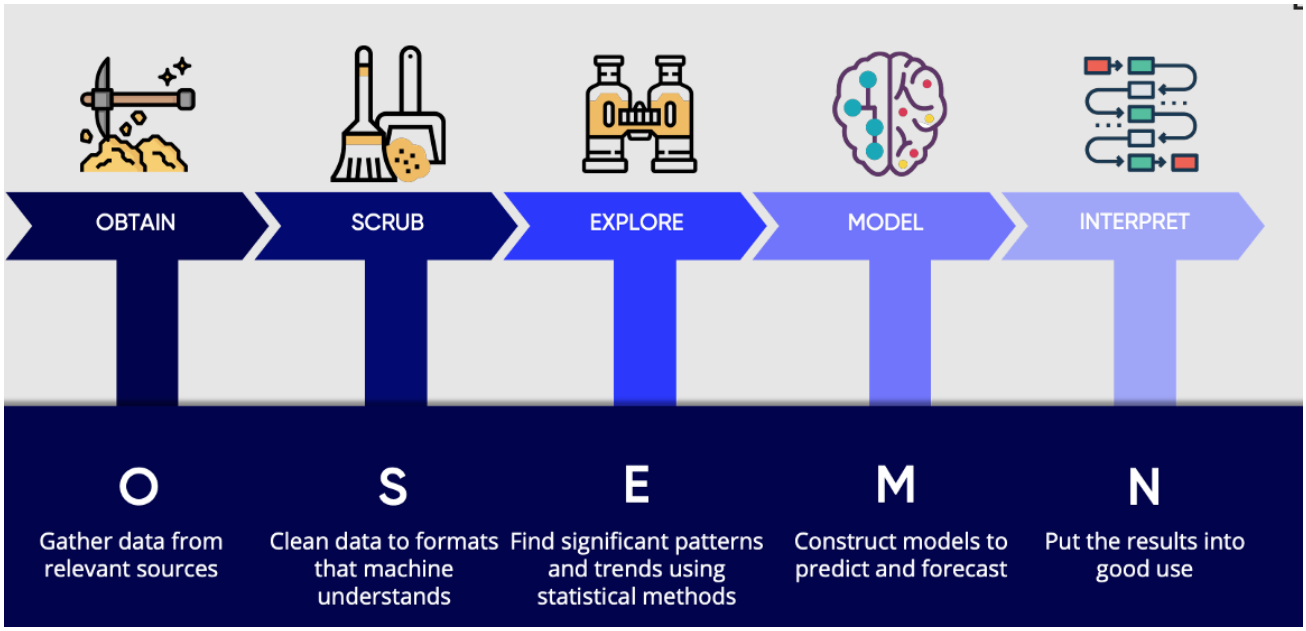
Time Series Data Analysis Workshop



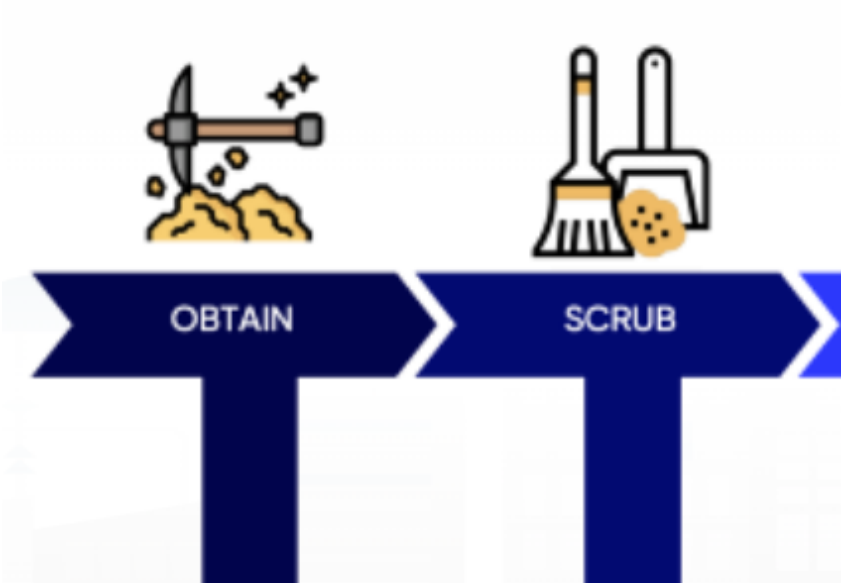
import section of required libraries

In [1]:
```
import os
# Set CUDA_VISIBLE_DEVICES to an empty string (disabling GPU)
os.environ['CUDA_VISIBLE_DEVICES'] = ''
import pandas as pd #dataset management
import matplotlib.pyplot as plt #plot visualization
import seaborn as sns #plot visualization
from sklearn.model_selection import train_test_split # dataset handling
from sklearn.preprocessing import StandardScaler # to scale the data for the AI model
import tensorflow as tf # tensorflow for simple DNN dev
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import mean_absolute_error #evaluate resuts
```

```
2023-11-28 03:02:29.037795: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library libcudart.so.10.1
```

Assess the impact of a single event

Analysis of NO2 Levels in 2020 for the city of Florence



EU air quality standards https://environment.ec.europa.eu/topics/air/air-quality/eu-air-quality-standards_en

