

DAFTAR ISI

BAB I PENDAHULUAN	2
1.1 Latar Belakang	2
1.2 Tujuan	2
BAB II PEMBAHASAN.....	3
A. Machine Learning.....	3
B. Supervised Learning.....	3
C. Mean Squared Error	3
D. Linear Regression	3
E. K-Neighbors Regressor.....	4
F. Random Forest	4
G. Gradient Boosting Regressor.....	4
H. Neural Networks	4
BAB III ANALISA DAN PERANCANGAN.....	5
1. Struktur File.....	5
2. Mengeload, Membersihkan dan Mempersiapkan data.....	5
3. Training	7
4. Performa Model	11
BAB IV PENUTUP.....	11
4.1 Kesimpulan	11
4.2 Saran.....	11

BAB I PENDAHULUAN

1.1 Latar Belakang

Dalam era transformasi digital, penerapan teknologi kecerdasan buatan semakin mendominasi berbagai sektor, termasuk properti. Pada proyek ini, kami fokus pada pengembangan model supervised learning menggunakan jaringan saraf tiruan (neural network) untuk memprediksi harga rumah di California. Melibatkan data-data historis dan variabel yang relevan, proyek ini bertujuan untuk meningkatkan akurasi prediksi harga rumah, memfasilitasi pengambilan keputusan yang lebih tepat, dan merespons dinamika pasar properti yang cepat berubah. Dengan menerapkan konsep-konsep dasar dalam supervised learning, kami berharap model ini dapat memberikan kontribusi signifikan dalam mendukung analisis pasar real estate di California.

1.2 Tujuan

- Menerapkan Teknik Supervised Learning dengan Neural Networks untuk menciptakan model yang memprediksi harga perumahan di California.
- Menganalisa akurasi berbagai model lain selain Neural Networks dalam memprediksi harga perumahan.

BAB II PEMBAHASAN

A. Machine Learning

Machine Learning atau Pembelajaran Mesin merupakan teknik pendekatan dari Artificial Intelligent (AI) yang digunakan untuk menirukan hingga menggantikan peran manusia dalam melakukan aktivitas hingga memecahkan masalah. Secara singkat Machine Learning adalah mesin yang dibuat supaya dapat belajar dan melakukan pekerjaan tanpa arahan dari penggunanya. Menurut Arthur Samuel yakni seorang pelopor Amerika di bidang permainan komputer dan kecerdasan buatan menyatakan bahwa Machine Learning merupakan cabang ilmu yang mempelajari tentang bagaimana cara untuk memberikan kemampuan bagi komputer untuk belajar tanpa secara eksplisit diprogram. Sebuah Machine Learning dapat melakukannya apabila didasarkan pada ide yang didapat dari data sebelumnya dan mengidentifikasi pola serta membuat keputusan mengguguk sedikit campur tangan manusia atau penggunanya. Jadi, kemampuan Machine Learning tidak terbatas selagi ia mempelajarinya.

B. Supervised Learning

Supervised learning adalah paradigma pembelajaran mesin di mana algoritma atau model diajarkan untuk memahami dan membuat prediksi atau keputusan berdasarkan contoh data yang sudah diberi label. Dalam supervised learning, model diberikan dataset latihan yang berisi pasangan input dan output yang sesuai. Tugas utama model adalah menemukan hubungan atau pola yang ada di antara input dan output tersebut, sehingga ketika diberikan input baru yang belum pernah dilihat sebelumnya, model dapat menghasilkan output yang akurat atau mendekati output yang diinginkan. Tujuan akhir dari supervised learning adalah untuk membuat model yang mampu menggeneralisasi dari data pelatihan ke data baru, sehingga dapat digunakan untuk memprediksi atau mengklasifikasikan data yang tidak terlihat sebelumnya dengan tingkat akurasi yang tinggi. Model yang digunakan pada praktikum ini adalah, Linear Regression, K-Neighbors Regressor, Random Forest, Gradient Boosting Regressor dan Neural Networks. Hanya model Neural Networks yang akan diexport dan dibuat model dalam bentuk file.

