Date-01/11/2023

Team ID-718

Project Title-Website Traffic Analysis using Data Analytics

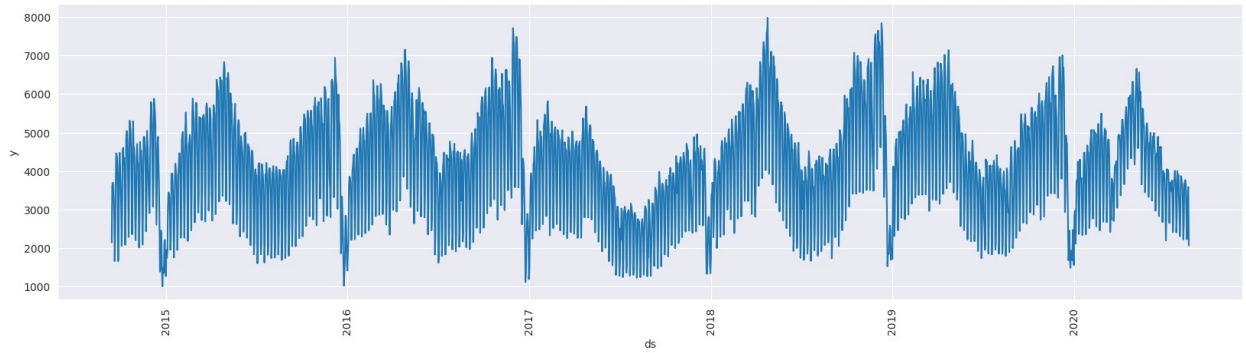
```
import os
import warnings
import random
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from datetime import timedelta

warnings.filterwarnings('ignore')
sns.set_style('darkgrid')

random.seed(42)
np.random.seed(42)
os.environ['PYTHONHASHSEED'] = str(42)

class suppress_stdout_stderr(object):
    """
    A context manager for doing a "deep suppression" of stdout and
stderr in
    Python, i.e. will suppress all print, even if the print originates
in a
    compiled C/Fortran sub-function.
       This will not suppress raised exceptions, since exceptions are
printed
    to stderr just before a script exits, and after the context
manager has
    exited (at least, I think that is why it lets exceptions through).

    """

    def __init__(self):
        # Open a pair of null files
        self.null_fds = [os.open(os.devnull, os.O_RDWR) for x in
```

```
range(2)]
        # Save the actual stdout (1) and stderr (2) file descriptors.
        self.save_fds = (os.dup(1), os.dup(2))

    def __enter__(self):
        # Assign the null pointers to stdout and stderr.
        os.dup2(self.null_fds[0], 1)
        os.dup2(self.null_fds[1], 2)

    def __exit__(self, *_):
        # Re-assign the real stdout/stderr back to (1) and (2)
        os.dup2(self.save_fds[0], 1)
        os.dup2(self.save_fds[1], 2)
        # Close the null files
        os.close(self.null_fds[0])
        os.close(self.null_fds[1])

data = pd.read_csv('/content/archive (7).zip')

data.head()
```

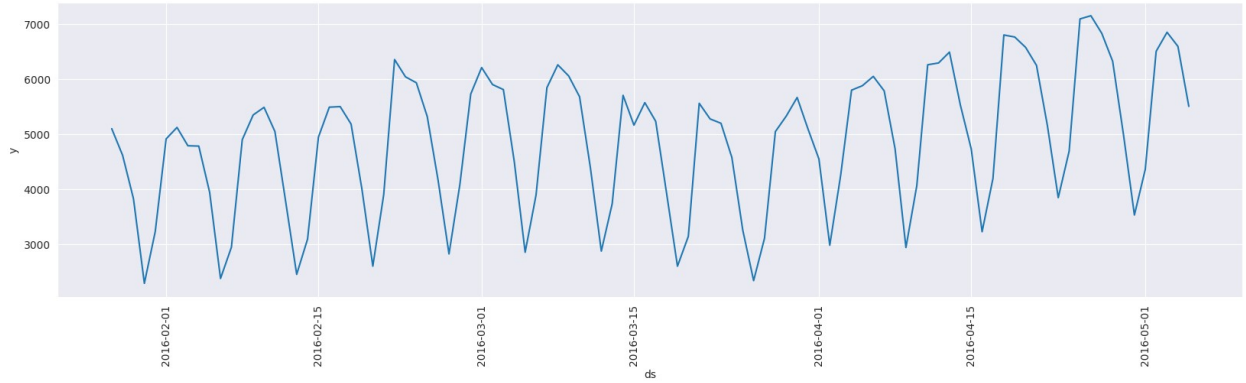
	Row	Day	Day.Of.Week	Date	Page.Loads	Unique.Visits	\
0	1	Sunday	1	9/14/2014	2,146	1,582	
1	2	Monday	2	9/15/2014	3,621	2,528	
2	3	Tuesday	3	9/16/2014	3,698	2,630	
3	4	Wednesday	4	9/17/2014	3,667	2,614	
4	5	Thursday	5	9/18/2014	3,316	2,366	

	First.Time.Visits	Returning.Visits
0	1,430	152
1	2,297	231
2	2,352	278
3	2,327	287
4	2,130	236

```
data = data[['Date', 'Page.Loads']]
data['ds'] = pd.to_datetime(data['Date'])
data = data.drop('Date', axis=1)
data = data.rename(columns={'Page.Loads': 'y'})
data['y'] = data['y'].str.replace(',', '').astype(float)
data = data[['ds', 'y']]

plt.figure(figsize=(20, 5))
sns.lineplot(data=data, x='ds', y='y')
plt.xticks(rotation=90)
plt.show()
```




```
plt.figure(figsize=(20, 5))
sns.lineplot(data=data.iloc[500:600], x='ds', y='y')
plt.xticks(rotation=90)
plt.show()
```




```
ts = pd.date_range(start=data.ds.min(), end=data.ds.max(), freq='D')

len(ts) == len(data)

True

train_series = data[data.ds < (data.ds.max() -
timedelta(days=30))].copy()
test_series = data[data.ds >= (data.ds.max() -
