AAI-530 - Final Team Project - Group 6

PROJECT CONSTANTS

```
#
------------------------------------------------------------------------
------ #
# GOOGLE DRIVE
#
------------------------------------------------------------------------
------ #
GOOGLE_DRIVE_FOLDER_PATH = "530 Final/IoT AAI-530 Final Project"
SHORTCUT = "/content/IoT AAI-530 Final Project"

#
------------------------------------------------------------------------
------ #
# GIT REPOSITORY
#
------------------------------------------------------------------------
------ #
REPO_ROOT_FOLDER_NAME = "project"
REPO_URL = "https://github.com/aai530-group6/project.git"
REPO_DIR = f"{SHORTCUT}/{REPO_ROOT_FOLDER_NAME}"
REPO_REPORT_DIR = f"{REPO_DIR}/report"

#
------------------------------------------------------------------------
------ #
# DATASET
#
------------------------------------------------------------------------
------ #
DATASET_FILENAME = "sleep_score_data_fitbit.csv"
FITBIT_SLEEP_SCORE_DATASET_URL =
f"https://huggingface.co/datasets/aai530-group6/pmdata-sleep_scores/
resolve/main/{DATASET_FILENAME}?download=true"
```

INSTALLS

```
%%bash
pip install --quiet --progress-bar off \
    black[jupyter] \
    dataprep \
    huggingface-hub \
    isort \
    pdfkit \
```

```
    pythonnet \
    scikeras \
    WeasyPrint
```

```
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source of the following dependency conflicts.
bigframes 0.21.0 requires sqlalchemy<3.0dev,>=1.4, but you have sqlalchemy 1.3.24 which is incompatible.
ipython-sql 0.5.0 requires sqlalchemy>=2.0, but you have sqlalchemy 1.3.24 which is incompatible.
panel 1.3.8 requires bokeh<3.4.0,>=3.2.0, but you have bokeh 2.4.3 which is incompatible.
```

IMPORTS

```
import contextlib
import hashlib
import os
import pathlib
import zipfile
from datetime import datetime
from functools import partial
from itertools import product
from typing import List, Tuple

import google.colab
import isort
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import pdfkit
import requests
import seaborn as sns
import tensorflow as tf
from dataprep.clean import *
from dataprep.datasets import load_dataset
from dataprep.eda import *
from google.colab import data_table
from keras.layers import Dense, Dropout
from keras.models import Sequential
from keras.preprocessing.sequence import pad_sequences
from scikeras.wrappers import KerasRegressor
from sklearn.ensemble import RandomForestClassifier,
RandomForestRegressor
from sklearn.metrics import (classification_report, confusion_matrix,
                             mean_absolute_error, mean_squared_error,
r2_score)
from sklearn.model_selection import (GridSearchCV, RandomizedSearchCV,
```

```
                                    train_test_split)
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import LabelEncoder, MinMaxScaler,
StandardScaler
from tensorflow.keras.callbacks import (EarlyStopping, History,
LambdaCallback,
                                       LearningRateScheduler)
from tensorflow.keras.layers import (LSTM, BatchNormalization,
Bidirectional,
                                    Dense, Dropout)
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l1_l2, l2

%reload_ext autoreload
%autoreload 2
%matplotlib inline
```

HELPER FUNCTIONS

```
def unzip(zip_file_path: str, extraction_directory: str) -> None:
    """
    Unzips a zip file into a specified directory, checking if the zip
file
    exists before extraction.

    :param zip_file_path: Path to the zip file.
    :type zip_file_path: str
    :param extraction_directory: Target directory for extraction.
    :type extraction_directory: str
    :return: None
    """
    if os.path.exists(zip_file_path):
        with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
            zip_ref.extractall(extraction_directory)
        print("Extraction completed successfully.")
    else:
        print("The zip file does not exist.")


def compute_hash(file_path: str) -> str:
    """
    Computes the SHA-256 hash for a given file, reading in binary mode
and
    processing in blocks for efficiency with large files.

    :param file_path: Path to the file for hash computation.
    :type file_path: str
    :return: Hexadecimal string of the file's SHA-256 hash.
```

```python
    :rtype: str
    """
    sha256_hash = hashlib.sha256()
    with open(file_path, 'rb') as f:
        for byte_block in iter(lambda: f.read(4096), b""):
            sha256_hash.update(byte_block)
    return sha256_hash.hexdigest()


def should_download(url: str, save_path: str) -> bool:
    """
    Determines if a file should be downloaded by checking its
existence and
    comparing the content hash with the online version.

    :param url: URL of the file to potentially download.
    :type url: str
    :param save_path: Local path where the file would be saved.
    :type save_path: str
    :return: True if download is needed, False otherwise.
    :rtype: bool
    """
    try:
        response = requests.get(url, stream=True)
        # DON'T DOWNLOAD IF INACCESSIBLE
        if response.status_code != 200:
            return False
        # DON'T DOWNLOAD IF SAME FILE
        downloaded_hash = hashlib.sha256(response.content).hexdigest()
        if os.path.exists(save_path):
            existing_hash = compute_hash(save_path)
            if existing_hash == downloaded_hash:
                return False

    except Exception as e:
        print(f"An error occurred: {e}")
        return False
    return True


def download(url: str, save_path: str) -> str:
    """
    Downloads a file from a URL to a local path if it's absent or
outdated.
    Raises an exception for non-200 HTTP status during download.

    :param url: URL of the file to download.
    :type url: str
    :param save_path: Local path to save the downloaded file.
    :type save_path: str
```

```
    :raises Exception: For non-200 HTTP status during download.
    :return: None. Directly writes to disk if download occurs.
    :rtype: None
    """
    if should_download(url, save_path):
        response = requests.get(url)
        if response.status_code != 200:
            raise Exception(f"DOWNLOAD FAILED:
{response.status_code}")
        with open(save_path, 'wb') as f:
            f.write(response.content)
    print(f"DOWNLOADED: {save_path}")
```

SETUP

```
#
----------------------------------------------------------------------
------ #
# CREATE GOOGLE DRIVE FOLDER SHORTCUT IN TOP-LEVEL UI FILE BROWSER
DIRECTORY
#
----------------------------------------------------------------------
------ #
# MOUNT GOOGLE DRIVE
with contextlib.redirect_stdout(open(os.devnull, 'w')):
    google.colab.drive.mount("/content/drive", force_remount=True)

# DEFINE PATHS
base_drive_path = pathlib.Path("/content/drive/My Drive")
project_path = base_drive_path / GOOGLE_DRIVE_FOLDER_PATH
shortcut_folder_name = GOOGLE_DRIVE_FOLDER_PATH.split('/')[-1]  #
HANDLE NESTED
shortcut_path = pathlib.Path(f"/content/{shortcut_folder_name}")

# ENSURE PROJECT FOLDER AND PARENT FOLDERS
project_path.mkdir(parents=True, exist_ok=True)

# CREATE FOLDER SHORTCUT IF NON-EXISTENT
if not shortcut_path.exists() or not shortcut_path.is_symlink():
    if shortcut_path.exists():
        shortcut_path.unlink()
    shortcut_path.symlink_to(project_path, target_is_directory=True)
print(f"SHORTCUT: {shortcut_path} --> {project_path}")
```

```
SHORTCUT: /content/IoT AAI-530 Final Project --> /content/drive/My
Drive/530 Final/IoT AAI-530 Final Project
```

```
#
----------------------------------------------------------------------
------ #
```

```
# CLONE PROJECT REPOSITORY
#
-----------------------------------------------------------------------
------ #
if not os.path.exists(REPO_DIR):
    !cd "{SHORTCUT}" && git clone "{REPO_URL}"
print(f"CLONED: {REPO_DIR}")
```

```
CLONED: /content/IoT AAI-530 Final Project/project
```

```
#
-----------------------------------------------------------------------
------ #
# DOWNLOAD DATASET
#
-----------------------------------------------------------------------
------ #
DATASET_FILEPATH = os.path.join(SHORTCUT, DATASET_FILENAME)
download(FITBIT_SLEEP_SCORE_DATASET_URL, DATASET_FILEPATH)
```

```
DOWNLOADED: /content/IoT AAI-530 Final
Project/sleep_score_data_fitbit.csv
```

```
#
-----------------------------------------------------------------------
------ #
# CONFIGURE MATPLOTLIB
#
-----------------------------------------------------------------------
------ #
plt.rcParams['figure.autolayout'] = True
```

```
#
-----------------------------------------------------------------------
------ #
# MISCELLANEOUS
#
-----------------------------------------------------------------------
------ #
!rm -rf /content/sample_data  # REMOVE SAMPLE DATA FOLDER
```

LOAD

```
# data_table.enable_dataframe_formatter()
df_original = pd.read_csv(DATASET_FILEPATH)
df_original.head()
```


```
{"repr_error":"'str' object has no attribute
'empty'","type":"dataframe","variable_name":"df_original"}
```

```
dfs = {}
dfs['sleep_score'] = df_original.copy()
df = dfs['sleep_score']
```

PREPROCESSING

```
# REMOVE UNNECESSARY COLUMN
if 'sleep_log_entry_id' in df.columns:
    df.drop('sleep_log_entry_id', axis=1, inplace=True)

# DATETIME
df['timestamp'] = pd.to_datetime(df['timestamp'].str[:-6], format='%Y-
%m-%dT%H:%M:%S')
df.sort_values('timestamp', inplace=True)
df['day_of_week'] = df['timestamp'].dt.day_name()
df['year'] = df['timestamp'].dt.year
df['month'] = df['timestamp'].dt.month
df['hour'] = df['timestamp'].dt.hour

# ENCODE DAY_OF_WEEK
df['day_of_week_encoded'] =
LabelEncoder().fit_transform(df['day_of_week'])

# IMPUTE MISSING VALUES
df.fillna(method='ffill', inplace=True)

# EXPORT PREPROCESSED DATASET
df.to_csv(f'{REPO_REPORT_DIR}/preprocessed_dataset.csv', index=False)
df.head()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 1836,\n  \"fields\":
[\n    {\n      \"column\": \"timestamp\",\n      \"properties\": {\n
\"dtype\": \"date\",\n        \"min\": \"2023-09-28 14:00:00\",\n
\"max\": \"2024-02-26 00:00:00\",\n        \"num_unique_values\":
1817,\n        \"samples\": [\n          \"2024-01-28 14:38:30\",\n
\"2023-11-24 16:38:00\",\n          \"2023-10-14 16:02:00\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"overall_score\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 7,\n        \"min\": 35,\n
\"max\": 94,\n        \"num_unique_values\": 50,\n        \"samples\":
[\n          68,\n          46,\n          55\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"composition_score\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2,\n        \"min\": 12,\n        \"max\": 25,\n
\"num_unique_values\": 14,\n        \"samples\": [\n          25,\n
23,\n          18\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"revitalization_score\",\n      \"properties\": {\n        \"dtype\":
```

```
\"number\",\n        \"std\": 3,\n        \"min\": 5,\n
\"max\": 25,\n        \"num_unique_values\": 19,\n        \"samples\":
[\n          20,\n          21,\n          12\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"duration_score\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
5,\n        \"min\": 3,\n        \"max\": 47,\n
\"num_unique_values\": 31,\n        \"samples\": [\n          15,\n
41,\n          30\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"deep_sleep_in_minutes\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 27,\n        \"min\": 0,\n
\"max\": 183,\n        \"num_unique_values\": 145,\n
\"samples\": [\n          63,\n          118,\n          40\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"resting_heart_rate\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 7,\n        \"min\": 44,\n
\"max\": 76,\n        \"num_unique_values\": 33,\n        \"samples\":
[\n          74,\n          48,\n          69\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"restlessness\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.03751126203084699,\n        \"min\": 0.0153846153846153,\n
\"max\": 0.2947658402203856,\n        \"num_unique_values\": 1794,\n
\"samples\": [\n          0.1394316163410301,\n
0.1207048458149779,\n          0.0552763819095477\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"day_of_week\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 7,\n        \"samples\": [\n
\"Thursday\",\n          \"Friday\",\n          \"Tuesday\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"year\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0,\n        \"min\": 2023,\n        \"max\": 2024,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          2024,\n
2023\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"month\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 4,\n        \"min\": 1,\n        \"max\": 12,\n
\"num_unique_values\": 6,\n        \"samples\": [\n          9,\n
10\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"hour\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 2,\n        \"min\": 0,\n        \"max\": 23,\n
\"num_unique_values\": 20,\n        \"samples\": [\n          14,\n
1\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
```

```
\"day_of_week_encoded\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 2,\n        \"min\": 0,\n
\"max\": 6,\n        \"num_unique_values\": 7,\n        \"samples\":
[\n          4,\n          0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df"}
```


#Exploratory Data Analysis

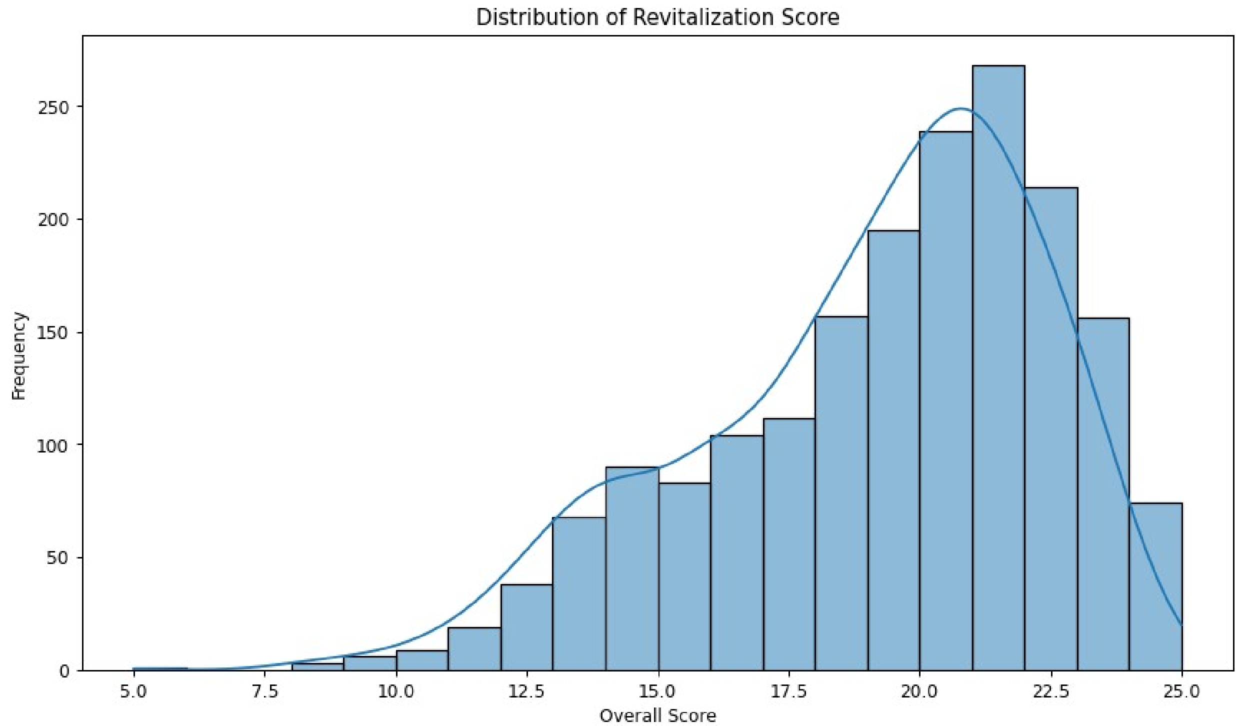
```
report = create_report(df, title="Exploratory Data Analysis")
report.save(f"{REPO_REPORT_DIR}/exploratory_data_analysis.html")
report

Computing apply-7b1b6841f03f1de3117b72b5645bdc94:   0%|          |
0/1966 [00:02<?,
?it/s]/usr/local/lib/python3.10/dist-packages/dask/core.py:121:
RuntimeWarning: invalid value encountered in divide
  return func(*(_execute_task(a, cache) for a in args))

Report has been saved to /content/IoT AAI-530 Final
Project/project/report/exploratory_data_analysis.html!



plot_missing(df)

Computing isnull-e056295165e69ce52f5697cc82b6890f:   0%|          |
0/166 [00:00<?,
?it/s]/usr/local/lib/python3.10/dist-packages/dask/core.py:121:
RuntimeWarning: invalid value encountered in divide
  return func(*(_execute_task(a, cache) for a in args))

<dataprep.eda.container.Container at 0x7e42f183bf10>

plot_correlation(df, "revitalization_score")



<dataprep.eda.container.Container at 0x7e42f16a2cb0>

plot_missing(df)

Computing isnull-e056295165e69ce52f5697cc82b6890f:   0%|          |
0/166 [00:00<?,
?it/s]/usr/local/lib/python3.10/dist-packages/dask/core.py:121:
RuntimeWarning: invalid value encountered in divide
  return func(*(_execute_task(a, cache) for a in args))

<dataprep.eda.container.Container at 0x7e437f348df0>

plot_correlation(df)
```

```
<dataprep.eda.container.Container at 0x7e42f199c6d0>
```

