Use markdown to label each (sub)question neatly.

This notebook serves as your report. All your answers should be presented within it.

You can submit multiple notebooks (e.g. 1 notebook per part / question).

Before submission, remember to tidy up the notebook and retain only relevant parts.

PartB Question 1

In [ ]:
```
# Setting the seed here is sufficient.
# If you don't plan to use these starter code, make sure you add this cell.

SEED = 42

import os
os.environ['TF_CUDNN_DETERMINISTIC'] = '1'

os.environ["CUDA_VISIBLE_DEVICES"] = "-1"

import random
random.seed(SEED)

import numpy as np
np.random.seed(SEED)

import tensorflow as tf
tf.random.set_seed(SEED)
```

In [ ]:
```
import graphviz
import pydot_ng as pydot
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import Normalization, StringLookup, IntegerLookup
from math import floor
```

In [ ]:
```
import pandas as pd
df = pd.read_csv('hdb_price_prediction.csv')
df
```

Out[ ]:

	month	year	full_address	nearest_stn	dist_to_nearest_stn	dist_to_dhoby	degree_centrality	e
0	1	2017	406 ANG MO KIO AVENUE 10	Ang Mo Kio	1.007264	7.006044	0.016807	
1	1	2017	108 ANG MO KIO AVENUE 4	Ang Mo Kio	1.271389	7.983837	0.016807	
2	1	2017	602 ANG MO KIO AVENUE 5	Yio Chu Kang	1.069743	9.090700	0.016807	
3	1	2017	465 ANG MO KIO AVENUE 10	Ang Mo Kio	0.946890	7.519889	0.016807	
4	1	2017	601 ANG MO KIO AVENUE 5	Yio Chu Kang	1.092551	9.130489	0.016807	
...	...	...	...	...	...	...	...	
133407	6	2022	877 YISHUN STREET 81	Khatib	0.475885	12.738721	0.016807	
133408	1	2022	633 YISHUN STREET 61	Khatib	0.774113	13.229106	0.016807	
133409	2	2022	633 YISHUN STREET 61	Khatib	0.774113	13.229106	0.016807	
133410	2	2022	632 YISHUN STREET 61	Khatib	0.700595	13.222912	0.016807	
133411	5	2022	605 YISHUN STREET 61	Khatib	0.603845	13.592586	0.016807	

133412 rows × 13 columns

In [ ]:

```python
# The functions in this cell are adapted from https://keras.io/examples/structured_data,
# It is the same link as the one mentioned in the question paper (Q1b)

def dataframe_to_dataset(dataframe):
    dataframe = dataframe.copy()

    labels = dataframe.pop("resale_price")
    ds = tf.data.Dataset.from_tensor_slices((dict(dataframe), labels))
    ds = ds.shuffle(buffer_size=len(dataframe))
    return ds


def encode_numerical_feature(feature, name, dataset):
    # Create a Normalization layer for our feature
    normalizer = Normalization()

    # Prepare a Dataset that only yields our feature
```

```
    feature_ds = dataset.map(lambda x, y: x[name])
    feature_ds = feature_ds.map(lambda x: tf.expand_dims(x, -1))

    # Learn the statistics of the data
    normalizer.adapt(feature_ds)

    # Normalize the input feature
    encoded_feature = normalizer(feature)
    return encoded_feature


def encode_categorical_feature(feature, name, dataset, is_string):
    lookup_class = StringLookup if is_string else IntegerLookup
    # Create a lookup layer which will turn strings into integer indices
    lookup = lookup_class(output_mode="binary") # NOTE: as mentioned in the question pa

    # Prepare a Dataset that only yields our feature
    feature_ds = dataset.map(lambda x, y: x[name])
    feature_ds = feature_ds.map(lambda x: tf.expand_dims(x, -1))

    # Learn the set of possible string values and assign them a fixed integer index
    lookup.adapt(feature_ds)

    # Turn the string input into integer indices
    encoded_feature = lookup(feature)
    return encoded_feature
```

In [ ]:
```
from keras import backend as K
from math import sqrt

def root_mean_squared_error(y_true, y_pred):
        return K.sqrt(K.mean(K.square(y_pred - y_true)))
```

In [ ]:
```
def r2(y_true, y_pred):
    '''
    # Obtained from https://jmlb.github.io/ml/2017/03/20/CoeffDetermination_CustomMetri
    # TODO: you have to find out how to use it in your code
    '''
    SS_res = K.sum(K.square( y_true - y_pred ))
    SS_tot = K.sum(K.square( y_true - K.mean(y_true) ) )
    return ( 1 - SS_res/(SS_tot + K.epsilon()) )
```

Question 1

In [ ]:
```
# Split data

train_dataframe = df[df['year']<= 2020]
test_dataframe =  df[df['year']>2020]
```

In [ ]:
```
train_dataframe.head()
```

Out[ ]:

	month	year	full_address	nearest_stn	dist_to_nearest_stn	dist_to_dhoby	degree_centrality	eigenv
0	1	2017	406 ANG MO KIO AVENUE 10	Ang Mo Kio	1.007264	7.006044	0.016807	
1	1	2017	108 ANG MO KIO AVENUE 4	Ang Mo Kio	1.271389	7.983837	0.016807	
2	1	2017	602 ANG MO KIO AVENUE 5	Yio Chu Kang	1.069743	9.090700	0.016807	
3	1	2017	465 ANG MO KIO AVENUE 10	Ang Mo Kio	0.946890	7.519889	0.016807	
4	1	2017	601 ANG MO KIO AVENUE 5	Yio Chu Kang	1.092551	9.130489	0.016807	

In [ ]:

```
category_not_used = ["full_address", "nearest_stn"]
train_dataframe = train_dataframe.drop(category_not_used, axis = 1)
test_dataframe = test_dataframe.drop(category_not_used, axis = 1)

train_ds = dataframe_to_dataset(train_dataframe)
test_ds = dataframe_to_dataset(test_dataframe)

train_ds = train_ds.batch(256)
test_ds = test_ds.batch(256)
```

Question 1A

Why is this done instead of using random train/test split?

