

Huy_Hung_Phạm_MIS780A2_Task1

September 6, 2025

1 MIS780 Advanced AI For Business - Assessment 2 - T2 2025

1.1 Task Number: Solar Power Prediction from Tabular Data

Student Name: *Huy Hung Phạm*

Student ID: *s224212292*

1.2 Table of Content

1. Executive Summary
2. Data Preprocessing
3. Predictive Modeling
4. Experiments Report
5. Role of GenAI

1. Executive Summary

Accurate prediction of solar photovoltaic (PV) power output is crucial for mitigating the inherent variability in renewable energy sources due to fluctuating weather conditions. This project tackles the challenge of enhancing the reliability of PV generation forecasts to promote grid stability, optimize energy storage solutions, and support sustainable energy management practices. A comprehensive dataset was employed, encompassing solar power output alongside meteorological variables (temperature, humidity, radiation) and solar geometry parameters (zenith, azimuth, and angle of incidence), which was divided into 70% for training and 30% for testing.

A range of predictive models was developed and assessed, starting with a Multiple Linear Regression (MLR) baseline, including rigorous feature selection, and a variety of Artificial Neural Networks (ANNs) with different architectures. The MLR model demonstrated a correlation coefficient of 0.81, with a Mean Absolute Error (MAE) of 420.01 and a Root Mean Square Error (RMSE) of 544.31, highlighting its interpretability while exposing its limitations in capturing nonlinear interactions. In contrast, the ANN models exhibited superior performance by effectively learning complex nonlinear dependencies among environmental and geometric variables. Notably, ANN 11 (with a configuration of 128–96–64–32 hidden nodes, utilizing ReLU activation and Adam optimization) and ANN 12 (featuring 64–32 nodes, employing Tanh activation and Nadam optimization with dropout) achieved remarkable results, with correlation coefficients exceeding 0.88, and a reduction in RMSE to 431.14 and an MAE of 290.42. The training and validation curves for these models displayed smooth convergence, while scatter plots indicated predictions closely aligned along the 45-degree diagonal, underscoring robust generalization and high accuracy.

These findings corroborate contemporary literature illustrating the superiority of ANNs over regression methodologies for solar forecasting (Hanis Nasuha Amer et al., 2023; Keddouda et al., 2023). The results indicate that ANN models can significantly enhance forecasting precision, yielding substantial real-world advantages, including diminished dependence on fossil-fuel backup systems, optimized scheduling of energy storage, and improved grid reliability. Nevertheless, effective implementation necessitates meticulous attention to data quality, comprehensive feature selection, regular retraining to accommodate seasonal fluctuations and panel degradation, and seamless integration with existing energy management frameworks.

In summary, the project illustrates that while regression models serve as a useful benchmark, ANN architectures offer enhanced accuracy and superior predictive capabilities, positioning them as a more effective solution for solar forecasting applications. With careful implementation, ANN-based models can significantly contribute to the transition towards sustainable energy systems by facilitating more reliable and efficient utilization of solar power resources.

1.2.1 2.1 Data preparation

Load some standard Python libraries.

```
[171]: from __future__ import print_function
import os
import math
import datetime
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Next, load Sklearn and its wrappers

```
[172]: from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import ConfusionMatrixDisplay as_
    ↪SklearnConfusionMatrixDisplay # Import with an alias
```

```
[173]: pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
```

Load dataset on the book

```
[174]: solar_data = pd.read_csv("/content/Task 1 Dataset Solar Power.csv")
display(solar_data.head(10))
print('Number of records read: ', solar_data.size)
```

Index	temperature_2_m_above_gnd	relative_humidity_2_m_above_gnd	\
0	1	2.17	31
1	2	2.31	27
2	3	3.65	33
3	4	5.82	30
4	5	7.73	27
5	6	8.69	29
6	7	9.72	27
7	8	10.07	28
8	9	9.38	32
9	10	6.54	47

mean_sea_level_pressure_MSL	total_precipitation_sfc	snowfall_amount_sfc	\
1035.0	0.0	0.0	0.0
1035.1	0.0	0.0	0.0
1035.4	0.0	0.0	0.0
1035.4	0.0	0.0	0.0
1034.4	0.0	0.0	0.0
1034.6	0.0	0.0	0.0
1034.0	0.0	0.0	0.0
1034.1	0.0	0.0	0.0
1033.9	0.0	0.0	0.0
1035.1	0.0	0.0	0.0

total_cloud_cover_sfc	high_cloud_cover_high_cld_lay	\
0.0	0	
0.0	0	
0.0	0	
0.0	0	
0.0	0	
0.0	0	
0.0	0	
0.0	0	
0.0	0	
0.0	0	

medium_cloud_cover_mid_cld_lay	low_cloud_cover_low_cld_lay	\
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0

	shortwave_radiation_backwards_sfc	wind_speed_10_m_above_gnd	\	
0	0.00	6.37		
1	1.78	5.15		
2	108.58	4.68		
3	258.10	3.60		
4	375.58	6.21		
5	449.45	6.29		
6	469.92	5.96		
7	436.99	7.07		
8	353.33	9.42		
9	228.73	7.92		
	wind_direction_10_m_above_gnd	wind_speed_80_m_above_gnd	\	
0	312.71	9.36		
1	294.78	5.99		
2	270.00	3.89		
3	323.13	3.55		
4	10.01	6.76		
5	23.63	7.10		
6	25.02	6.61		
7	14.74	7.63		
8	6.58	10.50		
9	360.00	12.25		
	wind_direction_80_m_above_gnd	wind_speed_900_mb	wind_direction_900_mb \	
0	22.62	6.62	337.62	
1	32.74	4.61	321.34	
2	56.31	3.76	286.70	
3	23.96	3.08	339.44	
4	25.20	6.62	22.38	
5	30.47	6.92	27.90	
6	29.36	6.44	26.57	
7	19.29	7.52	16.70	
8	5.91	10.14	6.12	
9	1.68	11.53	1.79	
	wind_gust_10_m_above_gnd	angle_of_incidence	zenith	azimuth \
0	24.48	58.753108	83.237322	128.33543
1	21.96	45.408585	75.143041	139.65530
2	14.04	32.848282	68.820648	152.53769
3	19.80	22.699288	64.883536	166.90159
4	16.56	19.199908	63.795208	182.13526
5	17.28	25.088167	65.700860	197.22062
6	12.96	36.197514	70.351498	211.21422
7	10.80	49.073008	77.228779	223.65641
8	11.52	62.371140	85.585411	234.50308
9	12.24	105.866560	121.463180	216.06756

```
generated_power_kw
0           454.100950
1          1411.999400
2          2214.849300
3          2527.609200
4          2640.203400
5          2546.081600
6          2270.320700
7          1063.830200
8           86.817611
9           9.666667

Number of records read: 92686
```

```
[175]: # Finding column types
solar_data.dtypes
```

```
[175]: Index                  int64
temperature_2_m_above_gnd      float64
relative_humidity_2_m_above_gnd int64
mean_sea_level_pressure_MSL    float64
total_precipitation_sfc       float64
snowfall_amount_sfc           float64
total_cloud_cover_sfc         float64
high_cloud_cover_high_cld_lay int64
medium_cloud_cover_mid_cld_lay int64
low_cloud_cover_low_cld_lay   int64
shortwave_radiation_backwards_sfc float64
wind_speed_10_m_above_gnd     float64
wind_direction_10_m_above_gnd  float64
wind_speed_80_m_above_gnd     float64
wind_direction_80_m_above_gnd  float64
wind_speed_900_mb              float64
wind_direction_900_mb          float64
wind_gust_10_m_above_gnd      float64
angle_of_incidence             float64
zenith                         float64
azimuth                         float64
generated_power_kw             float64
dtype: object
```

1.2.2 2.2. Identification of missing values

```
[176]: # Identification of missing values
missing = solar_data.isnull().sum()
missing = missing[missing > 0]
missing.sort_values(ascending=False)
```

```
[176]: Series([], dtype: int64)
```

```
[177]: solar_data.describe(include='all')
```

```
[177]:      Index  temperature_2_m_above_gnd  \
count    4213.000000          4213.000000
mean     2107.000000          15.068111
std      1216.332671          8.853677
min      1.000000           -5.350000
25%     1054.000000          8.390000
50%     2107.000000          14.750000
75%     3160.000000          21.290000
max     4213.000000          34.900000

      relative_humidity_2_m_above_gnd  mean_sea_level_pressure_MSL  \
count                4213.000000          4213.000000
mean                 51.361025         1019.337812
std                  23.525864          7.022867
min                  7.000000          997.500000
25%                 32.000000          1014.500000
50%                 48.000000          1018.100000
75%                 70.000000          1023.600000
max                100.000000          1046.800000

      total_precipitation_sfc  snowfall_amount_sfc  total_cloud_cover_sfc  \
count                4213.000000          4213.000000          4213.000000
mean                 0.031759          0.002808          34.056990
std                  0.170212          0.038015          42.843638
min                  0.000000          0.000000          0.000000
25%                 0.000000          0.000000          0.000000
50%                 0.000000          0.000000          8.700000
75%                 0.000000          0.000000         100.000000
max                 3.200000          1.680000         100.000000

      high_cloud_cover_high_cld_lay  medium_cloud_cover_mid_cld_lay  \
count                4213.000000          4213.000000
mean                 14.458818          20.023499
std                  30.711707          36.387948
min                  0.000000          0.000000
25%                 0.000000          0.000000
50%                 0.000000          0.000000
75%                 9.000000          10.000000
max                100.000000         100.000000