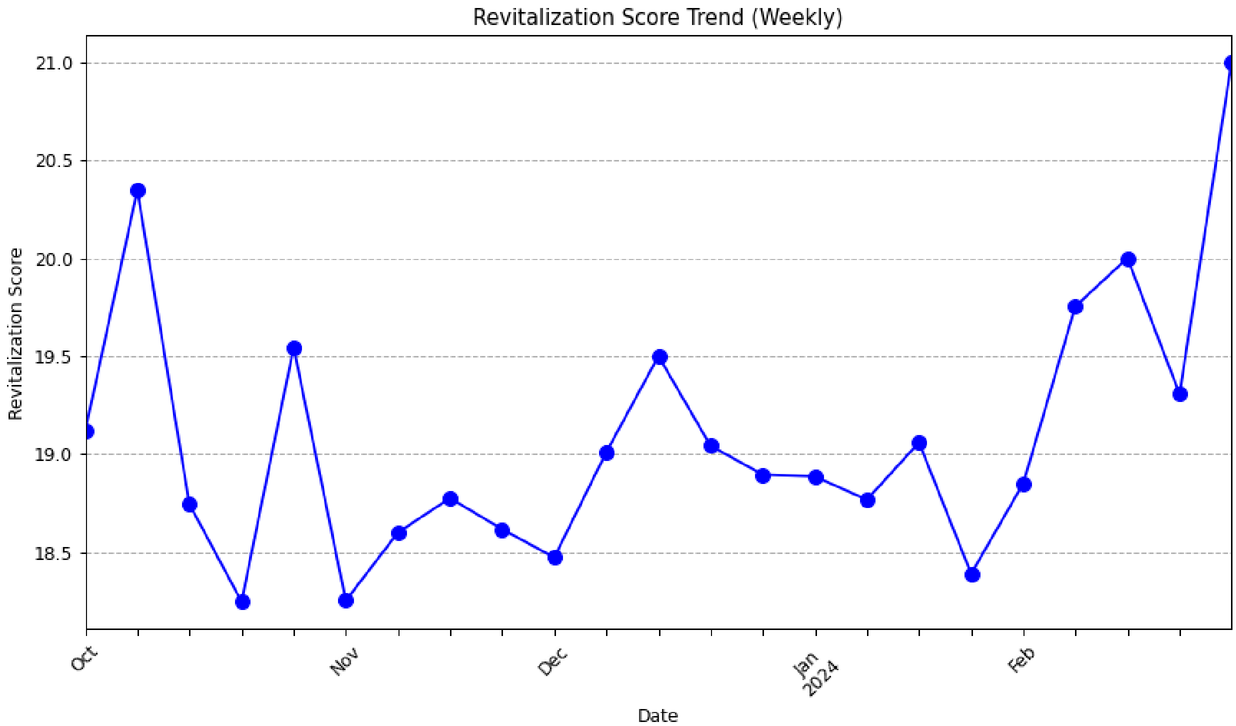
```
df.describe()
```


{"summary":"{\n \"name\": \"df\",\n \"rows\": 8,\n \"fields\": [\n   {\n   \"column\": \"overall_score\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 627.324807749624,\n   \"min\": 7.7955543982671545,\n   \"max\": 1836.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   76.46078431372548,\n   77.0,\n   1836.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"composition_score\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 643.2946429896806,\n   \"min\": 2.3909156390714728,\n   \"max\": 1836.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   19.24727668845316,\n   19.5,\n   1836.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"revitalization_score\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 643.6000612435225,\n   \"min\": 3.316146319825092,\n   \"max\": 1836.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   18.96078431372549,\n   20.0,\n   1836.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"duration_score\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 638.7663019584965,\n   \"min\": 3.0,\n   \"max\": 1836.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   38.25272331154684,\n   39.0,\n   1836.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"deep_sleep_in_minutes\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 626.0440561086242,\n   \"min\": 0.0,\n   \"max\": 1836.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   73.25816993464052,\n   72.5,\n   1836.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"resting_heart_rate\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 631.1366944405644,\n   \"min\": 7.089620258953896,\n   \"max\": 1836.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   58.5838779956427,\n   59.0,\n   1836.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"restlessness\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 649.0888941232266,\n   \"min\": 0.0153846153846153,\n   \"max\": 1836.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   0.09068656659335517,\n   0.0845005531251571,\n   1836.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\":

```
\"year\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 708.7835613624678,\n        \"min\": 0.4768245358245512,\n
\"max\": 2024.0,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          2023.349128540305,\n          2024.0,\n
0.4768245358245512\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"month\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 646.6965904872458,\n        \"min\": 1.0,\n        \"max\":
1836.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
7.679193899782135,\n          10.0,\n          1836.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"hour\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 644.8342898260952,\n
\"min\": 0.0,\n        \"max\": 1836.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
15.016339869281046,\n          15.0,\n          1836.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"day_of_week_encoded\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
648.1164672741548,\n        \"min\": 0.0,\n        \"max\": 1836.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
3.0016339869281046,\n          3.0,\n          1836.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}

if not 'df_eda' in globals():
    df_eda = df.copy()

# DISTRIBUTION OF OVERALL SLEEP SCORE
plt.figure(figsize=(10, 6), dpi=88)
sns.histplot(df_eda['revitalization_score'], kde=True, bins=20)
plt.title('Distribution of Revitalization Score')
plt.xlabel('Overall Score')
plt.ylabel('Frequency')
plt.savefig(f'{REPO_REPORT_DIR}/revitalization_score_distribution.svg'
, format='svg')
plt.show()
```

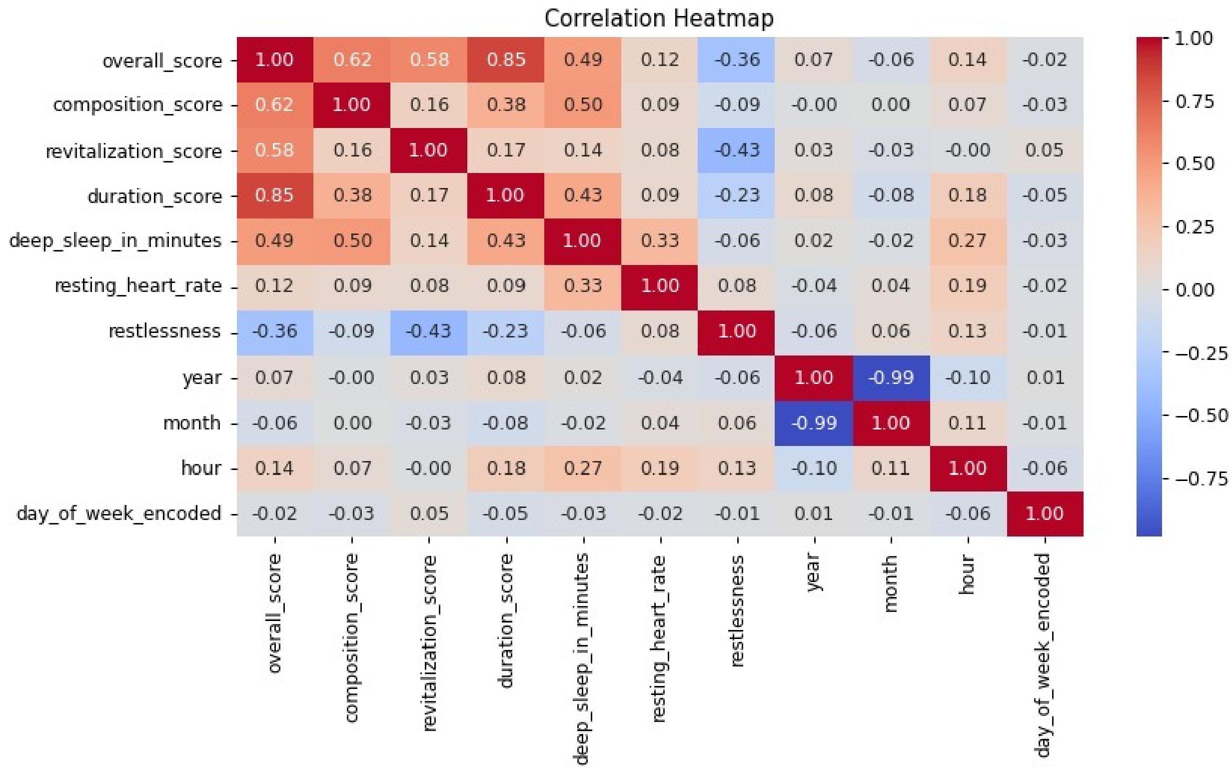



```
# OVERALL SLEEP SCORE TREND (WEEKLY)
if df_eda.index.dtype != 'datetime64[ns]':
    df_eda.set_index('timestamp', inplace=True)

plt.figure(figsize=(10, 6), dpi=88)
ax = df_eda.resample('W')['revitalization_score'].mean().plot(
    style='o-',
    color='blue',
    markersize=8
)

ax.set_title('Revitalization Score Trend (Weekly)')
ax.set_xlabel('Date')
ax.set_ylabel('Revitalization Score')
ax.grid(True, axis='y', linestyle='--')

if len(df.index) > 15:
    ax.set_xticks(df_eda.index[::2])
    plt.xticks(rotation=45)

plt.tight_layout()
plt.savefig(f"{REPO_REPORT_DIR}/weekly_revitalization_score_trend.svg"
, format='svg')
plt.show()
```

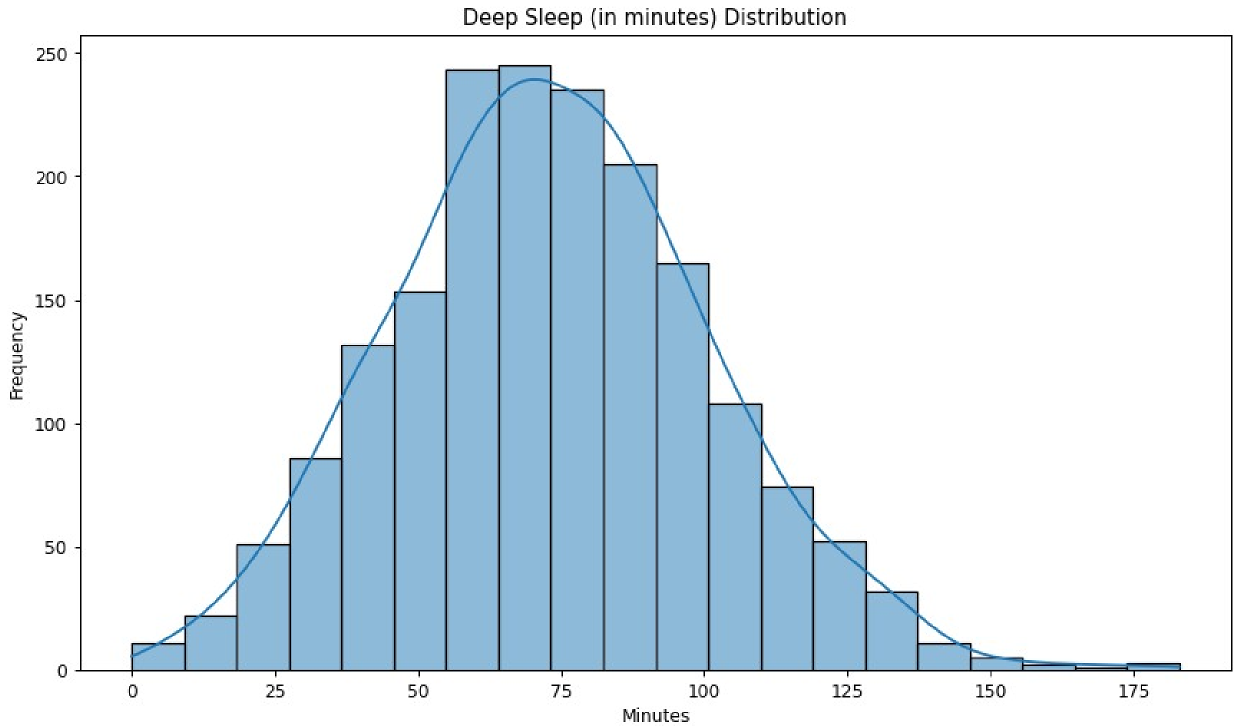



```
# CORRELATION HEATMAP
plt.figure(figsize=(10, 6), dpi=88)
sns.heatmap(df_eda.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.savefig(f'{REPO_REPORT_DIR}/correlation_heatmap.svg',
format='svg')
plt.show()
```

