UTS Deep Learning Number 1

Name: Austin Kane
NIM: 270222932
Dataset: Dataset B
Video Link: Video1

1. Artificial Neural Network

Anda adalah seorang Data Scientist di perusahaan komersil. Tugas kalian adalah membuat model **Artificial Neural Network (ANN)** yang dapat menilai produktivitas setiap team dari perusahaan produsen pakaian. Download dataset yang telah disediakan dan kerjakan task berikut ini.

1. [LO2 – 5 Poin] Lakukan **Exploratory Data Analysis (EDA)** untuk memahami kondisi data. Jelaskan semua masalah yang anda temukan pada dataset anda. Lakukan pre-processing pada dataset anda sesuai dengan hasil EDA anda, termasuk memisahkan dataset anda menjadi train, val, dan test dengan proporsi 70:10:20.
2. [LO2, LO3, & LO4 – 10 Poin] Buatlah **2 baseline model** dengan jumlah layer dan neuron yang berbeda. 1 model berupa **Sequential Model** dan 1 lagi berupa **Functional Model**. Semua hidden layer wajib menggunakan activation function bernama ReLU dan memiliki jumlah neuron minimal 2 kali lipat dari dimensi input data. Kedua model harus memiliki minimal 2 hidden layer. Lakukan training pada kedua model tersebut dengan minimal 10 epoch.
3. [LO2, LO3, & LO4 – 10 Poin] Lakukan **modifikasi pada kedua model anda**. Anda dapat mengubah jumlah neuron dan layer ataupun activation function dari hidden layer. Anda juga dapat melakukan **hyperparameter fine-tuning** pada model anda. Lakukan training pada 2 modifikasi model anda.
4. [LO2, LO3, & LO4 – 10 Poin] Lakukan **evaluasi pada 4 model** yang sudah anda buat menggunakan **minimal 3 evaluation metrics**, lalu bandingkan, analisis, dan simpulkan hasilnya.
5. [LO1, LO2 – 5 Poin] Buatlah **video presentasi** yang menjelaskan langkah-langkah pengerjaan anda serta hasil analisis anda dengan durasi maksimal 15 menit.

Import Libraries

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

import keras
from keras import layers
import keras_tuner as kt
from tensorflow import keras
from tensorflow.keras import layers, Input, Model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import RobustScaler
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
```

Read the Data

In [2]:
```
data = pd.read_parquet("Data/01/dataset_1B.parquet")
data
```

Out[2]:

	date	quarter	day	Team Code	smv	wip	over_time	incentive	idle_time	idle_men
0	1/1/2015	Quarter1	Thursday	8	26.16	1108.0	7080	98	0.0	0
1	1/1/2015	Quarter1	Thursday	1	3.94	NaN	960	0	0.0	0
2	1/1/2015	Quarter1	Thursday	11	11.41	968.0	3660	50	0.0	0
3	1/1/2015	Quarter1	Thursday	12	11.41	968.0	3660	50	0.0	0
4	1/1/2015	Quarter1	Thursday	6	25.90	1170.0	1920	50	0.0	0
...	...	...	...	...	...	...	...	...	...	...
1192	2016-03-10	Quarter2	Wednesday	10	2.90	NaN	960	0	0.0	0
1193	2016-03-10	Quarter2	Wednesday	8	3.90	NaN	960	0	0.0	0
1194	2016-03-10	Quarter2	Wednesday	7	3.90	NaN	960	0	0.0	0
1195	2016-03-10	Quarter2	Wednesday	9	2.90	NaN	1800	0	0.0	0
1196	2016-03-10	Quarter2	Wednesday	6	2.90	NaN	720	0	0.0	0

1197 rows × 13 columns

Explaratory Data Analysis (EDA)

Features Description

- date: Date of the assessment
- day: 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: Standard Minute Value, a measure of the time allocated for a task.
- wip: Work In Progress, the number of products that are unfinished.
- over_time: The amount of overtime worked, measured in minutes.
- incentive: The incentive provided to the workers, measured in USD.
- idle_time: The amount of time workers were idle, measured in minutes.
- idle_men: The number of workers who were idle.
- no_of_style_change: The number of style changes that occurred.
- no_of_workers: The total number of workers.
- productivity_score: The productivity score of the team, measured as a percentage.

Check For Missing Values

In [3]:
```
count_missing = data.isna().sum()
percent_missing = data.isna().sum() * 100 / len(data)
missing_value_data = pd.DataFrame({'Count': count_missing, 'Percentage': round(percent_missin
missing_value_data.sort_values('Count', ascending=False)
```

Out[3]:

	Count	Percentage
wip	506	42.27
quarter	0	0.00
day	0	0.00
Team Code	0	0.00
date	0	0.00
smv	0	0.00
over_time	0	0.00
incentive	0	0.00
idle_time	0	0.00
idle_men	0	0.00
no_of_style_change	0	0.00
no_of_workers	0	0.00
productivity_score	0	0.00

There are lots of missing value on the wip

Check for Duplicates

In [4]:
```
print(data.duplicated().sum())
```

```
0
```

The dataset has no duplicates which is good

Check the Datatype

In [5]:
```
data.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
```

- date is supposed to be in datetime datatype.
- Team code does not indicates an order, therfore we need to transform it to categorical.
- no_of_workers cannot be a decimal, therefore we need to round it.

Check for Inconsistent Data

In [6]:
```
for col in data.columns:
    print(f"{col}: {data[col].nunique()}")
```

```
date: 118
quarter: 5
day: 6
Team Code: 12
smv: 70
wip: 548
over_time: 143
incentive: 48
idle_time: 12
idle_men: 10
no_of_style_change: 3
no_of_workers: 66
productivity_score: 803
```

In [7]:
```
for col in data.columns:
    if data[col].nunique()<200:
        print(f"{col}:\n{data[col].unique()}\n")
```

```
date:
['1/1/2015' '2016-01-01' '1/3/2015' '2016-01-03' '1/4/2015' '2016-01-04'
 '1/5/2015' '2016-01-05' '1/6/2015' '2016-01-06' '1/7/2015' '2016-01-07'
 '1/8/2015' '2016-01-08' '1/10/2015' '2016-01-10' '1/11/2015' '2016-01-11'
 '1/12/2015' '2016-01-12' '1/13/2015' '2016-01-13' '1/14/2015'
 '2016-01-14' '1/15/2015' '2016-01-15' '1/17/2015' '2016-01-17'
 '1/18/2015' '2016-01-18' '1/19/2015' '2016-01-19' '1/20/2015'
 '2016-01-20' '1/21/2015' '2016-01-21' '1/22/2015' '2016-01-22'
 '1/24/2015' '2016-01-24' '1/25/2015' '2016-01-25' '1/26/2015'
 '2016-01-26' '1/27/2015' '2016-01-27' '1/28/2015' '2016-01-28'
 '1/29/2015' '2016-01-29' '1/31/2015' '2016-01-31' '2/1/2015' '2016-02-01'
 '2/2/2015' '2016-02-02' '2/3/2015' '2016-02-03' '2/4/2015' '2016-02-04'
 '2/5/2015' '2016-02-05' '2/7/2015' '2016-02-07' '2/8/2015' '2016-02-08'
 '2/9/2015' '2016-02-09' '2/10/2015' '2016-02-10' '2/11/2015' '2016-02-11'
 '2/12/2015' '2016-02-12' '2/14/2015' '2016-02-14' '2/15/2015'
 '2016-02-15' '2/16/2015' '2016-02-16' '2/17/2015' '2016-02-17'
 '2/18/2015' '2016-02-18' '2/19/2015' '2016-02-19' '2/22/2015'
 '2016-02-22' '2/23/2015' '2016-02-23' '2/24/2015' '2016-02-24'
 '2/25/2015' '2016-02-25' '2/26/2015' '2016-02-26' '2/28/2015'
 '2016-02-28' '3/1/2015' '2016-02-29' '3/2/2015' '2016-03-01' '3/3/2015'
 '2016-03-02' '3/4/2015' '2016-03-03' '3/5/2015' '2016-03-04' '3/7/2015'
 '2016-03-06' '3/8/2015' '2016-03-07' '3/9/2015' '2016-03-08' '3/10/2015'
 '2016-03-09' '3/11/2015' '2016-03-10']