The rationale is to predict resale prices is to used past data as the training dataset to predict future values. Hence the training dataset are used for year <= 2020 and the test dataset are filled by the more recent dataset

Question 1B

In [ ]:

```
#Categorical feature encoded as integer
month = keras.Input(shape=(1,), name="month", dtype="int64")

# Categorical feature encoded as string
flat_model_type = keras.Input(shape=(1,), name="flat_model_type", dtype="string")
storey_range = keras.Input(shape=(1,), name="storey_range", dtype="string")

# Numerical features
dist_to_nearest_stn = keras.Input(shape=(1,), name="dist_to_nearest_stn")
dist_to_dhoby = keras.Input(shape=(1,), name="dist_to_dhoby")
degree_centrality = keras.Input(shape=(1,), name="degree_centrality")
eigenvector_centrality = keras.Input(shape=(1,), name="eigenvector_centrality")
remaining_lease_years = keras.Input(shape=(1,), name="remaining_lease_years")
```

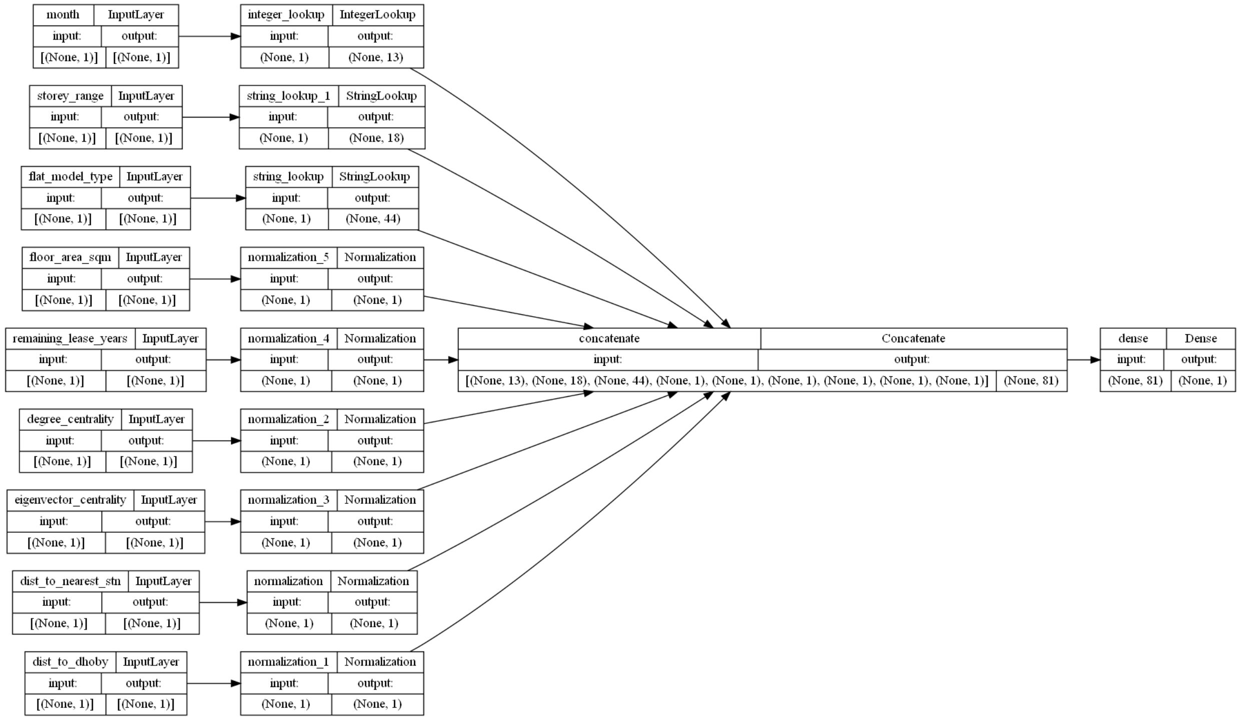
```
floor_area_sqm = keras.Input(shape=(1,), name="floor_area_sqm")
#resale_price = keras.Input(shape=(1,), name="resale_price")

all_inputs = [month,flat_model_type,storey_range,dist_to_nearest_stn,
            dist_to_dhoby,degree_centrality,eigenvector_centrality,remaining_lease_year


#Integer categorical features
month_encoded = encode_categorical_feature(month, "month",train_ds, False)

#String categorical features
flat_model_type_encoded = encode_categorical_feature(flat_model_type, "flat_model_type"
storey_range_encoded = encode_categorical_feature(storey_range, "storey_range",train_ds

#Numerical features
dist_to_nearest_stn_encoded = encode_numerical_feature(dist_to_nearest_stn, "dist_to_nea
dist_to_dhoby_encoded = encode_numerical_feature(dist_to_dhoby, "dist_to_dhoby", train_
degree_centrality_encoded = encode_numerical_feature(degree_centrality, "degree_central
eigenvector_centrality_encoded = encode_numerical_feature(eigenvector_centrality, "eige
remaining_lease_year_encoded = encode_numerical_feature(remaining_lease_years, "remainin
floor_area_sqm_encoded = encode_numerical_feature(floor_area_sqm,"floor_area_sqm", trai

all_features = layers.Concatenate()(
                [
                    month_encoded,
                    storey_range_encoded,
                    flat_model_type_encoded,
                    floor_area_sqm_encoded,
                    remaining_lease_year_encoded,
                    degree_centrality_encoded,
                    eigenvector_centrality_encoded,
                    dist_to_nearest_stn_encoded,
                    dist_to_dhoby_encoded
                ]
)
output = layers.Dense(1, activation="linear")(all_features)
adam_model = keras.Model(all_inputs, output)
adam_model.compile(optimizer="adam", loss= "mse",metrics=[r2])
```