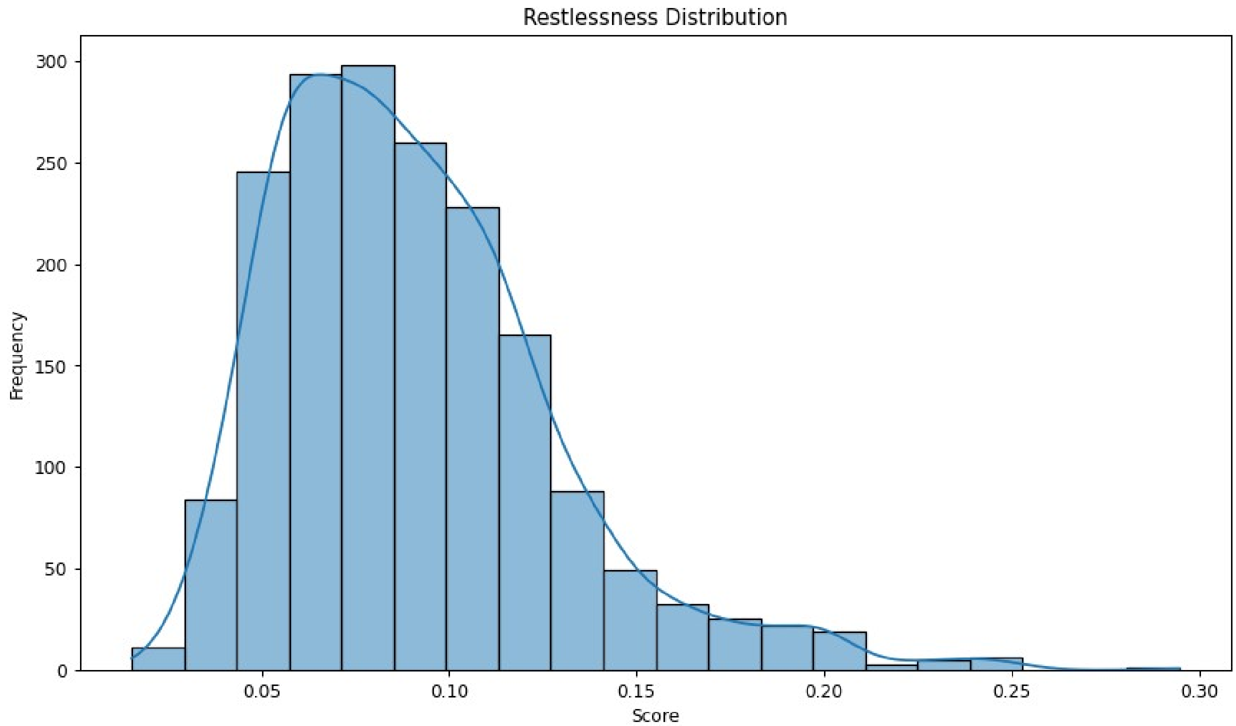
```
<ipython-input-23-b7c9e5de08c2>:3: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it
will default to False. Select only valid columns or specify the value
of numeric_only to silence this warning.
  sns.heatmap(df_eda.corr(), annot=True, cmap='coolwarm', fmt=".2f")
```




```python
# DEEP SLEEP DISTRIBUTION
plt.figure(figsize=(10, 6), dpi=88)
sns.histplot(df_eda['deep_sleep_in_minutes'], kde=True, bins=20)
plt.title('Deep Sleep (in minutes) Distribution')
plt.xlabel('Minutes')
plt.ylabel('Frequency')
plt.savefig(f'{REPO_REPORT_DIR}/deep_sleep_distribution.svg',
format='svg')
plt.show()
```




```python
# RESTLESSNESS DISTRIBUTION
plt.figure(figsize=(10, 6), dpi=88)
sns.histplot(df_eda['restlessness'], kde=True, bins=20)
plt.title('Restlessness Distribution')
plt.xlabel('Score')
plt.ylabel('Frequency')
plt.savefig(f'{REPO_REPORT_DIR}/restlessness_distribution.svg',
format='svg')
plt.show()
```



FEATURE ENGINEERING

```
#
------------------------------------------------------------------------
------ #
# LAG FEATURES
#
------------------------------------------------------------------------
------ #
df = df_eda
for lag in range(1, 8):
    df[f'composition_score_lag{lag}'] =
df['composition_score'].shift(lag)
    df[f'deep_sleep_in_minutes_lag{lag}'] =
df['deep_sleep_in_minutes'].shift(lag)
    df[f'duration_score_lag{lag}'] = df['duration_score'].shift(lag)
    df[f'overall_score_lag{lag}'] = df['overall_score'].shift(lag)
    df[f'restlessness_lag{lag}'] = df['restlessness'].shift(lag)
    df[f'revitalization_score_lag{lag}'] =
df['revitalization_score'].shift(lag)

# DROP ROWS WITH NAN VALUES CREATED BY LAGGING
df.dropna(inplace=True)

#
------------------------------------------------------------------------
------ #
```

```
# ADD NOISE
#
---------------------------------------------------------------------------
------ #
noise_strength = 0.02
features_to_noise = ['deep_sleep_in_minutes', 'resting_heart_rate',
'restlessness'] + \
                    [f'deep_sleep_in_minutes_lag{lag}' for lag in
range(1, 8)] + \
                    [f'restlessness_lag{lag}' for lag in range(1, 8)]
for feature in features_to_noise:
    df[feature] += np.random.normal(0, noise_strength, df.shape[0])

# DEFINE FEATURES AND TARGET
features_columns = ['deep_sleep_in_minutes', 'resting_heart_rate',
'restlessness', 'composition_score', 'duration_score'] + \
                    [f'{col}_lag{lag}' for lag in range(1, 8) for col
in ['deep_sleep_in_minutes', 'revitalization_score', 'restlessness',
'composition_score', 'duration_score']]
features = df[features_columns]
target = df['revitalization_score']

# SCALE THE FEATURES
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_features = scaler.fit_transform(features)
```

