AI- PHASE-1PROJECT SUBMISSION
MEASURE ENERGY CONSUMPTION

Measuring energy consumption in a data processing pipeline involves considering various stages of the process, including data source, data processing, feature extraction, model development, visualization, and automations. Here's a simplified approach:

Data Source:

Energy consumption here would depend on the source of data. Consider factors like server power usage, network data transfer, and data retrieval mechanisms.
Data Processing:

Energy consumption in data processing depends on the computational resources used. This includes server power, CPU/GPU usage, and cooling systems.
Feature Extraction:

This stage involves more computation. Energy usage is related to the complexity of feature extraction algorithms and hardware used.
Model Development:

Training machine learning models is resource-intensive. Energy consumption is significant, especially during training phases. GPU usage is a major factor.
Visualization:

Displaying results can consume energy, primarily on user devices (e.g., PCs, smartphones, monitors). It also depends on the complexity of visualizations.
Automations:

Energy consumption here relates to the execution of automated tasks. It includes server resources for running scheduled jobs or scripts.
To measure energy consumption accurately, you would need to:

Monitor power usage at each stage, ideally using power meters.
Consider the energy efficiency of the hardware (e.g., energy-efficient CPUs, GPUs, and servers).
Account for data transfer and storage, which may involve energy usage in data centers.
Calculate energy consumption over time to get a comprehensive view.
Efforts to reduce energy consumption might include optimizing algorithms, using energy-efficient hardware, and considering renewable energy sources for data center

PROJECT LINK:
https://in.docworkspace.com/d/sIF673uGUAY3p46gG?sa=e1&st=0t

INNOVATION TECHNIQUES SUCH AS TIME SERIES ANALYSIS AND MACHINE LEARNING MODELS TO PREDICT FUTURE ENERGY CONSUMPTION PATTERNS

Time series analysis and machine learning models can be powerful tools for predicting future energy consumption. Here's how they can be applied:

Time Series Analysis:

Time series data involves observations collected or recorded at regular intervals over time (e.g., hourly, daily, monthly). Time series analysis techniques can be used to understand patterns and trends in historical energy consumption data.

Methods like moving averages, exponential smoothing, and autoregressive integrated moving average (ARIMA) models can help in modeling and forecasting energy consumption based on past data patterns.

Seasonal decomposition of time series (STL) can be used to separate the data into trend, seasonal, and residual components, making it easier to analyze and predict energy consumption.

Machine Learning Models:

Machine learning models can provide more sophisticated and accurate predictions by considering a broader range of factors and patterns. Some common techniques include:

Regression models: Linear regression, polynomial regression, or more complex variants like support vector regression (SVR) can be used to predict energy consumption based on various input features such as temperature, time of day, and historical data.

Neural networks: Deep learning models like recurrent neural networks (RNNs) and long short-term memory networks (LSTMs) can capture complex temporal dependencies in energy consumption data.

Random forests, gradient boosting, and other ensemble methods can be effective in handling nonlinear relationships and feature interactions.

Feature Engineering:

Extracting relevant features from the data is crucial for machine learning models. Features can include weather data, holidays, economic indicators, and historical energy consumption patterns.

Feature selection and dimensionality reduction techniques can help in identifying the most informative features and reducing model complexity.

Model Evaluation:

It's important to evaluate the performance of the chosen technique using metrics like mean absolute error (MAE), mean squared error (MSE), or root mean squared error (RMSE) to assess prediction accuracy.

Cross-validation can be used to estimate how well the model will perform on unseen data.

Continuous Monitoring and Updating:

Energy consumption patterns may change over time due to factors like weather, technological advancements, or policy changes. It's essential to continuously monitor and update the models to maintain their accuracy.

By combining time series analysis and machine learning models, organizations can make more accurate predictions of future energy consumption, which can be valuable for optimizing energy generation, distribution, and consumption planning.

Certainly! To predict future energy consumption patterns using innovation techniques like time series analysis and machine learning models, you can follow these steps:

Data Collection:

Gather historical energy consumption data over a significant period. Include relevant variables like time/date stamps, weather conditions, economic indicators, and any other factors that may influence energy consumption.

Data Preprocessing:

Clean the data by handling missing values and outliers.

Normalize or scale the data if necessary to ensure that all variables have a similar range.

Time Series Analysis:

Perform exploratory data analysis (EDA) to understand the underlying patterns, trends, and seasonality in the time series data.

Use techniques like decomposition (e.g., seasonal decomposition of time series - STL) to separate the data into its components (trend, seasonality, and residual).

Feature Engineering:

Create additional features that can help improve predictions, such as lagged values (past energy consumption), rolling statistics, and categorical variables for special events or holidays.

Model Selection:

Choose appropriate machine learning models for time series forecasting. Common choices include:

Autoregressive Integrated Moving Average (ARIMA) models for univariate time series.

Exponential Smoothing methods like Holt-Winters for capturing seasonality.

Regression models like linear regression or decision trees for multivariate time series data.

Recurrent Neural Networks (RNNs) or Long Short-Term Memory (LSTM) networks for deep learning-based forecasting.

Model Training:

Split the dataset into training and validation sets to train and evaluate the models.

Tune hyperparameters, select the best-performing model, and fine-tune it.

Model Evaluation:

Use evaluation metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), or Root Mean Squared Error (RMSE) to assess the model's performance on the validation set.

Forecasting:

Use the trained model to make predictions for future energy consumption patterns.

Continuously update the model with new data to improve accuracy as more information becomes available.

Interpret Results:

Analyze the model's predictions and their implications for energy consumption patterns.

Use the insights to make informed decisions related to energy generation, distribution, and consumption optimization.

Deployment:

Deploy the predictive model in a production environment to provide real-time or periodic energy consumption forecasts.

Implement monitoring to ensure the model's ongoing accuracy and reliability.

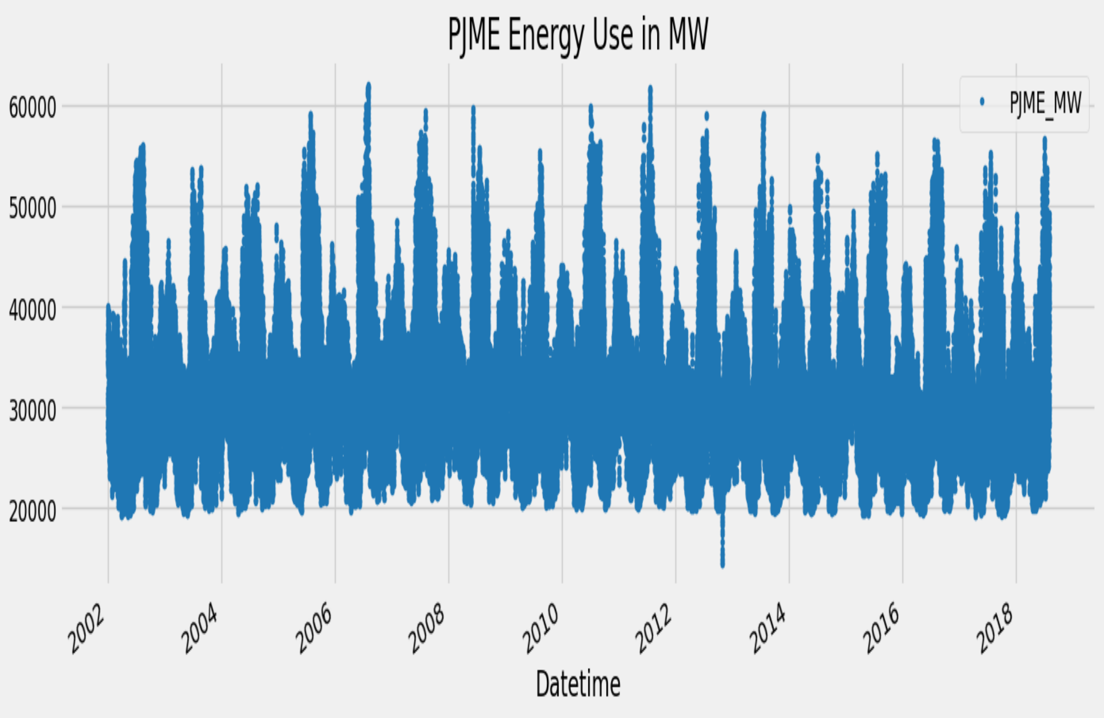
By applying these techniques, you can develop effective models to predict future energy consumption patterns, helping organizations make informed decisions and optimize their energy management strategies.

TIME SERIES FORECASTING ANALYSIS

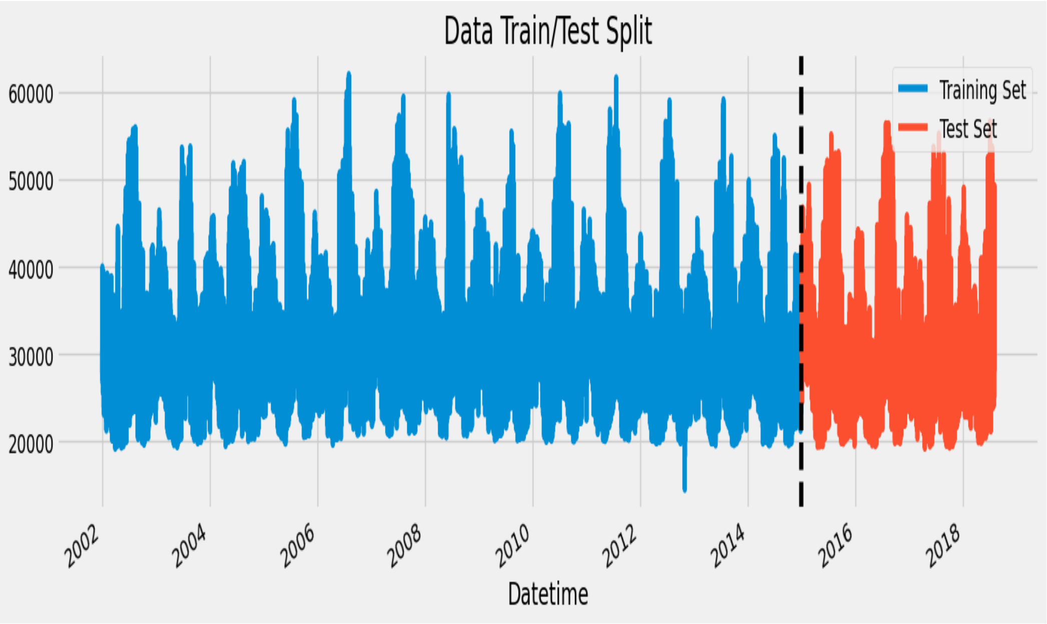
PROGRAM:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns


