AI Project - Melbourne Housing Dataset

Yahya Nashat Jad 0222840

Ward halim muhsen 0222057

The data can be accesed from: https://www.kaggle.com/datasets/dansbecker/melbourne-housing-snapshot

This project analyzes Melbourne's real estate market using machine learning to predict housing prices based on property features. The dataset contains 13,581 instance with 21 attributes including location, land size, room counts, and property type.

Acknowledgements

This is intended as a static (unchanging) snapshot of https://www.kaggle.com/anthonypino/melbourne-housing-market. It was created in September 2017. Additionally, homes with no Price have been removed.

Data Visualization

In [67]:
```
import pandas as pd
import sklearn as sk
import numpy as np
import matplotlib.pyplot as plt
```

In [68]:
```
data=pd.read_csv('melb_data.csv')
```

In [69]:
```
#visualizing columns and their types and number of instances
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13580 entries, 0 to 13579
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Suburb         13580 non-null  object 
 1   Address        13580 non-null  object 
 2   Rooms          13580 non-null  int64  
 3   Type           13580 non-null  object 
 4   Price          13580 non-null  float64
 5   Method         13580 non-null  object 
 6   SellerG        13580 non-null  object 
 7   Date           13580 non-null  object 
 8   Distance       13580 non-null  float64
 9   Postcode       13580 non-null  float64
 10  Bedroom2       13580 non-null  float64
 11  Bathroom       13580 non-null  float64
 12  Car            13518 non-null  float64
 13  Landsize       13580 non-null  float64
 14  BuildingArea   7130 non-null   float64
 15  YearBuilt      8205 non-null   float64
 16  CouncilArea    12211 non-null  object 
 17  Lattitude      13580 non-null  float64
 18  Longtitude     13580 non-null  float64
 19  Regionname     13580 non-null  object 
 20  Propertycount  13580 non-null  float64
dtypes: float64(12), int64(1), object(8)
memory usage: 2.2+ MB
None
```

In [70]:

```python
# data numeric measures
data.describe()
```

Out[70]:

	Rooms	Price	Distance	Postcode	Bedroom2	Bathroom
count	13580.000000	1.358000e+04	13580.000000	13580.000000	13580.000000	13580.000000
mean	2.937997	1.075684e+06	10.137776	3105.301915	2.914728	1.534242
std	0.955748	6.393107e+05	5.868725	90.676964	0.965921	0.691712
min	1.000000	8.500000e+04	0.000000	3000.000000	0.000000	0.000000
25%	2.000000	6.500000e+05	6.100000	3044.000000	2.000000	1.000000
50%	3.000000	9.030000e+05	9.200000	3084.000000	3.000000	1.000000
75%	3.000000	1.330000e+06	13.000000	3148.000000	3.000000	2.000000
max	10.000000	9.000000e+06	48.100000	3977.000000	20.000000	8.000000

In [71]:

```python
# categorical data measures
data.describe(include=[np.object_])
```

Out[71]:

	Suburb	Address	Type	Method	SellerG	Date	CouncilArea	Regionname
count	13580	13580	13580	13580	13580	13580	12211	13580
unique	314	13378	3	5	268	58	33	8
top	Reservoir	5 Charles St	h	S	Nelson	27/05/2017	Moreland	Southern Metropolitan
freq	359	3	9449	9022	1565	473	1163	4695

In [72]:

```
# check for missing data
data.isnull().sum()
```

Out[72]:

```
Suburb             0
Address            0
Rooms              0
Type               0
Price              0
Method             0
SellerG            0
Date               0
Distance           0
Postcode           0
Bedroom2           0
Bathroom           0
Car               62
Landsize           0
BuildingArea    6450
YearBuilt       5375
CouncilArea     1369
Lattitude          0
Longtitude         0
Regionname         0
Propertycount      0
dtype: int64
```

BuildingArea and YearBuilt have too many missing values therefore they will be dropped

In [74]:

```
data.drop(['BuildingArea','YearBuilt'], axis=1,inplace = True )
```

In [75]:

```
import warnings
# Ignore all warnings
warnings.filterwarnings('ignore')
print(data['CouncilArea'].value_counts().head())
print("\n\n---------------------------------------------\n"
      , data['Car'].value_counts().head())
data['CouncilArea'].fillna('Moreland' , inplace=True)
data['Car'].fillna(2.0 , inplace=True)

data.isnull().sum()
```

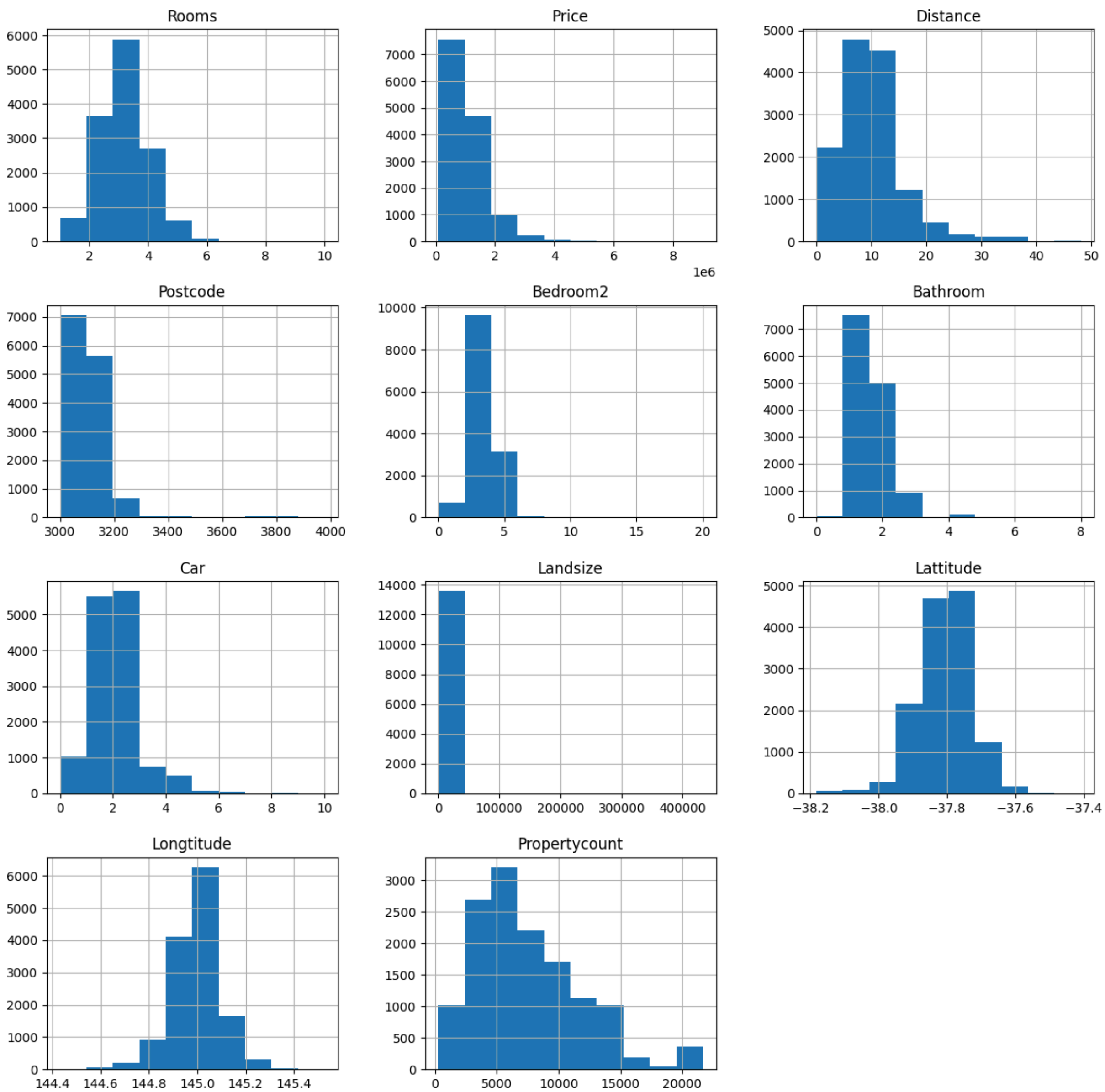
```
CouncilArea
Moreland         1163
Boroondara       1160
Moonee Valley     997
Darebin           934
Glen Eira         848
Name: count, dtype: int64


--------------------------------------------
 Car
2.0    5591
1.0    5509
0.0    1026
3.0     748
4.0     506
Name: count, dtype: int64
```

