Introduction

The garment industry refers to the production and distribution of clothing and related accessories, which also includes the textile sector that creates fibers and fabrics. As a significant part of the global manufacturing landscape, the garment industry employs millions of people and contributes trillions of dollars to the economy. Given its scale, maximizing workforce productivity is essential for the industry's success. This project focuses on estimating the productivity score for each team in a garment factory by analyzing a dataset related to worker productivity using an Artificial Neural Network (ANN) model in deep learning. ANN is particularly effective at identifying complex patterns within data, allowing us to pinpoint key factors that influence productivity scores. Through systematic data collection, preprocessing, model development, and evaluation, we aim to derive valuable insights that can improve efficiency and assist in strategic decision-making within the garment industry.

```
# Install keras tuner
!pip install keras-tuner
```

```
Requirement already satisfied: keras-tuner in
/usr/local/lib/python3.11/dist-packages (1.4.7)
Requirement already satisfied: keras in
/usr/local/lib/python3.11/dist-packages (from keras-tuner) (3.8.0)
Requirement already satisfied: packaging in
/usr/local/lib/python3.11/dist-packages (from keras-tuner) (24.2)
Requirement already satisfied: requests in
/usr/local/lib/python3.11/dist-packages (from keras-tuner) (2.32.3)
Requirement already satisfied: kt-legacy in
/usr/local/lib/python3.11/dist-packages (from keras-tuner) (1.0.5)
Requirement already satisfied: absl-py in
/usr/local/lib/python3.11/dist-packages (from keras->keras-tuner)
(1.4.0)
Requirement already satisfied: numpy in
/usr/local/lib/python3.11/dist-packages (from keras->keras-tuner)
(2.0.2)
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-
packages (from keras->keras-tuner) (13.9.4)
Requirement already satisfied: namex in
/usr/local/lib/python3.11/dist-packages (from keras->keras-tuner)
(0.0.8)
Requirement already satisfied: h5py in /usr/local/lib/python3.11/dist-
packages (from keras->keras-tuner) (3.13.0)
Requirement already satisfied: optree in
/usr/local/lib/python3.11/dist-packages (from keras->keras-tuner)
(0.15.0)
Requirement already satisfied: ml-dtypes in
/usr/local/lib/python3.11/dist-packages (from keras->keras-tuner)
(0.4.1)
Requirement already satisfied: charset-normalizer<4,>=2 in
```

```
/usr/local/lib/python3.11/dist-packages (from requests->keras-tuner)
(3.4.1)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.11/dist-packages (from requests->keras-tuner)
(3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.11/dist-packages (from requests->keras-tuner)
(2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.11/dist-packages (from requests->keras-tuner)
(2025.1.31)
Requirement already satisfied: typing-extensions>=4.5.0 in
/usr/local/lib/python3.11/dist-packages (from optree->keras->keras-
tuner) (4.13.1)
Requirement already satisfied: markdown-it-py>=2.2.0 in
/usr/local/lib/python3.11/dist-packages (from rich->keras->keras-
tuner) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in
/usr/local/lib/python3.11/dist-packages (from rich->keras->keras-
tuner) (2.18.0)
Requirement already satisfied: mdurl~=0.1 in
/usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0-
>rich->keras->keras-tuner) (0.1.2)
```

```python
# Import Library
import keras_tuner as kt

import pandas as pd
import numpy as np
import random

import tensorflow as tf
from tensorflow.keras.layers import Dense, Dropout, Input
from tensorflow.keras.models import Sequential,Model
from tensorflow.keras.utils import plot_model
from tensorflow.keras.regularizers import l2
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping

import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import
RobustScaler,StandardScaler,MinMaxScaler,LabelEncoder,OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.metrics import r2_score, mean_absolute_error,
mean_squared_error

from warnings import filterwarnings
```

```
filterwarnings('ignore')

SEED_VALUE=1234
random.seed(SEED_VALUE)
np.random.seed(SEED_VALUE)

# Load Dataset
df = pd.read_parquet("dataset_1B.parquet")
df.head()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 1197,\n  \"fields\":
[\n    {\n      \"column\": \"date\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 118,\n
\"samples\": [\n          \"2/3/2015\",\n          \"2016-02-23\",\n
\"1/4/2015\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"quarter\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          \"Quarter2\",\n          \"Quarter5\",\n
\"Quarter3\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"day\",\n      \"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 6,\n        \"samples\": [\n
\"Thursday\",\n          \"Saturday\",\n          \"Wednesday\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Team Code\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3,\n        \"min\": 1,\n        \"max\": 12,\n
\"num_unique_values\": 12,\n        \"samples\": [\n          5,\n
10,\n          8\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"smv\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 10.943219199514333,\n        \"min\": 2.9,\n        \"max\":
54.56,\n        \"num_unique_values\": 70,\n        \"samples\": [\n
14.61,\n          26.16,\n          30.1\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"wip\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1837.4550011056342,\n
\"min\": 7.0,\n        \"max\": 23122.0,\n
\"num_unique_values\": 548,\n        \"samples\": [\n
1287.0,\n          970.0,\n          958.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"over_time\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3348,\n        \"min\": 0,\n        \"max\": 25920,\n
\"num_unique_values\": 143,\n        \"samples\": [\n          5820,\n
6780,\n          7470\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"incentive\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 160,\n        \"min\": 0,\n
```

```
\"max\": 3600,\n        \"num_unique_values\": 48,\n
\"samples\": [\n          55,\n          65,\n          81\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"idle_time\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 12.709756518546547,\n        \"min\":
0.0,\n        \"max\": 300.0,\n        \"num_unique_values\": 12,\n
\"samples\": [\n          4.0,\n          3.5,\n          0.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"idle_men\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3,\n        \"min\": 0,\n        \"max\": 45,\n
\"num_unique_values\": 10,\n        \"samples\": [\n          25,\n
10,\n          30\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"no_of_style_change\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 2,\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          0,\n          1,\n          2\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"no_of_workers\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
22.61704313753719,\n        \"min\": -57.0,\n        \"max\": 89.0,\n
\"num_unique_values\": 66,\n        \"samples\": [\n          50.0,\n
46.0,\n          59.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"productivity_score\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 18.154945385667887,\n        \"min\": -
100.0,\n        \"max\": 112.044,\n        \"num_unique_values\":
803,\n        \"samples\": [\n          97.462,\n          97.008,\n
80.031\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df"}
```


Structure

```
# Shape of the dataset
df.shape
```

```
(1197, 13)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1197 entries, 0 to 1196
Data columns (total 13 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
```

```
 0   date                 1197 non-null   object 
 1   quarter              1197 non-null   object 
 2   day                  1197 non-null   object 
 3   Team Code            1197 non-null   int64  
 4   smv                  1197 non-null   float64
 5   wip                  691 non-null    float64
 6   over_time            1197 non-null   int64  
 7   incentive            1197 non-null   int64  
 8   idle_time            1197 non-null   float64
 9   idle_men             1197 non-null   int64  
 10  no_of_style_change   1197 non-null   int64  
 11  no_of_workers        1197 non-null   float64
 12  productivity_score   1197 non-null   float64
dtypes: float64(5), int64(5), object(3)
memory usage: 121.7+ KB
```

The dataset consists of 1197 entries (rows) and 13 columns (features), as indicated by its shape of (1197, 13).

Based on the information of the dataset, It includes three categorical columns—date, quarter, and day, along with 10 numerical columns such as Team Code, smv (Standard Minute Value), wip (Work In Progress), over_time, incentive, idle_time, idle_men, no_of_style_change, no_of_workers, and productivity_score. Further analysis is needed to determine the appropriate handling of these variables.

Below are the detailed features in the dataset:

- date: The date of assessment
- day: The day of the Week
- quarter: The quarter of the year when the data was recorded (e.g., Quarter1, Quarter2)
- Team Code: A unique identifier for the team.
- smv: The Standard Minute Value, which indicates the time allocated for specific task.
- wip: Work In Progress, reflecting number of unfinished products
- over_time: The amount of overtime worked, measured in minutes.
- incentive: The financial incentive provided to the workers, measured in USD.
- idle_time: The total time that workers were idle, also measured in minutes.
- idle_men: The number of workers who were idle / not engaged in work.
- no_of_style_change: The total number of style changes that occurred.
- no_of_workers: The total number of workers in the team.
- productivity_score: The team’s productivity score, expressed as a percentage.

```
# Display all columns in the dataset
df.columns

Index(['date', 'quarter', 'day', 'Team Code', 'smv', 'wip',
'over_time',
       'incentive', 'idle_time', 'idle_men', 'no_of_style_change',
```

```
       'no_of_workers', 'productivity_score'],
      dtype='object')
```

```python
# Statistics / Numerical Analysis
df.describe().T
```

{"summary":"{\n \"name\": \"df\",\n \"rows\": 10,\n \"fields\": [\n   {\n   \"column\": \"count\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 160.01124960451997,\n   \"min\": 691.0,\n   \"max\": 1197.0,\n   \"num_unique_values\": 2,\n   \"samples\": [\n   691.0,\n   1197.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"mean\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1444.3283091080204,\n   \"min\": 0.15037593984962405,\n   \"max\": 4567.460317460317,\n   \"num_unique_values\": 10,\n   \"samples\": [\n   34.3375104427736,\n   15.062172096908938\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"std\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1139.304886133768,\n   \"min\": 0.4278478565061976,\n   \"max\": 3348.823562883226,\n   \"num_unique_values\": 10,\n   \"samples\": [\n   22.61704313753719,\n   10.943219199514333\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"min\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 35.23396246678921 5,\n   \"min\": -100.0,\n   \"max\": 7.0,\n   \"num_unique_values\": 6,\n   \"samples\": [\n   1.0,\n   2.9\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"25%\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 488.8610825639074,\n   \"min\": 0.0,\n   \"max\": 1440.0,\n   \"num_unique_values\": 7,\n   \"samples\": [\n   3.0,\n   3.94\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"50%\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1253.2270162674747,\n   \"min\": 0.0,\n   \"max\": 3960.0,\n   \"num_unique_values\": 7,\n   \"samples\": [\n   6.0,\n   15.26\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"75%\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 2183.381233001351,\n   \"min\": 0.0,\n   \"max\": 6960.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   24.26,\n   0.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"max\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 10197.615852575293,\n   \"min\": 2.0,\n   \"max\": 25920.0,\n   \"num_unique_values\": 10,\n   \"samples\": [\n

89.0,\n   54.56\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

The summary statistics of the dataset reveal important insights into various numerical features. The Team Code has a mean of 6.42 with a limited range, indicating a small number of teams. Meanwhile, the Incentive varies widely, with a mean of 38.21 USD, although some teams receive none. Additionally, the average idle time and number of idle men are low, suggesting minimal inefficiencies. However, there are negative values in the "no_of_workers" column (−57) and the "productivity_score" column (−100), which need to be addressed during preprocessing to clean the data. Overall, these statistics guide further investigations and analyses aimed at optimizing productivity in the garment factory.

Data Cleaning

```
# Check duplicate data
df.duplicated().sum()
```

```
np.int64(0)
```

By checking duplicate data in the dataset, it was confirmed that there are no duplicates, ensuring that it is free from redundancy and ready for further analysis.

```
# Check missing value
df.isna().sum()
```

```
date                  0
quarter               0
day                   0
Team Code             0
smv                   0
wip                 506
over_time             0
incentive             0
idle_time             0
idle_men              0
no_of_style_change    0
no_of_workers         0
productivity_score    0
dtype: int64
```

After checking the missing values for each column, it was found that "wip" (work in progress) column has 506 missing values. Meanwhile, other columns contain no missing data.

This highlights the importance of addressing missing values in "wip" column, that require us to decide whether impute or use other handling methods when preprocessing.

```python
# Rename column
df = df.rename(columns={'wip': 'work_in_progress', 'smv':
'std_minute_value'})
df.head()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 1197,\n  \"fields\":
[\n    {\n      \"column\": \"date\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 118,\n
\"samples\": [\n          \"2/3/2015\",\n          \"2016-02-23\",\n
\"1/4/2015\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"quarter\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          \"Quarter2\",\n          \"Quarter5\",\n
\"Quarter3\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"day\",\n      \"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 6,\n        \"samples\": [\n
\"Thursday\",\n          \"Saturday\",\n          \"Wednesday\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Team Code\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3,\n        \"min\": 1,\n        \"max\": 12,\n
\"num_unique_values\": 12,\n        \"samples\": [\n          5,\n
10,\n          8\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"std_minute_value\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 10.943219199514333,\n        \"min\":
2.9,\n        \"max\": 54.56,\n        \"num_unique_values\": 70,\n
\"samples\": [\n          14.61,\n          26.16,\n          30.1\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"work_in_progress\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1837.4550011056342,\n        \"min\": 7.0,\n        \"max\": 23122.0,\
n        \"num_unique_values\": 548,\n        \"samples\": [\n
1287.0,\n          970.0,\n          958.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"over_time\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3348,\n        \"min\": 0,\n        \"max\": 25920,\n
\"num_unique_values\": 143,\n        \"samples\": [\n          5820,\n
6780,\n          7470\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"incentive\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 160,\n        \"min\": 0,\n
\"max\": 3600,\n        \"num_unique_values\": 48,\n
\"samples\": [\n          55,\n          65,\n          81\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"idle_time\",\n      \"properties\": {\n        \"dtype\":
```

```
\"number\",\n        \"std\": 12.709756518546547,\n        \"min\":
0.0,\n        \"max\": 300.0,\n        \"num_unique_values\": 12,\n
\"samples\": [\n          4.0,\n          3.5,\n          0.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"idle_men\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3,\n        \"min\": 0,\n        \"max\": 45,\n
\"num_unique_values\": 10,\n        \"samples\": [\n          25,\n
10,\n          30\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"no_of_style_change\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 2,\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          0,\n          1,\n          2\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"no_of_workers\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
22.61704313753719,\n        \"min\": -57.0,\n        \"max\": 89.0,\n
\"num_unique_values\": 66,\n        \"samples\": [\n          50.0,\n
46.0,\n          59.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"productivity_score\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 18.154945385667887,\n        \"min\": -
100.0,\n        \"max\": 112.044,\n        \"num_unique_values\":
803,\n        \"samples\": [\n          97.462,\n          97.008,\n
80.031\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df"}
```


I decided to rename the "wip" column to "work_in_progress" and the "smv" column to "std_minute_value" to make them easier to understand.

```
# Convert "date" column into datetime format
df['date'] = pd.to_datetime(df['date'], format = 'mixed')

df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1197 entries, 0 to 1196
Data columns (total 13 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   date                 1197 non-null   datetime64[ns]
 1   quarter              1197 non-null   object
 2   day                  1197 non-null   object
 3   Team Code            1197 non-null   int64
 4   std_minute_value     1197 non-null   float64
 5   work_in_progress     691 non-null    float64
 6   over_time            1197 non-null   int64
```

```
 7   incentive            1197 non-null   int64
 8   idle_time            1197 non-null   float64
 9   idle_men             1197 non-null   int64
 10  no_of_style_change   1197 non-null   int64
 11  no_of_workers        1197 non-null   float64
 12  productivity_score   1197 non-null   float64
dtypes: datetime64[ns](1), float64(5), int64(5), object(2)
memory usage: 121.7+ KB
```

```
# Split "date" column into 3 new column and drop "date" column
df['Date'] = df['date'].dt.day
df['Month'] = df['date'].dt.month
df['Year'] = df['date'].dt.year

df = df.drop(columns = 'date')
```

Next, I decided to convert the "date" column from an object to a datetime format to ensure the data type is correct. Additionally, I split the "date" column into 3 separate columns which are "Day", "Month" and "Year" to maintain clarity regarding the "quarter" column are related to months in a year. This approach helps to avoid any confusion in the analysis.

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1197 entries, 0 to 1196
Data columns (total 15 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   quarter              1197 non-null   object
 1   day                  1197 non-null   object
 2   Team Code            1197 non-null   int64
 3   std_minute_value     1197 non-null   float64
 4   work_in_progress     691 non-null    float64
 5   over_time            1197 non-null   int64
 6   incentive            1197 non-null   int64
 7   idle_time            1197 non-null   float64
 8   idle_men             1197 non-null   int64
 9   no_of_style_change   1197 non-null   int64
 10  no_of_workers        1197 non-null   float64
 11  productivity_score   1197 non-null   float64
 12  Date                 1197 non-null   int32
 13  Month                1197 non-null   int32
 14  Year                 1197 non-null   int32
dtypes: float64(5), int32(3), int64(5), object(2)
memory usage: 126.4+ KB
```

Now, the "date" column has been dropped, and the 3 newly created column are set in numerical format.

```python
# Numerical Analysis
df.describe().T
```


{"summary":"{\n \"name\": \"df\",\n \"rows\": 13,\n \"fields\": [\n   {\n   \"column\": \"count\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 140.33914964498294,\n   \"min\": 691.0,\n   \"max\": 1197.0,\n   \"num_unique_values\": 2,\n   \"samples\": [\n   691.0,\n   1197.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"mean\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1337.8633025679371,\n   \"min\": 0.15037593984962405,\n   \"max\": 4567.460317460317,\n   \"num_unique_values\": 13,\n   \"samples\": [\n   1.717627401837928,\n   73.36704010025063\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"std\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1014.5339489507038,\n   \"min\": 0.4278478565061976,\n   \"max\": 3348.823562883226,\n   \"num_unique_values\": 13,\n   \"samples\": [\n   0.7375261810563015,\n   18.154945385667887\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"min\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 563.0484819950437,\n   \"min\": -100.0,\n   \"max\": 2015.0,\n   \"num_unique_values\": 7,\n   \"samples\": [\n   1.0,\n   2.9\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"25%\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 664.7999150872388,\n   \"min\": 0.0,\n   \"max\": 2015.0,\n   \"num_unique_values\": 10,\n   \"samples\": [\n   1.0,\n   3.94\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"50%\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1186.1962432085065,\n   \"min\": 0.0,\n   \"max\": 3960.0,\n   \"num_unique_values\": 10,\n   \"samples\": [\n   2.0,\n   15.26\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"75%\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1950.2954183206284,\n   \"min\": 0.0,\n   \"max\": 6960.0,\n   \"num_unique_values\": 11,\n   \"samples\": [\n   0.0,\n   9.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"max\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 9075.24221067502,\n   \"min\": 2.0,\n   \"max\": 25920.0,\n   \"num_unique_values\": 13,\n   \"samples\": [\n   3.0,\n   112.044\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
df['no_of_workers'] = df['no_of_workers'].abs()
df['productivity_score'] = df['productivity_score'].abs()

df.describe().T
```


{"summary":"{\n \"name\": \"df\",\n \"rows\": 13,\n \"fields\": [\n
{\n   \"column\": \"count\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 140.33914964498294,\n
\"min\": 691.0,\n   \"max\": 1197.0,\n
\"num_unique_values\": 2,\n   \"samples\": [\n   691.0,\n
1197.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"mean\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 1337.8478970353888,\n   \"min\": 0.15037593984962405,\n
\"max\": 4567.460317460317,\n   \"num_unique_values\": 13,\n
\"samples\": [\n   1.717627401837928,\n
73.53412447786133\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"std\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 1014.5701984729498,\n   \"min\": 0.4278478565061976,\n
\"max\": 3348.823562883226,\n   \"num_unique_values\": 13,\n
\"samples\": [\n   0.7375261810563015,\n
17.46529787377751\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"min\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 558.0123906002539,\n   \"min\": 0.0,\n   \"max\":
2015.0,\n   \"num_unique_values\": 7,\n   \"samples\": [\n
1.0,\n   2.9\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"25%\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 664.799881598523,\n   \"min\": 0.0,\n   \"max\":
2015.0,\n   \"num_unique_values\": 10,\n   \"samples\": [\n
1.0,\n   3.94\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"50%\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 1186.1962432085065,\n   \"min\": 0.0,\n   \"max\":
3960.0,\n   \"num_unique_values\": 10,\n   \"samples\": [\n
2.0,\n   15.26\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"75%\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 1950.2952334881873,\n   \"min\": 0.0,\n   \"max\":
6960.0,\n   \"num_unique_values\": 11,\n   \"samples\": [\n
0.0,\n   9.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"max\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 9075.24221067502,\n   \"min\": 2.0,\n   \"max\":
25920.0,\n   \"num_unique_values\": 13,\n   \"samples\": [\n
3.0,\n   112.044\n   ],\n   \"semantic_type\": \"\",\
n   \"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe"}

I decided to convert the negative values in the "no_of_workers" column and "productivity_score" column to ensure that all entries are positive, which maintains the integrity of the data and allows for accurate analysis in predicting productivity in the garment factory. This is crucial because the number of workers cannot be negative, and a productivity score should not be negative either, as it represents performance relative to a defined standard.

