

Mata Kuliah Aplikasi Data Scientist

# Laporan Tugas P4: Penerapan Numpy

Dosen Pengampu: Ledy Elsera Astrianty, S.Kom., M.Kom.



**Disusun oleh:**

- Lathif Ramadhan (5231811022)
- Andini Angel M. (5231811029)
- Rama Panji N. (5231811033)
- Giffari Riyanda P. (5231811036)

**PROGRAM STUDI SAINS DATA**

**FAKULTAS SAINS DAN TEKNOLOGI**

**UNIVERSITAS TEKNOLOGI YOGYAKARTA**

**YOGYAKARTA**

**2025**

## Deskripsi Dataset

Dataset Diamonds adalah kumpulan data yang berisi informasi tentang atribut-atribut berlian dan harganya. Dataset ini digunakan untuk memahami hubungan antara karakteristik fisik berlian (seperti berat, kualitas potongan, warna, kejernihan, dan dimensi) dengan harga yang ditetapkan untuk berlian tersebut.

Dataset ini sering digunakan dalam analisis data, visualisasi data, dan pembelajaran mesin untuk:

- Mempelajari faktor-faktor apa saja yang memengaruhi harga berlian.
- Membangun model prediktif untuk memperkirakan harga berlian berdasarkan atributnya.
- Menganalisis distribusi dan pola data untuk memahami pasar berlian.

Dataset Diamonds terdiri dari **53.940 baris** (observasi) dan **10 kolom** (variabel). Setiap baris mewakili satu berlian, dan kolom-kolomnya menggambarkan atribut-atribut berlian tersebut.

**Link Sumber Dataset:** <http://kaggle.com/datasets/shivam2503/diamonds>

### Kolom-Kolom dalam Dataset

Berikut adalah penjelasan detail tentang setiap kolom dalam dataset:

1. **carat**
  - **Tipe Data:** Numerik (float)
  - **Deskripsi:** Berat berlian dalam satuan carat (1 carat = 0,2 gram).
  - **Contoh Nilai:** 0.23, 1.52, 2.01
2. **cut**
  - **Tipe Data:** Kategorikal (object)
  - **Deskripsi:** Kualitas potongan berlian. Potongan menentukan seberapa baik berlian memantulkan cahaya.
  - **Kategori:**
    - Fair
    - Good
    - Very Good
    - Premium
    - Ideal
3. **color**
  - **Tipe Data:** Kategorikal (object)
  - **Deskripsi:** Warna berlian. Warna dinilai dari D (terbaik, tidak berwarna) hingga J (sedikit berwarna).
  - **Kategori:** D, E, F, G, H, I, J
4. **clarity**
  - **Tipe Data:** Kategorikal (object)

- **Deskripsi:** Kejernihan berlian. Kejernihan mengukur seberapa bersih berlian dari inklusi (cacat internal) dan bekas luka.
- **Kategori:**
  - I1 (termasuk inklusi yang terlihat)
  - SI2 (inklusi kecil yang terlihat)
  - SI1 (inklusi kecil yang sulit terlihat)
  - VS2 (inklusi sangat kecil yang sulit terlihat)
  - VS1 (inklusi sangat kecil yang sangat sulit terlihat)
  - VVS2 (inklusi sangat sangat kecil)
  - VVS1 (inklusi sangat sangat kecil)
  - IF (tidak ada inklusi, internal flawless)

#### 5. **depth**

- **Tipe Data:** Numerik (float)
- **Deskripsi:** Persentase kedalaman berlian. Dihitung sebagai  $z / \text{mean}(x, y) * 100$ , di mana z adalah kedalaman dan x, y adalah panjang dan lebar.
- **Contoh Nilai:** 61.5, 62.3, 59.8

#### 6. **table**

- **Tipe Data:** Numerik (float)
- **Deskripsi:** Lebar bagian atas berlian relatif terhadap titik terlebar.
- **Contoh Nilai:** 55.0, 60.0, 65.0

#### 7. **price**

- **Tipe Data:** Numerik (integer)
- **Deskripsi:** Harga berlian dalam dolar AS. Ini adalah target variabel (variabel dependen) dalam banyak kasus analisis prediktif.
- **Contoh Nilai:** 326, 1000, 15000