Pollutant	Concentration	Averaging period	Legal nature
Fine particles ($PM_{2.5}$)	25 µg/m^3	1 year	Target value to be met as of 1.1.2010 Limit value to be met as of 1.1.2015
Fine particles ($PM_{2.5}$)	20 µg/m^3	1 year	Stage 2 limit value to be met as of 1.1.2020 ***
Sulphur dioxide (SO_2)	350 µg/m^3	1 hour	Limit value to be met as of 1.1.2005
Sulphur dioxide (SO_2)	125 µg/m^3	24 hours	Limit value to be met as of 1.1.2005
Nitrogen dioxide (NO_2)	200 µg/m^3	1 hour	Limit value to be met as of 1.1.2010
Nitrogen dioxide (NO_2)	40 µg/m^3	1 year	Limit value to be met as of 1.1.2010 *

Introduction

In this analysis, we aim to explore and understand the average annual NO2 levels for the year 2020. The dataset used for this analysis is located in "./datasets/datasetNO22020.csv". NO2, or nitrogen dioxide, is a common air pollutant that can have various environmental and health impacts. Examining its levels over time can provide insights into air quality trends.

Dataset Description

The dataset contains multivariate information related to the NO2 concentration in Florence for the year 2020. Each row represents a specific measurement, and the columns include the attributes of the dataset. In detail the dataset has daily granularity

Methodology

To determine the average annual NO2 level for 2020, we will perform the following steps using Python and relevant libraries:

1. Load the dataset into a Pandas DataFrame.
2. Calculate the average NO2 concentration
3. Check the validity graphically visualizing the progressive mean no2 for the year 2020

In [2]:
```
# 1 Load the dataset
dataset_path_no2 = "./datasets/datasetNO22020.csv"
df_no2 = pd.read_csv(dataset_path_no2)
df_no2.head()
```

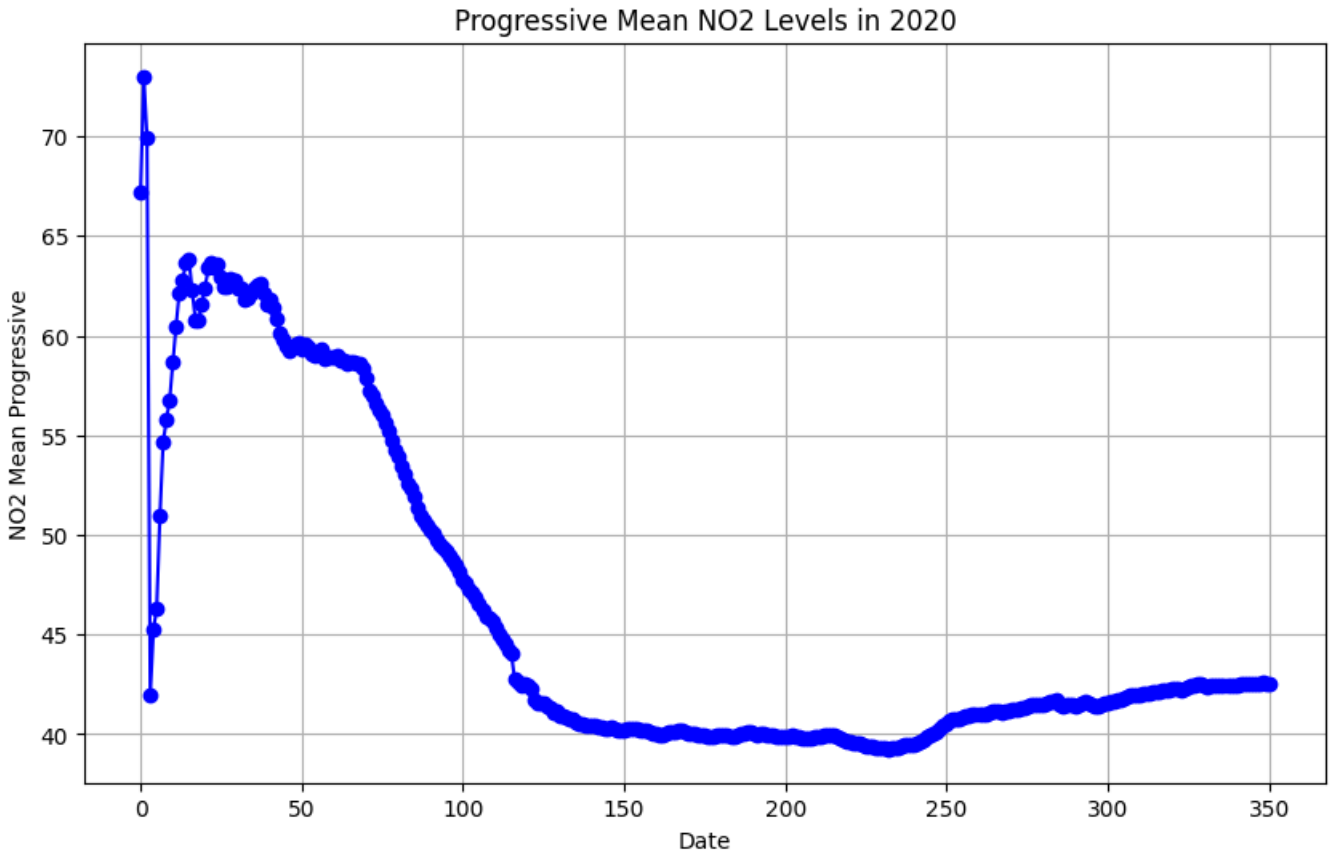
Out[2]:

	data	NO2	no2cumulato	no2medioprogressivo
0	2020-01-01	67.181818	67.181818	67.181818
1	2020-01-02	78.818182	146.000000	73.000000
2	2020-01-03	63.800000	209.800000	69.933333
3	2020-01-06	42.000000	251.800000	41.966667
4	2020-01-07	64.857143	316.657143	45.236735