LSTM - Deep Learning Neural Network

```
# SET LSTM CONFIGURATION
basic_features = ['deep_sleep_in_minutes', 'resting_heart_rate',
'restlessness']
all_features = basic_features + ['overall_score']
lags = range(1, 8)
sequence_length = 7
pred_horizon = 1
test_size = 0.2
epochs = 150
batch_size = 32

# DEFINE SEQUENCE LENGTH AND PREDICTIVE HORIZON
sequence_length = 7
predictive_horizon = 1

# CREATE SEQUENCES
X, y = [], []
for i in range(len(features) - sequence_length - predictive_horizon +
1):
    X.append(features[i:i + sequence_length])
```

```
    y.append(target[i + sequence_length + predictive_horizon - 1])
X, y = np.array(X), np.array(y)

# SPLIT THE DATA INTO TRAINING AND VALIDATION SETS
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
shuffle=False)

# BUILD MODEL
reg = l1_l2(l1=0.01, l2=0.01)
model = Sequential([
    Bidirectional(
        LSTM(units=128, return_sequences=True,
kernel_regularizer=reg),
        input_shape=X_train.shape[1:]
    ),
    BatchNormalization(), Dropout(0.5),
    LSTM(units=64, return_sequences=False, kernel_regularizer=reg),
    BatchNormalization(), Dropout(0.5),
    Dense(32, activation='relu', kernel_regularizer=reg),
    BatchNormalization(), Dropout(0.5),
    Dense(1, activation='linear')
])
model.compile(optimizer='adam', loss='mean_squared_error',
metrics=['mse', 'mae'])

# TRAIN WITH EARLY STOPPING
early_stopping = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)
history = model.fit(X_train, y_train, epochs=epochs,
batch_size=batch_size,
                    validation_data=(X_val, y_val),
callbacks=[early_stopping])

Epoch 1/150
46/46 [==============================] - 11s 41ms/step - loss:
423.9555 - mse: 367.1572 - mae: 18.7393 - val_loss: 408.1885 -
val_mse: 357.1116 - val_mae: 18.6386
Epoch 2/150
46/46 [==============================] - 2s 34ms/step - loss: 397.2213
- mse: 349.3310 - mae: 18.2924 - val_loss: 403.5255 - val_mse:
358.1507 - val_mae: 18.6660
Epoch 3/150
46/46 [==============================] - 0s 10ms/step - loss: 376.5329
- mse: 332.4427 - mae: 17.8017 - val_loss: 367.5074 - val_mse:
324.4140 - val_mae: 17.7379
Epoch 4/150
46/46 [==============================] - 1s 27ms/step - loss: 355.9953
- mse: 313.6239 - mae: 17.2579 - val_loss: 323.5381 - val_mse:
281.8344 - val_mae: 16.4950
Epoch 5/150
```

```
46/46 [==============================] - 2s 34ms/step - loss: 335.3920
- mse: 294.1961 - mae: 16.6552 - val_loss: 282.4524 - val_mse:
241.7560 - val_mae: 15.2316
Epoch 6/150
46/46 [==============================] - 0s 10ms/step - loss: 306.1168
- mse: 265.9099 - mae: 15.7263 - val_loss: 255.4778 - val_mse:
215.8031 - val_mae: 14.3381
Epoch 7/150
46/46 [==============================] - 0s 10ms/step - loss: 273.9003
- mse: 234.6863 - mae: 14.6674 - val_loss: 280.5317 - val_mse:
241.7457 - val_mae: 15.2283
Epoch 8/150
46/46 [==============================] - 1s 28ms/step - loss: 238.5849
- mse: 200.1450 - mae: 13.3995 - val_loss: 228.7477 - val_mse:
190.6781 - val_mae: 13.4167
Epoch 9/150
46/46 [==============================] - 5s 121ms/step - loss:
206.0975 - mse: 168.3900 - mae: 12.1186 - val_loss: 205.7617 -
val_mse: 168.4336 - val_mae: 12.5856
Epoch 10/150
46/46 [==============================] - 0s 10ms/step - loss: 168.7228
- mse: 131.7069 - mae: 10.5046 - val_loss: 206.8967 - val_mse:
170.2296 - val_mae: 12.6600
Epoch 11/150
46/46 [==============================] - 0s 10ms/step - loss: 144.5073
- mse: 108.1567 - mae: 9.2227 - val_loss: 170.1111 - val_mse: 134.0718
- val_mae: 11.1259
Epoch 12/150
46/46 [==============================] - 0s 10ms/step - loss: 119.0937
- mse: 83.3782 - mae: 7.8936 - val_loss: 243.5741 - val_mse: 208.2208
- val_mae: 14.0799
Epoch 13/150
46/46 [==============================] - 0s 10ms/step - loss: 102.1958
- mse: 67.1552 - mae: 6.8932 - val_loss: 134.8192 - val_mse: 100.1261
- val_mae: 9.4852
Epoch 14/150
46/46 [==============================] - 0s 10ms/step - loss: 88.1847
- mse: 53.8213 - mae: 6.0993 - val_loss: 110.4764 - val_mse: 76.4819 -
val_mae: 8.1531
Epoch 15/150
46/46 [==============================] - 0s 10ms/step - loss: 80.9220
- mse: 47.2089 - mae: 5.6425 - val_loss: 78.0341 - val_mse: 44.5803 -
val_mae: 6.0278
Epoch 16/150
46/46 [==============================] - 0s 10ms/step - loss: 74.3777
- mse: 41.2406 - mae: 5.2318 - val_loss: 45.2198 - val_mse: 12.4679 -
val_mae: 3.0286
Epoch 17/150
46/46 [==============================] - 0s 10ms/step - loss: 69.0501
```

```
- mse: 36.6606 - mae: 4.9190 - val_loss: 110.1043 - val_mse: 78.1089 -
val_mae: 8.2478
Epoch 18/150
46/46 [==============================] - 0s 10ms/step - loss: 67.7623
- mse: 36.1034 - mae: 4.8274 - val_loss: 91.4941 - val_mse: 60.1429 -
val_mae: 7.1290
Epoch 19/150
46/46 [==============================] - 0s 11ms/step - loss: 64.8694
- mse: 33.7818 - mae: 4.6784 - val_loss: 44.9711 - val_mse: 14.1676 -
val_mae: 3.2181
Epoch 20/150
46/46 [==============================] - 0s 10ms/step - loss: 64.5075
- mse: 33.9392 - mae: 4.7374 - val_loss: 44.5177 - val_mse: 14.2167 -
val_mae: 3.2328
Epoch 21/150
46/46 [==============================] - 1s 27ms/step - loss: 62.5089
- mse: 32.4809 - mae: 4.6108 - val_loss: 52.0929 - val_mse: 22.4117 -
val_mae: 4.1304
Epoch 22/150
46/46 [==============================] - 0s 10ms/step - loss: 61.8370
- mse: 32.4452 - mae: 4.6362 - val_loss: 40.8264 - val_mse: 11.7234 -
val_mae: 2.6177
Epoch 23/150
46/46 [==============================] - 0s 10ms/step - loss: 58.7380
- mse: 29.8503 - mae: 4.4101 - val_loss: 55.4780 - val_mse: 26.8597 -
val_mae: 4.5411
Epoch 24/150
46/46 [==============================] - 0s 10ms/step - loss: 59.1557
- mse: 30.8049 - mae: 4.4709 - val_loss: 57.2373 - val_mse: 29.2075 -
val_mae: 4.7599
Epoch 25/150
46/46 [==============================] - 2s 53ms/step - loss: 56.9039
- mse: 29.1186 - mae: 4.3396 - val_loss: 37.7602 - val_mse: 10.1920 -
val_mae: 2.5227
Epoch 26/150
46/46 [==============================] - 0s 10ms/step - loss: 56.1369
- mse: 28.8258 - mae: 4.3808 - val_loss: 52.9972 - val_mse: 25.9944 -
val_mae: 4.4574
Epoch 27/150
46/46 [==============================] - 1s 30ms/step - loss: 55.0028
- mse: 28.2936 - mae: 4.2741 - val_loss: 54.2514 - val_mse: 27.8811 -
val_mae: 4.6321
Epoch 28/150
46/46 [==============================] - 2s 34ms/step - loss: 55.0505
- mse: 28.9168 - mae: 4.3235 - val_loss: 38.6304 - val_mse: 12.7209 -
val_mae: 3.0658
Epoch 29/150
46/46 [==============================] - 0s 10ms/step - loss: 54.0421
- mse: 28.3683 - mae: 4.2609 - val_loss: 42.2284 - val_mse: 16.8583 -
```

```
val_mae: 3.5468
Epoch 30/150
46/46 [==============================] - 0s 10ms/step - loss: 54.0469
- mse: 28.8306 - mae: 4.3437 - val_loss: 35.0650 - val_mse: 9.9983 -
val_mae: 2.5237
Epoch 31/150
46/46 [==============================] - 0s 10ms/step - loss: 52.1979
- mse: 27.3317 - mae: 4.2019 - val_loss: 41.1260 - val_mse: 16.4917 -
val_mae: 3.0737
Epoch 32/150
46/46 [==============================] - 2s 34ms/step - loss: 51.3019
- mse: 26.9024 - mae: 4.1004 - val_loss: 33.8491 - val_mse: 9.6984 -
val_mae: 2.5249
Epoch 33/150
46/46 [==============================] - 0s 10ms/step - loss: 50.7444
- mse: 26.8603 - mae: 4.1514 - val_loss: 34.0354 - val_mse: 10.4204 -
val_mae: 2.7334
Epoch 34/150
46/46 [==============================] - 1s 26ms/step - loss: 49.8089
- mse: 26.4652 - mae: 4.1119 - val_loss: 42.4648 - val_mse: 19.3722 -
val_mae: 3.8093
Epoch 35/150
46/46 [==============================] - 2s 34ms/step - loss: 49.2621
- mse: 26.4271 - mae: 4.1778 - val_loss: 39.6650 - val_mse: 17.0915 -
val_mae: 3.5743
Epoch 36/150
46/46 [==============================] - 0s 10ms/step - loss: 48.5204
- mse: 26.1917 - mae: 4.1685 - val_loss: 32.5600 - val_mse: 10.4709 -
val_mae: 2.7390
Epoch 37/150
46/46 [==============================] - 2s 47ms/step - loss: 48.2121
- mse: 26.2514 - mae: 4.1437 - val_loss: 39.1518 - val_mse: 17.4024 -
val_mae: 3.6108
Epoch 38/150
46/46 [==============================] - 1s 16ms/step - loss: 46.7858
- mse: 25.2493 - mae: 4.0590 - val_loss: 36.7446 - val_mse: 15.4149 -
val_mae: 3.3780
Epoch 39/150
46/46 [==============================] - 1s 16ms/step - loss: 44.7962
- mse: 23.6772 - mae: 3.9470 - val_loss: 33.4765 - val_mse: 12.5936 -
val_mae: 3.0420
Epoch 40/150
46/46 [==============================] - 0s 10ms/step - loss: 45.8863
- mse: 25.1797 - mae: 4.0399 - val_loss: 42.0667 - val_mse: 21.5712 -
val_mae: 4.0412
Epoch 41/150
46/46 [==============================] - 1s 18ms/step - loss: 44.8575
- mse: 24.5874 - mae: 4.0495 - val_loss: 44.9112 - val_mse: 24.8905 -
val_mae: 4.3641
```

```
Epoch 42/150
46/46 [==============================] - 0s 10ms/step - loss: 44.2933
- mse: 24.5016 - mae: 4.0025 - val_loss: 45.3249 - val_mse: 25.7430 -
val_mae: 4.4413
Epoch 43/150
46/46 [==============================] - 2s 52ms/step - loss: 42.7139
- mse: 23.3281 - mae: 3.9054 - val_loss: 36.2762 - val_mse: 17.1153 -
val_mae: 3.5781
Epoch 44/150
46/46 [==============================] - 0s 10ms/step - loss: 42.9373
- mse: 23.9718 - mae: 3.9349 - val_loss: 31.4890 - val_mse: 12.7099 -
val_mae: 3.0641
Epoch 45/150
46/46 [==============================] - 2s 46ms/step - loss: 42.2456
- mse: 23.6522 - mae: 3.8502 - val_loss: 28.2233 - val_mse: 9.7817 -
val_mae: 2.5277
Epoch 46/150
46/46 [==============================] - 0s 10ms/step - loss: 42.9294
- mse: 24.6040 - mae: 3.9798 - val_loss: 33.9256 - val_mse: 15.7085 -
val_mae: 3.4168
Epoch 47/150
46/46 [==============================] - 0s 10ms/step - loss: 40.4400
- mse: 22.3551 - mae: 3.8124 - val_loss: 27.6489 - val_mse: 9.7257 -
val_mae: 2.5365
Epoch 48/150
46/46 [==============================] - 2s 50ms/step - loss: 41.9194
- mse: 24.1493 - mae: 3.9483 - val_loss: 27.9208 - val_mse: 10.2960 -
val_mae: 2.7130
Epoch 49/150
46/46 [==============================] - 0s 10ms/step - loss: 38.9638
- mse: 21.5052 - mae: 3.7268 - val_loss: 27.0412 - val_mse: 9.7844 -
val_mae: 2.5194
Epoch 50/150
46/46 [==============================] - 1s 26ms/step - loss: 39.9968
- mse: 22.9070 - mae: 3.8160 - val_loss: 26.7083 - val_mse: 9.7825 -
val_mae: 2.5481
Epoch 51/150
46/46 [==============================] - 2s 35ms/step - loss: 41.0913
- mse: 24.3185 - mae: 4.0020 - val_loss: 27.8056 - val_mse: 11.2014 -
val_mae: 2.8483
Epoch 52/150
46/46 [==============================] - 0s 10ms/step - loss: 38.7617
- mse: 22.3318 - mae: 3.7897 - val_loss: 25.9337 - val_mse: 9.7174 -
val_mae: 2.5445
Epoch 53/150
46/46 [==============================] - 2s 53ms/step - loss: 38.6071
- mse: 22.5560 - mae: 3.8446 - val_loss: 25.6351 - val_mse: 9.7434 -
val_mae: 2.5568
Epoch 54/150
```

```
46/46 [==============================] - 0s 10ms/step - loss: 38.0757
- mse: 22.2675 - mae: 3.8000 - val_loss: 26.9446 - val_mse: 11.2802 -
val_mae: 2.8523
Epoch 55/150
46/46 [==============================] - 1s 29ms/step - loss: 38.4593
- mse: 22.9810 - mae: 3.8700 - val_loss: 25.0581 - val_mse: 9.7871 -
val_mae: 2.5702
Epoch 56/150
46/46 [==============================] - 2s 34ms/step - loss: 37.0483
- mse: 21.8937 - mae: 3.8120 - val_loss: 27.5319 - val_mse: 12.5267 -
val_mae: 3.0422
Epoch 57/150
46/46 [==============================] - 0s 10ms/step - loss: 37.7709
- mse: 22.9257 - mae: 3.8621 - val_loss: 26.4317 - val_mse: 11.7231 -
val_mae: 2.9260
Epoch 58/150
46/46 [==============================] - 2s 54ms/step - loss: 36.1591
- mse: 21.4996 - mae: 3.6981 - val_loss: 27.0163 - val_mse: 12.4228 -
val_mae: 3.0292
Epoch 59/150
46/46 [==============================] - 0s 10ms/step - loss: 38.2668
- mse: 23.7440 - mae: 3.9295 - val_loss: 28.3044 - val_mse: 13.9333 -
val_mae: 3.2067
Epoch 60/150
46/46 [==============================] - 1s 28ms/step - loss: 36.6945
- mse: 22.4797 - mae: 3.8135 - val_loss: 29.8130 - val_mse: 15.7700 -
val_mae: 3.4224
Epoch 61/150
46/46 [==============================] - 0s 10ms/step - loss: 36.2375
- mse: 22.3456 - mae: 3.8093 - val_loss: 29.4234 - val_mse: 15.6486 -
val_mae: 3.4088
Epoch 62/150
46/46 [==============================] - 0s 10ms/step - loss: 36.0629
- mse: 22.3874 - mae: 3.8149 - val_loss: 25.8966 - val_mse: 12.3716 -
val_mae: 3.0201
Epoch 63/150
46/46 [==============================] - 2s 49ms/step - loss: 35.9960
- mse: 22.6341 - mae: 3.8976 - val_loss: 24.7223 - val_mse: 11.5100 -
val_mae: 2.8957
Epoch 64/150
46/46 [==============================] - 0s 10ms/step - loss: 35.0330
- mse: 21.9211 - mae: 3.7846 - val_loss: 26.1161 - val_mse: 13.1054 -
val_mae: 3.1150
Epoch 65/150
46/46 [==============================] - 0s 10ms/step - loss: 34.1893
- mse: 21.3023 - mae: 3.6973 - val_loss: 26.8711 - val_mse: 14.1328 -
val_mae: 3.2334
Epoch 66/150
46/46 [==============================] - 2s 52ms/step - loss: 35.0217
```

```
- mse: 22.4445 - mae: 3.8024 - val_loss: 25.3270 - val_mse: 12.9080 -
val_mae: 3.0906
Epoch 67/150
46/46 [==============================] - 0s 10ms/step - loss: 34.0533
- mse: 21.7308 - mae: 3.7927 - val_loss: 26.1994 - val_mse: 13.9939 -
val_mae: 3.2130
Epoch 68/150
46/46 [==============================] - 1s 29ms/step - loss: 33.6074
- mse: 21.5299 - mae: 3.7041 - val_loss: 22.5260 - val_mse: 10.5758 -
val_mae: 2.7593
Epoch 69/150
46/46 [==============================] - 2s 34ms/step - loss: 33.2238
- mse: 21.3975 - mae: 3.7250 - val_loss: 22.1315 - val_mse: 10.4417 -
val_mae: 2.7383
Epoch 70/150
46/46 [==============================] - 0s 10ms/step - loss: 33.4093
- mse: 21.7732 - mae: 3.7864 - val_loss: 23.2201 - val_mse: 11.6782 -
val_mae: 2.9205
Epoch 71/150
46/46 [==============================] - 1s 15ms/step - loss: 32.5696
- mse: 21.1100 - mae: 3.7051 - val_loss: 22.0797 - val_mse: 10.6349 -
val_mae: 2.7689
Epoch 72/150
46/46 [==============================] - 1s 17ms/step - loss: 33.6752
- mse: 22.2621 - mae: 3.8540 - val_loss: 22.9077 - val_mse: 11.5674 -
val_mae: 2.9039
Epoch 73/150
46/46 [==============================] - 0s 10ms/step - loss: 33.5180
- mse: 22.3002 - mae: 3.7960 - val_loss: 21.8027 - val_mse: 10.7137 -
val_mae: 2.7771
Epoch 74/150
46/46 [==============================] - 1s 27ms/step - loss: 33.1324
- mse: 22.1568 - mae: 3.8292 - val_loss: 22.0857 - val_mse: 11.2373 -
val_mae: 2.8487
Epoch 75/150
46/46 [==============================] - 2s 35ms/step - loss: 32.4786
- mse: 21.6992 - mae: 3.7714 - val_loss: 22.0380 - val_mse: 11.3636 -
val_mae: 2.8709
Epoch 76/150
46/46 [==============================] - 0s 10ms/step - loss: 31.8243
- mse: 21.2375 - mae: 3.7218 - val_loss: 22.3866 - val_mse: 11.8929 -
val_mae: 2.9548
Epoch 77/150
46/46 [==============================] - 1s 33ms/step - loss: 32.0016
- mse: 21.5414 - mae: 3.7256 - val_loss: 23.6653 - val_mse: 13.2994 -
val_mae: 3.1361
Epoch 78/150
46/46 [==============================] - 0s 10ms/step - loss: 32.6096
- mse: 22.2983 - mae: 3.7949 - val_loss: 21.7465 - val_mse: 11.5012 -
```

```
val_mae: 2.8933
Epoch 79/150
46/46 [==============================] - 1s 27ms/step - loss: 30.9164
- mse: 20.7567 - mae: 3.6594 - val_loss: 20.0213 - val_mse: 9.9447 -
val_mae: 2.6338
Epoch 80/150
46/46 [==============================] - 2s 34ms/step - loss: 32.0897
- mse: 22.1283 - mae: 3.7651 - val_loss: 21.9576 - val_mse: 12.1217 -
val_mae: 2.9881
Epoch 81/150
46/46 [==============================] - 0s 10ms/step - loss: 30.6874
- mse: 20.9364 - mae: 3.6620 - val_loss: 21.4149 - val_mse: 11.7815 -
val_mae: 2.9386
Epoch 82/150
46/46 [==============================] - 2s 52ms/step - loss: 30.6843
- mse: 21.1611 - mae: 3.7370 - val_loss: 21.7351 - val_mse: 12.3338 -
val_mae: 3.0173
Epoch 83/150
46/46 [==============================] - 0s 10ms/step - loss: 31.4509
- mse: 22.1691 - mae: 3.7906 - val_loss: 21.0683 - val_mse: 11.8952 -
val_mae: 2.9600
Epoch 84/150
46/46 [==============================] - 1s 28ms/step - loss: 29.6772
- mse: 20.6033 - mae: 3.6202 - val_loss: 20.3074 - val_mse: 11.3628 -
val_mae: 2.8708
Epoch 85/150
46/46 [==============================] - 1s 16ms/step - loss: 30.3723
- mse: 21.4773 - mae: 3.7343 - val_loss: 20.0543 - val_mse: 11.2087 -
val_mae: 2.8482
Epoch 86/150
46/46 [==============================] - 0s 10ms/step - loss: 29.3043
- mse: 20.5203 - mae: 3.6435 - val_loss: 18.4413 - val_mse: 9.7243 -
val_mae: 2.5514
Epoch 87/150
46/46 [==============================] - 3s 66ms/step - loss: 30.5965
- mse: 21.9423 - mae: 3.7841 - val_loss: 18.5551 - val_mse: 10.0052 -
val_mae: 2.6475
Epoch 88/150
46/46 [==============================] - 0s 10ms/step - loss: 29.5193
- mse: 21.0304 - mae: 3.7191 - val_loss: 18.8591 - val_mse: 10.4407 -
val_mae: 2.7368
Epoch 89/150
46/46 [==============================] - 1s 15ms/step - loss: 29.1853
- mse: 20.8088 - mae: 3.7044 - val_loss: 18.5357 - val_mse: 10.2751 -
val_mae: 2.7079
Epoch 90/150
46/46 [==============================] - 0s 10ms/step - loss: 29.3594
- mse: 21.1606 - mae: 3.7650 - val_loss: 18.8566 - val_mse: 10.7322 -
val_mae: 2.7810
```

```
Epoch 91/150
46/46 [==============================] - 1s 18ms/step - loss: 28.8250
- mse: 20.7724 - mae: 3.6798 - val_loss: 18.0437 - val_mse: 10.0581 -
val_mae: 2.6545
Epoch 92/150
46/46 [==============================] - 0s 10ms/step - loss: 29.1174
- mse: 21.2120 - mae: 3.7040 - val_loss: 17.8925 - val_mse: 10.0883 -
val_mae: 2.6705
Epoch 93/150
46/46 [==============================] - 2s 52ms/step - loss: 27.9915
- mse: 20.2873 - mae: 3.6341 - val_loss: 18.4333 - val_mse: 10.8453 -
val_mae: 2.7974
Epoch 94/150
46/46 [==============================] - 0s 10ms/step - loss: 27.9030
- mse: 20.4203 - mae: 3.6366 - val_loss: 17.7392 - val_mse: 10.3581 -
val_mae: 2.7256
Epoch 95/150
46/46 [==============================] - 1s 15ms/step - loss: 27.5687
- mse: 20.2449 - mae: 3.6084 - val_loss: 18.1839 - val_mse: 10.9152 -
val_mae: 2.8084
Epoch 96/150
46/46 [==============================] - 0s 10ms/step - loss: 28.7622
- mse: 21.5107 - mae: 3.7283 - val_loss: 17.2786 - val_mse: 10.0493 -
val_mae: 2.6598
Epoch 97/150
46/46 [==============================] - 0s 10ms/step - loss: 28.5491
- mse: 21.4109 - mae: 3.7355 - val_loss: 17.4446 - val_mse: 10.3837 -
val_mae: 2.7289
Epoch 98/150
46/46 [==============================] - 0s 10ms/step - loss: 27.1993
- mse: 20.2306 - mae: 3.6127 - val_loss: 17.5832 - val_mse: 10.7022 -
val_mae: 2.7787
Epoch 99/150
46/46 [==============================] - 2s 53ms/step - loss: 28.4847
- mse: 21.6407 - mae: 3.7701 - val_loss: 17.9625 - val_mse: 11.1527 -
val_mae: 2.8369
Epoch 100/150
46/46 [==============================] - 0s 10ms/step - loss: 28.6658
- mse: 21.8674 - mae: 3.7438 - val_loss: 17.6124 - val_mse: 10.8345 -
val_mae: 2.7949
Epoch 101/150
46/46 [==============================] - 1s 15ms/step - loss: 27.6158
- mse: 20.9130 - mae: 3.6905 - val_loss: 18.7968 - val_mse: 12.2257 -
val_mae: 3.0005
Epoch 102/150
46/46 [==============================] - 0s 10ms/step - loss: 27.7391
- mse: 21.2788 - mae: 3.7578 - val_loss: 17.1576 - val_mse: 10.8155 -
val_mae: 2.7953
Epoch 103/150
```

```
46/46 [==============================] - 0s 10ms/step - loss: 26.9554
- mse: 20.6980 - mae: 3.6737 - val_loss: 16.2837 - val_mse: 10.1244 -
val_mae: 2.6786
Epoch 104/150
46/46 [==============================] - 0s 10ms/step - loss: 27.9021
- mse: 21.8191 - mae: 3.7551 - val_loss: 15.8349 - val_mse: 9.8356 -
val_mae: 2.5866
Epoch 105/150
46/46 [==============================] - 2s 34ms/step - loss: 26.1165
- mse: 20.1771 - mae: 3.5620 - val_loss: 15.7797 - val_mse: 9.8526 -
val_mae: 2.6033
Epoch 106/150
46/46 [==============================] - 0s 10ms/step - loss: 26.2609
- mse: 20.2897 - mae: 3.6190 - val_loss: 15.7575 - val_mse: 9.8015 -
val_mae: 2.5796
Epoch 107/150
46/46 [==============================] - 2s 45ms/step - loss: 26.1880
- mse: 20.3027 - mae: 3.6093 - val_loss: 18.0770 - val_mse: 12.2521 -
val_mae: 3.0040
Epoch 108/150
46/46 [==============================] - 1s 16ms/step - loss: 25.9330
- mse: 20.1914 - mae: 3.6059 - val_loss: 17.4344 - val_mse: 11.7776 -
val_mae: 2.9373
Epoch 109/150
46/46 [==============================] - 0s 10ms/step - loss: 26.7410
- mse: 21.1563 - mae: 3.7144 - val_loss: 16.2781 - val_mse: 10.7573 -
val_mae: 2.7857
Epoch 110/150
46/46 [==============================] - 2s 51ms/step - loss: 25.8550
- mse: 20.3621 - mae: 3.6256 - val_loss: 15.8384 - val_mse: 10.3829 -
val_mae: 2.7292
Epoch 111/150
46/46 [==============================] - 0s 10ms/step - loss: 26.1247
- mse: 20.6817 - mae: 3.6989 - val_loss: 15.3481 - val_mse: 9.9599 -
val_mae: 2.6360
Epoch 112/150
46/46 [==============================] - 1s 27ms/step - loss: 26.8535
- mse: 21.5139 - mae: 3.7399 - val_loss: 15.2349 - val_mse: 9.9366 -
val_mae: 2.6298
Epoch 113/150
46/46 [==============================] - 2s 34ms/step - loss: 25.5918
- mse: 20.3362 - mae: 3.6815 - val_loss: 15.2338 - val_mse: 10.0462 -
val_mae: 2.6534
Epoch 114/150
46/46 [==============================] - 0s 10ms/step - loss: 25.7729
- mse: 20.6715 - mae: 3.6507 - val_loss: 14.7328 - val_mse: 9.7494 -
val_mae: 2.5498
Epoch 115/150
46/46 [==============================] - 2s 53ms/step - loss: 25.1330
```

```
- mse: 20.1975 - mae: 3.6211 - val_loss: 14.6629 - val_mse: 9.7638 -
val_mae: 2.5590
Epoch 116/150
46/46 [==============================] - 0s 10ms/step - loss: 25.0303
- mse: 20.1296 - mae: 3.5905 - val_loss: 14.6155 - val_mse: 9.7292 -
val_mae: 2.5418
Epoch 117/150
46/46 [==============================] - 1s 26ms/step - loss: 25.6914
- mse: 20.8725 - mae: 3.7048 - val_loss: 14.9450 - val_mse: 10.1546 -
val_mae: 2.6853
Epoch 118/150
46/46 [==============================] - 1s 16ms/step - loss: 25.0104
- mse: 20.2493 - mae: 3.6220 - val_loss: 14.5967 - val_mse: 9.8919 -
val_mae: 2.6154
Epoch 119/150
46/46 [==============================] - 0s 10ms/step - loss: 24.4526
- mse: 19.8330 - mae: 3.5738 - val_loss: 14.2219 - val_mse: 9.7134 -
val_mae: 2.5441
Epoch 120/150
46/46 [==============================] - 2s 52ms/step - loss: 24.9080
- mse: 20.4467 - mae: 3.5914 - val_loss: 14.4401 - val_mse: 10.0030 -
val_mae: 2.6466
Epoch 121/150
46/46 [==============================] - 0s 10ms/step - loss: 25.7281
- mse: 21.3252 - mae: 3.7517 - val_loss: 14.2050 - val_mse: 9.8879 -
val_mae: 2.6202
Epoch 122/150
46/46 [==============================] - 2s 46ms/step - loss: 24.0251
- mse: 19.8140 - mae: 3.5830 - val_loss: 13.9622 - val_mse: 9.8542 -
val_mae: 2.6032
Epoch 123/150
46/46 [==============================] - 0s 10ms/step - loss: 25.3654
- mse: 21.3006 - mae: 3.7384 - val_loss: 13.8263 - val_mse: 9.7335 -
val_mae: 2.5499
Epoch 124/150
46/46 [==============================] - 0s 10ms/step - loss: 24.9788
- mse: 20.8481 - mae: 3.6526 - val_loss: 14.4195 - val_mse: 10.2555 -
val_mae: 2.7035
Epoch 125/150
46/46 [==============================] - 2s 50ms/step - loss: 25.0734
- mse: 20.9789 - mae: 3.7239 - val_loss: 15.3511 - val_mse: 11.3139 -
val_mae: 2.8633
Epoch 126/150
46/46 [==============================] - 0s 10ms/step - loss: 26.0986
- mse: 22.0964 - mae: 3.8100 - val_loss: 14.6227 - val_mse: 10.6430 -
val_mae: 2.7674
Epoch 127/150
46/46 [==============================] - 1s 27ms/step - loss: 24.6393
- mse: 20.7191 - mae: 3.7014 - val_loss: 13.9528 - val_mse: 10.0974 -
val_mae: 2.6719
```