Out[75]:

```
Suburb           0
Address          0
Rooms            0
Type             0
Price            0
Method           0
SellerG          0
Date             0
Distance         0
Postcode         0
Bedroom2         0
Bathroom         0
Car              0
Landsize         0
CouncilArea      0
Lattitude        0
Longtitude       0
Regionname       0
Propertycount    0
dtype: int64
```

In [76]:

```
#looking at the features shape to see how the data is distributed in each feature
fig=data.hist(figsize=(15,15))
```



Looking For Correlations

In [78]:
```
numeric_data = data.select_dtypes(include=['number'])
corr_matrix = data.select_dtypes(include=['number']).corr()
corr_matrix['Price'].sort_values(ascending=False)
```

```
Out[78]: Price            1.000000
Rooms            0.496634
Bedroom2         0.475951
Bathroom         0.467038
Car              0.239109
Longtitude       0.203656
Postcode         0.107867
Landsize         0.037507
Propertycount   -0.042153
Distance        -0.162522
Lattitude       -0.212934
Name: Price, dtype: float64
```

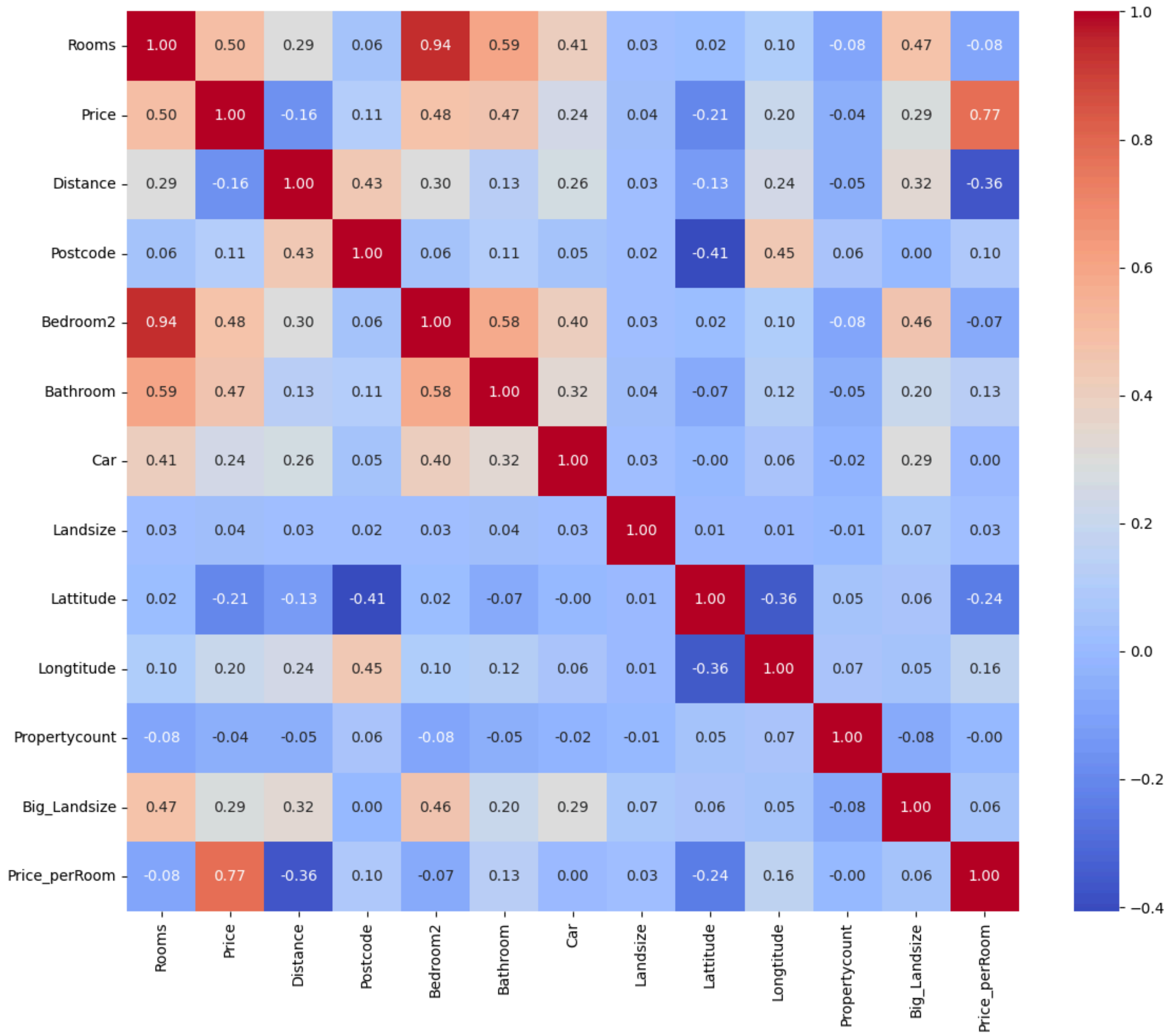
Rooms and Bedroom2 and Bathroom and Propertycount have good correlation with the price but we will look for better

```
In [80]: numeric_data['Big_Landsize'] = (numeric_data['Landsize'] > 150 ).astype(int)
numeric_data['Price_perRoom'] = (numeric_data['Price'] / numeric_data['Rooms']).whe
data['Big_Landsize'] = (data['Landsize'] > 150).astype(int)
data['Price_perRoom'] = (data['Price'] / data['Rooms']).where(data['Rooms'] > 0, da
corr_matrix = numeric_data.corr()
corr_matrix['Price'].sort_values(ascending=False)
```

```
Out[80]: Price            1.000000
Price_perRoom    0.772426
Rooms            0.496634
Bedroom2         0.475951
Bathroom         0.467038
Big_Landsize     0.291414
Car              0.239109
Longtitude       0.203656
Postcode         0.107867
Landsize         0.037507
Propertycount   -0.042153
Distance        -0.162522
Lattitude       -0.212934
Name: Price, dtype: float64
```

