```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
import pandas as pd
from sklearn.model_selection import train_test_split
import keras
from keras.layers import Dense
from keras.models import Sequential
import matplotlib.pyplot as plt
from keras.layers import Input
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score
```

https://www.kaggle.com/datasets/shree1992/housedata/data

```
df = pd.read_csv('priceprediction.csv')

df.head()
```

```
                 date      price  bedrooms  bathrooms  sqft_living
sqft_lot  \
0  2014-05-02 00:00:00   313000.0       3.0       1.50         1340
7912
1  2014-05-02 00:00:00  2384000.0       5.0       2.50         3650
9050
2  2014-05-02 00:00:00   342000.0       3.0       2.00         1930
11947
3  2014-05-02 00:00:00   420000.0       3.0       2.25         2000
8030
4  2014-05-02 00:00:00   550000.0       4.0       2.50         1940
10500

   floors  waterfront  view  condition  sqft_above  sqft_basement
yr_built  \
0     1.5           0     0          3        1340              0
1955
1     2.0           0     4          5        3370            280
1921
2     1.0           0     0          4        1930              0
1966
3     1.0           0     0          4        1000           1000
1963
4     1.0           0     0          4        1140            800
1976

   yr_renovated                    street       city  statezip country

0          2005      18810 Densmore Ave N  Shoreline  WA 98133     USA

1             0           709 W Blaine St    Seattle  WA 98119     USA

2             0  26206-26214 143rd Ave SE       Kent  WA 98042     USA
```

```
3              0           857 170th Pl NE   Bellevue  WA 98008     USA

4           1992          9105 170th Ave NE    Redmond  WA 98052     USA
```

```
print(df.isnull().sum())
```

```
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
street           0
city             0
statezip         0
country          0
dtype: int64
```

```
#df.fillna(df.mean(), inplace=True)
#df.fillna(df.mode().iloc[0], inplace=True)

df.drop(columns=['date'], inplace=True)  # Remove date column

from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
df['city'] = label_encoder.fit_transform(df['city'])  # Convert city
names to numbers
df['street'] = label_encoder.fit_transform(df['street'])
df['statezip'] = label_encoder.fit_transform(df['statezip'])
df['country'] = label_encoder.fit_transform(df['country'])

print(df.dtypes)
```

```
price            float64
bedrooms         float64
bathrooms        float64
sqft_living        int64
sqft_lot           int64
floors           float64
waterfront         int64
```

```
view                int64
condition           int64
sqft_above          int64
sqft_basement       int64
yr_built            int64
yr_renovated        int64
street              int64
city                int64
statezip            int64
country             int64
dtype: object
```

```
X = df.loc[:, df.columns != 'price']
y = df['price']
n_cols = X.shape[1]

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)



model1 = Sequential()
model1.add(Input(shape=(n_cols,)))
model1.add(Dense(50, activation='relu'))
model1.add(Dense(32, activation='relu'))
model1.add(Dense(1, activation='linear'))
model1.compile(optimizer='adam', loss='mean_squared_error')
history1 = model1.fit(X_train, y_train, epochs=10 , batch_size=15)
```

```
Epoch 1/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 9.4747e-04
Epoch 2/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0047
Epoch 3/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 0.0030
Epoch 4/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0021
Epoch 5/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0026
Epoch 6/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 0.0313
Epoch 7/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 9.5147e-04
Epoch 8/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0013
Epoch 9/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 0.0046
Epoch 10/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 6.5084e-04
```

```
model2 = Sequential()
model2.add(Input(shape=(n_cols,)))
```

```
model2.add(Dense(50, activation='relu'))
model2.add(Dense(32, activation='relu'))
model2.add(Dense(1, activation='linear'))
model2.compile(optimizer='adam', loss='mean_absolute_error')
history2 = model2.fit(X_train, y_train, epochs=10 , batch_size=15)
```

```
Epoch 1/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 0.0014
Epoch 2/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 0.0012
Epoch 3/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0017
Epoch 4/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0124
Epoch 5/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 5.6327e-04
Epoch 6/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0012
Epoch 7/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0032
Epoch 8/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0011
Epoch 9/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0025
Epoch 10/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0020
```

```
scaler_X = StandardScaler()
X_train = scaler_X.fit_transform(X_train)
X_test = scaler_X.transform(X_test)

scaler_y = MinMaxScaler()
y_train = scaler_y.fit_transform(y_train.values.reshape(-1, 1))
y_test = scaler_y.transform(y_test.values.reshape(-1, 1))

model3 = Sequential()
model3.add(Input(shape=(n_cols,)))
model3.add(Dense(50, activation='relu'))
model3.add(Dense(32, activation='relu'))
model3.add(Dense(1, activation='linear'))
model3.compile(optimizer='adam', loss='mean_squared_error')
history3 = model3.fit(X_train, y_train, epochs=10 , batch_size=15)
```

```
Epoch 1/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 0.0029
Epoch 2/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 9.2571e-04
Epoch 3/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0017
Epoch 4/10
```

```
246/246 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0058
Epoch 5/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 4.7461e-04
Epoch 6/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 8.2941e-04
Epoch 7/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 0.0011
Epoch 8/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0052
Epoch 9/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0022
Epoch 10/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 0.0019
```

```
model4 = Sequential()
model4.add(Input(shape=(n_cols,)))
model4.add(Dense(50, activation='relu'))
model4.add(Dense(32, activation='relu'))
model4.add(Dense(1, activation='linear'))
model4.compile(optimizer='adam', loss='mean_absolute_error')
history4 = model4.fit(X_train, y_train, epochs=10 , batch_size=15)
```

```
Epoch 1/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 0.0024
Epoch 2/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0064
Epoch 3/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0022
Epoch 4/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 5.0927e-04
Epoch 5/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0017
Epoch 6/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0010
Epoch 7/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0104
Epoch 8/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0025
Epoch 9/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0011
Epoch 10/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0046
```