      low_cloud_cover_low_cld_lay  shortwave_radiation_backwards_sfc  \
count                4213.000000          4213.000000
mean                 21.373368         387.759036
```

std	38.013885		278.459293
min	0.000000		0.000000
25%	0.000000		142.400000
50%	0.000000		381.810000
75%	10.000000		599.860000
max	100.000000		952.300000
count	4213.000000	4213.000000	\
mean	16.228787	195.078452	
std	9.876948	106.626782	
min	0.000000	0.540000	
25%	9.010000	153.190000	
50%	14.460000	191.770000	
75%	21.840000	292.070000	
max	61.180000	360.000000	
count	4213.000000	4213.000000	\
mean	18.978483	191.166862	
std	11.999960	108.760021	
min	0.000000	1.120000	
25%	10.140000	130.240000	
50%	16.240000	187.770000	
75%	26.140000	292.040000	
max	66.880000	360.000000	
count	4213.000000	4213.000000	\
mean	16.36319	192.447911	4213.000000
std	9.88533	106.516195	20.583489
min	0.00000	1.120000	12.648899
25%	9.18000	148.220000	0.720000
50%	14.49000	187.990000	11.160000
75%	21.97000	288.000000	18.000000
max	61.11000	360.000000	27.000000
count	4213.000000	4213.000000	4213.000000
mean	50.837490	59.980947	1134.347313
std	26.638965	19.857711	937.957247
min	3.755323	17.727761	0.000595
25%	29.408181	45.291631	231.700450
50%	47.335557	62.142611	971.642650
75%	69.197492	74.346737	2020.966700
max	121.635920	128.415370	3056.794100

1.2.3 2.3. Splitting dataset

```
[178]: #random state is the student id
seed = 224212292
train_size, valid_size, test_size = (0.7, 0.3, 0.0)
solar_train, solar_valid = train_test_split(solar_data,
                                             test_size=valid_size,
                                             random_state=seed)
```

```
[179]: label_col = 'generated_power_kw'
solar_y_train = solar_train[[label_col]]
solar_x_train = solar_train.drop(label_col, axis=1)
solar_y_valid = solar_valid[[label_col]]
solar_x_valid = solar_valid.drop(label_col, axis=1)

print('Size of training set: ', len(solar_x_train))
print('Size of validation set: ', len(solar_x_valid))
```

Size of training set: 2949
Size of validation set: 1264

Before the data can be applied to a deep learning model. Missing values needs to be dealed with, and the data needs to be scaled to [-1,1] range.

```
[180]: print('Missing training values before imputation = ', solar_x_train.isnull().
           ~sum().sum())
print('Missing validation values before imputation = ', solar_x_valid.isnull().
           ~sum().sum())

imputer = SimpleImputer(missing_values=np.nan, strategy='mean').
           ~fit(solar_x_train)
solar_x_train = pd.DataFrame(imputer.transform(solar_x_train),
                             columns = solar_x_train.columns, index = ~
                             solar_x_train.index)
solar_x_valid = pd.DataFrame(imputer.transform(solar_x_valid),
                             columns = solar_x_valid.columns, index = ~
                             solar_x_valid.index)

print('Missing training values after imputation = ', solar_x_train.isnull().
           ~sum().sum())
print('Missing validation values after imputation = ', solar_x_valid.isnull().
           ~sum().sum())
```

Missing training values before imputation = 0
Missing validation values before imputation = 0
Missing training values after imputation = 0
Missing validation values after imputation = 0

Creating a scaling model using training set and use it to scale both training and validation data.

```
[181]: scaler = MinMaxScaler(feature_range=(0, 1), copy=True).fit(solar_x_train)
solar_x_train = pd.DataFrame(scaler.transform(solar_x_train),
                             columns = solar_x_train.columns, index = solar_x_train.index)
solar_x_valid = pd.DataFrame(scaler.transform(solar_x_valid),
                             columns = solar_x_valid.columns, index = solar_x_valid.index)

print('X train min =', round(solar_x_train.min().min(),4), ' ; max =', round(solar_x_train.max().max(), 4))
print('X valid min =', round(solar_x_valid.min().min(),4), ' ; max =', round(solar_x_valid.max().max(), 4))
```

X train min = 0.0 ; max = 1.0
X valid min = -0.0165 ; max = 1.0

```
[182]: solar_x_valid.head(10)
```

	Index	temperature_2_m_above_gnd	relative_humidity_2_m_above_gnd	\
3278	0.778147	0.394743	0.880435	
108	0.025178	0.442053	0.250000	
1161	0.275297	0.529662	0.358696	
2273	0.539430	0.892616	0.054348	
872	0.206651	0.349687	0.576087	
2939	0.697625	0.764205	0.250000	
913	0.216390	0.454819	0.195652	
3528	0.837530	0.525407	0.478261	
3641	0.864371	0.389987	0.706522	
3749	0.890024	0.513141	0.402174	
	mean_sea_level_pressure_MSL	total_precipitation_sfC	\	
3278	0.375258	0.0		
108	0.523711	0.0		
1161	0.294845	0.0		
2273	0.348454	0.0		
872	0.649485	0.0		
2939	0.356701	0.0		
913	0.581443	0.0		
3528	0.501031	0.0		
3641	0.251546	0.0		
3749	0.630928	0.0		
	snowfall_amount_sfC	total_cloud_cover_sfC	\	
3278	0.0	1.000		
108	0.0	0.000		
1161	0.0	1.000		
2273	0.0	0.000		

872	0.0	0.000
2939	0.0	0.000
913	0.0	0.009
3528	0.0	0.000
3641	0.0	1.000
3749	0.0	0.081
	high_cloud_cover_high_cld_lay	medium_cloud_cover_mid_cld_lay \
3278	0.00	1.0
108	0.00	0.0
1161	1.00	1.0
2273	0.00	0.0
872	0.00	0.0
2939	0.00	0.0
913	0.03	0.0
3528	0.00	0.0
3641	1.00	1.0
3749	0.27	0.0
	low_cloud_cover_low_cld_lay	shortwave_radiation_backwards_sfc \
3278	1.0	0.000000
108	0.0	0.475701
1161	1.0	0.249533
2273	0.0	0.779439
872	0.0	0.057009
2939	0.0	0.852336
913	0.0	0.478505
3528	0.0	0.334579
3641	0.0	0.385981
3749	0.0	0.498131
	wind_speed_10_m_above_gnd	wind_direction_10_m_above_gnd \
3278	0.333933	0.459523
108	0.364662	0.721193
1161	0.349624	0.599677
2273	0.241909	0.613448
872	0.102975	0.835754
2939	0.117686	0.499249
913	0.075351	0.106048
3528	0.230794	0.052356
3641	0.516018	0.519307
3749	0.122916	0.044956
	wind_speed_80_m_above_gnd	wind_direction_80_m_above_gnd \
3278	0.409988	0.460198
108	0.389504	0.718072
1161	0.376794	0.601065

2273	0.232955	0.610653	
872	0.316537	0.884191	
2939	0.113038	0.498300	
913	0.084031	0.107559	
3528	0.224880	0.055243	
3641	0.571770	0.522437	
3749	0.107207	0.045460	
		\	
3278	0.352970	0.458036	
108	0.366225	0.718207	
1161	0.353461	0.601176	
2273	0.238423	0.609870	
872	0.145475	0.837717	
2939	0.117820	0.498440	
913	0.084929	0.090755	
3528	0.225495	0.052218	
3641	0.522500	0.518307	
3749	0.113729	0.056091	
		0.089744	
	angle_of_incidence	zenith	azimuth
3278	0.793923	0.720470	0.139281
108	0.164835	0.417536	0.604974
1161	0.521808	0.395644	0.886561
2273	0.295049	0.113273	0.276960
872	0.592004	0.516092	0.187291
2939	0.154967	0.167487	0.666857
913	0.340585	0.314864	0.285191
3528	0.497034	0.539685	0.804845
3641	0.172912	0.374410	0.641674
3749	0.105257	0.374711	0.497081

3. Predictive Modeling

```
[183]: import tensorflow as tf
from tensorflow.keras import metrics
from tensorflow.keras import regularizers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Nadam, RMSprop
```

These libraries from TensorFlow/Keras provide the essential tools to build neural networks (Sequential, Dense), prevent overfitting (Dropout, regularizers), optimize learning (Nadam, RMSprop), and evaluate performance (metrics), all powered by the TensorFlow backend (tf).

```
[184]: arr_x_train = np.array(solar_x_train)
arr_y_train = np.array(solar_y_train)
arr_x_valid = np.array(solar_x_valid)
```

```

arr_y_valid = np.array(solar_y_valid)

print('Training shape:', arr_x_train.shape)
print('Training samples: ', arr_x_train.shape[0])
print('Validation samples: ', arr_x_valid.shape[0])

```

Training shape: (2949, 21)
 Training samples: 2949
 Validation samples: 1264

3.1 Define Early Stopping callback

```
[185]: early_stopping = EarlyStopping(monitor='val_loss', patience=10,
                                     restore_best_weights=True)
```

This EarlyStopping callback stops training if validation loss does not improve for 10 epochs and restores the model's best weights to prevent overfitting.