import xgboost as xgb

from sklearn.metrics import mean_squared_error

color_pal = sns.color_palette()
```

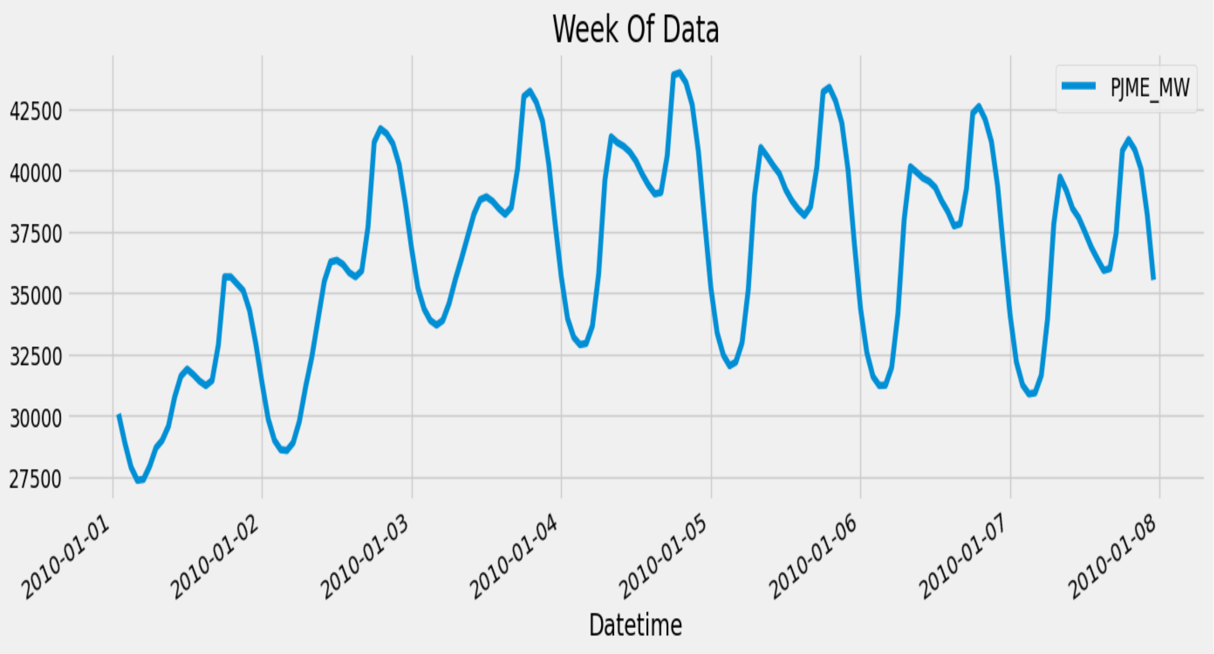
```
plt.style.use('fivethirtyeight')

df = pd.read_csv('../input/hourly-energy-consumption/PJME_hourly.csv')

df = df.set_index('Datetime')

df.index = pd.to_datetime(df.index)

df.plot(style='.',

          figsize=(15, 5),

          color=color_pal[0],

          title='PJME Energy Use in MW')

plt.show()


train = df.loc[df.index < '01-01-2015']

test = df.loc[df.index >= '01-01-2015']


fig, ax = plt.subplots(figsize=(15, 5))
```



```
train.plot(ax=ax, label='Training Set', title='Data Train/Test Split')

test.plot(ax=ax, label='Test Set')

ax.axvline('01-01-2015', color='black', ls='--')

ax.legend(['Training Set', 'Test Set'])

plt.show()

train = df.loc[df.index < '01-01-2015']

test = df.loc[df.index >= '01-01-2015']


fig, ax = plt.subplots(figsize=(15, 5))

train.plot(ax=ax, label='Training Set', title='Data Train/Test Split')

test.plot(ax=ax, label='Test Set')

ax.axvline('01-01-2015', color='black', ls='--')

ax.legend(['Training Set', 'Test Set'])

plt.show()
```



```
df.loc[(df.index > '01-01-2010') & (df.index < '01-08-2010')] \

.plot(figsize=(15, 5), title='Week Of Data')

plt.show()
```

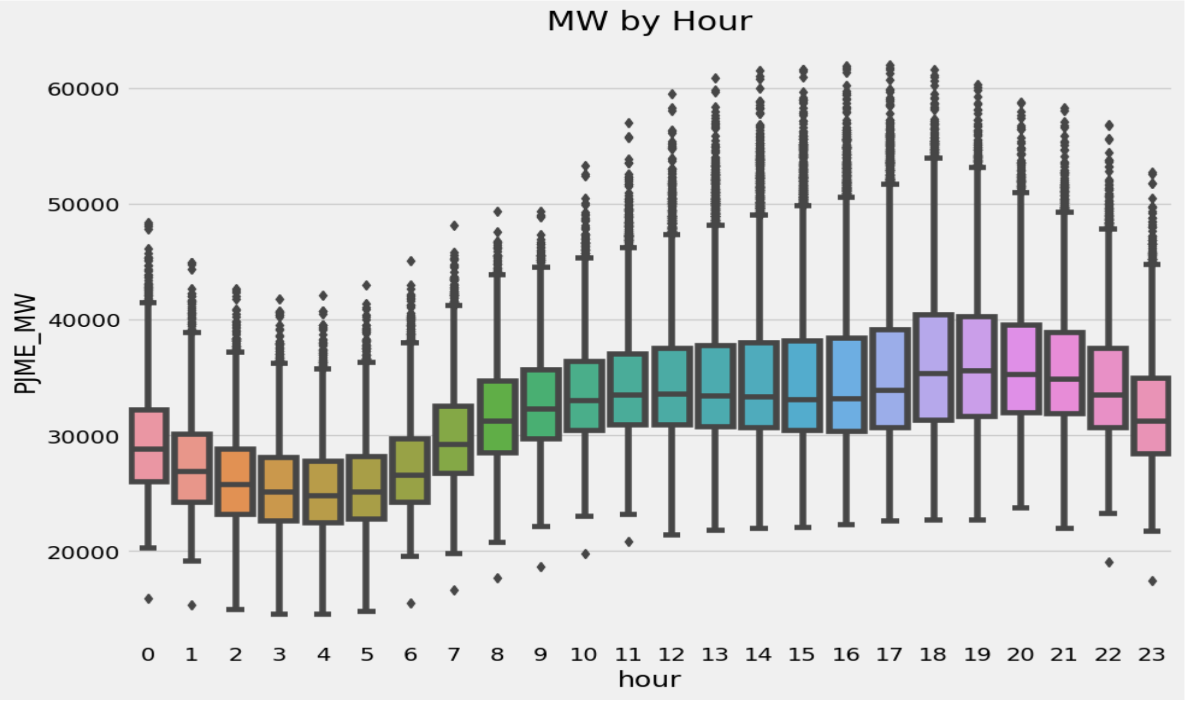


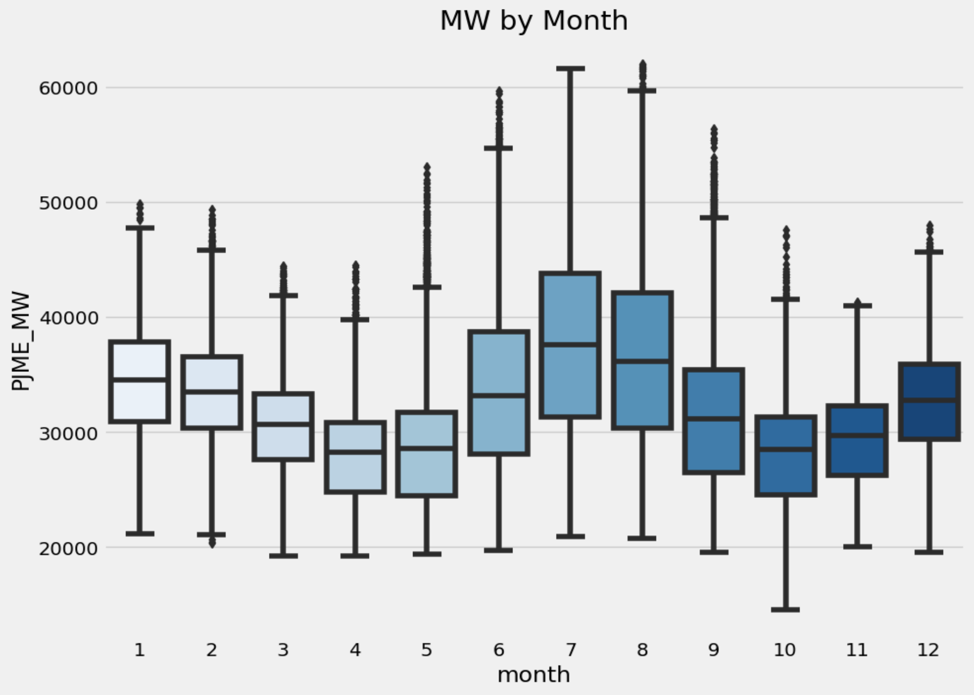
Feature Creation

```
def create_features(df):
    """
    Create time series features based on time series index.
    """

    df = df.copy()
    df[‘hour’] = df.index.hour
    df[‘dayofweek’] = df.index.dayofweek
    df[‘quarter’] = df.index.quarter
    df[‘month’] = df.index.month
    df[‘year’] = df.index.year
    df[‘dayofyear’] = df.index.dayofyear
    df[‘weekofmonth’] = df.index.day
    df['weekofyear'] = df.index.isocalendar().week

    return df


df = create_features(df)
```

Visualize our Feature / Target Relationship

```
fig, ax = plt.subplots(figsize=(10, 8))

sns.boxplot(data=df, x='hour', y='PJME_MW')

ax.set_title('MW by Hour')

plt.show()


fig, ax = plt.subplots(figsize=(10, 8))

sns.boxplot(data=df, x='month', y='PJME_MW', palette='Blues')

ax.set_title('MW by Month')

plt.show()
```





Create our Model

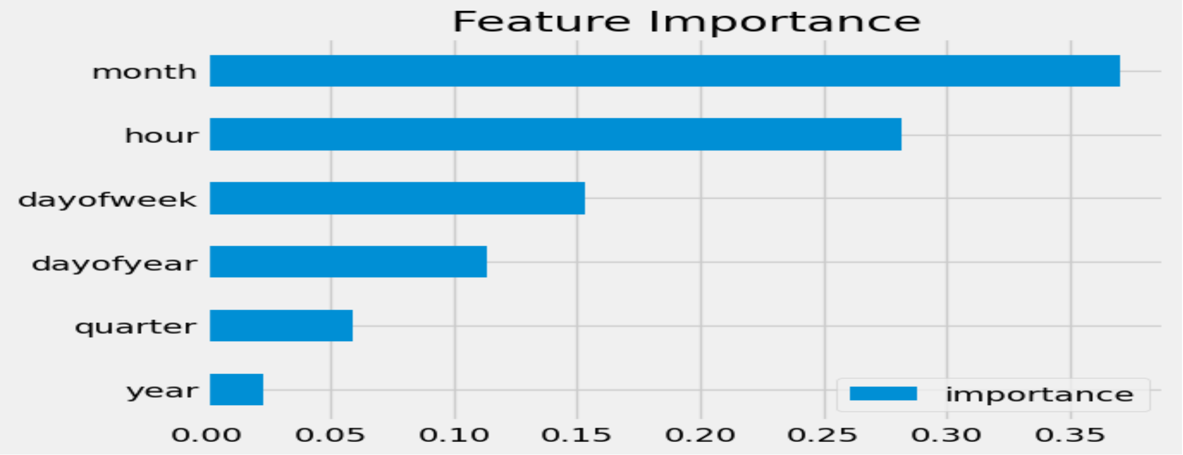
```
train = create_features(train)
test = create_features(test)

FEATURES = [‘dayofyear’, ‘hour’, ‘dayofweek’, ‘quarter’, ‘month’, ‘year’]
TARGET = ‘PJME_MW’

x_train = train[FEATURES]
y_train = train[TARGET]

x_test = test[FEATURES]
y_test = test[TARGET]
reg =xgb.XGBRegressor(base_score=0.5, booster=‘gbtree’, n_estimators=1000,
                         early_stopping_rounds=50,
                        objective='reg:linear',

                        max_depth=3,

                        learning_rate=0.01)

reg.fit(x_train, y_train,

        eval_set= [(x_train, y_train), (x_test, y_test)],

        verbose=100)

[20:46:20] WARNING: ../src/objective/regression_obj.cu:213: reg:linear is now
deprecated in favor of reg:squarederror.