In [3]:
```
# 2 mean NO2 concentration
mean_NO2_2020 = df_no2['NO2'].mean()
print("Average NO2 Concentration:", mean_NO2_2020)
```

```
Average NO2 Concentration: 43.51029992714151
```

In [4]:
```
# 3 Plotting the cumulative average NO2 values
plt.figure(figsize=(10, 6))
plt.plot(df_no2['no2medioprogressivo'], marker='o', linestyle='-', color='b')
plt.title('Progressive Mean NO2 Levels in 2020')
plt.xlabel('Date')
plt.ylabel('NO2 Mean Progressive')
plt.grid(True)
plt.show()
```



Study the interaction between a set of values

Analysis of Data from Pont du Gard

Introduction

In this analysis, we aim to explore and understand the patterns in data collected from Pont du Gard. The dataset used for this analysis is located in "./datasets/pdg.csv". This dataset encompasses various features, including environmental factors, bike and people counts, weather conditions, and social media activity around Pont du Gard. The goal is to identify potential correlations and gain insights into the dynamics of this location.

Dataset Description

The dataset includes the following key attributes:

- `dateObserved` : Date of observation
- `Hour` : Hour of the day
- `DayofTheWeek` : Day of the week
- `isWeekEndDay` : Indicator for weekend day
- `Month` : Month of the year
- `isHoliday` : Indicator for holiday
- Bike counts (BikeCounted_1 to BikeCounted_10)
- People counts (peopleCounted_1 to peopleCounted_14)
- Environmental factors (temp, humidity, precip, pressure, windspeed)
- Social media activity (tweets_volume, tweets_neg, tweets_neu, tweets_pos)

The dataset has daily granularity, providing a comprehensive view of various factors around Pont du Gard.

Methodology

1. Load the dataset into a Pandas DataFrame.
2. Check eventual missing values
3. Correlation analysis of the variables
4. Dataset dimensionality reduction based on the found insights

In [5]:
```
# 1 Load the dataset
dataset_path_pdg = pd.read_csv("./datasets/pdg.csv")
```

```
dataset_path_pdg.columns
```

Out[5]:
```
Index(['dateObserved', 'Hour', 'DayofTheWeek', 'isWeekEndDay', 'Month',
       'isHoliday', 'BikeCounted_1', 'BikeCounted_2', 'BikeCounted_3',
       'BikeCounted_4', 'BikeCounted_5', 'BikeCounted_6', 'BikeCounted_7',
       'BikeCounted_8', 'BikeCounted_9', 'BikeCounted_10', 'peopleCounted_1',
       'peopleCounted_2', 'peopleCounted_3', 'peopleCounted_4',
       'peopleCounted_5', 'peopleCounted_6', 'peopleCounted_7',
       'peopleCounted_8', 'peopleCounted_9', 'peopleCounted_10',
       'peopleCounted_11', 'peopleCounted_12', 'peopleCounted_13',
       'peopleCounted_14', 'temp', 'humidity', 'precip', 'pressure',
       'windspeed', 'tweets_volume', 'tweets_neg', 'tweets_neu', 'tweets_pos'],
      dtype='object')
```