ARCHITECTURE

In [ ]:

```
# `rankdir='LR'` is to make the graph horizontal.
keras.utils.plot_model(adam_model, show_shapes=True, rankdir="LR")
```

Out[ ]:



Question 1C

Training **Adam** model and **SGD** model

In [ ]:

```
history = {}
no_epochs = 50
batch_size = 256
history["adam_model"] = adam_model.fit(train_ds, epochs=no_epochs,batch_size=batch_size
```

```
Epoch 1/50
c:\Users\JoeTe\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\engin
e\functional.py:566: UserWarning: Input dict contained keys ['year'] which did not m
atch any model input. They will be ignored by the model.
  inputs = self._flatten_to_reference_inputs(inputs)
```

```
342/342 [==============================] - 3s 6ms/step - loss: 219585937408.0000 - r
2: -8.3474 - val_loss: 301486735360.0000 - val_r2: -10.0832
Epoch 2/50
342/342 [==============================] - 2s 5ms/step - loss: 219584561152.0000 - r
2: -8.3520 - val_loss: 301485064192.0000 - val_r2: -10.0722
Epoch 3/50
342/342 [==============================] - 2s 5ms/step - loss: 219583217664.0000 - r
2: -8.3501 - val_loss: 301483458560.0000 - val_r2: -10.1026
Epoch 4/50
342/342 [==============================] - 2s 6ms/step - loss: 219581857792.0000 - r
2: -8.3437 - val_loss: 301481951232.0000 - val_r2: -10.0800
Epoch 5/50
342/342 [==============================] - 2s 6ms/step - loss: 219580661760.0000 - r
2: -8.3458 - val_loss: 301480378368.0000 - val_r2: -10.1214
Epoch 6/50
342/342 [==============================] - 2s 5ms/step - loss: 219579383808.0000 - r
2: -8.3513 - val_loss: 301478903808.0000 - val_r2: -10.0838
Epoch 7/50
342/342 [==============================] - 2s 4ms/step - loss: 219578007552.0000 - r
2: -8.3379 - val_loss: 301477199872.0000 - val_r2: -10.0794
Epoch 8/50
342/342 [==============================] - 2s 5ms/step - loss: 219576696832.0000 - r
2: -8.3468 - val_loss: 301475495936.0000 - val_r2: -10.0875
Epoch 9/50
342/342 [==============================] - 2s 4ms/step - loss: 219575369728.0000 - r
2: -8.3361 - val_loss: 301473988608.0000 - val_r2: -10.0987
Epoch 10/50
342/342 [==============================] - 2s 4ms/step - loss: 219573944320.0000 - r
2: -8.3610 - val_loss: 301472579584.0000 - val_r2: -10.0676
Epoch 11/50
342/342 [==============================] - 2s 4ms/step - loss: 219572666368.0000 - r
2: -8.3384 - val_loss: 301470941184.0000 - val_r2: -10.0713
Epoch 12/50
342/342 [==============================] - 2s 4ms/step - loss: 219571273728.0000 - r
2: -8.3493 - val_loss: 301469401088.0000 - val_r2: -10.0838
Epoch 13/50
342/342 [==============================] - 2s 4ms/step - loss: 219570012160.0000 - r
2: -8.3537 - val_loss: 301467828224.0000 - val_r2: -10.0720
Epoch 14/50
342/342 [==============================] - 2s 4ms/step - loss: 219568635904.0000 - r
2: -8.3496 - val_loss: 301466255360.0000 - val_r2: -10.0802
Epoch 15/50
342/342 [==============================] - 2s 4ms/step - loss: 219567390720.0000 - r
2: -8.3436 - val_loss: 301464649728.0000 - val_r2: -10.0789
Epoch 16/50
342/342 [==============================] - 2s 4ms/step - loss: 219566030848.0000 - r
2: -8.3404 - val_loss: 301463076864.0000 - val_r2: -10.0780
Epoch 17/50
342/342 [==============================] - 2s 5ms/step - loss: 219564539904.0000 - r
2: -8.3417 - val_loss: 301461536768.0000 - val_r2: -10.0888
Epoch 18/50
342/342 [==============================] - 2s 6ms/step - loss: 219563360256.0000 - r
2: -8.3549 - val_loss: 301459800064.0000 - val_r2: -10.0647
Epoch 19/50
342/342 [==============================] - 2s 5ms/step - loss: 219562082304.0000 - r
2: -8.3613 - val_loss: 301458391040.0000 - val_r2: -10.0815
Epoch 20/50
342/342 [==============================] - 2s 5ms/step - loss: 219560624128.0000 - r
2: -8.3505 - val_loss: 301456883712.0000 - val_r2: -10.0901
Epoch 21/50
```

```
342/342 [==============================] - 2s 5ms/step - loss: 219559247872.0000 - r
2: -8.3440 - val_loss: 301455212544.0000 - val_r2: -10.0895
Epoch 22/50
342/342 [==============================] - 2s 5ms/step - loss: 219557904384.0000 - r
2: -8.3422 - val_loss: 301453672448.0000 - val_r2: -10.0798
Epoch 23/50
342/342 [==============================] - 2s 5ms/step - loss: 219556626432.0000 - r
2: -8.3484 - val_loss: 301452099584.0000 - val_r2: -10.0744