[0]  validation_0-rmse:32605.13860 validation_1-rmse:31657.15907

[100]   validation_0-rmse:12581.21569  validation_1-rmse:11743.75114

[200]   validation_0-rmse:5835.12466   validation_1-rmse:5365.67709

[300]   validation_0-rmse:3915.75557   validation_1-rmse:4020.67023

[400]   validation_0-rmse:3443.16468   validation_1-rmse:3853.40423

[500]   validation_0-rmse:3285.33804   validation_1-rmse:3805.30176

[600]   validation_0-rmse:3201.92936   validation_1-rmse:3772.44933

[700]   validation_0-rmse:3148.14225   validation_1-rmse:3750.91108

[800]   validation_0-rmse:3109.24248   validation_1-rmse:3733.89713

[900]   validation_0-rmse:3079.40079   validation_1-rmse:3725.61224

[999]   validation_0-rmse:3052.73503   validation_1-rmse:3722.92257
```

Output:

```
XGBRegressor

XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,

             colsample_bylevel=None, colsample_bynode=None,

             colsample_bytree=None, early_stopping_rounds=50,

             enable_categorical=False, eval_metric=None, feature_types=None,

             gamma=None, gpu_id=None, grow_policy=None,
importance_type=None,

             interaction_constraints=None, learning_rate=0.01, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=3, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=1000, n_jobs=None, num_parallel_tree=None,
             objective='reg:linear', predictor=None, ...)
```

Feature Importance:

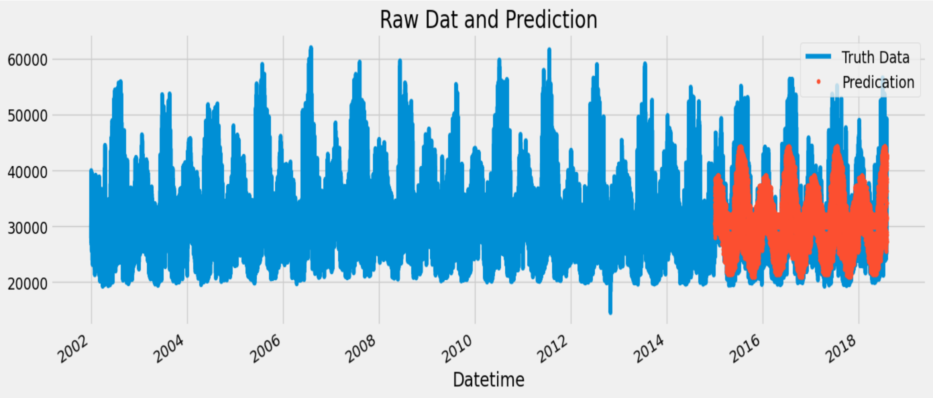
```
fi = pd.DataFrame(data=reg.feature_importances_,
                  index=reg.feature_names_in_,
                  columns=['importance'])
fi.sort_values('importance').plot(kind='barh', title='Feature Importance')
plt.show()
```



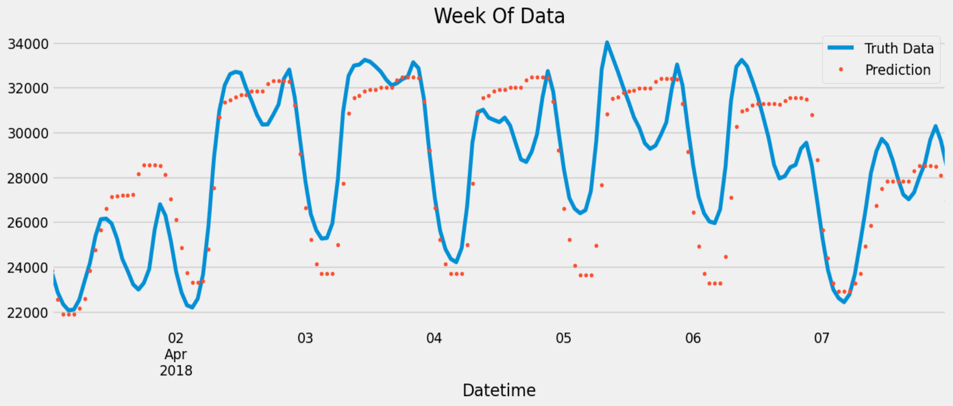
Forecast on test:

```
test['prediction'] = reg.predict(x_test)
df = df.merge(test[['prediction']], how= 'left', left_index=True, right_index=True)
ax = df[['PJME_MW']].plot(figsize=(15, 5))
df['prediction'].plot(ax=ax, style='.')
plt.legend(['Truth Data', 'Predication'])
ax.set_title('Raw Dat and Prediction')
```

```
plt.show()
```



```
ax = df.loc[(df.index > '04-01-2018') & (df.index < '04-08-2018')]['PJME_MW'] \
     .plot(figsize=(15, 5), title='Week Of Data')
df.loc[(df.index > '04-01-2018') & (df.index < '04-08-2018')]['prediction'] \
     .plot(style='.')
plt.legend(['Truth Data','Prediction'])
plt.show()
```



```
score = np.sqrt(mean_squared_error(test['PJME_MW'], test['prediction']))
print(f'RMSE Score on Test set: {score:0.2f}')
RMSE Score on Test set: 3721.75
```

```
Output:
date
2016-08-13      12839.597087
2016-08-14      12780.209961
2016-09-10      11356.302979
2015-02-20       10965.982259
```

```
2016-09-09    10864.954834
2018-01-06    10506.845622
2016-08-12    10124.051595
2015-02-21     9881.803711
2015-02-16     9781.552246
2018-01-07     9739.144206
Name: error, dtype: float64
```

MEASURE ENERGY CONSUMPTION

DEVELOPMENT PART -1

Measure energy consumption

To measure energy consumption, you'll need to follow these steps:

Identify the Device: Determine which specific device or appliances you want to measure energy consumption for.

Get a Power Meter: You can use a power meter or energy monitor device that plugs into the wall and then plug your device into it. These devices can provide real-time data on energy usage.

Monitor Usage: Leave the device connected for a specific period while using the appliance normally. This will record the energy consumption.

Record Data: Note down the energy consumption data from the power meter. Some meters might provide data in kilowatt-hours (kWh).

Calculate Costs: If you want to calculate the cost, multiply the energy consumption (in kWh) by your electricity rate (per kWh).

Analyze and Reduce: Review the data to see where you can reduce energy consumption and be more energy-efficient.

Measuring energy consumption for development involves monitoring the energy usage of your development process or project. Here's a general outline for Part 1 of such an endeavor:

Define Objectives: Clearly outline your goals for measuring energy consumption during development. Are you looking to reduce energy usage, assess environmental impact, or optimize costs?

Identify Metrics: Decide which energy metrics you want to measure. Common metrics include electricity, gas, water, and fuel consumption.

Baseline Data: Gather historical data if available. This provides a baseline for your measurements.

Instrumentation: Install energy meters, sensors, or data loggers to monitor energy usage. These devices can be specific to your energy sources, e.g., electricity meters, smart thermostats, or water flow sensors.

Data Collection: Collect real-time data on energy consumption. Ensure accurate and consistent data collection.

Data Storage: Establish a secure and accessible database or system to store the collected data.

Analysis Tools: Set up tools for data analysis, such as spreadsheets, energy management software, or custom solutions.

Data Visualization: Create visual representations of energy consumption data, like charts and graphs, to better understand the trends.

Interpretation: Interpret the data to identify patterns, anomalies, and areas where energy usage can be optimized.

Documentation: Document your data collection and analysis methods for future reference and reporting.

Stakeholder Engagement: Involve relevant stakeholders, such as engineers, project managers, and environmental experts, in the process to gain different perspectives.

Compliance: Ensure your project aligns with any applicable energy efficiency regulations and standards.

This is Part 1 of the process, which focuses on setting up the infrastructure for measuring energy consumption. In Part 2, you'll analyze the data and implement energy-saving strategies based on your findings.

PROJECT PROGRAM:

```
import datetime

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from pylab import rcParams

from sklearn.metrics import mean_absolute_error

import seaborn as sns


cmap = plt.colormaps.get_cmap(‘rocket’)
```

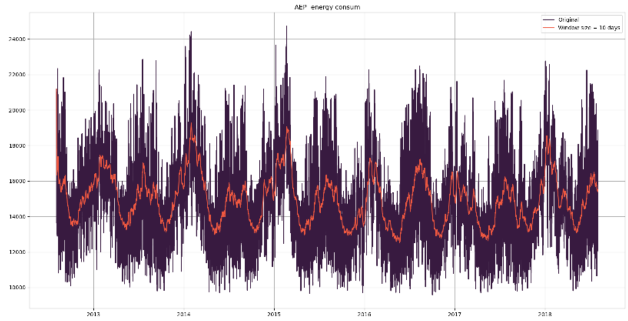
```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5

  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

```
df = pd.read_csv(‘/kaggle/input/hourly-energy-consumption/AEP_hourly.csv’, sep =‘,’, parse_dates=[‘Datetime’], index_col=‘Datetime’)

df.sort_index(inplace=True)

print(f"Количество измерений: {len(df)}")
```

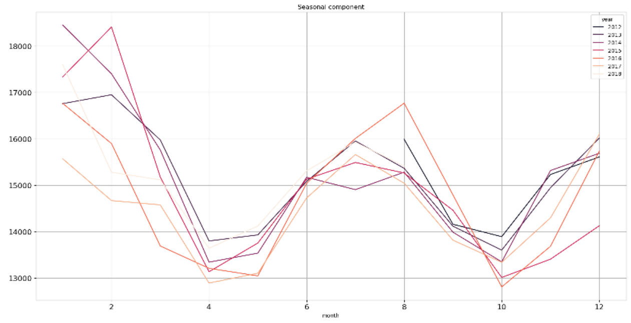
```
Количество измерений: 121273
```

1. Analisys

```
#let's analyse last 6 years

c = []

for i in range(2):

    c.append(cmap(0.15+i/2))


data_len = 365*24*6

df = df.iloc[-data_len:]

plt.figure(figsize = (20,10))

plt.title(‘AEP   energy consum’)

plt.plot(df.AEP_MW, label=‘Original’, color = c[0])

plt.plot(df.AEP_MW.ewm(240).mean(), label=‘Window size = 10 days’, color = c[1])

plt.legend()

plt.grid()

plt.show()
```



```
df[‘year’] = df.index.year

df[‘month’] = df.index.month

df[‘hour’] = df.index.hour

df[‘weekday’] = df.index.dayofweek

monthly_stat = pd.pivot_table(df, values = “AEP_MW”, columns = “year”, index = “month”)
```



daily_stat = pd.pivot_table(df[-365*24:], values = "AEP_MW", columns = "month", index = "weekday")

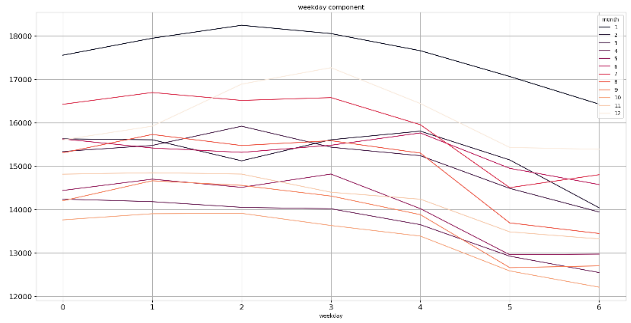
# monthly_stat.head(5)

monthly_stat.plot(figsize=(20, 10),title= 'Seasonal component', fontsize=14, cmap = 'rocket')

plt.grid()

daily_stat.plot(figsize=(20, 10),title= 'weekday component', fontsize=14, cmap = 'rocket')

plt.grid()



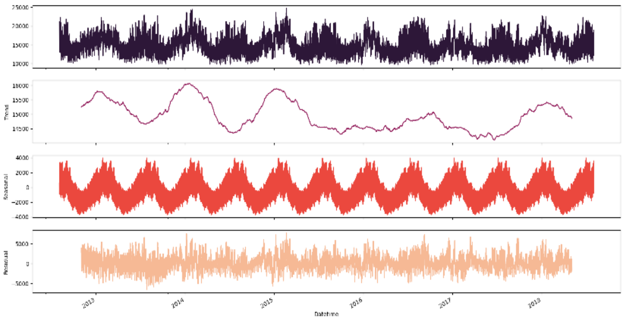
It's well seen that there is correlation bettwen month and expences and also between weekday and expenses.