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

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

Team Code:
[ 8  1 11 12  6  7  2  3  9 10  5  4]

smv:
[26.16  3.94 11.41 25.9  28.08 19.87 19.31  2.9  23.69  4.15 11.61 45.67
 21.98 31.83 12.52 42.41 20.79 50.48  4.3  22.4  42.27 27.13 14.61 51.02
 22.52 14.89 22.94 48.68 41.19 48.84 26.87 20.4  49.1  15.26 54.56 40.99
 29.12  4.08 42.97 15.09 30.4  48.18 20.1  38.09 18.79 23.54 50.89 24.26
 20.55 30.1  25.31 10.05 18.22  5.13 29.4  30.33 19.68 21.25  4.6   3.9
 22.53 21.82 27.48 26.66 20.2  15.28 26.82 16.1  23.41 30.48]

over_time:
[ 7080   960  3660  1920  6720  6900  6000  6480  2160  7200  1440  6600
  5640  1560  6300  6540 13800  6975  7020  6780  4260  6660  4320  6960
  2400  3840  4800  4440  1800  2700 10620 10350  9900  5310 10170  4470
 10530 10440  5490  5670  9720 12600 10050 15120 14640   900 25920 10260
  2760  4710  9540  7680  3600  6420  7980  3240  8220  6930  8460  7350
  5400  1620  1980  2970  7320  5100  3390  1260  3420  8970  4950 10080
  9810  6570  5040  4380  3630  8280  6120  5580  3720  5760  7470 10500
  6360  4140  8400 12180  9000 15000 10770 12000  9360  3060  2520   720
  3780 10320   360  6840  1080  1200  4080   240  5880  6240  4200  3960
   600  2280  5940  1320  5460  2040  4020  3000  3360  5820  6060  2640
  7500  2880   120  3300     0  3480  7380  4560  7140  5160  5280   840
  5520   480  8160  5700  2820  5340  1680  7560  1700  4680  3120]

incentive:
[  98    0   50   38   45   34   44   63   56   40   60   26   75   23
   35   69   88   30   54   37   70   27   21   24   94   29   81   55
  119   90  113   46  100   53   93   49  138   33   32   62   65  960
 1080 2880 3600 1440 1200   25]

idle_time:
[  0.   90.  150.  270.  300.    2.    5.    8.    4.5   3.5   4.    6.5]

idle_men:
[ 0 10 15 45 37 30 35 20 25 40]
```

```
no_of_style_change:
[0 1 2]

no_of_workers:
[ 59.    8.   30.5  56.   57.5  55.   54.   18.   60.   12.   20.   17.
  56.5  54.5  29.5  31.5  31.   55.5  58.   10.   16.   -8.   32.   58.5
 -55.   15.    5.   57.   53.   51.5   2.    9.    7.   19.   28.   34.
  89.   14.   25.   52.    4.   21.   35.  -10.   51.   33.   11.   33.5
 -33.   22.   26.   27.  -57.   59.5  50.   44.   49.   47.   48.   42.
  24.   45.   46.   39.   38.    6. ]
```

- There are inconsistent formating on the date, with "YYYY-MM-DD" and "MM/DD/YYY"
- There are no quarter 5

Minor Data Cleaning

Converting Team Code to Category

In [8]:
```
data['Team Code'] = data['Team Code'].astype('category')
```

Converting Date to Datetime

In [9]:
```
def parse_mixed_dates(date_str):
    for format in ("%Y-%m-%d", "%m/%d/%Y"):
        try:
            return pd.to_datetime(date_str, format=format)
        except (ValueError, TypeError):
            continue
    return pd.NaT

data['date'] = data['date'].apply(parse_mixed_dates)
```

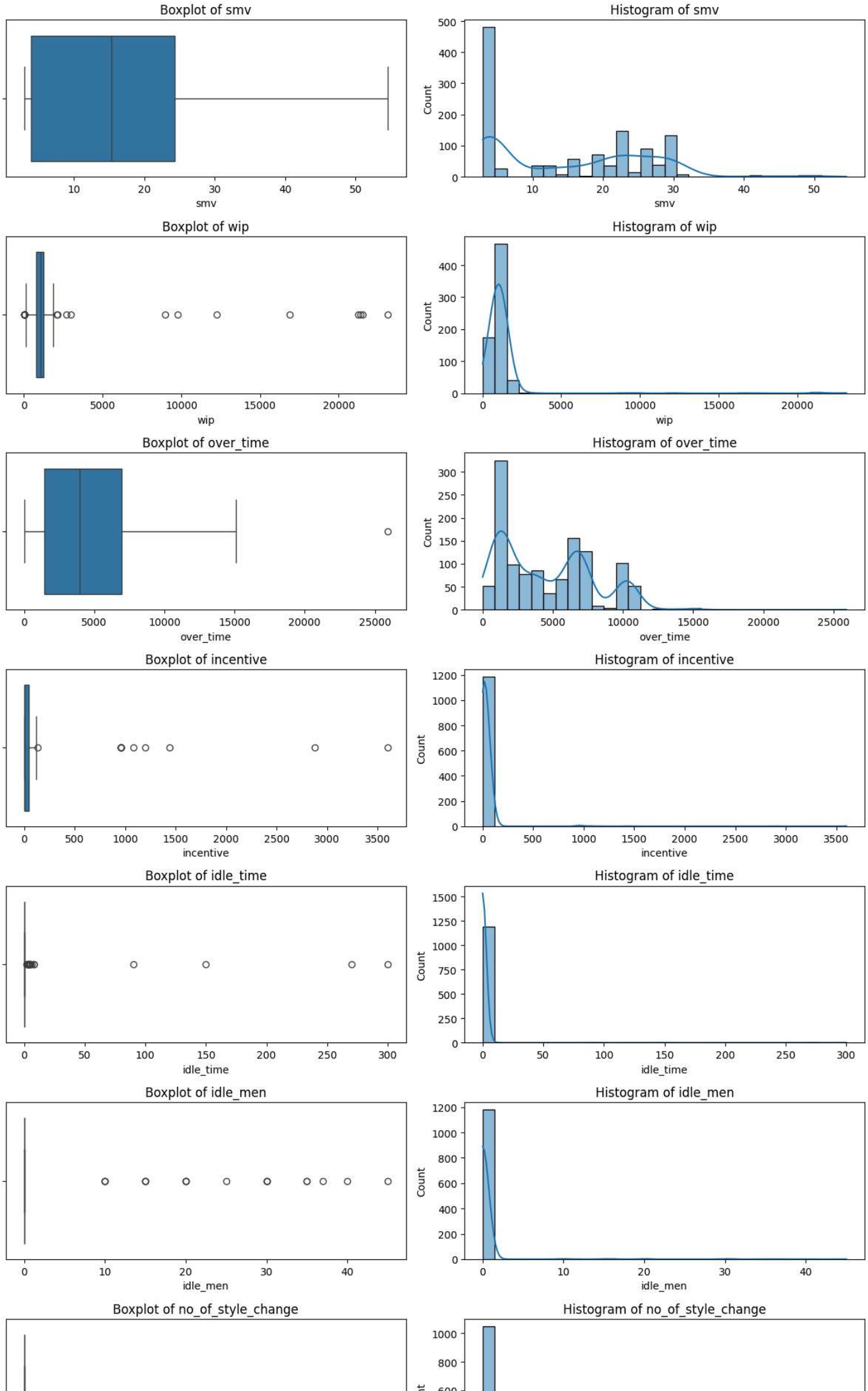
Data Visualization

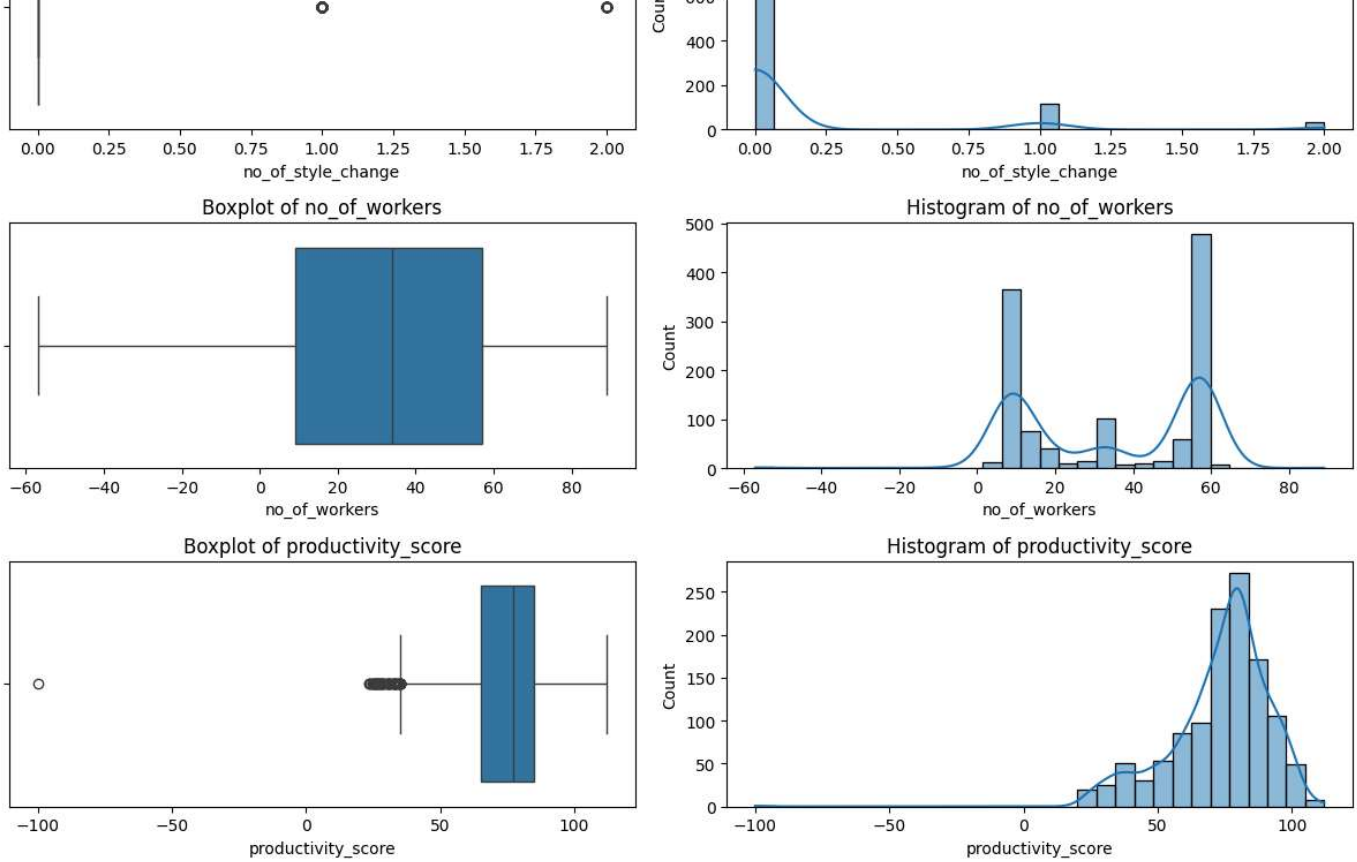
Plot for Numerical Data

In [10]:
```
def num_plot(data):
    num_cols = data.select_dtypes(include=np.number).columns
    n = len(num_cols)
    fig, axes = plt.subplots(nrows=n, ncols=2, figsize=(12, 3 * n))