timedelta(days=30))].copy()

def draw_predictions(data: pd.DataFrame) -> None:
    series = data.set_index('ds').unstack().reset_index(drop=False)
    series.columns = ['type', 'ds', 'value']

    plt.figure(figsize=(20, 5))
    sns.lineplot(data=series, x='ds', y='value', hue='type')
    plt.xticks(rotation=90)
    plt.show()
```

```
import prophet

from sklearn.metrics import mean_squared_error,
mean_absolute_percentage_error
from sklearn.model_selection import train_test_split

prophet.diagnostics.logging.disable(level=50)

fb = prophet.Prophet()

with suppress_stdout_stderr():
    fb.fit(train_series)

predictions = fb.make_future_dataframe(periods=len(test_series),
freq='D')
forecast = fb.predict(predictions)

forecast.head()
```

```
          ds        trend   yhat_lower   yhat_upper  trend_lower
trend_upper  \
0 2014-09-14  3321.815056  1454.416474  2670.465521  3321.815056
3321.815056
1 2014-09-15  3324.983907  3062.576858  4332.983263  3324.983907
3324.983907
2 2014-09-16  3328.152759  3228.569039  4483.500481  3328.152759
3328.152759
3 2014-09-17  3331.321610  3276.471527  4473.969106  3331.321610
3331.321610
4 2014-09-18  3334.490462  3044.393060  4245.000597  3334.490462
3334.490462

   additive_terms  additive_terms_lower  additive_terms_upper
weekly  \
0    -1283.358783          -1283.358783          -1283.358783 -
872.493065
1      372.291546            372.291546            372.291546
734.249390
2      531.827881            531.827881            531.827881
845.430212
3      520.806516            520.806516            520.806516
787.155469
4      314.040051            314.040051            314.040051
534.752192

   weekly_lower  weekly_upper      yearly  yearly_lower  yearly_upper
\
0   -872.493065   -872.493065 -410.865718   -410.865718   -410.865718

1    734.249390    734.249390 -361.957844   -361.957844   -361.957844
```

```
2    845.430212    845.430212 -313.602331   -313.602331   -313.602331

3    787.155469    787.155469 -266.348952   -266.348952   -266.348952

4    534.752192    534.752192 -220.712142   -220.712142   -220.712142


   multiplicative_terms  multiplicative_terms_lower  \
0                   0.0                         0.0
1                   0.0                         0.0
2                   0.0                         0.0
3                   0.0                         0.0
4                   0.0                         0.0

   multiplicative_terms_upper         yhat
0                         0.0  2038.456273
1                         0.0  3697.275453
2                         0.0  3859.980640
3                         0.0  3852.128127
4                         0.0  3648.530512
```

```
forecast.columns
```

```
