```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES
# TO THE CORRECT LOCATION (/kaggle/input) IN YOUR NOTEBOOK,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.

import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'wind-solar-electricity-production:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-sets%2F3570391%2F6217083%2Fbundle

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'), target_is_directory=True)
except FileExistsError:
  pass
try:
  os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'), target_is_directory=True)
except FileExistsError:
  pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfile:
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue

print('Data source import complete.')
```

```
# This Python 3 environment comes with many helpful analytics libraries installed
```

```
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & I
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

```
/kaggle/input/wind-solar-electricity-production/intermittent-renewables-production-france.csv
```

Aim: which month have higest Wind&Solar Power Production

```
#Importing The Required Libraries
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.feature_selection import SelectKBest
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
import warnings
warnings.filterwarnings("ignore")
from sklearn.metrics import classification_report,ConfusionMatrixDisplay
```

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

```
df=pd.read_csv("/kaggle/input/wind-solar-electricity-production/intermittent-renewables-production-france.csv")
df
```

	Date and Hour	Date	StartHour	EndHour	Source	Production	dayOfYear	dayName	monthName
0	2020-07-22 20:00:00+02:00	2020-07-22	20:00:00	21:00:00	Solar	244.0	204	Wednesday	July
1	2020-07-23 07:00:00+02:00	2020-07-23	07:00:00	08:00:00	Solar	223.0	205	Thursday	July
2	2020-07-23 16:00:00+02:00	2020-07-23	16:00:00	17:00:00	Solar	2517.0	205	Thursday	July
3	2020-07-23 19:00:00+02:00	2020-07-23	19:00:00	20:00:00	Solar	658.0	205	Thursday	July
4	2020-07-23 23:00:00+02:00	2020-07-23	23:00:00	24:00:00	Solar	0.0	205	Thursday	July
...	...	...	...	...	...	...	...	...	...
59801	2023-06-30 06:00:00+02:00	2023-06-30	06:00:00	07:00:00	Solar	55.0	181	Friday	June
59802	2023-06-30 13:00:00+02:00	2023-06-30	13:00:00	14:00:00	Solar	4554.0	181	Friday	June
59803	2023-06-30 14:00:00+02:00	2023-06-30	14:00:00	15:00:00	Solar	4589.0	181	Friday	June
59804	2023-06-30 16:00:00+02:00	2023-06-30	16:00:00	17:00:00	Solar	4173.0	181	Friday	June
59805	2023-06-30 18:00:00+02:00	2023-06-30	18:00:00	19:00:00	Solar	2404.0	181	Friday	June

59806 rows × 9 columns

PREPROCESSING

```
df.isna().sum()
```

```
Date and Hour    0
Date             0
StartHour        0
EndHour          0
Source           0
Production       2
dayOfYear        0
dayName          0
monthName        0
dtype: int64
```

```
df.dropna(inplace=True)
```

```
df.dtypes
```

```
Date and Hour     object
Date              object
StartHour         object
EndHour           object
Source            object
Production       float64
dayOfYear          int64
dayName           object
monthName         object
dtype: object
```

```
df.drop("Date and Hour",axis=1,inplace=True)
df.drop("Date",axis=1,inplace=True)
```

```
df['StartHour'] = df['StartHour'].replace('24:00:00', '00:00:00')
df['EndHour'] = df['EndHour'].replace('24:00:00', '00:00:00')

# Step 3: Convert the time columns to pandas datetime objects
df['StartHour'] = pd.to_datetime(df['StartHour'])
df['EndHour'] = pd.to_datetime(df['EndHour'])

# Step 4: Perform the subtraction operation and calculate the time difference
# Replace 'new_column_name' with the desired name for the new column
df['TimeDifference'] = df['EndHour'] - df['StartHour']
```