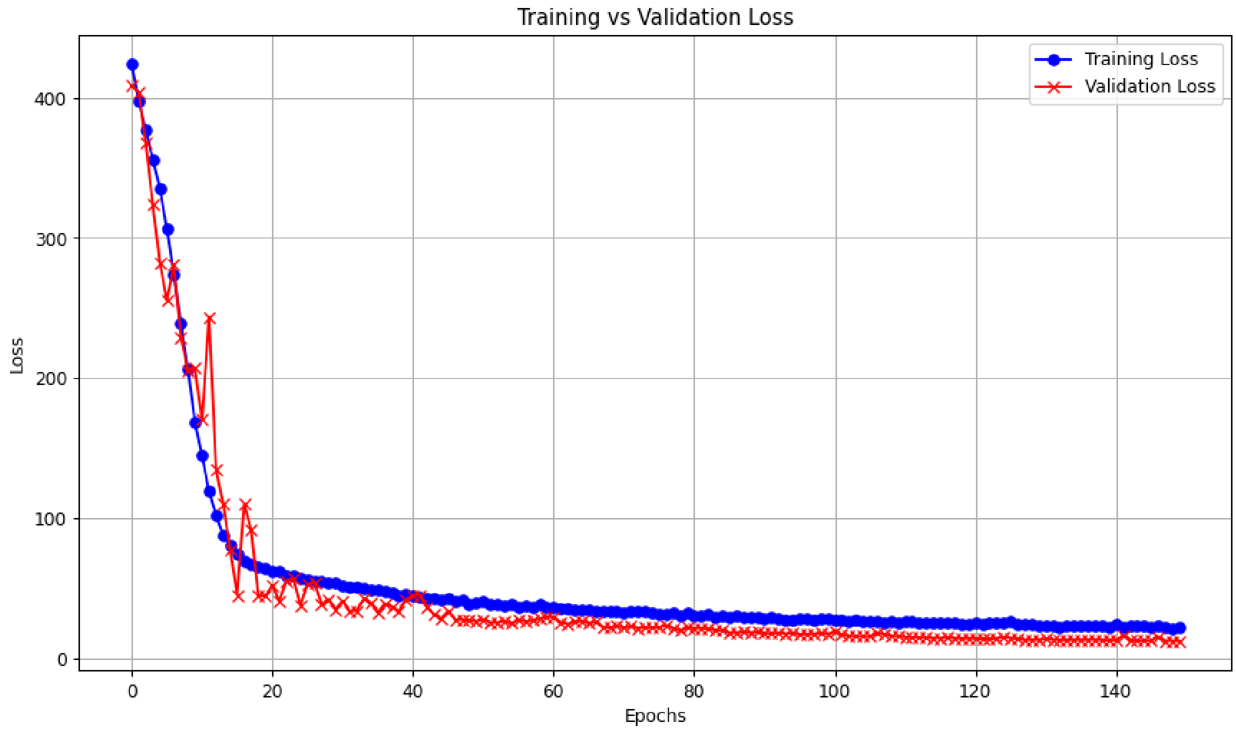
```
Epoch 128/150
46/46 [==============================] - 2s 54ms/step - loss: 23.9292
- mse: 20.1598 - mae: 3.6610 - val_loss: 13.4133 - val_mse: 9.7476 -
val_mae: 2.5581
Epoch 129/150
46/46 [==============================] - 0s 10ms/step - loss: 23.9932
- mse: 20.3948 - mae: 3.6631 - val_loss: 13.3651 - val_mse: 9.7910 -
val_mae: 2.5744
Epoch 130/150
46/46 [==============================] - 1s 16ms/step - loss: 23.3894
- mse: 19.8317 - mae: 3.5365 - val_loss: 13.4822 - val_mse: 9.9478 -
val_mae: 2.6314
Epoch 131/150
46/46 [==============================] - 0s 10ms/step - loss: 23.3085
- mse: 19.8362 - mae: 3.5923 - val_loss: 13.8389 - val_mse: 10.4781 -
val_mae: 2.7400
Epoch 132/150
46/46 [==============================] - 2s 45ms/step - loss: 23.2964
- mse: 19.9163 - mae: 3.5846 - val_loss: 13.6271 - val_mse: 10.2345 -
val_mae: 2.6995
Epoch 133/150
46/46 [==============================] - 2s 34ms/step - loss: 22.7201
- mse: 19.3571 - mae: 3.5051 - val_loss: 13.2866 - val_mse: 9.9350 -
val_mae: 2.5126
Epoch 134/150
46/46 [==============================] - 0s 10ms/step - loss: 23.3620
- mse: 19.9648 - mae: 3.6043 - val_loss: 13.1737 - val_mse: 9.7414 -
val_mae: 2.5369
Epoch 135/150
46/46 [==============================] - 1s 16ms/step - loss: 23.5145
- mse: 20.0802 - mae: 3.6202 - val_loss: 13.4005 - val_mse: 9.9280 -
val_mae: 2.6290
Epoch 136/150
46/46 [==============================] - 1s 17ms/step - loss: 23.5407
- mse: 20.0680 - mae: 3.6114 - val_loss: 13.3650 - val_mse: 9.9279 -
val_mae: 2.6270
Epoch 137/150
46/46 [==============================] - 0s 10ms/step - loss: 23.6062
- mse: 20.1329 - mae: 3.6231 - val_loss: 13.3931 - val_mse: 9.8962 -
val_mae: 2.6179
Epoch 138/150
46/46 [==============================] - 1s 28ms/step - loss: 23.3195
- mse: 19.8405 - mae: 3.6176 - val_loss: 13.5461 - val_mse: 10.1707 -
val_mae: 2.6887
Epoch 139/150
46/46 [==============================] - 1s 16ms/step - loss: 23.5010
- mse: 20.1566 - mae: 3.5958 - val_loss: 13.2004 - val_mse: 9.8764 -
val_mae: 2.5195
Epoch 140/150
```

```
46/46 [==============================] - 0s 10ms/step - loss: 22.7324
- mse: 19.4871 - mae: 3.6039 - val_loss: 12.9319 - val_mse: 9.7209 -
val_mae: 2.5371
Epoch 141/150
46/46 [==============================] - 2s 51ms/step - loss: 24.0249
- mse: 20.8336 - mae: 3.6822 - val_loss: 13.2140 - val_mse: 10.0581 -
val_mae: 2.6632
Epoch 142/150
46/46 [==============================] - 0s 10ms/step - loss: 22.5384
- mse: 19.4545 - mae: 3.5930 - val_loss: 16.3357 - val_mse: 13.3342 -
val_mae: 3.1375
Epoch 143/150
46/46 [==============================] - 1s 26ms/step - loss: 22.9381
- mse: 19.9451 - mae: 3.6134 - val_loss: 13.1431 - val_mse: 10.1304 -
val_mae: 2.6826
Epoch 144/150
46/46 [==============================] - 2s 34ms/step - loss: 23.0466
- mse: 20.0644 - mae: 3.6025 - val_loss: 12.7320 - val_mse: 9.7691 -
val_mae: 2.5626
Epoch 145/150
46/46 [==============================] - 0s 10ms/step - loss: 23.0846
- mse: 20.0957 - mae: 3.6300 - val_loss: 12.7812 - val_mse: 9.7509 -
val_mae: 2.5287
Epoch 146/150
46/46 [==============================] - 2s 47ms/step - loss: 22.8079
- mse: 19.8043 - mae: 3.5808 - val_loss: 12.7111 - val_mse: 9.7375 -
val_mae: 2.5541
Epoch 147/150
46/46 [==============================] - 1s 15ms/step - loss: 22.9520
- mse: 19.9919 - mae: 3.5918 - val_loss: 15.6866 - val_mse: 12.7280 -
val_mae: 3.0615
Epoch 148/150
46/46 [==============================] - 0s 10ms/step - loss: 22.1547
- mse: 19.1939 - mae: 3.5444 - val_loss: 12.6254 - val_mse: 9.7278 -
val_mae: 2.5397
Epoch 149/150
46/46 [==============================] - 0s 10ms/step - loss: 21.6572
- mse: 18.8255 - mae: 3.5014 - val_loss: 12.6197 - val_mse: 9.8439 -
val_mae: 2.5240
Epoch 150/150
46/46 [==============================] - 1s 18ms/step - loss: 22.3830
- mse: 19.6216 - mae: 3.5329 - val_loss: 12.5203 - val_mse: 9.7612 -
val_mae: 2.5270

# EVALUATE
val_loss, val_mse, val_mae = model.evaluate(X_val, y_val)
print(f"Validation Loss: {val_loss}")
print(f"Validation Mean Squared Error: {val_mse}")
print(f"Validation Mean Absolute Error: {val_mae}")
```