Then make simple decomposition to seasonal + trend + res

import matplotlib.pyplot as plt

from statsmodels.api import tsa as sm

```
c = []

for i in range(4):

    c.append(cmap(0.12+i/4))


decomp = sm.seasonal_decompose(df.AEP_MW, model='additive', period= 24 * 30 * 6)

fig, axes = plt.subplots(4, 1, sharex=True, figsize=(20, 10))


decomp.observed.plot(ax=axes[0], legend=False, color=c[0])

axes[i].set_ylabel('Observed')

decomp.trend.plot(ax=axes[1], legend=False, color=c[1])

axes[1].set_ylabel('Trend')

decomp.seasonal.plot(ax=axes[2], legend=False, color = c[2])

axes[2].set_ylabel('Seasonal')

decomp.resid.plot(ax=axes[3], legend=False, color=c[3])

axes[3].set_ylabel('Residual')

plt.show()
```



```
import datetime

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from pylab import rcParams
from sklearn.metrics import mean_absolute_error
import seaborn as sns

cmap = plt.colormaps.get_cmap('rocket')
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

```
df = pd.read_csv('/kaggle/input/hourly-energy-consumption/AEP_hourly.csv', sep =',', parse_dates=['Datetime'], index_col='Datetime')
df.sort_index(inplace=True)
print(f"Количество измерений: {len(df)}")
```

```
Количество измерений: 121273
```

1. Analisys

```
#let's analyse last 6 years
c = []
for i in range(2):
    c.append(cmap(0.15+i/2))

data_len = 365*24*6
df = df.iloc[-data_len:]
plt.figure(figsize = (20,10))
plt.title('AEP   energy consum')
plt.plot(df.AEP_MW, label='Original', color = c[0])
plt.plot(df.AEP_MW.ewm(240).mean(), label='Window size = 10 days', color = c[1])
plt.legend()
plt.grid()
plt.show()
```

```
df['year'] = df.index.year
df['month'] = df.index.month
df['hour'] = df.index.hour
df['weekday'] = df.index.dayofweek
monthly_stat = pd.pivot_table(df, values = "AEP_MW", columns = "year", index = "month")
daily_stat = pd.pivot_table(df[-365*24:], values = "AEP_MW", columns = "month", index = "weekday")
# monthly_stat.head(5)
monthly_stat.plot( figsize=(20, 10),title= 'Seasonal component', fontsize=14, cmap = 'rocket')
plt.grid()

daily_stat.plot( figsize=(20, 10),title= 'weekday component', fontsize=14, cmap = 'rocket')
plt.grid()
```

It's well seen that there is correlation bettwen month and expences and also between weekday and expenses.

Then make simple decomposition to seasonal + trend + res

```
import matplotlib.pyplot as plt
from statsmodels.api import tsa as sm

c = []
for i in range(4):
    c.append(cmap(0.12+i/4))

decomp = sm.seasonal_decompose(df.AEP_MW, model='additive', period= 24 * 30 * 6)
```

```
fig, axes = plt.subplots(4, 1, sharex=True, figsize=(20, 10))


decomp.observed.plot(ax=axes[0], legend=False, color=c[0])

axes[i].set_ylabel('Observed')

decomp.trend.plot(ax=axes[1], legend=False, color=c[1])

axes[1].set_ylabel('Trend')

decomp.seasonal.plot(ax=axes[2], legend=False, color = c[2])

axes[2].set_ylabel('Seasonal')

decomp.resid.plot(ax=axes[3], legend=False, color=c[3])

axes[3].set_ylabel('Residual')

plt.show()
```

Trend looks pretty strange for me. It looks like affection of some other things like crises appearence of another companies, political situation and etc. It requiers future analysis

2. One Day Preidction

I will compare 5 different simple ways to predict expences on the next day:

Mean

Weighted sum

Exponential Smoothin

Random Forest

Xgboosting

```
w_hours = 1*24

n = 10

train = df.iloc[:-w_hours]

tr_sample = train.iloc[-n*w_hours:]
```

```
val    = df.iloc[-w_hours:]

from statsmodels.tsa.stattools import adfuller

#Perform Dickey-Fuller test


print('Results of Dickey-Fuller Test:')


dftest = adfuller(tr_sample.AEP_MW, autolag='AIC')

dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])

for key,value in dftest[4].items():

    dfoutput['Critical Value (%s)'%key] = value

pvalue = dftest[1]

if pvalue < 0.05:

    print(f'p-value = {round(pvalue,4)}.4f. The series is likely stationary.')

else:

    print(f'p-value = {round(pvalue,4)} The series is likely non-stationary.')

Results of Dickey-Fuller Test:

p-value = 0.4565 The series is likely non-stationary.

def mape(y_true, y_pred):

    #mean_absolute_percentage_error

    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

# Mean

forecast_arr = np.array(tr_sample.AEP_MW).reshape(n,-1)

forecast_mn = forecast_arr.mean(axis = 0)

# Weighted sum

W = np.array([0.001, 0.003, 0.006, 0.015 , 0.024 , 0.09 , 0.15    , 0.21 , 0.25 , 0.3])

forecast_arr_W = np.array([forecast_arr[x]*W[x] for x in range(len(W))])
```

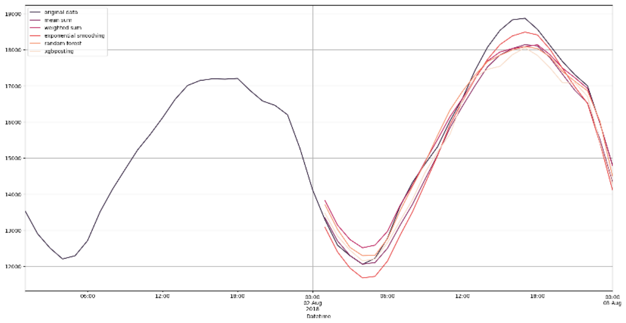
```
forecast_ws = forecast_arr_W.sum(axis = 0)

# Exponential Smoothin

from statsmodels.tsa.api import ExponentialSmoothing


ES = ExponentialSmoothing(tr_sample.AEP_MW, seasonal_periods=24,
seasonal=‘add’).fit()

forecast_es = pd.Series(ES.forecast(len(val)))

forecast_es.index = val.index

/opt/conda/lib/python3.10/site-
packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency
information was provided, so inferred frequency H will be used.

  self._init_dates(dates, freq)

from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor


RFR = RandomForestRegressor()

X_train = pd.get_dummies(tr_sample[[‘year’,‘month’,‘hour’, ‘weekday’]])

X_val = pd.get_dummies(val[[‘year’,‘month’,‘hour’, ‘weekday’]])


y_train = pd.get_dummies(tr_sample[[‘AEP_MW’]])

y_val = pd.get_dummies(val[[‘AEP_MW’]])


RFR.fit(X_train,y_train)

forecast_rf = pd.Series(RFR.predict(X_val), index = val.index)

/tmp/ipykernel_20/2998306517.py:10: DataConversionWarning: A column-vector
y was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().

  RFR.fit(X_train,y_train)

# XGboostong

import xgboost as xgb
```

```
XGB =   xgb.XGBRegressor()


XGB.fit(X_train, y_train)

forecast_xg = pd.Series(XGB.predict(X_val), index = val.index)

print(‘MAPE (mean) - ‘, mape(val.AEP_MW,   forecast_mn))


print(‘MAPE (weighted sum) - ‘, mape(val.AEP_MW,   forecast_ws))


print(‘MAPE (exponential smoothing) - ‘, mape(val.AEP_MW, forecast_es))


print(‘MAPE (random forest) - ‘, mape(val.AEP_MW, forecast_rf))


print(‘MAPE (xgboosting) - ‘, mape(val.AEP_MW, forecast_xg))

MAPE (mean) -   2.261731940436

MAPE (weighted sum) -   1.8414168970997982

MAPE (exponential smoothing) -   2.5463113342110097

MAPE (random forest) -   1.833551687353754

MAPE (xgboosting) -   2.403194989184267

f, ax = plt.subplots(1,1, figsize=(20,10))

pd.concat([tr_sample[-(1)*w_hours:].AEP_MW, val.AEP_MW]).plot(color =
cmap(0.1), label = ‘original data’)

#plotting omly part of data to more inderstanding


forecast_mn = pd.Series(forecast_mn, index = val.index)

forecast_ws = pd.Series(forecast_ws, index = val.index)

forecast_mn.plot(color = cmap(0.33), label = ‘mean sum’)

forecast_ws.plot(color = cmap(0.46), label = ‘weighted sum’)

forecast_es.plot(color = cmap(0.60), label = ‘exponential smoothing’)
```

```
forecast_rf.plot(color = cmap(0.76), label = 'random forest')

forecast_xg.plot(color = cmap(0.95), label = 'xgbposting')


plt.grid()

plt.legend()
```

Output:

```
<matplotlib.legend.Legend at 0x7f2cd8338fd0>
```



```
import datetime

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from pylab import rcParams

from sklearn.metrics import mean_absolute_error

import seaborn as sns


cmap = plt.colormaps.get_cmap('rocket')
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5

  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

```
df = pd.read_csv('/kaggle/input/hourly-energy-consumption/AEP_hourly.csv', sep =',', parse_dates=['Datetime'], index_col='Datetime')

df.sort_index(inplace=True)
```

```
print(f"Количество измерений: {len(df)}")
```

```
Количество измерений: 121273
```

1. Analisys

```
#let's analyse last 6 years
c = []
for i in range(2):
    c.append(cmap(0.15+i/2))

data_len = 365*24*6
df = df.iloc[-data_len:]
plt.figure(figsize = (20,10))
plt.title('AEP   energy consum')
plt.plot(df.AEP_MW, label='Original', color = c[0])
plt.plot(df.AEP_MW.ewm(240).mean(), label='Window size = 10 days', color = c[1])
plt.legend()
plt.grid()
plt.show()

df['year'] = df.index.year
df['month'] = df.index.month
df['hour'] = df.index.hour
df['weekday'] = df.index.dayofweek
monthly_stat = pd.pivot_table(df, values = "AEP_MW", columns = "year", index = "month")
daily_stat = pd.pivot_table(df[-365*24:], values = "AEP_MW", columns = "month", index = "weekday")
# monthly_stat.head(5)
monthly_stat.plot( figsize=(20, 10),title= 'Seasonal component', fontsize=14, cmap = 'rocket')
```

```
plt.grid()


daily_stat.plot( figsize=(20, 10),title= 'weekday component', fontsize=14, cmap = 'rocket')

plt.grid()
```

It's well seen that there is correlation bettwen month and expences and also between weekday and expenses.