```
df
```

	StartHour	EndHour	Source	Production	dayOfYear	dayName	monthName	TimeDifference
0	2024-04-04 20:00:00	2024-04-04 21:00:00	Solar	244.0	204	Wednesday	July	0 days 01:00:00
1	2024-04-04 07:00:00	2024-04-04 08:00:00	Solar	223.0	205	Thursday	July	0 days 01:00:00
2	2024-04-04 16:00:00	2024-04-04 17:00:00	Solar	2517.0	205	Thursday	July	0 days 01:00:00
3	2024-04-04 19:00:00	2024-04-04 20:00:00	Solar	658.0	205	Thursday	July	0 days 01:00:00
4	2024-04-04 23:00:00	2024-04-04 00:00:00	Solar	0.0	205	Thursday	July	-1 days +01:00:00
...	...	...	...	...	...	...	...	...
59801	2024-04-04 06:00:00	2024-04-04 07:00:00	Solar	55.0	181	Friday	June	0 days 01:00:00
59802	2024-04-04 13:00:00	2024-04-04 14:00:00	Solar	4554.0	181	Friday	June	0 days 01:00:00
59803	2024-04-04 14:00:00	2024-04-04 15:00:00	Solar	4589.0	181	Friday	June	0 days 01:00:00
59804	2024-04-04 16:00:00	2024-04-04 17:00:00	Solar	4173.0	181	Friday	June	0 days 01:00:00
59805	2024-04-04 18:00:00	2024-04-04 19:00:00	Solar	2404.0	181	Friday	June	0 days 01:00:00

59804 rows × 8 columns

```
df['Total_time'] = df['TimeDifference'].dt.components['hours']
df = df.drop(['TimeDifference', 'StartHour', 'EndHour'], axis=1)
```

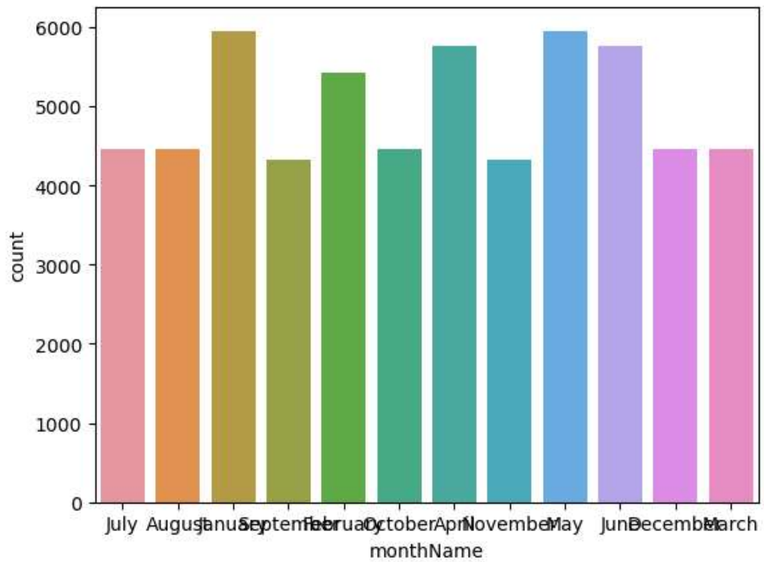
```
df.dtypes
```

```
Source          object
Production     float64
dayOfYear        int64
dayName         object
monthName       object
Total_time       int64
dtype: object
```

VISUVALIZATIONS

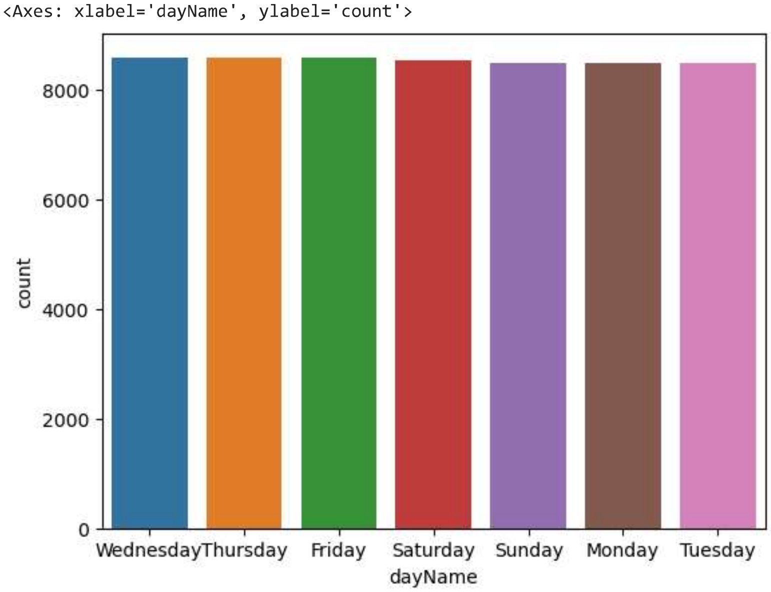
```
sns.countplot(x='monthName',data=df)
```