#### 8. **x**

- **Tipe Data:** Numerik (float)
- **Deskripsi:** Panjang berlian dalam milimeter.
- **Contoh Nilai:** 4.01, 5.62, 6.50

#### 9. **y**

- **Tipe Data:** Numerik (float)
- **Deskripsi:** Lebar berlian dalam milimeter.
- **Contoh Nilai:** 4.00, 5.60, 6.45

#### 10. **z**

- **Tipe Data:** Numerik (float)
- **Deskripsi:** Kedalaman berlian dalam milimeter.
- **Contoh Nilai:** 2.43, 3.52, 4.02

# SCRIPT DAN PENJELASAN NUMPY

Link Proyek Colab:

<https://colab.research.google.com/drive/17aTcQnxAKs4pAvUO3HVw3UaU6WggECwm?usp=sharing>

## 1. Mengimport Library

```
import numpy as np
import pandas as pd

print(np.__version__)
print(pd.__version__)
```

1.26.4  
2.2.2

- **NumPy** digunakan untuk perhitungan numerik.
- **Pandas** digunakan untuk membaca dataset dan mengelola data dalam format tabel.
- `print(np.__version__)` digunakan untuk memeriksa versi **NumPy**, karena beberapa fitur mungkin berbeda di setiap versi.

## 2. Load Dataset

```
[ ] df = pd.read_csv('https://raw.githubusercontent.com/LatiefDataVisionary/data-science-application-college-task/refs/heads/main/datasets/diamonds.csv')
```

## 3. Melihat Informasi Dasar Dataset (Pandas)

```
[ ] df.head() # menampilkan 5 baris pertama
```

	Unnamed: 0	carat	cut	color	clarity	depth	table	price	x	y	z
0	1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

- Dataset diunduh menggunakan `pd.read_csv()`.
- `df.head()` menampilkan **5 baris pertama dataset** agar kita bisa melihat bagaimana data disusun.

```
[ ] df.info() # Menampilkan informasi dataset
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 11 columns):
#   Column      Non-Null Count  Dtype
---  -
0   Unnamed: 0   53940 non-null  int64
1   carat        53940 non-null  float64
2   cut          53940 non-null  object
3   color        53940 non-null  object
4   clarity      53940 non-null  object
5   depth        53940 non-null  float64
6   table        53940 non-null  float64
7   price        53940 non-null  int64
8   x            53940 non-null  float64
9   y            53940 non-null  float64
10  z            53940 non-null  float64
dtypes: float64(6), int64(2), object(3)
memory usage: 4.5+ MB
```

```
[ ] df.describe() # Menampilkan statistik deskriptif
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 11 columns):
#   Column      Non-Null Count  Dtype
---  -
0   Unnamed: 0   53940 non-null  int64
1   carat        53940 non-null  float64
2   cut          53940 non-null  object
3   color        53940 non-null  object
4   clarity      53940 non-null  object
5   depth        53940 non-null  float64
6   table        53940 non-null  float64
7   price        53940 non-null  int64
8   x            53940 non-null  float64
9   y            53940 non-null  float64
10  z            53940 non-null  float64
dtypes: float64(6), int64(2), object(3)
memory usage: 4.5+ MB
```

	Unnamed: 0	carat	depth	table	price	x	y	z
count	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000
mean	26970.500000	0.797940	61.749405	57.457184	3932.799722	5.731157	5.734526	3.538734
std	15571.281097	0.474011	1.432621	2.234491	3989.439738	1.121761	1.142135	0.705699
min	1.000000	0.200000	43.000000	43.000000	326.000000	0.000000	0.000000	0.000000
25%	13485.750000	0.400000	61.000000	56.000000	950.000000	4.710000	4.720000	2.910000
50%	26970.500000	0.700000	61.800000	57.000000	2401.000000	5.700000	5.710000	3.530000
75%	40455.250000	1.040000	62.500000	59.000000	5324.250000	6.540000	6.540000	4.040000
max	53940.000000	5.010000	79.000000	95.000000	18823.000000	10.740000	58.900000	31.800000

- `df.info()` membantu kita mengetahui **jumlah data, tipe data di setiap kolom, dan apakah ada data kosong**.
- `df.describe()` memberikan **ringkasan statistik** seperti **mean, standar deviasi, nilai minimum, dan maksimum** untuk setiap kolom numerik.