EDA (Exploratory Data Analysis)

```
# Missing values
df.isna().sum()
```

```
quarter                  0
day                      0
Team Code                0
std_minute_value         0
work_in_progress       506
over_time                0
incentive                0
idle_time                0
idle_men                 0
no_of_style_change       0
no_of_workers            0
productivity_score       0
Date                     0
Month                    0
Year                     0
dtype: int64
```

```
# Check skew value for the work_in_progress column
df.work_in_progress.skew()
```

```
np.float64(9.741786273952965)
```

```
# Fill missing value with median
df['work_in_progress'].fillna(df['work_in_progress'].median(),inplace=True)
df.work_in_progress.median()
```

```
1039.0
```

It was observed that there are 506 missing values in the "work_in_progress" column. I checked the skewness to understand the distribution of the data, which revealed a skewness of 9.74, indicating a highly right-skewed distribution. This suggests that there are a significant number of lower values, with a few high values pulling the mean upward. Consequently, to determine the best method for imputing the missing values, I decided to use the median, because it is a robust statistic that is less influenced by outliers, making it a more accurate representation of the central tendency in this case.

```python
# Re-check missing values
df.isna().sum()
```

```
quarter               0
day                   0
Team Code             0
std_minute_value      0
work_in_progress      0
over_time             0
incentive             0
idle_time             0
idle_men              0
no_of_style_change    0
no_of_workers         0
productivity_score    0
Date                  0
Month                 0
Year                  0
dtype: int64
```

Now, the data has no missing values and is clean, making it ready for further analysis. This preparation allows us to confidently proceed with tasks such as modeling and predicting productivity scores within the garment dataset.

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1197 entries, 0 to 1196
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   quarter             1197 non-null   object
 1   day                 1197 non-null   object
 2   Team Code           1197 non-null   int64
 3   std_minute_value    1197 non-null   float64
 4   work_in_progress    1197 non-null   float64
 5   over_time           1197 non-null   int64
 6   incentive           1197 non-null   int64
 7   idle_time           1197 non-null   float64
 8   idle_men            1197 non-null   int64
 9   no_of_style_change  1197 non-null   int64
 10  no_of_workers       1197 non-null   float64
 11  productivity_score  1197 non-null   float64
 12  Date                1197 non-null   int32
 13  Month               1197 non-null   int32
 14  Year                1197 non-null   int32
dtypes: float64(5), int32(3), int64(5), object(2)
memory usage: 126.4+ KB
```

```python
# Separating categorical and numerical features based on dataset info
cat_cols = ['quarter', 'day']
num_cols = ['std_minute_value', 'over_time', 'incentive', 'idle_time',
'idle_men', 'no_of_style_change', 'no_of_workers',
'productivity_score', 'Date', 'Month', 'Year']

# Value count for each columns
for i in df.columns:
    x = df[i].value_counts()
    print(f'{x}\n')
```

```
quarter
Quarter1    360
Quarter2    335
Quarter4    248
Quarter3    210
Quarter5     44
Name: count, dtype: int64

day
Wednesday    208
Sunday       203
Tuesday      201
Thursday     199
Monday       199
Saturday     187
Name: count, dtype: int64

Team Code
8     109
2     109
4     105
1     105
9     104
10    100
12     99
7      96
3      95
6      94
5      93
11     88
Name: count, dtype: int64

std_minute_value
3.94     192
2.90     108
22.52    103
30.10     79
4.15      76
        ...
```

```
38.09      1
48.18      1
30.40      1
50.89      1
20.20      1
Name: count, Length: 70, dtype: int64

work_in_progress
1039.0    511
1282.0      4
1144.0      3
1193.0      3
1069.0      3
         ...
817.0       1
1576.0      1
1262.0      1
953.0       1
1161.0      1
Name: count, Length: 548, dtype: int64

over_time
960     129
1440    111
6960     61
6840     48
1200     39
       ...
5700      1
1680      1
1700      1
4680      1
3120      1
Name: count, Length: 143, dtype: int64

incentive
0       604
50      113
63       61
45       54
30       52
23       38
38       29
60       28
40       27
75       24
113      21
88       19
34       17
56       14
```

```
26        9
55        7
81        7
100       7
65        6
69        6
70        6
960       5
35        5
44        4
94        4
90        3
49        2
27        2
46        2
119       2
24        2
98        1
29        1
54        1
37        1
21        1
138       1
33        1
53        1
93        1
62        1
32        1
1080      1
2880      1
3600      1
1440      1
1200      1
25        1
Name: count, dtype: int64

idle_time
0.0      1179
3.5         3
2.0         2
8.0         2
4.0         2
4.5         2
5.0         2
90.0        1
270.0       1
150.0       1
300.0       1
6.5         1
```

```
Name: count, dtype: int64

idle_men
0     1179
10       3
15       3
30       3
20       3
35       2
37       1
45       1
25       1
40       1
Name: count, dtype: int64

no_of_style_change
0    1050
1     114
2      33
Name: count, dtype: int64

no_of_workers
8.0     262
58.0    114
57.0    109
59.0     75
10.0     60
       ...
26.0      1
47.0      1
48.0      1
24.0      1
6.0       1
Name: count, Length: 61, dtype: int64

productivity_score
80.040    25
75.065    17
80.012    13
85.014    12
97.187    12
          ..
37.047     1
97.756     1
94.560     1
84.053     1
77.815     1
Name: count, Length: 802, dtype: int64

Date
```

```
10    65
7     64
8     62
3     61
4     59
1     58
11    54
5     52
12    42
24    42
28    42
22    42
25    42
17    41
2     41
9     41
26    40
18    40
19    39
15    38
14    38
6     28
29    28
31    24
13    22
27    21
23    19
21    19
16    18
20    15
Name: count, dtype: int64

Month
1    542
2    451
3    204
Name: count, dtype: int64

Year
2015    701
2016    496
Name: count, dtype: int64
```

I performed value counts for each column in the dataset to analyze the distribution of categorical and numerical features. In the "Quarter" column, Quarter 1 had the highest frequency with 360 entries, while the "Day" column showed Wednesday as the most common day with 208 entries. The "Team Code" values were fairly balanced, with Team 8 and Team 2 each having 109 entries. The "Standard Minute Value" ranged across 70 unique values, with the most frequent being 3.94. The "Work In Progress" column showed that 1039.0 was the most common value, and the majority of entries in "Idle Time" and "Idle Men" reported zero. The "Incentive" column revealed that 604 entries received no incentive, while the "Productivity Score" had 803 unique values. Overall, this analysis provides valuable insights into the dataset, aiding in understanding trends and patterns related to productivity in the garment factory.

```
# Check unique value in categorical column
for i in cat_cols:
    print(f'{i}: {df[i].nunique()}')
    print(df[i].unique())
    print()

quarter: 5
['Quarter1' 'Quarter2' 'Quarter3' 'Quarter4' 'Quarter5']

day: 6
['Thursday' 'Saturday' 'Sunday' 'Monday' 'Tuesday' 'Wednesday']
```

I checked the unique values in the categorical columns, and it showed that there are 5 quarters. This indicates that Quarter 5 needs to be addressed since there should only be 4 quarters in total. Additionally, in the "day" column, there are only 6 days represented, with no entries for Friday.

```
# Filter the dataset for any records associated with that specific quarter
df[df['quarter'] == 'Quarter5']
```


```
{"summary":"{\n  \"name\": \"df[df['quarter'] == 'Quarter5']\",\n  \"rows\": 44,\n  \"fields\": [\n    {\n      \"column\": \"quarter\",\n      \"properties\": {\n        \"dtype\": \"category\",\n        \"num_unique_values\": 1,\n        \"samples\": [\n          \"Quarter5\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"day\",\n      \"properties\": {\n        \"dtype\": \"category\",\n        \"num_unique_values\": 2,\n        \"samples\": [\n          \"Saturday\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"Team Code\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 3,\n        \"min\": 1,\n        \"max\": 12,\n        \"num_unique_values\": 12,\n        \"samples\": [\n          12\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"std_minute_value\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 10.94775058469517,\n        \"min\": 2.9,\n        \"max\": 50.89,\n        \"num_unique_values\": 14,\n        \"samples\": [\n          4.08\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"work_in_progress\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 235.44061859356583,\n        \"min\": 282.0,\n        \"max\": 1601.0,\n
```

```
\"num_unique_values\": 24,\n        \"samples\": [\n          1436.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"over_time\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2729,\n        \"min\": 240,\n        \"max\": 7080,\n
\"num_unique_values\": 18,\n        \"samples\": [\n          6840\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"incentive\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
41,\n        \"min\": 0,\n        \"max\": 113,\n
\"num_unique_values\": 12,\n        \"samples\": [\n          56\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"idle_time\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.0,\n        \"min\": 0.0,\n        \"max\": 0.0,\n
\"num_unique_values\": 1,\n        \"samples\": [\n          0.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"idle_men\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0,\n        \"min\": 0,\n        \"max\": 0,\n
\"num_unique_values\": 1,\n        \"samples\": [\n          0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"no_of_style_change\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0,\n        \"min\": 0,\n        \"max\": 0,\n
\"num_unique_values\": 1,\n        \"samples\": [\n          0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"no_of_workers\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
23.20755007642304,\n        \"min\": 2.0,\n        \"max\": 59.0,\n
\"num_unique_values\": 16,\n        \"samples\": [\n          57.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"productivity_score\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
18.385227641756785,\n        \"min\": 28.698,\n        \"max\":
100.046,\n        \"num_unique_values\": 30,\n        \"samples\": [\n
60.071\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Date\",\n      \"properties\": {\n        \"dtype\": \"int32\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n          31\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Month\",\n      \"properties\":
{\n        \"dtype\": \"int32\",\n        \"num_unique_values\": 1,\n
\"samples\": [\n          1\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Year\",\n      \"properties\": {\n        \"dtype\":
\"int32\",\n        \"num_unique_values\": 2,\n        \"samples\": [\
n          2016\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```

```
# Drop Quarter5
df = df[df['quarter'] != 'Quarter5']
df['quarter'].unique()

array(['Quarter1', 'Quarter2', 'Quarter3', 'Quarter4'], dtype=object)
```

I decided to drop the Quarter5 data because, upon filtering, it showed that there are only 30 entries associated with Quarter5. This small number is not significant enough to justify creating a separate quarter, especially since it deviates from the standard four-quarter structure usually used in financial and performance analyses. Allocating these entries to Quarter1 could lead to misinterpretations of the data, as it may distort the understanding of seasonality and trends within the dataset.

```
# Check the unique entries
df['Team Code'].unique()

array([ 8,  1, 11, 12,  6,  7,  2,  3,  9, 10,  5,  4])

df['no_of_style_change'].unique()

array([0, 1, 2])

df['Month'].unique()

array([1, 2, 3], dtype=int32)

df['Year'].unique()

array([2015, 2016], dtype=int32)

cat_cols = ['quarter', 'day', 'Month', 'Year', 'Team Code',
'no_of_style_change']
num_cols = ['std_minute_value', 'over_time', 'incentive', 'idle_time',
'idle_men', 'no_of_workers', 'productivity_score', 'Date']
```

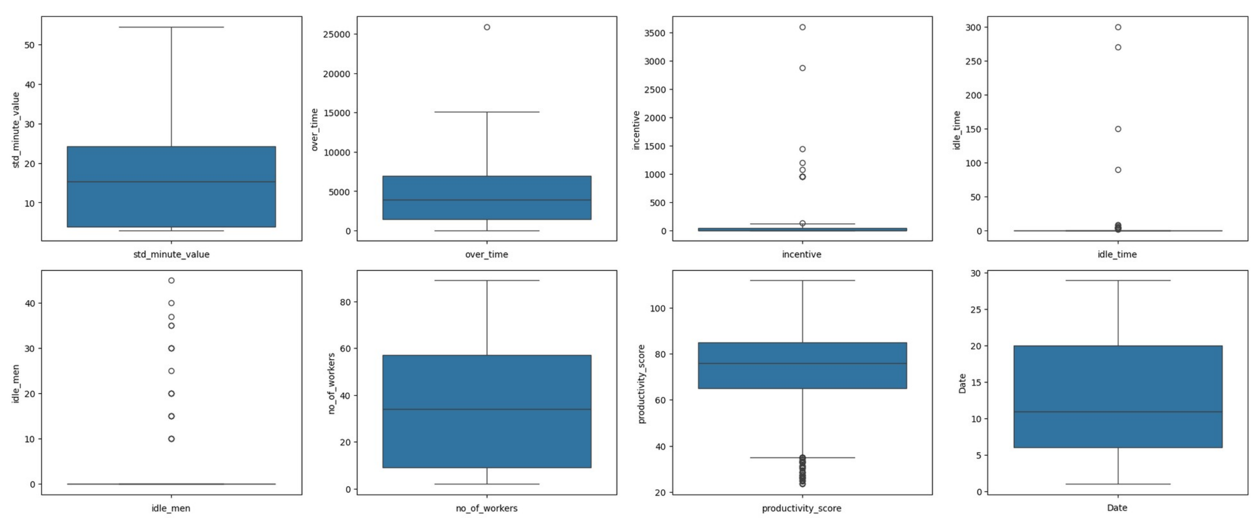
Change the "month," "year," "team code," and "no_of_style_change" columns into categorical types because the "month" column contains only values of 1, 2, or 3, representing three months; the "year" column consists only of the years 2015 and 2016; the "team code" ranges from 0 to 12; and the "no_of_style_change" column includes values of 0, 1, and 2.

```
from scipy.stats import skew, kurtosis

for col in num_cols:
    col_skew = skew(df[col].dropna())      # dropna to avoid issues
with NaNs
    col_kurt = kurtosis(df[col].dropna())  # by default, Fisher’s
definition (normal ==> kurtosis 0)
    print(f"{col} - Skewness: {col_skew:.2f}, Kurtosis:
{col_kurt:.2f}")
kurtosis(df[col].dropna(), fisher=False)
```

```
std_minute_value - Skewness: 0.39, Kurtosis: -0.85
over_time - Skewness: 0.68, Kurtosis: 0.42
incentive - Skewness: 15.53, Kurtosis: 288.07
idle_time - Skewness: 20.14, Kurtosis: 424.41
idle_men - Skewness: 9.65, Kurtosis: 98.58
no_of_workers - Skewness: -0.12, Kurtosis: -1.78
productivity_score - Skewness: -0.81, Kurtosis: 0.36
Date - Skewness: 0.31, Kurtosis: -1.13

np.float64(1.8664251572797201)
```

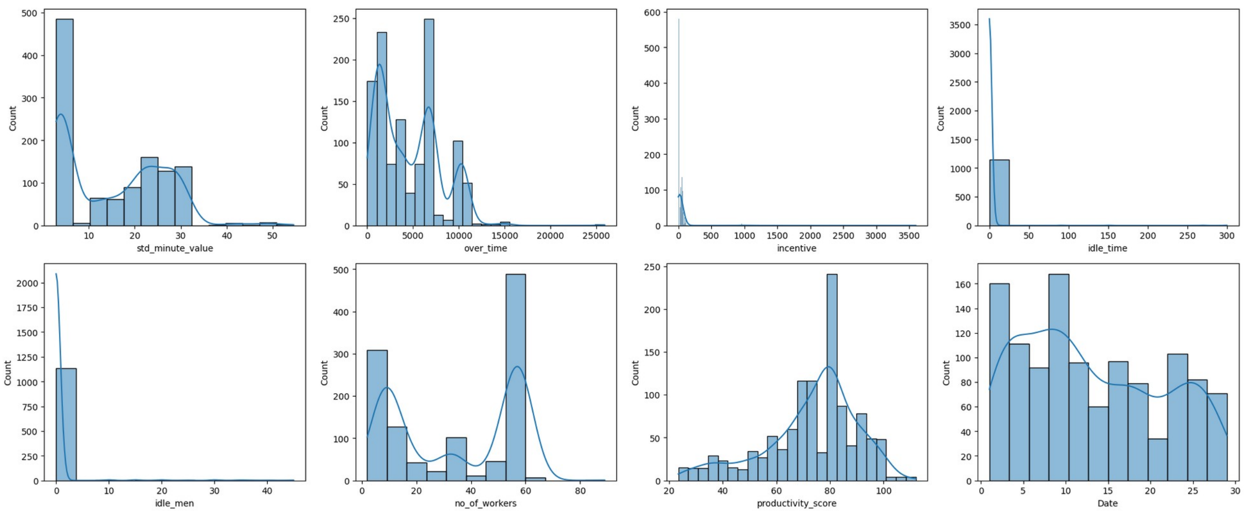
Then, I also checked the skewness value for the numerical features to assess the distribution of the data. This helps determine whether the data is normally distributed or if it exhibits significant skewness, which could impact the choice of statistical methods and models used in the analysis. A skewness value of 20.14 (idle_time column) indicates a highly right-skewed distribution. This means that the majority of the data points are concentrated on the left side of the distribution. In practical terms, this high level of skewness can affect statistical analyses and modeling, making it necessary to consider data transformations or other approaches to better handle the distribution and ensure accurate insights.

```
# Boxplot
fig = plt.figure(figsize=(20, 20))

for i, col in enumerate(num_cols, 1):
    plt.subplot(5, 4, i)
    sns.boxplot(df[col])
    plt.xlabel(col)

plt.tight_layout()
plt.show()
```



I decided to keep the outliers because removing them would result in a loss of valuable data. Since outliers can sometimes represent significant events or rare occurrences in the dataset, preserving them allows for a more comprehensive analysis and helps ensure that important insights are not overlooked.

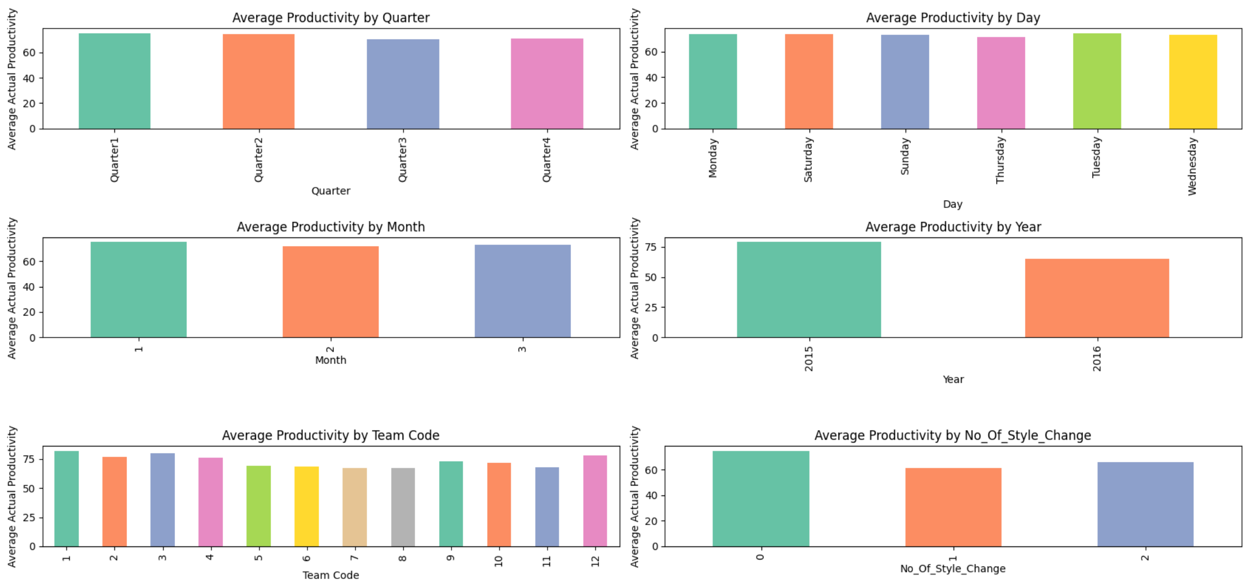
```
# Distribution of numerical features - Histogram
fig = plt.figure(figsize=(20, 20))

for i, col in enumerate(num_cols, 1):
    plt.subplot(5, 4, i)
    sns.histplot(df[col], kde=True)
    plt.xlabel(col)

plt.tight_layout()
plt.show()
```



The distributions of the numerical features indicate that the Standard Minute Value and Over Time features are left-skewed. In contrast, the Incentive, Idle Time, and Idle Men features are heavily peaked at zero. I chose not to handle these columns to avoid losing data. The Number of Workers displays a bimodal pattern, reflecting variability in team sizes, while the Productivity Score peaks around 80, suggesting that many teams perform well. Lastly, the Date distribution shows fluctuations, indicating varying levels of activity throughout the month.