```
[186]: def model_11(x_size, y_size):
    t_model = Sequential()
    t_model.add(Dense(128, activation="relu", input_shape=(x_size,)))
    t_model.add(Dense(96, activation="relu"))
    t_model.add(Dense(64, activation="relu"))
    t_model.add(Dense(32, activation="relu"))
    t_model.add(Dense(y_size))
    t_model.compile(
        loss='mean_squared_error',
        optimizer=Adam(learning_rate=0.001),
        metrics=[metrics.mae])
    return(t_model)
```

```
[187]: model = model_11(arr_x_train.shape[1], arr_y_train.shape[1])
model.summary()
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
    super().__init__(activity_regularizer=activity_regularizer, **kwargs)

Model: "sequential_7"
```

Layer (type)	Output Shape	Param #
dense_29 (Dense)	(None, 128)	2,816
dense_30 (Dense)	(None, 96)	12,384

dense_31 (Dense)	(None, 64)	6,208
dense_32 (Dense)	(None, 32)	2,080
dense_33 (Dense)	(None, 1)	33

Total params: 23,521 (91.88 KB)

Trainable params: 23,521 (91.88 KB)

Non-trainable params: 0 (0.00 B)

The Sequential neural network architecture characterized as a multilayer perceptron (MLP), specifically structured for regression tasks. The model comprises four hidden layers with neuron counts of 128, 96, 64, and 32, respectively. The output layer consists of a single neuron, enabling the model to deliver continuous predictions. Overall, there are 23,521 trainable parameters in the network, facilitating the capture of intricate relationships within the dataset. The architecture systematically reduces dimensionality from 128 to 1, allowing the network to extract and condense complex nonlinear patterns through iterative transformations before generating the final output.

3.2 Fitting the model to prediction

```
[188]: history = model.fit(arr_x_train, arr_y_train,
    batch_size= 64,
    epochs=100,
    shuffle=True,
    verbose=2,
    validation_data=(arr_x_valid, arr_y_valid))
```

```
Epoch 1/100
47/47 - 3s - 71ms/step - loss: 2163449.0000 - mean_absolute_error: 1132.9225 -
val_loss: 1926463.2500 - val_mean_absolute_error: 1042.0187
Epoch 2/100
47/47 - 1s - 12ms/step - loss: 1422574.2500 - mean_absolute_error: 962.5966 -
val_loss: 997526.8750 - val_mean_absolute_error: 875.1594
Epoch 3/100
47/47 - 0s - 6ms/step - loss: 947099.0625 - mean_absolute_error: 856.0670 -
val_loss: 847081.7500 - val_mean_absolute_error: 805.3026
Epoch 4/100
47/47 - 0s - 4ms/step - loss: 795596.8750 - mean_absolute_error: 783.1177 -
val_loss: 706257.8750 - val_mean_absolute_error: 734.5305
Epoch 5/100
47/47 - 0s - 4ms/step - loss: 662343.4375 - mean_absolute_error: 702.7713 -
val_loss: 575579.6250 - val_mean_absolute_error: 643.5261
```

```
Epoch 6/100
47/47 - 0s - 4ms/step - loss: 542386.2500 - mean_absolute_error: 618.0560 -
val_loss: 460064.6562 - val_mean_absolute_error: 568.1095
Epoch 7/100
47/47 - 0s - 7ms/step - loss: 422317.1875 - mean_absolute_error: 535.4118 -
val_loss: 350977.1250 - val_mean_absolute_error: 485.8901
Epoch 8/100
47/47 - 0s - 4ms/step - loss: 325562.0625 - mean_absolute_error: 454.4042 -
val_loss: 281402.5625 - val_mean_absolute_error: 414.4666
Epoch 9/100
47/47 - 0s - 4ms/step - loss: 274482.8125 - mean_absolute_error: 395.9904 -
val_loss: 261854.5781 - val_mean_absolute_error: 385.8439
Epoch 10/100
47/47 - 0s - 4ms/step - loss: 261731.3906 - mean_absolute_error: 373.8172 -
val_loss: 249826.5000 - val_mean_absolute_error: 373.5110
Epoch 11/100
47/47 - 0s - 4ms/step - loss: 258002.6250 - mean_absolute_error: 370.6875 -
val_loss: 246721.2969 - val_mean_absolute_error: 369.7480
Epoch 12/100
47/47 - 0s - 5ms/step - loss: 251697.9531 - mean_absolute_error: 365.6560 -
val_loss: 247551.0938 - val_mean_absolute_error: 369.3434
Epoch 13/100
47/47 - 0s - 6ms/step - loss: 252117.8750 - mean_absolute_error: 366.8315 -
val_loss: 244782.4844 - val_mean_absolute_error: 365.7354
Epoch 14/100
47/47 - 0s - 4ms/step - loss: 245945.0938 - mean_absolute_error: 359.2311 -
val_loss: 243943.8750 - val_mean_absolute_error: 364.6578
Epoch 15/100
47/47 - 0s - 6ms/step - loss: 243609.3906 - mean_absolute_error: 357.4225 -
val_loss: 237236.6875 - val_mean_absolute_error: 358.6289
Epoch 16/100
47/47 - 0s - 6ms/step - loss: 242354.3125 - mean_absolute_error: 354.7432 -
val_loss: 236014.5000 - val_mean_absolute_error: 357.2889
Epoch 17/100
47/47 - 0s - 10ms/step - loss: 242433.9688 - mean_absolute_error: 354.2529 -
val_loss: 237429.4219 - val_mean_absolute_error: 360.0073
Epoch 18/100
47/47 - 0s - 8ms/step - loss: 239081.6250 - mean_absolute_error: 350.2469 -
val_loss: 231520.9375 - val_mean_absolute_error: 353.6848
Epoch 19/100
47/47 - 0s - 8ms/step - loss: 235829.2500 - mean_absolute_error: 348.0358 -
val_loss: 229097.1094 - val_mean_absolute_error: 350.9286
Epoch 20/100
47/47 - 1s - 12ms/step - loss: 234374.2500 - mean_absolute_error: 344.6357 -
val_loss: 228360.0938 - val_mean_absolute_error: 351.1695
Epoch 21/100
47/47 - 0s - 8ms/step - loss: 231145.2344 - mean_absolute_error: 343.0561 -
val_loss: 230015.3438 - val_mean_absolute_error: 351.9773
```

```
Epoch 22/100
47/47 - 0s - 8ms/step - loss: 233448.7344 - mean_absolute_error: 345.8269 -
val_loss: 226023.8281 - val_mean_absolute_error: 346.7642
Epoch 23/100
47/47 - 1s - 12ms/step - loss: 227196.3125 - mean_absolute_error: 337.4640 -
val_loss: 224198.7344 - val_mean_absolute_error: 345.6639
Epoch 24/100
47/47 - 0s - 4ms/step - loss: 228287.4062 - mean_absolute_error: 341.3134 -
val_loss: 234045.2344 - val_mean_absolute_error: 353.5875
Epoch 25/100
47/47 - 0s - 6ms/step - loss: 223736.1875 - mean_absolute_error: 334.1256 -
val_loss: 234663.2969 - val_mean_absolute_error: 355.9551
Epoch 26/100
47/47 - 0s - 4ms/step - loss: 231623.9844 - mean_absolute_error: 345.0016 -
val_loss: 219382.2344 - val_mean_absolute_error: 339.6729
Epoch 27/100
47/47 - 0s - 5ms/step - loss: 223296.2344 - mean_absolute_error: 334.8330 -
val_loss: 220338.0000 - val_mean_absolute_error: 339.7646
Epoch 28/100
47/47 - 0s - 6ms/step - loss: 217986.2812 - mean_absolute_error: 329.1315 -
val_loss: 217761.7656 - val_mean_absolute_error: 336.7242
Epoch 29/100
47/47 - 0s - 6ms/step - loss: 218242.0781 - mean_absolute_error: 329.3405 -
val_loss: 229434.7812 - val_mean_absolute_error: 347.8871
Epoch 30/100
47/47 - 0s - 4ms/step - loss: 221246.7031 - mean_absolute_error: 332.3931 -
val_loss: 215418.8281 - val_mean_absolute_error: 334.2823
Epoch 31/100
47/47 - 0s - 4ms/step - loss: 213354.3594 - mean_absolute_error: 324.7118 -
val_loss: 214778.7344 - val_mean_absolute_error: 333.1645
Epoch 32/100
47/47 - 0s - 6ms/step - loss: 212123.0000 - mean_absolute_error: 321.6934 -
val_loss: 212672.2031 - val_mean_absolute_error: 331.7041
Epoch 33/100
47/47 - 0s - 4ms/step - loss: 208915.5156 - mean_absolute_error: 319.4643 -
val_loss: 218051.0312 - val_mean_absolute_error: 336.3247
Epoch 34/100
47/47 - 0s - 4ms/step - loss: 211049.8438 - mean_absolute_error: 324.2273 -
val_loss: 211167.5625 - val_mean_absolute_error: 325.9102
Epoch 35/100
47/47 - 0s - 7ms/step - loss: 208238.9688 - mean_absolute_error: 318.5451 -
val_loss: 210375.1406 - val_mean_absolute_error: 324.9481
Epoch 36/100
47/47 - 0s - 5ms/step - loss: 206578.7031 - mean_absolute_error: 316.5885 -
val_loss: 208649.5625 - val_mean_absolute_error: 323.3675
Epoch 37/100
47/47 - 0s - 4ms/step - loss: 208776.9375 - mean_absolute_error: 319.9309 -
val_loss: 206890.2188 - val_mean_absolute_error: 320.4789
```