## 4. Menggunakan NumPy untuk Analisis Numerik

```
[ ] # Mengambil kolom 'price' sebagai array NumPy
    prices = df['price'].values

    # Menghitung rata-rata harga
    mean_price = np.mean(prices)
    print(f'Rata-rata harga: ${float(mean_price):.2f}')
```

⇒ Rata-rata harga: \$3932.80

```
[ ] # Menghitung median harga
    median_price = np.median(prices)
    print(f'Median harga: ${float(median_price):.2f}')
```

⇒ Median harga: \$2401.00

```
[ ] # Menghitung standar deviasi harga
    std_price = np.std(prices)
    print(f"Standar deviasi harga: {float(std_price):.2f}")
```

⇒ Standar deviasi harga: 3989.40

```
[ ] # Menghitung nilai maksimum dan minimum harga
    max_price = np.max(prices)
    min_price = np.min(prices)

    print(f'Harga maksimum: ${max_price:.2f}')
    print(f'Harga minimum: ${min_price:.2f}')
```

⇒ Harga maksimum: \$18823.00  
Harga minimum: \$326.00

- **Mean:** Nilai rata-rata dari harga berlian.
- **Median:** Nilai tengah dari data harga berlian (lebih tahan terhadap outlier).
- **Standar Deviasi:** Mengukur **seberapa jauh harga berlian tersebar dari nilai rata-rata**.
- **Maksimum & Minimum:** Harga tertinggi dan terendah dari berlian dalam dataset.

```
[ ] # Menghitung korelasi antara 'carat' dan 'price'
    carat = df['carat'].values
    correlation_carat_price = np.corrcoef(carat, prices)[0, 1]
    print(f'Korelasi antara carat dan price: {correlation_carat_price:.2f}')
```

➡ Korelasi antara carat dan price: 0.92

```
[ ] # Menghitung nilai maksimum dan minimum carat
    max_carat = np.max(carat)
    min_carat = np.min(carat)

    print(f'Carat maksimum: {max_carat:.2f}')
    print(f'Carat minimum: {min_carat:.2f}')
```

➡ Carat maksimum: 5.01  
Carat minimum: 0.20

- **Korelasi antara berat (carat) dan harga (price) dihitung menggunakan `np.corrcoef()`.**
- **Jika nilainya positif mendekati 1, berarti semakin besar berat berlian, semakin mahal harganya.**

### Filtering Data dengan Numpy

```
[ ] # Filter Harga diatas $10.000
    high_prices = prices[prices > 10000]
    print(f'Jumlah berlian dengan harga diatas $10.000: {len(high_prices)}')
```

➡ Jumlah berlian dengan harga diatas \$10.000: 5222

```
[ ] # Filter harga dibawah $1.000
    low_prices = prices[prices > 1000]
    print(f'Jumlah berlian dengan harga dibawah $1.000: {len(low_prices)}')
```

➡ Jumlah berlian dengan harga dibawah \$1.000: 39416

```
[ ] # Filter harga diantara $1.000-$5.000
    prices_1k_to_5k = prices[(prices >= 1000) & (prices <= 5000)]
    print(f'Jumlah berlian dengan harga diantara $1.000-$5.000: {len(prices_1k_to_5k)}')
```

➡ Jumlah berlian dengan harga diantara \$1.000-\$5.000: 24727

```
[ ] # Filter harga diantara $5.000-$10.000
    prices_1k_to_5k = prices[(prices > 5000) & (prices <= 10000)]
    print(f'Jumlah berlian dengan harga diantara $5.000-$10.000: {len(prices_1k_to_5k)}')
```

➡ Jumlah berlian dengan harga diantara \$5.000-\$10.000: 9492

```
[ ] # Filter carat diatas 1.5
    high_carats = carat[carat > 1.5]
    print(f'Jumlah berlian dengan carat diatas 1.5: {len(high_carats)}')
```

➡ Jumlah berlian dengan carat diatas 1.5: 5442

```
[ ] # Filter carat dibawah 1.5
    low_carats = carat[carat < 1.5]
    print(f'Jumlah berlian dengan carat dibawah 1.5: {len(low_carats)}')
```

➡ Jumlah berlian dengan carat dibawah 1.5: 47705

- Memfilter harga berlian yang lebih dari \$10.000 dan kurang dari \$1.000 menggunakan NumPy.