    for i, col in enumerate(num_cols):
        sns.boxplot(x=data[col], ax=axes[i, 0])
        axes[i, 0].set_title(f'Boxplot of {col}')

        sns.histplot(data[col], kde=True, bins=30, ax=axes[i, 1])
        axes[i, 1].set_title(f'Histogram of {col}')

    fig.suptitle("Boxplot and Histogram of Numerical Columns", fontsize=20, y=1)
    plt.tight_layout()
    plt.show()
```

In [11]:
```
num_plot(data)
```

Boxplot and Histogram of Numerical Columns



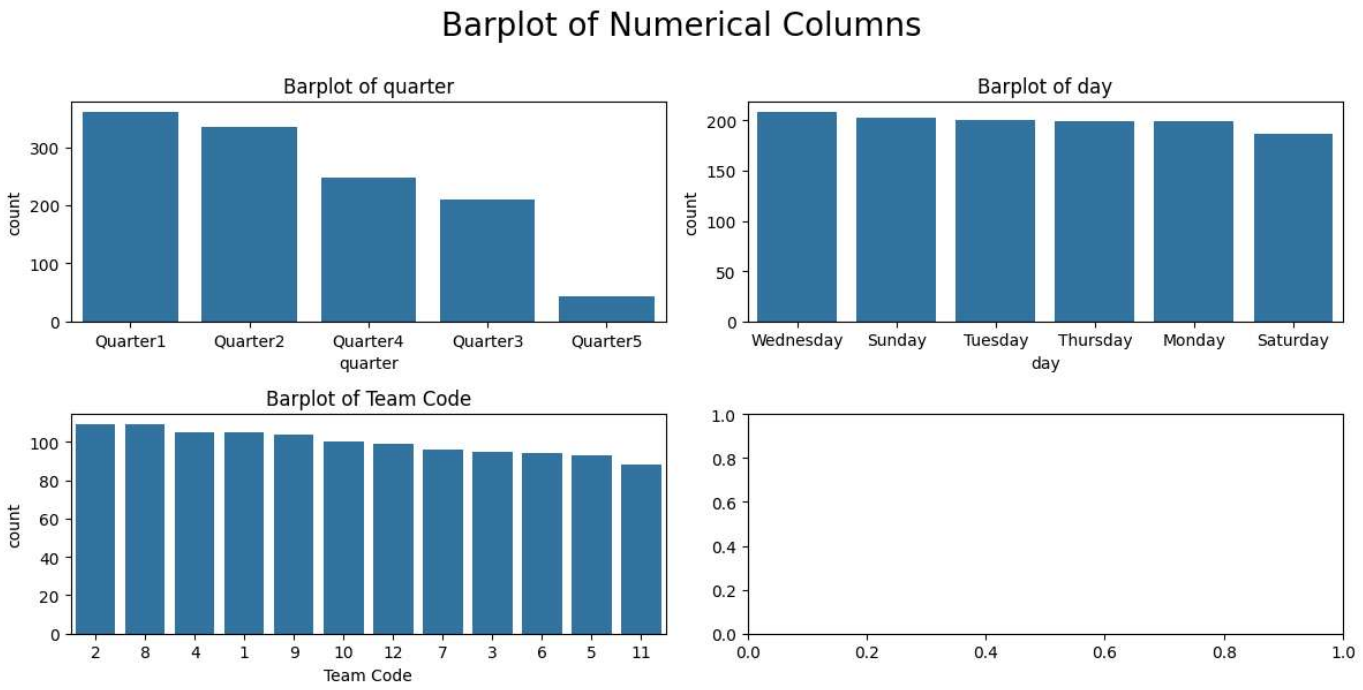


- wip, over_time, incentive, idle_time, and idle_men has outliers
- no_of_workers and productivity_score have negative values

Plot for Categorical Data

In [12]:
```
def cat_plot(data):
    cat_cols = data.select_dtypes(exclude=['number', 'datetime']).columns
    n = len(cat_cols)
    fig, axes = plt.subplots(nrows=round(n/2), ncols=2, figsize=(12, 2 * n))

    axes = axes.flatten()
    for i, col in enumerate(cat_cols):
        order = data[col].value_counts().index
        sns.countplot(data=data, x=col, ax=axes[i], order=order)
        axes[i].set_title(f'Barplot of {col}')

    fig.suptitle("Barplot of Numerical Columns", fontsize=20, y=1)
    plt.tight_layout()
    plt.show()
```

In [13]:
```
cat_plot(data)
```



- The distibution of team_code and day are pretty uniform, which is nice

Outliers

Work In Progress

In [14]:
```
data[data["wip"]>5000]
```

Out[14]:

	date	quarter	day	Team Code	smv	wip	over_time	incentive	idle_time	idle_men	no_of
561	2015-02-02	Quarter1	Monday	1	22.94	16882.0	7020	113	0.0	0	
563	2015-02-02	Quarter1	Monday	2	22.52	21385.0	7020	88	0.0	0	
564	2015-02-02	Quarter1	Monday	3	22.52	21266.0	6840	70	0.0	0	
565	2015-02-02	Quarter1	Monday	10	22.52	21540.0	6720	88	0.0	0	
568	2015-02-02	Quarter1	Monday	12	15.26	12261.0	3600	63	0.0	0	
569	2016-02-02	Quarter1	Monday	4	22.52	23122.0	5940	50	0.0	0	
570	2016-02-02	Quarter1	Monday	9	29.12	8992.0	6960	55	0.0	0	
572	2015-02-02	Quarter1	Monday	11	20.55	9792.0	6480	60	0.0	0	

Overtime

In [15]:
```
data[data["over_time"]>16000]
```

Out[15]:

	date	quarter	day	Team Code	smv	wip	over_time	incentive	idle_time	idle_men	no_of_s
146	2016-01-08	Quarter2	Thursday	11	12.52	287.0	25920	38	0.0	0	

Incentive

In [16]:
```
data[data["incentive"]>500]
```

Out[16]:

	date	quarter	day	Team Code	smv	wip	over_time	incentive	idle_time	idle_men	no_of_sty
1128	2015-03-09	Quarter2	Monday	11	2.90	NaN	0	960	0.0	0	
1129	2015-03-09	Quarter2	Monday	12	4.60	NaN	0	1080	0.0	0	
1130	2015-03-09	Quarter2	Monday	5	3.94	NaN	0	2880	0.0	0	
1133	2015-03-09	Quarter2	Monday	9	2.90	NaN	0	3600	0.0	0	
1137	2016-03-08	Quarter2	Monday	3	4.60	NaN	0	1440	0.0	0	
1138	2016-03-08	Quarter2	Monday	4	3.94	NaN	0	960	0.0	0	
1139	2016-03-08	Quarter2	Monday	1	3.94	NaN	0	960	0.0	0	
1143	2016-03-08	Quarter2	Monday	2	3.90	NaN	0	1200	0.0	0	
1148	2016-03-08	Quarter2	Monday	10	2.90	NaN	0	960	0.0	0	
1149	2016-03-08	Quarter2	Monday	8	3.90	NaN	0	960	0.0	0	

Idle Time

In [17]:
```
data[data["idle_time"]>50]
```

Out[17]:

	date	quarter	day	Team Code	smv	wip	over_time	incentive	idle_time	idle_men	no_o
615	2015-02-04	Quarter1	Wednesday	5	30.10	326.0	5820	0	90.0	10	
617	2015-02-04	Quarter1	Wednesday	4	30.10	287.0	6060	23	150.0	15	
650	2016-02-07	Quarter1	Saturday	7	24.26	658.0	6960	0	270.0	45	
654	2015-02-07	Quarter1	Saturday	8	24.26	652.0	6840	0	300.0	37	

Idle Men

In [18]:
```
data[data["idle_men"]>5]
```

Out[18]:

	date	quarter	day	Team Code	smv	wip	over_time	incentive	idle_time	idle_men	no
615	2015-02-04	Quarter1	Wednesday	5	30.10	326.0	5820	0	90.0	10	
617	2015-02-04	Quarter1	Wednesday	4	30.10	287.0	6060	23	150.0	15	
650	2016-02-07	Quarter1	Saturday	7	24.26	658.0	6960	0	270.0	45	
654	2015-02-07	Quarter1	Saturday	8	24.26	652.0	6840	0	300.0	37	
775	2015-02-15	Quarter3	Sunday	8	30.10	507.0	5880	40	2.0	10	
798	2015-02-16	Quarter3	Monday	8	30.10	7.0	7080	27	2.0	10	
818	2015-02-17	Quarter3	Tuesday	8	29.40	179.0	0	23	5.0	30	
822	2016-02-17	Quarter3	Tuesday	10	18.22	741.0	0	0	8.0	35	
841	2016-02-18	Quarter3	Wednesday	10	19.68	1119.0	5640	0	8.0	35	
843	2016-02-18	Quarter3	Wednesday	8	29.40	962.0	4560	0	4.5	30	
848	2015-02-19	Quarter3	Thursday	5	30.10	276.0	600	63	3.5	15	
860	2015-02-19	Quarter3	Thursday	7	30.10	444.0	0	0	5.0	20	
880	2015-02-22	Quarter4	Sunday	7	30.10	627.0	6960	0	3.5	20	
882	2015-02-22	Quarter4	Sunday	5	30.10	450.0	5700	0	4.5	25	
996	2015-03-01	Quarter1	Sunday	11	11.61	347.0	0	50	4.0	20	
1001	2015-03-01	Quarter1	Sunday	7	30.10	934.0	6960	0	3.5	15	
1046	2016-03-02	Quarter1	Tuesday	2	15.28	157.0	5400	0	6.5	30	
1085	2015-03-05	Quarter1	Thursday	7	30.10	834.0	1200	0	4.0	40	

Confusing Data

Quarter 5 data

In [19]:
```
data[data["quarter"] == "Quarter5"].head()
```

Out[19]:

	date	quarter	day	Team Code	smv	wip	over_time	incentive	idle_time	idle_men	no_of_
498	2015-01-29	Quarter5	Thursday	2	22.52	1416.0	6840	113	0.0	0	
499	2015-01-29	Quarter5	Thursday	4	4.30	NaN	1200	0	0.0	0	
500	2015-01-29	Quarter5	Thursday	3	22.52	1287.0	6840	100	0.0	0	
501	2016-01-29	Quarter5	Thursday	4	22.52	1444.0	6900	88	0.0	0	
502	2015-01-29	Quarter5	Thursday	10	22.52	1088.0	6720	88	0.0	0	

In [20]:
```
quarter_min_max = []