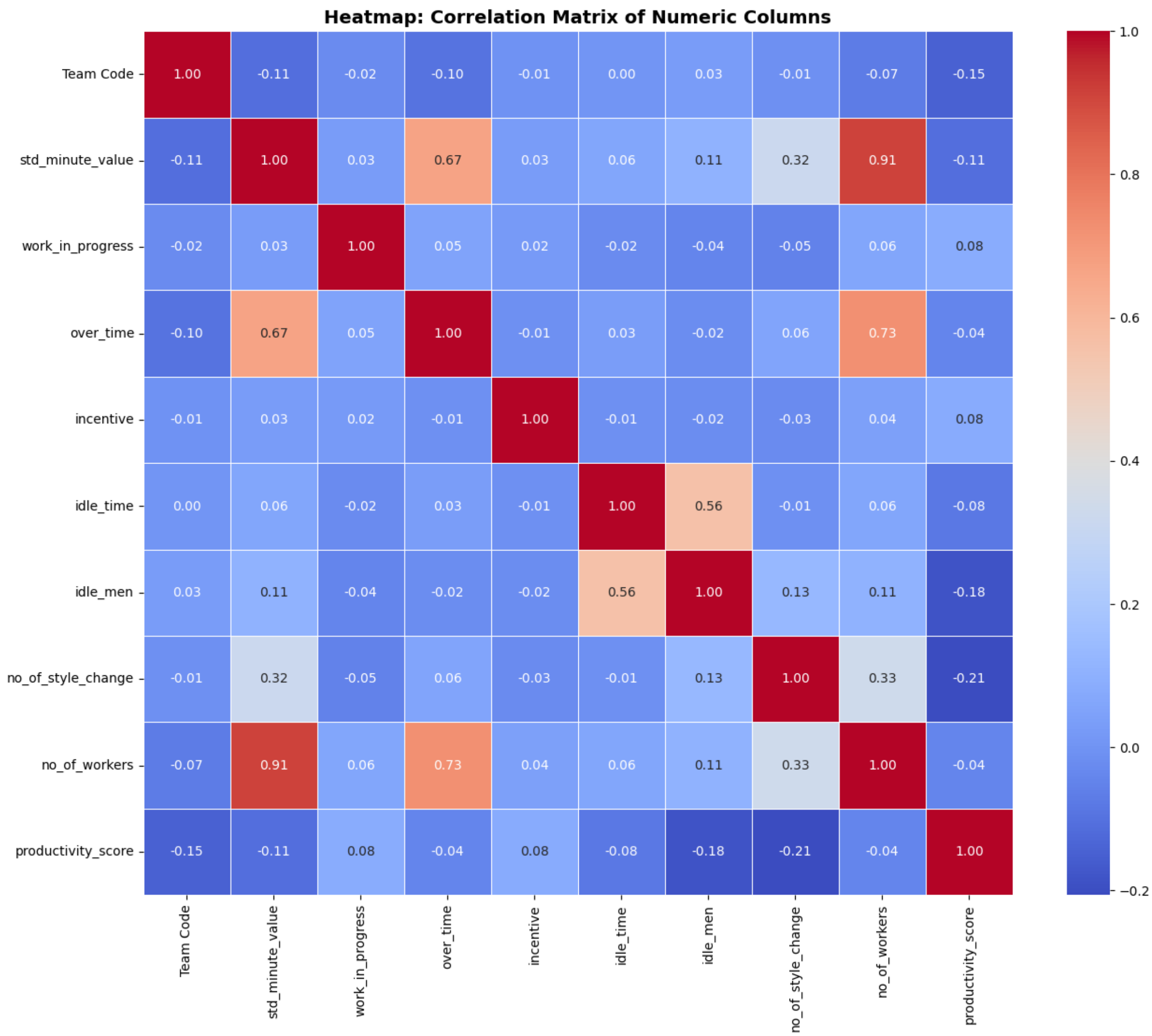
```
# Barplot

# Create subplots
fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(17, 8))

# Flatten the axes array for easier indexing
for i, ax in enumerate(axes.ravel()):
    if i < len(cat_cols):  # Ensure we don't run into an error if
cat_cols has fewer than 6 items
        avg_prod = df.groupby(cat_cols[i])
['productivity_score'].mean()
        avg_prod.plot.bar(ax=ax, color=sns.color_palette('Set2',
n_colors=len(avg_prod)))
```

```
        ax.set_title(f"Average Productivity by {cat_cols[i].title()}")
        ax.set_xlabel(cat_cols[i].title())
        ax.set_ylabel("Average Actual Productivity")
    else:
        ax.axis('off')  # Hide any unused subplots

# Adjust layout
fig.tight_layout()
plt.show()
```



The count plots reveal important insights across various categorical features. In the **quarter** distribution, Quarter 1 has the highest entries, followed by Quarter 2, indicating a concentration of activities in the first half of the year. The **day** plot shows that Sunday and Thursday are the busiest, while Saturday and Monday have lower counts. For **months**, January and February are notably more active than March. In terms of **year**, 2015 has significantly more entries than 2016, suggesting a higher volume of data collection in that year. The **team code** counts are balanced, with Team 8 slightly leading, and the **number of style changes** overwhelmingly reflects no changes, indicating infrequent style alterations. Overall, these patterns offer valuable insights for operational and productivity strategies in the garment industry.

```
# Heatmap
numeric_columns = df.select_dtypes(include=['int64',
'float64']).columns
correlation_matrix = df[numeric_columns].corr()

plt.figure(figsize=(15, 12))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
fmt='.2f', linewidths=0.5)
plt.title('Heatmap: Correlation Matrix of Numeric Columns',
fontsize=14, fontweight='bold')
plt.show()
```



From the heapmap, it shows that there are significant relationship among the features in garment dataset. Strong correlation between no_of_workers and std_minute_value (0.91) indicate that task require more time tend to involve larger teams. Negative Correlation such as Team Code and productivity score (-0.15) implies higher team assignment may relate to lower productivity levels

```
# delete date month year
df = df.drop(columns=['Date', 'Month', 'Year'])
```

Encoding

```
quarter_map = {'Quarter1': 1, 'Quarter2': 2, 'Quarter3': 3,
'Quarter4': 4}
df['quarter'] = df['quarter'].map(quarter_map)
```

```
day_map = {
    'Monday': 0,
    'Tuesday': 1,
    'Wednesday': 2,
    'Thursday': 3,
    'Friday': 4,
    'Saturday': 5,
    'Sunday': 6,
}
df['day'] = df['day'].map(day_map)
```

Map the unique values in quarter and day columns to their corresponding numerical representations, this mapping method assign numerical values based on the unique categories present in each column.

```
encoder = OneHotEncoder(sparse_output=False, drop=None)
encoded_array = encoder.fit_transform(df[['Team Code']])

# Convert to integer type (if needed)
encoded_array = encoded_array.astype(int)

# Get column names
encoded_cols = encoder.get_feature_names_out(['Team Code'])

# Make it a DataFrame
df_encoded = pd.DataFrame(encoded_array, columns=encoded_cols,
index=df.index)

# Combine with original DataFrame
df_final = pd.concat([df.drop(columns=['Team Code']), df_encoded],
axis=1)
```

Then, I performed one-hot encoding for the Team Code because it contains discrete categorical values ranging from 1 to 12. One hot encoding effectively converts these categories into binary columns, which help to ensure the team code are treated as distinct categories, improving model's ability to learn patterns associated with each team.

Target Features

The goal is predicting productivity score each team in garment company.

Target Feature : "productivity_score"

```
# Split Target Features
x = df_final.drop(['productivity_score'], axis=1)  # Drop the target
column from X
y = df_final['productivity_score']  # Target variable y
```

```
x.head()
```

{"type":"dataframe","variable_name":"x"}

Since "productivity_score" is the target variable, it is removed from x and assigned to y for model training.

Split Data

Step 1 (test_size=0.3):

70% of the data is used for the train set.

30% of the data is used for the temporary set (x_temp, y_temp).

Step 2 (test_size=1/3):

From the 30% temporary data, 1/3 becomes the test set (which is 10% of the total data).

2/3 becomes the validation set (which is 20% of the total data).

So, with these two splits:

70% of the data is used for training.

20% of the data is used for validation.

10% of the data is used for testing.

```
# Split into train, validation, and test sets
x_train, x_temp, y_train, y_temp = train_test_split(x, y,
test_size=0.3, random_state=42)
x_val, x_test, y_val, y_test = train_test_split(x_temp, y_temp,
test_size=1/3, random_state=42)
```

Next, split the data into training, validation, and testing sets to ensure proper model evaluation. First, 80% of the data is allocated to the training set (x_train, y_train), while the remaining 20% is used as the test set (x_test, y_test). Then, 25% of the training set is further split into a validation set (x_val, y_val), ensuring a well-balanced dataset for effective model training, hyperparameter tuning, and final evaluation.

```
numeric_cols = x_train.select_dtypes(include=['int64',
'float64']).columns
categorical_cols = x_train.select_dtypes(include=['object']).columns
```

Scaling

```
scaler = StandardScaler()
x_train_scaled_num = scaler.fit_transform(x_train[numeric_cols])
x_val_scaled_num = scaler.transform(x_val[numeric_cols])
x_test_scaled_num = scaler.transform(x_test[numeric_cols])

x_train_scaled = pd.DataFrame(x_train_scaled_num,
columns=numeric_cols, index=x_train.index)
x_val_scaled = pd.DataFrame(x_val_scaled_num, columns=numeric_cols,
index=x_val.index)
x_test_scaled = pd.DataFrame(x_test_scaled_num, columns=numeric_cols,
index=x_test.index)

x_train_scaled = pd.concat([x_train_scaled,
x_train[categorical_cols]], axis=1)
x_val_scaled = pd.concat([x_val_scaled, x_val[categorical_cols]],
axis=1)
x_test_scaled = pd.concat([x_test_scaled, x_test[categorical_cols]],
axis=1)
```

In this project, I employed StandardScaler to preprocess the input features for the Artificial Neural Network (ANN) aimed at predicting productivity. The decision was based on several key factors:

1. Handling Outliers: The dataset contains various outliers, and using MinMaxScaler could compress the scale of non-outlier data, negatively impacting the model. StandardScaler, which normalizes features based on their mean and standard deviation, is less sensitive to extreme values.

2. Addressing Skewed Distributions: Many features are not uniformly distributed and exhibit skewness. MinMaxScaler assumes a uniform spread, which isn't suitable here, while StandardScaler effectively centers the data around zero and scales it according to its spread.

3. Enhancing Neural Network Stability and Performance: Standardization leads to faster and more reliable convergence for neural networks. By ensuring inputs have a mean of 0 and a standard deviation of 1, each neuron receives data on a consistent scale, resulting in more stable gradient updates during training.

With these considerations, StandardScaler is the optimal choice for scaling in this scenario, promoting robust model performance and efficient training.

Building Tensor Dataset

```
train_ds =
tf.data.Dataset.from_tensor_slices((x_train_scaled,y_train)).batch(32)
.shuffle(10)
```

```
test_ds =
tf.data.Dataset.from_tensor_slices((x_test_scaled,y_test)).batch(32)
val_ds =
tf.data.Dataset.from_tensor_slices((x_val_scaled,y_val)).batch(32)

# Print x_train shape
print(x_train_scaled.shape)

(807, 22)

train_ds

<_ShuffleDataset element_spec=(TensorSpec(shape=(None, 22),
dtype=tf.float64, name=None), TensorSpec(shape=(None,),
dtype=tf.float64, name=None))>
```

Baseline Model

A baseline model is a starting point in a project. It is a simple model that helps us to know how weel basic methods perform.

Sequential

```
model = tf.keras.Sequential([
    Dense(128, activation='relu', input_shape=(22,)), # Neuron minimal
2X from input
    Dense(64, activation='relu'),
    Dense(1, activation='linear')
])
```

The first model is a Sequential model with three hidden layers. The first layer has 128 neurons, followed by a second layer with 64 neurons, both using ReLU activation to capture complex patterns in the data.

```
model.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss='mean_squared_error',
    metrics=['mean_absolute_error', 'mean_squared_error']
)

# Model Summary
model.summary()

Model: "sequential"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━
━━━━━┓
┃ Layer (type)                     ┃ Output Shape           ┃
```

```
Param # ┃
┣━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╋━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╋━━━━━━━━━━━━━
━━━━━┫
│ dense (Dense)                             │ (None, 128)                    │
2,944 │
├──────────────────────────────────────────┼───────────────────────────────┼─────────────
─────┤
│ dense_1 (Dense)                           │ (None, 64)                     │
8,256 │
├──────────────────────────────────────────┼───────────────────────────────┼─────────────
─────┤
│ dense_2 (Dense)                           │ (None, 1)                      │
65 │
└──────────────────────────────────────────┴───────────────────────────────┴─────────────
─────┘

 Total params: 11,265 (44.00 KB)

 Trainable params: 11,265 (44.00 KB)

 Non-trainable params: 0 (0.00 B)
```

The minimum required number of epochs is 10, but in this case, I used 50 epochs to give the model more time to learn and potentially achieve better performance.

```
# Train the model
history = model.fit(x_train_scaled, y_train,
                    validation_data=(x_val_scaled, y_val),
                    epochs=50,
                    batch_size=32,
                    verbose=1)

Epoch 1/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 5s 64ms/step - loss: 5358.6362 -
mean_absolute_error: 71.0948 - mean_squared_error: 5358.6362 -
val_loss: 4651.6113 - val_mean_absolute_error: 66.2169 -
val_mean_squared_error: 4651.6113
Epoch 2/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - loss: 4678.4561 -
mean_absolute_error: 65.9596 - mean_squared_error: 4678.4561 -
val_loss: 3248.8472 - val_mean_absolute_error: 54.5327 -
val_mean_squared_error: 3248.8472
Epoch 3/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 2986.9148 -
mean_absolute_error: 51.5474 - mean_squared_error: 2986.9148 -
val_loss: 1464.6555 - val_mean_absolute_error: 35.1155 -
val_mean_squared_error: 1464.6555
Epoch 4/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - loss: 1168.5156 -
mean_absolute_error: 30.5262 - mean_squared_error: 1168.5156 -
```

```
val_loss: 415.9604 - val_mean_absolute_error: 17.6726 -
val_mean_squared_error: 415.9604
Epoch 5/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 357.2380 -
mean_absolute_error: 15.6365 - mean_squared_error: 357.2380 -
val_loss: 297.3177 - val_mean_absolute_error: 13.2375 -
val_mean_squared_error: 297.3177
Epoch 6/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - loss: 293.4005 -
mean_absolute_error: 13.2418 - mean_squared_error: 293.4005 -
val_loss: 272.6161 - val_mean_absolute_error: 12.5761 -
val_mean_squared_error: 272.6161
Epoch 7/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 292.6281 -
mean_absolute_error: 13.2627 - mean_squared_error: 292.6281 -
val_loss: 262.5137 - val_mean_absolute_error: 12.4080 -
val_mean_squared_error: 262.5137
Epoch 8/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 282.7032 -
mean_absolute_error: 12.9848 - mean_squared_error: 282.7032 -
val_loss: 258.5030 - val_mean_absolute_error: 12.1534 -
val_mean_squared_error: 258.5030
Epoch 9/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - loss: 247.2510 -
mean_absolute_error: 12.0886 - mean_squared_error: 247.2510 -
val_loss: 252.7126 - val_mean_absolute_error: 11.9085 -
val_mean_squared_error: 252.7126
Epoch 10/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 280.4243 -
mean_absolute_error: 12.7915 - mean_squared_error: 280.4243 -
val_loss: 251.5667 - val_mean_absolute_error: 11.8845 -
val_mean_squared_error: 251.5667
Epoch 11/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 256.1978 -
mean_absolute_error: 12.3029 - mean_squared_error: 256.1978 -
val_loss: 249.9779 - val_mean_absolute_error: 11.7245 -
val_mean_squared_error: 249.9779
Epoch 12/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - loss: 223.1900 -
mean_absolute_error: 11.3838 - mean_squared_error: 223.1900 -
val_loss: 246.1589 - val_mean_absolute_error: 11.6395 -
val_mean_squared_error: 246.1589
Epoch 13/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - loss: 233.5306 -
mean_absolute_error: 11.5611 - mean_squared_error: 233.5306 -
val_loss: 243.6043 - val_mean_absolute_error: 11.6043 -
val_mean_squared_error: 243.6043
Epoch 14/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - loss: 239.5105 -
```

