**Hands-on Activity**

**Forecasting Power Consumption**

This hands-on activity is designed to evaluate candidates’ ability to preprocess data, implement forecasting models, and analyze their performance. Below is a detailed questionnaire and the goals to guide the activity.

**Data Description:**

Please download the dataset using the following link. <https://www.kaggle.com/datasets/fedesoriano/electric-power-consumption/data>

You can also find the data description there.

**Part 1: Understanding the Dataset**

1. What are the key features in the dataset, and how are they relevant to forecasting power consumption? **(Use SHAP analysis)**
2. What challenges might arise in forecasting time series data from this dataset? **(Give some figures from your analysis)**
3. How does weather (temperature, humidity, wind speed) potentially impact power consumption in each zone? **(Give some statistics from your analysis)**

**Part 2: Data Preprocessing**

1. You need to handle missing or inconsistent data in this dataset. Consider all necessary steps to process the data according to your needs.
2. I suggest refraining from applying feature engineering to compute additional time-based features. Instead, focus on using the raw data directly to implement your long-sequence modeling.
3. How would you split the dataset into training, validation, and test sets for time series forecasting? Write the code to achieve this.
4. Convert your time series data into tokens for use with the transformer models.

**Part 3: Model Implementation**

8. Implement the following forecasting models:

* **Vanilla Transformer**: <https://arxiv.org/abs/1706.03762>
* **PatchTST**: <https://arxiv.org/abs/2211.14730> (GitHub Repo: <https://github.com/yuqinie98/PatchTST> )

9. Compare the model architectures according to your understanding.

10.Feel free to use your own customized model that you think would deliver improved performance. **(Optional)**

**Part 4: Fine-tune each model for best performance:**

11. Experiment with different hyperparameters such as learning rate, batch size, sequence length, number of attention heads, and feedforward dimensions.

12. Adjust dropout rates and weight initialization to prevent overfitting.

13. Use appropriate optimizers and learning rate schedulers for training stability.

14. Apply early stopping based on validation loss or another evaluation metric.

**Part 5: Evaluation and Visualization**

15. What metrics would you use to evaluate forecasting performance?

16. Implement the evaluation metrics.

17. Visualize:

* Actual vs. Predicted power consumption for each model.
* Performance metrics across the three models for each zone.

18. Which model performs best overall? Are there any zones where performance varies significantly?

**Deliverables**

1. **Code**: Upload Python scripts to GitHub and provide comprehensive and clear instructions on how to implement the code in README.md. You may follow the structure given below.

├── README.md

├── data/

│ └── tetouan\_power\_data.csv

├── src/

│ ├── data\_preprocessing.py

│ ├── vanilla\_transformer.py

│ ├── patchtst.py

│ └── train\_evaluate.py

├── notebooks/

│ └── model\_analysis.ipynb

├── requirements.txt

1. **Report**: A detailed report that includes all the parts from 1 to 5.

**Important Note:**

* It's not necessary to finish everything. Do your best-there are no right or wrong answers. We're interested in seeing your approach to problem-solving and your dedication.
* If the code and content are not your own, please ensure to include appropriate documentation and references.

## **🔹 Part 1: Understanding the Dataset**

### **Step 1: Download and Load the Dataset**

You can download the dataset from Kaggle:  
🔗 [**Electric Power Consumption Dataset**](https://www.kaggle.com/datasets/fedesoriano/electric-power-consumption/data)

#### **Code to Load the Dataset in Python**

python

CopyEdit

import pandas as pd

# Load the dataset

df = pd.read\_csv("household\_power\_consumption.txt", sep=";", parse\_dates={'datetime': ['Date', 'Time']}, infer\_datetime\_format=True, na\_values=['?'], low\_memory=False)

# Display basic information

print(df.info())

print(df.head())

### **Step 2: Identify Key Features (SHAP Analysis)**

We use **SHAP (SHapley Additive exPlanations)** to analyze feature importance.  
**Installation:**

bash

CopyEdit

pip install shap

#### **Code to Perform SHAP Analysis**

python

CopyEdit

import shap

import xgboost as xgb

# Remove null values

df.dropna(inplace=True)

# Define predictor variables (excluding datetime)

X = df.drop(columns=["datetime", "Global\_active\_power"])

y = df["Global\_active\_power"]

# Train a simple XGBoost model

model = xgb.XGBRegressor()

model.fit(X, y)

# Explain the model’s predictions using SHAP

explainer = shap.Explainer(model)

shap\_values = explainer(X)

# Visualize feature importance

shap.summary\_plot(shap\_values, X)

✅ **Key features relevant to power consumption**:

* **Global\_active\_power** (Total power consumption).
* **Voltage** (Power supply stability).
* **Sub\_metering\_1, Sub\_metering\_2, Sub\_metering\_3** (Power consumption in different zones).

