Website Traffic Time Series

Author: Aditya Srivastava
Date: Feb 25th 2022

Introduction

Often companies want to analyze the brand value they possess in the digital space or qunatify their online user engagement. One of the ways is by monitoring the traffic that their website produces, tracking growth based on new users, analyzing trend of existing users for retention and churn rate, uncovering certain trend patterns and forecasting the traffic that is supposed to follow to be better equiped for the future and take business along with computational decisions accordingly.
This project is based on analyzing the current trend and seasonalitites as well as predict the future traffic landings on a website from new users (first time visits) to quantify growth using machine learning time series model.

Data Source

The pageviews data is that of the website http://statforecasting.com/ of almost 6 years of data from 2014 to 2020. The data is publicly available and can be downloaded from Kaggle.

Data Dictionary

Column	Description	Data Type
Row	Row number	Integer
Day	Day of the week in text	String
Day.Of.Week	Day of the week in numeric	Integer
Date	Date in mm/dd/yyyy format	String
Page.Loads	Daily number of pages loaded	String
Unique.Visits	Daily number of visitors from whose IP addresses there haven't been hits on any page in over 6 hours	String
First.Time.Visits	Number of unique visitors who do not have a cookie identifying them as a previous customer	String
Returning.Visits	Number of unique visitors minus first time visitors	String

Understanding Data Structure

In [1]:

```
# Importing libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.formula.api import ols
from datetime import datetime
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.stattools import kpss
from statsmodels.tsa.seasonal import seasonal_decompose
import statsmodels.api as sm
from statsmodels.tsa.arima_model import ARIMA
import pmdarima as pm
import itertools
from prophet import Prophet
from prophet.plot import add_changepoints_to_plot, plot_cross_validation_metric
from prophet.diagnostics import cross_validation, performance_metrics
import warnings
import logging

logging.disable(logging.INFO)
warnings.filterwarnings('ignore')
%matplotlib inline
```

In [63]:

```
# reading data

df = pd.read_csv('daily-website-visitors.csv')
df.head()
```

Out[63]:

	Row	Day	Day.Of.Week	Date	Page.Loads	Unique.Visits	First.Time.Visits	Returning.Visits
0	1	Sunday	1	9/14/2014	2,146	1,582	1,430	152
1	2	Monday	2	9/15/2014	3,621	2,528	2,297	231
2	3	Tuesday	3	9/16/2014	3,698	2,630	2,352	278
3	4	Wednesday	4	9/17/2014	3,667	2,614	2,327	287
4	5	Thursday	5	9/18/2014	3,316	2,366	2,130	236

In [3]:

```
# structure of dataframe
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2167 entries, 0 to 2166
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Row                2167 non-null   int64
 1   Day                2167 non-null   object
 2   Day.Of.Week        2167 non-null   int64
 3   Date               2167 non-null   object
 4   Page.Loads         2167 non-null   object
 5   Unique.Visits      2167 non-null   object
 6   First.Time.Visits  2167 non-null   object
 7   Returning.Visits   2167 non-null   object
dtypes: int64(2), object(6)
memory usage: 135.6+ KB
```

There are 2167 rows and 8 columns where each row represents the number of pageviews and visitors on a daily basis.

Based on the structure of the dataframe, the following code removes redundant and no-value columns, changes data types of page metrics to integers and standardizes the column names.

In [64]:

```
# making structural changes for standardization

df.drop(['Row', 'Day.Of.Week'], axis = 1, inplace = True)
df = df.replace(',', '', regex = True)
columns = df.columns
metric_columns = columns[2:]
for column in metric_columns:
    df[column] = df[column].astype('int64')
df['Date'] = pd.to_datetime(df['Date'], format = '%m/%d/%Y')
df.rename(columns = {
    'Day': 'day',
    'Date': 'date',
    'Page.Loads': 'n_page_loads',
    'Unique.Visits': 'n_unique_visits',
    'First.Time.Visits': 'n_first_visits',
    'Returning.Visits': 'n_return_visits'
}, inplace = True)
```

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2167 entries, 0 to 2166
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   day              2167 non-null   object
 1   date             2167 non-null   datetime64[ns]
 2   n_page_loads     2167 non-null   int64
 3   n_unique_visits  2167 non-null   int64
 4   n_first_visits   2167 non-null   int64
 5   n_return_visits  2167 non-null   int64
dtypes: datetime64[ns](1), int64(4), object(1)
memory usage: 101.7+ KB
```

In [6]:

```
df.head()
```

Out[6]:

	day	date	n_page_loads	n_unique_visits	n_first_visits	n_return_visits
0	Sunday	2014-09-14	2146	1582	1430	152
1	Monday	2014-09-15	3621	2528	2297	231
2	Tuesday	2014-09-16	3698	2630	2352	278
3	Wednesday	2014-09-17	3667	2614	2327	287
4	Thursday	2014-09-18	3316	2366	2130	236

The data is cleaned and ready for further analysis.

In [7]:

```
# checking for missing values
df.isna().sum()
```

Out[7]:

```
day                0
date               0
n_page_loads       0
n_unique_visits    0
n_first_visits     0
n_return_visits    0
dtype: int64
```

No imputation is required as there were no missing values.

In [8]:

```
# getting date range
print(f"The data ranges from {df['date'].min().date()} to {df['date'].max().date()}")
```

```
The data ranges from 2014-09-14 to 2020-08-19
```

Following is the summary statistics of the dataframe.

In [9]:

```
# summary statistics
df.describe().T
```

Out[9]:

	count	mean	std	min	25%	50%	75%	max
n_page_loads	2167.0	4116.989386	1350.977843	1002.0	3114.5	4106.0	5020.5	7984.0
n_unique_visits	2167.0	2943.646516	977.886472	667.0	2226.0	2914.0	3667.5	5541.0
n_first_visits	2167.0	2431.824181	828.704688	522.0	1830.0	2400.0	3038.0	4616.0
n_return_visits	2167.0	511.822335	168.736370	133.0	388.5	509.0	626.5	1036.0

- The website mostly garners traffic from new users
- All of the metrics seem to be normally distributed as mean and median are very close

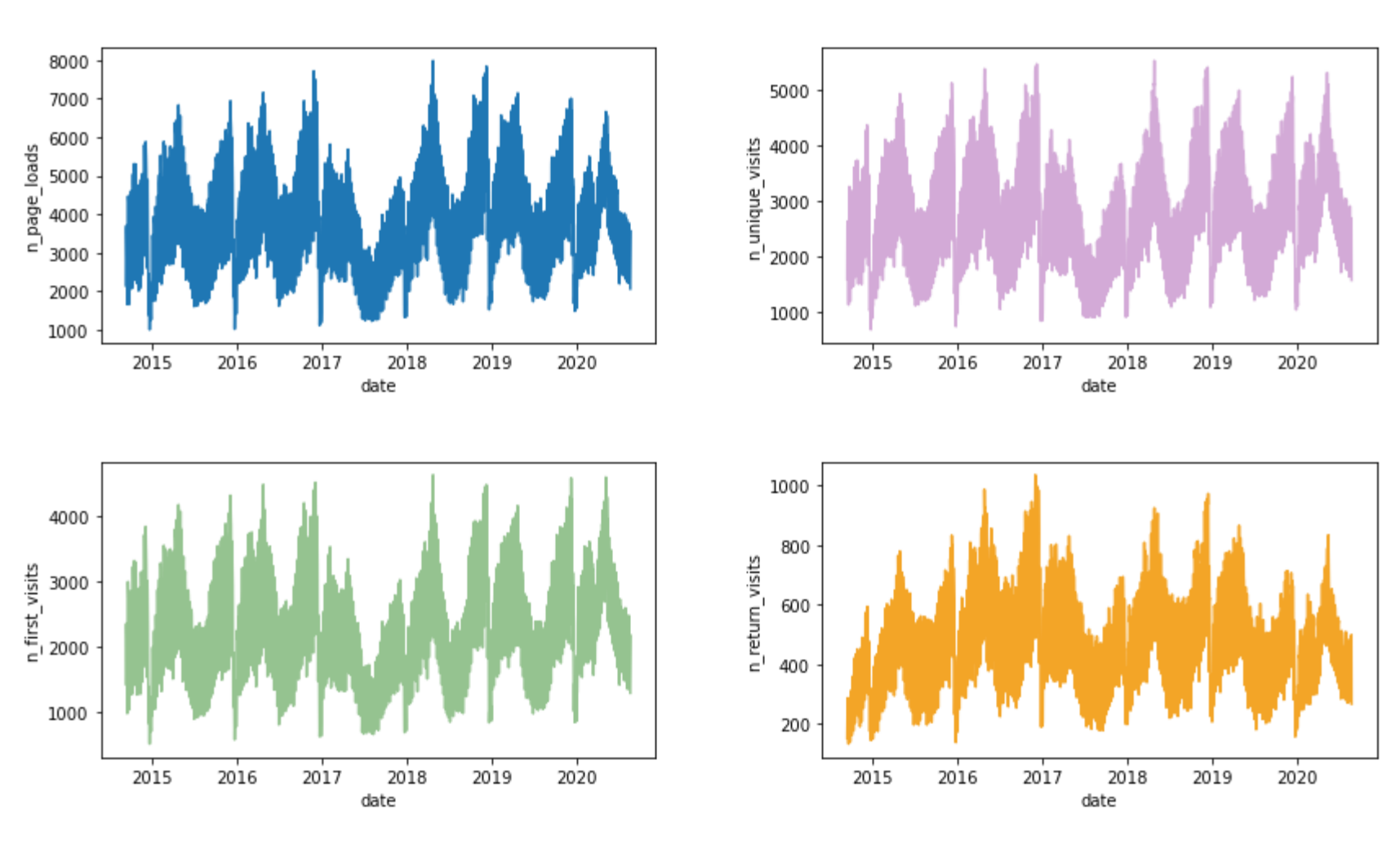
Exploratory Data Analysis

In [10]:

```
# plotting trend graph for all page metrics

figure, axes = plt.subplots(2, 2,
                            figsize = (14, 8),
                            gridspec_kw = {
                                'hspace': 0.4,
                                'wspace': 0.3
                            })
sns.lineplot(x='date', y='n_page_loads', data=df, ax = axes[0, 0])
sns.lineplot(x='date', y='n_unique_visits', color='#D3AAD7', data=df, ax = axes[0, 1])
sns.lineplot(x='date', y='n_first_visits', color='#95C390', data=df, ax = axes[1, 0])
sns.lineplot(x='date', y='n_return_visits', color='#F3A527', data=df, ax = axes[1, 1])
_ = figure.suptitle('Trend of website metrics', fontsize=15)
```

Trend of website metrics



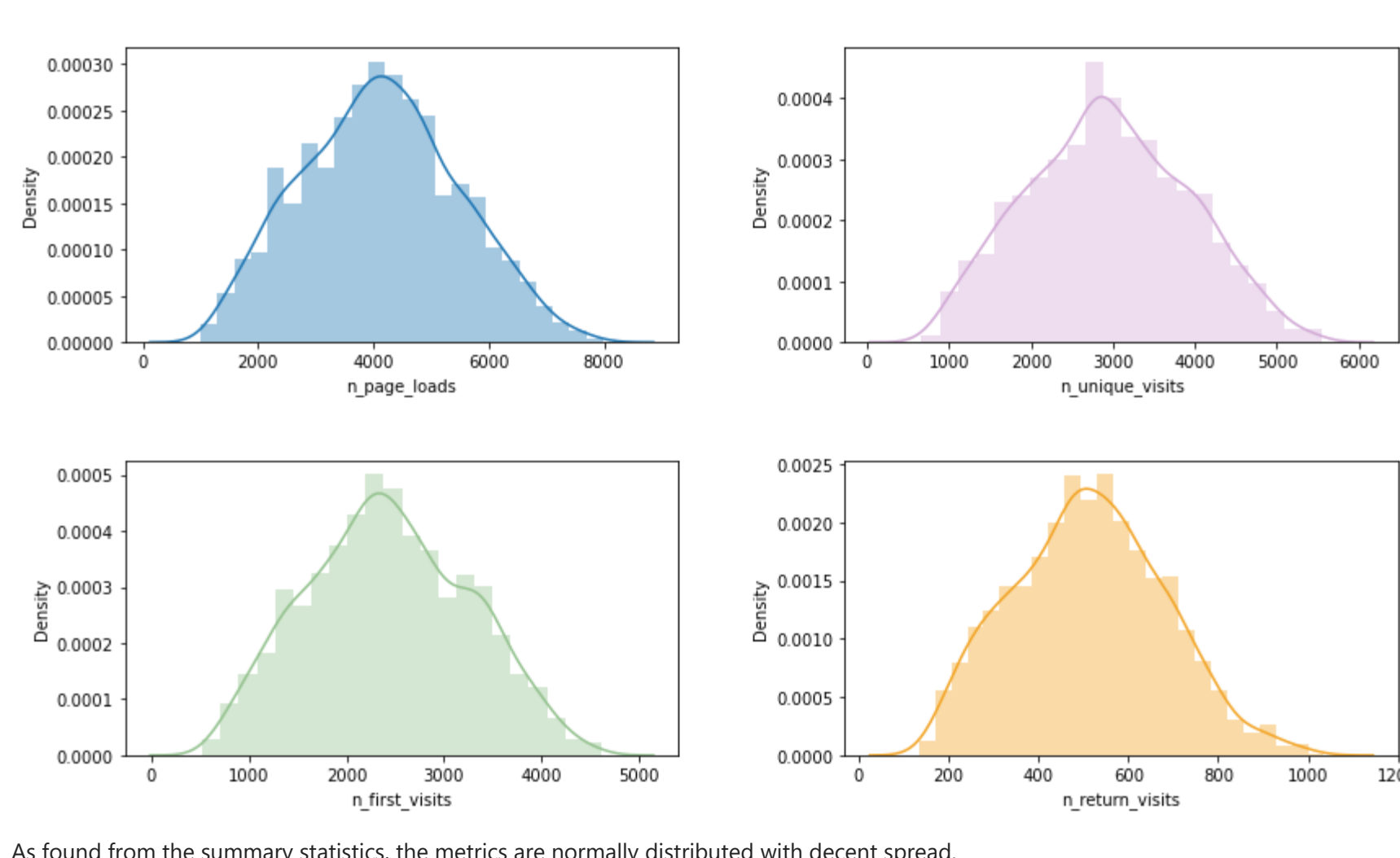
- All the metrics exhibit similar behaviour and are highly correlated
- There is seasonality in all the metrics

In [11]:

```
# getting distribution of page metrics

figure, axes = plt.subplots(2, 2,
                            figsize = (14, 8),
                            gridspec_kw = {
                                'hspace': 0.4,
                                'wspace': 0.3
                            })
sns.distplot(df['n_page_loads'], ax = axes[0, 0])
sns.distplot(df['n_unique_visits'], color='#D3AAD7', ax = axes[0, 1])
sns.distplot(df['n_first_visits'], color='#95C390', ax = axes[1, 0])
sns.distplot(df['n_return_visits'], color='#F3A527', ax = axes[1, 1])
_ = figure.suptitle('Distribution of website metrics', fontsize=15)
```

Distribution of website metrics



As found from the summary statistics, the metrics are normally distributed with decent spread.

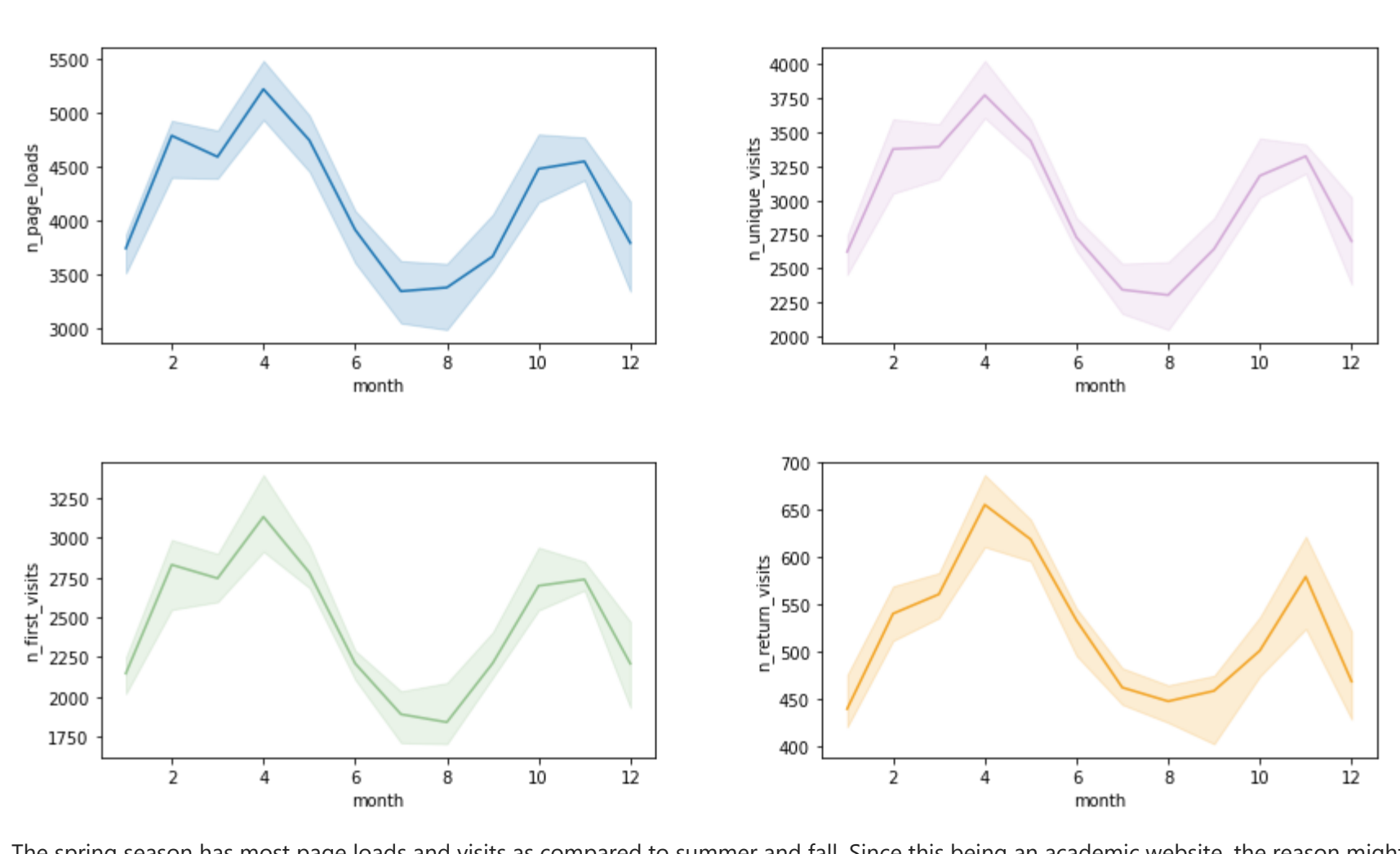
In [12]:

```
# getting month-wise trend of page metrics for seasonality

df['year'] = df['date'].dt.year
df['month'] = df['date'].dt.month

figure, axes = plt.subplots(2, 2,
                            figsize = (14, 8),
                            gridspec_kw = {
                                'hspace': 0.4,
                                'wspace': 0.3
                            })
sns.lineplot(x='month', y='n_page_loads', data=df, estimator = np.median, ax = axes[0, 0])
sns.lineplot(x='month', y='n_unique_visits', color='#D3AAD7', data=df, estimator = np.median, ax = axes[0, 1])
sns.lineplot(x='month', y='n_first_visits', color='#95C390', data=df, estimator = np.median, ax = axes[1, 0])
sns.lineplot(x='month', y='n_return_visits', color='#F3A527', data=df, estimator = np.median, ax = axes[1, 1])
_ = figure.suptitle('Month-wise trend of website metrics', fontsize=15)
```

Month-wise trend of website metrics



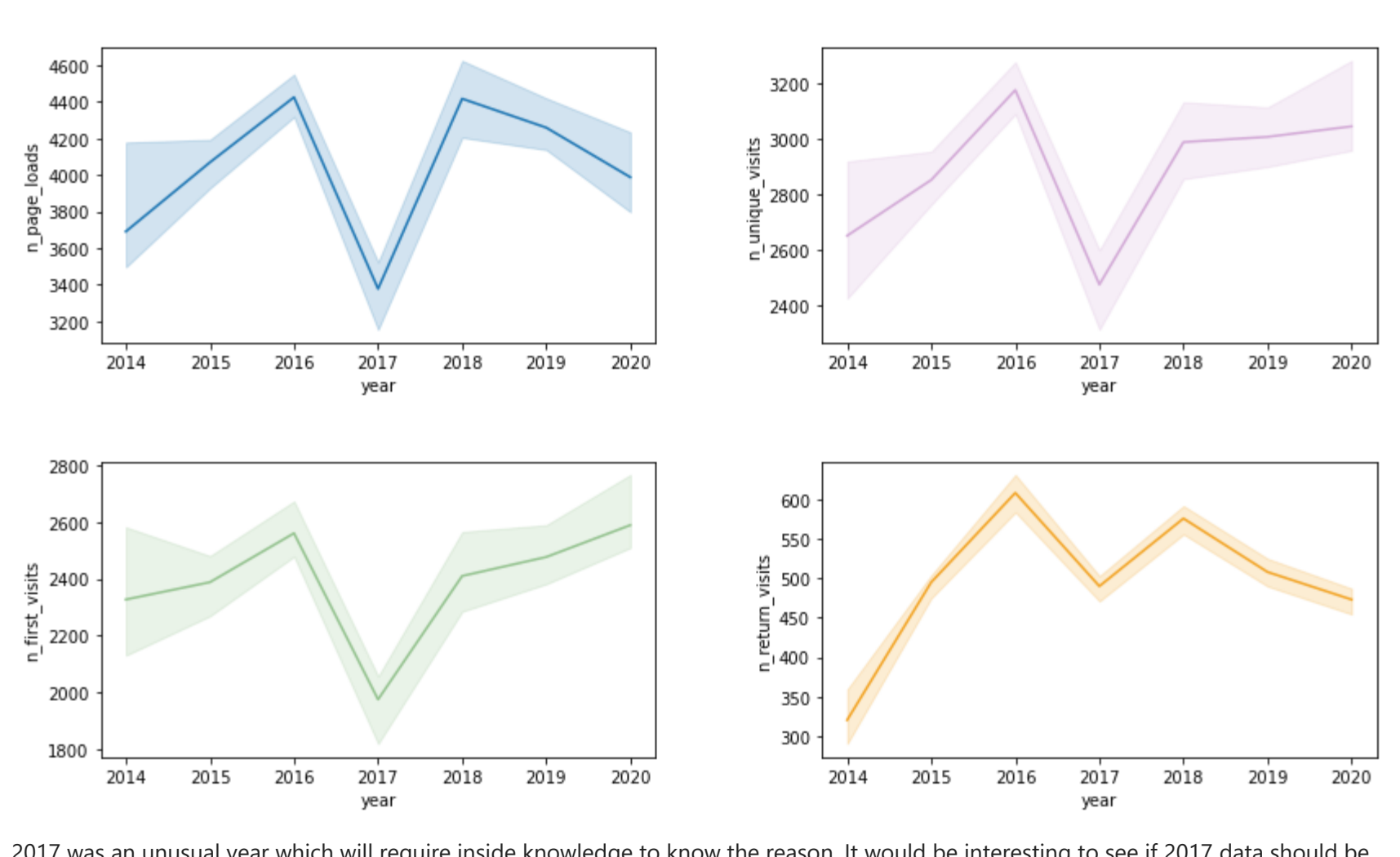
The spring season has most page loads and visits as compared to summer and fall. Since this being an academic website, the reason might be that the course is taught in spring.

In [13]:

```
# getting year-wise trend of page metrics

figure, axes = plt.subplots(2, 2,
                            figsize = (14, 8),
                            gridspec_kw = {
                                'hspace': 0.4,
                                'wspace': 0.3
                            })
sns.lineplot(x='year', y='n_page_loads', data=df, estimator = np.median, ax = axes[0, 0])
sns.lineplot(x='year', y='n_unique_visits', color='#D3AAD7', data=df, estimator = np.median, ax = axes[0, 1])
sns.lineplot(x='year', y='n_first_visits', color='#95C390', data=df, estimator = np.median, ax = axes[1, 0])
sns.lineplot(x='year', y='n_return_visits', color='#F3A527', data=df, estimator = np.median, ax = axes[1, 1])
_ = figure.suptitle('Trend graph of website metrics', fontsize=15)
```

Trend graph of website metrics



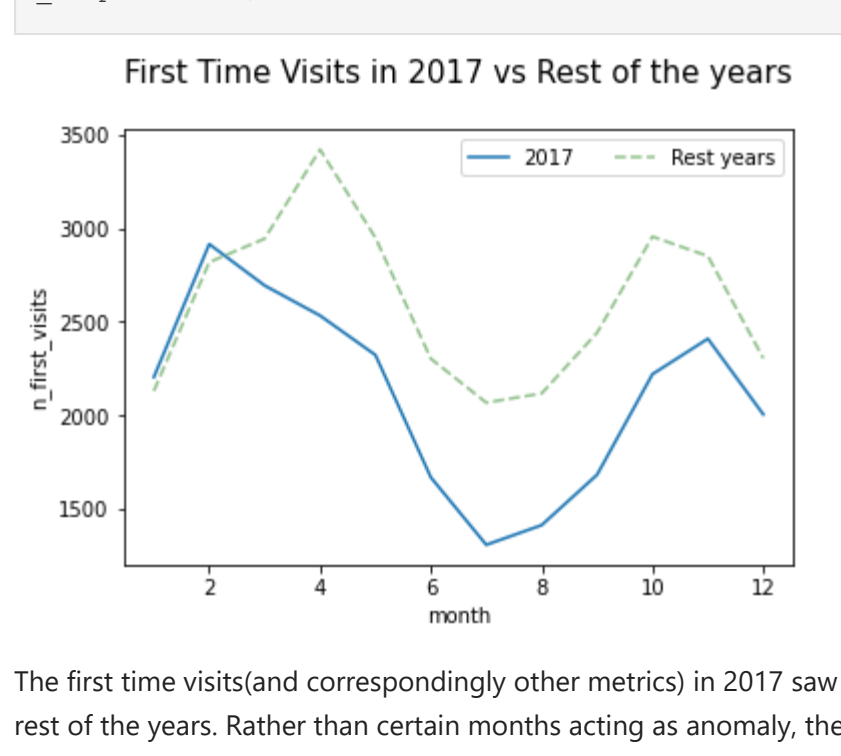
2017 was an unusual year which will require inside knowledge to know the reason. It would be interesting to see if 2017 data should be utilized while modelling. In the later years, while the page loads and return visits are on decline, the number of new users coming onto the website shows an upward trend. This could mean overall the website is showcasing decline, but there are some relevant pages content-wise and quality-wise which is attracting new users to come onto those pages.

In [14]:

```
# investigating 2017 if the metrics drop was consistent across months

year_2017 = df[df['year'] == 2017]
rest_year = df[df['year'] != 2017]
l1 = sns.lineplot(x='month', y='n_first_visits', data=year_2017, estimator = np.median, ci=None)
l2 = sns.lineplot(x='month', y='n_first_visits', data=rest_year, estimator = np.median, ci=None, color='#95C39
labels = ['2017', 'Rest years']
plt.legend([l1, l2], labels=labels, ncol=2)
_ = plt.title('First Time Visits in 2017 vs Rest of the years\n', fontsize=15)
```

First Time Visits in 2017 vs Rest of the years



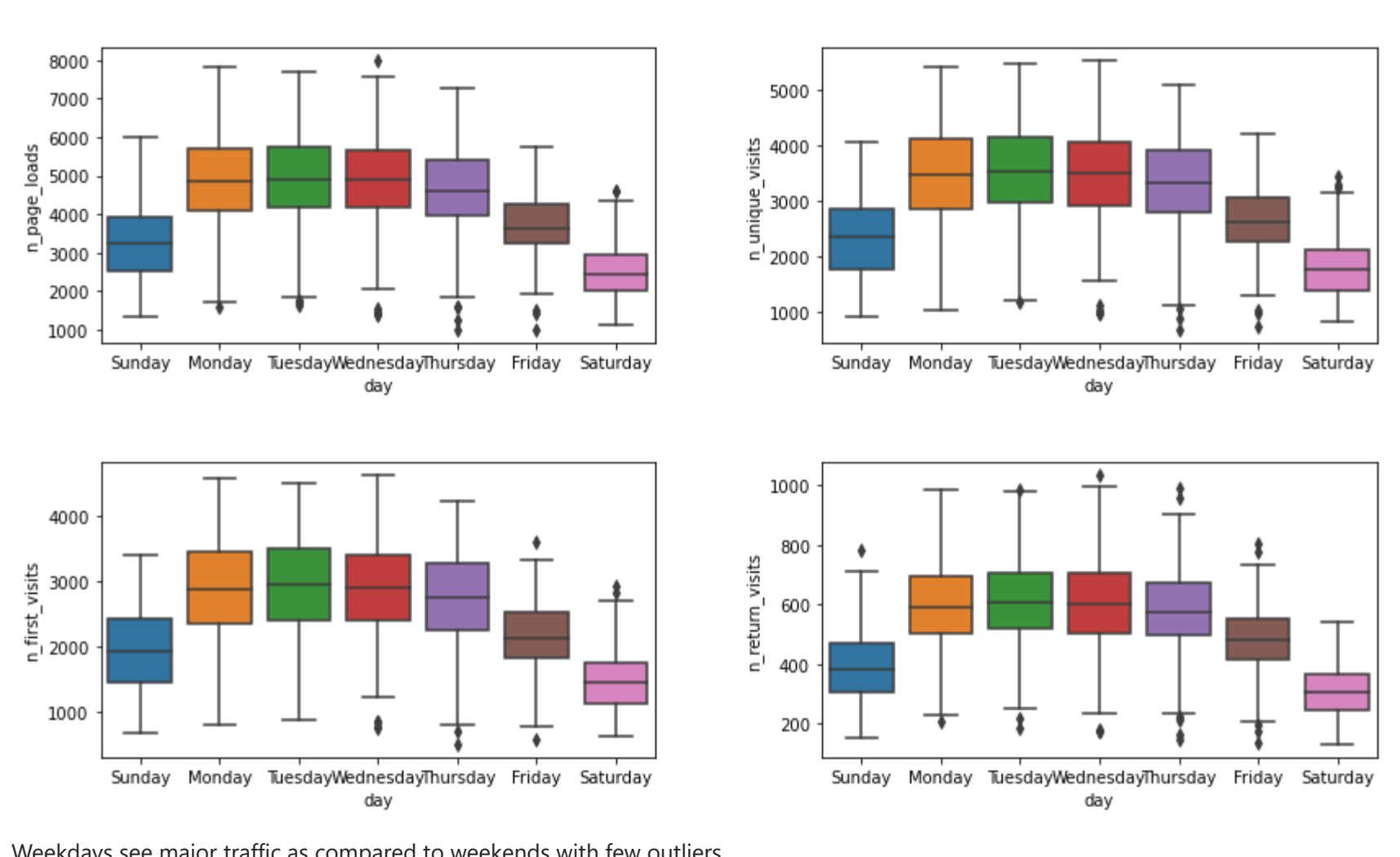
The first time visits(and correspondingly other metrics) in 2017 saw a decline which was consistent from March-December as compared to rest of the years. Rather than certain months acting as anomaly, the issue seems to be something else.

In [15]:

```
# day-wise page metrics

figure, axes = plt.subplots(2, 2,
                            figsize = (14, 8),
                            gridspec_kw = {
                                'hspace': 0.4,
                                'wspace': 0.3
                            })
sns.boxplot(x='day', y='n_page_loads', data=df, ax = axes[0, 0])
sns.boxplot(x='day', y='n_unique_visits', data=df, ax = axes[0, 1])
sns.boxplot(x='day', y='n_first_visits', data=df, ax = axes[1, 0])
sns.boxplot(x='day', y='n_return_visits', data=df, ax = axes[1, 1])
_ = figure.suptitle('Day-wise website metrics', fontsize=15)
```

Day-wise website metrics



Weekdays see major traffic as compared to weekends with few outliers.

Linear Regression

Going forward, since the business problem has been narrowed to only estimating the first time visits for next periods, only this has been kept.

In [65]:

```
# keeping only first time visits at date level
df = df[['date', 'n_first_visits']]
```

In [17]:

```
# Regressing first visits on date

model_df = df.copy()
model_df['date'] = model_df['date'].map(datetime.toordinal)
model = ols('n_first_visits ~ date', data=model_df).fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:         n_first_visits   R-squared:                       0.007
Model:                            OLS   Adj. R-squared:                  0.006
Method:                 Least Squares   F-statistic:                     14.77
Date:                Fri, 25 Feb 2022   Prob (F-statistic):           0.000125
Time:                        19:28:23   Log-Likelihood:                -17629.
No. Observations:                2167   AIC:                         3.526e+04
Df Residuals:                    2165   BIC:                         3.527e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept  -7.787e+04   2.09e+04     -3.727      0.000   -1.19e+05   -3.69e+04
date           0.1090      0.028      3.843      0.000       0.053       0.165
==============================================================================
Omnibus:                       90.461   Durbin-Watson:                   0.464
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               41.384
Skew:                           0.100   Prob(JB):                     1.03e-09
Kurtosis:                       2.353   Cond. No.                     8.67e+08
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 8.67e+08. This might indicate that there are
strong multicollinearity or other numerical problems.
```

- There is a statistical significance with respect to time as seen from p-value which concludes that the observations are time-dependent.
- R-squared is 0.7% which is insignificant. The variance that is shown in new users visits cannot be explained via linear regression.

From the above conclusions, it can be inferred that this dataset is appropriate for a time series analysis.
The time series algorithms that will be used in the following sections for forecasting are ARIMA and Facebook Prophet

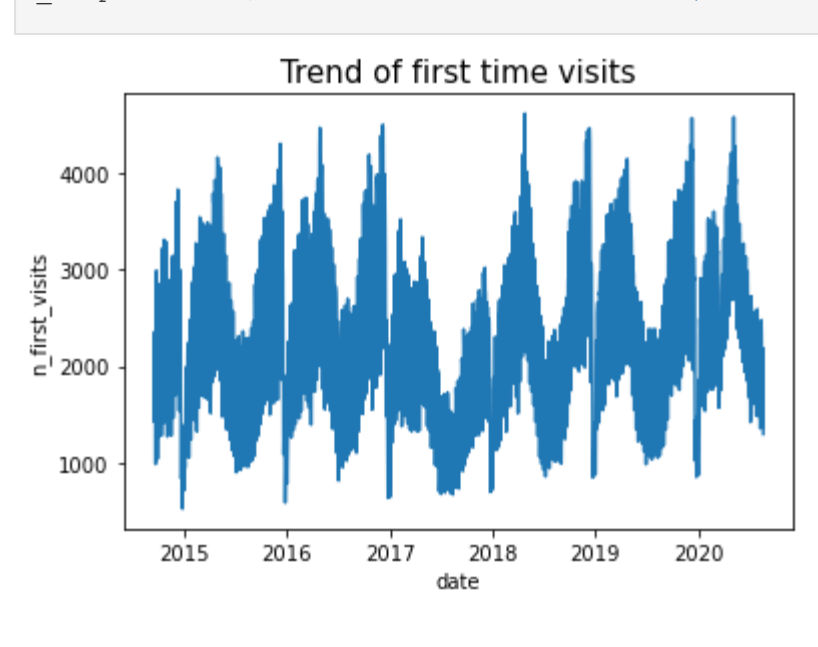
ARIMA

In [18]:

```
# ploting first time visits

sns.lineplot(x='date', y='n_first_visits', data=df)
_ = plt.title('Trend of first time visits', fontsize=15)
```

Trend of first time visits

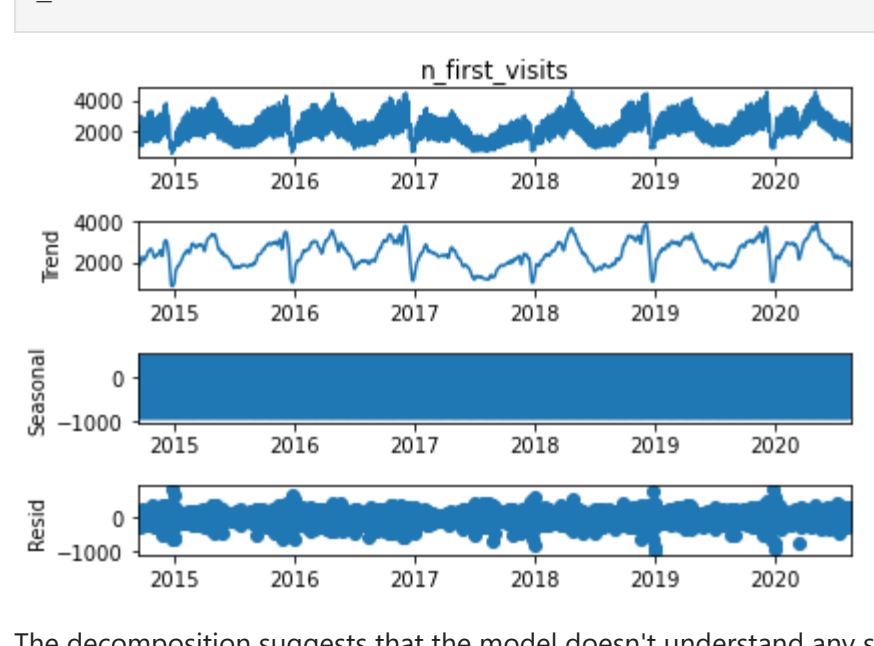


- Visually, it looks like a stationary time-series which is mean reverting at around 2200 first time visits and has constant variance ranging from 800-4200
- There seems to be a seasonality of a year. Since this being a record of academic webiste, the seasonality in all likelihood depends upon in which semester the courses are taught.

In [19]:

```
# decomposing time-series into trend and seasonality

df_copy = df.copy()
df_copy.index = df['date']
result = seasonal_decompose(df_copy['n_first_visits'], model='additive')
_ = result.plot()
```



The decomposition suggests that the model doesn't understand any seasonality. To account for this, weekly and yearly seasonality are manually introduced along with making the data stationary with respect to variance.

In [20]:

```
# making data stationary wrt variance

df_copy = df.copy()
df_copy['n_first_visits'] = np.log(df_copy['n_first_visits'] + 1)
```

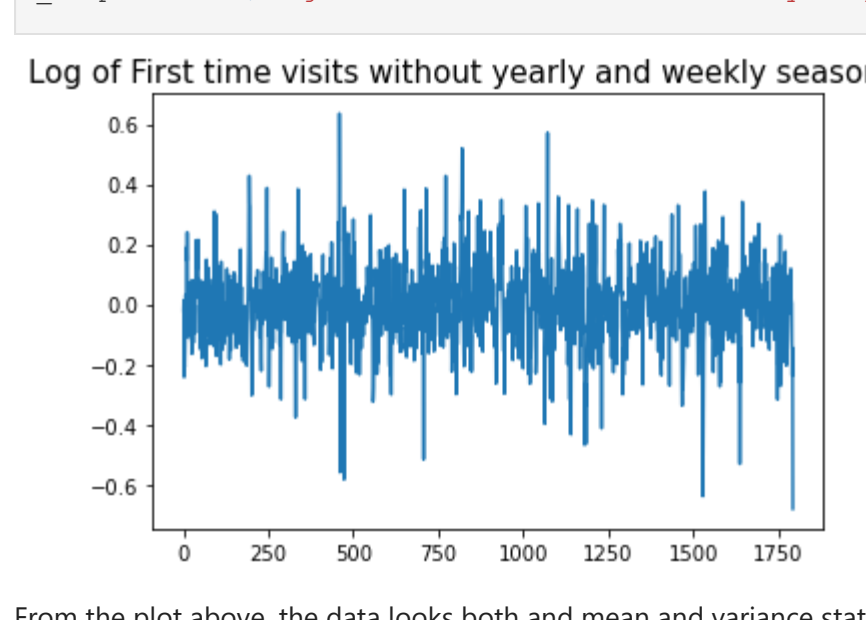
In [21]:

```
# getting differenced data with weekly and yearly seasonality

def difference(dataset, interval):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return pd.DataFrame({'differenced_value': diff})

differenced = difference(df_copy['n_first_visits'], 7)
differenced = difference(differenced['differenced_value'], 365)
differenced = differenced.to_numpy()
plt.plot(differenced)
_ = plt.title('Log of First time visits without yearly and weekly seasonality', fontsize=15)
```

Log of First time visits without yearly and weekly seasonality



From the plot above, the data looks both and mean and variance stationary. To formally test this, ADF and KPSS tests are used.

In [22]:

```
# checking if the seasonality-differenced data is stationary

adf_result = adfuller(differenced)
print(f"ADF: {adf_result[1]:.4f}, which states that our hypothesis of being a stationary time series is {True i
statistic, p_value, n_lags, critical_values = kpss(differenced)
print(f'KPSS Test: The series is {"not " if p_value < 0.05 else ""}stationary')
```

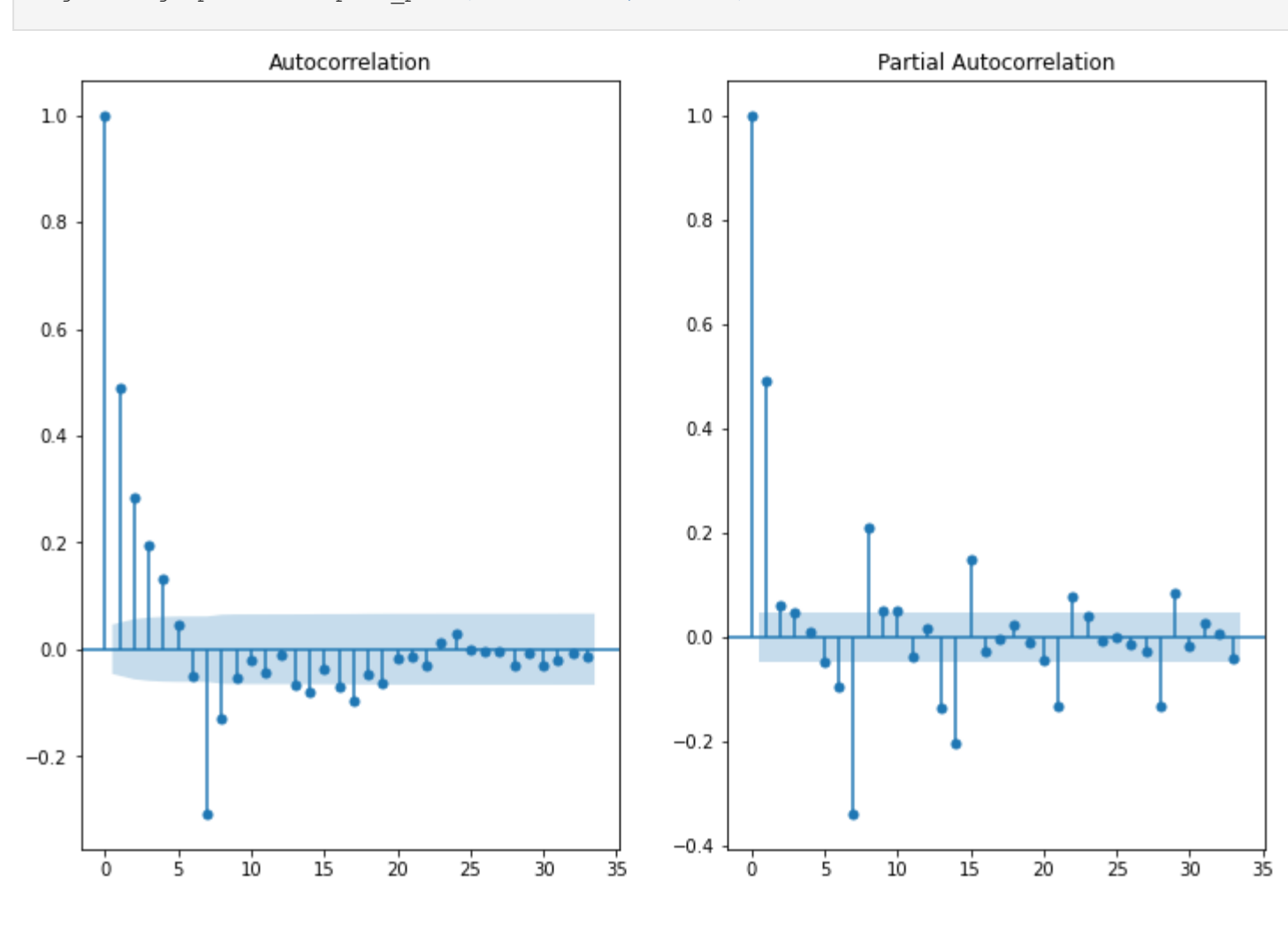
```
ADF: 0.0000, which states that our hypothesis of being a stationary time series is True
KPSS Test: The series is stationary
```

ACF and PACF plots are plotted to check if there is any seasonality left and estimating the AR and MA process.

In [23]:

```
# plotting ACF and PACF plots

fig = plt.figure(figsize=(12, 8))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)
fig = sm.graphics.tsa.plot_acf(differenced, ax=ax1)
fig = sm.graphics.tsa.plot_pacf(differenced, ax=ax2)
```



- ACF shows a dampening effect while PACF is oscillating around 0 as positive and negative
- The above could mean the series is auto-regressive
- The confidence interval continues to cut PACF till the 3rd lag
- The model could be ARIMA(3, 0, 0)
- Other values will also be tested along with auto-arima

In [66]:

```
# testing different ARIMA models based on AIC and BIC

AIC, BIC = [], []
ar, d, ma = [3, 8, 7, 3], 0, [0, 0, 9, 4]
for p, q in zip(ar, ma):
    model = ARIMA(differenced, order=(p, d, q)).fit()
    AIC.append(int(model.aic))
    BIC.append(int(model.bic))

print("AIC's:", AIC)
print("BIC's", BIC)
print(f"The ARIMA model according to AIC is ARIMA({ar[AIC.index(min(AIC))]}, {d}, {ma[AIC.index(min(AIC))]})")
print(f"The ARIMA model according to BIC is ARIMA({ar[BIC.index(min(BIC))]}, {d}, {ma[BIC.index(min(BIC))]})")
```

```
AIC's: [-2695, -3019, -3343, -2961]
BIC's [-2668, -2964, -3244, -2912]
The ARIMA model according to AIC is ARIMA(7, 0, 9)
The ARIMA model according to BIC is ARIMA(7, 0, 9)
```

In [25]:

```
# using auto-arima for finding optimum values of AR and MA on the differenced data

model = pm.auto_arima(differenced, start_p=0, start_q=0,
                      max_p=9, max_q=9,
                      m=1,
                      d=0,
                      max_order=18,
                      stationary=True,
                      seasonal=False,
                      start_P=0,
                      D=None,
                      trace=False,
                      error_action='ignore',
                      suppress_warnings=True,
                      stepwise=False)
```

In [26]:

```
model.summary()
```

Out[26]:

SARIMAX Results

Dep. Variable:	y	No. Observations:	1795
Model:	SARIMAX(7, 0, 9)	Log Likelihood	1680.851
Date:	Fri, 25 Feb 2022	AIC	-3325.702
Time:	19:43:39	BIC	-3226.832
Sample:	0	HQIC	-3289.200
	- 1795		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	-0.0002	0.001	-0.247	0.805	-0.002	0.002
ar.L1	0.2919	0.178	1.638	0.101	-0.057	0.641
ar.L2	0.1100	0.218	0.504	0.614	-0.318	0.538
ar.L3	0.1156	0.089	1.298	0.194	-0.059	0.290
ar.L4	0.0307	0.033	0.916	0.360	-0.035	0.096
ar.L5	0.0081	0.029	0.282	0.778	-0.048	0.064
ar.L6	0.0027	0.029	0.092	0.927	-0.054	0.059
ar.L7	0.1621	0.027	6.096	0.000	0.110	0.214
ma.L1	0.2203	0.180	1.227	0.220	-0.132	0.572
ma.L2	0.0393	0.194	0.203	0.840	-0.341	0.420
ma.L3	0.0311	0.035	0.883	0.377	-0.038	0.100
ma.L4	0.0517	0.035	1.493	0.135	-0.016	0.120
ma.L5	0.0154	0.040	0.386	0.699	-0.063	0.093
ma.L6	0.0766	0.038	2.023	0.043	0.002	0.151
ma.L7	-0.8203	0.037	-22.454	0.000	-0.892	-0.749
ma.L8	-0.1848	0.141	-1.312	0.190	-0.461	0.091
ma.L9	-0.0481	0.142	-0.338	0.735	-0.327	0.231
sigma2	0.0089	0.000	45.011	0.000	0.009	0.009

Ljung-Box (L1) (Q):	0.20	Jarque-Bera (JB):	1716.43
Prob(Q):	0.65	Prob(JB):	0.00
Heteroskedasticity (H):	0.89	Skew:	-0.32
Prob(H) (two-sided):	0.14	Kurtosis:	7.75

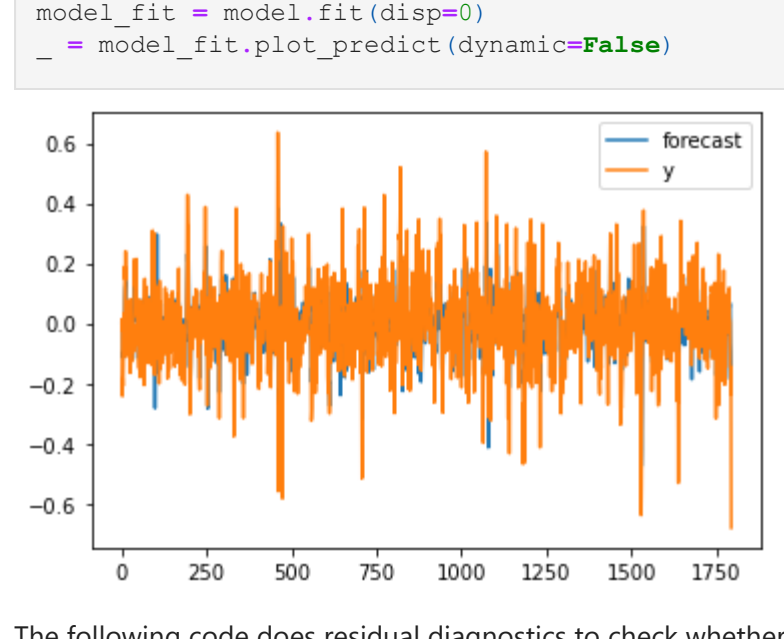
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

From manual checking and auto-arima, ARIMA(7, 0, 9) seems to be the best model.

In [67]:

```
# making our final model

model = ARIMA(differenced, order=(7, 0, 9))
model_fit = model.fit(disp=0)
_ = model_fit.plot_predict(dynamic=False)
```



The following code does residual diagnostics to check whether the model fitted does not create any correlation between the residuals or that the model fitted seems reasonable.
RMSE on trained data is also calculated.

In [68]:

```
# Residual diagnostics

residuals = pd.DataFrame(model_fit.resid, columns=['residual'])
predicted = differenced[0].item() - residuals['residual']
for i in range(len(residuals['residual'])):
    predicted[i] = np.exp(predicted[i] + df_copy['n_first_visits'][365 + i] + df_copy['n_first_visits'][7 + i]
train = df['n_first_visits'][372:].to_numpy()
print(f'RMSE on trained period is: {np.sqrt(np.mean((predicted - train) ** 2)):.2f}')
residuals = pd.DataFrame(model_fit.resid, columns=['residual'])
fig, ax = plt.subplots(2, 2, figsize=(12, 8))
sns.lineplot(x=residuals.index, y="residual", data=residuals, ax=ax[0, 0])
ax[0, 0].set_title("Residual plot")
sns.distplot(residuals["residual"], ax=ax[0, 1])
ax[0, 1].set_title("Residual Density")
sm.graphics.tsa.plot_acf(residuals['residual'], ax=ax[1, 0])
_ = sm.graphics.tsa.plot_pacf(residuals['residual'], ax=ax[1, 1])
```