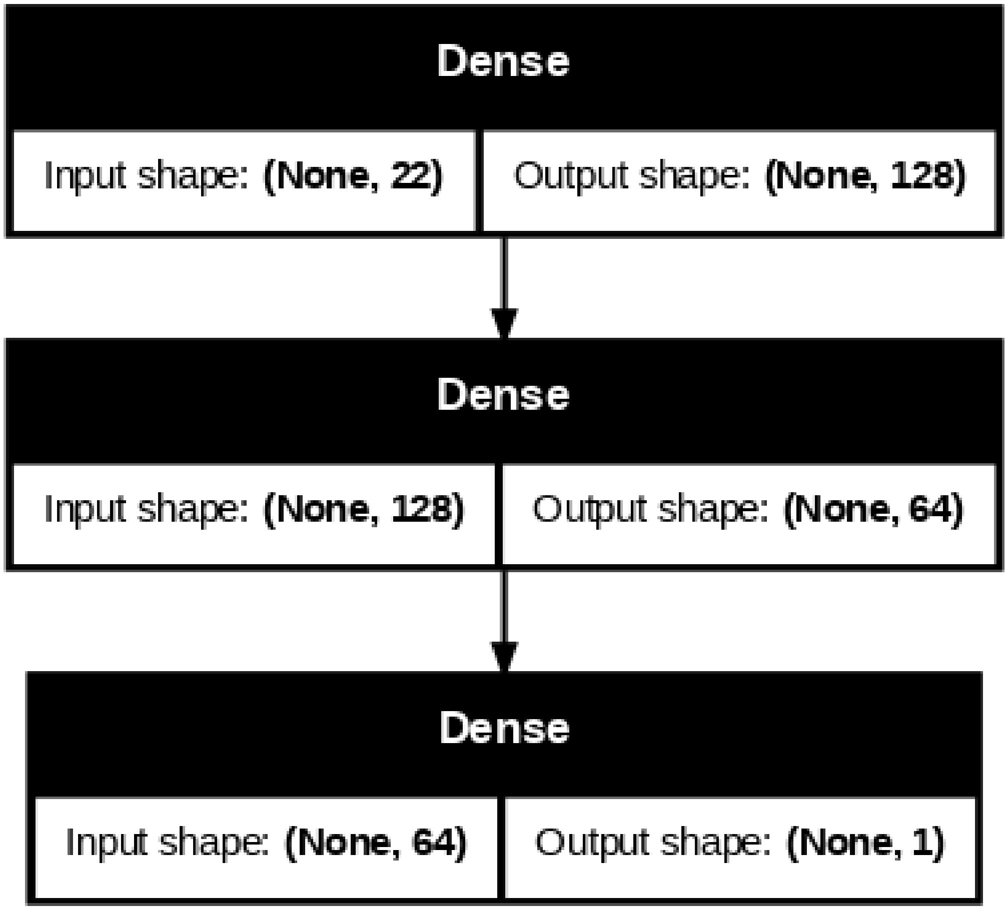
```
mean_absolute_error: 11.9045 - mean_squared_error: 239.5105 -
val_loss: 241.5325 - val_mean_absolute_error: 11.5142 -
val_mean_squared_error: 241.5325
Epoch 15/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 225.6774 -
mean_absolute_error: 11.5201 - mean_squared_error: 225.6774 -
val_loss: 241.7192 - val_mean_absolute_error: 11.4719 -
val_mean_squared_error: 241.7192
Epoch 16/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 207.6405 -
mean_absolute_error: 11.0020 - mean_squared_error: 207.6405 -
val_loss: 239.9426 - val_mean_absolute_error: 11.4519 -
val_mean_squared_error: 239.9426
Epoch 17/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 231.7247 -
mean_absolute_error: 11.7652 - mean_squared_error: 231.7247 -
val_loss: 238.0210 - val_mean_absolute_error: 11.2842 -
val_mean_squared_error: 238.0210
Epoch 18/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - loss: 224.5269 -
mean_absolute_error: 11.2000 - mean_squared_error: 224.5269 -
val_loss: 236.8561 - val_mean_absolute_error: 11.3659 -
val_mean_squared_error: 236.8561
Epoch 19/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - loss: 228.3874 -
mean_absolute_error: 11.6245 - mean_squared_error: 228.3874 -
val_loss: 238.9326 - val_mean_absolute_error: 11.2855 -
val_mean_squared_error: 238.9326
Epoch 20/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - loss: 214.1136 -
mean_absolute_error: 10.9041 - mean_squared_error: 214.1136 -
val_loss: 236.8804 - val_mean_absolute_error: 11.2621 -
val_mean_squared_error: 236.8804
Epoch 21/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - loss: 212.7799 -
mean_absolute_error: 10.8769 - mean_squared_error: 212.7799 -
val_loss: 231.9994 - val_mean_absolute_error: 11.1343 -
val_mean_squared_error: 231.9994
Epoch 22/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 222.4274 -
mean_absolute_error: 11.2558 - mean_squared_error: 222.4274 -
val_loss: 233.1278 - val_mean_absolute_error: 11.1454 -
val_mean_squared_error: 233.1278
Epoch 23/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 214.4915 -
mean_absolute_error: 11.0484 - mean_squared_error: 214.4915 -
val_loss: 234.3152 - val_mean_absolute_error: 11.1391 -
val_mean_squared_error: 234.3152
Epoch 24/50
```

```
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 212.6380 -
mean_absolute_error: 11.2341 - mean_squared_error: 212.6380 -
val_loss: 232.0361 - val_mean_absolute_error: 11.1735 -
val_mean_squared_error: 232.0361
Epoch 25/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - loss: 174.7941 -
mean_absolute_error: 9.9115 - mean_squared_error: 174.7941 - val_loss:
230.3878 - val_mean_absolute_error: 11.1396 - val_mean_squared_error:
230.3878
Epoch 26/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 226.4064 -
mean_absolute_error: 11.5877 - mean_squared_error: 226.4064 -
val_loss: 234.8536 - val_mean_absolute_error: 11.1456 -
val_mean_squared_error: 234.8536
Epoch 27/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 214.7414 -
mean_absolute_error: 10.8571 - mean_squared_error: 214.7414 -
val_loss: 235.1540 - val_mean_absolute_error: 11.1974 -
val_mean_squared_error: 235.1540
Epoch 28/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 205.0749 -
mean_absolute_error: 10.6132 - mean_squared_error: 205.0749 -
val_loss: 232.0501 - val_mean_absolute_error: 11.1346 -
val_mean_squared_error: 232.0501
Epoch 29/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 202.8737 -
mean_absolute_error: 10.6755 - mean_squared_error: 202.8737 -
val_loss: 232.5561 - val_mean_absolute_error: 11.1213 -
val_mean_squared_error: 232.5561
Epoch 30/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 194.5984 -
mean_absolute_error: 10.4603 - mean_squared_error: 194.5984 -
val_loss: 233.1612 - val_mean_absolute_error: 11.2009 -
val_mean_squared_error: 233.1612
Epoch 31/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 206.9041 -
mean_absolute_error: 10.8685 - mean_squared_error: 206.9041 -
val_loss: 232.9809 - val_mean_absolute_error: 11.1372 -
val_mean_squared_error: 232.9809
Epoch 32/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 194.3673 -
mean_absolute_error: 10.6117 - mean_squared_error: 194.3673 -
val_loss: 232.1484 - val_mean_absolute_error: 11.0373 -
val_mean_squared_error: 232.1484
Epoch 33/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 199.3568 -
mean_absolute_error: 10.4863 - mean_squared_error: 199.3568 -
val_loss: 232.0366 - val_mean_absolute_error: 11.3342 -
val_mean_squared_error: 232.0366
```

```
Epoch 34/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 209.0277 -
mean_absolute_error: 10.9494 - mean_squared_error: 209.0277 -
val_loss: 232.1294 - val_mean_absolute_error: 11.0627 -
val_mean_squared_error: 232.1294
Epoch 35/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 171.7456 -
mean_absolute_error: 9.7488 - mean_squared_error: 171.7456 - val_loss:
231.6498 - val_mean_absolute_error: 11.1930 - val_mean_squared_error:
231.6498
Epoch 36/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 183.6318 -
mean_absolute_error: 10.3989 - mean_squared_error: 183.6318 -
val_loss: 231.7758 - val_mean_absolute_error: 11.1517 -
val_mean_squared_error: 231.7758
Epoch 37/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 184.4121 -
mean_absolute_error: 10.2515 - mean_squared_error: 184.4121 -
val_loss: 230.2932 - val_mean_absolute_error: 11.1040 -
val_mean_squared_error: 230.2932
Epoch 38/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 209.8366 -
mean_absolute_error: 10.8548 - mean_squared_error: 209.8366 -
val_loss: 229.6699 - val_mean_absolute_error: 10.9702 -
val_mean_squared_error: 229.6699
Epoch 39/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 193.0451 -
mean_absolute_error: 10.4089 - mean_squared_error: 193.0451 -
val_loss: 231.5444 - val_mean_absolute_error: 11.1093 -
val_mean_squared_error: 231.5444
Epoch 40/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 207.1039 -
mean_absolute_error: 10.6506 - mean_squared_error: 207.1039 -
val_loss: 234.1166 - val_mean_absolute_error: 11.1125 -
val_mean_squared_error: 234.1166
Epoch 41/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 176.0679 -
mean_absolute_error: 9.9634 - mean_squared_error: 176.0679 - val_loss:
232.9312 - val_mean_absolute_error: 11.0181 - val_mean_squared_error:
232.9312
Epoch 42/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 182.3673 -
mean_absolute_error: 10.0537 - mean_squared_error: 182.3673 -
val_loss: 231.9330 - val_mean_absolute_error: 11.0332 -
val_mean_squared_error: 231.9330
Epoch 43/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 173.7043 -
mean_absolute_error: 9.7998 - mean_squared_error: 173.7043 - val_loss:
227.8138 - val_mean_absolute_error: 11.0959 - val_mean_squared_error:
```

```
227.8138
Epoch 44/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 178.6293 -
mean_absolute_error: 10.0558 - mean_squared_error: 178.6293 -
val_loss: 232.7407 - val_mean_absolute_error: 10.9020 -
val_mean_squared_error: 232.7407
Epoch 45/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 199.9169 -
mean_absolute_error: 10.4984 - mean_squared_error: 199.9169 -
val_loss: 231.7280 - val_mean_absolute_error: 11.0270 -
val_mean_squared_error: 231.7280
Epoch 46/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 168.5697 -
mean_absolute_error: 9.6296 - mean_squared_error: 168.5697 - val_loss:
232.0538 - val_mean_absolute_error: 11.2452 - val_mean_squared_error:
232.0538
Epoch 47/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 199.2104 -
mean_absolute_error: 10.7417 - mean_squared_error: 199.2104 -
val_loss: 234.8258 - val_mean_absolute_error: 10.9460 -
val_mean_squared_error: 234.8258
Epoch 48/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 175.0897 -
mean_absolute_error: 9.9881 - mean_squared_error: 175.0897 - val_loss:
230.7715 - val_mean_absolute_error: 11.1372 - val_mean_squared_error:
230.7715
Epoch 49/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 185.4055 -
mean_absolute_error: 10.2561 - mean_squared_error: 185.4055 -
val_loss: 234.3742 - val_mean_absolute_error: 10.9478 -
val_mean_squared_error: 234.3742
Epoch 50/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 174.3193 -
mean_absolute_error: 9.7612 - mean_squared_error: 174.3193 - val_loss:
232.0248 - val_mean_absolute_error: 10.9970 - val_mean_squared_error:
232.0248
```

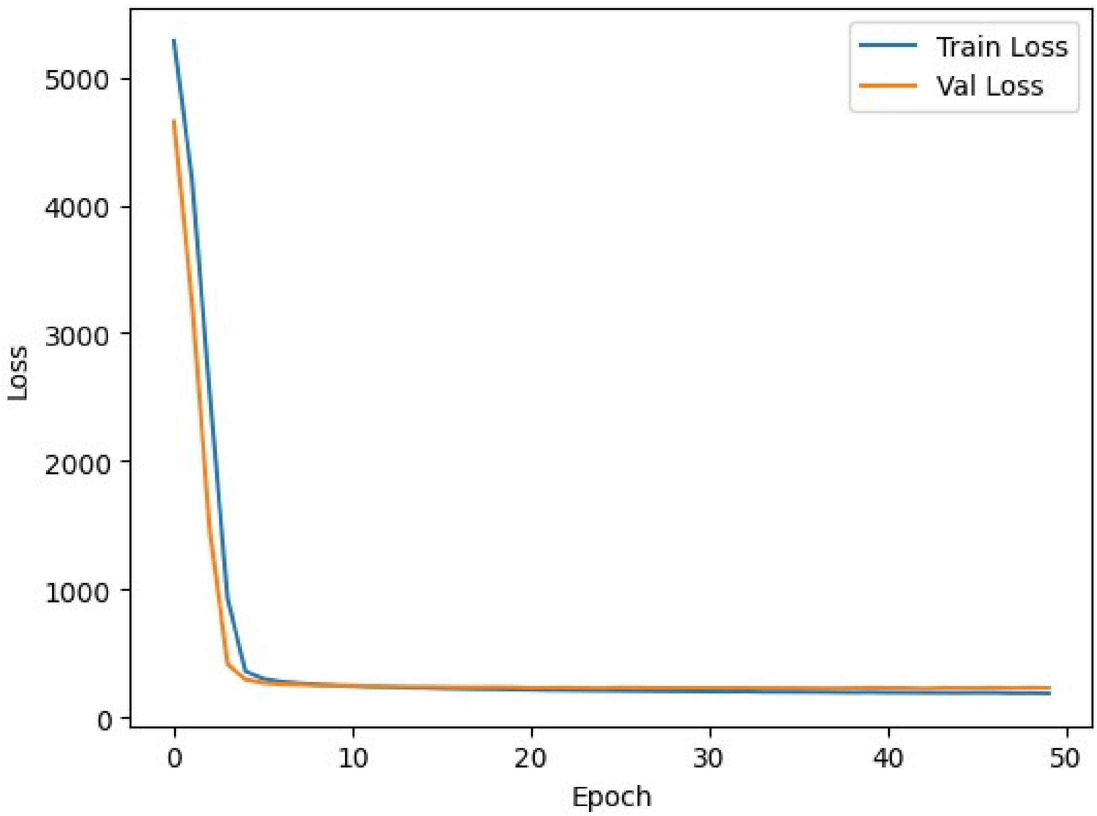
```
plot_model(model,show_shapes=True, dpi=75)
```




```
# Prediction
y_pred = model.predict(x_test_scaled)

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
```

Evaluate

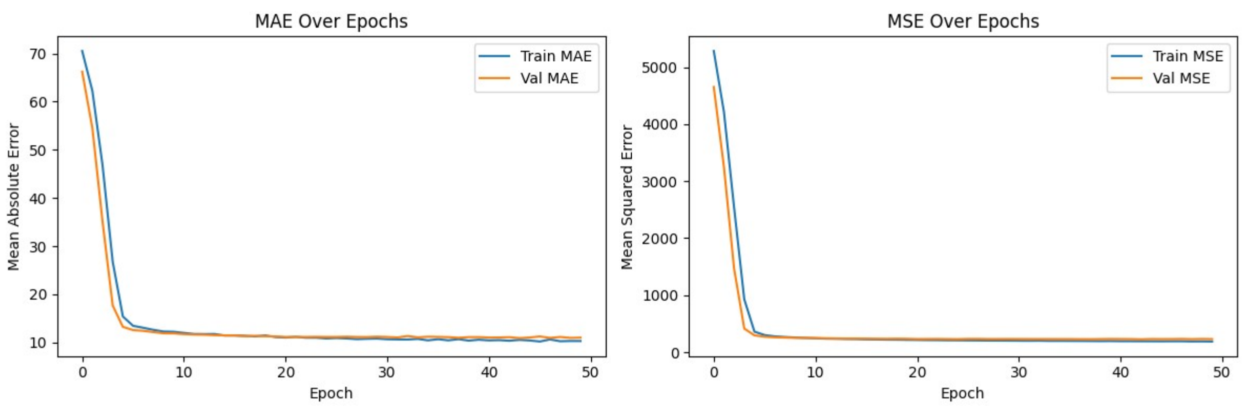
```
train_loss = history.history['loss']
val_loss = history.history['val_loss']
plt.plot(train_loss,label="Train Loss")
plt.plot(val_loss,label='Val Loss')
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



The loss curve shows that both training and validation loss dropped significantly in the first few epochs and then gradually stabilized, indicating the model learned quickly without overfitting. The small gap between train and validation loss suggests good generalization.
However, since the loss decreased very rapidly, it might also indicate that the model is too simple for the dataset and may not be capturing more complex patterns.

```
# 3 Metric Analysis
R2 = r2_score(y_test, y_pred)
MAE = mean_absolute_error(y_test, y_pred)
MSE = mean_squared_error(y_test, y_pred)

print("R² Score=", R2)
print("Mean Absolute Error=", MAE)
print("Mean Squared Error=", MSE)

R² Score= 0.12921236289162408
Mean Absolute Error= 12.86850448937252
Mean Squared Error= 308.3501299756072

# Plot Mean Absolute Error (MAE)
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['mean_absolute_error'], label='Train MAE')
plt.plot(history.history['val_mean_absolute_error'], label='Val MAE')
```

```
plt.xlabel('Epoch')
plt.ylabel('Mean Absolute Error')
plt.title('MAE Over Epochs')
plt.legend()

# Plot Mean Squared Error (MSE)
plt.subplot(1, 2, 2)
plt.plot(history.history['mean_squared_error'], label='Train MSE')
plt.plot(history.history['val_mean_squared_error'], label='Val MSE')
plt.xlabel('Epoch')
plt.ylabel('Mean Squared Error')
plt.title('MSE Over Epochs')
plt.legend()

plt.tight_layout()
plt.show()
```



The model learned quickly, as shown by the loss curves stabilizing without overfitting. However, the final metrics are

1. $R^2 = 0.129$,
2. MAE = 12.87, and
3. MSE = 308.35

indicate that the model's predictive power is low, explaining only 12.9% of the variance. This suggests the model is too simple for the dataset, and improvements are needed. High MAE and MSE value also indicate that the still need more modification to predict accurately.

Functional API

```
from tensorflow.keras.layers import Input

inputs = Input(shape=(x_train_scaled.shape[1],))

x = Dense(128, activation='relu')(inputs)
x = Dense(64, activation='relu')(x)
outputs = Dense(1, activation='linear')(x)
```

The second model is built using the Functional API with three layers. The first layer has 128 neurons with ReLU activation, followed by a second layer with 64 neurons, also using ReLU activation to capture more complex patterns.

```
model_1 = Model(inputs=inputs, outputs=outputs)

model_1.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss='mean_squared_error',
    metrics=['mean_absolute_error', 'mean_squared_error']
)

# Model Summary
model_1.summary()
```

```
Model: "functional_1"
```

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 22)	0
dense_3 (Dense)	(None, 128)	2,944
dense_4 (Dense)	(None, 64)	8,256
dense_5 (Dense)	(None, 1)	65

```
 Total params: 11,265 (44.00 KB)

 Trainable params: 11,265 (44.00 KB)

 Non-trainable params: 0 (0.00 B)
```

The minimum required number of epochs in this model is also 10, but in this case, I also used 50 epochs to give the model more time to learn and potentially achieve better performance.

```
# Train the model
history = model_1.fit(x_train_scaled, y_train,
```

```
                    validation_data=(x_val_scaled, y_val),
                    epochs=50,
                    batch_size=32,
                    verbose=1)
```

```
Epoch 1/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - loss: 5449.6519 -
mean_absolute_error: 71.6416 - mean_squared_error: 5449.6519 -
val_loss: 4719.4854 - val_mean_absolute_error: 66.7400 -
val_mean_squared_error: 4719.4854
Epoch 2/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 4672.2324 -
mean_absolute_error: 66.0836 - mean_squared_error: 4672.2324 -
val_loss: 3360.1025 - val_mean_absolute_error: 55.5893 -
val_mean_squared_error: 3360.1025
Epoch 3/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 2949.0913 -
mean_absolute_error: 51.2472 - mean_squared_error: 2949.0913 -
val_loss: 1486.9247 - val_mean_absolute_error: 35.4299 -
val_mean_squared_error: 1486.9247
Epoch 4/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1160.7073 -
mean_absolute_error: 30.6357 - mean_squared_error: 1160.7073 -
val_loss: 388.8639 - val_mean_absolute_error: 16.6818 -
val_mean_squared_error: 388.8639
Epoch 5/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 383.6802 -
mean_absolute_error: 15.9054 - mean_squared_error: 383.6802 -
val_loss: 288.8310 - val_mean_absolute_error: 12.9639 -
val_mean_squared_error: 288.8310
Epoch 6/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 281.7173 -
mean_absolute_error: 12.8925 - mean_squared_error: 281.7173 -
val_loss: 272.9529 - val_mean_absolute_error: 12.5817 -
val_mean_squared_error: 272.9529
Epoch 7/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 302.8765 -
mean_absolute_error: 13.4368 - mean_squared_error: 302.8765 -
val_loss: 261.6469 - val_mean_absolute_error: 12.4264 -
val_mean_squared_error: 261.6469
Epoch 8/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 278.5079 -
mean_absolute_error: 13.2208 - mean_squared_error: 278.5079 -
val_loss: 257.1906 - val_mean_absolute_error: 12.2425 -
val_mean_squared_error: 257.1906
Epoch 9/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 255.0793 -
mean_absolute_error: 12.3952 - mean_squared_error: 255.0793 -
val_loss: 252.3163 - val_mean_absolute_error: 11.9487 -
val_mean_squared_error: 252.3163
```