C. Mean Squared Error

Mean Squared Error (MSE) adalah metrik evaluasi dalam supervised learning yang mengukur rata-rata dari kuadrat selisih antara nilai aktual dan nilai prediksi dari model. Untuk setiap titik data dalam dataset, selisih antara nilai sebenarnya (ground truth) dan nilai yang diprediksi oleh model dijumlahkan, kemudian hasilnya dikuadratkan. MSE kemudian dihitung sebagai rata-rata dari seluruh nilai kuadrat selisih tersebut. MSE memberikan gambaran tentang seberapa baik model dapat meminimalkan kesalahan prediksi secara global. Semakin kecil nilai MSE, semakin baik model dalam menghasilkan prediksi yang mendekati nilai sebenarnya. MSE sering digunakan sebagai fungsi objektif (loss function) dalam proses pelatihan model untuk mengarahkan model agar dapat mengoptimalkan prediksinya. Pada Praktikum ini, semua model akan diuji dengan Mean Squared Error.

D. Linear Regression

Regresi linear adalah metode dalam supervised learning yang digunakan untuk memodelkan hubungan linier antara satu atau lebih variabel independen (input) dengan variabel dependen (output) kontinu. Tujuannya adalah mencari garis atau hyperplane terbaik yang dapat

meminimalkan selisih antara nilai prediksi dan nilai sebenarnya. Model regresi linear dievaluasi berdasarkan seberapa baik garis atau hyperplane tersebut sesuai dengan data, dan metrik umum yang digunakan adalah Mean Squared Error (MSE).

E. K-Neighbors Regressor

K-Neighbors Regressor adalah algoritma dalam machine learning yang digunakan untuk tugas regresi. Algoritma ini beroperasi dengan cara menentukan nilai prediksi untuk suatu titik data berdasarkan rata-rata nilai target dari K tetangga terdekatnya dalam ruang fitur. Ketika suatu titik data baru perlu diprediksi, algoritma mengidentifikasi K tetangga terdekatnya dalam ruang fitur, dan nilai prediksi dihasilkan berdasarkan rata-rata nilai target dari tetangga-tetangga tersebut. K adalah parameter yang dapat diatur, dan pengguna dapat menyesuaikannya sesuai kebutuhan. K-Neighbors Regressor sering digunakan dalam kasus di mana hubungan antara fitur dan target bersifat lokal atau non-linier, dan algoritma ini memberikan pendekatan yang sederhana namun efektif dalam memodelkan pola pada data regresi.

F. Random Forest

Random Forest adalah algoritma machine learning yang termasuk dalam kategori ensemble learning. Ensemble learning menggunakan beberapa model untuk meningkatkan kinerja dan kestabilan prediksi. Dalam konteks Random Forest, model yang digunakan adalah pohon keputusan (decision tree). Algoritma ini bekerja dengan cara membangun banyak pohon keputusan secara acak dan menggabungkan hasil prediksi mereka untuk menghasilkan prediksi yang lebih akurat dan stabil.

G. Gradient Boosting Regressor

Gradient Boosting Regressor adalah algoritma machine learning yang juga termasuk dalam kategori ensemble learning. Algoritma ini bekerja dengan cara menggabungkan beberapa model lemah (weak learners), umumnya berupa pohon keputusan sederhana, untuk membentuk model yang lebih kuat. Proses ini dilakukan secara iteratif, dengan setiap iterasi berfokus pada mengoreksi kesalahan prediksi model sebelumnya. Pada setiap iterasi, model baru dibangun dengan memberikan lebih banyak "perhatian" pada data yang belum diprediksi dengan baik oleh model sebelumnya. Hal ini dilakukan dengan memperhitungkan gradien dari fungsi kerugian terhadap prediksi sebelumnya. Gradient Boosting Regressor secara bertahap meningkatkan kinerja prediksi dengan mengurangi kesalahan prediksi dari iterasi sebelumnya. Algoritma ini memiliki kemampuan mengatasi overfitting dan sering digunakan untuk tugas regresi pada berbagai jenis data.

H. Neural Networks

Neural Networks, atau jaringan saraf, adalah model komputasi terinspirasi dari struktur dan fungsi otak manusia. Mereka terdiri dari "neuron" buatan atau unit pemrosesan informasi, yang terhubung dalam lapisan-lapisan. Setiap koneksi antar neuron memiliki bobot yang memodifikasi sinyal. Neural Networks terdiri dari lapisan input, lapisan tersembunyi, dan lapisan output. Proses pelatihan melibatkan penyesuaian bobot agar model dapat menghasilkan prediksi yang akurat. Jenis umum dari Neural Networks adalah Artificial Neural Networks (ANN), yang terdiri dari input layer, hidden layer, dan output layer.