Index(['ds', 'trend', 'yhat_lower', 'yhat_upper', 'trend_lower',
'trend_upper',
       'additive_terms', 'additive_terms_lower',
'additive_terms_upper',
       'weekly', 'weekly_lower', 'weekly_upper', 'yearly',
'yearly_lower',
       'yearly_upper', 'multiplicative_terms',
'multiplicative_terms_lower',
       'multiplicative_terms_upper', 'yhat'],
      dtype='object')
```

```
v_fb_df = test_series.copy()
v_fb_df = v_fb_df.merge(forecast[['ds', 'yhat']], on='ds', how='left')
```

```
v_fb_df.head()
```

```
          ds       y         yhat
0 2020-07-20  3749.0  3972.500083
1 2020-07-21  3786.0  4080.418696
2 2020-07-22  4002.0  4016.757792
3 2020-07-23  3823.0  3757.091685
4 2020-07-24  3430.0  2810.845350
```

```
np.sqrt(mean_squared_error(v_fb_df['y'], v_fb_df['yhat']))
```

```
514.2903167848739
```

```
mean_absolute_percentage_error(v_fb_df['y'], v_fb_df['yhat'])
```

```
0.13679515661319644

draw_predictions(v_fb_df)
```


```
gbt_data = train_series.merge(forecast, on='ds', how='left')
train_gbt, val_gbt = train_test_split(gbt_data, test_size=0.15,
random_state=42)

import re
import pandas as pd
from typing import Optional
from itertools import product
from tqdm import tqdm
from holidays.holiday_base import HolidayBase
from prophet import Prophet

class ProphetsEnsemble:
    """An ensemble of Prophet models with different aggregation
functions and frequencies."""

    def __init__(self, freq: str, levels: list, agg_fn: list,
holidays_getter: HolidayBase = None):
        """Initializes an ensemble of Prophet models."""
        self.freq = freq
        self.levels = ['_'.join(x) for x in product(levels, agg_fn)]
        self.h_getter = holidays_getter
        self.prophets_ = dict()
        self.is_fitted_ = False

    @staticmethod
    def _resample(data: pd.DataFrame, freq: str, how: str) ->
pd.DataFrame:
        """Resamples a time series DataFrame."""
        if how not in ['median', 'mean', 'sum']:
            raise NotImplementedError(f'Unknown function {how}. Only
[median, mean, sum] are supported.')
        return
```

```
data.set_index('ds').resample(freq).agg(how).reset_index(drop=False)

    @staticmethod
    def _merge_key_gen(x, level: str) -> str:
        """Generates a key for merging DataFrames based on the
frequency."""
        freq = re.sub('[\d]', '', level.split('_')[0])
        if freq == 'H':
            return f'{x.year}-{x.month}-{x.day}-{x.hour}'
        elif freq in ['D', 'M']:
            return f'{x.year}-{x.month}-{x.day}' if freq == 'D' else
f'{x.year}-{x.month}'
        elif freq == 'W':
            return f'{x.isocalendar().year}-{x.isocalendar().week}'
        raise NotImplementedError(f'Only [H, D, W, M] are supported.