```
Epoch 10/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 268.9190 -
mean_absolute_error: 12.8541 - mean_squared_error: 268.9190 -
val_loss: 251.0285 - val_mean_absolute_error: 11.8848 -
val_mean_squared_error: 251.0285
Epoch 11/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 253.7633 -
mean_absolute_error: 12.1895 - mean_squared_error: 253.7633 -
val_loss: 249.3957 - val_mean_absolute_error: 11.7386 -
val_mean_squared_error: 249.3957
Epoch 12/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 215.2738 -
mean_absolute_error: 11.3137 - mean_squared_error: 215.2738 -
val_loss: 244.9789 - val_mean_absolute_error: 11.5944 -
val_mean_squared_error: 244.9789
Epoch 13/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 239.7575 -
mean_absolute_error: 11.9111 - mean_squared_error: 239.7575 -
val_loss: 244.9755 - val_mean_absolute_error: 11.5975 -
val_mean_squared_error: 244.9755
Epoch 14/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 226.6929 -
mean_absolute_error: 11.3882 - mean_squared_error: 226.6929 -
val_loss: 244.3934 - val_mean_absolute_error: 11.4206 -
val_mean_squared_error: 244.3934
Epoch 15/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 212.6314 -
mean_absolute_error: 11.0316 - mean_squared_error: 212.6314 -
val_loss: 241.6690 - val_mean_absolute_error: 11.3856 -
val_mean_squared_error: 241.6690
Epoch 16/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 223.6355 -
mean_absolute_error: 11.5694 - mean_squared_error: 223.6355 -
val_loss: 238.4850 - val_mean_absolute_error: 11.2654 -
val_mean_squared_error: 238.4850
Epoch 17/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 221.3133 -
mean_absolute_error: 11.2275 - mean_squared_error: 221.3133 -
val_loss: 237.2972 - val_mean_absolute_error: 11.2516 -
val_mean_squared_error: 237.2972
Epoch 18/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 221.4541 -
mean_absolute_error: 11.3452 - mean_squared_error: 221.4541 -
val_loss: 238.7753 - val_mean_absolute_error: 11.3478 -
val_mean_squared_error: 238.7753
Epoch 19/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 211.2988 -
mean_absolute_error: 11.1453 - mean_squared_error: 211.2988 -
val_loss: 235.6654 - val_mean_absolute_error: 11.2473 -
```

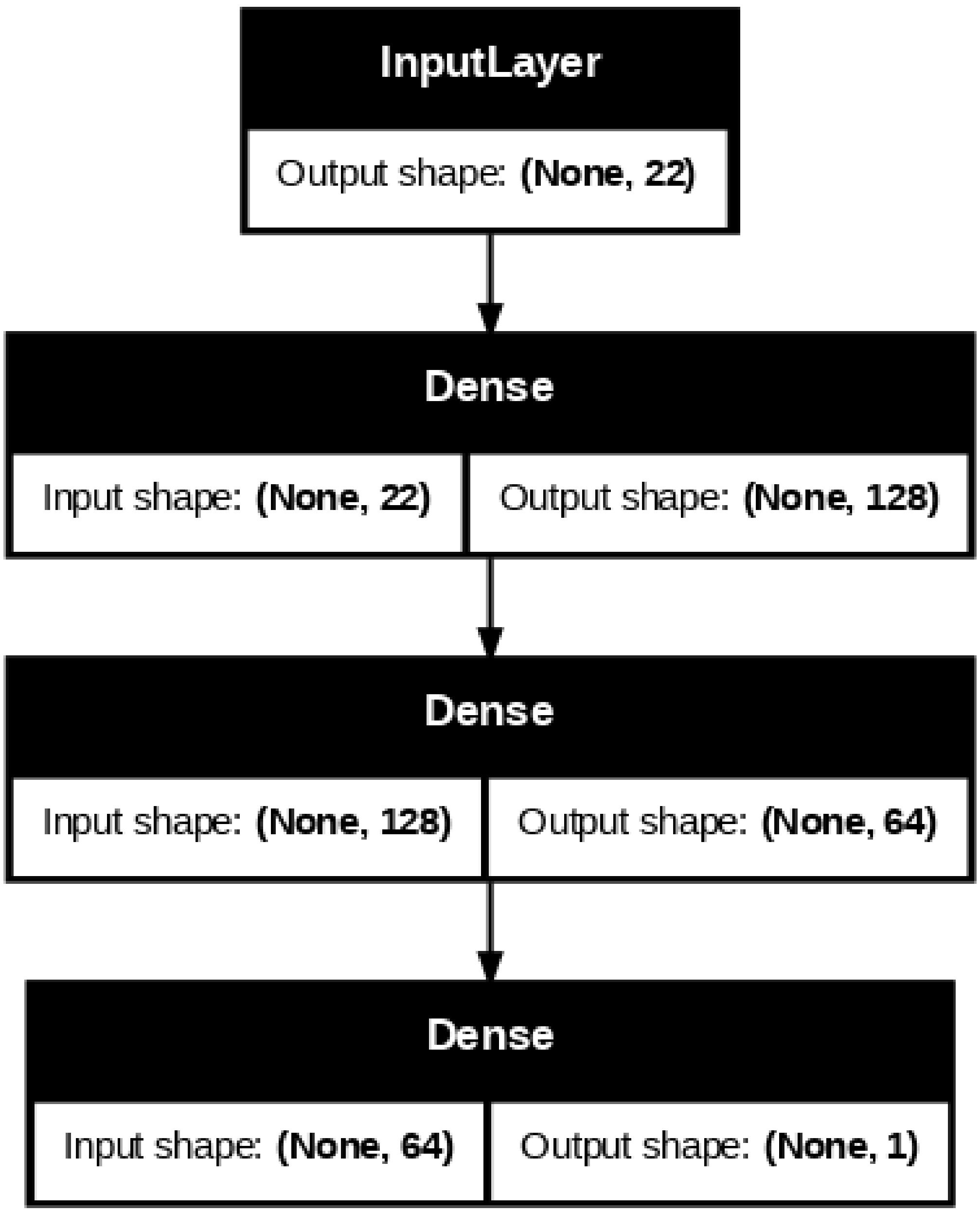
```
val_mean_squared_error: 235.6654
Epoch 20/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 206.2696 -
mean_absolute_error: 10.9548 - mean_squared_error: 206.2696 -
val_loss: 236.5964 - val_mean_absolute_error: 11.0988 -
val_mean_squared_error: 236.5964
Epoch 21/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 197.1338 -
mean_absolute_error: 10.8700 - mean_squared_error: 197.1338 -
val_loss: 236.8905 - val_mean_absolute_error: 11.2747 -
val_mean_squared_error: 236.8905
Epoch 22/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 200.4566 -
mean_absolute_error: 10.9924 - mean_squared_error: 200.4566 -
val_loss: 237.7415 - val_mean_absolute_error: 11.2063 -
val_mean_squared_error: 237.7415
Epoch 23/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 229.6906 -
mean_absolute_error: 11.6454 - mean_squared_error: 229.6906 -
val_loss: 238.9200 - val_mean_absolute_error: 11.2707 -
val_mean_squared_error: 238.9200
Epoch 24/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 187.5894 -
mean_absolute_error: 10.3850 - mean_squared_error: 187.5894 -
val_loss: 236.9964 - val_mean_absolute_error: 11.3057 -
val_mean_squared_error: 236.9964
Epoch 25/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 192.1990 -
mean_absolute_error: 10.5181 - mean_squared_error: 192.1990 -
val_loss: 235.8731 - val_mean_absolute_error: 11.2080 -
val_mean_squared_error: 235.8731
Epoch 26/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 201.0883 -
mean_absolute_error: 10.8371 - mean_squared_error: 201.0883 -
val_loss: 235.9014 - val_mean_absolute_error: 11.1982 -
val_mean_squared_error: 235.9014
Epoch 27/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 214.3575 -
mean_absolute_error: 10.9688 - mean_squared_error: 214.3575 -
val_loss: 234.0916 - val_mean_absolute_error: 11.0751 -
val_mean_squared_error: 234.0916
Epoch 28/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 183.3612 -
mean_absolute_error: 10.2086 - mean_squared_error: 183.3612 -
val_loss: 235.3814 - val_mean_absolute_error: 11.1152 -
val_mean_squared_error: 235.3814
Epoch 29/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 185.6768 -
mean_absolute_error: 10.2622 - mean_squared_error: 185.6768 -
```

```
val_loss: 235.7099 - val_mean_absolute_error: 11.1809 -
val_mean_squared_error: 235.7099
Epoch 30/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 198.3161 -
mean_absolute_error: 10.6166 - mean_squared_error: 198.3161 -
val_loss: 236.4979 - val_mean_absolute_error: 11.2212 -
val_mean_squared_error: 236.4979
Epoch 31/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 196.0229 -
mean_absolute_error: 10.4616 - mean_squared_error: 196.0229 -
val_loss: 235.2513 - val_mean_absolute_error: 11.1368 -
val_mean_squared_error: 235.2513
Epoch 32/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 189.6390 -
mean_absolute_error: 10.4473 - mean_squared_error: 189.6390 -
val_loss: 235.2974 - val_mean_absolute_error: 11.0936 -
val_mean_squared_error: 235.2974
Epoch 33/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 192.6635 -
mean_absolute_error: 10.4926 - mean_squared_error: 192.6635 -
val_loss: 235.0291 - val_mean_absolute_error: 11.1484 -
val_mean_squared_error: 235.0291
Epoch 34/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 191.3441 -
mean_absolute_error: 10.5039 - mean_squared_error: 191.3441 -
val_loss: 236.7380 - val_mean_absolute_error: 11.1253 -
val_mean_squared_error: 236.7380
Epoch 35/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 181.9235 -
mean_absolute_error: 10.0873 - mean_squared_error: 181.9235 -
val_loss: 236.2333 - val_mean_absolute_error: 11.1987 -
val_mean_squared_error: 236.2333
Epoch 36/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 205.1174 -
mean_absolute_error: 11.0868 - mean_squared_error: 205.1174 -
val_loss: 236.7294 - val_mean_absolute_error: 11.0927 -
val_mean_squared_error: 236.7294
Epoch 37/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 193.8205 -
mean_absolute_error: 10.5269 - mean_squared_error: 193.8205 -
val_loss: 236.9320 - val_mean_absolute_error: 11.1058 -
val_mean_squared_error: 236.9320
Epoch 38/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 187.4594 -
mean_absolute_error: 10.1406 - mean_squared_error: 187.4594 -
val_loss: 238.5517 - val_mean_absolute_error: 11.1805 -
val_mean_squared_error: 238.5517
Epoch 39/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 202.6889 -
```

```
mean_absolute_error: 10.6678 - mean_squared_error: 202.6889 -
val_loss: 236.6911 - val_mean_absolute_error: 11.1205 -
val_mean_squared_error: 236.6911
Epoch 40/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 221.0382 -
mean_absolute_error: 11.2026 - mean_squared_error: 221.0382 -
val_loss: 239.5552 - val_mean_absolute_error: 11.1308 -
val_mean_squared_error: 239.5552
Epoch 41/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 193.4230 -
mean_absolute_error: 10.4609 - mean_squared_error: 193.4230 -
val_loss: 237.8185 - val_mean_absolute_error: 11.1688 -
val_mean_squared_error: 237.8185
Epoch 42/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 186.9290 -
mean_absolute_error: 10.1775 - mean_squared_error: 186.9290 -
val_loss: 236.7873 - val_mean_absolute_error: 11.2795 -
val_mean_squared_error: 236.7873
Epoch 43/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 177.7643 -
mean_absolute_error: 10.2073 - mean_squared_error: 177.7643 -
val_loss: 237.3213 - val_mean_absolute_error: 11.0702 -
val_mean_squared_error: 237.3213
Epoch 44/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 184.3974 -
mean_absolute_error: 10.2182 - mean_squared_error: 184.3974 -
val_loss: 237.2388 - val_mean_absolute_error: 11.0781 -
val_mean_squared_error: 237.2388
Epoch 45/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 177.1453 -
mean_absolute_error: 10.0129 - mean_squared_error: 177.1453 -
val_loss: 240.5054 - val_mean_absolute_error: 11.2322 -
val_mean_squared_error: 240.5054
Epoch 46/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 183.9633 -
mean_absolute_error: 10.0597 - mean_squared_error: 183.9633 -
val_loss: 239.3889 - val_mean_absolute_error: 11.1797 -
val_mean_squared_error: 239.3889
Epoch 47/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 185.9175 -
mean_absolute_error: 10.1802 - mean_squared_error: 185.9175 -
val_loss: 240.2192 - val_mean_absolute_error: 11.0950 -
val_mean_squared_error: 240.2192
Epoch 48/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 186.1756 -
mean_absolute_error: 10.1265 - mean_squared_error: 186.1756 -
val_loss: 237.3077 - val_mean_absolute_error: 11.1763 -
val_mean_squared_error: 237.3077
Epoch 49/50
```

```
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 196.4990 -
mean_absolute_error: 10.6038 - mean_squared_error: 196.4990 -
val_loss: 239.4495 - val_mean_absolute_error: 11.1059 -
val_mean_squared_error: 239.4495
Epoch 50/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 167.2845 -
mean_absolute_error: 9.6527 - mean_squared_error: 167.2845 - val_loss:
240.3244 - val_mean_absolute_error: 11.1592 - val_mean_squared_error:
240.3244
```

```
plot_model(model_1, show_shapes=True, dpi=75)
```

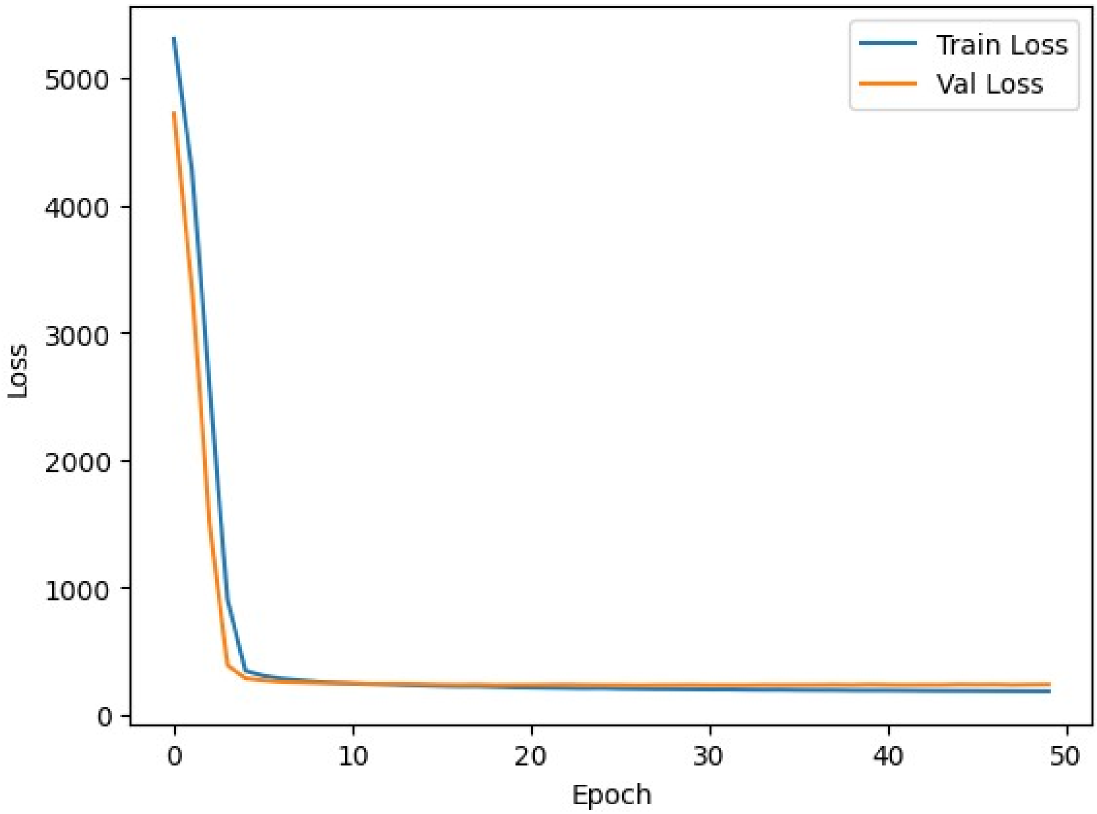


```
y_pred = model_1.predict(x_test_scaled)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
```

Evaluate

```
train_loss = history.history['loss']
val_loss = history.history['val_loss']
plt.plot(train_loss, label="Train Loss")
plt.plot(val_loss, label='Val Loss')
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



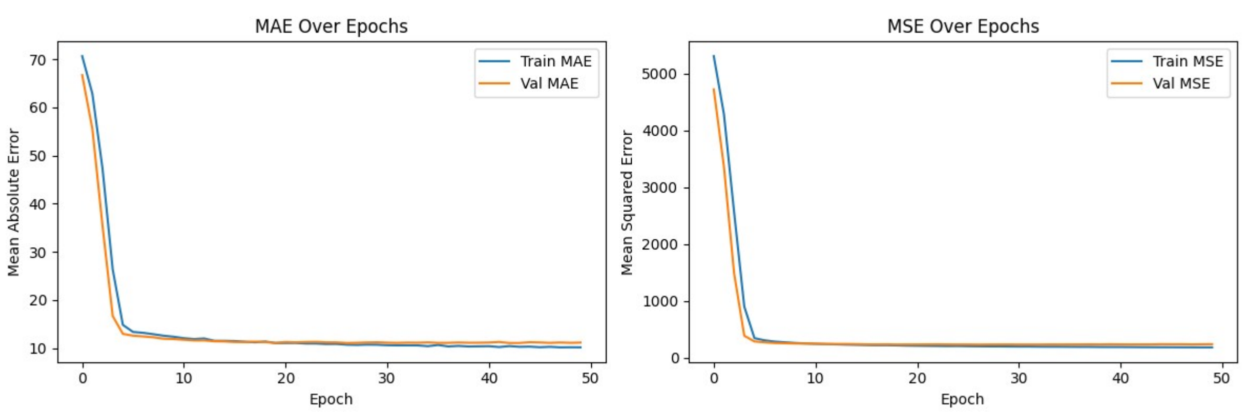
The loss curve for the functional model shows that both training and validation loss drop significantly, just like in the sequential model. The smaller gap between the two suggests better generalization. However, the quick decrease in the plot indicates that the model might be too simple for the data, which could mean it's not capturing all the necessary details.

```
R2 = r2_score(y_test, y_pred)
MAE = mean_absolute_error(y_test, y_pred)
MSE = mean_squared_error(y_test, y_pred)
```

```python
print("R² Score=", R2)
print("Mean Absolute Error=", MAE)
print("Mean Squared Error=", MSE)
```

```
R² Score= 0.10655987020032043
Mean Absolute Error= 13.075829557353057
Mean Squared Error= 316.3714876154902
```

```python
# Plot Mean Absolute Error (MAE)
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['mean_absolute_error'], label='Train MAE')
plt.plot(history.history['val_mean_absolute_error'], label='Val MAE')
plt.xlabel('Epoch')
plt.ylabel('Mean Absolute Error')
plt.title('MAE Over Epochs')
plt.legend()

# Plot Mean Squared Error (MSE)
plt.subplot(1, 2, 2)
plt.plot(history.history['mean_squared_error'], label='Train MSE')
plt.plot(history.history['val_mean_squared_error'], label='Val MSE')
plt.xlabel('Epoch')
plt.ylabel('Mean Squared Error')
plt.title('MSE Over Epochs')
plt.legend()

plt.tight_layout()
plt.show()
```



The Functional API model also learned quickly as it shown by the plot of loss curve stable and not overfitting. but the statistical result is

1. $R^2 = 0.106$,
2. MAE = 13.07, and
3. MSE = 316.37

which means the model's predictive capability is low, with an R^2 score of 0.106, indicating that it explains only 10.6% of the variance in the data. The high values of MSE and MAE further indicate that the model struggles to provide accurate predictions, as it has a substantial average error of approximately 13.07 units.

Comparison Both Baseline Model

Both the Sequential and Functional API models displayed similar learning curves with a rapid reduction in loss, indicating that they both learned from the training data quickly. However, the low values of R^2, high values of MAE, and MSE reveal that both models struggle to accurately predict productivity scores based on the data. Overall, the statistical results show that the Sequential model performs slightly better than the Functional API model, yielding more accurate predictions with lower error metrics.

In conclusion, both models appear to be too simplistic for the dataset, making it challenging to effectively predict productivity scores.

Modify Model

Next, I modified the ANN model to achieve better performance compared to the baseline model by adding Dropout for regularization to reduce overfitting, and BatchNormalization to stabilize and speed up the training process, resulting in more efficient learning and improved generalization.

Sequential

```
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.regularizers import l2

model_seq = Sequential([
    Dense(128, activation='elu', kernel_regularizer=l2(0.001), 
input_shape=(22,)),
    BatchNormalization(),
    Dropout(0.3),

    Dense(64, activation='elu', kernel_regularizer=l2(0.001)),
    BatchNormalization(),
    Dropout(0.3),

    Dense(64, activation='elu', kernel_regularizer=l2(0.001)),
    BatchNormalization(),
    Dropout(0.3),

    Dense(64, activation='elu', kernel_regularizer=l2(0.001)),
    BatchNormalization(),

    Dense(1, activation='linear')
])
```

To enhance the performance of the sequential model, I made several modifications. I replaced the activation function with the Exponential Linear Unit (ELU) for each layer. ELU is advantageous because it helps mitigate the risk of vanishing gradients and avoids the "dying ReLU" problem, where neurons become inactive during training. Unlike ReLU, ELU allows for negative values, maintaining a mean output closer to zero, which supports better gradient flow and faster convergence.