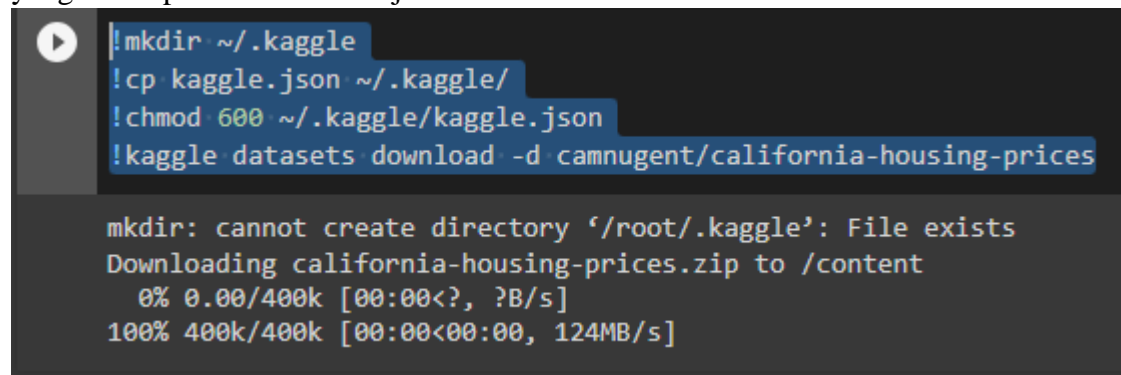
Pada Praktikum ini, Neural Networks sederhana akan dirancang dengan library Tensorflow, dengan input layer, layer dengan aktivasi relu, dan layer dengan aktivasi linear.

BAB III ANALISA DAN PERANCANGAN

Untuk Praktikum ini, digunakan google Collabs untuk mengambil data, mendesain model, dan merancang dan melatih model.

1. Struktur File

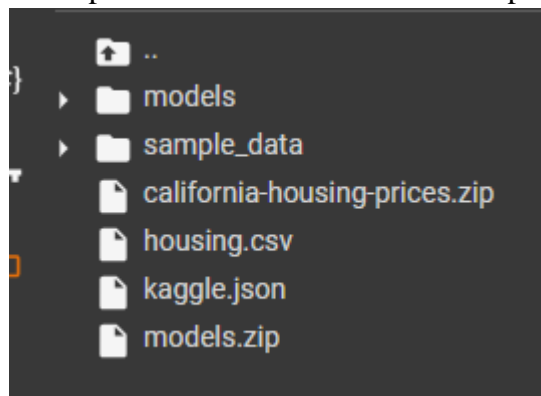
Dataset didapat dari kaggle dengan code berikut pada Google Collabs, atau bisa diambil dari .zip yang terlampir dalam folder ujian.



```
!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
!kaggle datasets download -d camnugent/california-housing-prices

mkdir: cannot create directory '/root/.kaggle': File exists
Downloading california-housing-prices.zip to /content
 0% 0.00/400k [00:00<?, ?B/s]
100% 400k/400k [00:00<00:00, 124MB/s]
```

Dataset tersebut kemudian di-unzip. Berikut adalah struktur folder pada file collab.



(Folder Sample data berasal dari Google Collab. Folder models boleh dikosongkan diawal penjalanan program.)

2. Mengeload, Membersihkan dan Mempersiapkan data

Pertama, data harus di load. Agar model tidak ter-bias oleh bagaimana data tersebut di sortir, (terkadang data tersebut telah disort sehingga dapat mempengaruhi pembuatan model) data akan di shuffle.

```
[17] pd_housing = pd.read_csv('/content/housing.csv')
pd_housing.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.5431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

Load data