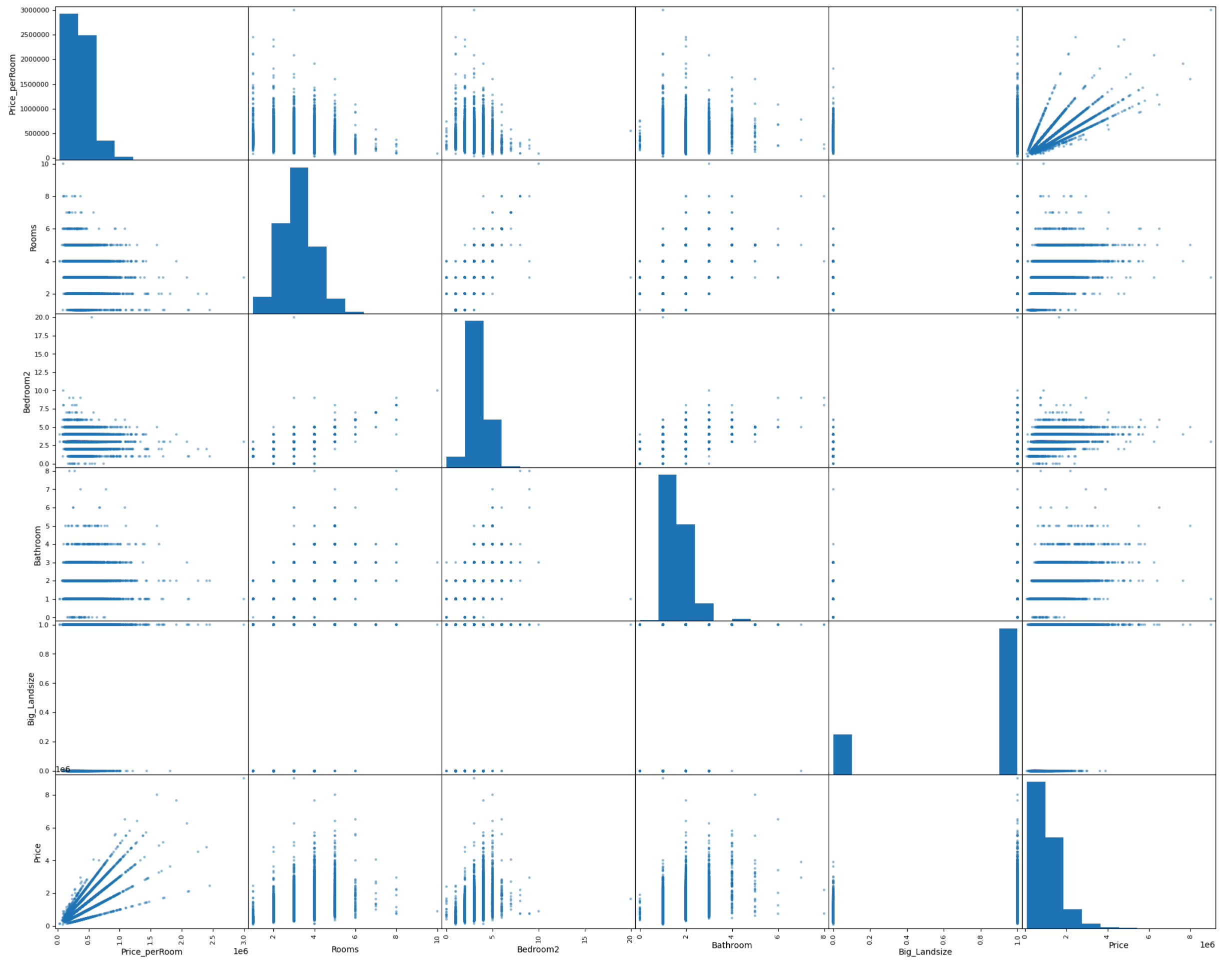
As we can see from the above correlation values, we found a high correlation between the new features and the 'Price' feature specially the 'Price_perRoom' feature.

```
In [82]: import seaborn as sns

plt.figure(figsize=(12, 10))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm')
plt.tight_layout()
plt.show()
```



In [83]:

```
#this is a plot to see how the features correlate
from pandas.plotting import scatter_matrix

features = ["Price_perRoom", "Rooms", "Bedroom2", "Bathroom","Big_Landsize" ,"Price
scatter_matrix(data[features], figsize=(25, 20))
plt.show()
```



In [84]:
```python
X = data.drop("Price",axis=1)
y = data['Price'].copy()
```

Full pipeline

In [86]:
```python
num_attribs= list(numeric_data.drop("Price",axis=1).columns)
cat_attribs= list(data.select_dtypes(include=[np.object_]).columns)
```

In [87]:
```python
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer

full_pipeline = ColumnTransformer([
    ('Numerical'         ,StandardScaler()       , num_attribs),
    ('Categorical'            ,OrdinalEncoder()     , cat_attribs),
])
```

In [88]:
```python
from sklearn.model_selection import train_test_split
#preparing the data to be used in training the models

data_prepared = full_pipeline.fit_transform(X)
X_trainfull, X_test, y_trainfull, y_test = train_test_split(data_prepared, y, test_
```

Training models using sklearn to compare them later with the Neural networks model using Keras