{freq} was recieved as input!')

    def _get_holidays(self, data: pd.DataFrame) ->
Optional[pd.DataFrame]:
        """Extracts holidays from the data."""
        if self.h_getter is None:
            return None
        holidays = data[['ds']].copy()
        holidays['holiday'] = holidays['ds'].apply(self.h_getter.get)
        return holidays.dropna()

    def _fit_level(self, data: pd.DataFrame, level: str) -> None:
        """Fits a Prophet model for a specific aggregation level."""
        resampled = self._resample(data, *level.split('_')) if level !
= self.freq else data.copy()
        fb = Prophet(holidays=self._get_holidays(resampled))
        with suppress_stdout_stderr():
            fb.fit(resampled)
        self.prophets_[level] = fb

    def _predict_level(self, periods: int, level: str) ->
pd.DataFrame:
        """Makes predictions for a specific aggregation level."""
        fb = self.prophets_[level]
        df = fb.make_future_dataframe(periods=periods,
freq=level.split('_')[0])
        forecasts = fb.predict(df)
        forecasts.columns = [f'{x}_{level}' for x in
forecasts.columns]
        return forecasts

    def _combine_levels(self, base_df: pd.DataFrame, data:
pd.DataFrame, level: str) -> pd.DataFrame:
        """Combines predictions from different aggregation levels."""
        key = lambda x: self._merge_key_gen(x, level)
```

```
        return (
            base_df.assign(key=base_df['ds'].apply(key))
            .merge(data.assign(key=data[f'ds_{level}'].apply(key)),
on='key', how='left')
            .drop(['key', f'ds_{level}'], axis=1)
        )

    @staticmethod
    def _drop_redundant(data: pd.DataFrame) -> pd.DataFrame:
        """Drops redundant features from the DataFrame."""
        redundant = [col for col in data.columns if col != 'ds' and
'yhat' not in col and len(data[col].unique()) == 1]
        return data.drop(redundant, axis=1)

    def fit(self, data: pd.DataFrame) -> None:
        """Fits the Prophet models for all aggregation levels."""
        for level in tqdm([self.freq] + self.levels, 'Fitting
prophets...'):
            self._fit_level(data, level)
        self.is_fitted_ = True

    def forecast(self, periods: int) -> pd.DataFrame:
        """Makes forecasts for all aggregation levels and combines
them."""
        assert self.is_fitted_, 'Model is not fitted'
        forecasts = [self._predict_level(periods, level) for level in
tqdm([self.freq] + self.levels, 'Forecasting...')]

        forecast = forecasts[0].rename(columns={f'ds_{self.freq}':
'ds', f'yhat_{self.freq}': 'yhat'})
        for level, fore in zip(self.levels, forecasts[1:]):
            forecast = self._combine_levels(forecast, fore, level)

        return self._drop_redundant(forecast)

pe = ProphetsEnsemble(freq='D', levels=['W', 'M'], agg_fn=['median'])

pe.fit(train_series)

Fitting prophets...: 100%|██████████| 3/3 [00:00<00:00,  3.01it/s]

pe_forecast = pe.forecast(len(test_series))

Forecasting...: 100%|██████████| 3/3 [00:00<00:00,  4.50it/s]

gbt_data = train_series.merge(pe_forecast, on='ds', how='left')
train_gbt, val_gbt = train_test_split(gbt_data, test_size=0.15,
random_state=42)

draw_predictions(forecast_df)
```


value
4000
3500
3000
2500
2000
1500
type
y
yhat
2020-07-21
2020-07-25
2020-07-29
2020-08-01
2020-08-05
2020-08-09
2020-08-13
2020-08-17
ds