## Filtering Data dengan Pandas

```
[ ] # Filter berlian dengan kualitas cut 'Ideal'
ideal_diamonds = df[df['cut'] == 'Ideal']
ideal_diamonds.head()
```



	Unnamed: 0	carat	cut	color	clarity	depth	table	price	x	y	z
0	1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
11	12	0.23	Ideal	J	VS1	62.8	56.0	340	3.93	3.90	2.46
13	14	0.31	Ideal	J	SI2	62.2	54.0	344	4.35	4.37	2.71
16	17	0.30	Ideal	I	SI2	62.0	54.0	348	4.31	4.34	2.68
39	40	0.33	Ideal	I	SI2	61.8	55.0	403	4.49	4.51	2.78

```
[ ] # Filter berlian dengan harga diatas $10.000
expensive_diamonds = df[df['price'] > 10000]
expensive_diamonds.head()
```



	Unnamed: 0	carat	cut	color	clarity	depth	table	price	x	y	z
21928	21929	1.70	Ideal	J	VS2	60.5	58.0	10002	7.73	7.74	4.68
21929	21930	1.03	Ideal	E	VVS2	60.6	59.0	10003	6.50	6.53	3.95
21930	21931	1.23	Very Good	G	VVS2	60.6	55.0	10004	6.93	7.02	4.23
21931	21932	1.25	Ideal	F	VS2	61.6	55.0	10006	6.93	6.96	4.28
21932	21933	2.01	Very Good	I	SI2	61.4	63.0	10009	8.19	7.96	4.96

- Menggunakan Pandas untuk memilih **berlian dengan kualitas potongan "Ideal" dan harga di atas \$10.000.**



## 6. Menghitung Persentil dengan Numpy

```
[ ] # Menghitung percentile untuk harga
percentile_25 = np.percentile(prices, 25)
percentile_50 = np.percentile(prices, 50) # median
percentile_75 = np.percentile(prices, 75)

print(f'Percentile 25: ${percentile_25:.2f}')
print(f'Percentile 50 (Median): ${percentile_50:.2f}')
print(f'Percentile 75: ${percentile_75:.2f}')
```

```
➡ Percentile 25: $950.00
Percentile 50 (Median): $2401.00
Percentile 75: $5324.25
```

```
[ ] # Menghitung percentile untuk carat
percentile_25 = np.percentile(carat, 25)
percentile_50 = np.percentile(carat, 50) # median
percentile_75 = np.percentile(carat, 75)

print(f'Percentile 25: {percentile_25:.2f}')
print(f'Percentile 50 (Median): {percentile_50:.2f}')
print(f'Percentile 75: {percentile_75:.2f}')
```

```
➡ Percentile 25: 0.40
Percentile 50 (Median): 0.70
Percentile 75: 1.04
```

- Persentil menunjukkan distribusi harga berlian dalam kelompok 25%, 50%, dan 75%.
- Persentil 50% sama dengan median harga.

## 7. Membuat Histogram dengan NumPy

```
[ ] # Membuat histogram untuk harga
    hist, bin_edges = np.histogram(prices, bins=10)
    print('Frekuensi Histogram')
    print(hist)
    print('Batas bin Histogram')
    print(bin_edges)
```

```
⇒ Frekuensi Histogram
[25335  9328  7393  3878  2364  1745  1306  1002   863   726]
Batas bin Histogram
[ 326.    2175.7  4025.4  5875.1  7724.8  9574.5 11424.2 13273.9 15123.6
 16973.3 18823. ]
```

- Membuat histogram untuk melihat **distribusi harga berlian** dalam 10 kelompok.

## 9. Operasi matematika dengan Numpy

```
[ ] # Menghitung total harga semua berlian
    total_prices = np.sum(prices)
    print(f'Tootal harga semua berlian: ${total_prices:.2f}')
```

```
⇒ Tootal harga semua berlian: $212135217.00
```

- Menghitung **total harga dari seluruh berlian dalam dataset**.