```
<Axes: xlabel='monthName', ylabel='count'>
```




```
sns.countplot(x='dayName',data=df)
```

```
<Axes: xlabel='dayName', ylabel='count'>
```




```
lst=["Source","dayName","monthName"]
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
for i in lst:
    df[i]=le.fit_transform(df[i])
```

```
x=df.drop(["monthName"],axis=1)
```

```
x=df.drop(['monthName'],axis=1)
x
```

	Source	Production	dayOfYear	dayName	Total_time
0	0	244.0	204	6	1
1	0	223.0	205	4	1
2	0	2517.0	205	4	1
3	0	658.0	205	4	1
4	0	0.0	205	4	1
...	...	...	...	...	...
59801	0	55.0	181	0	1
59802	0	4554.0	181	0	1
59803	0	4589.0	181	0	1
59804	0	4173.0	181	0	1
59805	0	2404.0	181	0	1

59804 rows × 5 columns

```
y=df.iloc[:,-2]
y
```

```
0        5
1        5
2        5
3        5
4        5
        ..
59801    6
59802    6
59803    6
59804    6
59805    6
Name: monthName, Length: 59804, dtype: int64
```

```
#scaling using standard scaler
ms=MinMaxScaler()
X_ms=ms.fit_transform(x)
X_ms
```

```
array([[0.        , 0.01416957, 0.55616438, 1.        , 0.        ],
       [0.        , 0.01295006, 0.55890411, 0.66666667, 0.        ],
       [0.        , 0.14616725, 0.55890411, 0.66666667, 0.        ],
       ...,
       [0.        , 0.26649245, 0.49315068, 0.        , 0.        ],
       [0.        , 0.24233449, 0.49315068, 0.        , 0.        ],
       [0.        , 0.13960511, 0.49315068, 0.        , 0.        ]])
```

```
#Performing train_test_split
X_train,X_test,y_train,y_test=train_test_split(X_ms,y,test_size=0.2,random_state=0)
```

MODEL CREATION

KNN

```
#KNN
knn=KNeighborsClassifier()
knn.fit(X_train,y_train)
y_pred=knn.predict(X_test)
y_pred
```

```
array([ 2,  3,  6, ...,  5, 10, 11])
```

```
knn1=KNeighborsClassifier(algorithm='auto',n_neighbors=9,weights='distance')
knn1.fit(X_train,y_train)
y_pred1=knn1.predict(X_test)
print(classification_report(y_test,y_pred1))
```

```
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      1180
           1       0.98      0.98      0.98       871
           2       0.99      0.99      0.99       867
           3       0.97      0.98      0.98      1034
           4       0.99      0.99      0.99      1240
           5       0.98      0.97      0.98       874
           6       0.98      0.98      0.98      1195
           7       0.98      0.97      0.97       921
           8       0.98      0.98      0.98      1146
           9       0.97      0.98      0.97       893
          10       0.97      0.97      0.97       872
          11       0.98      0.98      0.98       868

    accuracy                           0.98     11961
   macro avg       0.98      0.98      0.98     11961
weighted avg       0.98      0.98      0.98     11961
```

SVC

```
sv=SVC(C=10, gamma =1, kernel= 'rbf')
sv.fit(X_train,y_train)
y_pred2=sv.predict(X_test)
y_pred2
```

```
array([ 2,  3,  6, ...,  5, 10, 11])
```

```
print(classification_report(y_test,y_pred2))
```

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      1180
           1       0.96      0.99      0.98       871
           2       0.99      1.00      0.99       867
           3       0.98      0.98      0.98      1034
           4       0.99      0.99      0.99      1240
           5       1.00      0.97      0.98       874
           6       0.97      0.99      0.98      1195
           7       0.99      0.97      0.98       921
           8       0.99      0.98      0.99      1146
           9       1.00      0.98      0.99       893
          10       0.98      0.99      0.98       872
          11       0.98      0.97      0.97       868

    accuracy                           0.98     11961
   macro avg       0.98      0.98      0.98     11961
weighted avg       0.98      0.98      0.98     11961
```