In [90]:
```
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor
from sklearn.metrics import *
import warnings
# Ignore all warnings
warnings.filterwarnings('ignore')

model_names = [
    "K Nearest Neighbors",
    "Random Forest",
    "SVR",
    "Linear regression",
    "Decision Tree",
    "Ada boost Decision tree",
]

regressers = [
    KNeighborsRegressor(p=2 , weights='distance',n_neighbors=5),
    RandomForestRegressor(n_jobs=-1, random_state=42,n_estimators=200,max_depth=15,
    SVR(kernel='rbf',C=100,epsilon=0.1,gamma='scale'),
    LinearRegression(n_jobs=-1),
    DecisionTreeRegressor(max_depth=10, min_samples_split=4,min_samples_leaf=2,rand
    AdaBoostRegressor(DecisionTreeRegressor(max_depth=4,min_samples_leaf=3,random_s
]
result = ""
dict = {}
for model,name in zip(regressers,model_names):
    model.fit(X_trainfull , y_trainfull)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
#r2 score is a performance metric that is close of the accuracy, the more you get c
    r2 = r2_score(y_test, y_pred)
    dict[name] = {'rmse' : rmse , 'r2_score' : r2}

    result += f"{name}:\n"
    result += f"  RMSE: {rmse:.4f}\n"
    result += f"  R2 Score: {r2:.4f}\n\n"

print(result)
```

```
K Nearest Neighbors:
  RMSE: 616867.0732
  R2 Score: 0.0420

Random Forest:
  RMSE: 39461.3874
  R2 Score: 0.9961

SVR:
  RMSE: 653063.0216
  R2 Score: -0.0737

Linear regression:
  RMSE: 181885.5908
  R2 Score: 0.9167

Decision Tree:
  RMSE: 37638.8108
  R2 Score: 0.9964

Ada boost Decision tree:
  RMSE: 153431.4651
  R2 Score: 0.9407
```

This loop trains each model in the regressers list and shows their RMSE and R2 Score, and from what we see above, the best RMSE we got is 37638.8 from the Decision Tree Regresser which is excelent for the price ranges we have and a R2 Score of 0.9964 which is also considered very good

Neural Network using Keras

In [93]:

```
import tensorflow as tf
from tensorflow.keras.callbacks import EarlyStopping

X_train, X_valid, y_train, y_valid = train_test_split(X_trainfull, y_trainfull, tes

norm_layer = tf.keras.layers.Normalization(input_shape=X_train.shape[1:])
neural = tf.keras.Sequential([
    norm_layer,
    tf.keras.layers.Dense(40, activation="relu"),
    tf.keras.layers.Dense(40, activation="relu"),
    tf.keras.layers.Dense(20, activation="relu"),
    tf.keras.layers.Dense(20, activation="relu"),
    tf.keras.layers.Dense(1)
])

neural.summary()
optimizer = tf.keras.optimizers.Adam(learning_rate=0.01)

neural.compile(loss="mse", optimizer=optimizer, metrics=["RootMeanSquaredError"])
norm_layer.adapt(X_train)
```

```
early_stop = EarlyStopping(monitor='loss' , patience=8, restore_best_weights=True)

history = neural.fit(X_train, y_train,epochs=200, validation_data=(X_valid, y_valid

mse_test, rmse_test = neural.evaluate(X_test, y_test)
y_pred_neural = neural.predict(X_test).flatten()
```

Model: "sequential_1"

Layer (type)	Output Shape
normalization_1 (Normalization)	(None, 20)
dense_5 (Dense)	(None, 40)
dense_6 (Dense)	(None, 40)
dense_7 (Dense)	(None, 20)
dense_8 (Dense)	(None, 20)
dense_9 (Dense)	(None, 1)