Epoch 24/50
342/342 [==============================] - 2s 5ms/step - loss: 219555266560.0000 - r
2: -8.3529 - val_loss: 301450493952.0000 - val_r2: -10.0714
Epoch 25/50
342/342 [==============================] - 2s 5ms/step - loss: 219554070528.0000 - r
2: -8.3670 - val_loss: 301448986624.0000 - val_r2: -10.0714
Epoch 26/50
342/342 [==============================] - 2s 5ms/step - loss: 219552563200.0000 - r
2: -8.3584 - val_loss: 301447348224.0000 - val_r2: -10.0604
Epoch 27/50
342/342 [==============================] - 2s 5ms/step - loss: 219551416320.0000 - r
2: -8.3555 - val_loss: 301445808128.0000 - val_r2: -10.0936
Epoch 28/50
342/342 [==============================] - 2s 5ms/step - loss: 219549941760.0000 - r
2: -8.3427 - val_loss: 301444268032.0000 - val_r2: -10.0704
Epoch 29/50
342/342 [==============================] - 2s 5ms/step - loss: 219548729344.0000 - r
2: -8.3483 - val_loss: 301442629632.0000 - val_r2: -10.0780
Epoch 30/50
342/342 [==============================] - 2s 5ms/step - loss: 219547402240.0000 - r
2: -8.3525 - val_loss: 301441056768.0000 - val_r2: -10.0887
Epoch 31/50
342/342 [==============================] - 2s 5ms/step - loss: 219545976832.0000 - r
2: -8.3495 - val_loss: 301439549440.0000 - val_r2: -10.0753
Epoch 32/50
342/342 [==============================] - 2s 5ms/step - loss: 219544698880.0000 - r
2: -8.3439 - val_loss: 301438009344.0000 - val_r2: -10.0697
Epoch 33/50
342/342 [==============================] - 2s 5ms/step - loss: 219543224320.0000 - r
2: -8.3394 - val_loss: 301436403712.0000 - val_r2: -10.0841
Epoch 34/50
342/342 [==============================] - 2s 5ms/step - loss: 219541995520.0000 - r
2: -8.3379 - val_loss: 301434830848.0000 - val_r2: -10.0817
Epoch 35/50
342/342 [==============================] - 2s 5ms/step - loss: 219540668416.0000 - r
2: -8.3572 - val_loss: 301433356288.0000 - val_r2: -10.0745
Epoch 36/50
342/342 [==============================] - 2s 6ms/step - loss: 219539308544.0000 - r
2: -8.3376 - val_loss: 301431586816.0000 - val_r2: -10.0944
Epoch 37/50
342/342 [==============================] - 2s 5ms/step - loss: 219538112512.0000 - r
2: -8.3422 - val_loss: 301430079488.0000 - val_r2: -10.1003
Epoch 38/50
342/342 [==============================] - 2s 5ms/step - loss: 219536654336.0000 - r
2: -8.3458 - val_loss: 301428506624.0000 - val_r2: -10.0598
Epoch 39/50
342/342 [==============================] - 2s 5ms/step - loss: 219535310848.0000 - r
2: -8.3552 - val_loss: 301426999296.0000 - val_r2: -10.0878
Epoch 40/50
342/342 [==============================] - 2s 4ms/step - loss: 219533983744.0000 - r
2: -8.3535 - val_loss: 301425426432.0000 - val_r2: -10.0813
Epoch 41/50
```

```
342/342 [==============================] - 2s 4ms/step - loss: 219532525568.0000 - r
2: -8.3537 - val_loss: 301423788032.0000 - val_r2: -10.0829
Epoch 42/50
342/342 [==============================] - 2s 4ms/step - loss: 219531280384.0000 - r
2: -8.3425 - val_loss: 301422182400.0000 - val_r2: -10.0835
Epoch 43/50
342/342 [==============================] - 2s 5ms/step - loss: 219529920512.0000 - r
2: -8.3514 - val_loss: 301420707840.0000 - val_r2: -10.0769
Epoch 44/50
342/342 [==============================] - 2s 4ms/step - loss: 219528691712.0000 - r
2: -8.3413 - val_loss: 301419069440.0000 - val_r2: -10.0676
Epoch 45/50
342/342 [==============================] - 2s 4ms/step - loss: 219527348224.0000 - r
2: -8.3392 - val_loss: 301417463808.0000 - val_r2: -10.0957
Epoch 46/50
342/342 [==============================] - 2s 4ms/step - loss: 219526037504.0000 - r
2: -8.3425 - val_loss: 301415956480.0000 - val_r2: -10.0749
Epoch 47/50
342/342 [==============================] - 2s 4ms/step - loss: 219524759552.0000 - r
2: -8.3485 - val_loss: 301414416384.0000 - val_r2: -10.0613
Epoch 48/50
342/342 [==============================] - 2s 4ms/step - loss: 219523317760.0000 - r
2: -8.3431 - val_loss: 301412810752.0000 - val_r2: -10.0856
Epoch 49/50
342/342 [==============================] - 2s 4ms/step - loss: 219522039808.0000 - r
2: -8.3478 - val_loss: 301411434496.0000 - val_r2: -10.0833
Epoch 50/50
342/342 [==============================] - 2s 5ms/step - loss: 219520679936.0000 - r
2: -8.3501 - val_loss: 301409632256.0000 - val_r2: -10.0739
```