```
Epoch 38/100
47/47 - 0s - 4ms/step - loss: 202103.7812 - mean_absolute_error: 311.3329 -
val_loss: 203920.5938 - val_mean_absolute_error: 318.1519
Epoch 39/100
47/47 - 0s - 4ms/step - loss: 201735.9844 - mean_absolute_error: 310.6988 -
val_loss: 204974.7188 - val_mean_absolute_error: 320.8842
Epoch 40/100
47/47 - 0s - 6ms/step - loss: 199382.7656 - mean_absolute_error: 309.6711 -
val_loss: 213712.4844 - val_mean_absolute_error: 329.2657
Epoch 41/100
47/47 - 0s - 4ms/step - loss: 199693.0469 - mean_absolute_error: 308.1035 -
val_loss: 206498.4688 - val_mean_absolute_error: 322.1942
Epoch 42/100
47/47 - 0s - 6ms/step - loss: 198682.7812 - mean_absolute_error: 307.7085 -
val_loss: 202310.5625 - val_mean_absolute_error: 316.5106
Epoch 43/100
47/47 - 0s - 4ms/step - loss: 197975.1406 - mean_absolute_error: 307.6716 -
val_loss: 199970.8750 - val_mean_absolute_error: 314.4088
Epoch 44/100
47/47 - 0s - 5ms/step - loss: 195009.6250 - mean_absolute_error: 305.0161 -
val_loss: 200980.4062 - val_mean_absolute_error: 310.4848
Epoch 45/100
47/47 - 0s - 4ms/step - loss: 195206.8594 - mean_absolute_error: 303.3418 -
val_loss: 202902.9531 - val_mean_absolute_error: 314.0084
Epoch 46/100
47/47 - 0s - 6ms/step - loss: 193013.8594 - mean_absolute_error: 303.2848 -
val_loss: 208236.6094 - val_mean_absolute_error: 321.7507
Epoch 47/100
47/47 - 0s - 6ms/step - loss: 197358.1562 - mean_absolute_error: 307.1615 -
val_loss: 198666.2656 - val_mean_absolute_error: 316.1447
Epoch 48/100
47/47 - 0s - 4ms/step - loss: 190693.8281 - mean_absolute_error: 301.9201 -
val_loss: 196405.5625 - val_mean_absolute_error: 307.1931
Epoch 49/100
47/47 - 0s - 6ms/step - loss: 189219.0312 - mean_absolute_error: 298.5581 -
val_loss: 198908.4219 - val_mean_absolute_error: 309.5292
Epoch 50/100
47/47 - 0s - 4ms/step - loss: 190555.6250 - mean_absolute_error: 298.6484 -
val_loss: 195178.7656 - val_mean_absolute_error: 307.5101
Epoch 51/100
47/47 - 0s - 4ms/step - loss: 190649.7031 - mean_absolute_error: 299.0001 -
val_loss: 200233.5156 - val_mean_absolute_error: 313.2350
Epoch 52/100
47/47 - 0s - 7ms/step - loss: 191048.9688 - mean_absolute_error: 301.2269 -
val_loss: 206215.8750 - val_mean_absolute_error: 318.3689
Epoch 53/100
47/47 - 0s - 5ms/step - loss: 185728.0469 - mean_absolute_error: 294.6610 -
val_loss: 199346.3438 - val_mean_absolute_error: 310.3768
```

```
Epoch 54/100
47/47 - 0s - 4ms/step - loss: 186155.5938 - mean_absolute_error: 294.1284 -
val_loss: 193751.9688 - val_mean_absolute_error: 303.9312
Epoch 55/100
47/47 - 0s - 4ms/step - loss: 183748.1406 - mean_absolute_error: 291.9110 -
val_loss: 190821.7031 - val_mean_absolute_error: 301.5226
Epoch 56/100
47/47 - 0s - 6ms/step - loss: 183397.5938 - mean_absolute_error: 291.1760 -
val_loss: 191140.5000 - val_mean_absolute_error: 302.1918
Epoch 57/100
47/47 - 0s - 6ms/step - loss: 183078.0156 - mean_absolute_error: 292.0231 -
val_loss: 190969.8750 - val_mean_absolute_error: 300.6938
Epoch 58/100
47/47 - 0s - 6ms/step - loss: 187830.0000 - mean_absolute_error: 297.2120 -
val_loss: 192509.5312 - val_mean_absolute_error: 306.4725
Epoch 59/100
47/47 - 0s - 4ms/step - loss: 180552.3125 - mean_absolute_error: 288.4948 -
val_loss: 189863.2188 - val_mean_absolute_error: 299.7115
Epoch 60/100
47/47 - 0s - 4ms/step - loss: 181824.4844 - mean_absolute_error: 289.7216 -
val_loss: 195758.1719 - val_mean_absolute_error: 308.2994
Epoch 61/100
47/47 - 0s - 5ms/step - loss: 186335.9531 - mean_absolute_error: 298.6024 -
val_loss: 193006.8750 - val_mean_absolute_error: 300.4101
Epoch 62/100
47/47 - 0s - 6ms/step - loss: 182572.0312 - mean_absolute_error: 289.2956 -
val_loss: 192149.4219 - val_mean_absolute_error: 301.8349
Epoch 63/100
47/47 - 0s - 9ms/step - loss: 180528.0938 - mean_absolute_error: 290.0442 -
val_loss: 192985.6719 - val_mean_absolute_error: 303.6893
Epoch 64/100
47/47 - 1s - 12ms/step - loss: 179957.1719 - mean_absolute_error: 288.6674 -
val_loss: 188889.2344 - val_mean_absolute_error: 299.5445
Epoch 65/100
47/47 - 0s - 6ms/step - loss: 178207.0312 - mean_absolute_error: 286.2686 -
val_loss: 187097.7812 - val_mean_absolute_error: 296.4840
Epoch 66/100
47/47 - 0s - 6ms/step - loss: 176328.4844 - mean_absolute_error: 284.3730 -
val_loss: 190409.2969 - val_mean_absolute_error: 297.9579
Epoch 67/100
47/47 - 0s - 8ms/step - loss: 176303.9219 - mean_absolute_error: 283.6588 -
val_loss: 188197.0312 - val_mean_absolute_error: 298.6555
Epoch 68/100
47/47 - 0s - 6ms/step - loss: 179136.0625 - mean_absolute_error: 287.5706 -
val_loss: 186742.8125 - val_mean_absolute_error: 296.4107
Epoch 69/100
47/47 - 0s - 7ms/step - loss: 178036.0156 - mean_absolute_error: 287.4137 -
val_loss: 196708.3594 - val_mean_absolute_error: 308.5320
```

```
Epoch 70/100
47/47 - 1s - 11ms/step - loss: 175905.3438 - mean_absolute_error: 284.4182 -
val_loss: 185545.1406 - val_mean_absolute_error: 290.9470
Epoch 71/100
47/47 - 0s - 6ms/step - loss: 173854.3750 - mean_absolute_error: 281.2794 -
val_loss: 190199.0156 - val_mean_absolute_error: 300.3553
Epoch 72/100
47/47 - 0s - 4ms/step - loss: 174291.1875 - mean_absolute_error: 283.2283 -
val_loss: 189461.1719 - val_mean_absolute_error: 299.0211
Epoch 73/100
47/47 - 0s - 6ms/step - loss: 173138.1719 - mean_absolute_error: 281.5226 -
val_loss: 184413.4219 - val_mean_absolute_error: 291.6765
Epoch 74/100
47/47 - 0s - 7ms/step - loss: 174458.6875 - mean_absolute_error: 282.5288 -
val_loss: 189727.1094 - val_mean_absolute_error: 296.0741
Epoch 75/100
47/47 - 0s - 6ms/step - loss: 172801.9062 - mean_absolute_error: 280.6606 -
val_loss: 196721.8125 - val_mean_absolute_error: 311.9338
Epoch 76/100
47/47 - 0s - 5ms/step - loss: 175173.0469 - mean_absolute_error: 283.0292 -
val_loss: 185096.1562 - val_mean_absolute_error: 293.1737
Epoch 77/100
47/47 - 0s - 5ms/step - loss: 174014.1562 - mean_absolute_error: 284.0148 -
val_loss: 188554.3906 - val_mean_absolute_error: 292.1046
Epoch 78/100
47/47 - 0s - 4ms/step - loss: 177581.1094 - mean_absolute_error: 286.7706 -
val_loss: 184557.7031 - val_mean_absolute_error: 292.5211
Epoch 79/100
47/47 - 0s - 6ms/step - loss: 171033.2188 - mean_absolute_error: 278.7219 -
val_loss: 187902.8125 - val_mean_absolute_error: 299.2914
Epoch 80/100
47/47 - 0s - 4ms/step - loss: 170517.0000 - mean_absolute_error: 279.0638 -
val_loss: 183114.2812 - val_mean_absolute_error: 289.8596
Epoch 81/100
47/47 - 0s - 6ms/step - loss: 171687.7969 - mean_absolute_error: 281.4782 -
val_loss: 192215.4219 - val_mean_absolute_error: 299.1046
Epoch 82/100
47/47 - 0s - 6ms/step - loss: 171036.4688 - mean_absolute_error: 279.2445 -
val_loss: 184525.5156 - val_mean_absolute_error: 292.0282
Epoch 83/100
47/47 - 0s - 4ms/step - loss: 169713.9844 - mean_absolute_error: 276.7233 -
val_loss: 185775.0000 - val_mean_absolute_error: 293.2831
Epoch 84/100
47/47 - 0s - 4ms/step - loss: 168297.8750 - mean_absolute_error: 276.0887 -
val_loss: 183740.2969 - val_mean_absolute_error: 289.6924
Epoch 85/100
47/47 - 0s - 6ms/step - loss: 168795.7500 - mean_absolute_error: 276.5222 -
val_loss: 184452.1562 - val_mean_absolute_error: 294.0648
```