Additionally, I implemented L2 regularization with a strength of 0.001 to help prevent overfitting by penalizing large weights. Batch normalization is retained after each dense layer to stabilize learning and improve convergence further. Dropout layers are included to randomly disable a portion of neurons during training, further reducing the likelihood of overfitting. After experimenting with both activation functions, I found that using ELU yielded better results, with lower loss and improved accuracy during training and validation. Consequently, this modified structure aims to enhance the overall performance of the model while effectively addressing potential overfitting issues.

```
model_seq.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
    loss='mean_squared_error',
    metrics=['mean_absolute_error', 'mean_squared_error']
)
```

Also, using a learning rate of 0.001 for the optimizer helps to control the step size during training, allowing the model to converge more smoothly and efficiently.

```
# Model Summary
model_seq.summary()

Model: "sequential_1"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━
━━━━━┓
┃ Layer (type)                     ┃ Output Shape           ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━
━━━━━┩
│ dense_6 (Dense)                  │ (None, 128)            │
2,944 │
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ batch_normalization              │ (None, 128)            │
512 │
│ (BatchNormalization)             │                        │
│
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ dropout (Dropout)                │ (None, 128)            │
0 │
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
```

```
│ dense_7 (Dense)                   │ (None, 64)               │
8,256 │
├───────────────────────────────────┼──────────────────────────┼─────────────
───────┤
│ batch_normalization_1             │ (None, 64)               │
256 │
│ (BatchNormalization)              │                          │
│
├───────────────────────────────────┼──────────────────────────┼─────────────
───────┤
│ dropout_1 (Dropout)               │ (None, 64)               │
0 │
├───────────────────────────────────┼──────────────────────────┼─────────────
───────┤
│ dense_8 (Dense)                   │ (None, 64)               │
4,160 │
├───────────────────────────────────┼──────────────────────────┼─────────────
───────┤
│ batch_normalization_2             │ (None, 64)               │
256 │
│ (BatchNormalization)              │                          │
│
├───────────────────────────────────┼──────────────────────────┼─────────────
───────┤
│ dropout_2 (Dropout)               │ (None, 64)               │
0 │
├───────────────────────────────────┼──────────────────────────┼─────────────
───────┤
│ dense_9 (Dense)                   │ (None, 64)               │
4,160 │
├───────────────────────────────────┼──────────────────────────┼─────────────
───────┤
│ batch_normalization_3             │ (None, 64)               │
256 │
│ (BatchNormalization)              │                          │
│
├───────────────────────────────────┼──────────────────────────┼─────────────
───────┤
│ dense_10 (Dense)                  │ (None, 1)                │
65 │
└───────────────────────────────────┴──────────────────────────┴─────────────
───────┘

 Total params: 20,865 (81.50 KB)

 Trainable params: 20,225 (79.00 KB)

 Non-trainable params: 640 (2.50 KB)
```

Then, I trained the model with EarlyStopping (patience = 10), which halts training if the validation loss doesn't improve for 10 consecutive epochs, and used 250 epochs to give the model enough time to converge while preventing overfitting.

```
early_stopping = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)

# Train the model
history = model_seq.fit(x_train_scaled, y_train,
                        validation_data=(x_val_scaled, y_val),
                        epochs=250,
                        batch_size=32,
                        verbose=1,
                        callbacks=[early_stopping])
```

```
Epoch 1/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 5s 24ms/step - loss: 5724.2617 -
mean_absolute_error: 73.7766 - mean_squared_error: 5724.0103 -
val_loss: 5443.7822 - val_mean_absolute_error: 72.0558 -
val_mean_squared_error: 5443.5308
Epoch 2/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 5467.1777 -
mean_absolute_error: 72.0414 - mean_squared_error: 5466.9258 -
val_loss: 5380.5049 - val_mean_absolute_error: 71.7140 -
val_mean_squared_error: 5380.2534
Epoch 3/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 5511.7173 -
mean_absolute_error: 72.4151 - mean_squared_error: 5511.4653 -
val_loss: 5300.6914 - val_mean_absolute_error: 71.2235 -
val_mean_squared_error: 5300.4380
Epoch 4/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - loss: 5407.8628 -
mean_absolute_error: 71.7485 - mean_squared_error: 5407.6099 -
val_loss: 5169.8657 - val_mean_absolute_error: 70.3036 -
val_mean_squared_error: 5169.6118
Epoch 5/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - loss: 5281.1406 -
mean_absolute_error: 70.6701 - mean_squared_error: 5280.8867 -
val_loss: 5048.1245 - val_mean_absolute_error: 69.4427 -
val_mean_squared_error: 5047.8701
Epoch 6/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - loss: 5182.4531 -
mean_absolute_error: 70.0996 - mean_squared_error: 5182.1987 -
val_loss: 4861.6812 - val_mean_absolute_error: 68.0867 -
val_mean_squared_error: 4861.4258
Epoch 7/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 4943.2207 -
mean_absolute_error: 68.4068 - mean_squared_error: 4942.9644 -
val_loss: 4741.4385 - val_mean_absolute_error: 67.2018 -
val_mean_squared_error: 4741.1826
```

```
Epoch 8/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 4779.3091 -
mean_absolute_error: 67.3543 - mean_squared_error: 4779.0532 -
val_loss: 4507.3745 - val_mean_absolute_error: 65.4535 -
val_mean_squared_error: 4507.1172
Epoch 9/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 4650.5112 -
mean_absolute_error: 66.3739 - mean_squared_error: 4650.2534 -
val_loss: 4309.6143 - val_mean_absolute_error: 63.9513 -
val_mean_squared_error: 4309.3555
Epoch 10/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 4418.7729 -
mean_absolute_error: 64.6674 - mean_squared_error: 4418.5146 -
val_loss: 4041.0376 - val_mean_absolute_error: 61.8022 -
val_mean_squared_error: 4040.7786
Epoch 11/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 4089.6418 -
mean_absolute_error: 62.0279 - mean_squared_error: 4089.3823 -
val_loss: 3767.1206 - val_mean_absolute_error: 59.5700 -
val_mean_squared_error: 3766.8606
Epoch 12/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 4016.5298 -
mean_absolute_error: 61.5758 - mean_squared_error: 4016.2695 -
val_loss: 3499.3628 - val_mean_absolute_error: 57.2375 -
val_mean_squared_error: 3499.1018
Epoch 13/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 3713.9653 -
mean_absolute_error: 58.9370 - mean_squared_error: 3713.7041 -
val_loss: 3276.9456 - val_mean_absolute_error: 55.3109 -
val_mean_squared_error: 3276.6833
Epoch 14/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 3277.9785 -
mean_absolute_error: 55.2334 - mean_squared_error: 3277.7166 -
val_loss: 3000.5093 - val_mean_absolute_error: 52.6941 -
val_mean_squared_error: 3000.2458
Epoch 15/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 3058.1572 -
mean_absolute_error: 53.2022 - mean_squared_error: 3057.8943 -
val_loss: 2714.4734 - val_mean_absolute_error: 49.9686 -
val_mean_squared_error: 2714.2092
Epoch 16/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 2802.2651 -
mean_absolute_error: 50.6472 - mean_squared_error: 2802.0007 -
val_loss: 2460.8271 - val_mean_absolute_error: 47.4094 -
val_mean_squared_error: 2460.5623
Epoch 17/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 2495.0017 -
mean_absolute_error: 47.6535 - mean_squared_error: 2494.7368 -
val_loss: 2157.6091 - val_mean_absolute_error: 44.1781 -
```

```
val_mean_squared_error: 2157.3438
Epoch 18/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 2209.6882 -
mean_absolute_error: 44.5319 - mean_squared_error: 2209.4224 -
val_loss: 1907.0280 - val_mean_absolute_error: 41.3312 -
val_mean_squared_error: 1906.7616
Epoch 19/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 2023.2583 -
mean_absolute_error: 42.4075 - mean_squared_error: 2022.9917 -
val_loss: 1697.6505 - val_mean_absolute_error: 38.8011 -
val_mean_squared_error: 1697.3833
Epoch 20/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 1823.8800 -
mean_absolute_error: 40.0505 - mean_squared_error: 1823.6125 -
val_loss: 1451.3661 - val_mean_absolute_error: 35.6684 -
val_mean_squared_error: 1451.0978
Epoch 21/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1524.1216 -
mean_absolute_error: 36.3759 - mean_squared_error: 1523.8533 -
val_loss: 1262.1989 - val_mean_absolute_error: 33.0981 -
val_mean_squared_error: 1261.9298
Epoch 22/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 1326.7906 -
mean_absolute_error: 33.6771 - mean_squared_error: 1326.5215 -
val_loss: 1083.7455 - val_mean_absolute_error: 30.3992 -
val_mean_squared_error: 1083.4756
Epoch 23/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 1130.7609 -
mean_absolute_error: 31.0235 - mean_squared_error: 1130.4908 -
val_loss: 951.7398 - val_mean_absolute_error: 28.4463 -
val_mean_squared_error: 951.4692
Epoch 24/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 985.9446 -
mean_absolute_error: 28.7602 - mean_squared_error: 985.6738 -
val_loss: 802.2562 - val_mean_absolute_error: 26.0437 -
val_mean_squared_error: 801.9848
Epoch 25/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 871.2469 -
mean_absolute_error: 26.7054 - mean_squared_error: 870.9752 -
val_loss: 684.6730 - val_mean_absolute_error: 23.9663 -
val_mean_squared_error: 684.4006
Epoch 26/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 720.0665 -
mean_absolute_error: 24.0708 - mean_squared_error: 719.7939 -
val_loss: 585.7772 - val_mean_absolute_error: 21.8152 -
val_mean_squared_error: 585.5040
Epoch 27/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 574.5822 -
mean_absolute_error: 21.2405 - mean_squared_error: 574.3087 -
```

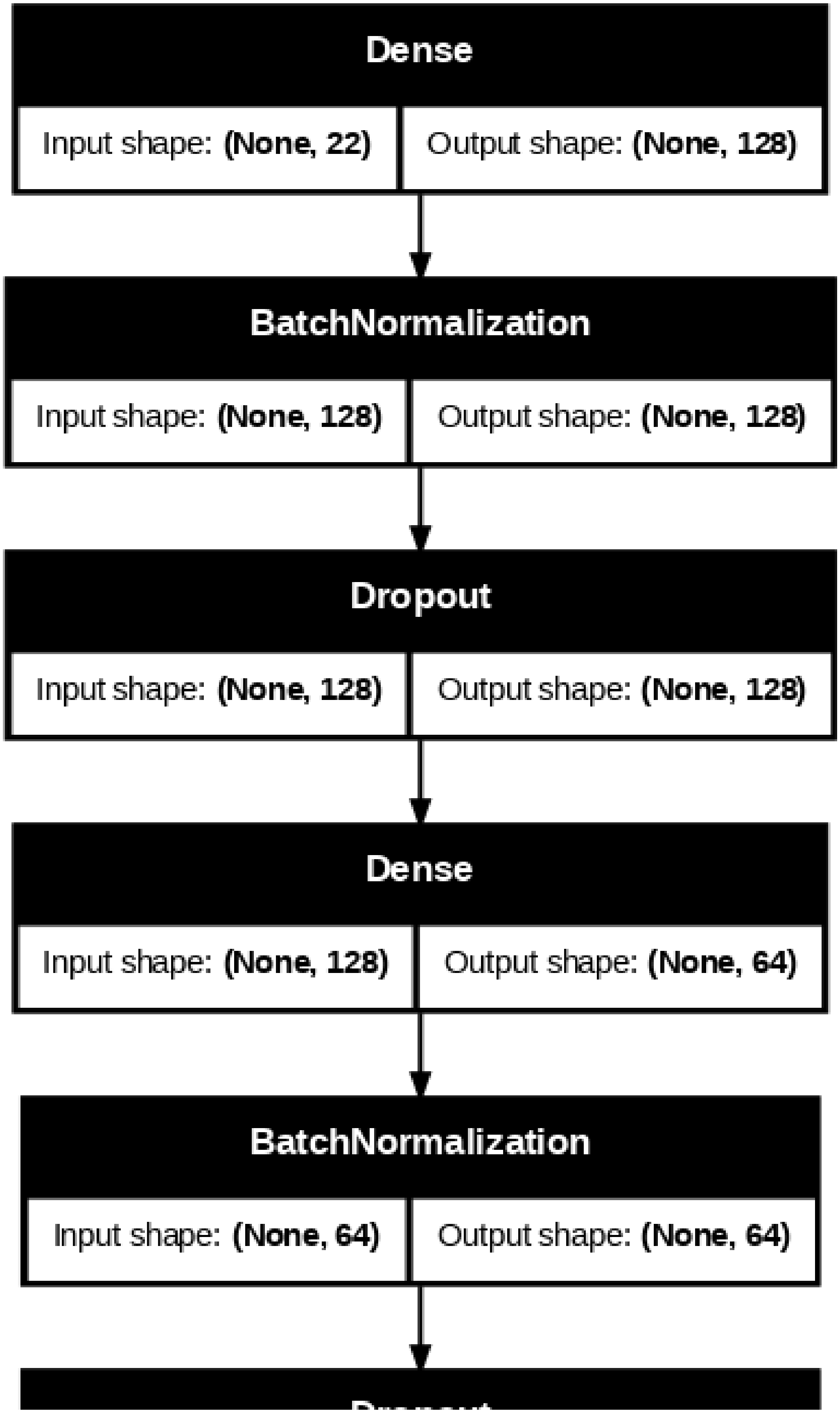
```
val_loss: 500.5959 - val_mean_absolute_error: 19.9472 -
val_mean_squared_error: 500.3216
Epoch 28/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 534.4766 -
mean_absolute_error: 20.4113 - mean_squared_error: 534.2021 -
val_loss: 428.6069 - val_mean_absolute_error: 18.3649 -
val_mean_squared_error: 428.3318
Epoch 29/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 459.3269 -
mean_absolute_error: 18.6241 - mean_squared_error: 459.0515 -
val_loss: 384.7834 - val_mean_absolute_error: 17.1221 -
val_mean_squared_error: 384.5074
Epoch 30/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 407.4894 -
mean_absolute_error: 17.4938 - mean_squared_error: 407.2133 -
val_loss: 350.3765 - val_mean_absolute_error: 16.1492 -
val_mean_squared_error: 350.0997
Epoch 31/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 335.8443 -
mean_absolute_error: 15.5015 - mean_squared_error: 335.5672 -
val_loss: 315.8338 - val_mean_absolute_error: 15.0055 -
val_mean_squared_error: 315.5560
Epoch 32/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 334.3896 -
mean_absolute_error: 15.5895 - mean_squared_error: 334.1115 -
val_loss: 280.6640 - val_mean_absolute_error: 13.7522 -
val_mean_squared_error: 280.3853
Epoch 33/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 289.1167 -
mean_absolute_error: 14.2967 - mean_squared_error: 288.8378 -
val_loss: 258.1236 - val_mean_absolute_error: 12.9254 -
val_mean_squared_error: 257.8441
Epoch 34/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 280.9151 -
mean_absolute_error: 14.0128 - mean_squared_error: 280.6354 -
val_loss: 243.9005 - val_mean_absolute_error: 12.5152 -
val_mean_squared_error: 243.6201
Epoch 35/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 236.2144 -
mean_absolute_error: 12.4882 - mean_squared_error: 235.9337 -
val_loss: 225.5256 - val_mean_absolute_error: 11.5096 -
val_mean_squared_error: 225.2443
Epoch 36/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 236.3746 -
mean_absolute_error: 12.4649 - mean_squared_error: 236.0932 -
val_loss: 219.6480 - val_mean_absolute_error: 11.3971 -
val_mean_squared_error: 219.3660
Epoch 37/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 238.5324 -
```

```
mean_absolute_error: 12.3212 - mean_squared_error: 238.2501 -
val_loss: 215.4549 - val_mean_absolute_error: 11.1420 -
val_mean_squared_error: 215.1719
Epoch 38/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 227.9316 -
mean_absolute_error: 11.9792 - mean_squared_error: 227.6484 -
val_loss: 211.9232 - val_mean_absolute_error: 10.9186 -
val_mean_squared_error: 211.6394
Epoch 39/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 236.2992 -
mean_absolute_error: 11.9718 - mean_squared_error: 236.0152 -
val_loss: 208.4695 - val_mean_absolute_error: 10.6013 -
val_mean_squared_error: 208.1848
Epoch 40/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 211.3193 -
mean_absolute_error: 11.2907 - mean_squared_error: 211.0345 -
val_loss: 205.8438 - val_mean_absolute_error: 10.4031 -
val_mean_squared_error: 205.5585
Epoch 41/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 206.9157 -
mean_absolute_error: 10.8756 - mean_squared_error: 206.6301 -
val_loss: 202.3220 - val_mean_absolute_error: 10.1489 -
val_mean_squared_error: 202.0357
Epoch 42/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 188.4791 -
mean_absolute_error: 10.8830 - mean_squared_error: 188.1927 -
val_loss: 209.5815 - val_mean_absolute_error: 10.5258 -
val_mean_squared_error: 209.2945
Epoch 43/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 205.8478 -
mean_absolute_error: 10.7959 - mean_squared_error: 205.5607 -
val_loss: 201.3459 - val_mean_absolute_error: 10.0688 -
val_mean_squared_error: 201.0582
Epoch 44/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 213.5929 -
mean_absolute_error: 11.1385 - mean_squared_error: 213.3049 -
val_loss: 203.2230 - val_mean_absolute_error: 10.1354 -
val_mean_squared_error: 202.9344
Epoch 45/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 210.2280 -
mean_absolute_error: 11.0206 - mean_squared_error: 209.9391 -
val_loss: 205.6993 - val_mean_absolute_error: 10.1175 -
val_mean_squared_error: 205.4099
Epoch 46/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 198.6393 -
mean_absolute_error: 10.5259 - mean_squared_error: 198.3496 -
val_loss: 204.2334 - val_mean_absolute_error: 10.1237 -
val_mean_squared_error: 203.9433
Epoch 47/250
```

```
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 186.1339 -
mean_absolute_error: 10.3697 - mean_squared_error: 185.8436 -
val_loss: 200.2683 - val_mean_absolute_error: 9.9804 -
val_mean_squared_error: 199.9776
Epoch 48/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 216.7175 -
mean_absolute_error: 11.0761 - mean_squared_error: 216.4266 -
val_loss: 199.4671 - val_mean_absolute_error: 10.0266 -
val_mean_squared_error: 199.1758
Epoch 49/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 187.4076 -
mean_absolute_error: 10.3623 - mean_squared_error: 187.1161 -
val_loss: 202.3458 - val_mean_absolute_error: 9.8667 -
val_mean_squared_error: 202.0537
Epoch 50/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 195.7316 -
mean_absolute_error: 10.4226 - mean_squared_error: 195.4393 -
val_loss: 204.8990 - val_mean_absolute_error: 9.8820 -
val_mean_squared_error: 204.6062
Epoch 51/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 182.9145 -
mean_absolute_error: 10.3950 - mean_squared_error: 182.6216 -
val_loss: 207.4431 - val_mean_absolute_error: 10.0829 -
val_mean_squared_error: 207.1497
Epoch 52/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 179.6785 -
mean_absolute_error: 10.2292 - mean_squared_error: 179.3851 -
val_loss: 208.4056 - val_mean_absolute_error: 10.0830 -
val_mean_squared_error: 208.1118
Epoch 53/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 199.2600 -
mean_absolute_error: 10.5399 - mean_squared_error: 198.9659 -
val_loss: 204.7027 - val_mean_absolute_error: 9.9881 -
val_mean_squared_error: 204.4080
Epoch 54/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 185.1533 -
mean_absolute_error: 10.2468 - mean_squared_error: 184.8585 -
val_loss: 205.8557 - val_mean_absolute_error: 10.0166 -
val_mean_squared_error: 205.5604
Epoch 55/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 175.9093 -
mean_absolute_error: 10.0490 - mean_squared_error: 175.6138 -
val_loss: 204.2377 - val_mean_absolute_error: 9.9379 -
val_mean_squared_error: 203.9418
Epoch 56/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 188.7411 -
mean_absolute_error: 10.1233 - mean_squared_error: 188.4451 -
val_loss: 202.5629 - val_mean_absolute_error: 9.9348 -
val_mean_squared_error: 202.2664
```

```
Epoch 57/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 181.0585 -
mean_absolute_error: 10.1216 - mean_squared_error: 180.7618 -
val_loss: 203.5051 - val_mean_absolute_error: 9.9108 -
val_mean_squared_error: 203.2078
Epoch 58/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 207.5447 -
mean_absolute_error: 10.9342 - mean_squared_error: 207.2473 -
val_loss: 205.9593 - val_mean_absolute_error: 9.9551 -
val_mean_squared_error: 205.6613
```

```
plot_model(model_seq,show_shapes=True, dpi=75)
```

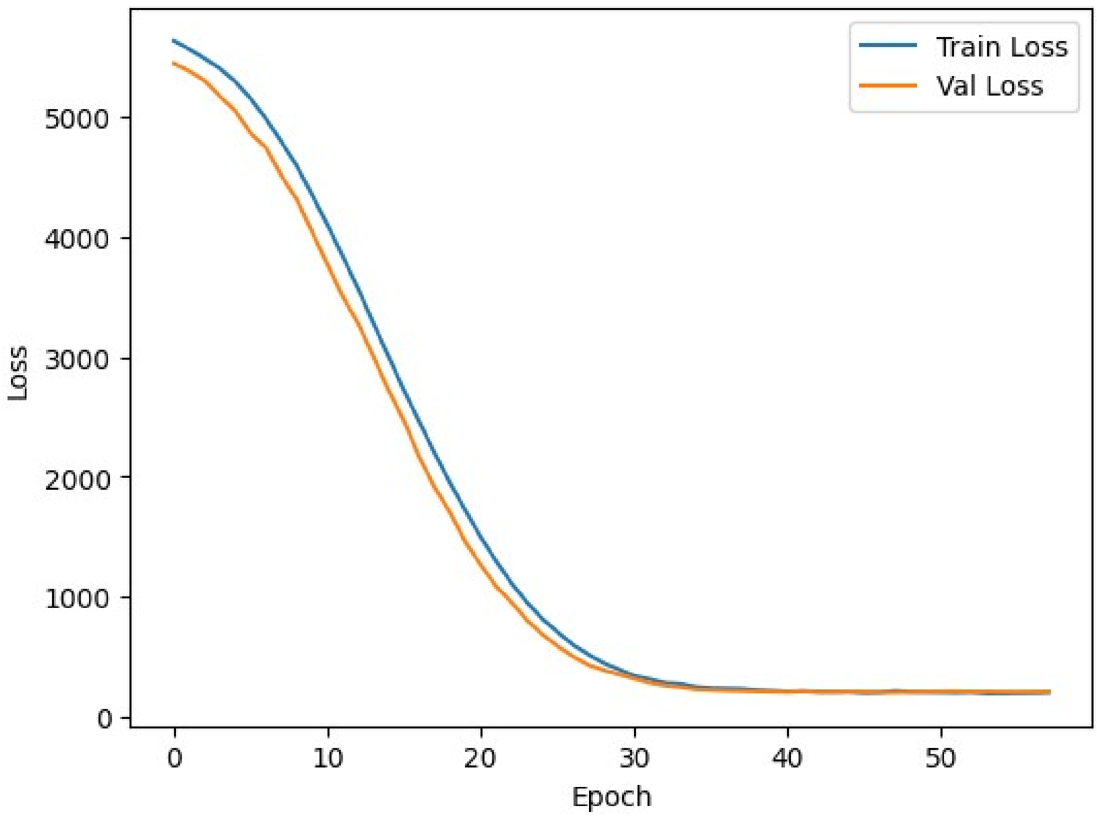

Dense
Input shape: (None, 22)
Output shape: (None, 128)
BatchNormalization
Input shape: (None, 128)
Output shape: (None, 128)
Dropout
Input shape: (None, 128)
Output shape: (None, 128)
Dense
Input shape: (None, 128)
Output shape: (None, 64)
BatchNormalization
Input shape: (None, 64)
Output shape: (None, 64)

```python
# Prediction
y_pred = model_seq.predict(x_test_scaled)
```

```
WARNING:tensorflow:5 out of the last 9 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_dist
ributed at 0x7c8d618e7880> triggered tf.function retracing. Tracing is
expensive and the excessive number of tracings could be due to (1)
creating @tf.function repeatedly in a loop, (2) passing tensors with
different shapes, (3) passing Python objects instead of tensors. For
(1), please define your @tf.function outside of the loop. For (2),
@tf.function has reduce_retracing=True option that can avoid
unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more
details.

 1/4 ━━━━━━━━━━━━━━━━━━━━ 0s 149ms/step

WARNING:tensorflow:6 out of the last 12 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_dist
ributed at 0x7c8d618e7880> triggered tf.function retracing. Tracing is
expensive and the excessive number of tracings could be due to (1)
creating @tf.function repeatedly in a loop, (2) passing tensors with
different shapes, (3) passing Python objects instead of tensors. For
(1), please define your @tf.function outside of the loop. For (2),
@tf.function has reduce_retracing=True option that can avoid
unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more
details.

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
```