```
[18] pd_housing_shuffled = pd_housing.sample(n=len(pd_housing), random_state=1)
pd_housing_shuffled.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
4712	-118.36	34.06	39.0	2610.0	670.0	1109.0	624.0	3.2500	355000.0	<1H OCEAN
2151	-119.78	36.78	37.0	2185.0	455.0	1143.0	438.0	1.9784	70700.0	INLAND
15927	-122.42	37.73	46.0	1819.0	411.0	1534.0	406.0	4.0132	229400.0	NEAR BAY
82	-122.28	37.81	52.0	340.0	97.0	200.0	87.0	1.5208	112500.0	NEAR BAY
8161	-118.13	33.82	37.0	1530.0	290.0	711.0	283.0	5.1795	225400.0	<1H OCEAN

Shuffle Data

Ada sebuah fitur yang bersifat non-numerik yaitu `ocean_proximity`. Sifat tersebut harus kita uraikan sehingga berbasis numerik agar bisa diterima oleh model untuk dilatih. Untuk itu, kita uraikan setiap enum dan menetapkan sebuah nilai boolean 0-1 untuk setiap sifat tersebut.

```
[19] pd.get_dummies(pd_housing_shuffled['ocean_proximity']).head()
```

	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN
4712	1	0	0	0	0
2151	0	1	0	0	0
15927	0	0	0	1	0
82	0	0	0	1	0
8161	1	0	0	0	0

Penguraian data `ocean_proximity` yang kategorikal menjadi numerik.

```
pd_housing_shuffled.drop('ocean_proximity', axis= 1).head()
```

Nilai kategorikal `ocean_proximity` pada data sebelumnya dihapus.

Konkatenasi ocean_proximity baru dengan data asli

```
pd_housing_final = pd.concat([pd_housing_shuffled.drop('ocean_proximity', axis=1), pd.get_dummies(pd_housing_shuffled['ocean_proximity'], axis=1)], axis=1)
pd_housing_final.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	NEAR OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN
4712	-118.36	34.06	39.0	2810.0	670.0	1100.0	634.0	3.2600	366000.0	1	0	0	0	0
2181	-119.70	26.70	37.0	2180.0	455.0	1143.0	436.0	1.9784	70700.0	0	1	0	0	0
16027	-122.42	37.73	46.0	1810.0	411.0	1554.0	496.0	4.0132	238400.0	0	0	0	1	0
82	-122.19	37.81	52.0	340.0	97.0	200.0	87.0	1.5208	112500.0	0	0	0	1	0
8161	-118.13	33.82	37.0	1550.0	290.0	711.0	283.0	6.1795	225400.0	1	0	0	0	0

Konkatenasi data utama dengan data ocean_proximity yang sudah bersifat numerik. Kemudian, ada record dalam dataset tersebut yang bersifat null. Record tersebut akan dihapus.

```
[25] pd_housing_final = pd_housing_final.dropna()
len(pd_housing_final)

20433
```

3. Training

Pertama, sebelum melatih model, data harus dipisah menjadi training set, validation set, dan test set. Validation set berfungsi untuk menguji model yang sudah dilatih di training set seberapa baik model tersebut dalam mengeneralisir data baru.

Training setup. Memisahkan X dan y ke dalam variabel masing masing sesuai dengan setnya.

```
X_train, y_train = train_pd.to_numpy()[ :, :-1], train_pd.to_numpy()[ :, -1]
X_val, y_val = val_pd.to_numpy()[ :, :-1], val_pd.to_numpy()[ :, -1]
X_test, y_test = test_pd.to_numpy()[ :, :-1], test_pd.to_numpy()[ :, -1]
X_test.shape
```

(1215, 13)

Lalu, dapat dilihat nilai-nilai dari dataset memiliki range yang sangat berbeda. Misal, total_room dengan median_house_age memiliki range yang sangat berbeda, sehingga pada model dapat memberikan pengaruh yang berbeda pula. Untuk mengatasi hal ini, dataset akan diskalakan, sebagai bagian dari preprocessing.

```

[40] from sklearn.preprocessing import StandardScaler
import numpy as np

scaler = StandardScaler().fit(X_train[:, :8])
def preprocessor(X):
    A = np.copy(X)
    A[:, :8] = scaler.transform(A[:, :8])
    return A

X_train_preprocessed = preprocessor(X_train)
X_train, X_val, X_test = preprocessor(X_train), preprocessor(X_val), preprocessor(X_test)

```

Kemudian ,training berbagai model dapat dimulai.

A. Linear Regression

```

[42] from sklearn.metrics import mean_squared_error as mse
from sklearn.linear_model import LinearRegression

lm = LinearRegression().fit(X_train, y_train)
mse(lm.predict(X_train), y_train, squared=False), mse(lm.predict(X_val), y_val, squared=False) #mse dari training set dan validation set.