Training of **SGD model** with learning rate **0.01**

In [ ]:

```
custom_optimizer=tf.keras.optimizers.SGD(learning_rate=0.01)
sgd_model = keras.Model(all_inputs, output)
sgd_model.compile(optimizer=custom_optimizer, loss="mse", metrics=[r2])
history["sgd_model"] = sgd_model.fit(train_ds, epochs=no_epochs,batch_size=batch_size, v
```

```
Epoch 1/50
342/342 [==============================] - 3s 6ms/step - loss: 18724683776.0000 - r2:
0.1967 - val_loss: 13789808640.0000 - val_r2: 0.4959
Epoch 2/50
342/342 [==============================] - 2s 5ms/step - loss: 5481839616.0000 - r2: 0.
7678 - val_loss: 12796226560.0000 - val_r2: 0.5320
Epoch 3/50
342/342 [==============================] - 2s 5ms/step - loss: 5026614272.0000 - r2: 0.
7875 - val_loss: 12325055488.0000 - val_r2: 0.5501
Epoch 4/50
342/342 [==============================] - 2s 5ms/step - loss: 4776948224.0000 - r2: 0.
7976 - val_loss: 12102066176.0000 - val_r2: 0.5574
Epoch 5/50
342/342 [==============================] - 2s 4ms/step - loss: 4609028096.0000 - r2: 0.
8049 - val_loss: 11926619136.0000 - val_r2: 0.5641
Epoch 6/50
342/342 [==============================] - 2s 4ms/step - loss: 4482668544.0000 - r2: 0.
8106 - val_loss: 11662412800.0000 - val_r2: 0.5736
Epoch 7/50
342/342 [==============================] - 2s 4ms/step - loss: 4384758272.0000 - r2: 0.
8140 - val_loss: 11666586624.0000 - val_r2: 0.5743
Epoch 8/50
342/342 [==============================] - 2s 4ms/step - loss: 4306899968.0000 - r2: 0.
8173 - val_loss: 11521402880.0000 - val_r2: 0.5791
Epoch 9/50
342/342 [==============================] - 2s 4ms/step - loss: 4242382848.0000 - r2: 0.
8203 - val_loss: 11452370944.0000 - val_r2: 0.5809
Epoch 10/50
342/342 [==============================] - 2s 4ms/step - loss: 4189079808.0000 - r2: 0.
8224 - val_loss: 11380891648.0000 - val_r2: 0.5841
Epoch 11/50
342/342 [==============================] - 2s 4ms/step - loss: 4144402176.0000 - r2: 0.
8241 - val_loss: 11411303424.0000 - val_r2: 0.5829
Epoch 12/50
342/342 [==============================] - 2s 4ms/step - loss: 4106333952.0000 - r2: 0.
8260 - val_loss: 11307262976.0000 - val_r2: 0.5865
Epoch 13/50
342/342 [==============================] - 2s 4ms/step - loss: 4072907776.0000 - r2: 0.
8270 - val_loss: 11243720704.0000 - val_r2: 0.5889
Epoch 14/50
342/342 [==============================] - 2s 4ms/step - loss: 4044713216.0000 - r2: 0.
8284 - val_loss: 11245513728.0000 - val_r2: 0.5885
Epoch 15/50
342/342 [==============================] - 2s 4ms/step - loss: 4019300608.0000 - r2: 0.
8295 - val_loss: 11111006208.0000 - val_r2: 0.5937
Epoch 16/50
342/342 [==============================] - 2s 4ms/step - loss: 3997265664.0000 - r2: 0.
8303 - val_loss: 11106009088.0000 - val_r2: 0.5936
Epoch 17/50
342/342 [==============================] - 2s 4ms/step - loss: 3977696768.0000 - r2: 0.
8314 - val_loss: 11057296384.0000 - val_r2: 0.5950
Epoch 18/50
342/342 [==============================] - 2s 4ms/step - loss: 3959637248.0000 - r2: 0.
8321 - val_loss: 11035296768.0000 - val_r2: 0.5958
Epoch 19/50
342/342 [==============================] - 2s 4ms/step - loss: 3944157440.0000 - r2: 0.
8327 - val_loss: 11040465920.0000 - val_r2: 0.5960
Epoch 20/50
342/342 [==============================] - 2s 4ms/step - loss: 3929256960.0000 - r2: 0.
8336 - val_loss: 10964644864.0000 - val_r2: 0.5986
```

```
Epoch 21/50
342/342 [==============================] - 2s 4ms/step - loss: 3916516864.0000 - r2: 0.
8336 - val_loss: 10949549056.0000 - val_r2: 0.5995
Epoch 22/50
342/342 [==============================] - 2s 4ms/step - loss: 3904264960.0000 - r2: 0.
8342 - val_loss: 10884148224.0000 - val_r2: 0.6011
Epoch 23/50
342/342 [==============================] - 2s 4ms/step - loss: 3894416896.0000 - r2: 0.
8350 - val_loss: 10865339392.0000 - val_r2: 0.6020
Epoch 24/50
342/342 [==============================] - 2s 4ms/step - loss: 3883611648.0000 - r2: 0.
8350 - val_loss: 10960077824.0000 - val_r2: 0.5986
Epoch 25/50
342/342 [==============================] - 2s 4ms/step - loss: 3873955328.0000 - r2: 0.
8357 - val_loss: 11016645632.0000 - val_r2: 0.5963
Epoch 26/50
342/342 [==============================] - 2s 4ms/step - loss: 3866191360.0000 - r2: 0.
8359 - val_loss: 10871563264.0000 - val_r2: 0.6018
Epoch 27/50
342/342 [==============================] - 2s 4ms/step - loss: 3858726912.0000 - r2: 0.
8366 - val_loss: 10842923008.0000 - val_r2: 0.6028
Epoch 28/50
342/342 [==============================] - 2s 4ms/step - loss: 3850375424.0000 - r2: 0.
8367 - val_loss: 10923956224.0000 - val_r2: 0.6004
Epoch 29/50
342/342 [==============================] - 2s 4ms/step - loss: 3843901184.0000 - r2: 0.
8370 - val_loss: 10925705216.0000 - val_r2: 0.6008
Epoch 30/50
342/342 [==============================] - 2s 4ms/step - loss: 3837209088.0000 - r2: 0.
8373 - val_loss: 10895963136.0000 - val_r2: 0.6011
Epoch 31/50
342/342 [==============================] - 2s 4ms/step - loss: 3831445248.0000 - r2: 0.
8371 - val_loss: 10731506688.0000 - val_r2: 0.6072
Epoch 32/50
342/342 [==============================] - 2s 4ms/step - loss: 3825593600.0000 - r2: 0.
8378 - val_loss: 10807692288.0000 - val_r2: 0.6042
Epoch 33/50
342/342 [==============================] - 2s 4ms/step - loss: 3819265792.0000 - r2: 0.
8374 - val_loss: 10939134976.0000 - val_r2: 0.5987
Epoch 34/50
342/342 [==============================] - 2s 4ms/step - loss: 3814961920.0000 - r2: 0.
8382 - val_loss: 10770751488.0000 - val_r2: 0.6057
Epoch 35/50
342/342 [==============================] - 2s 4ms/step - loss: 3809940736.0000 - r2: 0.
8381 - val_loss: 10925270016.0000 - val_r2: 0.5994
Epoch 36/50
342/342 [==============================] - 2s 4ms/step - loss: 3805572608.0000 - r2: 0.
8384 - val_loss: 10799664128.0000 - val_r2: 0.6041
Epoch 37/50
342/342 [==============================] - 2s 4ms/step - loss: 3800988928.0000 - r2: 0.
8386 - val_loss: 10752359424.0000 - val_r2: 0.6066
Epoch 38/50
342/342 [==============================] - 2s 4ms/step - loss: 3796920320.0000 - r2: 0.
8386 - val_loss: 10769947648.0000 - val_r2: 0.6054
Epoch 39/50
342/342 [==============================] - 2s 4ms/step - loss: 3792671744.0000 - r2: 0.
8390 - val_loss: 10778511360.0000 - val_r2: 0.6055
Epoch 40/50
342/342 [==============================] - 2s 4ms/step - loss: 3789245952.0000 - r2: 0.
8391 - val_loss: 10768480256.0000 - val_r2: 0.6048
```

```
Epoch 41/50
342/342 [==============================] - 2s 4ms/step - loss: 3785730048.0000 - r2: 0.
8395 - val_loss: 10746905600.0000 - val_r2: 0.6065
Epoch 42/50
342/342 [==============================] - 2s 4ms/step - loss: 3782271232.0000 - r2: 0.
8395 - val_loss: 10701347840.0000 - val_r2: 0.6086
Epoch 43/50
342/342 [==============================] - 2s 4ms/step - loss: 3778778624.0000 - r2: 0.
8395 - val_loss: 10712100864.0000 - val_r2: 0.6074
Epoch 44/50
342/342 [==============================] - 2s 4ms/step - loss: 3775596032.0000 - r2: 0.
8397 - val_loss: 10774897664.0000 - val_r2: 0.6047
Epoch 45/50
342/342 [==============================] - 2s 4ms/step - loss: 3772517120.0000 - r2: 0.
8399 - val_loss: 10763461632.0000 - val_r2: 0.6059
Epoch 46/50
342/342 [==============================] - 2s 4ms/step - loss: 3769706240.0000 - r2: 0.
8399 - val_loss: 10645620736.0000 - val_r2: 0.6105
Epoch 47/50
342/342 [==============================] - 2s 4ms/step - loss: 3767052032.0000 - r2: 0.
8400 - val_loss: 10574597120.0000 - val_r2: 0.6126
Epoch 48/50
342/342 [==============================] - 2s 4ms/step - loss: 3764005632.0000 - r2: 0.
8401 - val_loss: 10654348288.0000 - val_r2: 0.6100
Epoch 49/50
342/342 [==============================] - 2s 4ms/step - loss: 3761464064.0000 - r2: 0.
8403 - val_loss: 10667359232.0000 - val_r2: 0.6090
Epoch 50/50
342/342 [==============================] - 2s 4ms/step - loss: 3758382592.0000 - r2: 0.
8403 - val_loss: 10645596160.0000 - val_r2: 0.6101
```