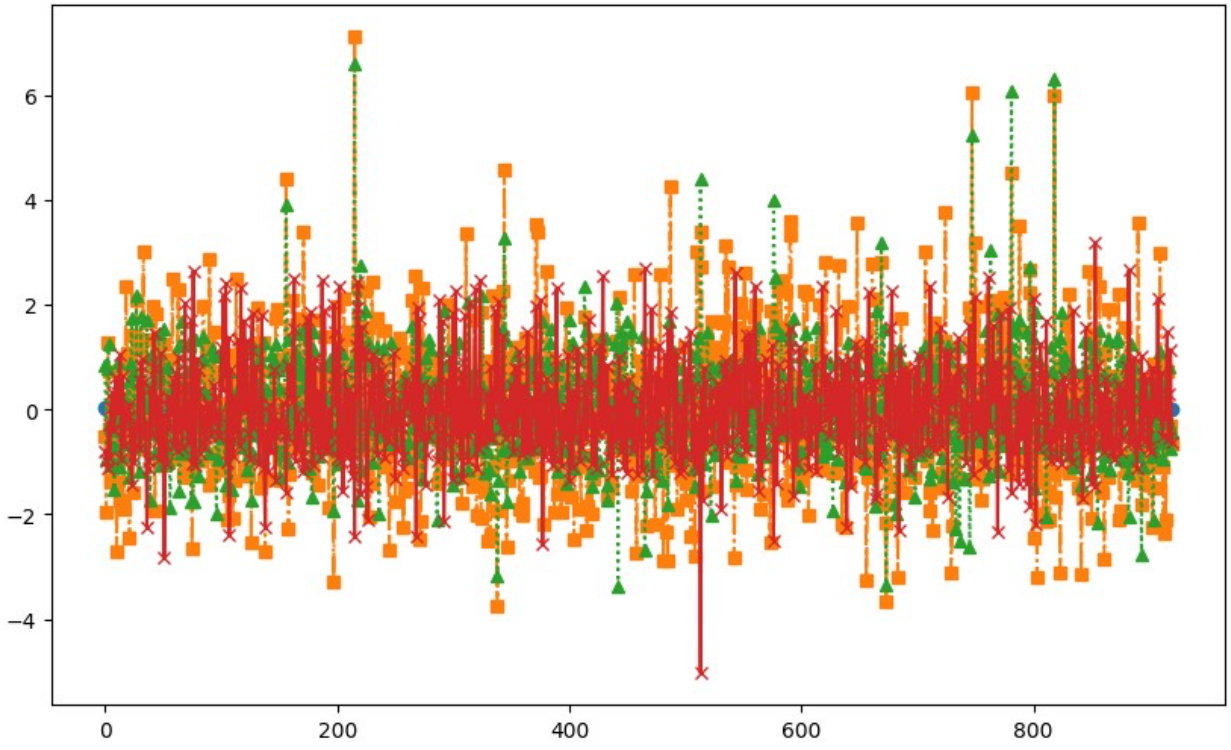
```
y_pred1 = model1.predict(X_test)
y_pred2 = model2.predict(X_test)
y_pred3 = model3.predict(X_test)
y_pred4 = model4.predict(X_test)
```

```
29/29 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```