## 10. Menghitung Z-Scores dengan Numpy

```
# Menyimpan data numerik ke numpy array
numeric_data = df.select_dtypes(include='number').iloc[:,1:].to_numpy()
# Menyimpan kolom data numerik ke numpy array
col_numeric = df.select_dtypes(include='number').iloc[:,1:].columns.to_list()

# Menghitung mean setiap kolom numerik
mean = np.mean(numeric_data, axis=0)
# Menghitung standar deviasi setiap kolom numerik
std = np.std(numeric_data, axis=0)

# Menghitung Z-Scores setiap kolom numerik
zscore_num = (numeric_data - mean) / std

# Ngeprint Z-Value
print(f"5 Value awal dari Z-Value Dari Kolom: \n{col_numeric} : \n{zscore_num[:5,:8]}")
```

```
5 Value awal dari Z-Value Dari Kolom:
['carat', 'depth', 'table', 'price', 'x', 'y', 'z'] :
[[-1.19816781 -0.17409151 -1.09967199 -0.90409516 -1.58783745 -1.53619556
  -1.57112919]
 [-1.24036129 -1.36073849  1.58552871 -0.90409516 -1.64132529 -1.65877419
  -1.74117497]
 [-1.19816781 -3.38501862  3.37566251 -0.9038445  -1.49869105 -1.45739502
  -1.74117497]
 [-1.07158736  0.45413336  0.24292836 -0.90208985 -1.36497146 -1.31730516
  -1.28771955]
 [-1.02939387  1.08235823  0.24292836 -0.90183918 -1.24016651 -1.21223777
  -1.11767377]]
```

- Menghitung Z-Score untuk tiap kolom dalam dataset, di mana masing-masing kolom mewakili fitur dari dataset asli dan setiap baris menggambarkan satu entri data.

## 10. Menghitung jumlah Outlier menggunakan Z-Scores dengan Numpy

```
# Mencari outlier
outlier = ( zscore_num > 3) | ( zscore_num < -3)

# Menghitung total outlier perkolom
total_z_kolom = np.sum(outlier, axis=0)

# Menghitung berapa outlier untuk seluruh kolom
total_z = np.unique(outlier, return_counts=True)[1][1]

# Menprint total Outlier tiap kolom
print(f"Total Outlier tiap kolom numerik : ")
for index, col in enumerate(col_numeric):
    print(f"{index + 1}. {col} = {total_z_kolom[index]}")
# Menprint total Outlier untuk semua kolom
print(f"\nJumlah Outlier di seluruh kolom adalah : {total_z}")
```

Total Outlier tiap kolom numerik :

1. carat = 439
2. depth = 685
3. table = 336
4. price = 1206
5. x = 43
6. y = 34
7. z = 55

Jumlah Outlier di seluruh kolom adalah : 2798

- Mengidentifikasi outlier pada tiap kolom dan menghitung jumlah total outlier dalam dataset dengan menggunakan Z-Score yang telah dihitung, dengan kriteria nilai di atas 3 atau di bawah -3 dianggap sebagai outlier.

## 11. Membuat Model Linear Regresi Menggunakan Numpy

```
▶ # Model Linear Regressi
class LinearRegression:
    def __init__(self, X, y=None, lr=0.01):
        # Params untuk menyimpan parameter dan gradientnya
        self.params = {}

        # inisiasi parameter
        # Menambah col baru dengan semua isinya bernilai 1 ke x
        # yang digunakan agar kita memiliki intercept di rumus linear regresi
        # yang menggunakan vector form
        self.X = np.concatenate((np.ones([np.shape(X)[0], 1]), X), axis=1)
        self.y = y
        # learning rate sebagai konstanta belajar seberapa cepat mengupdate gradient
        self.lr = lr
        # Inisiasi parameter linear regresi dengan menggunakan nilai random
        self.params['W'] = np.random.rand(1, self.X.shape[1])

    def predict(self, X, mode="test"):
        """Untuk Prediksi y Berdasarkan input X menggunakan model linear regresi"""
        if mode == "test":
            # Menambah col baru dengan semua isinya bernilai 1 ke x
            # yang digunakan agar kita memiliki intercept di rumus linear regresi
            # yang menggunakan vector form
            X = np.concatenate((np.ones([np.shape(X)[0], 1]), X), axis=1)
            # Memprediksi y berdasarkan X menggunakan model yang sudah dibuat
            output = np.dot(X, self.params["W"].T)
            return output

    def loss(self, output, y):
```

```

    """Menghitung loss"""
    # Menghitung loss menggunakan mean squared error dan dikali 1/2 agar
    # gradientnya nanti gampang dihitung
    return 1/2 * np.mean(np.square(output - y))

def batch(self, batch_size):
    """Menggunakan mini-batch untuk menghitung loss, output, dan gradien"""
    # Meninisiasi loss dan menshuffle X & y
    loss = 0
    # Menshuffle menggunakan fungsi np.random.permutation() sebagai index
    # lalu menggunakan index tersebut untuk menshuffle X dan y
    indices = np.random.permutation(len(self.y))
    X_shuffle = self.X[indices]
    y_shuffle = self.y[indices]

    # Membagi data menjadi minibatch yg diitung loss nya
    for i in range(0, len(self.y), batch_size):
        end = min(i + batch_size, len(self.y))
        # Membagi data per batch dengan banyak n (training example) sesuai
        # dengan batch_size yang dipilih
        X_batch = X_shuffle[i:end, :]
        y_batch = y_shuffle[i:end]
        # Kalulasi loss per batch
        a = self.calculate(X_batch, y_batch, "train", "gradient_descent")[0]

        # Melakukan running average dari loss sebelumnya agar mendapatkan rata
        # rata loss dari mini batch
        loss = loss + a * (end - i)
    # Return total loss dari loss yang didapatkan menggunakan running average
    # dan dibagi total training (n)
    return loss / len(self.y)