```
Epoch 86/100
47/47 - 0s - 7ms/step - loss: 168590.3125 - mean_absolute_error: 275.5399 -
val_loss: 185125.7656 - val_mean_absolute_error: 293.5458
Epoch 87/100
47/47 - 0s - 4ms/step - loss: 167544.6562 - mean_absolute_error: 275.5255 -
val_loss: 183258.9375 - val_mean_absolute_error: 289.1800
Epoch 88/100
47/47 - 0s - 4ms/step - loss: 168692.4375 - mean_absolute_error: 275.8908 -
val_loss: 183437.3281 - val_mean_absolute_error: 292.0080
Epoch 89/100
47/47 - 0s - 6ms/step - loss: 168140.5156 - mean_absolute_error: 276.4030 -
val_loss: 188793.0312 - val_mean_absolute_error: 293.9113
Epoch 90/100
47/47 - 0s - 9ms/step - loss: 169238.6406 - mean_absolute_error: 278.8911 -
val_loss: 182358.6875 - val_mean_absolute_error: 284.6890
Epoch 91/100
47/47 - 0s - 8ms/step - loss: 170282.7188 - mean_absolute_error: 279.2221 -
val_loss: 203646.3594 - val_mean_absolute_error: 310.0417
Epoch 92/100
47/47 - 0s - 7ms/step - loss: 168166.1562 - mean_absolute_error: 275.4760 -
val_loss: 188960.9062 - val_mean_absolute_error: 301.5909
Epoch 93/100
47/47 - 0s - 7ms/step - loss: 168954.9375 - mean_absolute_error: 278.8171 -
val_loss: 181588.3594 - val_mean_absolute_error: 289.5655
Epoch 94/100
47/47 - 0s - 8ms/step - loss: 167603.4219 - mean_absolute_error: 275.2266 -
val_loss: 184207.8125 - val_mean_absolute_error: 287.2204
Epoch 95/100
47/47 - 1s - 13ms/step - loss: 165960.8125 - mean_absolute_error: 273.0442 -
val_loss: 184720.9844 - val_mean_absolute_error: 293.8532
Epoch 96/100
47/47 - 0s - 4ms/step - loss: 165212.5781 - mean_absolute_error: 273.0289 -
val_loss: 183881.3594 - val_mean_absolute_error: 288.7343
Epoch 97/100
47/47 - 0s - 7ms/step - loss: 164902.4844 - mean_absolute_error: 273.3448 -
val_loss: 182512.1719 - val_mean_absolute_error: 284.1792
Epoch 98/100
47/47 - 0s - 6ms/step - loss: 167233.2969 - mean_absolute_error: 275.2500 -
val_loss: 181498.9062 - val_mean_absolute_error: 285.0037
Epoch 99/100
47/47 - 0s - 4ms/step - loss: 164845.9375 - mean_absolute_error: 271.7970 -
val_loss: 185239.7188 - val_mean_absolute_error: 289.9026
Epoch 100/100
47/47 - 0s - 6ms/step - loss: 165324.0312 - mean_absolute_error: 273.0680 -
val_loss: 182835.0312 - val_mean_absolute_error: 287.2046
```

3.3 Evaluate and report performance of the trained model

```
[189]: train_score = model.evaluate(arr_x_train, arr_y_train, verbose=0)
valid_score = model.evaluate(arr_x_valid, arr_y_valid, verbose=0)

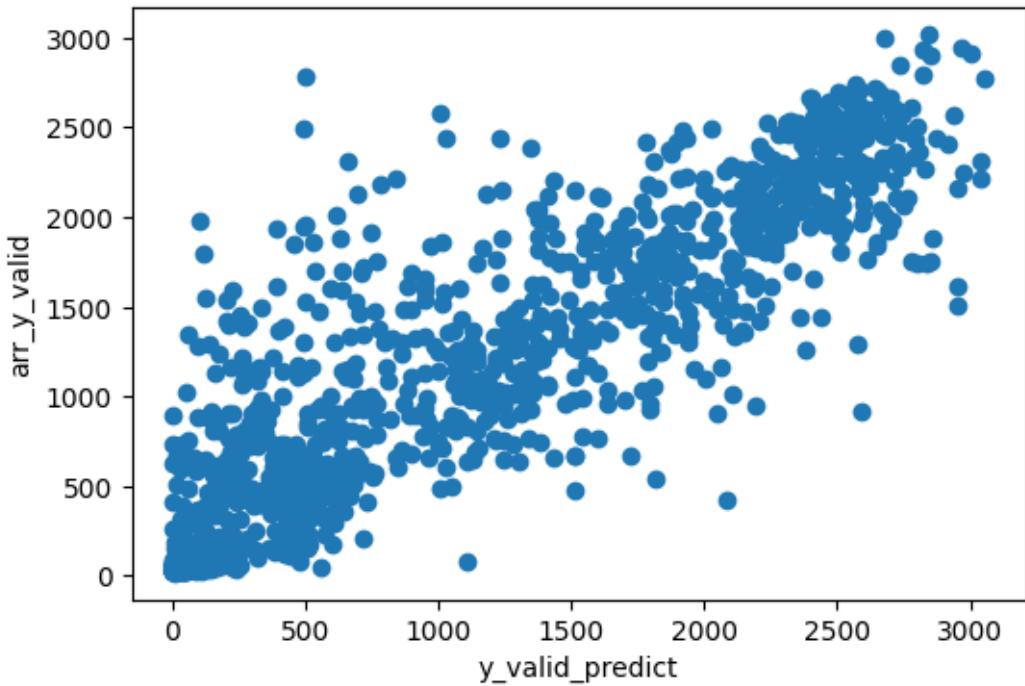
print('Train MAE: ', round(train_score[1], 2), ', Train Loss: ', round(train_score[0], 2))
print('Val MAE: ', round(valid_score[1], 2), ', Val Loss: ', round(valid_score[0], 2))
```

Train MAE: 266.88 , Train Loss: 160589.55
 Val MAE: 287.2 , Val Loss: 182835.03

```
[190]: y_valid_predict = model.predict(arr_x_valid)
plt.scatter(arr_y_valid, y_valid_predict)
plt.ylabel('arr_y_valid')
plt.xlabel('y_valid_predict')
plt.show()

corr_result = np.corrcoef(arr_y_valid.reshape(-1), y_valid_predict.reshape(-1))
print('The Correlation between true and predicted values is:', round(corr_result[0,1], 3))
```

40/40 1s 11ms/step



The Correlation between true and predicted values is: 0.889

```
[191]: def plot_hist(h, xsize=6, ysize=5):
    # Prepare plotting
    fig_size = plt.rcParams["figure.figsize"]
    plt.rcParams["figure.figsize"] = [xsize, ysize]

    # Get training and validation keys
    ks = list(h.keys())
    n2 = math.floor(len(ks)/2)
    train_keys = ks[0:n2]
    valid_keys = ks[n2:2*n2]

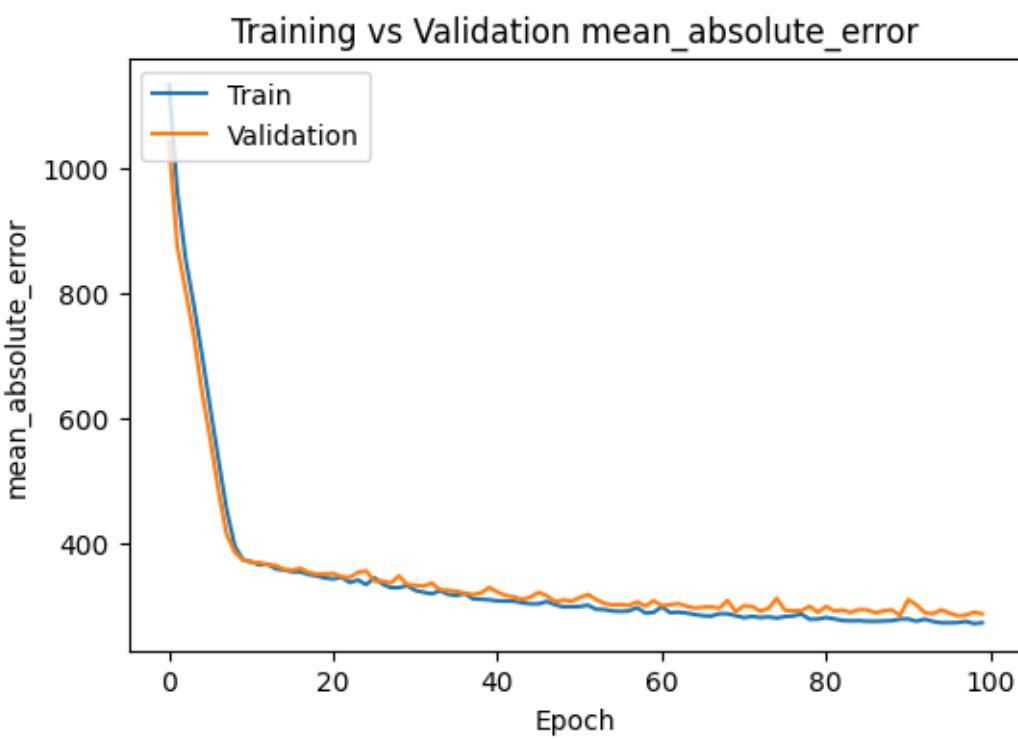
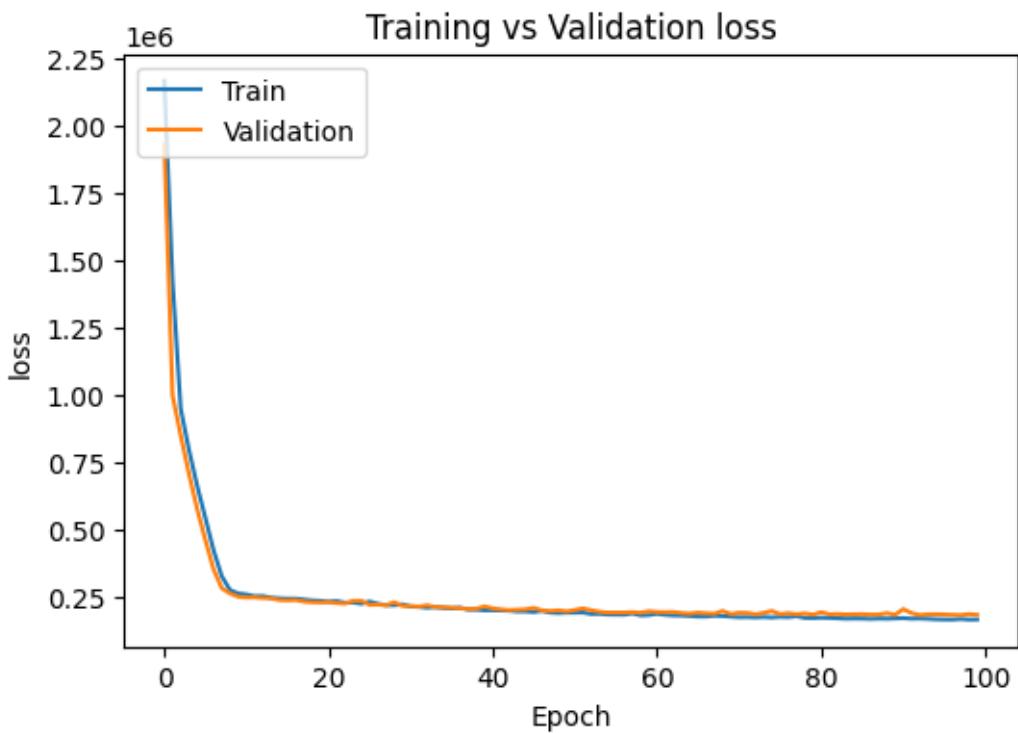
    # summarize history for different metrics
    for i in range(n2):
        plt.plot(h[train_keys[i]])
        plt.plot(h[valid_keys[i]])
        plt.title('Training vs Validation '+train_keys[i])
        plt.ylabel(train_keys[i])
        plt.xlabel('Epoch')
        plt.legend(['Train', 'Validation'], loc='upper left')
        plt.draw()
        plt.show()