for quarter in data["quarter"].unique():
    quarter_date = data[data["quarter"] == quarter]["date"]
    mmdd = quarter_date.dt.strftime("%m-%d")

    min_mmdd = mmdd.min()
    max_mmdd = mmdd.max()

    min_date = quarter_date[mmdd == min_mmdd].iloc[0].strftime("%d %b")
    max_date = quarter_date[mmdd == max_mmdd].iloc[0].strftime("%d %b")

    quarter_min_max.append({"Quarter": quarter, "Min Date": min_date, "Max Date": max_date})

quarter_min_max_df = pd.DataFrame(quarter_min_max).set_index("Quarter")
quarter_min_max_df.index.name = None
quarter_min_max_df
```

Out[20]:

	Min Date	Max Date
Quarter1	01 Jan	07 Mar
Quarter2	08 Jan	11 Mar
Quarter3	15 Jan	19 Feb
Quarter4	22 Jan	28 Feb
Quarter5	29 Jan	31 Jan

Quarter 5's date is between 29 jan and 31 jan, which is part of quarter 1

Negative no_of_workers

In [21]:
```
data[data["no_of_workers"]<0]
```

Out[21]:

	date	quarter	day	Team Code	smv	wip	over_time	incentive	idle_time	idle_men	no_
66	2016-01-05	Quarter1	Monday	9	2.90	NaN	1920	0	0.0	0	
79	2016-01-05	Quarter1	Monday	3	19.87	944.0	6600	45	0.0	0	
503	2015-01-29	Quarter5	Thursday	6	2.90	NaN	1200	0	0.0	0	
610	2015-02-04	Quarter1	Wednesday	6	18.79	941.0	3360	30	0.0	0	
697	2016-02-10	Quarter2	Tuesday	2	22.52	1512.0	0	88	0.0	0	

Negative productivity_score

In [22]:
```
data[data["productivity_score"]<0]
```

Out[22]:

	date	quarter	day	Team Code	smv	wip	over_time	incentive	idle_time	idle_men	no_of_
1120	2015-03-08	Quarter2	Sunday	10	21.82	1335.0	6000	30	0.0	0	

EDA Insights

1. **Missing Values**
   - wip has 506 missing values out of 1197 rows, which makes it 42% of the data.
   - Other columns do not have any missing values.
2. **Duplicated Values**
   - There are no duplicated values in the dataset.
3. **Datatypes**
   - Date's datatype is object, it need to be converted as datetime.
   - Team Code datatype is int, since it does not show oreder, it need to be converted as category.
   - wip datatype can be converted to int
   - no_of_workers datatype is float, since worker cannot be a decimal, it need to be converted as int.
4. **Inconsistent Data**
   - Date format is not consistent.
   - Quarter column has "Quarter5" data.
5. **Data Visualization**
   - Several columns has outliers.
   - Overtime value are pretty big.
   - Several columns has negative values, where is shouldn't.
   - Most of the numerical data distribution are skewed.
   - Most of the categorical data distribution are uniform.
6. **Outliers**
   - wip, overtime, incentive, and idle_time have extreme outliers.
7. **Confusing Data**

- Quarter 5 data can be changed into quarter 1.
- no_of_workers and productivity_score have values under 0.

Data Cleaning

To-do:

- Missing Value Handling (wip).
- Convert wip and no_of_workers to int.
- Convert Quarter 5 to Quarter 1.
- Convert overtime to hours.
- Drop extreme outliers on wip, overtime, incentive, and idle_time.
- Drop negative values on no_of_workers and productivity_score.

Handle Missing Values

In [23]:
```
data["wip"] = data["wip"].fillna(0)
data.isna().sum()
```

Out[23]:
```
date                   0
quarter                0
day                    0
Team Code              0
smv                    0
wip                    0
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
```

Convert Float to Int

In [24]:
```
data["wip"] = data["wip"].astype(int)
data["no_of_workers"] = data["no_of_workers"].astype(int)
data.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   category
 4   smv                  1197 non-null   float64
 5   wip                  1197 non-null   int64
 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   int64
 12  productivity_score   1197 non-null   float64
dtypes: category(1), datetime64[ns](1), float64(3), int64(6), object(2)
memory usage: 113.9+ KB
```

Convert Quarter5 to Quarter1

In [25]:
```
data["quarter"] = data["quarter"].replace("Quarter5", "Quarter1")
data["quarter"].unique()
```

Out[25]:
```
array(['Quarter1', 'Quarter2', 'Quarter3', 'Quarter4'], dtype=object)
```

Converting time columns from minutes to hours

In [26]:
```
overtime = []

overtime.append({"Step": "before", "Min Value": data['over_time'].min(), "Max Value": data['o
data['over_time'] = data['over_time'] / 60
overtime.append({"Step": "after", "Min Value": data['over_time'].min(), "Max Value": data['ov