```

```

def calculate(self, X, y, mode="test", solver_type="least_square"):
    """Untuk kalkulasi gradien output dan least square dari model"""
    # Mendapatkan output dari linear regresi
    output = self.predict(X, mode=mode)
    # Bila modenya train
    if mode == "train":
        # Training menggunakan gradient descent
        # Menghitung loss menggunakan output dan y
        loss = self.loss(output, y)

        # Menentukan solver type gradient descent atau metode least square
        if solver_type == "gradient_descent":
            # Menghitung gradient menggunakan rumus dibawah
            self.params['dW'] = np.dot((y[:, np.newaxis] - output).T, X) / len(y)
            # Update parameter berdasarkan gradient dikali konstanta belajar (lr/learning rate)
            self.params['W'] += self.lr * self.params['dW']
        # Bila menggunakan metode least square / metode kuadrat terkecil
        else:
            # Perhitungan menggunakan metode kuadrat terkecil agar mendapatkan
            # Parameter optimal untuk model secara langsung agar meminimalkan
            # loss function
            a = np.linalg.inv(np.dot(X.T, X))
            b = np.dot(X.T, y)
            c = np.dot(a, b)

            # Mengupdate parameter berdasarkan parameter optimal
            self.params['W'] = c[:, np.newaxis].T

    # Return output bila mode test
    else:
        return output

    # Return loss serta output
    return loss, output

```

```

def train(self, solver_type="least_square", epoch=50, batch_size=216):
    """Untuk train model linear regresi"""
    if solver_type == "gradient_descent":
        # Bila solver type gradient descent update parameter dilakukan sampai
        # epoch/iterasi tertentu
        for i in range(epoch):
            # Menghitung loss dari model yang di update parameternya
            loss = self.batch(batch_size)
            print(f"[Epoch {i + 1}] : loss = {loss}")
    else:
        # Menghitung loss yang sudah di update parameternya menggunakan least square/metode kuadrat terkecil
        loss, _ = self.calculate(self.X, self.y, "train", "least_square")
        print(f"loss : {loss}")

```

- Diatas merupakan code model dari Linear Regresi menggunakan class dan dengan numpy

### Penjelasan singkat :

- **Inisialisasi & Parameter**

Pada konstruktor, data input  $X$  diubah dengan menambahkan kolom konstanta (nilai 1) sehingga memungkinkan perhitungan intercept. Parameter bobot ( $W$ ) diinisiasi secara acak dan disimpan dalam dictionary bersama dengan gradientnya.

- **Metode Prediksi**

Metode predict mengalikan input (dengan tambahan intercept) dengan bobot ( $W$  atau  $\theta$ ) untuk menghasilkan output prediksi.

- **Fungsi Prediksi/Model Linear Regresi**

$$h(x) = \sum_{i=0}^d \theta_i x_i = \theta^T x$$

- **Fungsi Loss**

Fungsi loss menghitung nilai Mean Squared Error (MSE) dengan faktor 1/2 untuk mempermudah perhitungan gradient.

$$J(\theta) = \frac{1}{2} \sum_{i=1}^n (h_{\theta}(x^{(i)}) - y^{(i)})^2.$$

- **Perhitungan Mini-batch**

Metode batch melakukan pembagian data ke dalam mini-batch, menshuffle data, dan menghitung rata-rata loss untuk setiap batch. Ini digunakan dalam proses training dengan gradient descent.