```
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```

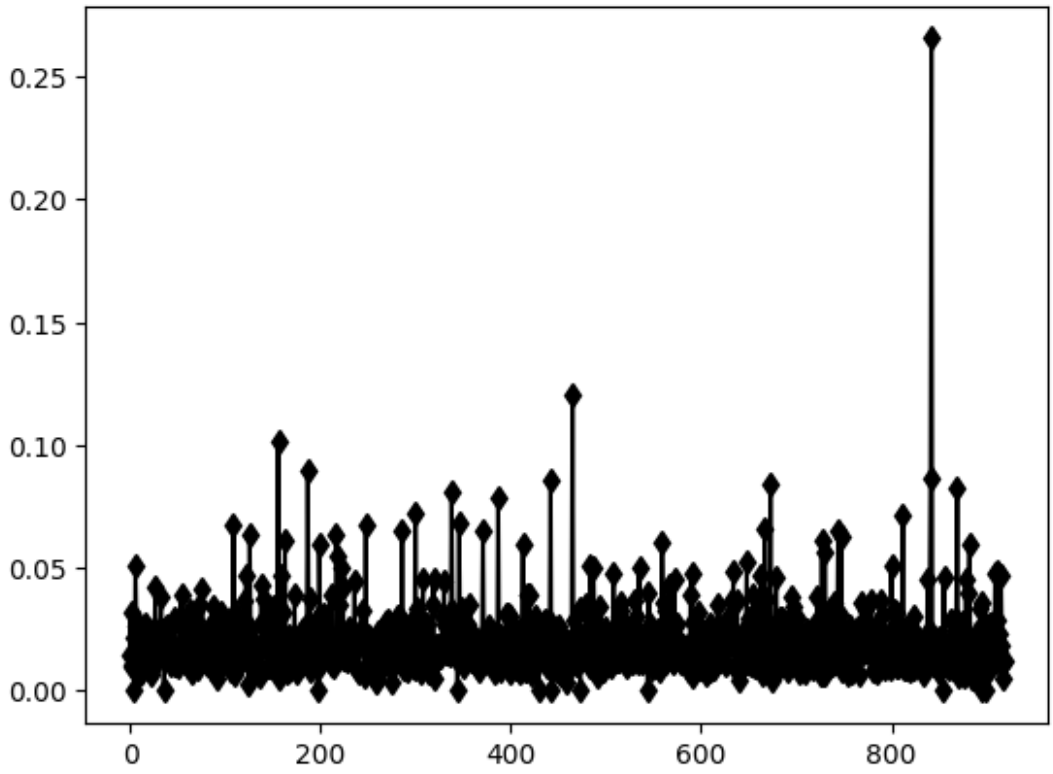
```
plt.figure(figsize=(10, 6))
plt.plot(y_pred1, label="Model 1 Predictions", linestyle="--",
marker="o")
plt.plot(y_pred2, label="Model 2 Predictions", linestyle="-.",
marker="s")
plt.plot(y_pred3, label="Model 3 Predictions", linestyle=":",
marker="^")
plt.plot(y_pred4, label="Model 4 Predictions", linestyle="-",
marker="x")
```

```
[<matplotlib.lines.Line2D at 0x30fc816d0>]
```




```
plt.plot(y_test, label="Actual Values", linestyle="-", marker="d",
color='black')
```

