_loss(Xtest, ytest, Ztest, test_models)

        train_losses.append(train_loss)
        test_losses.append(test_loss)

        if M % 5 == 0:
            padding=""
            if M == 5:
                padding = " "
            print(F"{M}{padding} |  {train_loss}  |  {test_loss}  |")

    # Plot everything
    X = list(range(1, MAX_M + 1))
    plt.plot(X, train_losses, color="skyblue", label="train losses", linewidth=2)
    plt.plot(X, test_losses, color="olive", label="test losses", linewidth=2)
    plt.legend()
    plt.savefig("outputs/q4e.png")
```