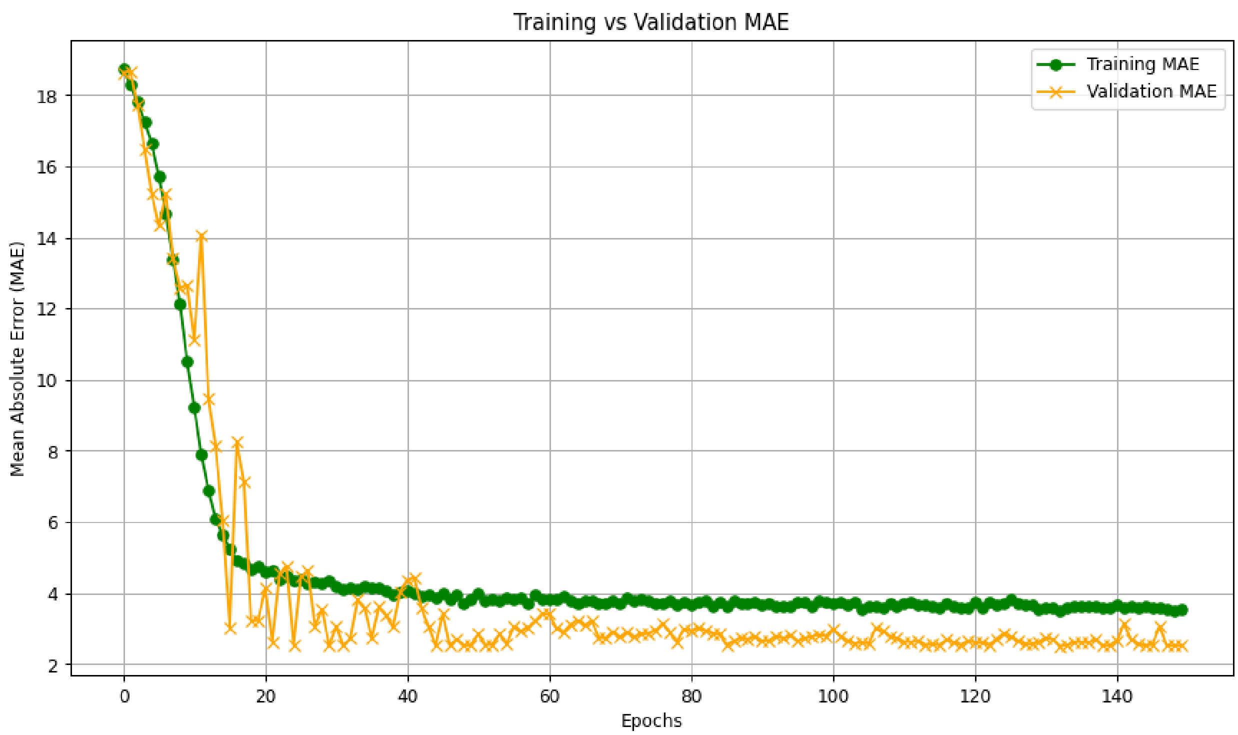
```
12/12 [==============================] - 0s 4ms/step - loss: 12.5203 -
mse: 9.7612 - mae: 2.5270
Validation Loss: 12.520334243774414
Validation Mean Squared Error: 9.761198997497559
Validation Mean Absolute Error: 2.526951789855957
```

Visualizing Validation and MAE

```
# PLOT TRAINING & VALIDATION LOSS
plt.figure(figsize=(10, 6), dpi=88)
plt.plot(history.history['loss'], label='Training Loss', color='blue',
linestyle='-', marker='o')
plt.plot(history.history['val_loss'], label='Validation Loss',
color='red', linestyle='-', marker='x')
plt.title('Training vs Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(loc='upper right')
plt.grid(True)
plt.savefig(f'{REPO_REPORT_DIR}/model_loss_plot.svg')
plt.show()

# EXPORT LOSS HISTORY TO CSV
loss_history_df = pd.DataFrame({
    'Epoch': range(1, len(history.history['loss']) + 1),
    'Training Loss': history.history['loss'],
    'Validation Loss': history.history['val_loss']
})
loss_history_df.to_csv(f'{REPO_REPORT_DIR}/loss_history.csv',
index=False)
```




```
# PLOT TRAINING & VALIDATION MAE
plt.figure(figsize=(10, 6), dpi=88)
plt.plot(history.history['mae'], label='Training MAE', color='green',
linestyle='-', marker='o')
plt.plot(history.history['val_mae'], label='Validation MAE',
color='orange', linestyle='-', marker='x')
plt.title('Training vs Validation MAE')
plt.xlabel('Epochs')
plt.ylabel('Mean Absolute Error (MAE)')
plt.legend(loc='upper right')
plt.grid(True)
plt.savefig(f'{REPO_REPORT_DIR}/model_mae_plot.svg')
plt.show()

# EXPORT MAE HISTORY TO CSV
mae_history_df = pd.DataFrame({
    'Epoch': range(1, len(history.history['mae']) + 1),
    'Training MAE': history.history['mae'],
    'Validation MAE': history.history['val_mae']
})
mae_history_df.to_csv(f'{REPO_REPORT_DIR}/mae_history.csv',
index=False)
```



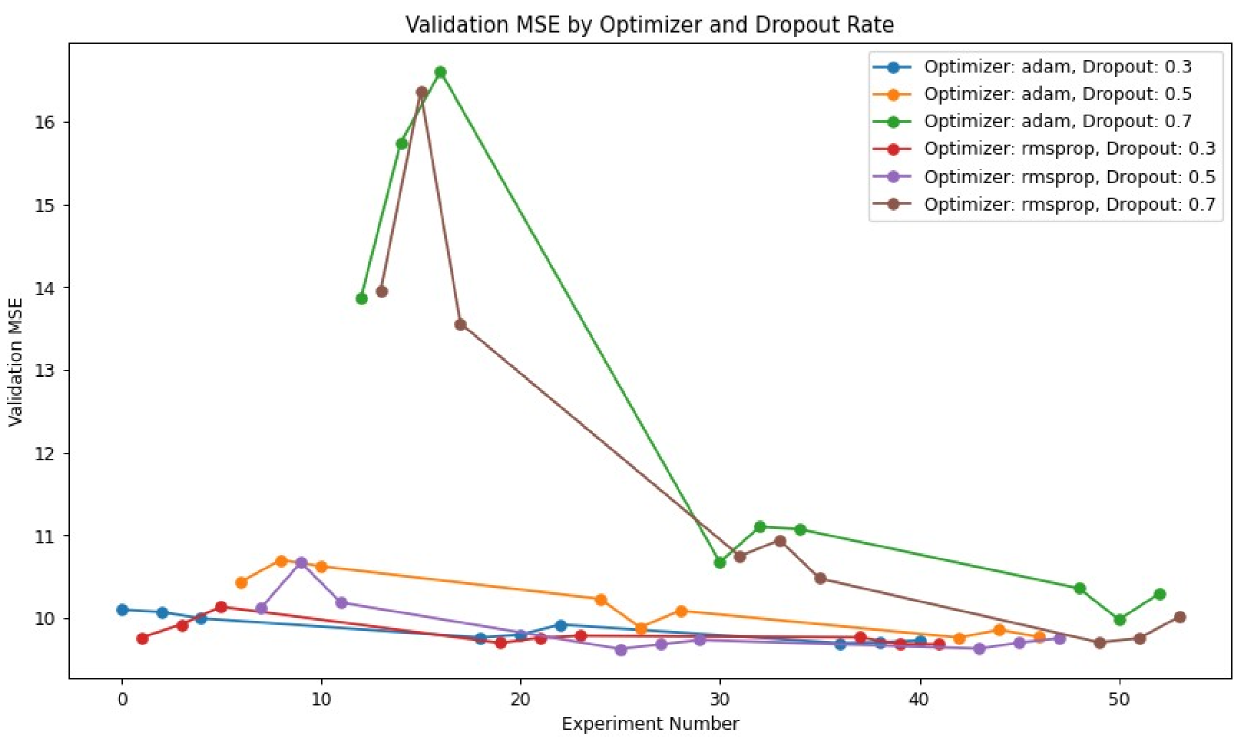
Hyperparameter Tuning

```
# DEFINE HYPERPARAMETER GRID
param_grid = {
    'units_layer1': [32, 64, 128],
    'dropout_rate': [0.3, 0.5, 0.7],
    'units_layer2': [16, 32, 64],
    'optimizer': ['adam', 'rmsprop']
}

best_val_mse = float('inf')
best_model = None
best_hyperparameters = {}
model_stats = []

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

for units_layer1, dropout_rate, units_layer2, optimizer in
product(*param_grid.values()):
    # BUILD MODEL
    model = Sequential([
        LSTM(units_layer1, input_shape=(7, 40)),
        Dropout(dropout_rate),
        Dense(units_layer2, activation='relu'),
        Dense(1, activation='linear')
```

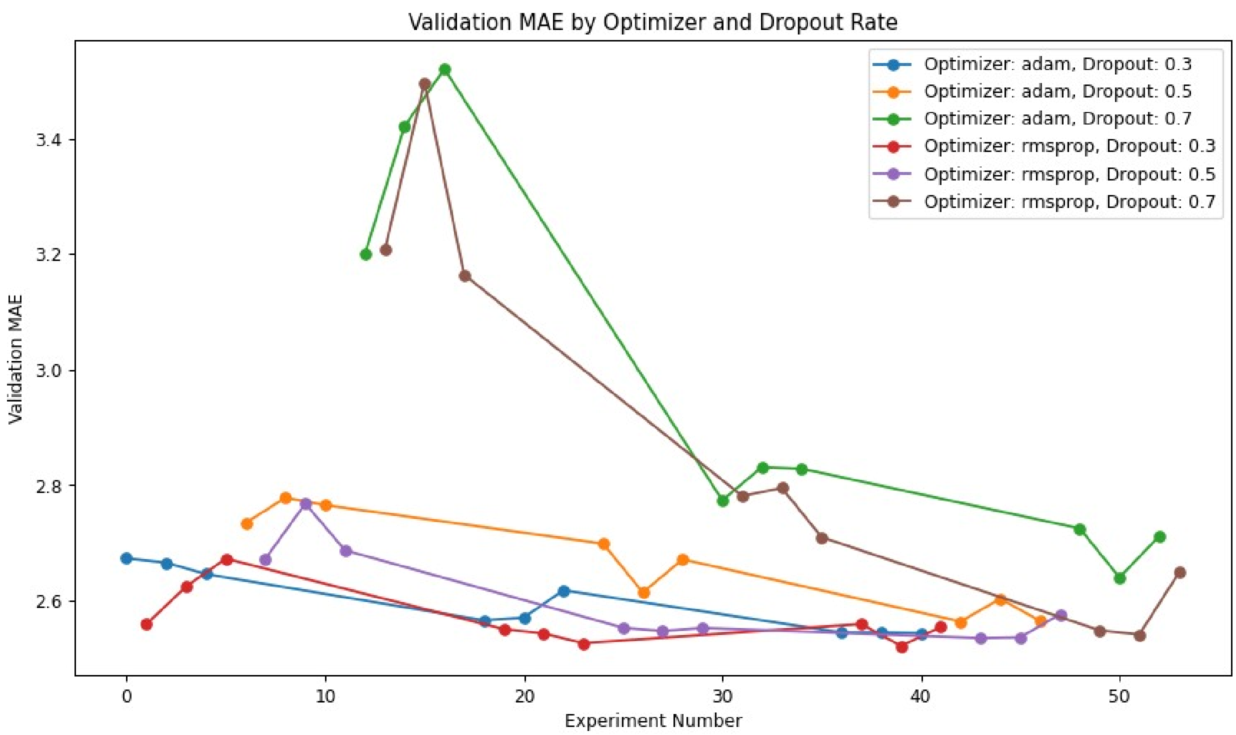
```
    ])
    model.compile(loss='mean_squared_error', optimizer=optimizer,
metrics=['mse', 'mae'])

    # TRAIN MODEL with early stopping
    model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=0,
validation_data=(X_val, y_val), callbacks=[early_stopping])

    # EVALUATE MODEL
    y_pred = model.predict(X_val)
    val_mse = mean_squared_error(y_val, y_pred)
    val_mae = mean_absolute_error(y_val, y_pred)

    # SAVE MODEL STATISTICS
    model_stats.append({
        'units_layer1': units_layer1,
        'dropout_rate': dropout_rate,
        'units_layer2': units_layer2,
        'optimizer': optimizer,
        'val_mse': val_mse,
        'val_mae': val_mae
    })

    # UPDATE BEST MODEL IF CURRENT MODEL IS BETTER
    if val_mse < best_val_mse:
        best_val_mse = val_mse
        best_model = model
        best_hyperparameters = {
            'units_layer1': units_layer1,
            'dropout_rate': dropout_rate,
            'units_layer2': units_layer2,
            'optimizer': optimizer
        }

# OUTPUT BEST HYPERPARAMETERS
print("Best Hyperparameters:", best_hyperparameters)

# FINAL EVALUATION ON VALIDATION SET
y_pred = best_model.predict(X_val)
val_mse = mean_squared_error(y_val, y_pred)
val_mae = mean_absolute_error(y_val, y_pred)
print(f"Validation Mean Squared Error: {val_mse}")
print(f"Validation Mean Absolute Error: {val_mae}")

# SAVE THE BEST MODEL
model_save_path = f"{REPO_REPORT_DIR}/best_model"
if not os.path.exists(model_save_path):
    os.makedirs(model_save_path)
best_model.save(model_save_path)
print(f"Best model saved to: {model_save_path}")
```

```
12/12 [==============================] - 1s 30ms/step
12/12 [==============================] - 1s 25ms/step
12/12 [==============================] - 1s 65ms/step
12/12 [==============================] - 1s 59ms/step
12/12 [==============================] - 1s 39ms/step
12/12 [==============================] - 1s 19ms/step
12/12 [==============================] - 1s 21ms/step
12/12 [==============================] - 0s 11ms/step
12/12 [==============================] - 0s 11ms/step
12/12 [==============================] - 1s 67ms/step
12/12 [==============================] - 0s 2ms/step
12/12 [==============================] - 1s 70ms/step
12/12 [==============================] - 0s 2ms/step
12/12 [==============================] - 0s 11ms/step
12/12 [==============================] - 1s 16ms/step
12/12 [==============================] - 1s 79ms/step
12/12 [==============================] - 1s 44ms/step
12/12 [==============================] - 1s 2ms/step
12/12 [==============================] - 1s 20ms/step
12/12 [==============================] - 0s 2ms/step
12/12 [==============================] - 1s 66ms/step
12/12 [==============================] - 0s 3ms/step
12/12 [==============================] - 1s 45ms/step
12/12 [==============================] - 1s 43ms/step
12/12 [==============================] - 1s 73ms/step
12/12 [==============================] - 1s 28ms/step
12/12 [==============================] - 1s 32ms/step
12/12 [==============================] - 1s 65ms/step
12/12 [==============================] - 1s 33ms/step
12/12 [==============================] - 0s 11ms/step
12/12 [==============================] - 0s 12ms/step
12/12 [==============================] - 0s 2ms/step
12/12 [==============================] - 1s 75ms/step
12/12 [==============================] - 1s 18ms/step
12/12 [==============================] - 0s 6ms/step
12/12 [==============================] - 0s 2ms/step
12/12 [==============================] - 1s 54ms/step
12/12 [==============================] - 0s 14ms/step
12/12 [==============================] - 1s 33ms/step
12/12 [==============================] - 0s 10ms/step
12/12 [==============================] - 0s 12ms/step
12/12 [==============================] - 1s 74ms/step
12/12 [==============================] - 1s 70ms/step
12/12 [==============================] - 1s 77ms/step
12/12 [==============================] - 1s 64ms/step
12/12 [==============================] - 1s 79ms/step
12/12 [==============================] - 1s 37ms/step
12/12 [==============================] - 1s 24ms/step
12/12 [==============================] - 0s 3ms/step
12/12 [==============================] - 1s 60ms/step
```

```
12/12 [==============================] - 1s 37ms/step
12/12 [==============================] - 0s 12ms/step
12/12 [==============================] - 0s 2ms/step
12/12 [==============================] - 1s 40ms/step
Best Hyperparameters: {'units_layer1': 64, 'dropout_rate': 0.5,
'units_layer2': 16, 'optimizer': 'rmsprop'}
12/12 [==============================] - 0s 29ms/step
Validation Mean Squared Error: 9.624261759485362
Validation Mean Absolute Error: 2.553310760079998
Best model saved to: /content/IoT AAI-530 Final
Project/project/report/best_model

df_model_stats = pd.DataFrame(model_stats)

# PLOT VALIDATION MSE FOR DIFFERENT HYPERPARAMETERS
plt.figure(figsize=(10, 6), dpi=88)
for optimizer in df_model_stats['optimizer'].unique():
    for dropout_rate in df_model_stats['dropout_rate'].unique():
        subset = df_model_stats[(df_model_stats['optimizer'] ==
optimizer) & (df_model_stats['dropout_rate'] == dropout_rate)]
        plt.plot(subset['val_mse'], label=f'Optimizer: {optimizer},
Dropout: {dropout_rate}', marker='o')
plt.title('Validation MSE by Optimizer and Dropout Rate')
plt.xlabel('Experiment Number')
plt.ylabel('Validation MSE')
plt.legend()
plt.show()
```