- **Perhitungan Gradient & Least Square**

Metode calculate melakukan perhitungan output dan loss. Jika mode adalah "train":

Dengan gradient descent, gradient dihitung dan bobot diupdate menggunakan learning rate.

- **Perhitungan Gradient descent**

$$\theta := \theta + \alpha (y^{(i)} - h_{\theta}(x^{(i)})) x^{(i)}$$



Dengan least square, solusi optimal untuk parameter dihitung secara langsung menggunakan invers matriks (metode kuadrat terkecil).

#### - Perhitungan Metode Kuadrat Terkecil

$$\theta = (X^T X)^{-1} X^T \vec{y}.$$

#### - Training Model

Metode train mengatur proses training:

Jika solver yang dipilih adalah gradient descent, model akan diupdate secara iteratif selama beberapa epoch.

Jika menggunakan metode least square, model langsung menghitung parameter optimal dan menampilkan loss akhir.

Menggunakan Gradient Descent sebagai metode optimasi linear regresi

```
[55] # Mengambil data untuk feature X dan label y yaitu 'carat' sebagai kolom yang akan di tebak, lalu menormalisasikanya
X = numeric_data[:, 1:] / np.max(numeric_data[:, 1:],axis=0)
y = numeric_data[:, 0]

# Ngesplit data menjadi train dan testing
X_train, y_train, X_test, y_test = X[:40000, :], y[:40000], X[40000:, :], y[40000:]

# Membuat objek linear regresi dengan data tersebut
model = LinearRegression(X_train, y_train, 0.1)

[56] # Melatih model linear regresi dan melihat loss dari model tersebut ke data
# Melatih menggunakan metode gradient descent (Karena datanya terlalu banyak)
model.train("gradient_descent", epoch=50, batch_size=512)
```

[Epoch 1] : loss = 0.20761858816905865  
[Epoch 2] : loss = 0.21020312309235534  
[Epoch 3] : loss = 0.2253740844088376  
[Epoch 4] : loss = 0.23527846521249177  
[Epoch 5] : loss = 0.24182678280617942  
[Epoch 6] : loss = 0.24628451183495162  
[Epoch 7] : loss = 0.24876718182018925  
[Epoch 8] : loss = 0.2505649598739332  
[Epoch 9] : loss = 0.25180606836337416  
[Epoch 10] : loss = 0.25247298840688753  
[Epoch 11] : loss = 0.25306535894128296  
[Epoch 12] : loss = 0.25337892518312294  
[Epoch 13] : loss = 0.25347603647498373  
[Epoch 14] : loss = 0.253751805677745  
[Epoch 15] : loss = 0.2540989923126807  
[Epoch 16] : loss = 0.25385567816794224  
[Epoch 17] : loss = 0.25415450283973556  
[Epoch 18] : loss = 0.25436372394983464  
[Epoch 19] : loss = 0.2544567038691225  
[Epoch 20] : loss = 0.2544398536619357  
[Epoch 21] : loss = 0.254434422673608  
[Epoch 22] : loss = 0.2546268720349297  
[Epoch 23] : loss = 0.2546495046116675  
[Epoch 24] : loss = 0.2549910111134721  
[Epoch 25] : loss = 0.2549925149813419  
[Epoch 26] : loss = 0.25504281513279337  
[Epoch 27] : loss = 0.2550710583170259  
[Epoch 28] : loss = 0.2554844343775943  
[Epoch 29] : loss = 0.255495673510218  
[Epoch 30] : loss = 0.2554448094337414  
[Epoch 31] : loss = 0.2556922413138098  
[Epoch 32] : loss = 0.2556162406145519  
[Epoch 33] : loss = 0.2557008998533479  
[Epoch 34] : loss = 0.25559025056267376  
[Epoch 35] : loss = 0.255911226644777