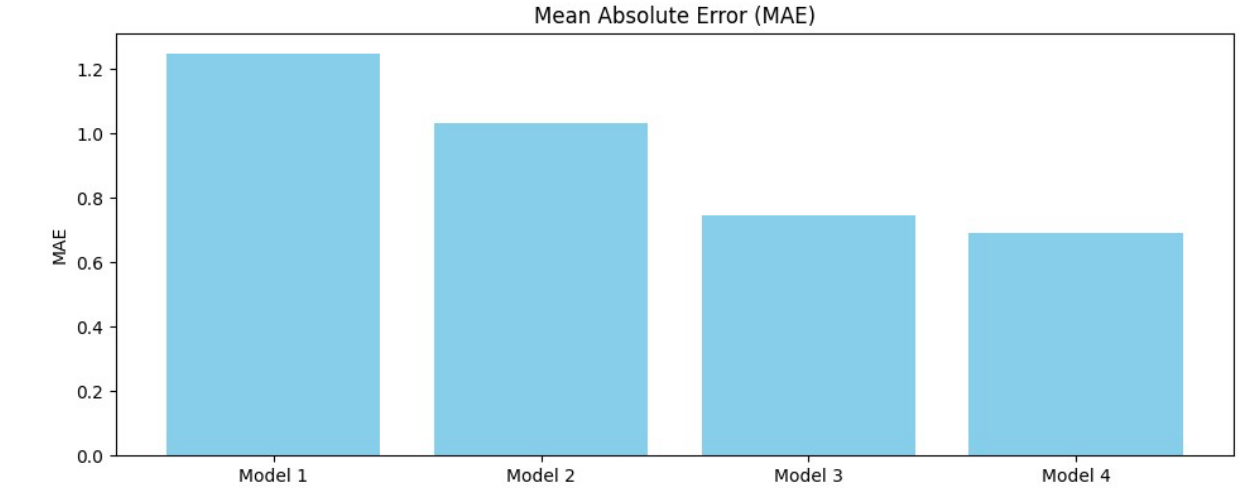
```
[<matplotlib.lines.Line2D at 0x3191e2f90>]
```