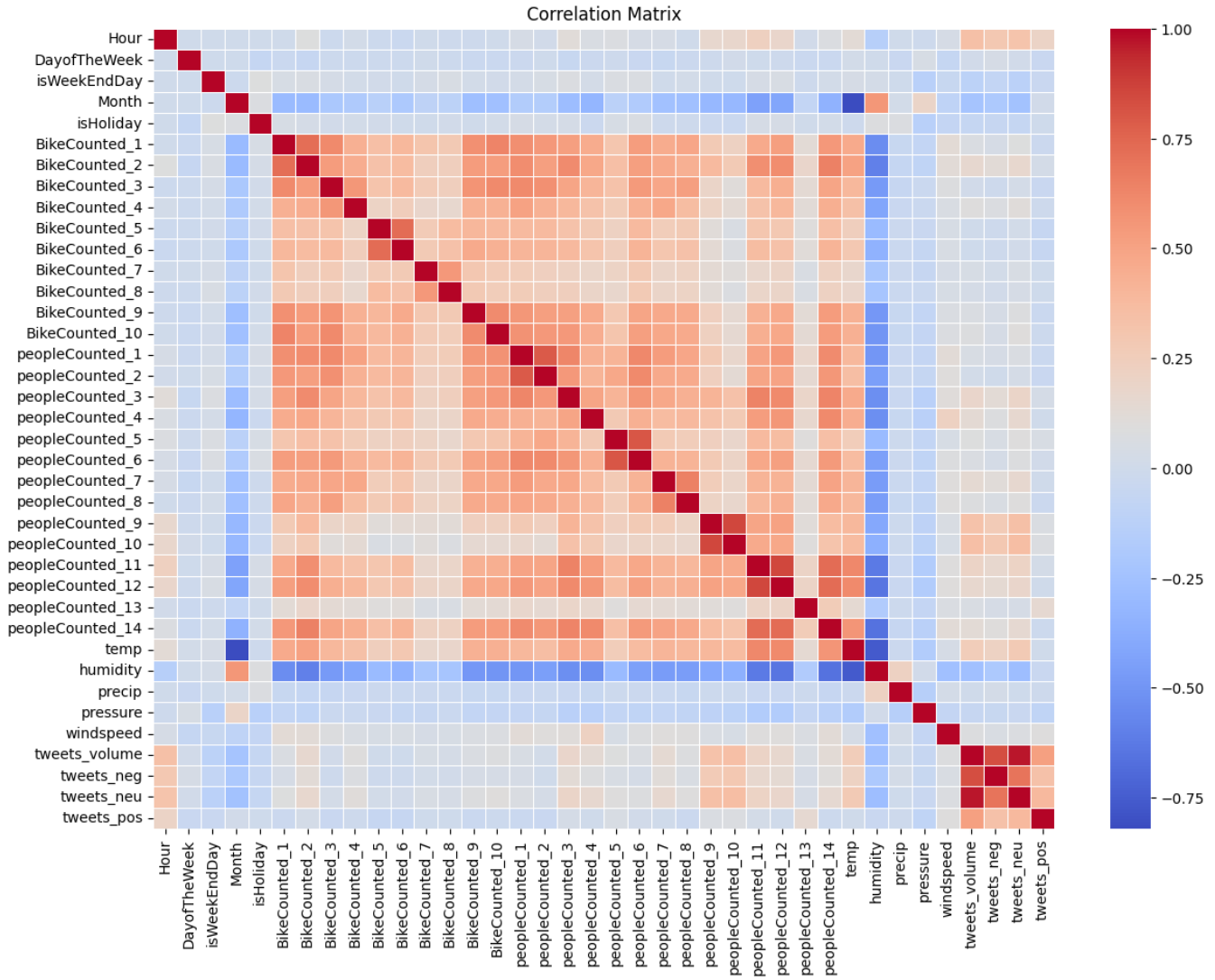
In [6]:
```
# 2 Check for null values
null_values = dataset_path_pdg.isnull().sum()
print(null_values)
```

```
dateObserved        0
Hour                0
DayofTheWeek        0
isWeekEndDay        0
Month               0
isHoliday           0
BikeCounted_1       0
BikeCounted_2       0
BikeCounted_3       0
BikeCounted_4       0
BikeCounted_5       0
BikeCounted_6       0
BikeCounted_7       0
BikeCounted_8       0
BikeCounted_9       0
BikeCounted_10      0
peopleCounted_1     0
peopleCounted_2     0
peopleCounted_3     0
peopleCounted_4     0
peopleCounted_5     0
peopleCounted_6     0
peopleCounted_7     0
peopleCounted_8     0
peopleCounted_9     0
peopleCounted_10    0
peopleCounted_11    0
peopleCounted_12    0
peopleCounted_13    0
peopleCounted_14    0
temp                0
humidity            0
precip              0
pressure            0
windspeed           0
tweets_volume       0
tweets_neg          0
tweets_neu          0
tweets_pos          0
dtype: int64
```

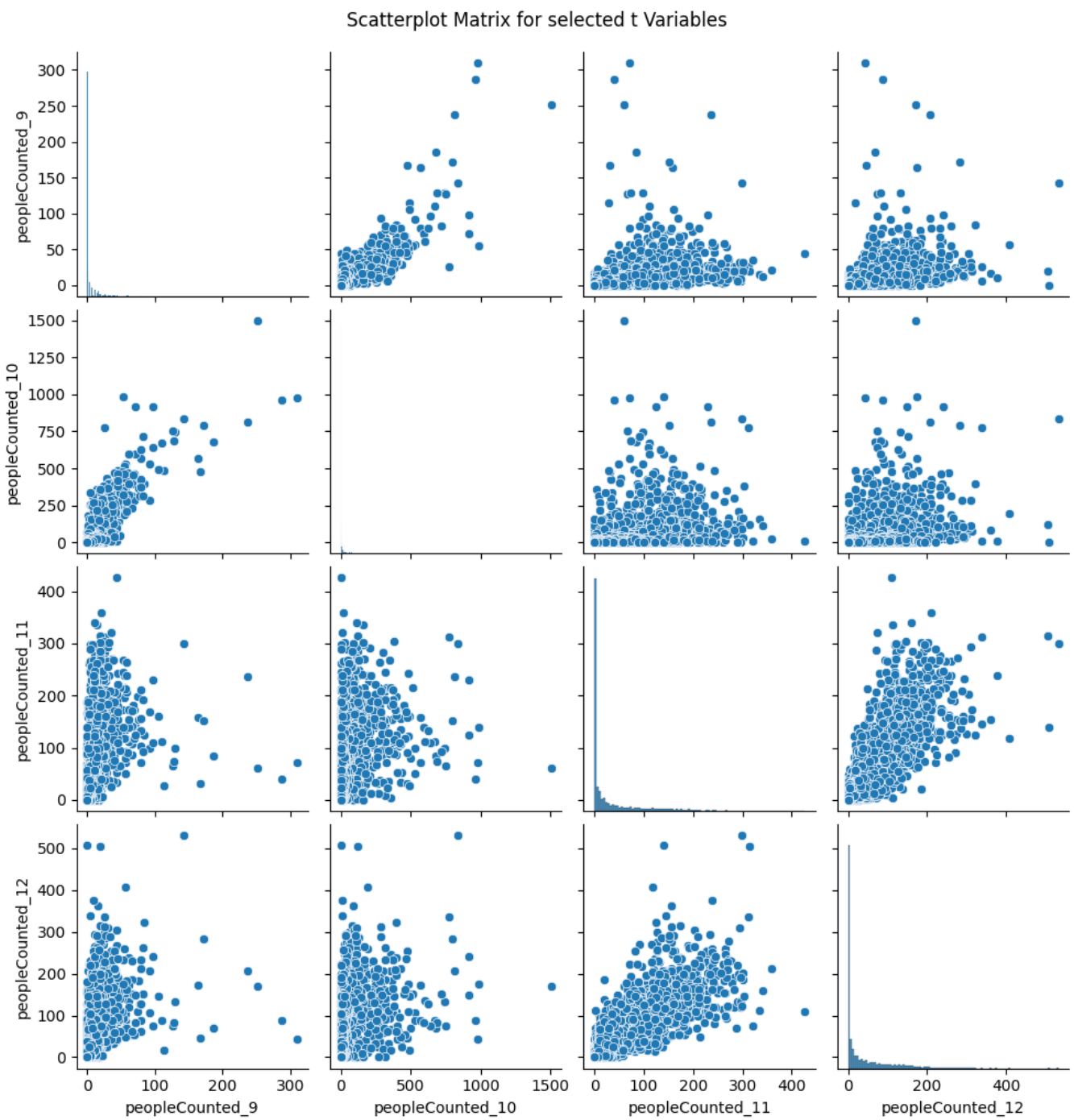
In [7]:
```
# 3 Correlation Analysis


# Calculate correlation matrix
correlation_matrix = dataset_path_pdg.corr()

# Display correlation matrix
#print(correlation_matrix)
```

```
# Set up the matplotlib figure
plt.figure(figsize=(14, 10))

# Create a heatmap of the correlation matrix
sns.heatmap(correlation_matrix, annot=False, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title('Correlation Matrix')
plt.show()
```

```
/tmp/ipykernel_3871661/4210571118.py:5: FutureWarning: The default value of numeric_only in Da
taFrame.corr is deprecated. In a future version, it will default to False. Select only valid c
olumns or specify the value of numeric_only to silence this warning.
  correlation_matrix = dataset_path_pdg.corr()
```