Total params: 3,782 (14.78 KB)
Trainable params: 3,741 (14.61 KB)
Non-trainable params: 41 (168.00 B)

```
Epoch 1/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - RootMeanSquaredError: 963604.6250 - loss: 976334028800.0000 - val_RootMeanSquaredError: 208812.9531 - val_loss: 43602849792.0000
Epoch 2/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 196359.6406 - loss: 38639693824.0000 - val_RootMeanSquaredError: 185679.9688 - val_loss: 34477051904.0000
Epoch 3/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 178167.1719 - loss: 31816962048.0000 - val_RootMeanSquaredError: 177932.0000 - val_loss: 31659798528.0000
Epoch 4/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 183717.7969 - loss: 34064617472.0000 - val_RootMeanSquaredError: 164822.7500 - val_loss: 27166539776.0000
Epoch 5/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 158205.1406 - loss: 25198088192.0000 - val_RootMeanSquaredError: 161400.7344 - val_loss: 26050199552.0000
Epoch 6/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 156266.6406 - loss: 24466370560.0000 - val_RootMeanSquaredError: 166677.6094 - val_loss: 27781427200.0000
Epoch 7/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 155310.1250 - loss: 24172572672.0000 - val_RootMeanSquaredError: 169498.1406 - val_loss: 28729620480.0000
Epoch 8/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 158128.4688 - loss: 25104959488.0000 - val_RootMeanSquaredError: 164658.0625 - val_loss: 27112280064.0000
Epoch 9/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 166937.3906 - loss: 28092598272.0000 - val_RootMeanSquaredError: 168084.1719 - val_loss: 28252289024.0000
Epoch 10/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 164732.6875 - loss: 27256557568.0000 - val_RootMeanSquaredError: 165200.2812 - val_loss: 27291133952.0000
Epoch 11/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 164172.2969 - loss: 27014825984.0000 - val_RootMeanSquaredError: 161584.8125 - val_loss: 26109652992.0000
Epoch 12/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 166777.1719 - loss: 27908141056.0000 - val_RootMeanSquaredError: 169402.8438 - val_loss: 28697325568.0000
Epoch 13/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 151269.9688 - loss: 22924986368.0000 - val_RootMeanSquaredError: 156061.5156 - val_loss: 24355198976.0000
Epoch 14/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 164741.6562 - loss: 27203053568.0000 - val_RootMeanSquaredError: 157688.1406 - val_loss: 24865548288.0000
```

```
Epoch 15/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 156279.1719 - loss:
24524445696.0000 - val_RootMeanSquaredError: 158035.2031 - val_loss: 24975126528.000
0
Epoch 16/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 152611.9062 - loss:
23343255552.0000 - val_RootMeanSquaredError: 167943.8281 - val_loss: 28205129728.000
0
Epoch 17/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 147289.1250 - loss:
21744451584.0000 - val_RootMeanSquaredError: 166254.6562 - val_loss: 27640612864.000
0
Epoch 18/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 159916.5156 - loss:
25714929664.0000 - val_RootMeanSquaredError: 175608.1094 - val_loss: 30838210560.000
0
Epoch 19/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 145763.5156 - loss:
21408700416.0000 - val_RootMeanSquaredError: 156332.1406 - val_loss: 24439736320.000
0
Epoch 20/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 159312.6875 - loss:
25444302848.0000 - val_RootMeanSquaredError: 158061.6562 - val_loss: 24983488512.000
0
Epoch 21/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 166598.5781 - loss:
28001155072.0000 - val_RootMeanSquaredError: 155923.6250 - val_loss: 24312176640.000
0
Epoch 22/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 148865.6406 - loss:
22218616832.0000 - val_RootMeanSquaredError: 163721.5781 - val_loss: 26804754432.000
0
Epoch 23/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 165012.9531 - loss:
27355957248.0000 - val_RootMeanSquaredError: 158796.8438 - val_loss: 25216438272.000
0
Epoch 24/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 156458.9844 - loss:
24699521024.0000 - val_RootMeanSquaredError: 164145.2969 - val_loss: 26943678464.000
0
Epoch 25/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 155320.1250 - loss:
24209688576.0000 - val_RootMeanSquaredError: 157768.8750 - val_loss: 24891017216.000
0
Epoch 26/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 140334.5156 - loss:
19797585920.0000 - val_RootMeanSquaredError: 198083.2969 - val_loss: 39236993024.000
0
Epoch 27/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 162549.5469 - loss:
27283767296.0000 - val_RootMeanSquaredError: 158705.8594 - val_loss: 25187551232.000
0
Epoch 28/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 141003.9062 - loss:
19978205184.0000 - val_RootMeanSquaredError: 167419.6562 - val_loss: 28029341696.000
0
```

```
Epoch 29/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 157583.5938 - loss: 
24930246656.0000 - val_RootMeanSquaredError: 184245.8125 - val_loss: 33946519552.000
0
Epoch 30/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 158985.8594 - loss: 
25577371648.0000 - val_RootMeanSquaredError: 159671.1875 - val_loss: 25494886400.000
0
Epoch 31/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 153280.4375 - loss: 
23515928576.0000 - val_RootMeanSquaredError: 158888.0312 - val_loss: 25245407232.000
0
Epoch 32/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 162900.3438 - loss: 
26663499776.0000 - val_RootMeanSquaredError: 157248.2188 - val_loss: 24727003136.000
0
Epoch 33/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 151795.0781 - loss: 
23180263424.0000 - val_RootMeanSquaredError: 150327.2344 - val_loss: 22598277120.000
0
Epoch 34/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 143724.5156 - loss: 
20710991872.0000 - val_RootMeanSquaredError: 154702.3438 - val_loss: 23932813312.000
0
Epoch 35/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 139473.6406 - loss: 
19587627008.0000 - val_RootMeanSquaredError: 140984.8125 - val_loss: 19876718592.000
0
Epoch 36/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 128503.4375 - loss: 
16580687872.0000 - val_RootMeanSquaredError: 124001.2734 - val_loss: 15376316416.000
0
Epoch 37/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 114375.7422 - loss: 
13142444032.0000 - val_RootMeanSquaredError: 114525.1797 - val_loss: 13116017664.000
0
Epoch 38/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 100234.2266 - loss: 
10080223232.0000 - val_RootMeanSquaredError: 109624.1797 - val_loss: 12017460224.000
0
Epoch 39/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 91799.6250 - loss: 
8458391552.0000 - val_RootMeanSquaredError: 91537.1328 - val_loss: 8379046400.0000
Epoch 40/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 92307.9688 - loss: 
8558575104.0000 - val_RootMeanSquaredError: 84380.0703 - val_loss: 7119995904.0000
Epoch 41/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 80515.1172 - loss: 
6535437824.0000 - val_RootMeanSquaredError: 86727.8516 - val_loss: 7521720320.0000
Epoch 42/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 82578.1875 - loss: 
6840122368.0000 - val_RootMeanSquaredError: 85369.6875 - val_loss: 7287983616.0000
Epoch 43/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 83715.7031 - loss: 
7047620608.0000 - val_RootMeanSquaredError: 98221.2734 - val_loss: 9647418368.0000
Epoch 44/200
```

```
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 76634.8438 - loss: 5908753408.0000 - val_RootMeanSquaredError: 71261.1250 - val_loss: 5078148096.0000
Epoch 45/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 76833.9922 - loss: 5927489024.0000 - val_RootMeanSquaredError: 90363.2031 - val_loss: 8165508096.0000
Epoch 46/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 77185.0312 - loss: 5975682560.0000 - val_RootMeanSquaredError: 73222.0312 - val_loss: 5361466368.0000
Epoch 47/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 70656.0547 - loss: 5021091328.0000 - val_RootMeanSquaredError: 72245.0156 - val_loss: 5219341824.0000
Epoch 48/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 67431.6719 - loss: 4581108736.0000 - val_RootMeanSquaredError: 76131.8594 - val_loss: 5796059648.0000
Epoch 49/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 70208.7734 - loss: 4995852288.0000 - val_RootMeanSquaredError: 71244.6484 - val_loss: 5075799552.0000
Epoch 50/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 69327.7969 - loss: 4812497408.0000 - val_RootMeanSquaredError: 64063.7852 - val_loss: 4104168448.0000
Epoch 51/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 66313.2188 - loss: 4422348800.0000 - val_RootMeanSquaredError: 66561.4453 - val_loss: 4430426112.0000
Epoch 52/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 62707.8594 - loss: 3961439744.0000 - val_RootMeanSquaredError: 61654.5664 - val_loss: 3801285376.0000
Epoch 53/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 59224.3633 - loss: 3528701952.0000 - val_RootMeanSquaredError: 66375.1875 - val_loss: 4405665792.0000
Epoch 54/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 57926.2188 - loss: 3384868096.0000 - val_RootMeanSquaredError: 67357.7109 - val_loss: 4537061376.0000
Epoch 55/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 61414.2344 - loss: 3786523392.0000 - val_RootMeanSquaredError: 77119.3516 - val_loss: 5947394048.0000
Epoch 56/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 69184.9688 - loss: 4809823744.0000 - val_RootMeanSquaredError: 59024.0312 - val_loss: 3483836416.0000
Epoch 57/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 62509.4727 - loss: 3929947904.0000 - val_RootMeanSquaredError: 56619.6172 - val_loss: 3205781248.0000
Epoch 58/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 57517.9258 - loss: 3326776064.0000 - val_RootMeanSquaredError: 60478.0234 - val_loss: 3657591296.0000
Epoch 59/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 59059.9102 - loss: 3497348096.0000 - val_RootMeanSquaredError: 56085.8750 - val_loss: 3145625344.0000
Epoch 60/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 49085.7344 - loss: 2422233344.0000 - val_RootMeanSquaredError: 67914.7891 - val_loss: 4612419072.0000
Epoch 61/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 54819.1406 - loss: 3020486400.0000 - val_RootMeanSquaredError: 53053.3984 - val_loss: 2814663168.0000
Epoch 62/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 50755.3047 - loss: 2579106816.0000 - val_RootMeanSquaredError: 59036.7461 - val_loss: 3485337344.0000
```

```
Epoch 63/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 52595.7852 - loss:
2772643840.0000 - val_RootMeanSquaredError: 53696.3320 - val_loss: 2883296000.0000
Epoch 64/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 49643.5547 - loss:
2494048512.0000 - val_RootMeanSquaredError: 49533.4375 - val_loss: 2453561600.0000
Epoch 65/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 52853.9570 - loss:
2824523520.0000 - val_RootMeanSquaredError: 50982.4609 - val_loss: 2599211520.0000
Epoch 66/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 50013.2578 - loss:
2520503552.0000 - val_RootMeanSquaredError: 50249.7617 - val_loss: 2525038592.0000
Epoch 67/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 50086.1914 - loss:
2547925760.0000 - val_RootMeanSquaredError: 46318.8750 - val_loss: 2145438080.0000
Epoch 68/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - RootMeanSquaredError: 50266.0938 - loss:
2538852096.0000 - val_RootMeanSquaredError: 52802.2227 - val_loss: 2788074752.0000
Epoch 69/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 50141.1250 - loss:
2538249728.0000 - val_RootMeanSquaredError: 49751.1133 - val_loss: 2475173120.0000
Epoch 70/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 49423.0156 - loss:
2462692864.0000 - val_RootMeanSquaredError: 48867.4961 - val_loss: 2388032000.0000
Epoch 71/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 40739.2773 - loss:
1667119744.0000 - val_RootMeanSquaredError: 46794.8008 - val_loss: 2189753344.0000
Epoch 72/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 40514.1484 - loss:
1659602304.0000 - val_RootMeanSquaredError: 44624.5312 - val_loss: 1991348864.0000
Epoch 73/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 43116.0273 - loss:
1889467136.0000 - val_RootMeanSquaredError: 45154.6250 - val_loss: 2038940160.0000
Epoch 74/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 44435.0195 - loss:
1984789248.0000 - val_RootMeanSquaredError: 46678.8984 - val_loss: 2178919680.0000
Epoch 75/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 40274.9688 - loss:
1639586944.0000 - val_RootMeanSquaredError: 58037.9883 - val_loss: 3368408064.0000
Epoch 76/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 52671.5312 - loss:
2787826688.0000 - val_RootMeanSquaredError: 46604.5547 - val_loss: 2171984384.0000
Epoch 77/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 42760.9453 - loss:
1838533760.0000 - val_RootMeanSquaredError: 43696.5430 - val_loss: 1909387904.0000
Epoch 78/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 47188.4219 - loss:
2313256960.0000 - val_RootMeanSquaredError: 42960.5156 - val_loss: 1845606016.0000
Epoch 79/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 39599.7344 - loss:
1580497152.0000 - val_RootMeanSquaredError: 61712.3008 - val_loss: 3808408064.0000
Epoch 80/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 39913.6289 - loss:
1601125376.0000 - val_RootMeanSquaredError: 48129.7969 - val_loss: 2316477184.0000
Epoch 81/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 43576.8945 - loss:
```

```
1905570176.0000 - val_RootMeanSquaredError: 45831.2695 - val_loss: 2100505088.0000
Epoch 82/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 39225.9961 - loss: 1558875904.0000 - val_RootMeanSquaredError: 46151.6641 - val_loss: 2129975936.0000
Epoch 83/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 43123.7500 - loss: 1890843648.0000 - val_RootMeanSquaredError: 50079.3398 - val_loss: 2507940096.0000
Epoch 84/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 40532.0039 - loss: 1646575232.0000 - val_RootMeanSquaredError: 44235.9258 - val_loss: 1956817152.0000
Epoch 85/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 39132.6016 - loss: 1558477696.0000 - val_RootMeanSquaredError: 43887.3086 - val_loss: 1926095872.0000
Epoch 86/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 43729.3086 - loss: 1933545344.0000 - val_RootMeanSquaredError: 41167.1055 - val_loss: 1694730624.0000
Epoch 87/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 35641.1367 - loss: 1277508864.0000 - val_RootMeanSquaredError: 41417.3203 - val_loss: 1715394560.0000
Epoch 88/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 37536.9141 - loss: 1419605632.0000 - val_RootMeanSquaredError: 43877.5977 - val_loss: 1925243648.0000
Epoch 89/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 40018.3242 - loss: 1611876992.0000 - val_RootMeanSquaredError: 41524.2109 - val_loss: 1724260096.0000
Epoch 90/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 35129.8438 - loss: 1237895808.0000 - val_RootMeanSquaredError: 38333.7344 - val_loss: 1469475328.0000
Epoch 91/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 34039.3164 - loss: 1171771264.0000 - val_RootMeanSquaredError: 41634.7305 - val_loss: 1733450752.0000
Epoch 92/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 37254.3828 - loss: 1422980224.0000 - val_RootMeanSquaredError: 41578.4805 - val_loss: 1728770048.0000
Epoch 93/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 42820.7695 - loss: 1863052544.0000 - val_RootMeanSquaredError: 41215.9766 - val_loss: 1698756608.0000
Epoch 94/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 37637.6328 - loss: 1422899456.0000 - val_RootMeanSquaredError: 42070.2422 - val_loss: 1769905152.0000
Epoch 95/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 40571.4922 - loss: 1672130432.0000 - val_RootMeanSquaredError: 40610.9336 - val_loss: 1649248000.0000
Epoch 96/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 36945.2266 - loss: 1372864256.0000 - val_RootMeanSquaredError: 43193.3203 - val_loss: 1865662976.0000
Epoch 97/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 36554.0977 - loss: 1346914176.0000 - val_RootMeanSquaredError: 40940.4453 - val_loss: 1676120064.0000
Epoch 98/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 38904.1680 - loss: 1525437440.0000 - val_RootMeanSquaredError: 51152.7461 - val_loss: 2616603392.0000
Epoch 99/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 51243.5156 - loss: 2650394112.0000 - val_RootMeanSquaredError: 35577.1758 - val_loss: 1265735424.0000
Epoch 100/200
```