    return
```

3.4 Ploting the true vs. predicted values.

```
[192]: hist = pd.DataFrame(history.history)

# Plot history
plot_hist(hist, xsize=6, ysize=4)
```



The scatter plot illustrates that the model exhibits a robust capacity for accurate predictions, effectively capturing the underlying relationship between the actual and predicted solar power values, characterized by a strong correlation. Although there is some variability observed at the extreme ranges, the bulk of the data points is densely clustered around the diagonal line, signifying a pronounced positive correlation between the predictions and actual measurements. Additionally, the training versus validation loss curve reveals rapid convergence, with stable and overlapping performance metrics across epochs. This indicates that the model is well-optimized, demonstrating effective generalization capabilities while avoiding overfitting.

```
[193]: train_score = model.evaluate(arr_x_train, arr_y_train, verbose=0)
valid_score = model.evaluate(arr_x_valid, arr_y_valid, verbose=0)

print('Train Accuracy: ', round(train_score[1], 2), ', Train Loss: ', round(train_score[0], 2))
print('Val Accuracy: ', round(valid_score[1], 2), ', Val Loss: ', round(valid_score[0], 2))
```

Train Accuracy: 266.88 , Train Loss: 160589.55
 Val Accuracy: 287.2 , Val Loss: 182835.03

```
[194]: from sklearn.metrics import mean_squared_error, mean_absolute_error
import numpy as np

# Assuming y_valid_predict contains the predictions from the Keras model
# and arr_y_valid contains the true validation values

rmse = np.sqrt(mean_squared_error(arr_y_valid, y_valid_predict))
mae = mean_absolute_error(arr_y_valid, y_valid_predict)

print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
```

Root Mean Squared Error (RMSE): 427.59
 Mean Absolute Error (MAE): 287.20

4. Experiments Report

1.2.4 4.1 Linear program

Heat map correlation

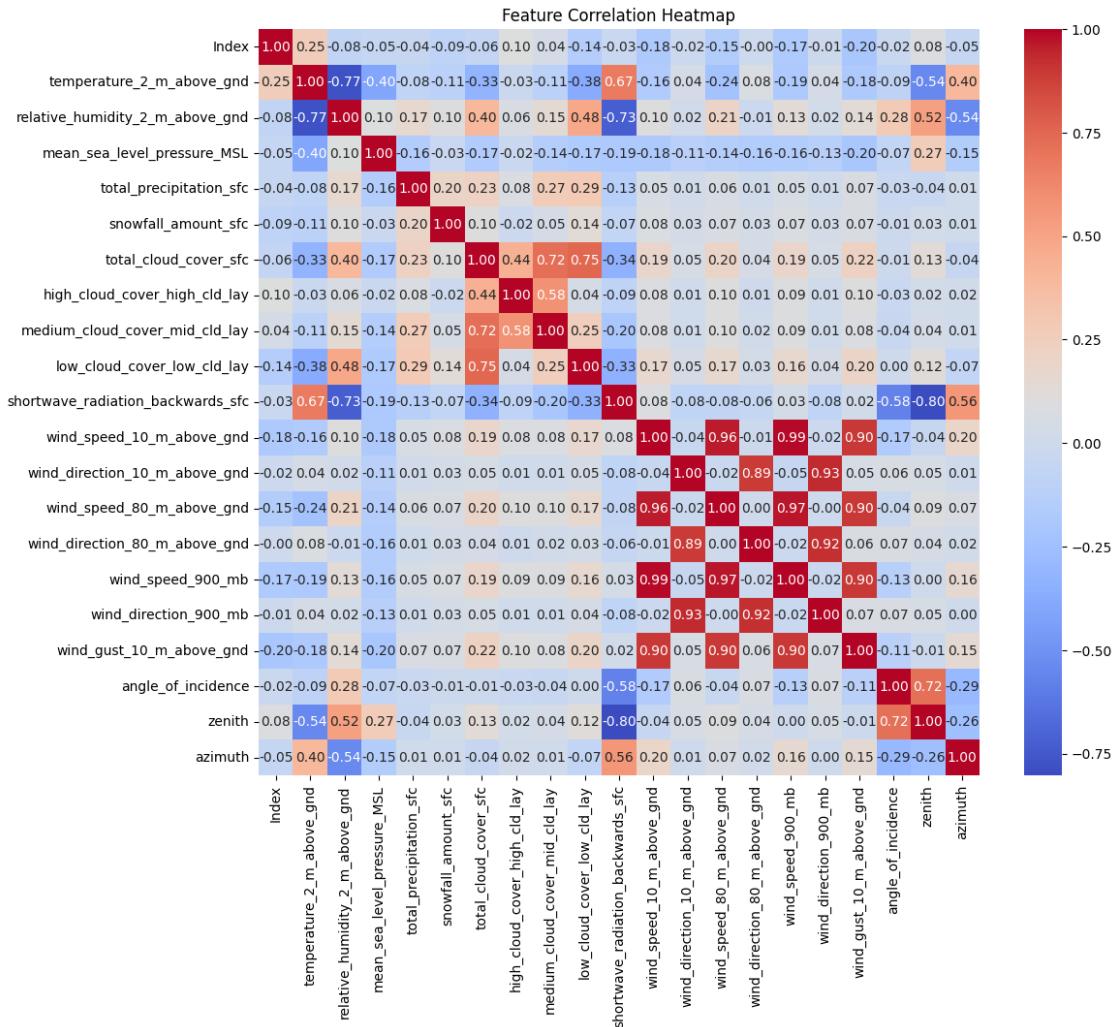
```
[195]: from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.feature_selection import SelectKBest, f_regression

# Create a Multiple Linear Regression model
```

```

linear_model = LinearRegression()
plt.figure(figsize=(12, 10))
correlation_matrix = solar_x_train.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Feature Correlation Heatmap')
plt.show()

```



Looking at the heatmap, ‘shortwave_radiation_backwards_sfc’, ‘angle_of_incidence’, and ‘zenith’ show relatively strong correlations with ‘generated_power_kw’. This makes intuitive sense, as these features directly relate to the amount of solar radiation reaching the solar panels.

‘Temperature_2_m_above_gnd’ and ‘relative_humidity_2_m_above_gnd’ can influence the efficiency of solar panels, even if their direct correlation on the heatmap might appear less strong than the radiation-related features.

‘Azimuth’ is also included as it represents the horizontal direction of the sun, which is relevant to the orientation of the solar panels and thus the amount of sunlight received.

These features were selected as a starting point for the linear regression model due to their potential to influence solar power generation, as indicated by the correlation analysis and domain knowledge. You can experiment with adding or removing features to see how it affects the model's performance.

4.2 Feature selection

```
[196]: selected_features = ['temperature_2_m_above_gnd',  
    ↪'relative_humidity_2_m_above_gnd', 'shortwave_radiation_backwards_sfc',  
    ↪'angle_of_incidence', 'zenith', 'azimuth']  
solar_x_train_selected = solar_x_train[selected_features]  
solar_x_valid_selected = solar_x_valid[selected_features]  
print(f"Selected Features: {list(selected_features)}")
```