In [8]:
```
# Extract relevant columns for bike counts
selected_columns = ['peopleCounted_9', 'peopleCounted_10', 'peopleCounted_11', 'peopleCounted_1
selected_data = dataset_path_pdg[selected_columns]

# Pairplot for bike count variables
sns.pairplot(selected_data)
plt.suptitle('Scatterplot Matrix for selected t Variables', y=1.02)
plt.show()
```



In [9]:
```
#check correlation values of the selected columns
type(correlation_matrix)
correlation_matrix[selected_columns].loc[selected_columns,:]
```

Out[9]:

	peopleCounted_9	peopleCounted_10	peopleCounted_11	peopleCounted_12
peopleCounted_9	1.000000	0.852024	0.493807	0.505087
peopleCounted_10	0.852024	1.000000	0.461711	0.479494
peopleCounted_11	0.493807	0.461711	1.000000	0.853220
peopleCounted_12	0.505087	0.479494	0.853220	1.000000

In [10]:
```
# 4 Dataset dimensionality reduction based on the found insights... to be completed
```

Forecast Future Values of a Time-Series using the previous values of one series (or also values from others)

Analysis of data from a bike rack in Siena

Introduction

In this analysis, we aim to forecast future values of a time-series using historical data from a bike rack in Siena. The dataset used for this analysis is located in "./datasets/bike_rack_siena.csv". Time-series forecasting is a crucial aspect of understanding trends and making predictions based on historical patterns. We will explore how to forecast the future availability of bikes at the Siena bike rack using previous values and environmental factors.

Dataset Description

The dataset has 15' time granularity and includes the following key attributes:

- `Date` : Date of observation
- `Time` : Time of observation
- `anno` : Year
- `mese` : Month
- `dayOfTheYear` : Day of the year
- `dayOfTheMonth` : Day of the month
- `dayOfTheWeek` : Day of the week
- `weekend` : Indicator for weekend
- `DateTime` : Combined date and time
- `freeStalls` : Available bike stalls
- `brokenBikes` : Number of broken bikes
- `availableBikes` : Number of available bikes
- `DateTime_1` : Combined date and time (another format)
- `maxTemp` : Maximum temperature
- `minTemp` : Minimum temperature
- `Temperature` : Temperature
- `Humidity` : Humidity
- `Pressure` : Atmospheric pressure
- `WindSpeed` : Wind speed
- `CloudCoverPerc` : Cloud cover percentage
- `rain` : Rainfall indicator

The dataset provides a detailed view of bike rack activity and environmental conditions.