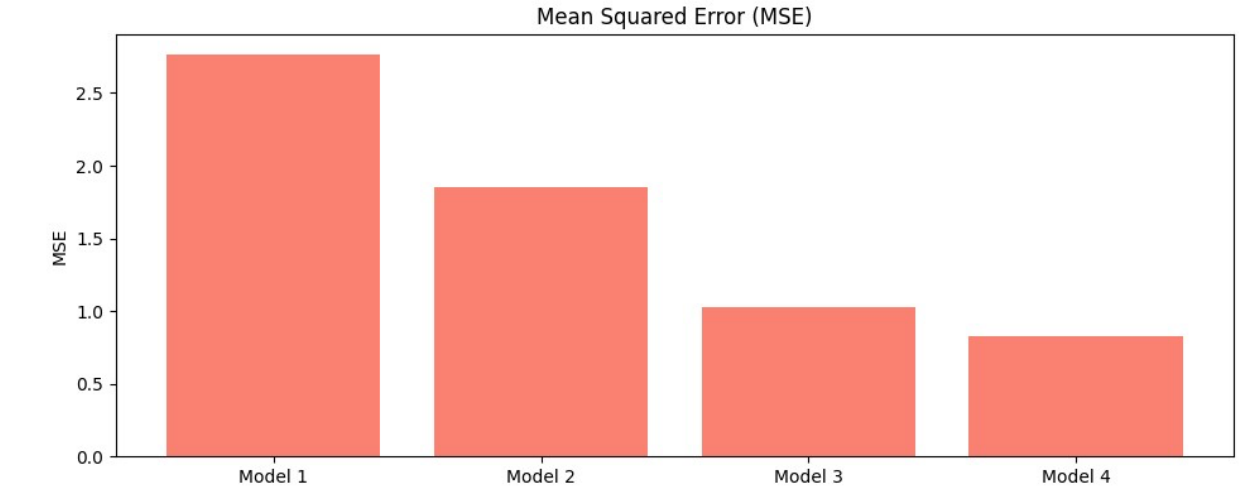


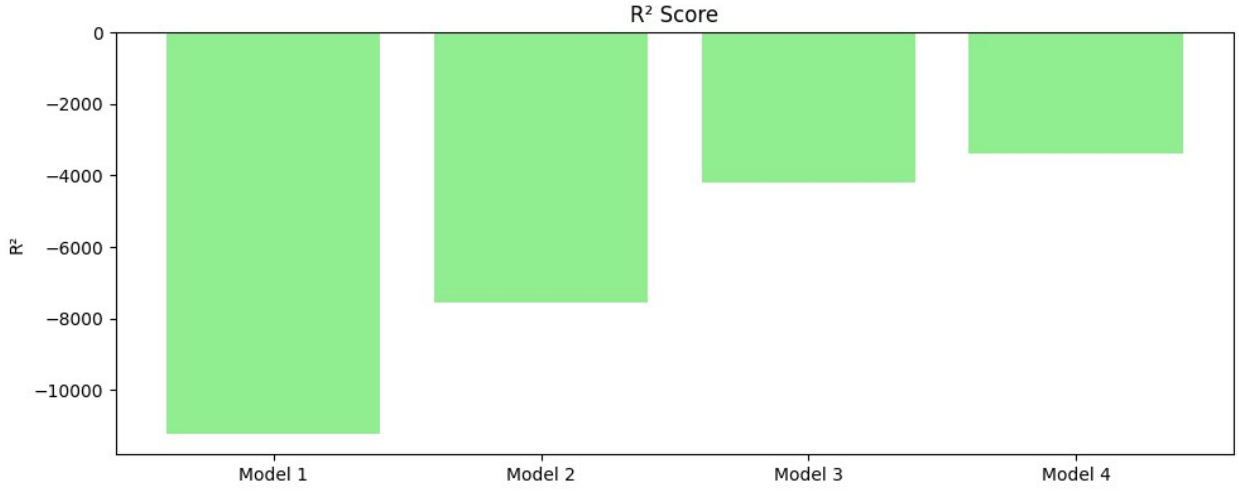

```
# Make predictions
y_pred1 = model1.predict(X_test)
y_pred2 = model2.predict(X_test)
y_pred3 = model3.predict(X_test)
y_pred4 = model4.predict(X_test)

# Calculate performance metrics for each model
mae1 = mean_absolute_error(y_test, y_pred1)
mae2 = mean_absolute_error(y_test, y_pred2)
mae3 = mean_absolute_error(y_test, y_pred3)
mae4 = mean_absolute_error(y_test, y_pred4)

mse1 = mean_squared_error(y_test, y_pred1)
mse2 = mean_squared_error(y_test, y_pred2)
mse3 = mean_squared_error(y_test, y_pred3)
mse4 = mean_squared_error(y_test, y_pred4)

r2_1 = r2_score(y_test, y_pred1)
r2_2 = r2_score(y_test, y_pred2)
r2_3 = r2_score(y_test, y_pred3)
r2_4 = r2_score(y_test, y_pred4)

# Store the results in a dictionary for easy access
metrics = {
```

```
    "Model 1": {"MAE": mae1, "MSE": mse1, "R²": r2_1},
    "Model 2": {"MAE": mae2, "MSE": mse2, "R²": r2_2},
    "Model 3": {"MAE": mae3, "MSE": mse3, "R²": r2_3},
    "Model 4": {"MAE": mae4, "MSE": mse4, "R²": r2_4},
}

# Plot comparison of MAE, MSE, and R² scores
labels = list(metrics.keys())
mae_values = [metrics[model]["MAE"] for model in labels]
mse_values = [metrics[model]["MSE"] for model in labels]
r2_values = [metrics[model]["R²"] for model in labels]

# Create subplots for each metric
fig, axs = plt.subplots(3, 1, figsize=(10, 12))

# Plot MAE
axs[0].bar(labels, mae_values, color='skyblue')
axs[0].set_title('Mean Absolute Error (MAE)')
axs[0].set_ylabel('MAE')

# Plot MSE
axs[1].bar(labels, mse_values, color='salmon')
axs[1].set_title('Mean Squared Error (MSE)')
axs[1].set_ylabel('MSE')

# Plot R² score
axs[2].bar(labels, r2_values, color='lightgreen')
axs[2].set_title('R² Score')
axs[2].set_ylabel('R²')

# Show plots
plt.tight_layout()
plt.show()

# Optionally, print the metrics
for model in labels:
    print(f"{model} - MAE: {metrics[model]['MAE']}, MSE:
{metrics[model]['MSE']}, R²: {metrics[model]['R²']}")
```

```
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```