overtime_df = pd.DataFrame(overtime).set_index("Step")
overtime_df.index.name = None
overtime_df
```

Out[26]:

	Min Value	Max Value
before	0.0	25920.0
after	0.0	432.0

Dropping Extreme Outliers

Work In Progress > 5000

In [27]:
```
data = data[data["wip"] < 5000]
```

Overtime > 260

In [28]:
```
data = data[data["over_time"] < 260]
```

Incentive > 500

In [29]:
```
data = data[data["incentive"] < 500]
```

Idle Time > 50

In [30]:
```
data = data[data["idle_time"] < 50]
```

Dropping Negative Values

Number of Workers > 0

In [31]:
```
data = data[data["no_of_workers"] > 0]
```

Productivity Score > 0

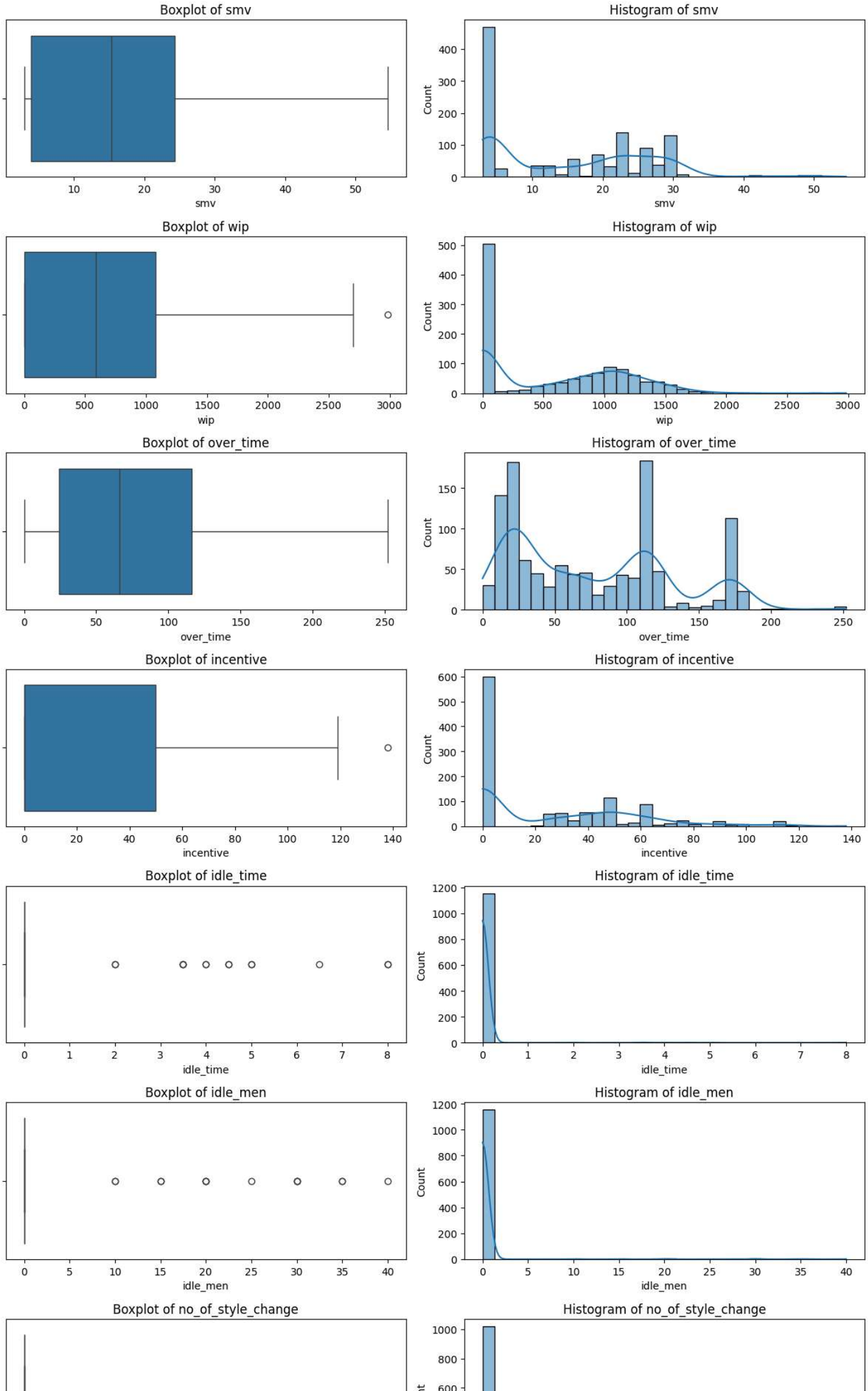
In [32]:
```
data = data[data["productivity_score"] > 0]
```

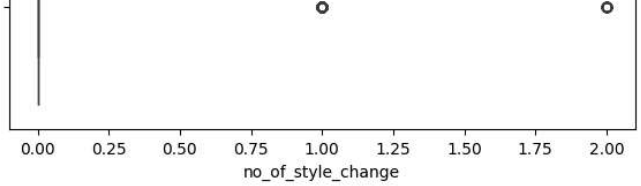
Data Visualization

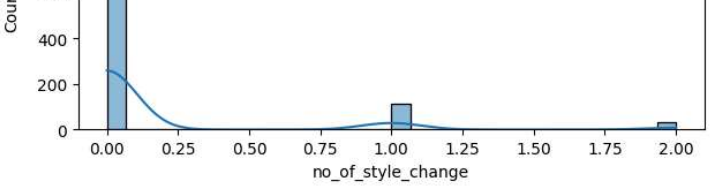
Plot of Numerical Columns

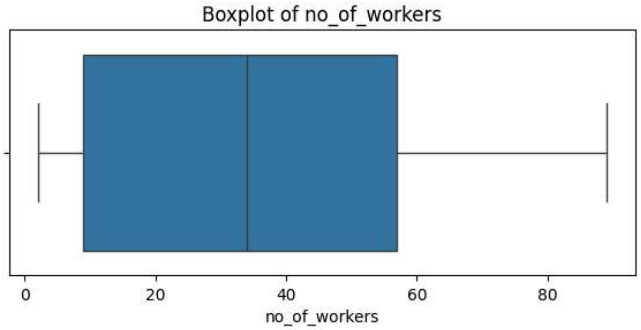
In [33]:
```
num_plot(data)
```

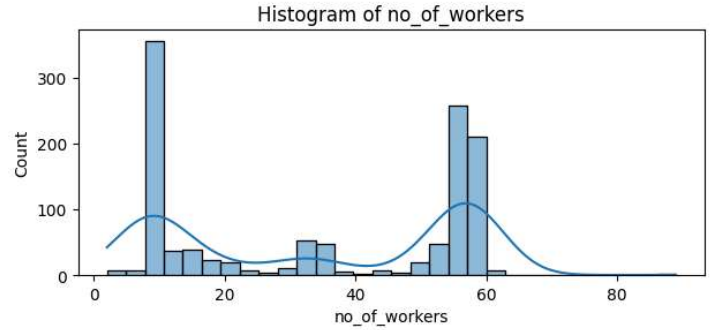
Boxplot and Histogram of Numerical Columns

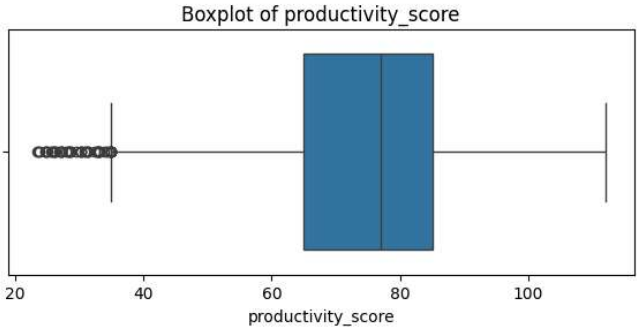


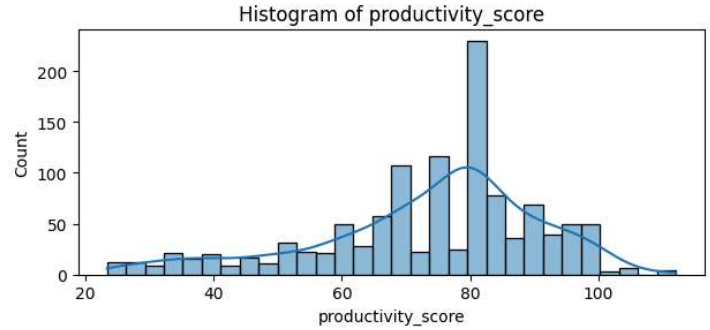






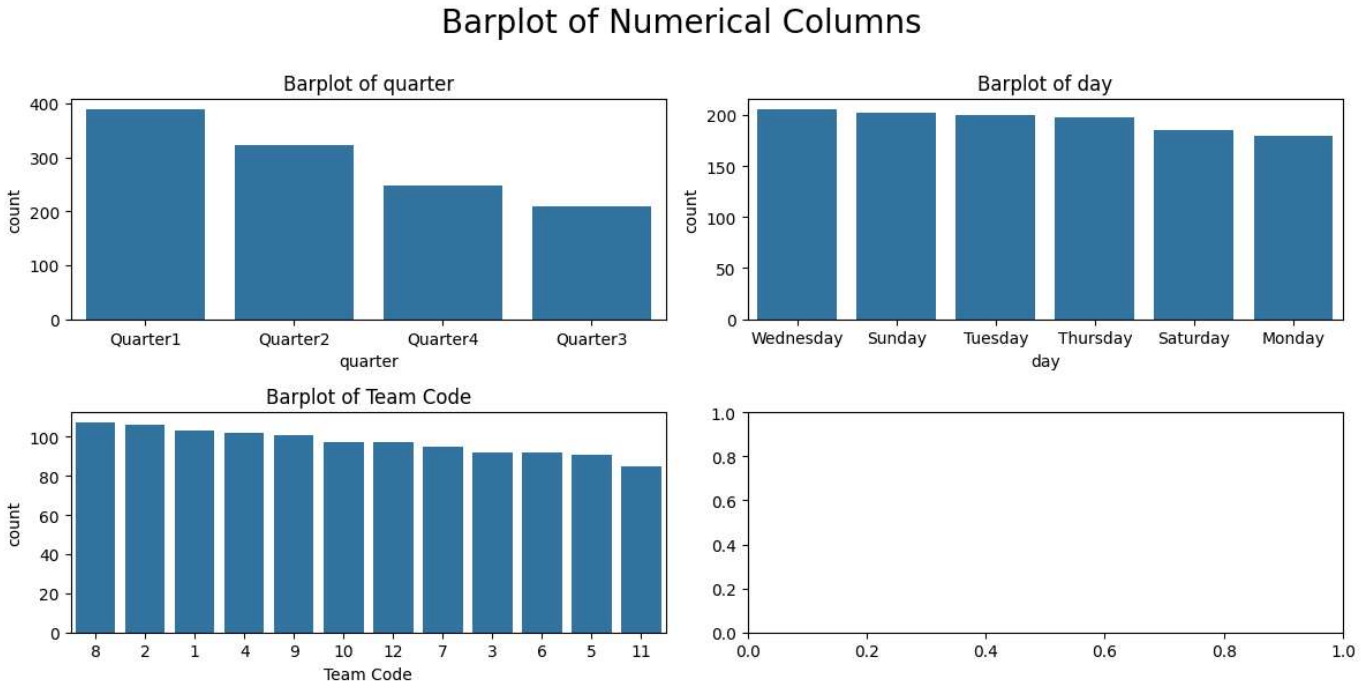






Plot of Categorical Columns

In [34]:
```
cat_plot(data)
```



Data Preprocessing

To-do:

- Split x and y data.
- Encode categorical columns.
- Split train, val, and test data.
- Scale numerical columns.

Split Data

Split x and y

In [35]:
```
x = data.drop(columns=["productivity_score", "date"])
y = data["productivity_score"]
```

Split Numerical and Categorical Columns

In [36]:
```
num_cols = x.select_dtypes(include=["number"]).columns
cat_cols = x.select_dtypes(exclude=["number"]).columns
```

Data Encoding

Ordinal Encoding

In [37]:
```
quarter_encoder = OrdinalEncoder(categories=[['Quarter1', 'Quarter2', 'Quarter3', 'Quarter4']