Methodology

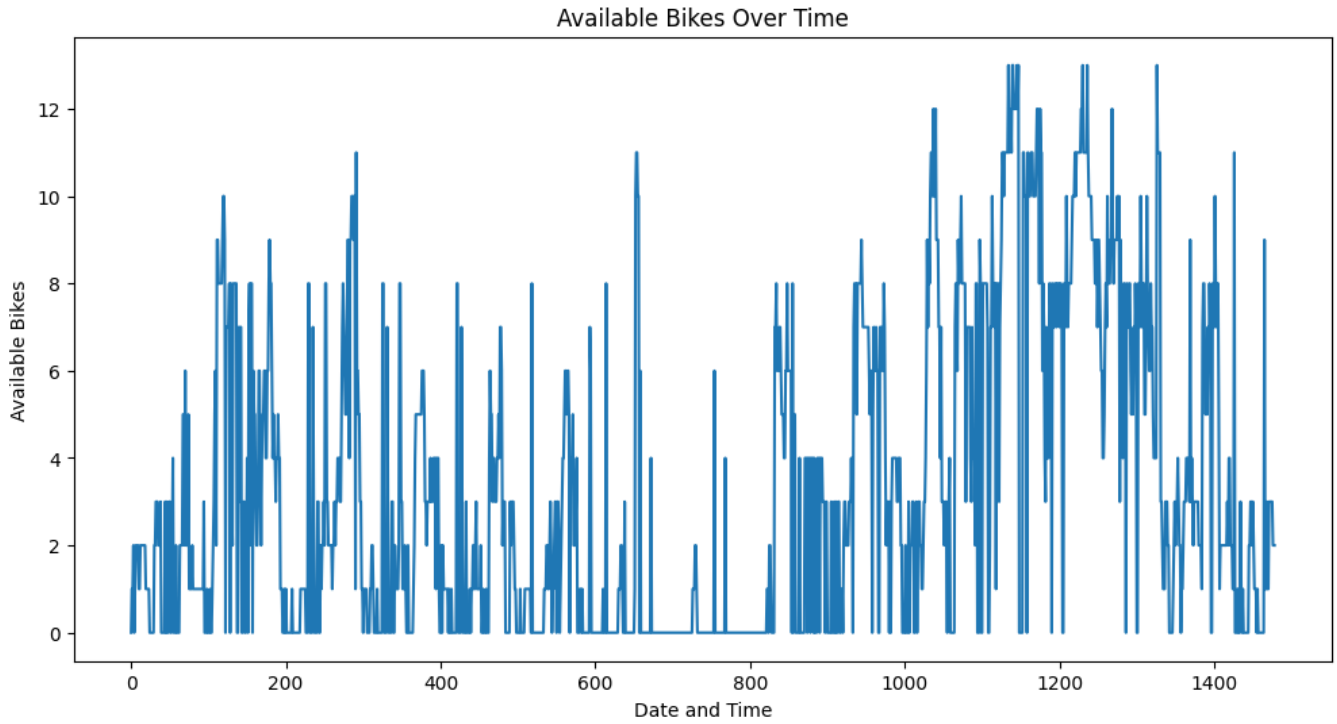
1. Load the dataset into a Pandas DataFrame.
2. Scrub the data at least the null values
3. Explore the data minimum plot the availableBikes and divide the dataset in training and testing 80/20
4. Train a simple time-series model (DNN 3 layer) to forecast the availaleBikes of the next day
5. Evaluate the model results Mean Absolute Error and visualization plot

In [11]:
```
# 1 Load the dataset
dataset_path_bike_rack = pd.read_csv("./datasets/bike_rack_siena.csv")
```

In [12]:
```
# 2 Scrub the data!
# Check for null values
null_values = dataset_path_bike_rack.isnull().sum()
print(null_values)
#dropped rain
dataset_path_bike_rack = dataset_path_bike_rack.drop(columns="rain")
```

```
Date                0
Time                0
anno                0
mese                0
dayOfTheYear        0
dayOfTheMonth       0
dayOfTheWeek        0
weekend             0
DateTime            0
freeStalls          0
brokenBikes         0
availableBikes      0
DateTime_1          0
maxTemp             0
minTemp             0
Temperature         0
Humidity            0
Pressure            0
WindSpeed           0
CloudCoverPerc      0
rain              317
dP                  0
dS                  0
PwAB                0
dtype: int64
```

In [13]:
```
# 3. Explore the data, plot the availableBikes, and divide the dataset into training and testi

# Divide the dataset into training and testing (80/20 split)
train_size = int(len(dataset_path_bike_rack) * 0.8)
train_data, test_data = dataset_path_bike_rack.iloc[:train_size], dataset_path_bike_rack.iloc[
train_data = train_data.reset_index()
test_data = test_data.reset_index()
# Plot availableBikes of the test set
plt.figure(figsize=(12, 6))
plt.plot(test_data['availableBikes'])
plt.title('Available Bikes Over Time')
plt.xlabel('Date and Time')
plt.ylabel('Available Bikes')
plt.show()
```



In [14]:
```
# set temporal windows of 96 timestep (1 day in advance)

# Feature selection (considering only 'availableBikes' and 'Temperature' for simplicity)
selected_features = ['availableBikes', 'Temperature']
train_features = train_data.loc[:len(train_data)-97, selected_features].values
test_features = test_data.loc[:len(test_data)-97, selected_features].values

# Standardize features space
scaler = StandardScaler()
train_features_scaled = scaler.fit_transform(train_features)
test_features_scaled = scaler.transform(test_features)

# Target variable
train_target = train_data.loc[96:,'availableBikes'].values
test_target = test_data.loc[96:,'availableBikes'].values
```

