Time Series Forecasting with Django

Contents

Overview

The purpose of this project is built for a client who is present at a remote location and would like to know how their future prediction might look like. For instance, a manager of a supermarket would like to know how following 12 months of sales might look like. Additionally, not only the client can look into the predictions but also the present data on the graph but also the data itself as a whole and the sample of it too, if he wishes to check it.

Overview of Django

For documentation for Django please click on the following link:

https://docs.djangoproject.com/en/2.0/

Django is a high-level Python Web framework, it is designed to reduce the effort required to create complex data-driven websites and web applications, with a special focus on less code, no-redundancy and being more explicit than implicit.

In order to create a project, you'll have to run the following command from command prompt in a directory where you would like to save you code:

$ Django-admin startproject website

Note: Avoid using the names like Django or test which may conflict with the Django itself or built-in Python package.

The above line of code creates a website folder consisting of manage.py and website subfolder. The website [subfolder of website] consists of the following:

- __init__.py :
- Settings.py: A Django settings file contains all the configuration of your Django installation. This document explains how settings work and which settings are available
- Urls.py : A clean, elegant URL scheme is an important detail in a high-quality Web application. There's no .php or .cgi required, and certainly none of that 0,2097,1-1-1928,00 nonsense. Django lets you design URLs however you want, with no framework limitations.
- wsgi.py: An entry-point for WSGI-compatible web servers to serve your project.

Note: Change your outer website directory

For you to verify that your Django project works run the following command:

$ python manage.py runserver

Expected output:

```
System check identified no issues (0 silenced).

You have unapplied migrations; your app may not work properly until they are applied. Run 'python manage.py migrate' to apply them.

March 21, 2018 - 18:20:32

Django version 1.11, using settings 'website.settings'

Starting development server at http://127.0.0.1:8000/

Quit the server with CTRL-BREAK.
```

By default the runserver command starts the development server on internal IP at port 8000. It can be altered with the following line of command:

$ python manage.py runserver 8080

Any changes made to the Python code need not be restarted the server code to take its effect, the development server automatically reloads any changes made to the server. Before creating polls there is one thing we know is a project is collection of configurations and apps for a website. A project can contain multiple apps and an app can be in multiple projects, similarly polls. You can create polls by following command:

$ python manage.py startapp polls

That will create a polls directory and following is our polls directory will look like *after* running the above command:

```
polls/
    __init__.py
    admin.py
    apps.py
    migrations/
        __init__.py
        models.py
        tests.py
        views.py
```

The next step is to point the root URLconf at the **polls.urls** module.
In **mysite/urls.py**, add an import for **django.urls.include** and insert an **include()** in the **urlpatterns** list, so you have:

```
from django.urls import include, path
from django.contrib import admin


urlpatterns = [
    path('polls/', include('polls.urls')),
    path('admin/', admin.site.urls),
]
```

Django encourages clean and elegant URL design, it doesn't put any cruft in URL like .php or .asp. Let us say, our polls/urls.py looks like the following:

```
from django.urls import path
from . import views
urlpatterns = [
    path('', views.index, name='index'),
]
```

And our **polls/views.py** looks like following:

```
from django.http import HttpResponse
def index(request):
    return HttpResponse("Hello, world. You're at the polls index.")
```

Start the browser and when you visit **127.0.0.1:8000/polls** you could see the HttpResponse of views.

Similarly, all the content appearing functions in our project is written in **views.py** [Check **Methods** topic to look what functions were used and their functionalities were briefed].

Note: In the project if you visit **127.0.0.1:8000/polls** it will give a "Page not found" since we haven't given an empty url expression. In addition you'll be able to view all the url expressions mentioned in projects' polls/urls.py and website/urls.py.

How does one make it so that Django knows which app view to create for a url when using the {% url %} template tag? The answer is to add namespaces to your URLconf. In the polls/urls.py file, go ahead and add an app_name to set the application namespace, for instance:

```
from django.conf.urls import url
urlpatterns = [
        url(r'^home/$', views.home , name = 'home'),
        url(r'^home/view/$', views.view_data, name = 'view_data'),
]
```

The elements present in our home html page invoking appropriate functions would look like the following:

```
<li><a href="{% url 'view_data' %}" >View the data </a></li>
    <li><a href="{% url 'sample_data' %}"> Sample view of the data</a></li>
```

FusionCharts

In the application project we have used FusionCharts Django wrapper which is a Python framework sole responsible for creating charts. For documentation for Django please click on the following link:

https://www.fusioncharts.com/ (Charts used – Fusion Charts)

FusionCharts Suite XT is software provider of data visualizations products (JavaScript Charts, Maps, and etc). It is used by top 80% of Fortune 500 companies including technology giants such as Apple, Google, Cisco, Facebook, Intel, LinkedIn, Microsoft, Hewlett-Packard, IBM, EMC, Nokia, Tibco, as well as The Weather Channel, NASA, and the Federal Government of the United .