```
Model 1 - MAE: 1.2474446146104057, MSE: 2.7628203271936504, R²: -
11245.404893197656
Model 2 - MAE: 1.0318428290087165, MSE: 1.8531256346962948, R²: -
7542.379133498624
Model 3 - MAE: 0.746621108700076, MSE: 1.0306071336791365, R²: -
```

```
4194.214939274279
Model 4 - MAE: 0.6900820352088723, MSE: 0.8290000410976051, R²: -
3373.548111903942
```

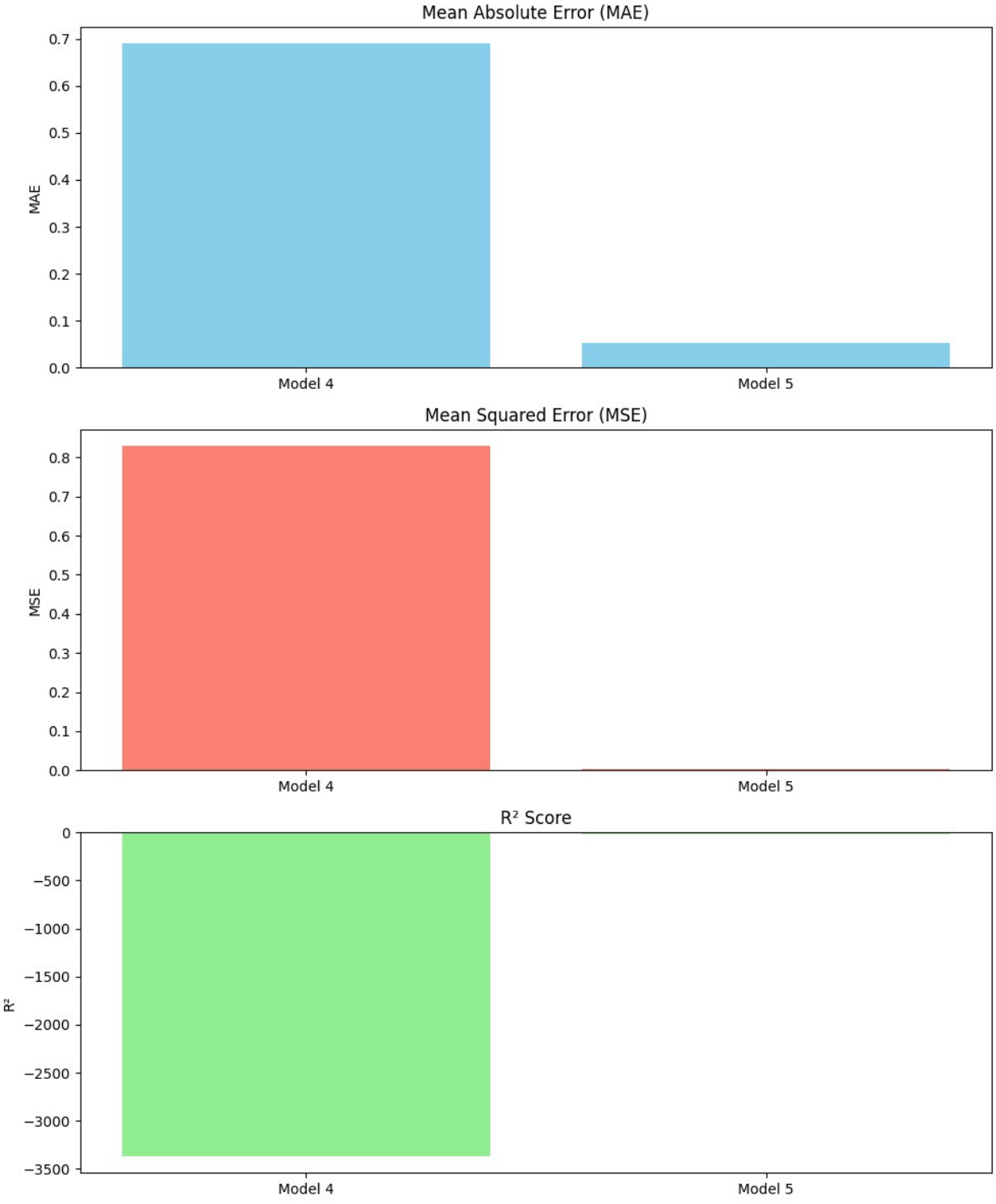
```
model5 = Sequential()
model5.add(Input(shape=(n_cols,)))
model5.add(Dense(128, activation='relu'))
model5.add(Dense(64, activation='relu'))
model5.add(Dense(32, activation='relu'))
model5.add(Dense(1, activation='linear'))
model5.compile(optimizer='adam', loss='mean_absolute_error')
history5 = model5.fit(X_train, y_train, epochs=10 , batch_size=15)
```