x[['quarter']] = quarter_encoder.fit_transform(x[['quarter']])
x.head()
```

Out[37]:

	quarter	day	Team Code	smv	wip	over_time	incentive	idle_time	idle_men	no_of_style_change
0	0.0	Thursday	8	26.16	1108	118.0	98	0.0	0	0
1	0.0	Thursday	1	3.94	0	16.0	0	0.0	0	0
2	0.0	Thursday	11	11.41	968	61.0	50	0.0	0	0
3	0.0	Thursday	12	11.41	968	61.0	50	0.0	0	0
4	0.0	Thursday	6	25.90	1170	32.0	50	0.0	0	0

One Hot Encoding

In [38]:
```
x = pd.get_dummies(x, columns=['day', 'Team Code'])
x.head()
```

Out[38]:

	quarter	smv	wip	over_time	incentive	idle_time	idle_men	no_of_style_change	no_of_workers
0	0.0	26.16	1108	118.0	98	0.0	0	0	59
1	0.0	3.94	0	16.0	0	0.0	0	0	8
2	0.0	11.41	968	61.0	50	0.0	0	0	30
3	0.0	11.41	968	61.0	50	0.0	0	0	30
4	0.0	25.90	1170	32.0	50	0.0	0	0	56

5 rows × 27 columns

Split Train, Val, and Test

In [39]:
```
x_temp, x_test, y_temp, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
x_train, x_val, y_train, y_val = train_test_split(x_temp, y_temp, test_size=0.125, random_sta
```

Data Scaling

Robust Scaler

In [40]:
```
scaler = RobustScaler()

x_train_scaled = scaler.fit_transform(x_train[num_cols])
scaled_train_df = pd.DataFrame(x_train_scaled, columns=num_cols, index=x_train.index)

x_val_scaled = scaler.transform(x_val[num_cols])
scaled_val_df = pd.DataFrame(x_val_scaled, columns=num_cols, index=x_val.index)

x_test_scaled = scaler.transform(x_test[num_cols])
scaled_test_df = pd.DataFrame(x_test_scaled, columns=num_cols, index=x_test.index)

x_train = pd.concat([x_train.drop(columns=num_cols), scaled_train_df], axis=1)
x_val = pd.concat([x_val.drop(columns=num_cols), scaled_val_df], axis=1)
x_test = pd.concat([x_test.drop(columns=num_cols), scaled_test_df], axis=1)
```

Baseline Model

Sequential Model

In [41]:
```
seq_model = keras.Sequential()
seq_model.add(layers.Dense(64, activation="relu", name="hidden_layer_1"))
seq_model.add(layers.Dense(64, activation="relu", name="hidden_layer_2"))
seq_model.add(layers.Dense(1, name="output"))

seq_model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae', 'mse'])

seq_model.fit(x_train, y_train, epochs=50, batch_size=32, validation_data=(x_val, y_val))
```

```
Epoch 1/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 5504.7612 - mae: 71.8252 - mse: 5504.7612 - val
_loss: 5668.9961 - val_mae: 73.7699 - val_mse: 5668.9961
Epoch 2/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 5217.4092 - mae: 69.7702 - mse: 5217.4092 - val
_loss: 5136.8867 - val_mae: 70.0348 - val_mse: 5136.8867
Epoch 3/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 4714.4067 - mae: 66.1176 - mse: 4714.4067 - val
_loss: 3936.9717 - val_mae: 60.6881 - val_mse: 3936.9714
Epoch 4/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 3333.2295 - mae: 54.2312 - mse: 3333.2295 - val
_loss: 2107.8555 - val_mae: 42.4624 - val_mse: 2107.8555
Epoch 5/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 1608.0673 - mae: 34.9341 - mse: 1608.0673 - val
_loss: 839.9581 - val_mae: 24.4527 - val_mse: 839.9581
Epoch 6/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 758.5063 - mae: 22.4143 - mse: 758.5063 - val_l
oss: 547.5551 - val_mae: 19.8708 - val_mse: 547.5551
Epoch 7/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 573.9938 - mae: 19.8148 - mse: 573.9938 - val_l
oss: 433.6828 - val_mae: 17.7192 - val_mse: 433.6828
Epoch 8/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 463.1651 - mae: 17.7639 - mse: 463.1651 - val_l
oss: 344.2337 - val_mae: 15.6128 - val_mse: 344.2337
Epoch 9/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 340.1209 - mae: 15.0049 - mse: 340.1209 - val_l
oss: 279.9636 - val_mae: 13.7626 - val_mse: 279.9636
Epoch 10/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 314.3559 - mae: 14.1629 - mse: 314.3559 - val_l
oss: 235.6391 - val_mae: 12.2253 - val_mse: 235.6391
Epoch 11/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 264.9501 - mae: 12.6229 - mse: 264.9501 - val_l
oss: 206.3419 - val_mae: 11.0262 - val_mse: 206.3419
Epoch 12/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 241.5059 - mae: 11.8711 - mse: 241.5059 - val_l
oss: 188.3999 - val_mae: 10.2108 - val_mse: 188.3999
Epoch 13/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 247.6159 - mae: 11.6885 - mse: 247.6159 - val_l
oss: 176.6660 - val_mae: 9.7075 - val_mse: 176.6660
Epoch 14/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 250.9586 - mae: 11.7194 - mse: 250.9586 - val_l
oss: 169.3237 - val_mae: 9.4073 - val_mse: 169.3237
Epoch 15/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 208.4812 - mae: 10.5663 - mse: 208.4812 - val_l
oss: 165.8121 - val_mae: 9.2881 - val_mse: 165.8121
Epoch 16/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 225.8416 - mae: 11.1314 - mse: 225.8416 - val_l
oss: 163.7234 - val_mae: 9.2185 - val_mse: 163.7234
Epoch 17/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 211.7412 - mae: 10.8105 - mse: 211.7412 - val_l
oss: 160.9023 - val_mae: 9.0818 - val_mse: 160.9023
Epoch 18/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 221.2393 - mae: 11.0987 - mse: 221.2393 - val_l
oss: 159.9969 - val_mae: 9.0332 - val_mse: 159.9969
Epoch 19/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 202.8541 - mae: 10.4705 - mse: 202.8541 - val_l
oss: 159.3655 - val_mae: 9.0114 - val_mse: 159.3655
Epoch 20/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 217.1963 - mae: 10.7137 - mse: 217.1963 - val_l
oss: 158.5885 - val_mae: 8.9427 - val_mse: 158.5885
Epoch 21/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 223.5965 - mae: 10.8788 - mse: 223.5965 - val_l
oss: 157.7520 - val_mae: 8.9232 - val_mse: 157.7520
Epoch 22/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 221.9824 - mae: 11.0945 - mse: 221.9824 - val_l
oss: 158.3699 - val_mae: 8.9233 - val_mse: 158.3699
```

```
Epoch 23/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 192.7599 - mae: 9.9693 - mse: 192.7599 - val_lo
ss: 157.7485 - val_mae: 8.9143 - val_mse: 157.7485
Epoch 24/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 207.2763 - mae: 10.3846 - mse: 207.2763 - val_l
oss: 157.2777 - val_mae: 8.8881 - val_mse: 157.2777
Epoch 25/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 192.0119 - mae: 10.1787 - mse: 192.0119 - val_l
oss: 156.0682 - val_mae: 8.8433 - val_mse: 156.0682
Epoch 26/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 222.9399 - mae: 11.0890 - mse: 222.9399 - val_l
oss: 156.9642 - val_mae: 8.8703 - val_mse: 156.9642
Epoch 27/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 224.5754 - mae: 11.0550 - mse: 224.5754 - val_l
oss: 154.7968 - val_mae: 8.7730 - val_mse: 154.7968
Epoch 28/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 225.3346 - mae: 10.7437 - mse: 225.3346 - val_l
oss: 154.5772 - val_mae: 8.7666 - val_mse: 154.5772
Epoch 29/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 214.8009 - mae: 10.5977 - mse: 214.8009 - val_l
oss: 154.7289 - val_mae: 8.7527 - val_mse: 154.7289
Epoch 30/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 178.4512 - mae: 9.7625 - mse: 178.4512 - val_lo
ss: 154.6402 - val_mae: 8.7396 - val_mse: 154.6402
Epoch 31/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 193.6827 - mae: 10.0211 - mse: 193.6827 - val_l
oss: 155.5786 - val_mae: 8.7768 - val_mse: 155.5786
Epoch 32/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 194.7743 - mae: 10.3503 - mse: 194.7743 - val_l
oss: 154.7045 - val_mae: 8.7486 - val_mse: 154.7045
Epoch 33/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 200.4285 - mae: 10.4102 - mse: 200.4285 - val_l
oss: 155.5760 - val_mae: 8.8312 - val_mse: 155.5760
Epoch 34/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 220.9917 - mae: 10.7125 - mse: 220.9917 - val_l
oss: 152.9458 - val_mae: 8.6773 - val_mse: 152.9458
Epoch 35/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 196.3436 - mae: 10.2015 - mse: 196.3436 - val_l
oss: 154.1483 - val_mae: 8.7234 - val_mse: 154.1483
Epoch 36/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 209.3959 - mae: 10.7489 - mse: 209.3959 - val_l
oss: 154.0728 - val_mae: 8.6889 - val_mse: 154.0728
Epoch 37/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 212.3273 - mae: 10.7120 - mse: 212.3273 - val_l
oss: 153.5978 - val_mae: 8.6723 - val_mse: 153.5978
Epoch 38/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 191.5018 - mae: 10.1543 - mse: 191.5018 - val_l
oss: 153.3468 - val_mae: 8.7308 - val_mse: 153.3468
Epoch 39/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 233.4690 - mae: 11.1455 - mse: 233.4690 - val_l
oss: 153.1415 - val_mae: 8.6478 - val_mse: 153.1415
Epoch 40/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 189.4325 - mae: 9.9484 - mse: 189.4325 - val_lo
ss: 152.3801 - val_mae: 8.6242 - val_mse: 152.3801
Epoch 41/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 197.1188 - mae: 10.2076 - mse: 197.1188 - val_l
oss: 153.7325 - val_mae: 8.6889 - val_mse: 153.7325
Epoch 42/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 201.5232 - mae: 10.1861 - mse: 201.5232 - val_l
oss: 152.4808 - val_mae: 8.6307 - val_mse: 152.4808
Epoch 43/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 192.4383 - mae: 10.0668 - mse: 192.4383 - val_l
oss: 152.3188 - val_mae: 8.5864 - val_mse: 152.3188
Epoch 44/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 208.6412 - mae: 10.3147 - mse: 208.6412 - val_l
oss: 152.1467 - val_mae: 8.5679 - val_mse: 152.1467
```