(68593.05578127236, 71382.43558330165)

```

Kode diatas adalah kegiatan fitting model Linear Regression. Digunakan kelas LinearRegression dari sklearn. MSE adalah kelas yang menghitung mean squared error dari model yang dibuat. Didapat mse dari training set sekitar 68 ribu, dan pada validation set 71 ribu. Error tersebut cukup tinggi, namun overfitting cukup rendah.

B. K-Neighbors Regressor

```

[43] from sklearn.neighbors import KNeighborsRegressor
knn = KNeighborsRegressor(n_neighbors =20).fit(X_train, y_train)
mse(knn.predict(X_train), y_train, squared=False), mse(knn.predict(X_val), y_val, squared=False) #mse dari training set dan validation set.

(57882.52368291895, 62789.015671624795)

```

Kode diatas, dengan struktur yang sama dengan kode Linear Regression, membuat model dengan KNN. Bisa dilihat error yang didapat dengan model ini lebih sedikit, namun training set dan validation set memiliki perbedaan yang cukup jauh. Ini menunjukkan adanya overfitting, yaitu ketika model dilatih dalam training set bekerja dengan baik namun gagal mengeneralisir data baru. Model ini bisa dipermainkan pada parameter `n_neighbors`. Awalnya, dicoba dengan nilai yang rendah, dan menghasilkan error yang cukup tinggi di kedua set.

C. Random Forest

```

[44] from sklearn.ensemble import RandomForestRegressor

rfr = RandomForestRegressor(max_depth=10).fit(X_train, y_train)
mse(rfr.predict(X_train), y_train, squared=False), mse(rfr.predict(X_val), y_val, squared=False) #mse dari training set dan validation set.

(43519.5867513588, 53834.38936887767)

```

Dengan random forest, didapat error yang lebih sedikit lagi. Namun, tetap terjadi overfitting. Nilai `max_depth` dapat disesuaikan agar error dapat dikurangi, namun dengan nilai yang besar durasi training pun akan bertambah juga.

D. Gradient Boost Regressor

```
[57] from sklearn.ensemble import GradientBoostingRegressor

gbr = GradientBoostingRegressor(n_estimators=250).fit(X_train, y_train)
mse(gbr.predict(X_train), y_train, squared=False), mse(gbr.predict(X_val), y_val, squared=False) #mse dari training set dan validation set.
```

(47274.82259072157, 51346.51283839868)

Dengan GBR, error didapat cukup rendah relatif dibandingkan model lain, dan performa pada kedua set tidak jauh beda.

E. Neural Network

```
[61] from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import *
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.metrics import RootMeanSquaredError
from tensorflow.keras.optimizers import Adam

simple_nn = Sequential()
simple_nn.add(InputLayer((13,)))
simple_nn.add(Dense(2, 'relu'))
simple_nn.add(Dense(1, 'linear'))

opt = Adam(learning_rate=.1)
cp = ModelCheckpoint('models/simple_nn', save_best_only=True)
simple_nn.compile(optimizer=opt, loss='mse', metrics=[RootMeanSquaredError()])

simple_nn.fit(x=X_train, y=y_train, validation_data = (X_val, y_val), callbacks=[cp], epochs=100)
```

Kode diatas mengimplementasikan pembuatan Model Neural Network. Sequential adalah model dari NN yang akan dibuat. Library layers terdiri dari layer-layer yang diperlukan dalam neural network. Model Checkpoint memberi kemampuan untuk memberi sebuah “Checkpoint” setelah menjalankan sebuah training terhadap seluruh datum. Library RMSE menghitung error MSE namun diakar. Adam adalah kelas untuk optimisasi parameter-parameter.

Pada assignment variable opt, dispesifikasikan learning_rate sebagai perubahan yang dapat dilakukan pada hubungan-hubungan tiap neuron setiap setelah checkpoint. Akan dilihat bahwa learning rate yang tinggi akan memberi waktu training yang cepat, namun akurasi yang rendah, sedangkan learning rate terlalu rendah akan menyebabkan training time yang rendah dengan perubahan error yang tidak banyak.