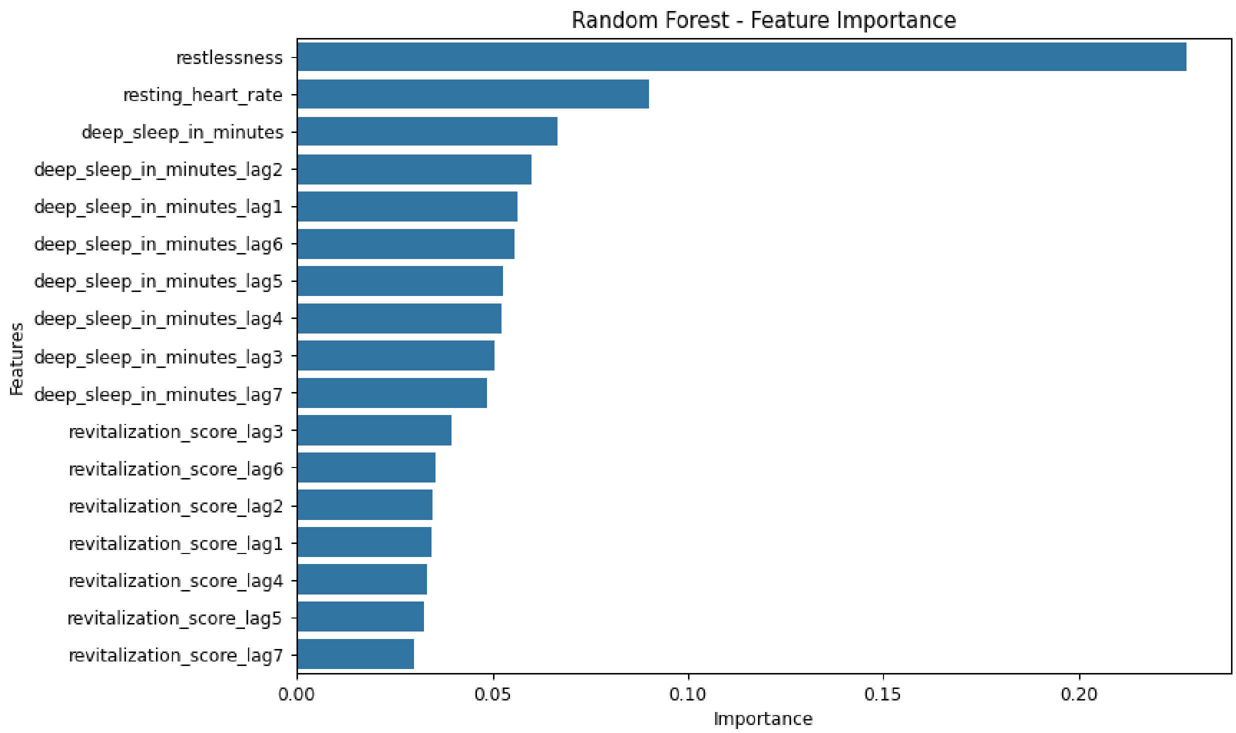
```
# PLOT VALIDATION MAE FOR DIFFERENT HYPERPARAMETERS
plt.figure(figsize=(10, 6), dpi=88)
for optimizer in df_model_stats['optimizer'].unique():
    for dropout_rate in df_model_stats['dropout_rate'].unique():
        subset = df_model_stats[(df_model_stats['optimizer'] ==
optimizer) & (df_model_stats['dropout_rate'] == dropout_rate)]
        plt.plot(subset['val_mae'], label=f'Optimizer: {optimizer},
Dropout: {dropout_rate}', marker='o')
plt.title('Validation MAE by Optimizer and Dropout Rate')
plt.xlabel('Experiment Number')
plt.ylabel('Validation MAE')
plt.legend()
plt.show()
```



Feature Importance

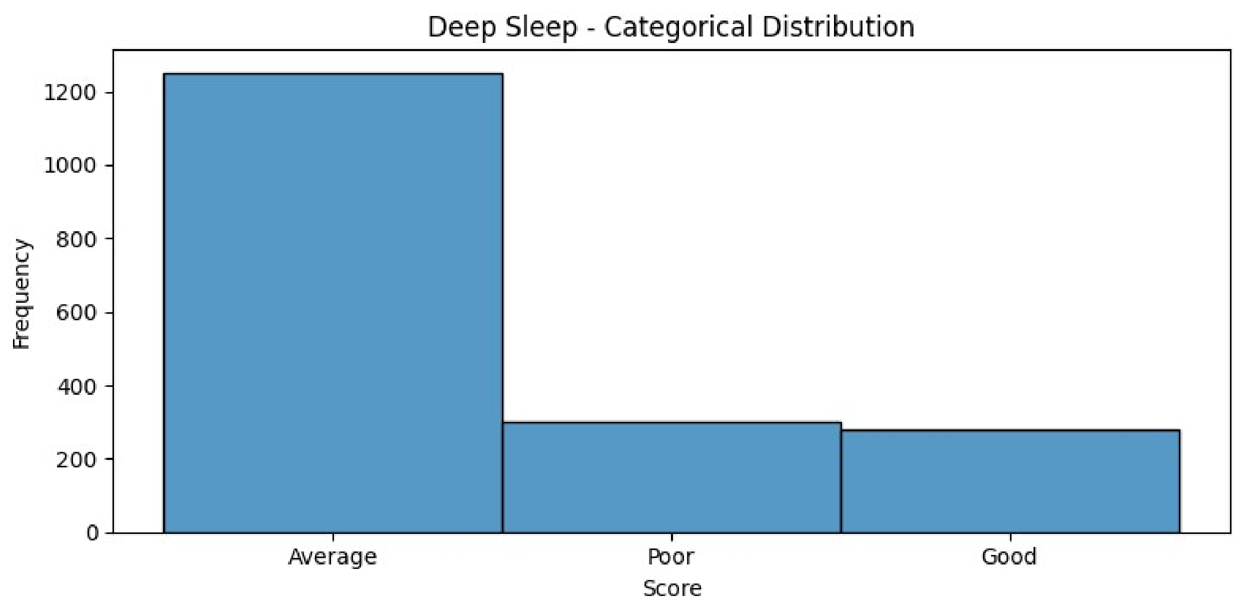
```
target = df['revitalization_score']
features_columns = ['deep_sleep_in_minutes', 'resting_heart_rate',
'restlessness'] + \
                   [f'deep_sleep_in_minutes_lag{lag}' for lag in
range(1, 8)] + \
                   [f'revitalization_score_lag{lag}' for lag in
range(1, 8)]

features = df[features_columns]
```

```
# FEATURE IMPORTANCES
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(features, target)

importances = rf.feature_importances_
importances_series = pd.Series(importances, index=features_columns)
sorted_importances = importances_series.sort_values(ascending=False)

# PLOT
plt.figure(figsize=(10, 6), dpi=88)
sns.barplot(x=sorted_importances, y=sorted_importances.index)
plt.xlabel('Importance')
plt.ylabel('Features')
plt.title('Random Forest - Feature Importance')

pd.DataFrame({
    'feature': sorted_importances.index,
    'importance': sorted_importances.values
}).to_csv(f'{REPO_REPORT_DIR}/feature_importances.csv', index=False)
plt.savefig(f'{REPO_REPORT_DIR}/feature_importance_plot.svg',
format='svg')
plt.show()
```



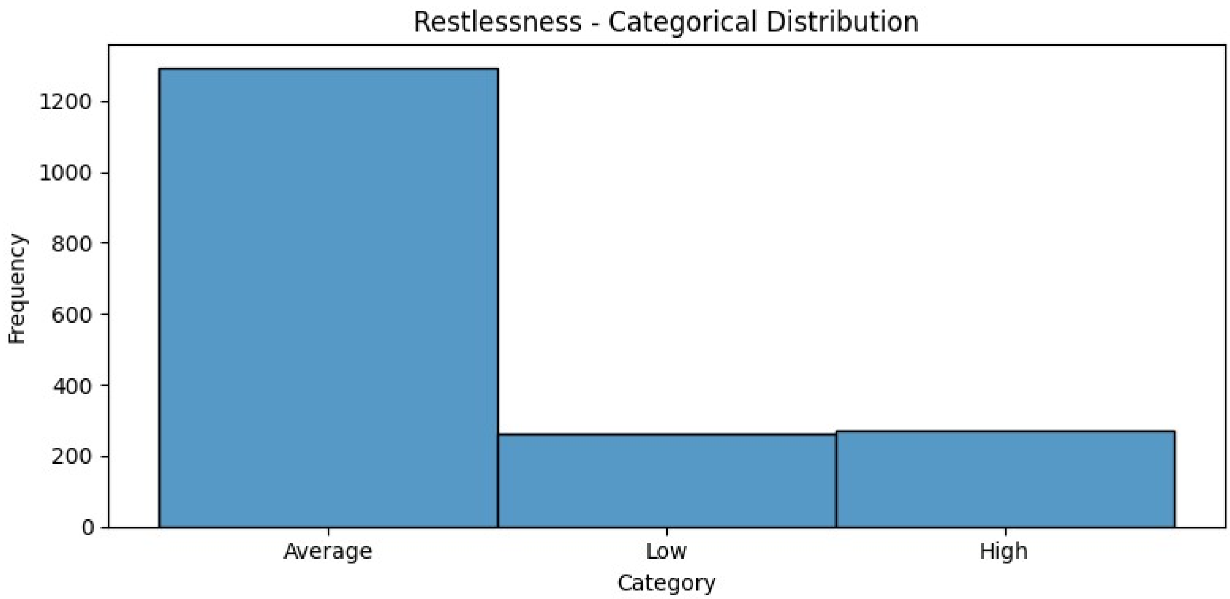

```
# BIN DEEP SLEEP BASED ON Z-SCORES
mean = df['deep_sleep_in_minutes'].mean()
std_dev = df['deep_sleep_in_minutes'].std()
```

```
df['z_score'] = (df['deep_sleep_in_minutes'] - mean) / std_dev
df['deep_sleep_category'] = df['z_score'].apply(lambda z: 'Poor' if z
< -1 else ('Average' if z < 1 else 'Good'))

# BIN RESTLESSNESS BASED ON Z-SCORES
mean_restlessness = df['restlessness'].mean()
std_dev_restlessness = df['restlessness'].std()
df['z_score_restlessness'] = (df['restlessness'] - mean_restlessness)
/ std_dev_restlessness
df['restlessness_category'] = df['z_score_restlessness'].apply(lambda
z: 'Low' if z < -1 else ('Average' if z < 1 else 'High'))

# DISTRIBUTION OF DEEP SLEEP - CATEGORICAL VARIABLE
plt.figure(figsize=(8, 4))
sns.histplot(df['deep_sleep_category'], kde=False, bins=20)
plt.title('Deep Sleep - Categorical Distribution')
plt.xlabel('Score')
plt.ylabel('Frequency')
plt.savefig(f'{REPO_REPORT_DIR}/feature_importance_plot.svg',
format='svg')
plt.show()
```




```
# PLOT THE DISTRIBUTION OF THE RESTLESSNESS CATEGORIES
plt.figure(figsize=(8, 4))
sns.histplot(df['restlessness_category'], kde=False, bins=20)
plt.title('Restlessness - Categorical Distribution')
plt.xlabel('Category')
plt.ylabel('Frequency')
plt.savefig(f'{REPO_REPORT_DIR}/restlessness_category.svg',
```