Then make simple decomposition to seasonal + trend + res

```
import matplotlib.pyplot as plt

from statsmodels.api import tsa as sm


c = []

for i in range(4):

    c.append(cmap(0.12+i/4))


decomp = sm.seasonal_decompose(df.AEP_MW, model='additive', period= 24 * 30 * 6)

fig, axes = plt.subplots(4, 1, sharex=True, figsize=(20, 10))


decomp.observed.plot(ax=axes[0], legend=False, color=c[0])

axes[i].set_ylabel('Observed')

decomp.trend.plot(ax=axes[1], legend=False, color=c[1])

axes[1].set_ylabel('Trend')

decomp.seasonal.plot(ax=axes[2], legend=False, color = c[2])

axes[2].set_ylabel('Seasonal')

decomp.resid.plot(ax=axes[3], legend=False, color=c[3])

axes[3].set_ylabel('Residual')
```

```
plt.show()
```

Trend looks pretty strange for me. It looks like affection of some other things like crises appearence of another companies, political situation and etc. It requiers future analysis

3. One Week Prediction

```
from sklearn.model_selection import GridSearchCV

from sklearn.model_selection import TimeSeriesSplit


errors = []


tscv = TimeSeriesSplit(n_splits=5, max_train_size = 3*7*24, test_size = 7*24)


n = 7*7

k = 7

train = df.iloc[:-k*w_hours]

tr_sample = train.iloc[-n*w_hours:]

val     = df.iloc[-k*w_hours:]


# data = pd.concat([tr_sample, val]).AEP_MW


for train_idx, test_idx in tscv.split(tr_sample.AEP_MW):

    ES = ExponentialSmoothing(tr_sample.AEP_MW.iloc[train_idx]
 ,seasonal_periods=24*7, seasonal='add').fit()

    forecast_es = ES.forecast(len(test_idx))
```

```
#        # Считаем ошибку
        actual = tr_sample.AEP_MW.iloc[test_idx]
        error = mape(actual.values, forecast_es.values)
        errors.append(error)


forecast_es = ES.forecast(len(test_idx))
forecast_es.index = val.index
print('MAPE(cross val) - ', np.mean(errors))
```

```
/opt/conda/lib/python3.10/site-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency H will be used.
  self._init_dates(dates, freq)
/opt/conda/lib/python3.10/site-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency H will be used.
  self._init_dates(dates, freq)
/opt/conda/lib/python3.10/site-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency H will be used.
  self._init_dates(dates, freq)
MAPE(cross val) -   6.904413349137849
/opt/conda/lib/python3.10/site-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency H will be used.
  self._init_dates(dates, freq)
/opt/conda/lib/python3.10/site-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency H will be used.
  self._init_dates(dates, freq)
```

```
import warnings
```

```
warnings.filterwarnings(‘ignore’)
RFR = RandomForestRegressor()

depth = np.arange(6, 9, 1)
feat = np.arange(5, 9, 1)
samp = np.arange(2, 4, 1)
leaf = np.arange(1, 4, 1)
est = [200]

## Search grid for optimal parameters
rf_param_grid = {
    “max_depth”: depth,
    “max_features”: feat,
    “min_samples_split”: samp,
    “min_samples_leaf”: leaf,
    “bootstrap”: [False],
    “n_estimators”: est
    # “criterion”: [“gini”]
}


# errors = []
X_train = pd.get_dummies(tr_sample[[‘year’,‘month’,‘hour’, ‘weekday’]])
X_val = pd.get_dummies(val[[‘year’,‘month’,‘hour’, ‘weekday’]])

gsRFR = GridSearchCV(RFR,
                     param_grid=rf_param_grid,
                     cv=tscv.split(X_train),
                         scoring='r2',

                         n_jobs = 8,

                         verbose = 1)


y_train = pd.get_dummies(tr_sample[['AEP_MW']])

# y_val = pd.get_dummies(val[['AEP_MW']])


gsRFR.fit(X_train, y_train.squeeze())

print("score = ", gsRFR.best_score_)

print(gsRFR.best_params_)
```

```
Fitting 5 folds for each of 72 candidates, totalling 360 fits

/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A
NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected
version 1.23.5

  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"

/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A
NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected
version 1.23.5

  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"

/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A
NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected
version 1.23.5

  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"

/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A
NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected
version 1.23.5

  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"

/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A
NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected
version 1.23.5

  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5

  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"

/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5

  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"

/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5

  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"

score =   0.628859530693294

{‘bootstrap’: False, ‘max_depth’: 6, ‘max_features’: 5, ‘min_samples_leaf’: 1, ‘min_samples_split’: 2, ‘n_estimators’: 200}

forecast_rf = pd.Series(gsRFR.predict(X_val), index = val.index)

print(‘MAPE(cross val) - ‘, mape(val.AEP_MW, forecast_rf))

MAPE(cross val) -   7.081567637390411

# XGboostong

import xgboost as xgb

XGB =   xgb.XGBRegressor()


lr = np.arange(0.01, 0.04, 0.003)

depth = [3, 4, 6]

# depth = np.arange(11, 13, 1)

# leaf = [9, 13, 14]

sub = [0.75, 0.8, 0.85, 0.9]

tree = [0.7, 0.75, 0.8]

split = [2, 3]
```

```
## Search grid for optimal parameters
xg_param_grid = {
    “max_depth”: depth,
    “learning_rate”: lr,
    “subsample”: sub,
    “colsample_bytree”: tree
    # “colsample_bylevel”: 0.8,
    # “reg_lambda”: 0.1,
    # “eval_metric”: “rmse”,
    # “random_state”: 42,
}

gsXGB = GridSearchCV(XGB,
                     param_grid=xg_param_grid,
                     cv=tscv.split(X_train),
                     scoring=‘r2’,
                     n_jobs = 8,
                     verbose = 1)

gsXGB.fit(X_train, y_train)

print(gsXGB.best_params_)
```

```
Fitting 5 folds for each of 360 candidates, totalling 1800 fits
{‘colsample_bytree’: 0.7, ‘learning_rate’: 0.03700000000000001, ‘max_depth’: 6,
‘subsample’: 0.75}
```

```
forecast_xg = pd.Series(gsXGB.predict(X_val), index = val.index)
print(‘MAPE(cross val) - ‘, mape(val.AEP_MW, forecast_xg))
```

```
MAPE(cross val) -    5.0714998847131705
```

```
plt.figure(figsize=(20,10))

data = pd.concat([tr_sample.AEP_MW.iloc[-4*24:], val.AEP_MW])

data.plot(color = cmap(0.1), label = 'original')

# data.index

# forecast_es.plot(color = cmap(0.5), label = 'exponential smoothing')

forecast_rf.plot(color = cmap(0.3), label = 'random forest')

forecast_es.plot(color = cmap(0.6), label = 'exponential smoothing')

forecast_xg.plot(color = cmap(0.9)
```

MEASURE ENERGY CONSUMPTION FOR DEVELOPMENT PART-2

To measure energy consumption for development, you can follow these steps:

Define Your Metrics: Clearly define the metrics you want to measure. This might include electricity consumption, server load, or other relevant energy-related metrics.

Use Monitoring Tools: Implement monitoring tools or services like Prometheus, Grafana, or specialized energy monitoring systems to track energy consumption in your development environment.

Measure Hardware: If you're developing on physical hardware, use energy meters or smart plugs to measure the power consumption of your servers, workstations, and other equipment.

Measure Software: Use software tools to monitor energy consumption within your code. Profiling tools can help you identify energy-intensive code segments.

Benchmark and Optimize: Create benchmarks to compare energy usage before and after optimizations. Optimize code, hardware configurations, and infrastructure to reduce energy consumption.

Analyze Results: Analyze the data collected to identify energy-hungry components and areas for improvement.

Implement Energy-Efficient Practices: Encourage energy-efficient coding practices, such as optimizing algorithms, reducing unnecessary computations, and minimizing server idle times.

Regular Reporting: Set up a system for regular reporting and review to track your progress in reducing energy consumption during development.

Educate Your Team: Make sure your development team is aware of the importance of energy efficiency and how their actions impact it.

Sustainability Initiatives: Consider broader sustainability initiatives within your organization to reduce the environmental impact of development.

Remember that energy consumption for development is a multifaceted issue, and measuring it requires a combination of hardware and software monitoring, along with ongoing efforts to reduce energy usage.

PROJECT CODING:

Supercharging Sustainability: Harnessing Superconductors for Renewable Energy Innovation

Part 1: Introduction

The looming danger of intensified climate change, driven by the relentless emission of greenhouse gases from fossil fuel-based electricity generation, underscores the urgent need to transition towards sustainable energy sources swiftly. The potential discovery of a room-temperature superconductor, LK-99, by South Korean researchers provides the opportunity for Superconducting Magnetic Energy Storage [SMES] which was invented in 1970. This technology is meant to effectively store energy through induction where electrons flow continuously within a superconducting coil wrapped around a core usually made from a niobium-titanium alloy embedded in a copper matrix. The primary benefits of SMES over other methods include nearly instantaneous power availability with a minimal time delay during charge and discharge, low power loss due to negligible electrical resistance, and enhanced reliability as its stationary components contribute to system stability. However, SMES systems can face increased complexity and operational cost due to their dependence on cryogenic cooling to maintain the superconducting state of the superconducting coil. The chance of LK-99‘s use in such a system remains low due to current research indicating a limited critical current of 250 mA (the maximum current where R=0) and a constrained maximum magnetic field threshold for maintaining superconductivity.