Evaluate

```python
train_loss = history.history['loss']
val_loss = history.history['val_loss']
plt.plot(train_loss,label="Train Loss")
plt.plot(val_loss,label='Val Loss')
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



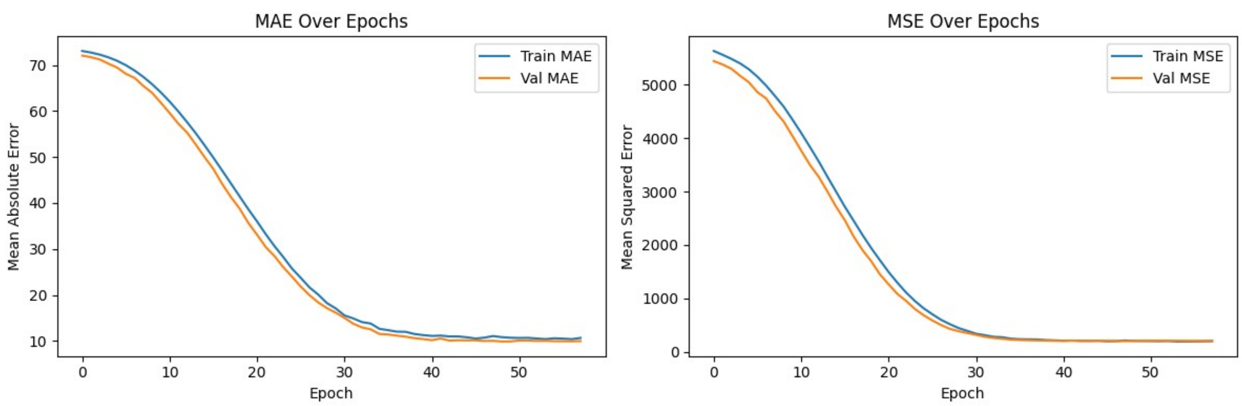
This loss curve plot is much improved, as the model trains more effectively. The loss starts to decline at around epoch 30 and stabilizes between epochs 40 and 50, indicating that the model is converging. The closeness of the training and validation loss suggests that the model is generalizing well to the validation data without significant overfitting. Overall, this behavior reflects that the adjustments made to the model such as using ELU activation and incorporating L2 regularization are having a positive impact on training dynamics and model performance.

```
R2 = r2_score(y_test, y_pred)
MAE = mean_absolute_error(y_test, y_pred)
MSE = mean_squared_error(y_test, y_pred)

print("R² Score=", R2)
print("Mean Absolute Error=", MAE)
print("Mean Squared Error=", MSE)
```

```
R² Score= 0.25563211127907237
Mean Absolute Error= 11.70159676440009
Mean Squared Error= 263.58428330350785
```

```
# Plot Mean Absolute Error (MAE)
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['mean_absolute_error'], label='Train MAE')
plt.plot(history.history['val_mean_absolute_error'], label='Val MAE')
```

```
plt.xlabel('Epoch')
plt.ylabel('Mean Absolute Error')
plt.title('MAE Over Epochs')
plt.legend()

# Plot Mean Squared Error (MSE)
plt.subplot(1, 2, 2)
plt.plot(history.history['mean_squared_error'], label='Train MSE')
plt.plot(history.history['val_mean_squared_error'], label='Val MSE')
plt.xlabel('Epoch')
plt.ylabel('Mean Squared Error')
plt.title('MSE Over Epochs')
plt.legend()

plt.tight_layout()
plt.show()
```



The modified Sequential model shows a relatively low R^2 score of 0.25, meaning it only explains 25% of the variance in the target variable. Additionally, the high MAE (11.7) and MSE (263.58) indicate that the model's predictions are not very close to the actual values. A low R^2 combined with high error values suggests that the model struggles to capture the underlying patterns in the data and may not be complex enough or is missing important features needed for accurate prediction.

Functional API

```
from tensorflow.keras.layers import Input

inputs = Input(shape=(x_train_scaled.shape[1],))

# Input layer
inputs = Input(shape=(22,))

# Hidden layers
x = Dense(128, activation='elu', kernel_regularizer=l2(0.001))(inputs)
x = BatchNormalization()(x)
x = Dropout(0.3)(x)
```

```
x = Dense(64, activation='elu', kernel_regularizer=l2(0.001))(x)
x = BatchNormalization()(x)
x = Dropout(0.3)(x)

x = Dense(64, activation='elu', kernel_regularizer=l2(0.001))(x)
x = BatchNormalization()(x)
x = Dropout(0.3)(x)

x = Dense(64, activation='elu', kernel_regularizer=l2(0.001))(x)
x = BatchNormalization()(x)

# Output layer
outputs = Dense(1, activation='linear')(x)

# Create model
model_func = Model(inputs=inputs, outputs=outputs)
```

This uses a Functional API model with a series of Dense layers and ReLU activation functions in the hidden layers to learn non-linear patterns. The model starts with an input layer that accepts 25 features. Each hidden layer uses BatchNormalization to stabilize training and Dropout (0.3 rate) to prevent overfitting. The architecture has four hidden layers with 128, 64, 64, and 64 units, respectively, and concludes with a linear activation function in the output layer, making it suitable for regression tasks.

```
model_func.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
    loss='mean_squared_error',
    metrics=['mean_absolute_error', 'mean_squared_error']
)
```

Additionally, the model is compiled with an Adam optimizer using a learning rate of 0.001, which helps to control the step size during training, ensuring efficient convergence.

```
# Show model summary
model_func.summary()

Model: "functional_3"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━
━━━━━┓
┃ Layer (type)                     ┃ Output Shape            ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━
━━━━━┩
│ input_layer_4 (InputLayer)       │ (None, 22)              │
0 │
├──────────────────────────────────┼─────────────────────────┼──────────
─────┤
```

```
│ dense_11 (Dense)                 │ (None, 128)               │
2,944 │
├──────────────────────────────────┼───────────────────────────┼──────────
───────┤
│ batch_normalization_4            │ (None, 128)               │
512 │
│ (BatchNormalization)             │                           │
│
├──────────────────────────────────┼───────────────────────────┼──────────
───────┤
│ dropout_3 (Dropout)              │ (None, 128)               │
0 │
├──────────────────────────────────┼───────────────────────────┼──────────
───────┤
│ dense_12 (Dense)                 │ (None, 64)                │
8,256 │
├──────────────────────────────────┼───────────────────────────┼──────────
───────┤
│ batch_normalization_5            │ (None, 64)                │
256 │
│ (BatchNormalization)             │                           │
│
├──────────────────────────────────┼───────────────────────────┼──────────
───────┤
│ dropout_4 (Dropout)              │ (None, 64)                │
0 │
├──────────────────────────────────┼───────────────────────────┼──────────
───────┤
│ dense_13 (Dense)                 │ (None, 64)                │
4,160 │
├──────────────────────────────────┼───────────────────────────┼──────────
───────┤
│ batch_normalization_6            │ (None, 64)                │
256 │
│ (BatchNormalization)             │                           │
│
├──────────────────────────────────┼───────────────────────────┼──────────
───────┤
│ dropout_5 (Dropout)              │ (None, 64)                │
0 │
├──────────────────────────────────┼───────────────────────────┼──────────
───────┤
│ dense_14 (Dense)                 │ (None, 64)                │
4,160 │
├──────────────────────────────────┼───────────────────────────┼──────────
───────┤
│ batch_normalization_7            │ (None, 64)                │
256 │
│ (BatchNormalization)             │                           │
```

```
│
├──────────────────────────────────┼─────────────────────────┼──────────
───────┤
│ dense_15 (Dense)                 │ (None, 1)               │
65 │
└──────────────────────────────────┴─────────────────────────┴──────────
───────┘

 Total params: 20,865 (81.50 KB)

 Trainable params: 20,225 (79.00 KB)

 Non-trainable params: 640 (2.50 KB)
```

```python
early_stopping = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)

# Train the model
history = model_func.fit(x_train_scaled, y_train,
                    validation_data=(x_val_scaled, y_val),
                    epochs=250,
                    batch_size=32,
                    verbose=1,
                    callbacks=[early_stopping])
```

```
Epoch 1/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 6s 34ms/step - loss: 5665.9287 -
mean_absolute_error: 73.1542 - mean_squared_error: 5665.6777 -
val_loss: 5429.6685 - val_mean_absolute_error: 71.9804 -
val_mean_squared_error: 5429.4170
Epoch 2/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - loss: 5512.6455 -
mean_absolute_error: 72.4187 - mean_squared_error: 5512.3931 -
val_loss: 5365.8384 - val_mean_absolute_error: 71.6263 -
val_mean_squared_error: 5365.5859
Epoch 3/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 5658.9316 -
mean_absolute_error: 73.4071 - mean_squared_error: 5658.6797 -
val_loss: 5251.0713 - val_mean_absolute_error: 70.8818 -
val_mean_squared_error: 5250.8184
Epoch 4/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 5446.0249 -
mean_absolute_error: 71.9690 - mean_squared_error: 5445.7710 -
val_loss: 5132.5806 - val_mean_absolute_error: 70.0485 -
val_mean_squared_error: 5132.3271
Epoch 5/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 5371.6499 -
mean_absolute_error: 71.5135 - mean_squared_error: 5371.3965 -
val_loss: 5047.9810 - val_mean_absolute_error: 69.4352 -
val_mean_squared_error: 5047.7261
```

```
Epoch 6/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 5356.7476 -
mean_absolute_error: 71.4196 - mean_squared_error: 5356.4927 -
val_loss: 4864.0693 - val_mean_absolute_error: 68.1199 -
val_mean_squared_error: 4863.8145
Epoch 7/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 5109.2930 -
mean_absolute_error: 69.8043 - mean_squared_error: 5109.0376 -
val_loss: 4621.4082 - val_mean_absolute_error: 66.3593 -
val_mean_squared_error: 4621.1519
Epoch 8/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 4918.2568 -
mean_absolute_error: 68.3500 - mean_squared_error: 4917.9995 -
val_loss: 4402.0698 - val_mean_absolute_error: 64.6534 -
val_mean_squared_error: 4401.8120
Epoch 9/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 4685.5166 -
mean_absolute_error: 66.6817 - mean_squared_error: 4685.2588 -
val_loss: 4210.3975 - val_mean_absolute_error: 63.1441 -
val_mean_squared_error: 4210.1382
Epoch 10/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 4490.5869 -
mean_absolute_error: 65.3843 - mean_squared_error: 4490.3281 -
val_loss: 3941.9592 - val_mean_absolute_error: 60.9800 -
val_mean_squared_error: 3941.6997
Epoch 11/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 4170.7676 -
mean_absolute_error: 62.6504 - mean_squared_error: 4170.5078 -
val_loss: 3734.9829 - val_mean_absolute_error: 59.2835 -
val_mean_squared_error: 3734.7219
Epoch 12/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 4074.5413 -
mean_absolute_error: 61.9526 - mean_squared_error: 4074.2805 -
val_loss: 3481.3250 - val_mean_absolute_error: 57.0951 -
val_mean_squared_error: 3481.0630
Epoch 13/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 3563.0049 -
mean_absolute_error: 57.6526 - mean_squared_error: 3562.7432 -
val_loss: 3215.4346 - val_mean_absolute_error: 54.7024 -
val_mean_squared_error: 3215.1714
Epoch 14/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 3412.1855 -
mean_absolute_error: 56.4578 - mean_squared_error: 3411.9224 -
val_loss: 2999.7119 - val_mean_absolute_error: 52.7570 -
val_mean_squared_error: 2999.4480
Epoch 15/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 3081.7068 -
mean_absolute_error: 53.4666 - mean_squared_error: 3081.4429 -
val_loss: 2697.0515 - val_mean_absolute_error: 49.8268 -
```

```
val_mean_squared_error: 2696.7869
Epoch 16/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 2879.0903 -
mean_absolute_error: 51.4180 - mean_squared_error: 2878.8254 -
val_loss: 2438.2827 - val_mean_absolute_error: 47.2296 -
val_mean_squared_error: 2438.0168
Epoch 17/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 2582.2778 -
mean_absolute_error: 48.4954 - mean_squared_error: 2582.0120 -
val_loss: 2157.3513 - val_mean_absolute_error: 44.1326 -
val_mean_squared_error: 2157.0845
Epoch 18/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 2327.5178 -
mean_absolute_error: 45.5622 - mean_squared_error: 2327.2512 -
val_loss: 1932.2089 - val_mean_absolute_error: 41.6023 -
val_mean_squared_error: 1931.9413
Epoch 19/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 1977.8993 -
mean_absolute_error: 41.7408 - mean_squared_error: 1977.6316 -
val_loss: 1670.6161 - val_mean_absolute_error: 38.4532 -
val_mean_squared_error: 1670.3477
Epoch 20/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 1816.7772 -
mean_absolute_error: 40.1720 - mean_squared_error: 1816.5084 -
val_loss: 1444.9323 - val_mean_absolute_error: 35.5606 -
val_mean_squared_error: 1444.6626
Epoch 21/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 1548.9656 -
mean_absolute_error: 36.4852 - mean_squared_error: 1548.6959 -
val_loss: 1273.0443 - val_mean_absolute_error: 33.3103 -
val_mean_squared_error: 1272.7738
Epoch 22/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 1349.4875 -
mean_absolute_error: 33.7933 - mean_squared_error: 1349.2168 -
val_loss: 1106.5234 - val_mean_absolute_error: 30.8554 -
val_mean_squared_error: 1106.2518
Epoch 23/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1212.7385 -
mean_absolute_error: 32.0659 - mean_squared_error: 1212.4667 -
val_loss: 944.6947 - val_mean_absolute_error: 28.3365 -
val_mean_squared_error: 944.4224
Epoch 24/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1006.6848 -
mean_absolute_error: 29.0044 - mean_squared_error: 1006.4122 -
val_loss: 824.4277 - val_mean_absolute_error: 26.4052 -
val_mean_squared_error: 824.1545
Epoch 25/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 877.6826 -
mean_absolute_error: 26.9926 - mean_squared_error: 877.4092 -
```

```
val_loss: 692.9370 - val_mean_absolute_error: 23.9764 -
val_mean_squared_error: 692.6630
Epoch 26/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 737.8717 -
mean_absolute_error: 24.3291 - mean_squared_error: 737.5974 -
val_loss: 600.7045 - val_mean_absolute_error: 22.2789 -
val_mean_squared_error: 600.4296
Epoch 27/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 613.8931 -
mean_absolute_error: 21.9988 - mean_squared_error: 613.6178 -
val_loss: 508.0497 - val_mean_absolute_error: 20.1819 -
val_mean_squared_error: 507.7736
Epoch 28/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 555.0670 -
mean_absolute_error: 20.8545 - mean_squared_error: 554.7906 -
val_loss: 433.8089 - val_mean_absolute_error: 18.4337 -
val_mean_squared_error: 433.5317
Epoch 29/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 476.7657 -
mean_absolute_error: 19.4838 - mean_squared_error: 476.4883 -
val_loss: 379.9834 - val_mean_absolute_error: 16.9539 -
val_mean_squared_error: 379.7053
Epoch 30/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 394.8156 -
mean_absolute_error: 17.1929 - mean_squared_error: 394.5372 -
val_loss: 329.7207 - val_mean_absolute_error: 15.6565 -
val_mean_squared_error: 329.4415
Epoch 31/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 362.0969 -
mean_absolute_error: 16.3908 - mean_squared_error: 361.8175 -
val_loss: 291.2494 - val_mean_absolute_error: 14.5457 -
val_mean_squared_error: 290.9693
Epoch 32/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 322.6794 -
mean_absolute_error: 15.2596 - mean_squared_error: 322.3991 -
val_loss: 258.1195 - val_mean_absolute_error: 13.1898 -
val_mean_squared_error: 257.8384
Epoch 33/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 284.5695 -
mean_absolute_error: 14.1321 - mean_squared_error: 284.2882 -
val_loss: 242.6507 - val_mean_absolute_error: 12.5396 -
val_mean_squared_error: 242.3686
Epoch 34/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 261.2295 -
mean_absolute_error: 13.3538 - mean_squared_error: 260.9471 -
val_loss: 230.9603 - val_mean_absolute_error: 11.8417 -
val_mean_squared_error: 230.6773
Epoch 35/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 245.7377 -
```