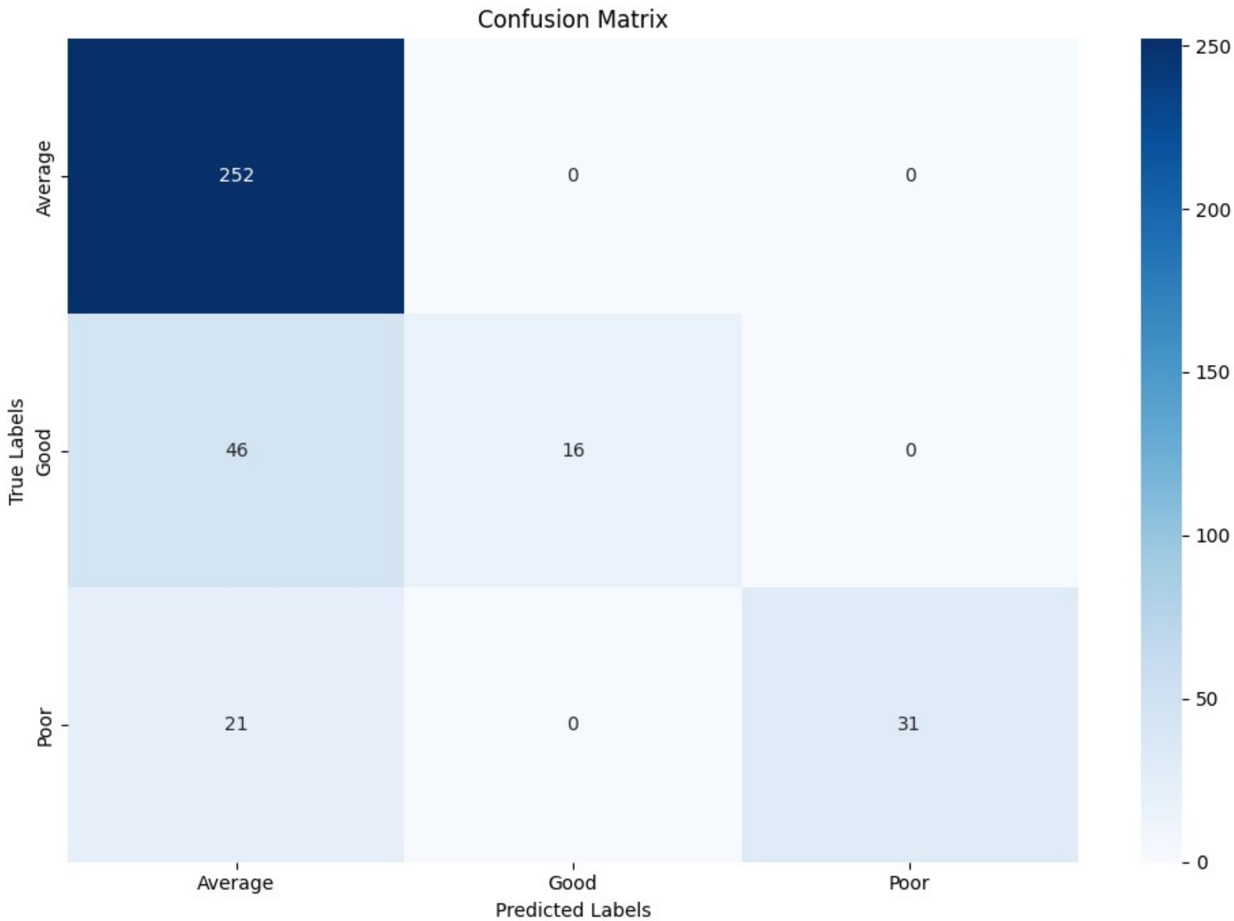
```
format='svg')
plt.show()
```



Traditional ML Classifier

```
# ENCODE CATEGORICAL VARIABLES
label_encoder_deep_sleep = LabelEncoder()
df['deep_sleep_category_encoded'] =
label_encoder_deep_sleep.fit_transform(df['deep_sleep_category'])

label_encoder_restlessness = LabelEncoder()
df['restlessness_category_encoded'] =
label_encoder_restlessness.fit_transform(df['restlessness_category'])

deep_sleep_features_columns = ['resting_heart_rate', 'restlessness'] +
\
    [f'deep_sleep_in_minutes_lag{lag}' for lag in range(1, 8)] + \
    [f'overall_score_lag{lag}' for lag in range(1, 8)] + \
    [f'restlessness_lag{lag}' for lag in range(1, 8)]

deep_sleep_features = df[deep_sleep_features_columns]
deep_sleep_target = df['deep_sleep_category_encoded']

scaler = StandardScaler()
deep_sleep_features_scaled = scaler.fit_transform(deep_sleep_features)

# MODEL BUILDING AND TUNING
param_grid = {'n_estimators': [50, 100, 200], 'max_depth': [5, 10,
None]}
```

```python
rf_clf = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(rf_clf, param_grid, cv=5)
grid_search.fit(deep_sleep_features_scaled, deep_sleep_target)
best_model = grid_search.best_estimator_

# SPLIT
X_train_deep_sleep, X_val_deep_sleep, y_train_deep_sleep,
y_val_deep_sleep = train_test_split(
    deep_sleep_features_scaled,
    deep_sleep_target,
    test_size=0.2,
    shuffle=False,
    random_state=0
)

# EVALUATION
y_val_pred = best_model.predict(X_val_deep_sleep)
plt.figure(figsize=(10, 7))
sns.heatmap(confusion_matrix(y_val_deep_sleep, y_val_pred),
annot=True, fmt='d', cmap='Blues',
xticklabels=label_encoder_deep_sleep.classes_,
yticklabels=label_encoder_deep_sleep.classes_)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')

pd.DataFrame(
    classification_report(y_val_deep_sleep, y_val_pred,
zero_division=0, output_dict=True)
).transpose().to_csv(f'{REPO_REPORT_DIR}/classification_report.csv',
index=True)
pd.DataFrame(
    confusion_matrix(y_val_deep_sleep, y_val_pred)
).to_csv(f'{REPO_REPORT_DIR}/confusion_matrix.csv', index=True)

plt.savefig(f'{REPO_REPORT_DIR}/ml_classifier-confusion_matrix.svg',
format='svg')
plt.show()
```




```
# RESTLESSNESS_CATEGORY CLASSIFICATION
restlessness_features_columns = ['resting_heart_rate',
'deep_sleep_in_minutes'] + \
    [f'restlessness_lag{lag}' for lag in range(1, 8)] + \
    [f'overall_score_lag{lag}' for lag in range(1, 8)] + \
    [f'deep_sleep_in_minutes_lag{lag}' for lag in range(1, 8)]

restlessness_features = df[restlessness_features_columns]
restlessness_target = df['restlessness_category_encoded']

# SPLIT
X_train_restlessness, X_val_restlessness, y_train_restlessness,
y_val_restlessness = train_test_split(
    restlessness_features,
    restlessness_target,
    test_size=0.2,
    shuffle=False,
    random_state=0
)

restlessness_classifier = RandomForestClassifier(n_estimators=100,
```

```
random_state=42)
restlessness_classifier.fit(X_train_restlessness,
y_train_restlessness)
y_val_pred_restlessness =
restlessness_classifier.predict(X_val_restlessness)

# EVALUATE THE CLASSIFIER
print("Classification report for restlessness_category:")
print(classification_report(y_val_restlessness,
y_val_pred_restlessness, zero_division=0))
print("Confusion matrix for restlessness_category:")
print(confusion_matrix(y_val_restlessness, y_val_pred_restlessness))

Classification report for restlessness_category:
              precision    recall  f1-score   support

           0       0.75      0.97      0.85       270
           1       0.43      0.12      0.19        50
           2       0.00      0.00      0.00        46

    accuracy                           0.73       366
   macro avg       0.39      0.36      0.34       366
weighted avg       0.61      0.73      0.65       366

Confusion matrix for restlessness_category:
[[263   7   0]
 [ 44   6   0]
 [ 45   1   0]]
```

Binary Classification

```
# ADJUSTING FOR BINARY
df['deep_sleep_category'] = df['z_score'].apply(lambda z: 'Low' if z <
-1 else 'High' if z > 1 else 'Discard')
df = df[df['deep_sleep_category'] != 'Discard'].copy()

# ENCODE
label_encoder_deep_sleep = LabelEncoder()
df.loc[:, 'deep_sleep_category_encoded'] =
label_encoder_deep_sleep.fit_transform(df['deep_sleep_category'])

# SET FEATURES
deep_sleep_features = df[deep_sleep_features_columns]
deep_sleep_target = df['deep_sleep_category_encoded']

X_train_deep_sleep, X_val_deep_sleep, y_train_deep_sleep,
y_val_deep_sleep = train_test_split(
    deep_sleep_features,
```

```
    deep_sleep_target,
    test_size=0.2,
    shuffle=False,
    random_state=0
)

deep_sleep_classifier = RandomForestClassifier(n_estimators=100,
random_state=42)
deep_sleep_classifier.fit(X_train_deep_sleep, y_train_deep_sleep)

# PREDICTIONS & EVALUATION
y_val_pred_deep_sleep =
deep_sleep_classifier.predict(X_val_deep_sleep)
print("Classification report for deep_sleep_category:")
print(classification_report(y_val_deep_sleep, y_val_pred_deep_sleep,
zero_division=0))
print("Confusion matrix for deep_sleep_category:")
print(confusion_matrix(y_val_deep_sleep, y_val_pred_deep_sleep))

Classification report for deep_sleep_category:
              precision    recall  f1-score   support

           0       0.83      0.79      0.81        63
           1       0.77      0.81      0.79        53

    accuracy                           0.80       116
   macro avg       0.80      0.80      0.80       116
weighted avg       0.80      0.80      0.80       116

Confusion matrix for deep_sleep_category:
[[50 13]
 [10 43]]

# ADJUSTING FOR BINARY
df = df.copy()
df.loc[:, 'restlessness_category'] =
df['z_score_restlessness'].apply(lambda z: 'Low' if z < -1 else 'High'
if z > 1 else 'Discard')
df = df.loc[df['restlessness_category'] != 'Discard'].copy()

# ENCODE
label_encoder_restlessness = LabelEncoder()
df.loc[:, 'restlessness_category_encoded'] =
label_encoder_restlessness.fit_transform(df['restlessness_category'])

# SET FEATURES
restlessness_features = df[restlessness_features_columns]
restlessness_target = df['restlessness_category_encoded']

X_train_restlessness, X_val_restlessness, y_train_restlessness,
```

```
y_val_restlessness = train_test_split(
    restlessness_features,
    restlessness_target,
    test_size=0.2,
    shuffle=False,
    random_state=0
)

restlessness_classifier = RandomForestClassifier(n_estimators=100,
random_state=42)
restlessness_classifier.fit(X_train_restlessness,
y_train_restlessness)

# PREDICTIONS & EVALUATION
y_val_pred_restlessness =
restlessness_classifier.predict(X_val_restlessness)
print("Classification report for restlessness_category:")
print(classification_report(y_val_restlessness,
y_val_pred_restlessness, zero_division=0))
print("Confusion matrix for restlessness_category:")
print(confusion_matrix(y_val_restlessness, y_val_pred_restlessness))
```

```
Classification report for restlessness_category:
              precision    recall  f1-score   support

           0       0.88      0.92      0.90        24
           1       0.75      0.67      0.71         9

    accuracy                           0.85        33
   macro avg       0.81      0.79      0.80        33
weighted avg       0.84      0.85      0.85        33

Confusion matrix for restlessness_category:
[[22  2]
 [ 3  6]]
```

```
# Colab2PDF v1.0.4 by Drengskapur (github.com/drengskapur/colab2pdf)
(License: GPL-3.0-or-later)
# @title {display-mode:"form"}
# @markdown ⬇️ Download PDF
def colab2pdf():
    ENABLE=True # @param {type:"boolean"}
    if ENABLE:
        !apt-get install librsvg2-bin
        import os, datetime, json, locale, pathlib, urllib, requests,
werkzeug, nbformat, google, yaml, warnings
        locale.setlocale(locale.LC_ALL, 'en_US.UTF-8')
        NAME =
pathlib.Path(werkzeug.utils.secure_filename(urllib.parse.unquote(reque
sts.get(f"http://{os.environ['COLAB_JUPYTER_IP']}:
```

```
{os.environ['KMP_TARGET_PORT']}/api/sessions").json()[0]["name"])))
        TEMP = pathlib.Path("/content/pdfs") /
f"{datetime.datetime.now().strftime('%Y%m%d_%H%M%S')}_{NAME.stem}";
TEMP.mkdir(parents=True, exist_ok=True)
        NB = [cell for cell in
nbformat.reads(json.dumps(google.colab._message.blocking_request("get_
ipynb", timeout_sec=600)["ipynb"]), as_version=4).cells if "--
Colab2PDF" not in cell.source]
        warnings.filterwarnings('ignore',
category=nbformat.validator.MissingIDFieldWarning)
        with (TEMP / f"{NAME.stem}.ipynb").open("w", encoding="utf-8")
as nb_copy: nbformat.write(nbformat.v4.new_notebook(cells=NB or
[nbformat.v4.new_code_cell("#")]), nb_copy)
        if not pathlib.Path("/usr/local/bin/quarto").exists():
            !wget -q "https://quarto.org/download/latest/quarto-linux-
amd64.deb" -P {TEMP} && dpkg -i {TEMP}/quarto-linux-amd64.deb >
/dev/null && quarto install tinytex --update-path
        with (TEMP / "config.yml").open("w", encoding="utf-8") as
file: yaml.dump({'include-in-header': [{"text": r"\
usepackage{fvextra}\DefineVerbatimEnvironment{Highlighting}{Verbatim}
{breaksymbolleft={},showspaces=false,showtabs=false,breaklines,breakan
ywhere,commandchars=\\\{\}}"}],'include-before-body': [{"text": r"\
DefineVerbatimEnvironment{verbatim}{Verbatim}
{breaksymbolleft={},showspaces=false,showtabs=false,breaklines}"}]},
file)
        !quarto render {TEMP}/{NAME.stem}.ipynb --metadata-
file={TEMP}/config.yml --to pdf -M latex-auto-install -M margin-
top=1in -M margin-bottom=1in -M margin-left=1in -M margin-right=1in
        google.colab.files.download(str(TEMP / f"{NAME.stem}.pdf"))
    print("Download PDF is not enabled.")
colab2pdf()
```

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  librsvg2-bin
0 upgraded, 1 newly installed, 0 to remove and 35 not upgraded.
Need to get 1,871 kB of archives.
After this operation, 6,019 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64
librsvg2-bin amd64 2.52.5+dfsg-3ubuntu0.2 [1,871 kB]
Fetched 1,871 kB in 2s (1,201 kB/s)
Selecting previously unselected package librsvg2-bin.
(Reading database ... 121749 files and directories currently
installed.)
Preparing to unpack .../librsvg2-bin_2.52.5+dfsg-
3ubuntu0.2_amd64.deb ...
Unpacking librsvg2-bin (2.52.5+dfsg-3ubuntu0.2) ...
```

```
Setting up librsvg2-bin (2.52.5+dfsg-3ubuntu0.2) ...
Processing triggers for man-db (2.10.2-1) ...
```