```
Selected Features: ['temperature_2_m_above_gnd',  
'relative_humidity_2_m_above_gnd', 'shortwave_radiation_backwards_sfc',  
'angle_of_incidence', 'zenith', 'azimuth']
```

4.3 Train the model on selected features and make prediction

```
[201]: linear_model.fit(solar_x_train_selected, solar_y_train)  
y_valid_predict_linear = linear_model.predict(solar_x_valid_selected)
```

4.4 Evaluate the model

```
[198]: mae_linear = mean_absolute_error(solar_y_valid, y_valid_predict_linear)  
rmse_linear = np.sqrt(mean_squared_error(solar_y_valid, y_valid_predict_linear))  
corr_linear = np.corrcoef(solar_y_valid.values.reshape(-1),  
    ↪y_valid_predict_linear.reshape(-1))[0, 1]  
  
print(f"Multiple Linear Regression Model (with Feature Selection) MAE:  
    ↪{mae_linear:.2f}")  
print(f"Multiple Linear Regression Model (with Feature Selection) RMSE:  
    ↪{rmse_linear:.2f}")  
print(f"Multiple Linear Regression Model (with Feature Selection) Correlation  
    ↪Coefficient: {corr_linear:.2f}")
```

```
Multiple Linear Regression Model (with Feature Selection) MAE: 420.01  
Multiple Linear Regression Model (with Feature Selection) RMSE: 544.31  
Multiple Linear Regression Model (with Feature Selection) Correlation  
Coefficient: 0.81
```

4.5 Model Validation

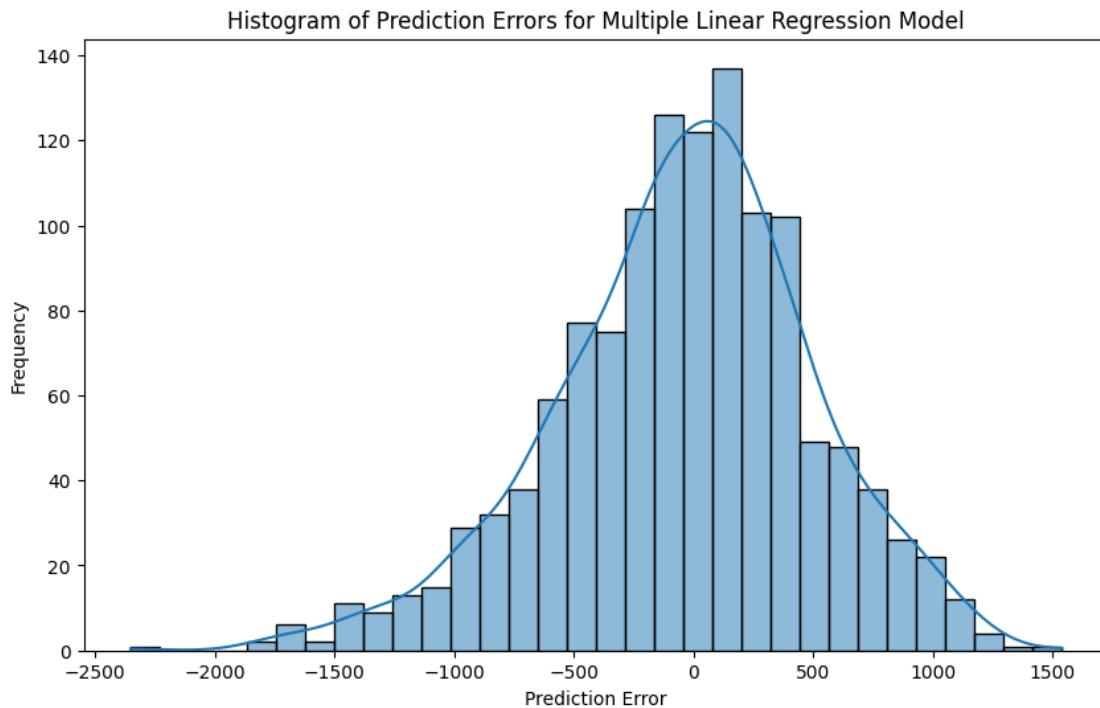
```
[199]: from sklearn.metrics import r2_score  
r2 = r2_score(solar_y_valid, y_valid_predict_linear)  
print(f" MultipleLinear Regression Model R-squared: {r2:.2f}")  
# Visualization of Prediction Histogram  
prediction_errors = solar_y_valid.values.flatten() - y_valid_predict_linear.  
    ↪flatten()
```

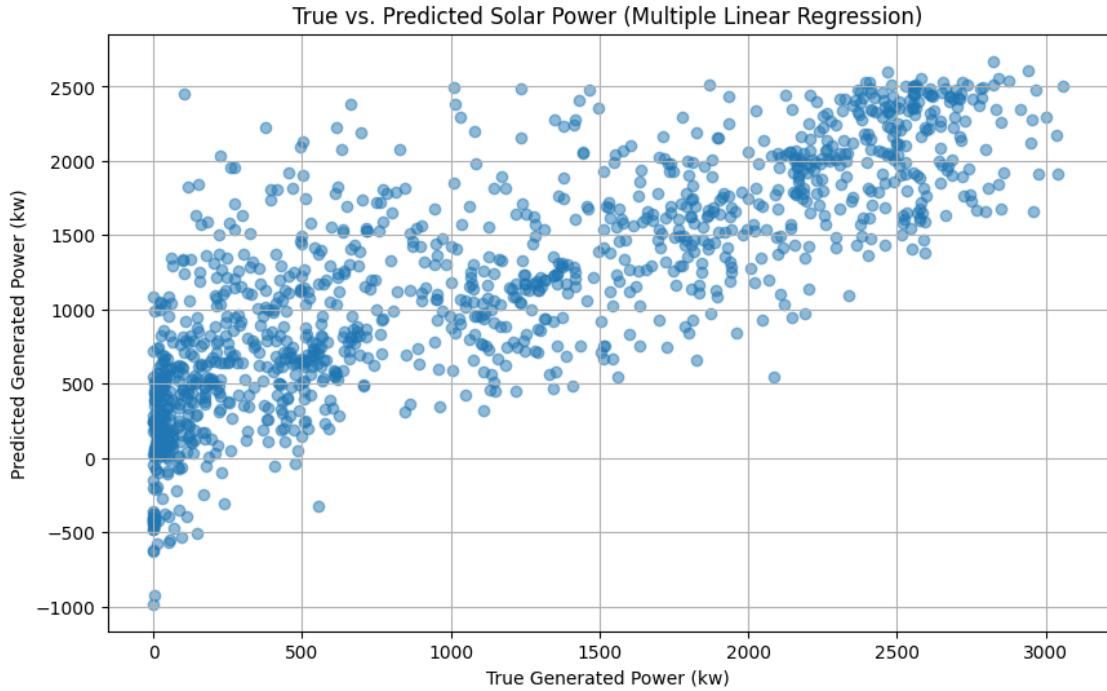
```