The primary objective of this notebook is to identify the plausibility of creating a solar battery that, when combined with a solar panel installation or any other renewable energy source, can sustain the energy consumption of an ideal American city. In order to do this, the data analysis algorithms will be employed for prediction and optimization

Part 1.1 Aurora City: Pioneering Sustainability in a Vibrant Metropolis

Let's begin by imagining we are in Aurora City with a population of 1,024,144 people, and is a bustling metropolis known for its vibrant cultural scene, modern architecture, and diverse population. The city boasts a mix of historic neighborhoods and futuristic skyscrapers, offering a unique blend of old-world charm and contemporary innovation. With a strong emphasis on sustainability, Aurora City is home to lush parks, efficient public transportation, and a commitment to renewable energy sources. Its world-class universities, thriving tech industry, and rich culinary scene make it a hub of creativity and opportunity. In line with that commitment to sustainability, the city's government has pledges to move towards 100% solar electricity generation within the next 10 years.

```
!pip install prophet

!pip install cplex

!pip install stable_baselines3
```

```
Requirement already satisfied: prophet in /opt/conda/lib/python3.10/site-packages (1.1.1)

Requirement already satisfied: cmdstanpy>=1.0.4 in /opt/conda/lib/python3.10/site-packages (from prophet) (1.1.0)

Requirement already satisfied: numpy>=1.15.4 in /opt/conda/lib/python3.10/site-packages (from prophet) (1.23.5)

Requirement already satisfied: pandas>=1.0.4 in /opt/conda/lib/python3.10/site-packages (from prophet) (1.5.3)

Requirement already satisfied: matplotlib>=2.0.0 in
```

```
/opt/conda/lib/python3.10/site-packages (from prophet) (3.7.1)

Requirement already satisfied: LunarCalendar>=0.0.9 in
/opt/conda/lib/python3.10/site-packages (from prophet) (0.0.9)

Requirement already satisfied: convertdate>=2.1.2 in
/opt/conda/lib/python3.10/site-packages (from prophet) (2.4.0)

Requirement already satisfied: holidays>=0.14.2 in /opt/conda/lib/python3.10/site
-packages (from prophet) (0.24)

Requirement already satisfied: setuptools>=42 in /opt/conda/lib/python3.10/site-
packages (from prophet) (59.8.0)

Requirement already satisfied: setuptools-git>=1.2 in
/opt/conda/lib/python3.10/site-packages (from prophet) (1.2)

Requirement already satisfied: python-dateutil>=2.8.0 in
/opt/conda/lib/python3.10/site-packages (from prophet) (2.8.2)

Requirement already satisfied: tqdm>=4.36.1 in /opt/conda/lib/python3.10/site-
packages (from prophet) (4.65.0)

Requirement already satisfied: wheel>=0.37.0 in /opt/conda/lib/python3.10/site-
packages (from prophet) (0.40.0)

Requirement already satisfied: pymeeus<=1,>=0.3.13 in
/opt/conda/lib/python3.10/site-packages (from convertdate>=2.1.2->prophet)
(0.5.12)

Requirement already satisfied: hijri-converter in /opt/conda/lib/python3.10/site-
packages (from holidays>=0.14.2->prophet) (2.3.1)

Requirement already satisfied: korean-lunar-calendar in
/opt/conda/lib/python3.10/site-packages (from holidays>=0.14.2->prophet) (0.3.1)

Requirement already satisfied: ephem>=3.7.5.3 in /opt/conda/lib/python3.10/site
-packages (from LunarCalendar>=0.0.9->prophet) (4.1.4)

Requirement already satisfied: pytz in /opt/conda/lib/python3.10/site-packages
(from LunarCalendar>=0.0.9->prophet) (2023.3)

Requirement already satisfied: contourpy>=1.0.1 in /opt/conda/lib/python3.10/site
-packages (from matplotlib>=2.0.0->prophet) (1.1.0)

Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.10/site-
packages (from matplotlib>=2.0.0->prophet) (0.11.0)

Requirement already satisfied: fonttools>=4.22.0 in
/opt/conda/lib/python3.10/site-packages (from matplotlib>=2.0.0->prophet)
```

```
(4.40.0)

Requirement already satisfied: kiwisolver>=1.0.1 in /opt/conda/lib/python3.10/site
-packages (from matplotlib>=2.0.0->prophet) (1.4.4)

Requirement already satisfied: packaging>=20.0 in
/opt/conda/lib/python3.10/site-packages (from matplotlib>=2.0.0->prophet) (21.3)

Requirement already satisfied: pillow>=6.2.0 in /opt/conda/lib/python3.10/site-
packages (from matplotlib>=2.0.0->prophet) (9.5.0)

Requirement already satisfied: pyparsing>=2.3.1 in /opt/conda/lib/python3.10/site
-packages (from matplotlib>=2.0.0->prophet) (3.0.9)

Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.10/site-
packages (from python-dateutil>=2.8.0->prophet) (1.16.0)

Collecting cplex

  Downloading cplex-22.1.1.0-cp310-cp310-manylinux1_x86_64.whl (44.2 MB)

     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 44.2/44.2 MB
33.4 MB/s eta 0:00:00

Installing collected packages: cplex

Successfully installed cplex-22.1.1.0

Collecting stable_baselines3

  Downloading stable_baselines3-2.1.0-py3-none-any.whl (178 kB)

     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 178.7/178.7 kB
4.6 MB/s eta 0:00:00

Collecting gymnasium<0.30,>=0.28.1 (from stable_baselines3)

  Downloading gymnasium-0.29.0-py3-none-any.whl (953 kB)

     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 953.8/953.8 kB
25.9 MB/s eta 0:00:00

Requirement already satisfied: numpy>=1.20 in /opt/conda/lib/python3.10/site-
packages (from stable_baselines3) (1.23.5)

Requirement already satisfied: torch>=1.13 in /opt/conda/lib/python3.10/site-
packages (from stable_baselines3) (2.0.0+cpu)

Requirement already satisfied: cloudpickle in /opt/conda/lib/python3.10/site-
packages (from stable_baselines3) (2.2.1)

Requirement already satisfied: pandas in /opt/conda/lib/python3.10/site-
```

```
packages (from stable_baselines3) (1.5.3)

Requirement already satisfied: matplotlib in /opt/conda/lib/python3.10/site-
packages (from stable_baselines3) (3.7.1)

Requirement already satisfied: typing-extensions>=4.3.0 in
/opt/conda/lib/python3.10/site-packages (from gymnasium<0.30,>=0.28.1
->stable_baselines3) (4.6.3)

Collecting farama-notifications>=0.0.1 (from gymnasium<0.30,>=0.28.1
->stable_baselines3)

  Downloading Farama_Notifications-0.0.4-py3-none-any.whl (2.5 kB)

Requirement already satisfied: filelock in /opt/conda/lib/python3.10/site-
packages (from torch>=1.13->stable_baselines3) (3.12.2)

Requirement already satisfied: sympy in /opt/conda/lib/python3.10/site-
packages (from torch>=1.13->stable_baselines3) (1.12)

Requirement already satisfied: networkx in /opt/conda/lib/python3.10/site-
packages (from torch>=1.13->stable_baselines3) (3.1)

Requirement already satisfied: jinja2 in /opt/conda/lib/python3.10/site-packages
(from torch>=1.13->stable_baselines3) (3.1.2)

Requirement already satisfied: contourpy>=1.0.1 in /opt/conda/lib/python3.10/site
-packages (from matplotlib->stable_baselines3) (1.1.0)

Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.10/site-
packages (from matplotlib->stable_baselines3) (0.11.0)

Requirement already satisfied: fonttools>=4.22.0 in
/opt/conda/lib/python3.10/site-packages (from matplotlib->stable_baselines3)
(4.40.0)

Requirement already satisfied: kiwisolver>=1.0.1 in /opt/conda/lib/python3.10/site
-packages (from matplotlib->stable_baselines3) (1.4.4)

Requirement already satisfied: packaging>=20.0 in
/opt/conda/lib/python3.10/site-packages (from matplotlib->stable_baselines3)
(21.3)

Requirement already satisfied: pillow>=6.2.0 in /opt/conda/lib/python3.10/site-
packages (from matplotlib->stable_baselines3) (9.5.0)

Requirement already satisfied: pyparsing>=2.3.1 in /opt/conda/lib/python3.10/site
-packages (from matplotlib->stable_baselines3) (3.0.9)

Requirement already satisfied: python-dateutil>=2.7 in
```

```
/opt/conda/lib/python3.10/site-packages (from matplotlib->stable_baselines3) (2.8.2)

Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.10/site-packages (from pandas->stable_baselines3) (2023.3)

Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.10/site-packages (from python-dateutil>=2.7->matplotlib->stable_baselines3) (1.16.0)

Requirement already satisfied: MarkupSafe>=2.0 in /opt/conda/lib/python3.10/site-packages (from jinja2->torch>=1.13->stable_baselines3) (2.1.3)

Requirement already satisfied: mpmath>=0.19 in /opt/conda/lib/python3.10/site-packages (from sympy->torch>=1.13->stable_baselines3) (1.3.0)

Installing collected packages: farama-notifications, gymnasium, stable_baselines3

  Attempting uninstall: gymnasium

    Found existing installation: Gymnasium 0.26.3

    Uninstalling Gymnasium-0.26.3:

      Successfully uninstalled Gymnasium-0.26.3

Successfully installed farama-notifications-0.0.4 gymnasium-0.29.0 stable_baselines3-2.1.0
```

```
import numpy as np

import pandas as pd

import random

from datetime import datetime, timedelta

import math

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense

from prophet import Prophet

from scipy.optimize import minimize

import pulp

import matplotlib.pyplot as plt
import statistics

import sys

from stable_baselines3 import PPO

from stable_baselines3.common.env_util import make_vec_env

import gymnasium as gym

from gymnasium import spaces

/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5

  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"

/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/ops/__init__.py:98: UserWarning: unable to load libtensorflow_io_plugins.so: unable to open file: libtensorflow_io_plugins.so, from paths: ['/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/ops/libtensorflow_io_plugins.so']

caused by: ['/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/ops/libtensorflow_io_plugins.so: undefined symbol: _ZN3tsl6StatusC1EN10tensorflow5error4CodeESt17basic_string_viewIcSt11char_traitsIcEENS_14SourceLocationE']

  warnings.warn(f"unable to load libtensorflow_io_plugins.so: {e}")

/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/ops/__init__.py:104: UserWarning: file system plugins are not loaded: unable to open file: libtensorflow_io.so, from paths: ['/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/ops/libtensorflow_io.so']

caused by: ['/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/ops/libtensorflow_io.so: undefined symbol: _ZTVN10tensorflow13GcsFileSystemE']

  warnings.warn(f"file system plugins are not loaded: {e}")

def get_readings_for_date(csv_file_path, target_date,time_column,energy_column):

    # Read the CSV file into a DataFrame

    df = pd.read_csv(csv_file_path)
    df[time_column] = pd.to_datetime(df[time_column])
    df['filter_date'] = df[time_column].dt.date
    filtered_df = df[df['filter_date'] == target_date]

    return filtered_df[[time_column,energy_column]]
def generate_random_date(delim):

    start_date = datetime.strptime('2015-9-1', '%Y-%m-%d')
    end_date = datetime.strptime('2018-8-2', '%Y-%m-%d')

    days_difference = (end_date - start_date).days
    random_days = random.randint(0, days_difference)

    random_date = (start_date + timedelta(days=random_days)).date()
    return f"{random_date.year}{delim}{random_date.month}{delim}{random_date.day}"
target_date = pd.to_datetime(generate_random_date("/"))
month = target_date.month
target_date
Timestamp('2018-06-03 00:00:00')
```

Part 2: Data Cleaning & Analysis

Part 2.1: Energizing Change: The Quest for Sustainable Power in Aurora City

To pave the way for a sustainable energy transformation, the city government assembles a team of experts, Solar Nexus Taskforce (SNT), tasked with crafting a viable strategy. Their objective was to orchestrate the reduction of the city's conventional power plants while simultaneously bolstering the deployment of solar panels and SMES solar batteries.