```
Epoch 45/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 199.4309 - mae: 10.1880 - mse: 199.4309 - val_l
oss: 153.6208 - val_mae: 8.7242 - val_mse: 153.6208
Epoch 46/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 182.3600 - mae: 9.7956 - mse: 182.3600 - val_lo
ss: 152.7604 - val_mae: 8.6333 - val_mse: 152.7604
Epoch 47/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 207.1481 - mae: 10.3265 - mse: 207.1481 - val_l
oss: 151.9153 - val_mae: 8.5254 - val_mse: 151.9153
Epoch 48/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 214.4109 - mae: 10.5944 - mse: 214.4109 - val_l
oss: 152.7718 - val_mae: 8.6249 - val_mse: 152.7718
Epoch 49/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 184.6982 - mae: 9.6214 - mse: 184.6982 - val_lo
ss: 151.6228 - val_mae: 8.5050 - val_mse: 151.6228
Epoch 50/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 192.8466 - mae: 10.0386 - mse: 192.8466 - val_l
oss: 154.2804 - val_mae: 8.7244 - val_mse: 154.2804
```

Out[41]: <keras.src.callbacks.history.History at 0x16dba133c80>

Functional Model

In [ ]:
```python
inputs = Input(shape=(x_train.shape[1],), name="input_layer")
x = layers.Dense(128, activation="relu", name="hidden_layer_1")(inputs)
x = layers.Dense(128, activation="relu", name="hidden_layer_2")(x)
outputs = layers.Dense(1, name="output")(x)

func_model = Model(inputs=inputs, outputs=outputs, name="Functional_Model")

func_model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae', 'mse'])

func_model.fit(x_train, y_train, epochs=50, batch_size=32, validation_data=(x_val, y_val))
```

```
Epoch 1/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 5477.3179 - mae: 71.7670 - mse: 5477.3179 - val
_loss: 5213.9565 - val_mae: 70.6248 - val_mse: 5213.9565
Epoch 2/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 4490.1055 - mae: 64.1722 - mse: 4490.1055 - val
_loss: 3242.7319 - val_mae: 54.5846 - val_mse: 3242.7319
Epoch 3/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 2487.1047 - mae: 45.2422 - mse: 2487.1047 - val
_loss: 860.0274 - val_mae: 24.6849 - val_mse: 860.0274
Epoch 4/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 783.0637 - mae: 22.5482 - mse: 783.0637 - val_l
oss: 431.2227 - val_mae: 17.6682 - val_mse: 431.2227
Epoch 5/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 444.8289 - mae: 17.1419 - mse: 444.8289 - val_l
oss: 310.4765 - val_mae: 14.7827 - val_mse: 310.4765
Epoch 6/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 359.7500 - mae: 14.9296 - mse: 359.7500 - val_l
oss: 235.0690 - val_mae: 12.2458 - val_mse: 235.0690
Epoch 7/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 260.7480 - mae: 12.2070 - mse: 260.7480 - val_l
oss: 197.8334 - val_mae: 10.8488 - val_mse: 197.8334
Epoch 8/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 257.5461 - mae: 11.9461 - mse: 257.5461 - val_l
oss: 179.4506 - val_mae: 10.0482 - val_mse: 179.4506
Epoch 9/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 226.3546 - mae: 11.2446 - mse: 226.3546 - val_l
oss: 171.2741 - val_mae: 9.5607 - val_mse: 171.2741
Epoch 10/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 242.1820 - mae: 11.3437 - mse: 242.1820 - val_l
oss: 168.8908 - val_mae: 9.4413 - val_mse: 168.8908
Epoch 11/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 223.2169 - mae: 11.0081 - mse: 223.2169 - val_l
oss: 166.6817 - val_mae: 9.3184 - val_mse: 166.6817
Epoch 12/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 221.7463 - mae: 11.0148 - mse: 221.7463 - val_l
oss: 166.1810 - val_mae: 9.2503 - val_mse: 166.1810
Epoch 13/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 210.5415 - mae: 10.7339 - mse: 210.5415 - val_l
oss: 163.2874 - val_mae: 9.0842 - val_mse: 163.2874
Epoch 14/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 218.2260 - mae: 10.8501 - mse: 218.2260 - val_l
oss: 164.3493 - val_mae: 9.1591 - val_mse: 164.3493
Epoch 15/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 236.2176 - mae: 11.3755 - mse: 236.2176 - val_l
oss: 164.9371 - val_mae: 9.1208 - val_mse: 164.9371
Epoch 16/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 225.2575 - mae: 11.0015 - mse: 225.2575 - val_l
oss: 162.7168 - val_mae: 9.0376 - val_mse: 162.7168
Epoch 17/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 237.8828 - mae: 11.2223 - mse: 237.8828 - val_l
oss: 162.1831 - val_mae: 8.9942 - val_mse: 162.1831
Epoch 18/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 215.4761 - mae: 10.6990 - mse: 215.4761 - val_l
oss: 162.1312 - val_mae: 8.9661 - val_mse: 162.1312
Epoch 19/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 219.2630 - mae: 10.7275 - mse: 219.2630 - val_l
oss: 163.0601 - val_mae: 9.0584 - val_mse: 163.0601
Epoch 20/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 222.1808 - mae: 10.9365 - mse: 222.1808 - val_l
oss: 161.7898 - val_mae: 8.9286 - val_mse: 161.7898
Epoch 21/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 202.2719 - mae: 10.2330 - mse: 202.2719 - val_l
oss: 162.1738 - val_mae: 9.0010 - val_mse: 162.1738
Epoch 22/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 220.7532 - mae: 10.7877 - mse: 220.7532 - val_l
oss: 161.0386 - val_mae: 8.9432 - val_mse: 161.0386
```