In [ ]:

```
data = {"Q1C_Adam_Train R^2": [history["adam_model"].history["r2"][-1]],
        "Q1C_Adam_Val R^2": [history["adam_model"].history["val_r2"][-1]],
        "SGD_Train R^2": [history["sgd_model"].history["r2"][-1]],
        "SGD_Val R^2": [history["sgd_model"].history["val_r2"][-1]]
       }
data_df = pd.DataFrame.from_dict(data)
data_df
```

Out[ ]:

	Q1C_Adam_Train R^2	Q1C_Adam_Val R^2	SGD_Train R^2	SGD_Val R^2
0	-8.350127	-10.073879	0.840261	0.610094

Report why the change to SGD fixes the problem faced when using Adam optimiser

From the table above, we can see the difference in R^2 for both Adam and SGD optimiser.

Adam algorithm is leverages on the power of adaptive learning rates methods to find individual learning rates for each parameter. Due to the low learning rate of the Adam optimiser, it is very likely that it does not have sufficient iteration time to converge to minima and yield a decent R^2 value

SGD algorithm with a learning rate of 0.01 allows the model to generalize faster and converge faster to minima

Question 1D

Training of **Adam** model with learning rate of **0.08**

In [ ]:
```
adam_optimizer = tf.keras.optimizers.Adam(learning_rate=0.08)
hidden_layer = layers.Dense(10, activation ="linear")(all_features)
Q1D_output = layers.Dense(1, activation="linear")(hidden_layer)
Q1D_adam_model = keras.Model(all_inputs, Q1D_output)
Q1D_adam_model.compile(optimizer=adam_optimizer, loss= "mse",metrics=[r2])
```