FusionCharts folder consists of themes and charts. To download the FusionCharts library using files placed in the folder of your project, you need to:

1. Fusioncharts is placed under a folder called static which is placed in polls of templates. [website\polls\templates\polls\static\fusioncharts]. A static file directory [STATICFILES_DIRS] must be linked to the mentioned path, this is in settings.py [website\website]
2. Insert the <script> tag where you want to include the source of the FusionCharts library link from the specific local folder in the project.

Note: Set your STATIC_ROOT path and run the command given below to update the static files:

```
Python manage.py collectstatic
```

Additionally, this fusioncharts folder is placed under the first hierarchy of website [application folder holds everything related to this web-app] and is used to import in the Views.py where all of our functions lie. Data for the chart can be in one of the following formats:

- JSON string
- XML string

- Python dictionaries

Following parameters are used in FusionCharts Python class constructor in the order they are described. The function assumes that you've already included the FusionCharts JavaScript library to your page:

- **chartType:** The chart you intended to plot needs to be mentioned in string. For example: ‘Column3D’ or Pie2D.
- **chartId:** ID for chart using which it will be recognized in the HTML page. Each chartId on the page must have a unique Id.
- **chartwidth:** This parameter is used for intended width of the chart. For instance, ‘400’ or ‘550’
- **chartheight:** Similar to above parameter, it is used to indicate the height of the chart. For example ‘350’.
- **containerId:** The id of the chart container. For example ‘chart-1’
- **dataFormat:** The type of the data used to feed the chart. For instance ‘json’ or ‘xml’
- **dataSource:** Actual data for the chart. For example {"chart":{},"data":[{"label":"Jan","value":"420000"}]}. If the format is jsonurl or xmlurl, a file with .json or .xml extension respectively is passed as value to the attribute.

Packages

There were number of packages, the following are the one of many that have been used along with the purpose:

- **from statsmodels.tsa.stattools import adfuller:** Since checking for stationarity of the series is one of the crucial steps, we are using the Dickey-Fuller test to determine the stationarity of the series.
- **From datetime import datetime:** This is used to convert any datetime column to datetime format in case if it isn't.
- **import pandas as pd:** Used for innumerable purposes
- **import numpy as np:** Just like pandas, numpy is essential package. Although, it's role in this application is mostly restricted between reshaping, usage of **log** or to bring certain data structure to array.
- **From Django.template import loader:** It is used to load user built template
- **from pyramid.arima import auto_arima:** is a no-nonsense statistical Python library with a solitary objective is to bring R's auto.arima functionality to Python. Pyramid operates by wrapping statsmodels.tsa.ARIMA and statsmodels.tsa.statespace.SARIMAX into one estimator class and creating a more user-friendly estimator interface for programmers familiar with **scikit-learn**. The advantage of using this package is that it can handle seasonal terms and it follows fit and predict conventions. The version used in this application of this package is **0.5-dev4**
- **from django.core.files.storage import FileSystemStorage**: The FileSystemStorage class implements basic file storage on a local filesystem. It inherits from Storage and provides implementations for all the public methods thereof.
- **from django.shortcuts import render**: Combines a given template with a given context dictionary and returns an HttpResponse object with that rendered text.
- **from django.template import loader:** The django.template.loader module provides functions such as get_template() for loading templates. They return a django.template.backends.django.Template which wraps the actual django.template.Template.
- **import random**: This module implements pseudo-random number generators for various distributions.

- from fusioncharts import FusionCharts: This package is responsible for implementing the charts on the webpage.