### **Step 3: Identify Challenges in Time Series Forecasting**

Challenges include:

* **Missing data** → Handle via interpolation or imputation.
* **Seasonality** → Detect with Fourier transforms.
* **Trend shifts** → Use rolling statistics to check drift.

**Code to Detect Seasonality and Trends:**

python

CopyEdit

import matplotlib.pyplot as plt

df.set\_index("datetime", inplace=True)

df["Global\_active\_power"].plot(figsize=(12,6), title="Power Consumption Over Time")

plt.show()

✅ **Figure interpretation** → Look for periodic trends and anomalies.

### **Step 4: Impact of Weather on Power Consumption**

python

CopyEdit

import seaborn as sns

sns.pairplot(df[['Temperature', 'Humidity', 'WindSpeed', 'Global\_active\_power']])

plt.show()

✅ **High correlation?** → Indicates potential impact of weather factors.

## **🔹 Part 2: Data Preprocessing**

### **Step 5: Handle Missing Data**

python

CopyEdit

df.fillna(method='ffill', inplace=True) # Forward fill missing values

✅ Ensures smooth time series data without gaps.

### **Step 6: Split Data into Train, Validation, and Test Sets**

python

CopyEdit

train\_size = int(len(df) \* 0.7)

val\_size = int(len(df) \* 0.15)

train = df.iloc[:train\_size]

val = df.iloc[train\_size:train\_size + val\_size]

test = df.iloc[train\_size + val\_size:]

✅ **Why?**

* **Train** → Model learns from historical data.
* **Validation** → Tune hyperparameters.
* **Test** → Evaluate final model performance.

### **Step 7: Convert Time Series Data into Tokens for Transformers**

python

CopyEdit

import numpy as np

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

df\_scaled = scaler.fit\_transform(df.drop(columns=["datetime"]))

# Convert time series into sequences

def create\_sequences(data, seq\_length):

X, y = [], []

for i in range(len(data) - seq\_length):

X.append(data[i:i+seq\_length])

y.append(data[i+seq\_length][0])

return np.array(X), np.array(y)

seq\_length = 24 # 24 hours of data

X, y = create\_sequences(df\_scaled, seq\_length)

✅ **Why?** → Transformers require input in tokenized form.

## **🔹 Part 3: Model Implementation**

### **Step 8: Implementing Vanilla Transformer**

python

CopyEdit

from transformers import BertModel

model = BertModel.from\_pretrained("bert-base-uncased")

✅ **Uses attention mechanism for long-range dependencies.**

### **Step 9: Implementing PatchTST**

Install PatchTST:

bash

CopyEdit

pip install torch torchvision

python

CopyEdit

import torch.nn as nn

class PatchTST(nn.Module):

def \_\_init\_\_(self):

super(PatchTST, self).\_\_init\_\_()

self.transformer = nn.Transformer(d\_model=128, num\_encoder\_layers=3)

def forward(self, x):

return self.transformer(x)

model = PatchTST()

✅ **PatchTST handles non-stationary data better.**

## **🔹 Part 4: Model Fine-tuning**

Experiment with different hyperparameters:

python

CopyEdit

learning\_rates = [0.001, 0.0005, 0.0001]

batch\_sizes = [32, 64]

attention\_heads = [4, 8]

✅ **Goal** → Find the best combination to improve accuracy.

## **🔹 Part 5: Evaluation & Visualization**

### **Step 10: Evaluate Model Performance**

Common **metrics**:

* **RMSE (Root Mean Square Error)**
* **MAE (Mean Absolute Error)**
* **MAPE (Mean Absolute Percentage Error)**

python

CopyEdit

from sklearn.metrics import mean\_squared\_error

def evaluate\_model(y\_true, y\_pred):

rmse = mean\_squared\_error(y\_true, y\_pred, squared=False)

print(f"RMSE: {rmse}")

evaluate\_model(y\_test, model.predict(X\_test))

✅ **Lower RMSE = Better model.**

### **Step 11: Visualizing Predictions**

python

CopyEdit

import matplotlib.pyplot as plt

plt.plot(y\_test, label="Actual")

plt.plot(model.predict(X\_test), label="Predicted", linestyle="dashed")

plt.legend()

plt.title("Actual vs Predicted Power Consumption")

plt.show()

✅ **Graph shows model accuracy visually.**

## **🔹 Deliverables**

### **GitHub Repository Structure**

css

CopyEdit

├── README.md

├── data/

│ └── household\_power\_consumption.csv

├── src/

│ ├── data\_preprocessing.py

│ ├── vanilla\_transformer.py

│ ├── patchtst.py

│ └── train\_evaluate.py

├── notebooks/

│ └── model\_analysis.ipynb

├── requirements.txt

✅ **README.md should contain clear instructions on how to run the project.**