```
mean_absolute_error: 12.8542 - mean_squared_error: 245.4545 -
val_loss: 223.9962 - val_mean_absolute_error: 11.4067 -
val_mean_squared_error: 223.7124
Epoch 36/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 260.6072 -
mean_absolute_error: 13.1069 - mean_squared_error: 260.3231 -
val_loss: 221.0378 - val_mean_absolute_error: 11.2504 -
val_mean_squared_error: 220.7527
Epoch 37/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 236.9454 -
mean_absolute_error: 12.6195 - mean_squared_error: 236.6602 -
val_loss: 215.0209 - val_mean_absolute_error: 10.7067 -
val_mean_squared_error: 214.7351
Epoch 38/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 214.9520 -
mean_absolute_error: 11.4784 - mean_squared_error: 214.6659 -
val_loss: 211.4196 - val_mean_absolute_error: 10.6173 -
val_mean_squared_error: 211.1328
Epoch 39/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 222.0407 -
mean_absolute_error: 11.7771 - mean_squared_error: 221.7537 -
val_loss: 206.2873 - val_mean_absolute_error: 10.3380 -
val_mean_squared_error: 205.9998
Epoch 40/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 225.8286 -
mean_absolute_error: 11.6632 - mean_squared_error: 225.5409 -
val_loss: 205.2498 - val_mean_absolute_error: 10.1172 -
val_mean_squared_error: 204.9616
Epoch 41/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 212.1503 -
mean_absolute_error: 11.4280 - mean_squared_error: 211.8618 -
val_loss: 207.1917 - val_mean_absolute_error: 10.1474 -
val_mean_squared_error: 206.9026
Epoch 42/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 197.7590 -
mean_absolute_error: 10.5632 - mean_squared_error: 197.4697 -
val_loss: 208.1175 - val_mean_absolute_error: 10.1674 -
val_mean_squared_error: 207.8277
Epoch 43/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - loss: 202.4885 -
mean_absolute_error: 10.9343 - mean_squared_error: 202.1985 -
val_loss: 208.3851 - val_mean_absolute_error: 10.1663 -
val_mean_squared_error: 208.0944
Epoch 44/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 221.7048 -
mean_absolute_error: 11.4107 - mean_squared_error: 221.4139 -
val_loss: 205.4678 - val_mean_absolute_error: 9.9670 -
val_mean_squared_error: 205.1765
Epoch 45/250
```

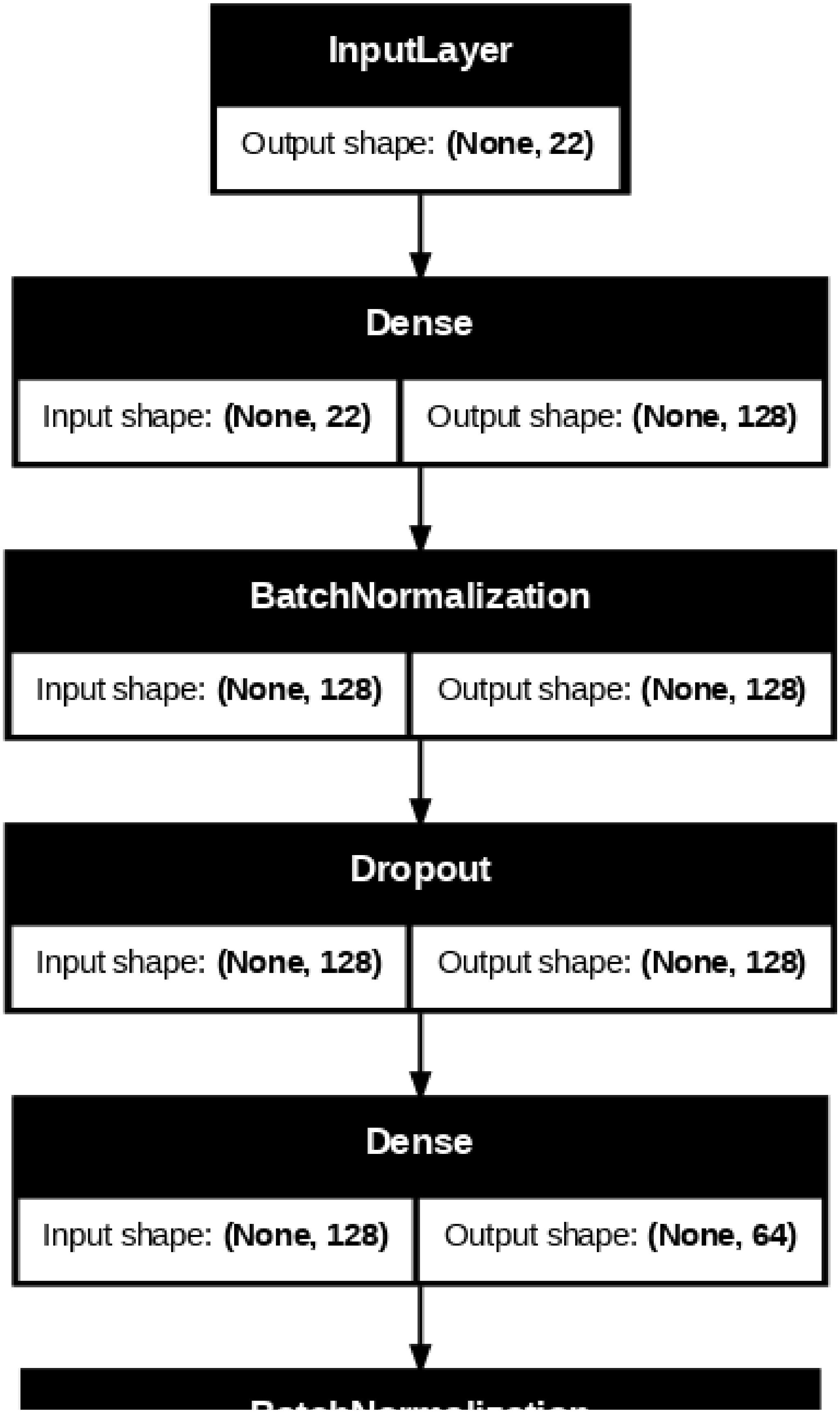
```
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - loss: 207.1124 -
mean_absolute_error: 11.1623 - mean_squared_error: 206.8208 -
val_loss: 201.2608 - val_mean_absolute_error: 9.9199 -
val_mean_squared_error: 200.9688
Epoch 46/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 193.7361 -
mean_absolute_error: 10.4989 - mean_squared_error: 193.4438 -
val_loss: 204.3364 - val_mean_absolute_error: 9.9699 -
val_mean_squared_error: 204.0435
Epoch 47/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 201.8724 -
mean_absolute_error: 10.7469 - mean_squared_error: 201.5794 -
val_loss: 201.5531 - val_mean_absolute_error: 9.7041 -
val_mean_squared_error: 201.2596
Epoch 48/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 187.1291 -
mean_absolute_error: 10.3637 - mean_squared_error: 186.8354 -
val_loss: 202.3413 - val_mean_absolute_error: 9.6968 -
val_mean_squared_error: 202.0470
Epoch 49/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 214.0778 -
mean_absolute_error: 10.8448 - mean_squared_error: 213.7832 -
val_loss: 203.1346 - val_mean_absolute_error: 9.7665 -
val_mean_squared_error: 202.8394
Epoch 50/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 198.3446 -
mean_absolute_error: 10.6867 - mean_squared_error: 198.0492 -
val_loss: 202.8846 - val_mean_absolute_error: 9.8146 -
val_mean_squared_error: 202.5884
Epoch 51/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 188.2829 -
mean_absolute_error: 10.2407 - mean_squared_error: 187.9866 -
val_loss: 202.8741 - val_mean_absolute_error: 9.8426 -
val_mean_squared_error: 202.5772
Epoch 52/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 199.5571 -
mean_absolute_error: 10.8990 - mean_squared_error: 199.2601 -
val_loss: 205.5434 - val_mean_absolute_error: 9.9500 -
val_mean_squared_error: 205.2458
Epoch 53/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 196.6833 -
mean_absolute_error: 10.5712 - mean_squared_error: 196.3855 -
val_loss: 200.9202 - val_mean_absolute_error: 9.6747 -
val_mean_squared_error: 200.6218
Epoch 54/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - loss: 205.8728 -
mean_absolute_error: 10.7653 - mean_squared_error: 205.5741 -
val_loss: 198.8845 - val_mean_absolute_error: 9.6965 -
val_mean_squared_error: 198.5854
```

```
Epoch 55/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 209.3855 -
mean_absolute_error: 10.8626 - mean_squared_error: 209.0862 -
val_loss: 200.1332 - val_mean_absolute_error: 9.9024 -
val_mean_squared_error: 199.8335
Epoch 56/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - loss: 193.5393 -
mean_absolute_error: 10.4671 - mean_squared_error: 193.2394 -
val_loss: 198.6743 - val_mean_absolute_error: 9.6895 -
val_mean_squared_error: 198.3739
Epoch 57/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - loss: 188.1380 -
mean_absolute_error: 10.3194 - mean_squared_error: 187.8374 -
val_loss: 198.0440 - val_mean_absolute_error: 9.7075 -
val_mean_squared_error: 197.7428
Epoch 58/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 201.6210 -
mean_absolute_error: 10.9581 - mean_squared_error: 201.3196 -
val_loss: 199.4951 - val_mean_absolute_error: 9.8770 -
val_mean_squared_error: 199.1931
Epoch 59/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 201.9776 -
mean_absolute_error: 10.7121 - mean_squared_error: 201.6754 -
val_loss: 197.4078 - val_mean_absolute_error: 9.7034 -
val_mean_squared_error: 197.1051
Epoch 60/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 199.7760 -
mean_absolute_error: 10.5606 - mean_squared_error: 199.4733 -
val_loss: 196.5782 - val_mean_absolute_error: 9.7409 -
val_mean_squared_error: 196.2749
Epoch 61/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 190.6182 -
mean_absolute_error: 10.3645 - mean_squared_error: 190.3147 -
val_loss: 197.4910 - val_mean_absolute_error: 9.8230 -
val_mean_squared_error: 197.1870
Epoch 62/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 194.5708 -
mean_absolute_error: 10.4087 - mean_squared_error: 194.2666 -
val_loss: 195.8092 - val_mean_absolute_error: 9.7470 -
val_mean_squared_error: 195.5045
Epoch 63/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 186.0467 -
mean_absolute_error: 10.2564 - mean_squared_error: 185.7419 -
val_loss: 199.8962 - val_mean_absolute_error: 9.7496 -
val_mean_squared_error: 199.5908
Epoch 64/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 183.3465 -
mean_absolute_error: 10.2090 - mean_squared_error: 183.0409 -
val_loss: 199.4720 - val_mean_absolute_error: 9.7988 -
```

```
val_mean_squared_error: 199.1659
Epoch 65/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 177.3903 -
mean_absolute_error: 10.0301 - mean_squared_error: 177.0840 -
val_loss: 201.4003 - val_mean_absolute_error: 9.8751 -
val_mean_squared_error: 201.0936
Epoch 66/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 177.5214 -
mean_absolute_error: 9.8270 - mean_squared_error: 177.2145 - val_loss:
199.1071 - val_mean_absolute_error: 9.7529 - val_mean_squared_error:
198.7996
Epoch 67/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 188.4021 -
mean_absolute_error: 10.3005 - mean_squared_error: 188.0944 -
val_loss: 204.8605 - val_mean_absolute_error: 10.0210 -
val_mean_squared_error: 204.5523
Epoch 68/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 195.3024 -
mean_absolute_error: 10.6146 - mean_squared_error: 194.9942 -
val_loss: 197.2351 - val_mean_absolute_error: 9.8085 -
val_mean_squared_error: 196.9263
Epoch 69/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 181.7300 -
mean_absolute_error: 9.9151 - mean_squared_error: 181.4210 - val_loss:
195.6443 - val_mean_absolute_error: 9.7471 - val_mean_squared_error:
195.3348
Epoch 70/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 192.4814 -
mean_absolute_error: 10.5670 - mean_squared_error: 192.1719 -
val_loss: 196.0975 - val_mean_absolute_error: 9.6810 -
val_mean_squared_error: 195.7875
Epoch 71/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 207.0093 -
mean_absolute_error: 10.4519 - mean_squared_error: 206.6990 -
val_loss: 199.4959 - val_mean_absolute_error: 9.8384 -
val_mean_squared_error: 199.1850
Epoch 72/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 213.9744 -
mean_absolute_error: 10.7788 - mean_squared_error: 213.6633 -
val_loss: 192.4788 - val_mean_absolute_error: 9.6919 -
val_mean_squared_error: 192.1671
Epoch 73/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 183.1068 -
mean_absolute_error: 10.4433 - mean_squared_error: 182.7949 -
val_loss: 197.3482 - val_mean_absolute_error: 9.7927 -
val_mean_squared_error: 197.0358
Epoch 74/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 203.7340 -
mean_absolute_error: 10.4998 - mean_squared_error: 203.4215 -
```

```
val_loss: 193.2769 - val_mean_absolute_error: 9.7967 -
val_mean_squared_error: 192.9640
Epoch 75/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 173.9892 -
mean_absolute_error: 9.7234 - mean_squared_error: 173.6761 - val_loss:
192.7126 - val_mean_absolute_error: 9.6827 - val_mean_squared_error:
192.3991
Epoch 76/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 166.4318 -
mean_absolute_error: 9.6195 - mean_squared_error: 166.1181 - val_loss:
195.2146 - val_mean_absolute_error: 9.7071 - val_mean_squared_error:
194.9005
Epoch 77/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 167.6431 -
mean_absolute_error: 9.7513 - mean_squared_error: 167.3288 - val_loss:
197.1027 - val_mean_absolute_error: 9.8481 - val_mean_squared_error:
196.7880
Epoch 78/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 190.0085 -
mean_absolute_error: 10.3525 - mean_squared_error: 189.6936 -
val_loss: 193.9883 - val_mean_absolute_error: 9.6975 -
val_mean_squared_error: 193.6729
Epoch 79/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 195.1637 -
mean_absolute_error: 10.4491 - mean_squared_error: 194.8483 -
val_loss: 196.9236 - val_mean_absolute_error: 9.8235 -
val_mean_squared_error: 196.6077
Epoch 80/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 162.8532 -
mean_absolute_error: 9.3572 - mean_squared_error: 162.5371 - val_loss:
198.4034 - val_mean_absolute_error: 9.9580 - val_mean_squared_error:
198.0868
Epoch 81/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 187.5797 -
mean_absolute_error: 10.0736 - mean_squared_error: 187.2630 -
val_loss: 193.2899 - val_mean_absolute_error: 9.6366 -
val_mean_squared_error: 192.9726
Epoch 82/250
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 181.5526 -
mean_absolute_error: 10.0136 - mean_squared_error: 181.2352 -
val_loss: 193.5690 - val_mean_absolute_error: 9.7739 -
val_mean_squared_error: 193.2511
```

```python
plot_model(model_func, show_shapes=True, dpi=75)
```



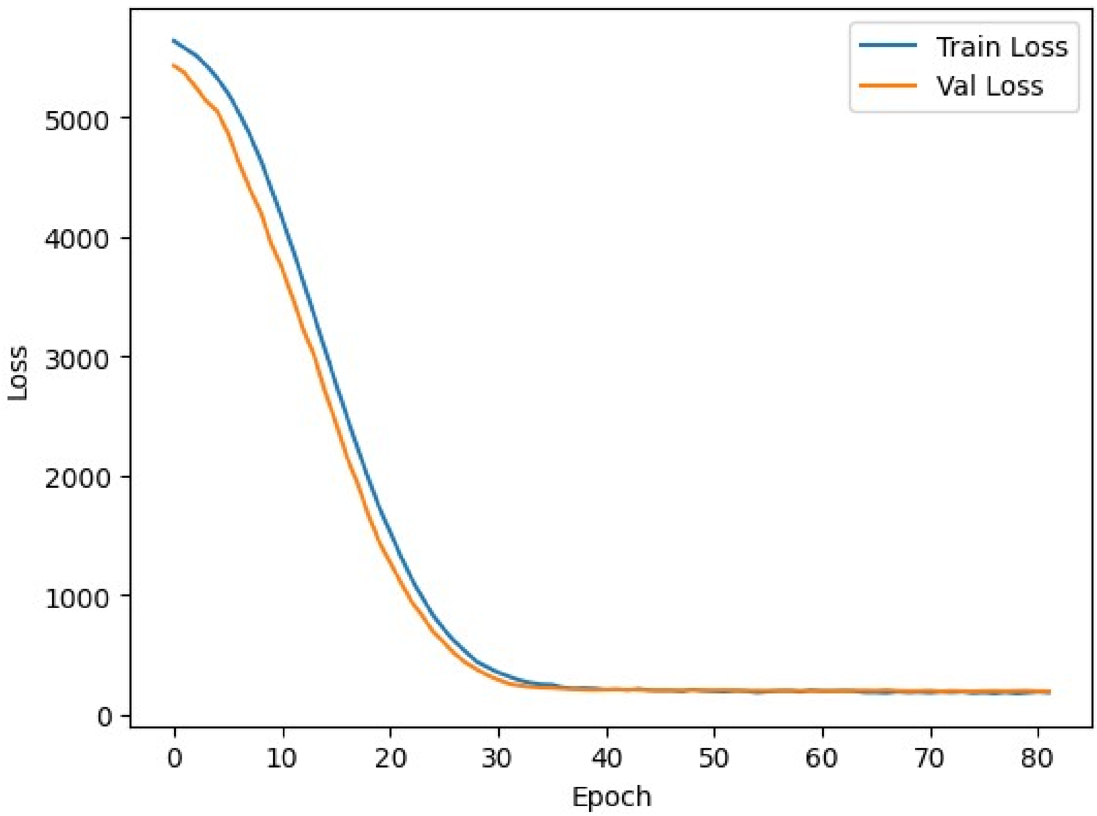
InputLayer
Output shape: (None, 22)
Dense
Input shape: (None, 22)
Output shape: (None, 128)
BatchNormalization
Input shape: (None, 128)
Output shape: (None, 128)
Dropout
Input shape: (None, 128)
Output shape: (None, 128)
Dense
Input shape: (None, 128)
Output shape: (None, 64)

```
y_pred = model_func.predict(x_test_scaled)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
```

Evaluate

```
train_loss = history.history['loss']
val_loss = history.history['val_loss']
plt.plot(train_loss, label="Train Loss")
plt.plot(val_loss, label='Val Loss')
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



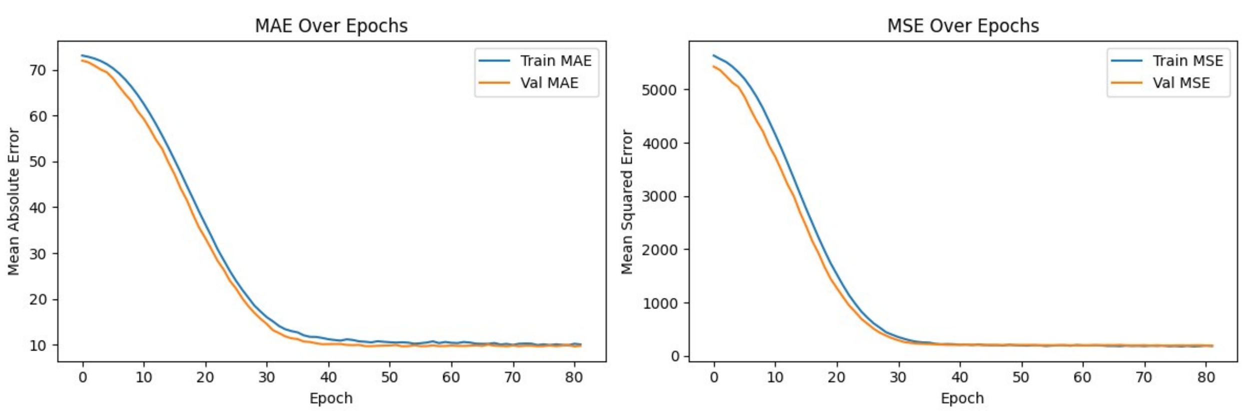
Based on the plot, the loss curve for both training and validation in modified functional API model shows a consistent decrease throughout epochs, which means the model is effectively learning from the training data while generating well with the validation dataset. The small gap between the two curves suggests that the model is not overfitting, and it can be concluded that it has converged by epoch 60.

```
R2 = r2_score(y_test, y_pred)
MAE = mean_absolute_error(y_test, y_pred)
MSE = mean_squared_error(y_test, y_pred)
```

```python
print("R² Score=", R2)
print("Mean Absolute Error=", MAE)
print("Mean Squared Error=", MSE)
```

```
R² Score= 0.30773368610369567
Mean Absolute Error= 11.077980034795301
Mean Squared Error= 245.13486270487005
```

```python
# Plot Mean Absolute Error (MAE)
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['mean_absolute_error'], label='Train MAE')
plt.plot(history.history['val_mean_absolute_error'], label='Val MAE')
plt.xlabel('Epoch')
plt.ylabel('Mean Absolute Error')
plt.title('MAE Over Epochs')
plt.legend()

# Plot Mean Squared Error (MSE)
plt.subplot(1, 2, 2)
plt.plot(history.history['mean_squared_error'], label='Train MSE')
plt.plot(history.history['val_mean_squared_error'], label='Val MSE')
plt.xlabel('Epoch')
plt.ylabel('Mean Squared Error')
plt.title('MSE Over Epochs')
plt.legend()

plt.tight_layout()
plt.show()
```



The statistical results for modified functional API model shows a low values of R^2 (0.3), and high values of MAE (11.07) and MSE (245.13). This means that the model's predictivity is limited because its only 30% of the variance in the data.

But based on the curve plots, both MSE and MAE decrease consistently throughout epochs, which means the model learn effectively without overfitting.

Comparison Both Modified Model

Both modified models shows that the loss curve decrease consistently, we can indicate that the models are learning effectively from the dataset and showing no overfitting. However, based on the statistical results such as values of R^2, MAE, and MSE, suggest that the model are not yet performing optimal predict for productivity score, as both values for R^2 is low, and high values of MAE and MSE.

Additionally, when we compare both modified model, the functional API model show better performance than the sequential model.

Conclusion

Based on the goal to estimate the productivity score for each team in a garment factory, I went through a comprehensive process involving data cleaning, visualization, preprocessing, model building, and evaluation. At first, both the baseline models (sequential and functional API) showed they learned fase\t because the loss curve were decreased in small epochs, but there is no indicate overfitting. While this was quite good, the results weren't good based on the statistical results with low R^2 and high MAE and MSE.

After modifying the models by adding batch normalization, dropout, change ReLU into ELU activation, and apply L2 regulation, I notice a better performance. The loss curve for both models are decrease consistently without any overfitting, which means the models were learning effectively. They even converged around epochs 50/60.

However, all statistical results still shows low values of R^2 and high values for both MAE and MSE, which means need to keep improving. This also might because of the dataset has its own challenges, such as quality issues or insufficient diversity in samples.

Presentation Video Link :
https://drive.google.com/file/d/1RJrgK1A4kKmcwhBMtvc1UYC9NBNQFPlV/view?usp=sharing