```
Epoch 1/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - loss: 0.1341
Epoch 2/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0455
Epoch 3/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0319
Epoch 4/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0349
Epoch 5/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0357
Epoch 6/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0313
Epoch 7/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0279
Epoch 8/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0322
Epoch 9/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0301
Epoch 10/10
246/246 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0333
```

```
# Make predictions for both models
y_pred4 = model4.predict(X_test)
y_pred5 = model5.predict(X_test)

# Calculate performance metrics for each model
mae4 = mean_absolute_error(y_test, y_pred4)
mae5 = mean_absolute_error(y_test, y_pred5)

mse4 = mean_squared_error(y_test, y_pred4)
mse5 = mean_squared_error(y_test, y_pred5)

r2_4 = r2_score(y_test, y_pred4)
r2_5 = r2_score(y_test, y_pred5)

# Store the results in a dictionary for easy access
metrics = {
```

```
    "Model 4": {"MAE": mae4, "MSE": mse4, "R²": r2_4},
    "Model 5": {"MAE": mae5, "MSE": mse5, "R²": r2_5},
}

# Plot comparison of MAE, MSE, and R² scores
labels = list(metrics.keys())
mae_values = [metrics[model]["MAE"] for model in labels]
mse_values = [metrics[model]["MSE"] for model in labels]
r2_values = [metrics[model]["R²"] for model in labels]

# Create subplots for each metric
fig, axs = plt.subplots(3, 1, figsize=(10, 12))

# Plot MAE
axs[0].bar(labels, mae_values, color='skyblue')
axs[0].set_title('Mean Absolute Error (MAE)')
axs[0].set_ylabel('MAE')

# Plot MSE
axs[1].bar(labels, mse_values, color='salmon')
axs[1].set_title('Mean Squared Error (MSE)')
axs[1].set_ylabel('MSE')

# Plot R² score
axs[2].bar(labels, r2_values, color='lightgreen')
axs[2].set_title('R² Score')
axs[2].set_ylabel('R²')

# Show plots
plt.tight_layout()
plt.show()

# Optionally, print the metrics
for model in labels:
    print(f"{model} - MAE: {metrics[model]['MAE']}, MSE:
{metrics[model]['MSE']}, R²: {metrics[model]['R²']}")
```

```
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
```




```
Model 4 - MAE: 0.6900820352088723, MSE: 0.8290000410976051, R²: -
3373.548111903942
Model 5 - MAE: 0.052906997774860076, MSE: 0.004150765104040483, R²: -
15.896207298435371
```