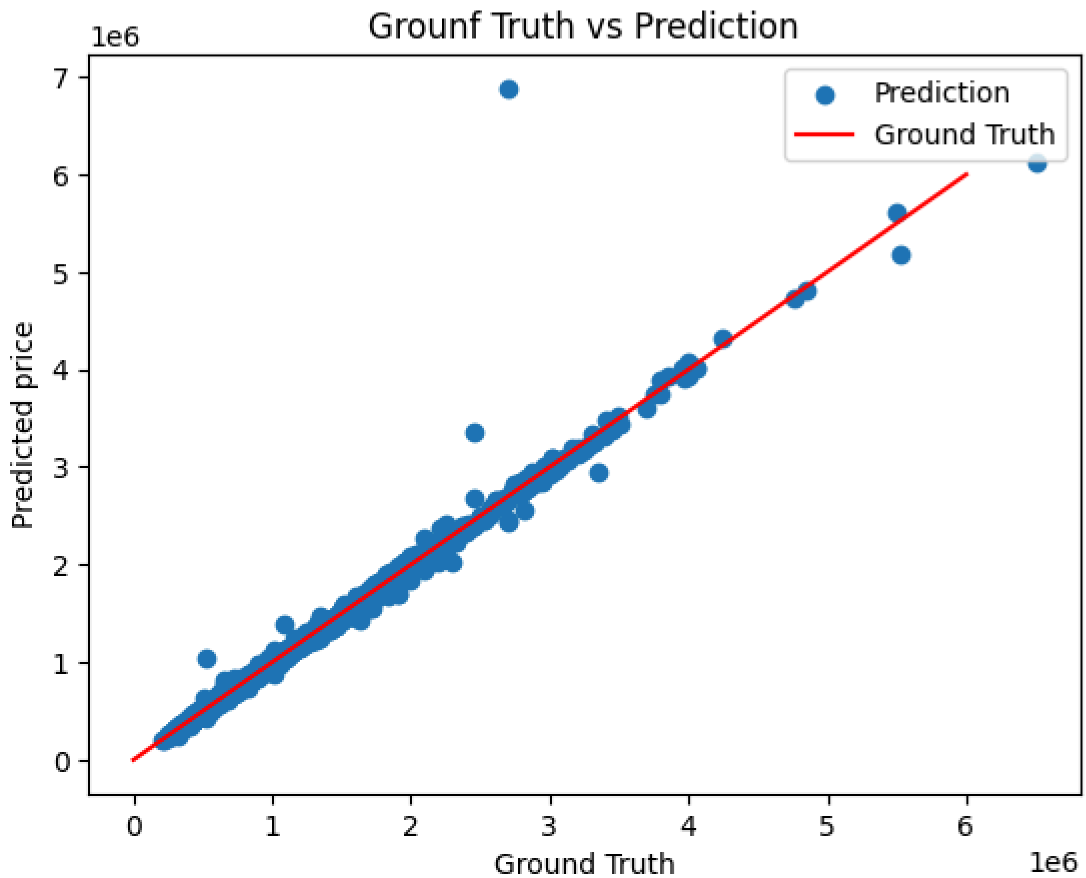
```
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 36549.5352 - loss: 1342183680.0000 - val_RootMeanSquaredError: 39003.7344 - val_loss: 1521291264.0000
Epoch 101/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 38568.2617 - loss: 1497288576.0000 - val_RootMeanSquaredError: 36647.7891 - val_loss: 1343060480.0000
Epoch 102/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 31019.2773 - loss: 969440256.0000 - val_RootMeanSquaredError: 41492.4023 - val_loss: 1721619328.0000
Epoch 103/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 35138.3125 - loss: 1238562048.0000 - val_RootMeanSquaredError: 41318.9219 - val_loss: 1707253376.0000
Epoch 104/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 38704.2344 - loss: 1504757504.0000 - val_RootMeanSquaredError: 34776.6016 - val_loss: 1209412096.0000
Epoch 105/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 31254.8965 - loss: 981165056.0000 - val_RootMeanSquaredError: 37276.0469 - val_loss: 1389503744.0000
Epoch 106/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 35468.9219 - loss: 1271557120.0000 - val_RootMeanSquaredError: 41148.7188 - val_loss: 1693217024.0000
Epoch 107/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 32397.8594 - loss: 1057109696.0000 - val_RootMeanSquaredError: 65427.2188 - val_loss: 4280720896.0000
Epoch 108/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 40654.1445 - loss: 1671513728.0000 - val_RootMeanSquaredError: 38120.9492 - val_loss: 1453206656.0000
Epoch 109/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 35405.9805 - loss: 1263163392.0000 - val_RootMeanSquaredError: 34739.2031 - val_loss: 1206812288.0000
Epoch 110/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - RootMeanSquaredError: 30565.7285 - loss: 943155584.0000 - val_RootMeanSquaredError: 37307.5859 - val_loss: 1391855872.0000
Epoch 111/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 30919.1699 - loss: 970263296.0000 - val_RootMeanSquaredError: 38879.1367 - val_loss: 1511587328.0000
Epoch 112/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 32315.9668 - loss: 1048612224.0000 - val_RootMeanSquaredError: 43190.4688 - val_loss: 1865416448.0000
Epoch 113/200
306/306 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - RootMeanSquaredError: 30925.8516 - loss: 959217472.0000 - val_RootMeanSquaredError: 40481.9219 - val_loss: 1638785920.0000
85/85 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - RootMeanSquaredError: 101308.0391 - loss: 1
1849688064.0000
85/85 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
```