```
Epoch 23/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 198.7735 - mae: 10.2084 - mse: 198.7735 - val_l
oss: 160.5444 - val_mae: 8.8854 - val_mse: 160.5444
Epoch 24/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 199.2764 - mae: 10.0374 - mse: 199.2764 - val_l
oss: 161.4732 - val_mae: 8.8638 - val_mse: 161.4732
Epoch 25/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 230.3239 - mae: 10.9256 - mse: 230.3239 - val_l
oss: 160.2961 - val_mae: 8.8455 - val_mse: 160.2961
Epoch 26/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 183.9105 - mae: 9.7651 - mse: 183.9105 - val_lo
ss: 161.8480 - val_mae: 9.0717 - val_mse: 161.8480
Epoch 27/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 200.8533 - mae: 10.3932 - mse: 200.8533 - val_l
oss: 160.7993 - val_mae: 8.8445 - val_mse: 160.7993
Epoch 28/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 213.2769 - mae: 10.4318 - mse: 213.2769 - val_l
oss: 159.6494 - val_mae: 8.7873 - val_mse: 159.6494
Epoch 29/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 205.3949 - mae: 10.3988 - mse: 205.3949 - val_l
oss: 159.6962 - val_mae: 8.8483 - val_mse: 159.6962
Epoch 30/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 207.1334 - mae: 10.3187 - mse: 207.1334 - val_l
oss: 159.8912 - val_mae: 8.8844 - val_mse: 159.8912
Epoch 31/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 184.7263 - mae: 9.8617 - mse: 184.7263 - val_lo
ss: 158.8924 - val_mae: 8.7305 - val_mse: 158.8924
Epoch 32/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 213.4040 - mae: 10.3371 - mse: 213.4040 - val_l
oss: 158.6633 - val_mae: 8.7578 - val_mse: 158.6633
Epoch 33/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 193.3154 - mae: 10.0528 - mse: 193.3154 - val_l
oss: 161.8643 - val_mae: 9.0006 - val_mse: 161.8643
Epoch 34/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 206.0676 - mae: 10.4603 - mse: 206.0676 - val_l
oss: 161.5537 - val_mae: 9.0280 - val_mse: 161.5537
Epoch 35/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 192.3108 - mae: 9.9359 - mse: 192.3108 - val_lo
ss: 160.3122 - val_mae: 8.8167 - val_mse: 160.3122
Epoch 36/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 189.0541 - mae: 9.9315 - mse: 189.0541 - val_lo
ss: 159.6208 - val_mae: 8.8521 - val_mse: 159.6208
Epoch 37/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 188.3832 - mae: 9.9047 - mse: 188.3832 - val_lo
ss: 158.6882 - val_mae: 8.8166 - val_mse: 158.6882
Epoch 38/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 204.4756 - mae: 10.1840 - mse: 204.4756 - val_l
oss: 159.7347 - val_mae: 8.8535 - val_mse: 159.7347
Epoch 39/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 179.9845 - mae: 9.6391 - mse: 179.9845 - val_lo
ss: 164.3344 - val_mae: 9.1313 - val_mse: 164.3344
Epoch 40/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 203.0279 - mae: 10.4721 - mse: 203.0279 - val_l
oss: 159.3918 - val_mae: 8.8268 - val_mse: 159.3918
Epoch 41/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 179.8627 - mae: 9.5583 - mse: 179.8627 - val_lo
ss: 159.7397 - val_mae: 8.8260 - val_mse: 159.7397
Epoch 42/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 177.5543 - mae: 9.5207 - mse: 177.5543 - val_lo
ss: 158.6549 - val_mae: 8.6877 - val_mse: 158.6549
Epoch 43/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 192.7318 - mae: 9.8331 - mse: 192.7318 - val_lo
ss: 160.7906 - val_mae: 8.8783 - val_mse: 160.7906
Epoch 44/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 184.8303 - mae: 9.6322 - mse: 184.8303 - val_lo
ss: 159.4144 - val_mae: 8.7556 - val_mse: 159.4144
```

```
Epoch 45/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 197.2618 - mae: 10.0086 - mse: 197.2618 - val_l
oss: 158.9460 - val_mae: 8.7659 - val_mse: 158.9460
Epoch 46/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 191.2375 - mae: 9.9924 - mse: 191.2375 - val_lo
ss: 163.4897 - val_mae: 9.0674 - val_mse: 163.4897
Epoch 47/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 199.8421 - mae: 10.1311 - mse: 199.8421 - val_l
oss: 166.8394 - val_mae: 9.2298 - val_mse: 166.8394
Epoch 48/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 191.6455 - mae: 9.9146 - mse: 191.6455 - val_lo
ss: 159.0334 - val_mae: 8.7978 - val_mse: 159.0334
Epoch 49/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 204.7462 - mae: 10.2522 - mse: 204.7462 - val_l
oss: 160.6244 - val_mae: 8.7502 - val_mse: 160.6244
Epoch 50/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 194.2894 - mae: 10.1433 - mse: 194.2894 - val_l
oss: 161.2140 - val_mae: 8.8670 - val_mse: 161.2140
```

Out[ ]: <keras.src.callbacks.history.History at 0x16dbd3af650>

Hyperparameter Fine-Tuning

Modified Sequential Model

In [43]:
```python
def build_seq_model(hp):
    model = Sequential()

    model.add(Dense(
        units=hp.Int('units_input', min_value=16, max_value=256, step=16),
        activation='relu'
    ))

    for i in range(hp.Int('num_hidden_layers', 1, 3)):
        model.add(Dense(
            units=hp.Int(f'units_{i}', min_value=16, max_value=256, step=16),
            activation='relu'
        ))

    model.add(Dense(1))

    model.compile(
        optimizer=Adam(hp.Choice('learning_rate', [1e-2, 1e-3, 1e-4])),
        loss='mse',
        metrics=['mae']
    )

    return model
```

In [44]:
```python
seq_tuner = kt.RandomSearch(
    build_seq_model,
    objective='val_mae',
    max_trials=50,
    executions_per_trial=3,
    directory='tuning',
    project_name='seq_tuning'
)

seq_tuner.search(
    x_train, y_train,
    epochs=50,
    validation_data=(x_val, y_val),
```

```
    batch_size=32,
    verbose=1)
```

```
Reloading Tuner from tuning\seq_tuning\tuner0.json
```

Modified Functional Model

In [45]:
```
def build_func_model(hp):
    input_layer = Input(shape=(x_train.shape[1],))

    x = Dense(
        units=hp.Int('units_input', min_value=16, max_value=256, step=16),
        activation='relu'
    )(input_layer)

    for i in range(hp.Int('num_hidden_layers', 1, 3)):
        x = Dense(
            units=hp.Int(f'units_{i}', min_value=16, max_value=256, step=16),
            activation='relu'
        )(x)

    output_layer = Dense(1)(x)
    model = Model(inputs=input_layer, outputs=output_layer)

    model.compile(
        optimizer=Adam(hp.Choice('learning_rate', [1e-2, 1e-3, 1e-4])),
        loss='mse',
        metrics=['mae']
    )

    return model
```

In [46]:
```
func_tuner = kt.RandomSearch(
    build_func_model,
    objective='val_mae',
    max_trials=50,
    executions_per_trial=3,
    directory='tuning',
    project_name='func_tuning'
)

func_tuner.search(
    x_train, y_train,
    epochs=50,
    validation_data=(x_val, y_val),
    batch_size=32,
    verbose=1
)
```

```
Reloading Tuner from tuning\func_tuning\tuner0.json
```

Evaluation

Sequential Model

In [51]:
```
y_pred = seq_model.predict(x_test)

test_loss, mae, mse = seq_model.evaluate(x_test, y_test)
r2 = r2_score(y_test, y_pred)

print("\nSequential Model Evaluation:")
print(f"MAE: {mae:.4f}")
```

```
print(f"MSE: {mse:.4f}")
print(f"R² : {r2:.4f}")
```

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 157.5601 - mae: 9.5146 - mse: 157.5601

Sequential Model Evaluation:
MAE: 9.6980
MSE: 166.0035
R² : 0.3187
```

Functional Model

In [52]:
```
y_pred = func_model.predict(x_test)

test_loss, mae, mse = func_model.evaluate(x_test, y_test)
r2 = r2_score(y_test, y_pred)

print("\nFunctional Model Evaluation:")
print(f"MAE: {mae:.4f}")
print(f"MSE: {mse:.4f}")
print(f"R² : {r2:.4f}")
```

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 154.7286 - mae: 9.3816 - mse: 154.7286

Functional Model Evaluation:
MAE: 9.5577
MSE: 163.6902
R² : 0.3282
```

Modified Sequential Model

In [ ]:
```
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    best_model = seq_tuner.get_best_models(1)[0]
y_pred_best = best_model.predict(x_test)

mae_best = mean_absolute_error(y_test, y_pred_best)
mse_best = mean_squared_error(y_test, y_pred_best)
r2_best = r2_score(y_test, y_pred_best)

print("\nTuned Sequential Model Evaluation:")
print(f"MAE: {mae_best:.4f}")
print(f"MSE: {mse_best:.4f}")
print(f"R² : {r2_best:.4f}")
```

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step

Tuned Sequential Model Evaluation:
MAE: 9.0844
MSE: 154.5892
R² : 0.3656
```

Modified Functional Model

In [56]:
```
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    best_model = func_tuner.get_best_models(1)[0]
y_pred_best = best_model.predict(x_test)

mae_best = mean_absolute_error(y_test, y_pred_best)
mse_best = mean_squared_error(y_test, y_pred_best)
r2_best = r2_score(y_test, y_pred_best)
```

```
print("\nTuned Functional Model Evaluation:")
print(f"MAE: {mae_best:.4f}")
print(f"MSE: {mse_best:.4f}")
print(f"R² : {r2_best:.4f}")
```

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step

Tuned Functional Model Evaluation:
MAE: 9.2057
MSE: 158.1641
R² : 0.3509
```

Conclusion

Model	MAE	MSE	R^2
Sequential (base)	9.6980	166.0035	0.3187
Functional (base)	9.5577	163.6902	0.3282
Tuned Sequential	**9.0844**	**154.5892**	**0.3656**
Tuned Functional	9.2057	158.1641	0.3509

The evaluation of both base and tuned models reveals clear performance differences across architectures and optimization levels. Initially, the Functional model outperformed the Sequential model slightly, achieving a lower Mean Absolute Error (MAE) of 9.5577 compared to 9.6980, along with marginally better Mean Squared Error (MSE) and R^2 scores. This suggests that even without tuning, the Functional API's flexibility may offer structural advantages for this regression task. However, both base models showed relatively modest performance, with R^2 values below 0.33, indicating a need for further optimization.

After applying hyperparameter tuning, both models improved significantly, but the Sequential model showed the greatest gains. Its MAE dropped to 9.0844, MSE to 154.5892, and R^2 increased to 0.3656, making it the best performer overall. Interestingly, the tuned Functional model also improved but fell slightly short, with an MAE of 9.2057 and R^2 of 0.3509. This outcome suggests that while the Functional model is more flexible in architecture, the Sequential model may have benefited more from the specific tuning space and data characteristics in this task.