In [15]:
```
# Build the DNN model
model = Sequential()
model.add(Dense(100, activation='relu', input_dim=train_features_scaled.shape[1]))
model.add(Dense(50, activation='relu'))
model.add(Dense(25, activation='relu'))
model.add(Dense(1))  # Output layer
```

```
2023-11-28 03:02:34.427092: I tensorflow/compiler/jit/xla_cpu_device.cc:41] Not creating XLA d
evices, tf_xla_enable_xla_devices not set
2023-11-28 03:02:34.428434: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Su
ccessfully opened dynamic library libcuda.so.1
2023-11-28 03:02:34.456780: E tensorflow/stream_executor/cuda/cuda_driver.cc:328] failed call
to cuInit: CUDA_ERROR_NO_DEVICE: no CUDA-capable device is detected
2023-11-28 03:02:34.456831: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:169] retriev
ing CUDA diagnostic information for host: ubuntu-Precision-5820-Tower
2023-11-28 03:02:34.456843: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:176] hostnam
e: ubuntu-Precision-5820-Tower
2023-11-28 03:02:34.457006: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:200] libcuda
reported version is: 515.43.4
2023-11-28 03:02:34.457033: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:204] kernel
reported version is: 515.43.4
2023-11-28 03:02:34.457039: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:310] kernel
version seems to match DSO: 515.43.4
2023-11-28 03:02:34.457884: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFl
ow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following C
PU instructions in performance-critical operations:  SSE4.1 SSE4.2 AVX AVX2 AVX512F FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2023-11-28 03:02:34.458600: I tensorflow/compiler/jit/xla_gpu_device.cc:99] Not creating XLA d
evices, tf_xla_enable_xla_devices not set
```

In [16]:
```
# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')  # Mean Squared Error loss for regr
```

In [17]:
```
# Train the model
model.fit(train_features_scaled, train_target, epochs=50, batch_size=32, verbose=1)
```

```
2023-11-28 03:02:34.537215: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:116] No
ne of the MLIR optimization passes are enabled (registered 2)
2023-11-28 03:02:34.554692: I tensorflow/core/platform/profile_utils/cpu_utils.cc:112] CPU Fre
quency: 3799900000 Hz
```