[Epoch 36] : loss = 0.25607863705192835  
[Epoch 37] : loss = 0.25609470630452413  
[Epoch 38] : loss = 0.2561539750152556  
[Epoch 39] : loss = 0.25649946741732976  
[Epoch 40] : loss = 0.25654552288652105  
[Epoch 41] : loss = 0.2564729712122764  
[Epoch 42] : loss = 0.2566429731282441  
[Epoch 43] : loss = 0.25661912733997055  
[Epoch 44] : loss = 0.2568877557598924  
[Epoch 45] : loss = 0.25683759771470166  
[Epoch 46] : loss = 0.256872705969071  
[Epoch 47] : loss = 0.2571626091135409  
[Epoch 48] : loss = 0.2573286598824779  
[Epoch 49] : loss = 0.2570905596349302  
[Epoch 50] : loss = 0.25716641245143307

- Model Linear Regresi di atas diterapkan untuk memprediksi nilai y (carat) dengan menggunakan semua fitur numerik kecuali kolom carat itu sendiri. Metode gradient descent digunakan dalam pelatihan, dan loss akhir yang diperoleh adalah 0,2571.

```
# Melakukan Testing
# Mendapatkan prediksi model berdasarkan X_test yaitu test data
output = model.predict(X_test)
print(f"Hasil output atau prediksi y berdasarkan X adalah : \n{output}")
print(f"\nLabel atau y yang asli sebagai perbandingan : \n{y_test}")

# Menghitung seberapa bagus model menggunakan test data menggunakan Mean Squared Error
mse = np.mean(np.square(output - y_test))
print(f"\n\nMean Squared Error dari Testing data Menggunakan Gradient Descent adalah = {mse}")
```

Hasil output atau prediksi y berdasarkan X adalah :

```
[[0.43200416]
 [0.43482444]
 [0.4357614 ]
 ...
 [0.728972  ]
 [0.83429386]
 [0.76255437]]
```

Label atau y yang asli sebagai perbandingan :

```
[0.41 0.41 0.41 ... 0.7 0.86 0.75]
```

Mean Squared Error dari Testing data Menggunakan Gradient Descent adalah = 0.04100089824477601

- Hasil Prediksi dan Mean Squared Error dari Testing data menggunakan Gradient Descent yaitu 0,041

Menggunakan metode kuadrat terkecil sebagai metode optimasi

```
[58] # Mengambil 15000 data pertama untuk feature X dan 5000 label y yaitu 'carat' sebagai kolom yang akan di tebak
# Data hanya 15000 yang dipakai karena metode kuadrat terkecil tidak dapat dipakai untuk data yang banyak
# Ngesplit data menjadi train dan testing
X_train, y_train, X_test, y_test = X[:15000, :], y[:15000], X[15000:20000, :], y[15000:20000]

# Membuat objek linear regresi dengan data tersebut
model = LinearRegression(X_train, y_train, 0.1)
```

```
[59] # Melatih model linear regresi dan melihat loss dari model tersebut ke data
# Melatih menggunakan metode kuadrat terkecil menggunakan 1000 data
model.train("least_square")
```

 loss : 0.22450624147633017

```
[60] # Melakukan Testing
# Mendapatkan prediksi model berdasarkan X_test yaitu test data
output = model.predict(X_test)
print(f"Hasil output atau prediksi y berdasarkan X adalah : \n{output}")
print(f"\nLabel atau y yang asli sebagai perbandingan : \n{y_test}")

# Menghitung seberapa bagus model menggunakan test data menggunakan Mean Squared Error
mse = np.mean(np.square(output - y_test))
print(f"\n\nMean Squared Error dari Testing data Menggunakan Metode Kuadrat Terkecil adalah = {mse}")
```

- Mengaplikasikan kembali model Linear Regresi dengan pendekatan metode kuadrat terkecil untuk memprediksi nilai carat, menggunakan fitur (X) yang sama. Loss nya untuk training adalah 0,224

Hasil output atau prediksi y berdasarkan X adalah :

```
[[1.22171809]
 [1.01841291]
 [0.99840497]
 ...
 [1.04579726]
 [1.39667305]
 [1.31261814]]
```

Label atau y yang asli sebagai perbandingan :

```
[1.3 1.01 1. ... 1.05 1.57 1.5 ]
```

Mean Squared Error dari Testing data Menggunakan Metode Kuadrat Terkecil adalah = 0.20825816177165213

- Prediksi y dan MSE dari Testing data menggunakan Metode Kuadrat Terkecil yaitu 0,2