Methods

The purpose and the functionality of the functions in the Views.py (Website/polls/Views.py) in the same order of appearance are described:

- Home: Here the uploaded dataset is saved with the help of inbuilt method called FileSystemStorage.
- Display_data: With the help of uploaded file extension an appropriate method is used to read the file. For instance, if file name = xyz.csv, then its extension would be .csv hence, the function would use read_csv.
- Proper_format: For example, if the dataset consists of a column of time related, it used as an index for the dataset such that every row has a unique date index with the help of parse_dates attribute.
- View_data: Displaying the whole dataset.
- Sample_data: With the help of random.randint we generate couple of random integers which in turn are used to view the random data points.
- Data_summary: Uploaded data's summary values are retrieved with the help of this function.
- Na_values: This function is used to check for any missing values.
- Checking_Stationarity: With the help of Dicky-Fuller test the function concludes that the available data is stationary or *not.*
- Graphical_v: The graphical method passes a single argument to this function which contains the dataset brought into the proper format. The main purpose of this method is view the original dataset.
- Moving_Averages: This function takes the request from the web page to display the outcome of moving averages with the help of graphing1.
- Exponential_Smoothing: This function implements Simple Exponential Smoothing technique where *alpha* is used as smoothing parameter.
- Handle_missing: The method expects array like structure and returns the missing values replaced with the mean outcome.
- Double_ exponential_smoothing: This method is used to smooth the uploaded series.
- Triple_exponential_smoothing: Holt Winters method is used to forecast twelve data points. Here we have encapsulated by three methods named initial_trend, initial_seasonal_components and triple_exponential.

    initial_trend: Finds the initial value of trend at a given point of time.

initial_seasonal_components: Similar to Initial_trend, this function tries to compute the value of seasonal component at given point of time.

triple_exponential: This function is solely responsible for forecasting.

- ARMA: For ARMA we have used auto_arima. In order to function it as ARMA instead of ARIMA the value of differencing is taken as *zero*.
- ARIMA: For ARIMA we have used auto_arima, this function finds the optimal hyper parameters. This function finds the best parameters through step-wise algorithm.

Note: For all the techniques used in the project we have forecasted twelve data points by default.

Elements

Check for Stationarity: The name itself reveals its functionality, this tab displays whether the uploaded dataset is stationary or *not.* To extend it features, the outcome of Dickey-Fuller test are displayed along with the previous output. [*Check* ***Checking_Stationarity*** *under* ***Methods****.*]

The output of this can be observed at the following url:

127.0.0.1:8000/polls/home/stationarity/

View data on the Graph: The sole usage of this element is used to graphically view the dataset as a whole on the chart. This is projected with the help of FusionCharts. [*Check* ***Graphical_v*** *from* ***Methods***]

The output of this element can and will be found at:

127.0.0.1:8000/polls/home/graphical/

Techniques I: This drop down element consists of two techniques, Simple average and Naïve method.

Techniques II: This elements gives a drop-down menu when cursor is placed on it. Each drop-down hyperlink is a function which gives out the forecasted and original data values. The techniques used here are Simple Exponential Smoothing, Double Exponential Smoothing and Holt Winters Method.

Techniques III: This is much similar to two drop down elements mentioned above. Here the forecasting techniques used are ARMA and ARIMA.

Home: Home is the opening page of the web application. Although, this is better known for uploading the dataset and various outcomes from Methods which are displaying the data, sample of the dataset, summary results of data and viewing any missing values.

Choose File: An uploaded file will be found on the media folder in our main directory of project [website]. This is possible by adding a path to media root. For instance:

```
MEDIA_URL = '/media/'

MEDIA_ROOT = os.path.join(BASE_DIR, 'media/')
```

The below line of code is added to options in Templates which is in settings.py:

```
django.template.context_processors.media'
```

View the data: On clicking on this element a request is sent to **view_data** method in views which is responsible for viewing the whole dataset on the homepage.

The output of this element can seen at:

127.0.0.1:8000/polls/home/view/

Sample view of the data: This element is connected to a method called **sample_data** on *views.py* which generates random data points from the uploaded dataset.

The output of this element can will be found at:

127.0.0.1:8000/polls/home/sample/

Summary of the data: The element explains itself, on clicking this element summary of the dataset is shown. This element is connected to a method called **data_summary**. The output consists of the count of the data points, mean, standard deviation, max value, name of the column and the datatype.

The output of this element can be found at the following url:

127.0.0.1:8000/polls/home/summary/

NA values of the data: Upon clicking a request is sent to na_values, it displays na values of the dataset. Although it displays zero as missing values are handled in proper_format method.

The output of this element can seen at the following url:

127.0.0.1:8000/polls/home/na/

Limitations

The following are the limitations of this application.

- The first and foremost limitation of this web-app is ability of handling the missing values. In this model we handle missing values by replacing them with the mean of the remaining values.
- This web-app handles univariate time series.