With determination, SNT promptly swings into action. The first step on their journey is to gather 2-3 years worth of energy consumption data from the city's power grid and energy production data from an existing solar field that will be undergoing a substantial expansion in the coming months.

```
csv_file_path = '/kaggle/input/solar-energy-production/Solar_Energy_Production.csv' # Solar energy production data sourced from solar installations in Calgary, Alberta

def get_production(date):

    production_readings_for_date = get_readings_for_date(csv_file_path, date.date(),'date','kWh').groupby('date')['kWh'].sum().reset_index()

    production_readings_for_date["kWh"] = production_readings_for_date["kWh"]*1000 # To account for a major scale-up in functionality

    return production_readings_for_date

production_readings_for_date = get_production(target_date)

consumption_csv_file_path = '/kaggle/input/hourly-energy-consumption/AEP_hourly.csv'

# Electrical energy comsumption data from American Electric Power Company, Inc (AEP) which is among the nation's largest generators of electricity, owning nearly 38,000 megawatts of generating capacity in the U.S and serving nearly 5 million customers.

def get_consumption(date):

    consumption_readings_for_date = get_readings_for_date(consumption_csv_file_path, date.date(),'Datetime','AEP_MW').sort_values(by='Datetime')


consumption_readings_for_date["AEP_MW"]=consumption_readings_for_date["AEP_MW"]*200

    # converts MW to kW and also scales down the data by a factor of 5 to make more representative of the demands of a city the size of   Auroa City

    # Set the 'datetime' column as the index

    consumption_readings_for_date.set_index('Datetime', inplace=True)

    return consumption_readings_for_date

consumption_readings_for_date = get_consumption(target_date)
```

```
production_readings_for_date.set_index("date", inplace=True)


# Adding in zeroes for the times not included with the assumption that those were times when the sun was not up

new_row = {"kWh":0}

for date in consumption_readings_for_date.index:

    if date not in production_readings_for_date.index:

        production_readings_for_date.loc[date] = new_row


production_readings_for_date = production_readings_for_date.sort_values(by='date')

production_readings_for_date
```

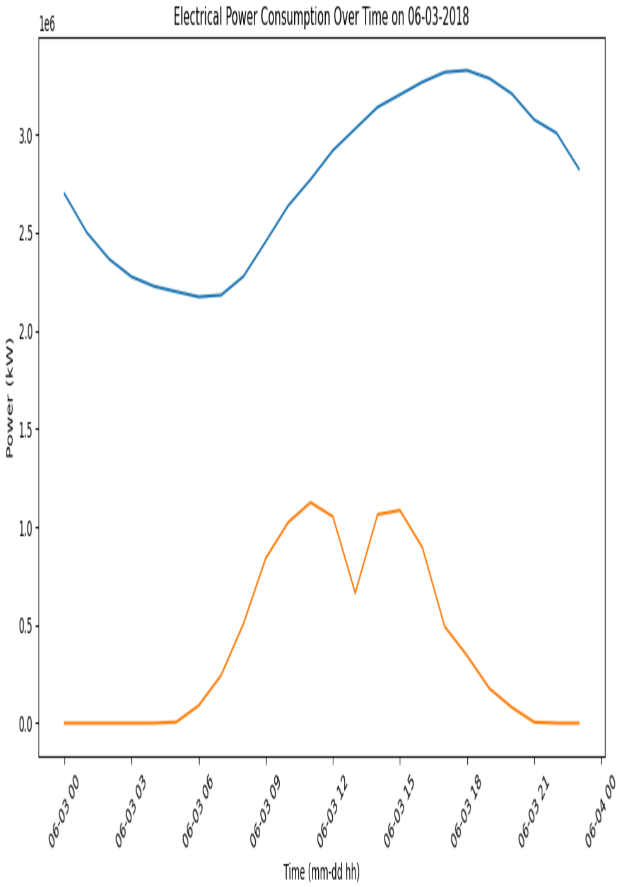
```
kWh

date

2018-06-03 00:00:00     0.0

2018-06-03 01:00:00     0.0

2018-06-03 02:00:00     0.0

2018-06-03 03:00:00     0.0

2018-06-03 04:00:00     0.0

2018-06-03 05:00:00     5364.0

2018-06-03 06:00:00     89479.0

2018-06-03 07:00:00     243391.0

2018-06-03 08:00:00     506163.0

2018-06-03 09:00:00     840801.0

2018-06-03 10:00:00     1022944.0

2018-06-03 11:00:00 1124456.0

2018-06-03 12:00:00     1052887.0

2018-06-03 13:00:00     662972.0
```

```
2018-06-03 14:00:00    1063243.0

2018-06-03 15:00:00    1084414.0

2018-06-03 16:00:00    895397.0

2018-06-03 17:00:00    491990.0

2018-06-03 18:00:00    343648.0

2018-06-03 19:00:00    176110.0

2018-06-03 20:00:00    80177.0

2018-06-03 21:00:00    4680.0

2018-06-03 22:00:00    0.0

2018-06-03 23:00:00    0.0
```

```
energy_demand = ((consumption_readings_for_date['AEP_MW']-production_readings_for_date["kWh"]).mean())/1e6

total = (consumption_readings_for_date['AEP_MW'].mean())/1e6

f"Energy Demand Unfilled by Solar Installation on {target_date.date()}: {round(energy_demand,2)} GWh ({round(energy_demand/total,3)*100}% of demand)"
```

```
'Energy Demand Unfilled by Solar Installation on 2018-06-03: 2.36 GWh (85.39999999999999% of demand)'
```

```
import matplotlib.pyplot as plt

# Plot datetime against energy consumption

plt.figure(figsize=(10, 6))  # Adjust the figure size as needed

plt.plot(consumption_readings_for_date.index, consumption_readings_for_date['AEP_MW'], production_readings_for_date.index, production_readings_for_date["kWh"])

plt.xlabel('Time (mm-dd hh)')

plt.ylabel('Power (kW)')

plt.title(f'Electrical Power Consumption Over Time on {datetime.strftime(target_date,"%m-%d-%Y")}')

plt.xticks(rotation=45)  # Rotate x-axis labels for better visibility

plt.tight_layout()
plt.show()
```



Supercharging Sustainability: Harnessing Superconductors for Renewable Energy Innovation

Part 1: Introduction

The looming danger of intensified climate change, driven by the relentless emission of greenhouse gases from fossil fuel-based electricity generation, underscores the urgent need to transition towards sustainable energy sources swiftly. The potential discovery of a room-temperature superconductor, LK-99, by South Korean researchers provides the opportunity for Superconducting Magnetic Energy Storage [SMES] which was invented in 1970. This technology is meant to effectively store energy through induction where electrons flow continuously within a superconducting coil wrapped around a core usually made from a niobium-titanium alloy embedded in a copper matrix. The primary benefits of SMES over other methods include nearly instantaneous power availability with a minimal time delay during charge and discharge, low power loss due to negligible electrical resistance, and enhanced reliability as its stationary components contribute to system stability. However, SMES systems can face increased complexity and operational cost due to their dependence on cryogenic cooling to maintain the superconducting state of the superconducting coil. The chance of LK-99‘s use in such a system remains low due to current research indicating a limited critical current of 250 mA (the maximum current where R=0) and a constrained maximum magnetic field threshold for maintaining superconductivity.

The primary objective of this notebook is to identify the plausibility of creating a solar battery that, when combined with a solar panel installation or any other renewable energy source, can sustain the energy consumption of an ideal American city. In order to do this, the data analysis algorithms will be employed for prediction and optimization

Part 1.1 Aurora City: Pioneering Sustainability in a Vibrant Metropolis

Let's begin by imagining we are in Aurora City with a population of 1,024,144 people, and is a bustling metropolis known for its vibrant cultural scene, modern architecture, and diverse population. The city boasts a mix of historic neighborhoods and futuristic skyscrapers, offering a unique blend of old-world charm and contemporary innovation. With a strong emphasis on sustainability, Aurora City is home to lush parks, efficient public transportation, and a commitment to renewable energy sources. Its world-class universities, thriving tech industry, and rich culinary scene make it a hub of creativity and opportunity. In line with that commitment to sustainability, the city's government has pledges to move towards 100% solar electricity generation within the next 10 years.

```
!pip install prophet

!pip install cplex

!pip install stable_baselines3

import numpy as np

import pandas as pd

import random

from datetime import datetime, timedelta

import math

import tensorflow as tf

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from prophet import Prophet
from scipy.optimize import minimize
import pulp
import matplotlib.pyplot as plt
import statistics
import sys
from stable_baselines3 import PPO
from stable_baselines3.common.env_util import make_vec_env
import gymnasium as gym
from gymnasium import spaces
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/ops/__init__.py:98: UserWarning: unable to load libtensorflow_io_plugins.so: unable to open file: libtensorflow_io_plugins.so, from paths: ['/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/ops/libtensorflow_io_plugins.so']
caused by: ['/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/ops/libtensorflow_io_plugins.so: undefined symbol: _ZN3tsl6StatusC1EN10tensorflow5error4CodeESt17basic_string_viewIcSt11char_traitsIcEENS_14SourceLocationE']
  warnings.warn(f"unable to load libtensorflow_io_plugins.so: {e}")
/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/ops/__init__.py:104: UserWarning: file system plugins are not loaded: unable to open file: libtensorflow_io.so, from paths: ['/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/ops/libtensorflow_io.so']
caused by: ['/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/ops/libtensorflow_io.so: undefined symbol: _ZTVN10tensorflow13GcsFileSystemE']
```

```
    warnings.warn(f"file system plugins are not loaded: {e}")

def get_readings_for_date(csv_file_path,
target_date,time_column,energy_column):

        # Read the CSV file into a DataFrame

        df = pd.read_csv(csv_file_path)

        df[time_column] = pd.to_datetime(df[time_column])

        df['filter_date'] = df[time_column].dt.date

        filtered_df = df[df['filter_date'] == target_date]


        return filtered_df[[time_column,energy_column]]

def generate_random_date(delim):


        start_date = datetime.strptime('2015-9-1', '%Y-%m-%d')

        end_date = datetime.strptime('2018-8-2', '%Y-%m-%d')


        days_difference = (end_date - start_date).days

        random_days = random.randint(0, days_difference)


        random_date = (start_date + timedelta(days=random_days)).date()

        return
f"{random_date.year}{delim}{random_date.month}{delim}{random_date.day}
"

target_date = pd.to_datetime(generate_random_date("/"))

month = target_date.month

target_date

Timestamp('2018-06-03 00:00:00')
```

Part 2: Data Cleaning & Analysis

Part 2.1: Energizing Change: The Quest for Sustainable Power in Aurora City

To pave the way for a sustainable energy transformation, the city government assembles a team of experts, Solar Nexus Taskforce (SNT), tasked with crafting a viable strategy. Their objective was to orchestrate the reduction of the city's conventional power plants while simultaneously bolstering the deployment of solar panels and SMES solar batteries.