plt.figure(figsize=(10, 6))
sns.histplot(prediction_errors, kde=True)
plt.title('Histogram of Prediction Errors for Multiple Linear Regression Model')
plt.xlabel('Prediction Error')
plt.ylabel('Frequency')
plt.show()
# Training and Prediction Plot (True vs. Predicted)
plt.figure(figsize=(10, 6))
plt.scatter(solar_y_valid, y_valid_predict_linear, alpha=0.5)
plt.title('True vs. Predicted Solar Power (Multiple Linear Regression)')
plt.xlabel('True Generated Power (kw)')
plt.ylabel('Predicted Generated Power (kw)')
plt.grid(True)
plt.show()

```

MultipleLinear Regression Model R-squared: 0.66





1.2.5 4.6 Keras models for experiment purpose.

ANN 1 demonstrated moderate performance with a correlation coefficient of $R = 0.622$ and an RMSE of 745.89. Training vs. validation loss curves steadily decline, but plateau at relatively high values. Scatter plot shows a wide spread around the diagonal. The model's simplistic architecture, consisting of only two hidden layers with 64 units each, constrained its ability to capture the complex nonlinear relationships inherent in solar power data.

In contrast, ANN 2 exhibited a significant enhancement in performance, achieving $R = 0.875$ and an RMSE of 452.69. Loss and MAE curves drop sharply and stabilize with minimal gap between training and validation. Scatter plot points align closely with the diagonal, showing strong correlation. This improvement can be attributed to its deeper architecture, which integrates layers of 128 and 64 nodes, alongside the use of the RMSprop optimizer.

ANN 3, despite its increased depth, performed poorly, reflected in an RMSE of 935.93 and an unstable correlation. Loss and MAE curves are unstable, with validation error fluctuating heavily. Scatter plot shows predictions scattered far from the diagonal, indicating poor correlation. The choice of the Tanh activation function, combined with the use of the SGD optimizer, led to unstable training dynamics. Tanh is prone to vanishing gradient issues in deep networks, and the fixed learning rate of SGD hindered necessary adaptive adjustments, culminating in divergence and inadequate generalization.

Finally, ANN 4 recorded the lowest performance metrics, with an R value of -0.384 and an RMSE of 1444.49. Loss and MAE curves remain flat with little improvement across epochs. Scatter plot

is diffuse, with no strong alignment to the diagonal. The architecture, featuring only 32 and 16 nodes, coupled with Sigmoid activations, was insufficiently complex to adequately model the data's intricacies. The Sigmoid activation function further exacerbated this limitation by introducing saturation effects that slowed learning and contributed to underfitting.

ANN 5 achieved a performance with an R value of 0.865 and RMSE of 469.02. The training and validation loss/MAE curves converged smoothly, indicating minimal overfitting. Predictions in the scatter plot were reasonably aligned along the diagonal, albeit with more dispersion than top models. The architecture, featuring multiple 128-node layers and the Nadam optimizer, facilitated stable learning, though some redundancy likely increased error rates.

ANN 6 was the best model, with an R of 0.875 and RMSE of 454.74. Its balanced architecture with 64 and 32 hidden nodes, RMSprop optimization, and dropout regularization provided strong generalization and prevented overfitting. The training and validation curves nearly overlapped, demonstrating excellent stability, while the scatter plot revealed tight clustering of predictions around the diagonal.

ANN 7 performed slightly worse than ANN 6, registering an R of 0.857 and RMSE of 484.22. Over-parameterization with larger hidden layers (256 each) led to mild overfitting, despite effective Adam optimization. The training-validation curves showed a noticeable gap, indicating some overfitting, and the scatter plot exhibited wider dispersion of predictions.

ANN 8 underperformed, with an R of 0.858 and RMSE of 480.19, primarily due to underfitting. Its shallow architecture (50 hidden nodes) with Tanh activation and SGD optimization limited its flexibility in capturing nonlinearities. The training and validation curves converged slowly, with validation error consistently higher, and predictions were loosely scattered in the scatter plot.

ANN 9 exhibited an R value of 0.858 and an RMSE of 480.19. The training and validation loss/MAE curves demonstrated steady convergence; however, they plateaued at higher error levels relative to the leading models, with a slight gap indicative of mild overfitting. The scatter plot of predictions revealed alignment with the diagonal, albeit with notable dispersion. The architecture featured multiple hidden layers configured as (100–50–25–16) and used Sigmoid activations, which imposed saturation effects that limited learning efficiency, resulting in suboptimal convergence and accuracy compared to the best-performing model.

In contrast, **ANN 10** significantly underperformed, achieving an R of 0.771 and an RMSE of 604.61. The training and validation curves exhibited a slower decline and plateaued at elevated error levels, reflecting a lack of effective learning. The scatter plot indicated poor correlation with actual values, showing predictions widely dispersed from the diagonal. Despite a more complex architecture of (200–100–50 nodes), the combination of a dropout rate of 0.4 and the selected optimizer constrained its learning capacity, adversely affecting predictive performance.

ANN 11 emerged as the top-performing model, realizing an R of 0.887 and an RMSE of 431.14. The training and validation curves closely overlapped, signaling smooth convergence with excellent stability and no evidence of overfitting. Predictions were tightly clustered along the diagonal in the scatter plot, demonstrating a strong correlation with actual values. Its balanced architecture of

(128–96–64–32 nodes), paired with ReLU activations and Adam optimization, provided sufficient depth and flexibility, allowing for superior generalization capabilities compared to both ANN 9 and ANN 10.

ANN 12 achieved strong performance metrics with an R of 0.888 and an RMSE of 438.3. The training and validation loss curves converged smoothly, indicating excellent stability, with predictions clustering along the diagonal in the scatter plot, albeit more dispersed than ANN 11. Its architecture (64-32 nodes), Tanh activations, Nadam optimization, and a dropout rate of 0.5 effectively prevented overfitting and enabled good generalization.

In contrast, **ANN 13** showed underperformance with an R of 0.541 and an RMSE of 795.53. The convergence of the training and validation curves was slow, and the scatter plot revealed poor clustering, indicating weak correlation. Despite a similar architecture to ANN 12, a small learning rate (0.0001) and Tanh activations limited training efficiency, leading to underfitting.

ANN 14 was a top performer, with an R of 0.889 and an RMSE of 445.57. Its closely overlapping training and validation loss curves reflected smooth convergence without overfitting, and predictions clustered tightly along the diagonal. With five hidden layers of 100 nodes, ReLU activations, and an RMSprop optimizer, it effectively captured complex nonlinear relationships.

ANN 15 performed the worst, with an R of -0.166 and an RMSE of 936.31. The unstable training and validation curves showed heavy fluctuations, and the scatter plot indicated minimal alignment with the diagonal. Its shallow 50-node architecture with Sigmoid activations and an SGD optimizer failed to capture nonlinear patterns, resulting in poor accuracy and negative correlation.

Optimal Models: ANN 11 and ANN 12

The results indicate that ANN 11 and ANN 12 are the optimal models for solar power prediction, achieving the lowest RMSE and MAE among the evaluated architectures. Specifically, ANN 11 recorded an RMSE of 431.14, MAE of 290.42, and a correlation coefficient (R) of 0.887, while ANN 12 exhibited an RMSE of 438.3, MAE of 294.07, and an R value of 0.888. These metrics indicate a strong predictive accuracy and good alignment between predicted and actual solar power outputs.

Both models demonstrate smooth and rapid convergence in training versus validation loss curves, with no significant overfitting and steadily decreasing loss values. The training versus validation MAE plots show consistent performance across both datasets.

Furthermore, the scatter plots of predicted versus actual values exhibit a dense distribution of data points around the line of perfect prediction, confirming the models' ability to accurately capture the nonlinear relationships between weather variables, solar geometry, and power output.

In summary, the stable convergence patterns and close alignment in the scatter plots confirm that ANN 11 and ANN 12 are reliable models suitable for real-world solar forecasting applications, where accuracy and consistency are critical for effective energy planning and management.

1.3 5. Model implication in real world

Predicting solar photovoltaic (PV) output is inherently challenging due to the variability of environmental factors like irradiance, temperature, and humidity. Recent studies highlight the advantages of artificial neural networks (ANNs) over traditional regression models in capturing these complex, nonlinear dynamics.

Hanis Nasuha Amer et al. (2023) demonstrated that an ANN optimized through feature selection can effectively forecast solar output for large-scale PV systems, focusing on critical predictors such as irradiance and solar geometry. This approach not only improves prediction accuracy but also reduces computational complexity.

Keddouda et al. (2023) further reaffirmed this by comparing ANN and multiple regression models across diverse PV operational conditions. While regression models provide interpretability, they often fail to encapsulate the intricate interactions among environmental variables. In contrast, ANNs exhibited superior predictive performance, aligning more closely with actual output and demonstrating lower error rates.

These findings underline the potential for ANN-based forecasting to optimize scheduling, enhance integration of solar energy into the grid, and reduce reliance on fossil fuels, ultimately leading to lower operational costs and improved sustainability.

However, successful real-world application necessitates robust data pipelines, effective feature selection, and regular retraining to adapt to changes over time. Overall, the literature clearly supports the adoption of ANNs as essential tools for solar output prediction and sustainable energy management (Hanis Nasuha Amer et al., 2023; Keddouda et al., 2023).

1.4 6. Reference

Hanis Nasuha Amer et al. (2023) ‘Solar power prediction based on Artificial Neural Network guided by feature selection for Large-scale Solar Photovoltaic Plant’, Energy Reports, 9(262–266), pp. 262–266. doi:10.1016/j.egyr.2023.09.141.

Keddouda, A. et al. (2023) ‘Solar photovoltaic power prediction using artificial neural network and multiple regression considering ambient and operating conditions’, Energy Conversion and Management, 288. doi:10.1016/j.enconman.2023.117186.

1.5 7. Role of GenAI

1.6 7. Role of Generative AI

Generative AI (GenAI) tools played a significant and multifaceted role as collaborative partners throughout the process of developing a model to forecast PM2.5 concentrations. Specifically, Google’s Gemini, integrated within the Google Colab environment, served as the primary GenAI tool. Its main function was to act as an intelligent assistant capable of interpreting code, diagnosing errors, suggesting solutions, and offering alternative approaches to complex programming and modeling challenges. In addition to Gemini, other general-purpose GenAI tools, such as large language models available via web interfaces, were utilized for broader conceptual understanding, reviewing documentation, and generating initial drafts of explanatory text for the report.

GenAI proved valuable as a collaborative partner in numerous instances. During the data pre-processing phase, it assisted in troubleshooting issues related to data loading, format conversion,

and handling missing values by providing potential causes for error messages and suggesting code snippets for corrections or more efficient implementations. In the predictive modeling stage, GenAI contributed by offering suggestions for modifying model architecture, such as adding dropout layers or alternative activation functions, and proposing hyperparameter values or ranges to explore during the tuning process. It also aided in understanding how different regularization techniques and optimizers affect model performance. This collaboration significantly accelerated the iterative development cycle, allowing for quicker identification and resolution of technical challenges and exploration of various modeling strategies.

However, the integration of GenAI was not without challenges, leading to critical reflection on its limitations and the need for human oversight. One key limitation encountered was the occasional generation of complex solutions that, while technically correct, were beyond the required scope or level of detail for the assessment. This required careful evaluation, simplification, and adaptation of the generated code or explanations to meet the project's specific needs and foundational concepts. Furthermore, relying solely on GenAI without a deep understanding of the underlying principles could result in the application of suboptimal or incorrectly implemented techniques. Ethical considerations, such as ensuring the originality of generated text and critically verifying the accuracy of technical suggestions, were paramount. This process underscored the importance of using GenAI not as a replacement for foundational knowledge or critical thinking, but as a supplemental tool that requires expert human guidance to validate outputs, ensure relevance, and maintain academic integrity. This experience highlights that while GenAI can significantly enhance productivity and problem-solving in model development, a solid understanding of the technical domain and strong evaluation skills are essential.