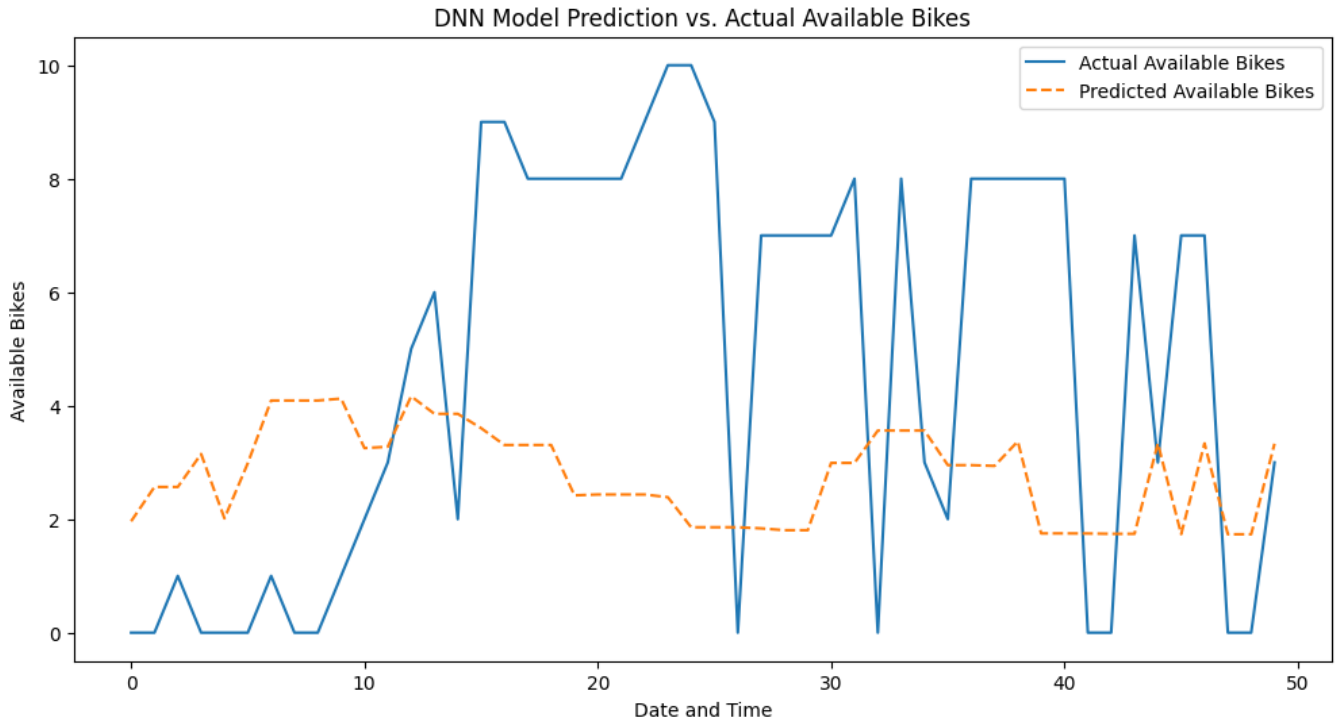
```
Epoch 1/50
182/182 [==============================] - 1s 1ms/step - loss: 11.9859
Epoch 2/50
182/182 [==============================] - 0s 2ms/step - loss: 6.8509
Epoch 3/50
182/182 [==============================] - 0s 2ms/step - loss: 6.6498
Epoch 4/50
182/182 [==============================] - 0s 2ms/step - loss: 6.6860
Epoch 5/50
182/182 [==============================] - 0s 2ms/step - loss: 6.6736
Epoch 6/50
182/182 [==============================] - 0s 1ms/step - loss: 6.6433
Epoch 7/50
182/182 [==============================] - 0s 1ms/step - loss: 6.6503
Epoch 8/50
182/182 [==============================] - 0s 1ms/step - loss: 6.4700
Epoch 9/50
182/182 [==============================] - 0s 2ms/step - loss: 6.5692
Epoch 10/50
182/182 [==============================] - 0s 1ms/step - loss: 6.6218
Epoch 11/50
182/182 [==============================] - 0s 1ms/step - loss: 6.3389
Epoch 12/50
182/182 [==============================] - 0s 1ms/step - loss: 6.8913
Epoch 13/50
182/182 [==============================] - 0s 1ms/step - loss: 6.3250
Epoch 14/50
182/182 [==============================] - 0s 1ms/step - loss: 6.5846
Epoch 15/50
182/182 [==============================] - 0s 1ms/step - loss: 6.5224
Epoch 16/50
182/182 [==============================] - 0s 1ms/step - loss: 6.3003
Epoch 17/50
182/182 [==============================] - 0s 1ms/step - loss: 6.3426
Epoch 18/50
182/182 [==============================] - 0s 1ms/step - loss: 6.3799
Epoch 19/50
182/182 [==============================] - 0s 973us/step - loss: 6.4972
Epoch 20/50
182/182 [==============================] - 0s 1ms/step - loss: 6.4537
Epoch 21/50
182/182 [==============================] - 0s 1ms/step - loss: 6.3061
Epoch 22/50
182/182 [==============================] - 0s 1ms/step - loss: 6.3645
Epoch 23/50
182/182 [==============================] - 0s 1ms/step - loss: 6.5395
Epoch 24/50
182/182 [==============================] - 0s 1ms/step - loss: 6.5323
Epoch 25/50
182/182 [==============================] - 0s 1ms/step - loss: 6.3609
Epoch 26/50
182/182 [==============================] - 0s 1ms/step - loss: 6.6324
Epoch 27/50
182/182 [==============================] - 0s 1ms/step - loss: 6.5429
Epoch 28/50
182/182 [==============================] - 0s 953us/step - loss: 6.4573
Epoch 29/50
182/182 [==============================] - 0s 1ms/step - loss: 6.3050
Epoch 30/50
182/182 [==============================] - 0s 1ms/step - loss: 6.3582
Epoch 31/50
182/182 [==============================] - 0s 1ms/step - loss: 6.7363
Epoch 32/50
182/182 [==============================] - 0s 1ms/step - loss: 6.6356
Epoch 33/50
182/182 [==============================] - 0s 1ms/step - loss: 6.2821
```

```
Epoch 34/50
182/182 [==============================] - 0s 1ms/step - loss: 6.3891
Epoch 35/50
182/182 [==============================] - 0s 1ms/step - loss: 6.2946
Epoch 36/50
182/182 [==============================] - 0s 1ms/step - loss: 6.5720
Epoch 37/50
182/182 [==============================] - 0s 1ms/step - loss: 6.2013
Epoch 38/50
182/182 [==============================] - 0s 982us/step - loss: 6.4854
Epoch 39/50
182/182 [==============================] - 0s 965us/step - loss: 6.4022
Epoch 40/50
182/182 [==============================] - 0s 1ms/step - loss: 6.4518
Epoch 41/50
182/182 [==============================] - 0s 1ms/step - loss: 6.5280
Epoch 42/50
182/182 [==============================] - 0s 1ms/step - loss: 6.2764
Epoch 43/50
182/182 [==============================] - 0s 898us/step - loss: 6.3683
Epoch 44/50
182/182 [==============================] - 0s 966us/step - loss: 6.5309
Epoch 45/50
182/182 [==============================] - 0s 1ms/step - loss: 6.7050
Epoch 46/50
182/182 [==============================] - 0s 1ms/step - loss: 6.4618
Epoch 47/50
182/182 [==============================] - 0s 1ms/step - loss: 6.4561
Epoch 48/50
182/182 [==============================] - 0s 1ms/step - loss: 6.4132
Epoch 49/50
182/182 [==============================] - 0s 1ms/step - loss: 6.4804
Epoch 50/50
182/182 [==============================] - 0s 2ms/step - loss: 6.2785
```

Out[17]:
```
<tensorflow.python.keras.callbacks.History at 0x7f62c445b4c0>
```

In [18]:
```python
# Make predictions on the test set
predictions = model.predict(test_features_scaled).flatten()

# Calculate Mean Absolute Error
mae = mean_absolute_error(test_target, predictions)
print(f'Mean Absolute Error: {mae}')

# Visualization plot
plt.figure(figsize=(12, 6))
plt.plot( test_target[:50], label='Actual Available Bikes')
plt.plot( predictions[:50], label='Predicted Available Bikes', linestyle='--')
plt.title('DNN Model Prediction vs. Actual Available Bikes')
plt.xlabel('Date and Time')
plt.ylabel('Available Bikes')
plt.legend()
plt.show()
```

```
Mean Absolute Error: 2.6593864688990516
```


DNN Model Prediction vs. Actual Available Bikes
Actual Available Bikes
Predicted Available Bikes
Available Bikes
Date and Time
0
2
4
6
8
10
0
10
20
30
40
50