With determination, SNT promptly swings into action. The first step on their journey is to gather 2-3 years worth of energy consumption data from the city's power grid and energy production data from an existing solar field that will be undergoing a substantial expansion in the coming months.

```
csv_file_path = '/kaggle/input/solar-energy-production/Solar_Energy_Production.csv' # Solar energy production data sourced from solar installations in Calgary, Alberta

def get_production(date):

    production_readings_for_date = get_readings_for_date(csv_file_path, date.date(),'date','kWh').groupby('date')['kWh'].sum().reset_index()

    production_readings_for_date["kWh"] = production_readings_for_date["kWh"]*1000 # To account for a major scale-up in functionality

    return production_readings_for_date

production_readings_for_date = get_production(target_date)

consumption_csv_file_path = '/kaggle/input/hourly-energy-consumption/AEP_hourly.csv'

# Electrical energy comsumption data from American Electric Power Company, Inc (AEP) which is among the nation's largest generators of electricity, owning nearly 38,000 megawatts of generating capacity in the U.S and serving nearly 5 million customers.

def get_consumption(date):

    consumption_readings_for_date = get_readings_for_date(consumption_csv_file_path, date.date(),'Datetime','AEP_MW').sort_values(by='Datetime')


consumption_readings_for_date["AEP_MW"]=consumption_readings_for_date["AEP_MW"]*200

    # converts MW to kW and also scales down the data by a factor of 5 to make
```

```
more representative of the demands of a city the size of    Auroa City

    # Set the 'datetime' column as the index

    consumption_readings_for_date.set_index('Datetime', inplace=True)

    return consumption_readings_for_date

consumption_readings_for_date = get_consumption(target_date)

production_readings_for_date.set_index("date", inplace=True)


# Adding in zeroes for the times not included with the assumption that those
were times when the sun was not up

new_row = {"kWh":0}

for date in consumption_readings_for_date.index:

    if date not in production_readings_for_date.index:

        production_readings_for_date.loc[date] = new_row


production_readings_for_date =
production_readings_for_date.sort_values(by='date')

production_readings_for_date
```

```
kWh

date

2018-06-03 00:00:00    0.0

2018-06-03 01:00:00    0.0

2018-06-03 02:00:00    0.0

2018-06-03 03:00:00    0.0

2018-06-03 04:00:00    0.0

2018-06-03 05:00:00    5364.0

2018-06-03 06:00:00    89479.0

2018-06-03 07:00:00    243391.0

2018-06-03 08:00:00    506163.0
```

```
2018-06-03 09:00:00	840801.0
2018-06-03 10:00:00	1022944.0
2018-06-03 11:00:00 1124456.0
2018-06-03 12:00:00	1052887.0
2018-06-03 13:00:00	662972.0
2018-06-03 14:00:00	1063243.0
2018-06-03 15:00:00	1084414.0
2018-06-03 16:00:00	895397.0
2018-06-03 17:00:00	491990.0
2018-06-03 18:00:00	343648.0
2018-06-03 19:00:00	176110.0
2018-06-03 20:00:00	80177.0
2018-06-03 21:00:00	4680.0
2018-06-03 22:00:00	0.0
2018-06-03 23:00:00	0.0
```

```
energy_demand = ((consumption_readings_for_date['AEP_MW']-production_readings_for_date["kWh"]).mean())/1e6
total = (consumption_readings_for_date['AEP_MW'].mean())/1e6
f"Energy Demand Unfilled by Solar Installation on {target_date.date()}: {round(energy_demand,2)} GWh ({round(energy_demand/total,3)*100}% of demand)"
```

```
'Energy Demand Unfilled by Solar Installation on 2018-06-03: 2.36 GWh (85.39999999999999% of demand)'
```

```
import matplotlib.pyplot as plt
# Plot datetime against energy consumption
plt.figure(figsize=(10, 6))  # Adjust the figure size as needed
plt.plot(consumption_readings_for_date.index, consumption_readings_for_date['AEP_MW'], production_readings_for_date.index, production_readings_for_date["kWh"])
plt.xlabel('Time (mm-dd hh)')
plt.ylabel('Power (kW)')

plt.title(f'Electrical Power Consumption Over Time on {datetime.strftime(target_date,"%m-%d-%Y")}')

plt.xticks(rotation=45)   # Rotate x-axis labels for better visibility

plt.tight_layout()

plt.show()
```

Part 2.2: Unveiling Challenges: The City's Energy Quest Confronts Shadows

Amid caffeine-fueled calculations and simulations, a stark truth emerges from the data scientists' endeavors. A formidable challenge looms over the team. Within the conference room, a somber air settles as the lead data scientist projects slide after slide of graphs and statistics that each echoes a disheartening tale, such as those above. Despite the expansion of solar installation, the city's energy demands seem poised to surpass the solar yield. This disconcerting disparity casts a shadow over the team's ambitions.

With sleeves rolled up, the team ventures deep into the night, engrossed in brainstorming sessions. Yet, they soon end up going in circles. Recognizing the need for clarity, the director makes a call to dismiss the team so they can rest and return with renewed creativity. With crushed and weary spirits, they go home uncertain about the fate of a project that had once shown so much promise.

Part 3: Energy Forecasting & Optimization

Part 3.1: Renewed Resolve: Crafting a Dynamic Energy Strategy for Aurora City's Future

With the dawn of a new day comes hours of deliberation, leading to a consensus that they will stay the course, but with a more powerful battery to bridge the gap and the traditional power plants will still be able to pick up any remaining slack. Overcoming that challenge renews the team's faith in the project. They are ready to move to the next stage of the process which is working out how the solar installation and battery will handle the gradual buildup in electrical power demand that will result as the city phases out electricity from traditional power plants. The team begins working on a machine learning model to take into account the season, time of day in hours, and battery's charge level to create a dynamic strategy to handle the load of the city's electricity demands.

Part 3.2: Entangled Choices: Navigating Neural Network Dilemmas in Energy Prediction

Regrettably, the team encounters another impasse, this time stemming from a difference in opinion about which tool to use. On one side are the data scientists, who were in favor of employing a technology known as Long Short-Term Memory (LSTM) neural networks. They believe these networks held the key to comprehending the complex puzzle of electricity distribution patterns.

The data scientists argue that LSTMs are like seasoned detectives of time. They can piece together patterns across various time frames – whether it is the daily rhythm, changing seasons, or even the subtle dance between different time intervals. This skill makes LSTMs an ideal choice to predict electricity distribution, as it depends on a delicate balance of many variables, like the time of day, the battery's energy, and the electricity demand.

On the opposing side, the software developers advocate for Gated Recurrent Unit (GRU) neural networks. They see these networks as intelligent and efficient navigators of data. "GRUs are like streamlined drivers that ensured a smooth ride even through intricate datasets, helping to keep the project on track and within budget" explained an impassioned junior developer.

As spirited debates ensue, the team's director steps in, recognizing merit in both arguments. After careful reflection, the director leans towards the LSTM approach, appreciating its prowess in capturing time-driven intricacies. Believing that LSTMs could equip the team with superior insights for accurate electricity distribution forecasts, the director's decision was made. The software developers, albeit begrudgingly, collaborate with the data science team to construct and train the LSTM neural network.

Yet, just as discussions settled, a new voice emerged. Inspired by a recent tech seminar, one of the team's interns proposes harnessing the predictive power of Prophet from Facebook. The intern explains that Prophet is a forecasting tool that excels at predicting time-series data with multiple patterns and trends. Its ability to automatically detect seasonality and handle missing data makes it well-suited for predicting electricity distribution patterns. The director, open to innovation, reweighs the options and ultimately embraces the intern's suggestion. With fresh hope, the team embarks on a new phase – navigating the uncharted waters of energy forecasting using a novel tool.

```
df = pd.read_csv(consumption_csv_file_path)

df

Datetime    AEP_MW

0    2004-12-31 01:00:00 13478.0

1    2004-12-31 02:00:00     12865.0

2    2004-12-31 03:00:00     12577.0

3    2004-12-31 04:00:00     12517.0

4    2004-12-31 05:00:00     12670.0

...    ...    ...

121268  2018-01-01 20:00:00 21089.0

121269  2018-01-01 21:00:00 20999.0

121270  2018-01-01 22:00:00 20820.0

121271  2018-01-01 23:00:00 20415.0

121272  2018-01-02 00:00:00     19993.0

121273 rows × 2 columns
```

```
consumption_df = pd.DataFrame()

consumption_df["ds"]=df["Datetime"]

consumption_df["y"]=df["AEP_MW"]*200

consumption_df["y"].max()

5139000.0

p_df = pd.read_csv(csv_file_path)

production_df = pd.DataFrame()

production_df["ds"] = p_df["date"]

production_df["y"]= p_df["kWh"]*1000

production_df

ds  y
```

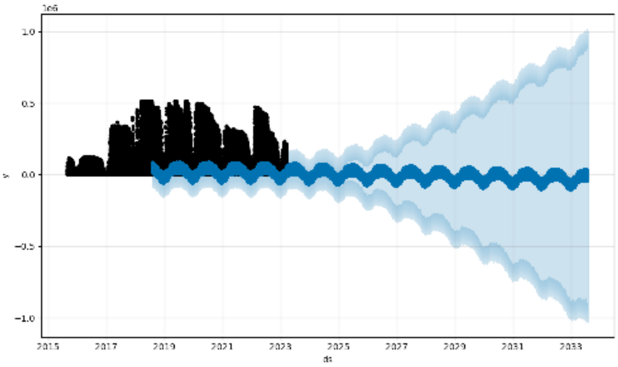
```
0    2017/09/11 08:00:00 AM 1130.0
1    2017/09/11 09:00:00 AM 2340.0
2    2017/09/11 10:00:00 AM 3656.0
3    2017/09/11 11:00:00 AM  4577.0
4    2017/09/11 12:00:00 PM  6506.0
...    ...    ...
258418  2023/03/12 03:00:00 PM      201285.0
258419  2023/03/12 04:00:00 PM      162582.0
258420 2023/03/12 05:00:00 PM      107060.0
258421  2023/03/12 06:00:00 PM      43074.0
258422 2023/03/12 07:00:00 PM      1788.0
258423 rows × 2 columns
```

```
c_model = Prophet()
c_model.fit(consumption_df)
future = c_model.make_future_dataframe(periods=131424, freq="H",
include_history=False)
future
```

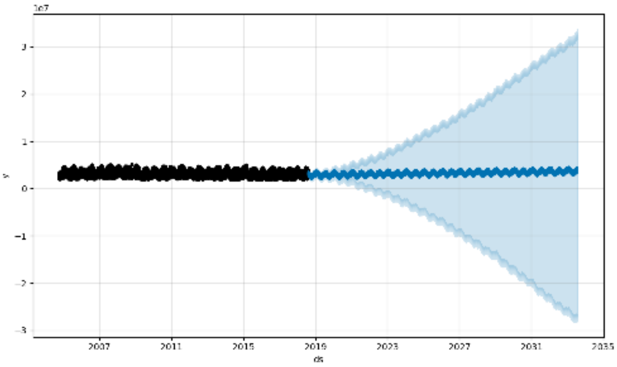
```
21:26:27 - cmdstanpy - INFO - Chain [1] start processing
21:27:58 - cmdstanpy - INFO - Chain [1] done processing
ds
0    2018-08-03 01:00:00
1    2018-08-03 02:00:00
2    2018-08-03 03:00:00
3    2018-08-03 04:00:00
4    2018-08-03 05:00:00
...    ...
131419   2033-07-30 20:00:00
```

```
131420   2033-07-30 21:00:00

131421   2033-07-30 22:00:00

131422   2033-07-30 23:00:00

131423   2033-07-31 00:00:00

131424 rows × 1 columns
```

```
p_model = Prophet()

p_model.fit(production_df)

21:28:44 - cmdstanpy - INFO - Chain [1] start processing

21:29:58 - cmdstanpy - INFO - Chain [1] done processing

<prophet.forecaster.Prophet at 0x7ae6b0ccebf0>

production_forecast=p_model.predict(future)

fig2 = p_model.plot(production_forecast)
```



```
forecast = c_model.predict(future)

fig1 = c_model.plot(forecast)
```


1e7
y
ds