In [94]:

```python
from sklearn.metrics import *

MSE=mean_squared_error(y_test,y_pred_neural)
RMSE = np.sqrt(MSE)
R2=r2_score(y_test,y_pred_neural)

dict ['Neural Networks'] = {'rmse' : round(RMSE , 1) , 'r2_score' : R2}

plt.scatter(y_test,y_pred_neural)
plt.plot([0,6000000],[0,6000000],c='r')
plt.xlabel('Ground Truth')
```

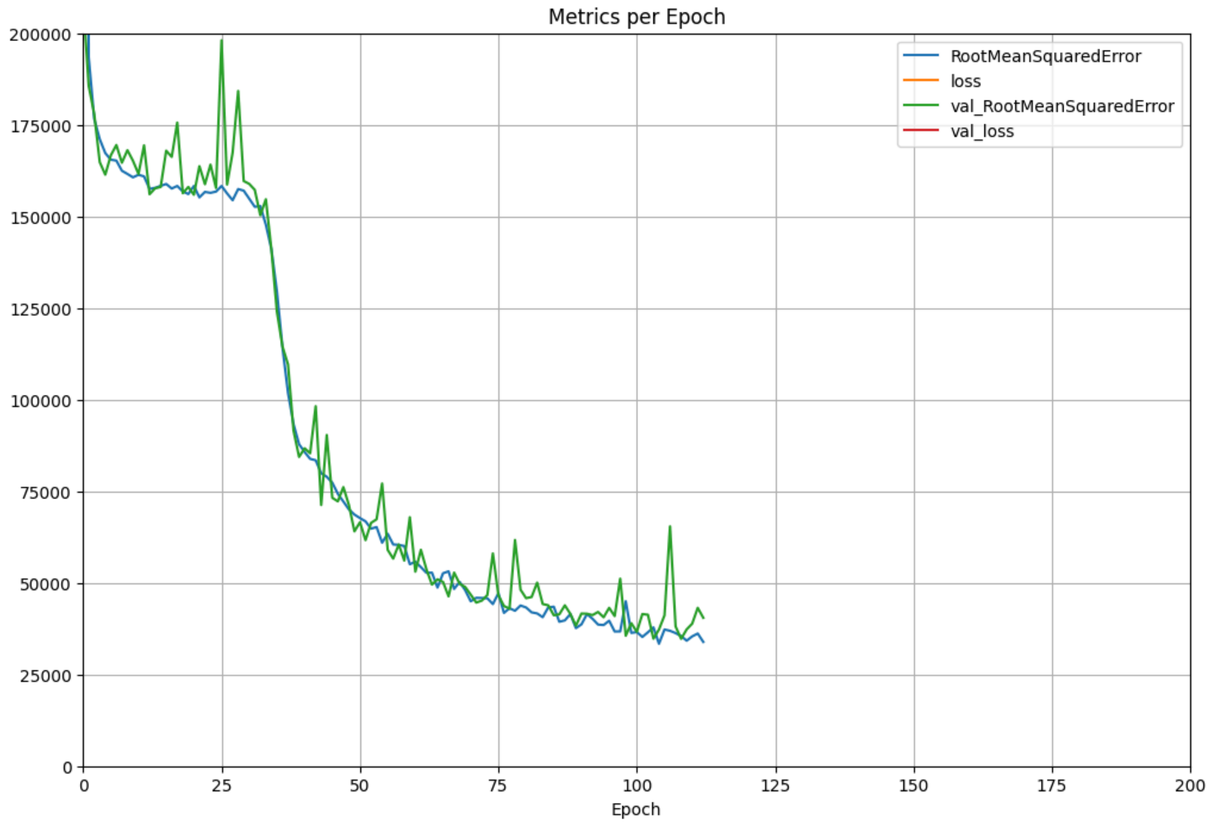
```
plt.ylabel('Predicted price')
plt.title('Grounf Truth vs Prediction')
plt.legend(['Prediction','Ground Truth'])
```