```

Epoch 87/100
563/563 [=====] - 1s 2ms/step - loss: 50316922880.0000 - root_mean_squared_error: 224314.3438 - va
Epoch 88/100
563/563 [=====] - 1s 2ms/step - loss: 50196054016.0000 - root_mean_squared_error: 224044.7656 - va
Epoch 89/100
563/563 [=====] - 1s 2ms/step - loss: 50074071040.0000 - root_mean_squared_error: 223772.3594 - va
Epoch 90/100
563/563 [=====] - 2s 3ms/step - loss: 49950973952.0000 - root_mean_squared_error: 223497.1406 - va
Epoch 91/100
563/563 [=====] - 1s 2ms/step - loss: 49826885632.0000 - root_mean_squared_error: 223219.3594 - va
Epoch 92/100
563/563 [=====] - 2s 3ms/step - loss: 49701888000.0000 - root_mean_squared_error: 222939.2031 - va
Epoch 93/100
563/563 [=====] - 1s 2ms/step - loss: 49575763968.0000 - root_mean_squared_error: 222656.1562 - va
Epoch 94/100
563/563 [=====] - 1s 2ms/step - loss: 49448534016.0000 - root_mean_squared_error: 222370.2656 - va
Epoch 95/100
563/563 [=====] - 1s 2ms/step - loss: 49320202240.0000 - root_mean_squared_error: 222081.5156 - va
Epoch 96/100
563/563 [=====] - 1s 2ms/step - loss: 49190907904.0000 - root_mean_squared_error: 221790.2344 - va
Epoch 97/100
563/563 [=====] - 1s 2ms/step - loss: 49060745216.0000 - root_mean_squared_error: 221496.6094 - va
Epoch 98/100
563/563 [=====] - 1s 2ms/step - loss: 48929476608.0000 - root_mean_squared_error: 221200.0781 - va
Epoch 99/100
563/563 [=====] - 1s 3ms/step - loss: 48797261824.0000 - root_mean_squared_error: 220901.0156 - va
Epoch 100/100
563/563 [=====] - 1s 3ms/step - loss: 48663879680.0000 - root_mean_squared_error: 220598.9062 - va
<keras.src.callbacks.History at 0x7a54902ae560>

```

Training Neural Network dengan Epoch 100, learning rate 0.001. Dapat dilihat error dari satu epoch ke epoch lainnya tidak berubah banyak.