In [ ]:
```
history["Q1D_adam_model"] = Q1D_adam_model.fit(train_ds, epochs=no_epochs,batch_size=ba
```

```
Epoch 1/50
```

```
c:\Users\JoeTe\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\engine\f
unctional.py:566: UserWarning: Input dict contained keys ['year'] which did not match a
ny model input. They will be ignored by the model.
  inputs = self._flatten_to_reference_inputs(inputs)
```

```
342/342 [==============================] - 2s 5ms/step - loss: 198604013568.0000 - r2: 
-7.4464 - val_loss: 226039087104.0000 - val_r2: -7.3163
Epoch 2/50
342/342 [==============================] - 2s 4ms/step - loss: 99525992448.0000 - r2: -
3.2406 - val_loss: 88990212096.0000 - val_r2: -2.2731
Epoch 3/50
342/342 [==============================] - 2s 4ms/step - loss: 23373901824.0000 - r2: 
0.0066 - val_loss: 25650937856.0000 - val_r2: 0.0618
Epoch 4/50
342/342 [==============================] - 2s 4ms/step - loss: 6420192256.0000 - r2: 0.
7287 - val_loss: 14056211456.0000 - val_r2: 0.4869
Epoch 5/50
342/342 [==============================] - 2s 4ms/step - loss: 5106189312.0000 - r2: 0.
7841 - val_loss: 12450711552.0000 - val_r2: 0.5452
Epoch 6/50
342/342 [==============================] - 2s 4ms/step - loss: 4765254144.0000 - r2: 0.
7983 - val_loss: 11993393152.0000 - val_r2: 0.5622
Epoch 7/50
342/342 [==============================] - 2s 4ms/step - loss: 4506056704.0000 - r2: 0.
8093 - val_loss: 11758059520.0000 - val_r2: 0.5708
Epoch 8/50
342/342 [==============================] - 2s 4ms/step - loss: 4306036224.0000 - r2: 0.
8175 - val_loss: 11429959680.0000 - val_r2: 0.5820
Epoch 9/50
342/342 [==============================] - 2s 4ms/step - loss: 4153517056.0000 - r2: 0.
8239 - val_loss: 11282122752.0000 - val_r2: 0.5877
Epoch 10/50
342/342 [==============================] - 2s 4ms/step - loss: 4039020288.0000 - r2: 0.
8285 - val_loss: 10980324352.0000 - val_r2: 0.5986
Epoch 11/50
342/342 [==============================] - 2s 4ms/step - loss: 3953182976.0000 - r2: 0.
8323 - val_loss: 11008365568.0000 - val_r2: 0.5972
Epoch 12/50
342/342 [==============================] - 2s 4ms/step - loss: 3890199040.0000 - r2: 0.
8349 - val_loss: 10823264256.0000 - val_r2: 0.6043
Epoch 13/50
342/342 [==============================] - 2s 4ms/step - loss: 3842161408.0000 - r2: 0.
8369 - val_loss: 10885229568.0000 - val_r2: 0.6007
Epoch 14/50
342/342 [==============================] - 2s 4ms/step - loss: 3806295808.0000 - r2: 0.
8381 - val_loss: 10814405632.0000 - val_r2: 0.6041
Epoch 15/50
342/342 [==============================] - 2s 4ms/step - loss: 3778650368.0000 - r2: 0.
8397 - val_loss: 10797533184.0000 - val_r2: 0.6039
Epoch 16/50
342/342 [==============================] - 2s 4ms/step - loss: 3757196544.0000 - r2: 0.
8405 - val_loss: 10597463040.0000 - val_r2: 0.6117
Epoch 17/50
342/342 [==============================] - 2s 4ms/step - loss: 3739115776.0000 - r2: 0.
8413 - val_loss: 10597354496.0000 - val_r2: 0.6124
Epoch 18/50
342/342 [==============================] - 2s 4ms/step - loss: 3725249024.0000 - r2: 0.
8420 - val_loss: 10437833728.0000 - val_r2: 0.6180
Epoch 19/50
342/342 [==============================] - 2s 4ms/step - loss: 3712825344.0000 - r2: 0.
8420 - val_loss: 10625016832.0000 - val_r2: 0.6109
Epoch 20/50
342/342 [==============================] - 2s 4ms/step - loss: 3702594560.0000 - r2: 0.
8427 - val_loss: 10349215744.0000 - val_r2: 0.6207
Epoch 21/50
```

```
342/342 [==============================] - 2s 4ms/step - loss: 3693874688.0000 - r2: 0.
8427 - val_loss: 10533180416.0000 - val_r2: 0.6137
Epoch 22/50
342/342 [==============================] - 2s 4ms/step - loss: 3686298624.0000 - r2: 0.
8432 - val_loss: 10248724480.0000 - val_r2: 0.6246
Epoch 23/50
342/342 [==============================] - 2s 4ms/step - loss: 3681099776.0000 - r2: 0.
8437 - val_loss: 10425411584.0000 - val_r2: 0.6182
Epoch 24/50
342/342 [==============================] - 2s 4ms/step - loss: 3674409984.0000 - r2: 0.
8440 - val_loss: 10288659456.0000 - val_r2: 0.6233
Epoch 25/50
342/342 [==============================] - 2s 4ms/step - loss: 3670163200.0000 - r2: 0.
8440 - val_loss: 10196599808.0000 - val_r2: 0.6263
Epoch 26/50
342/342 [==============================] - 2s 4ms/step - loss: 3665843712.0000 - r2: 0.
8443 - val_loss: 10562157568.0000 - val_r2: 0.6126
Epoch 27/50
342/342 [==============================] - 2s 4ms/step - loss: 3662608640.0000 - r2: 0.
8444 - val_loss: 10359180288.0000 - val_r2: 0.6204
Epoch 28/50
342/342 [==============================] - 2s 4ms/step - loss: 3659944704.0000 - r2: 0.
8444 - val_loss: 10430134272.0000 - val_r2: 0.6171
Epoch 29/50
342/342 [==============================] - 2s 4ms/step - loss: 3656346624.0000 - r2: 0.
8445 - val_loss: 10427124736.0000 - val_r2: 0.6173
Epoch 30/50
342/342 [==============================] - 2s 4ms/step - loss: 3653032192.0000 - r2: 0.
8450 - val_loss: 10528819200.0000 - val_r2: 0.6138
Epoch 31/50
342/342 [==============================] - 2s 4ms/step - loss: 3650757120.0000 - r2: 0.
8449 - val_loss: 10276255744.0000 - val_r2: 0.6234
Epoch 32/50
342/342 [==============================] - 2s 4ms/step - loss: 3649676544.0000 - r2: 0.
8449 - val_loss: 10162065408.0000 - val_r2: 0.6267
Epoch 33/50
342/342 [==============================] - 2s 4ms/step - loss: 3647041536.0000 - r2: 0.
8450 - val_loss: 10488784896.0000 - val_r2: 0.6149
Epoch 34/50
342/342 [==============================] - 2s 4ms/step - loss: 3645616640.0000 - r2: 0.
8450 - val_loss: 10336129024.0000 - val_r2: 0.6210
Epoch 35/50
342/342 [==============================] - 2s 4ms/step - loss: 3644200960.0000 - r2: 0.
8451 - val_loss: 10387428352.0000 - val_r2: 0.6184
Epoch 36/50
342/342 [==============================] - 2s 4ms/step - loss: 3644126208.0000 - r2: 0.
8452 - val_loss: 10216840192.0000 - val_r2: 0.6253
Epoch 37/50
342/342 [==============================] - 2s 4ms/step - loss: 3641288704.0000 - r2: 0.
8453 - val_loss: 10491762688.0000 - val_r2: 0.6150
Epoch 38/50
342/342 [==============================] - 2s 4ms/step - loss: 3641036544.0000 - r2: 0.
8456 - val_loss: 10530390016.0000 - val_r2: 0.6128
Epoch 39/50
342/342 [==============================] - 2s 4ms/step - loss: 3639185408.0000 - r2: 0.
8453 - val_loss: 10134671360.0000 - val_r2: 0.6284
Epoch 40/50
342/342 [==============================] - 2s 4ms/step - loss: 3638354944.0000 - r2: 0.
8457 - val_loss: 10211177472.0000 - val_r2: 0.6253
Epoch 41/50
```

```
342/342 [==============================] - 2s 4ms/step - loss: 3637105664.0000 - r2: 0.
8454 - val_loss: 10453860352.0000 - val_r2: 0.6160
Epoch 42/50
342/342 [==============================] - 2s 4ms/step - loss: 3637018624.0000 - r2: 0.
8452 - val_loss: 10295630848.0000 - val_r2: 0.6223
Epoch 43/50
342/342 [==============================] - 2s 5ms/step - loss: 3636144384.0000 - r2: 0.
8456 - val_loss: 10414814208.0000 - val_r2: 0.6184
Epoch 44/50
342/342 [==============================] - 2s 5ms/step - loss: 3635691520.0000 - r2: 0.
8457 - val_loss: 10462346240.0000 - val_r2: 0.6158
Epoch 45/50
342/342 [==============================] - 2s 4ms/step - loss: 3634302976.0000 - r2: 0.
8454 - val_loss: 10394300416.0000 - val_r2: 0.6190
Epoch 46/50
342/342 [==============================] - 2s 4ms/step - loss: 3634126336.0000 - r2: 0.
8453 - val_loss: 10220095488.0000 - val_r2: 0.6244
Epoch 47/50
342/342 [==============================] - 2s 4ms/step - loss: 3634429696.0000 - r2: 0.
8458 - val_loss: 10390547456.0000 - val_r2: 0.6195
Epoch 48/50
342/342 [==============================] - 2s 4ms/step - loss: 3633514752.0000 - r2: 0.
8456 - val_loss: 10355986432.0000 - val_r2: 0.6202
Epoch 49/50
342/342 [==============================] - 2s 4ms/step - loss: 3633320960.0000 - r2: 0.
8457 - val_loss: 10278689792.0000 - val_r2: 0.6235
Epoch 50/50
342/342 [==============================] - 2s 4ms/step - loss: 3632434176.0000 - r2: 0.
8457 - val_loss: 10125669376.0000 - val_r2: 0.6287
```