Out[94]: <matplotlib.legend.Legend at 0x1d5cf31b880>



In [126…

```
pd.DataFrame(history.history).plot(
    figsize=(12, 8),xlim=[0, 200], ylim= [0,2e5],grid=True, xlabel="Epoch")
plt.legend(loc="upper right")  # extra code
plt.title('Metrics per Epoch')
plt.show()
```



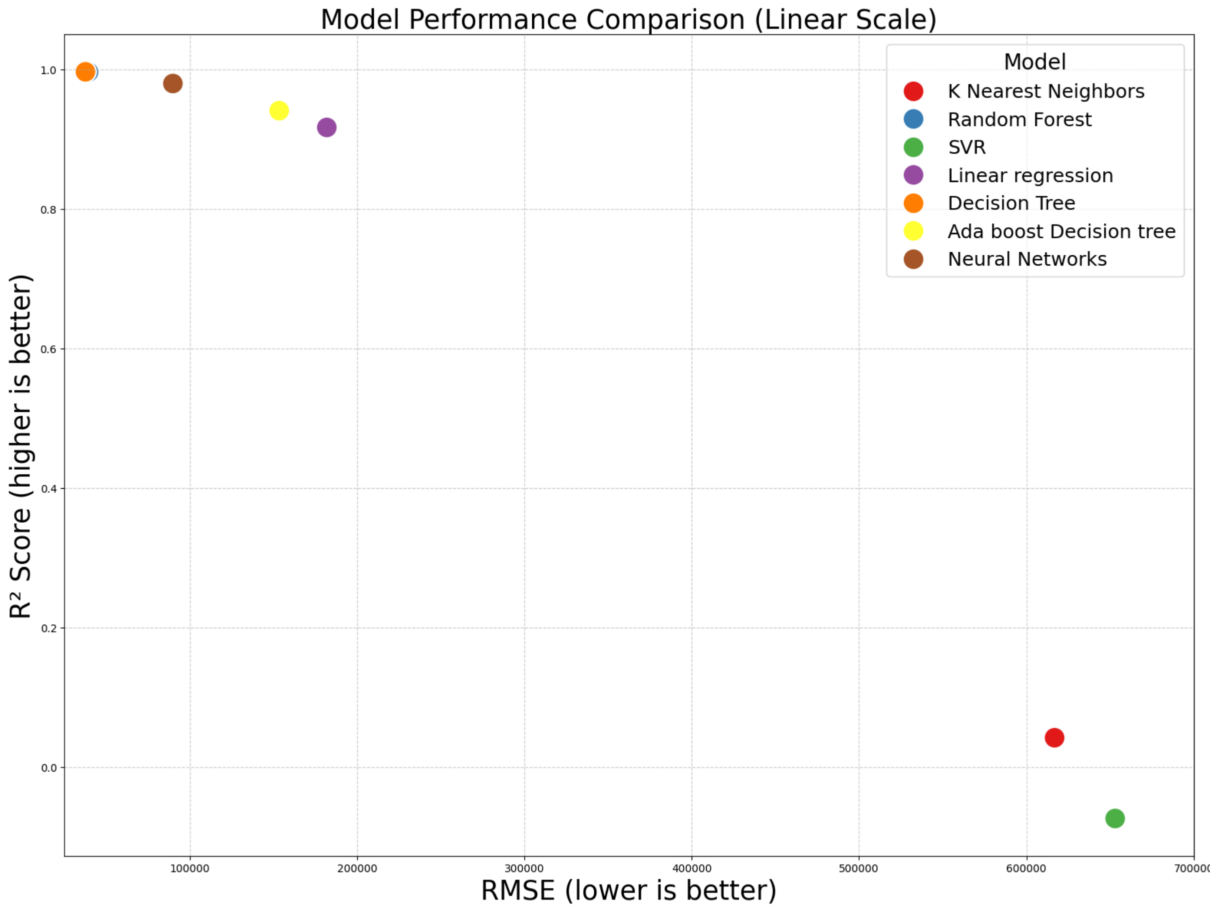
Comparing the Metrics of the Regressers with the Neural Networks

In [97]:
```
    result += f"Neural Networks:\n"
    result += f"  RMSE: {RMSE:.4f}\n"
    result += f"  R2 Score: {R2:.4f}\n\n"

print(result)
```

```
K Nearest Neighbors:
  RMSE: 616867.0732
  R2 Score: 0.0420

Random Forest:
  RMSE: 39461.3874
  R2 Score: 0.9961

SVR:
  RMSE: 653063.0216
  R2 Score: -0.0737

Linear regression:
  RMSE: 181885.5908
  R2 Score: 0.9167

Decision Tree:
  RMSE: 37638.8108
  R2 Score: 0.9964

Ada boost Decision tree:
  RMSE: 153431.4651
  R2 Score: 0.9407

Neural Networks:
  RMSE: 89881.5069
  R2 Score: 0.9797
```

In [120…

```
import seaborn as sns

model_data = pd.DataFrame(dict).T
model_data["Model"] = model_data.index

palette = sns.color_palette("Set1", n_colors=len(model_data))
plt.figure(figsize=(16, 12))
sns.scatterplot(data=model_data, x="rmse", y="r2_score", hue="Model", s=400, palett
plt.xlim(25000, 700000)
plt.xlabel("RMSE (lower is better)", fontsize=25)
plt.ylabel("R² Score (higher is better)", fontsize=25)
plt.title("Model Performance Comparison (Linear Scale)", fontsize=25)
plt.grid(True, linestyle="--", alpha=0.6)
plt.legend(title="Model", loc="best", fontsize=18, title_fontsize=20)
plt.tight_layout()
plt.show()
```



After running all the models on the same dataset, We conclude that for this DATASET not only **Neural Networks** gives excelent results, Other regressers such as **Decision Tree** also gives very good results as good as Neural Networks if not better.

The End of the Project