GaussianNB

```
nb=GaussianNB()
nb.fit(X_train,y_train)
y_pred2=nb.predict(X_test)
y_pred2
```

```
array([ 2,  3,  6, ...,  5, 10, 11])
```

```
print(classification_report(y_test,y_pred2))
```

```
              precision    recall  f1-score   support

           0       0.98      0.97      0.98      1180
           1       0.99      0.99      0.99       871
           2       1.00      0.98      0.99       867
           3       0.97      1.00      0.99      1034
           4       1.00      1.00      1.00      1240
           5       0.99      0.97      0.98       874
           6       0.96      0.98      0.97      1195
           7       0.97      0.95      0.96       921
```

```
           8       0.97      0.98      0.98      1146
           9       0.98      0.98      0.98       893
          10       0.97      0.99      0.98       872
          11       0.99      0.97      0.98       868

    accuracy                           0.98     11961
   macro avg       0.98      0.98      0.98     11961
weighted avg       0.98      0.98      0.98     11961
```

Decision Tree Classifier

```
dt=DecisionTreeClassifier(criterion='entropy',random_state=2,max_depth=10)
dt.fit(X_train,y_train)
y_pred3=dt.predict(X_test)
y_pred3
```

```
array([ 2,  3,  6, ...,  5, 10, 11])
```

```
print(classification_report(y_test,y_pred3))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1180
           1       1.00      1.00      1.00       871
           2       1.00      1.00      1.00       867
           3       1.00      1.00      1.00      1034
           4       1.00      1.00      1.00      1240
           5       1.00      1.00      1.00       874
           6       1.00      1.00      1.00      1195
           7       1.00      1.00      1.00       921
           8       1.00      1.00      1.00      1146
           9       1.00      1.00      1.00       893
          10       1.00      1.00      1.00       872
          11       1.00      1.00      1.00       868

    accuracy                           1.00     11961
   macro avg       1.00      1.00      1.00     11961
weighted avg       1.00      1.00      1.00     11961
```

Random Forest Classifier

```
rf=RandomForestClassifier(criterion= 'entropy', max_depth= None, min_samples_leaf= 1, min_samples_split= 4,n_estimators= 200)
rf.fit(X_train,y_train)
y_pred4=rf.predict(X_test)
y_pred4
```

```
array([ 2,  3,  6, ...,  5, 10, 11])
```

```
print(classification_report(y_test,y_pred4))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1180
           1       1.00      1.00      1.00       871
           2       1.00      1.00      1.00       867
           3       1.00      1.00      1.00      1034
           4       1.00      1.00      1.00      1240
           5       1.00      1.00      1.00       874
           6       1.00      1.00      1.00      1195
           7       1.00      1.00      1.00       921
           8       1.00      1.00      1.00      1146
           9       1.00      1.00      1.00       893
          10       1.00      1.00      1.00       872
          11       1.00      1.00      1.00       868

    accuracy                           1.00     11961
   macro avg       1.00      1.00      1.00     11961
weighted avg       1.00      1.00      1.00     11961
```

XG BOOST Classifier

```
xgb=XGBClassifier()
xgb.fit(X_train,y_train)
y_pred7=xgb.predict(X_test)
y_pred7
```

```
array([ 2,  3,  6, ...,  5, 10, 11])
```

```
print(classification_report(y_test,y_pred7))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1180
           1       1.00      1.00      1.00       871
           2       1.00      1.00      1.00       867
           3       1.00      1.00      1.00      1034
           4       1.00      1.00      1.00      1240
           5       1.00      1.00      1.00       874
           6       1.00      1.00      1.00      1195
           7       1.00      1.00      1.00       921
           8       1.00      1.00      1.00      1146
           9       1.00      1.00      1.00       893
          10       1.00      1.00      1.00       872
          11       1.00      1.00      1.00       868

    accuracy                           1.00     11961
   macro avg       1.00      1.00      1.00     11961
weighted avg       1.00      1.00      1.00     11961
```

HIGEST ACCURACY IS 1 in **decition tree,xg boost and Random forest**