Question 1E

In [ ]:

```
# Compare with a table and explain
Q1_data = {"Q1D_Adam_Train R^2": [history["Q1D_adam_model"].history["r2"][-1]],
        "Q1D_Adam_Val R^2": [history["Q1D_adam_model"].history["val_r2"][-1]],
        "Q1C_Adam_Train R^2": [history["adam_model"].history["r2"][-1]],
        "Q1C_Adam_Val R^2": [history["adam_model"].history["val_r2"][-1]],
        "SGD_Train R^2": [history["sgd_model"].history["r2"][-1]],
        "SGD_Val R^2": [history["sgd_model"].history["val_r2"][-1]]
        }


data_df = pd.DataFrame.from_dict(Q1_data)
data_df
```

Out[ ]:

	Q1D_Adam_Train R^2	Q1D_Adam_Val R^2	Q1C_Adam_Train R^2	Q1C_Adam_Val R^2	SGD_Train R^2	SGD_Val R^2
0	0.845678	0.628727	-8.350127	-10.073879	0.840261	0.610094

Compare the performance of 1C and 1D to the linear regression model and suggest reasons for the observations

From the table, the validation R^2 value from Q1D(0.628727) is higher than the linear regression model (0.627) and the validation R^2 value from Q1C(SGD 0.610094) is lower than the linear regression model(0.627).

In Q1C, the SGD model do not have the complexity to discover and learn the relationships between the input features and thus leading to a poorer result compared to the linear regression model.

In 1D, there is an additional hidden layer and a rise in learning rate for the Adam optimiser.Thus, performance of the network model will be better as with the hidden layer, the network model is able to capture more complexity and discover relationships between the features in the input.

Also, with the higher learning rate, the modified adam optimiser will converge faster.