```

Epoch 87/100
563/563 [=====] - 1s 1ms/step - loss: 4413952640.0000 - root_mean_squared_error: 66462.7266 - val_loss: 4708392128.0000 - val_root_mean_squared_error: 68690.7890
Epoch 88/100
563/563 [=====] - 1s 1ms/step - loss: 44139900432.0000 - root_mean_squared_error: 66476.2422 - val_loss: 4741308544.0000 - val_root_mean_squared_error: 68806.7811
Epoch 89/100
563/563 [=====] - 1s 1ms/step - loss: 4417533832.0000 - root_mean_squared_error: 66464.4922 - val_loss: 4737978368.0000 - val_root_mean_squared_error: 68832.9766
Epoch 90/100
563/563 [=====] - 1s 1ms/step - loss: 4418810640.0000 - root_mean_squared_error: 66459.1328 - val_loss: 4741090720.0000 - val_root_mean_squared_error: 68839.4821
Epoch 91/100
563/563 [=====] - 2s 3ms/step - loss: 4418146784.0000 - root_mean_squared_error: 66454.8538 - val_loss: 473803104.0000 - val_root_mean_squared_error: 68772.7056
Epoch 92/100
563/563 [=====] - 1s 2ms/step - loss: 4413131408.0000 - root_mean_squared_error: 66447.8281 - val_loss: 4731747008.0000 - val_root_mean_squared_error: 68845.8281
Epoch 93/100
563/563 [=====] - 1s 2ms/step - loss: 4413100736.0000 - root_mean_squared_error: 66432.6797 - val_loss: 4742091264.0000 - val_root_mean_squared_error: 68813.7631
Epoch 94/100
563/563 [=====] - 2s 3ms/step - loss: 441280400.0000 - root_mean_squared_error: 66420.5234 - val_loss: 4704062336.0000 - val_root_mean_squared_error: 68096.1016
Epoch 95/100
563/563 [=====] - 2s 3ms/step - loss: 4412301472.0000 - root_mean_squared_error: 66425.8538 - val_loss: 4758383104.0000 - val_root_mean_squared_error: 68921.0234
Epoch 96/100
563/563 [=====] - 2s 3ms/step - loss: 4409670920.0000 - root_mean_squared_error: 66409.9530 - val_loss: 4704083392.0000 - val_root_mean_squared_error: 68049.7188
Epoch 97/100
563/563 [=====] - 1s 2ms/step - loss: 4408487424.0000 - root_mean_squared_error: 66360.4375 - val_loss: 4711048320.0000 - val_root_mean_squared_error: 68041.1162
Epoch 98/100
563/563 [=====] - 1s 1ms/step - loss: 4405690880.0000 - root_mean_squared_error: 66405.3011 - val_loss: 4740048064.0000 - val_root_mean_squared_error: 68848.0131
Epoch 99/100
563/563 [=====] - 1s 1ms/step - loss: 4408134016.0000 - root_mean_squared_error: 66386.7031 - val_loss: 4741061330.0000 - val_root_mean_squared_error: 68831.5016
Epoch 100/100
563/563 [=====] - 1s 1ms/step - loss: 4407178240.0000 - root_mean_squared_error: 66388.5781 - val_loss: 4721081600.0000 - val_root_mean_squared_error: 68730.2266
Epoch 87/100
563/563 [=====] - 1s 2ms/step - loss: 4405706400.0000 - root_mean_squared_error: 66376.0038 - val_loss: 4735922176.0000 - val_root_mean_squared_error: 68818.0188
Epoch 88/100
563/563 [=====] - 1s 3ms/step - loss: 4406904448.0000 - root_mean_squared_error: 66369.4511 - val_loss: 4741428336.0000 - val_root_mean_squared_error: 68858.0112
Epoch 89/100
563/563 [=====] - 1s 3ms/step - loss: 4404471184.0000 - root_mean_squared_error: 66360.2011 - val_loss: 4736213984.0000 - val_root_mean_squared_error: 68708.2656
Epoch 90/100
563/563 [=====] - 1s 2ms/step - loss: 4403705344.0000 - root_mean_squared_error: 66360.4219 - val_loss: 4721338624.0000 - val_root_mean_squared_error: 68743.1016
Epoch 91/100
563/563 [=====] - 1s 2ms/step - loss: 4401506640.0000 - root_mean_squared_error: 66344.4375 - val_loss: 4727970304.0000 - val_root_mean_squared_error: 68790.2344
<keras.src.callbacks.History at 0x7a540340c100>

```

Training Neural Network dengan Epoch 100, learning rate 0.1. Dapat dilihat error dari satu epoch ke epoch lainnya berubah lebih banyak daripada sebelumnya.

4. Performa Model

Model NN dihitung MSE-nya.

```
from tensorflow.keras.models import load_model

simple_nn = load_model('models/simple_nn')
mse(simple_nn.predict(X_train), y_train, squared=False), mse(simple_nn.predict(X_val), y_val, squared=False) #mse dari training set dan validation set.
```

563/563 [=====] - 1s 3ms/step
19/19 [=====] - 0s 1ms/step
(06615.74445199296, 88641.16822682459)

BAB IV PENUTUP

4.1 Kesimpulan

Setelah menguji MSE dari beberapa model, termasuk Linear Regression, Random Forest, K Nearest Neighbors, dan Neural Network, Gradient Boosting menonjol sebagai model yang paling efektif untuk memprediksi harga perumahan di California. Model Gradient Boosting memberikan nilai error terendah, menandakan akurasi prediksi yang tinggi relatif dari model lainnya. Lebih lanjut, observasi bahwa tidak terjadi overfitting menunjukkan bahwa model Gradient Boosting mampu memgeneralisasi dengan baik pada data yang belum pernah dilihat sebelumnya. Ini menunjukkan bahwa pendekatan ensemble learning dengan Gradient Boosting adalah pilihan yang baik untuk tugas prediksi ini.

4.2 Saran

Ada banyak cara mengurangi error dan menambahkan akurasi dari model yang dibuat, dari mencari fitur-fitur baru, tweaking dari parameter model-model, dan lain-lain.