1. **Starting a django project**:
   In-order to start a django project, run the following command:

   ```
   django-admin startproject dashboard
   ```

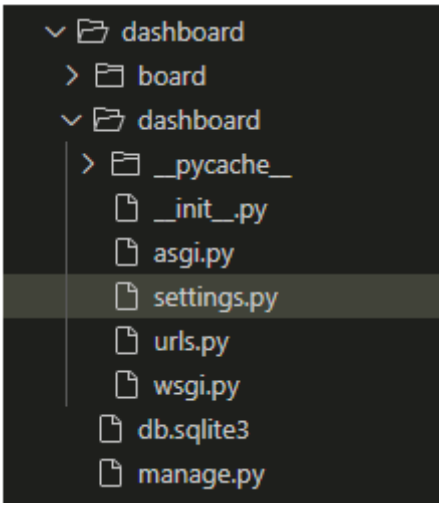
   - Replace '**dashboard**' with the name of your project.
   - It is assumed that Django is already installed.

2. **Starting an app**:
   Once the django project is created, we can create our app. We move inside the project folder (dashboard folder) using the following command:

   ```
   python manage.py startapp board
   ```

   - In-order to run this command, make sure you have python installed.
   - Replace 'board' with the name of your app.

After running the above 2 commands, a hierarchy of files will be generated. Now, we need to connect the app to our django project by modifying the settings.py file of the django project. The settings.py file is found at the following location:

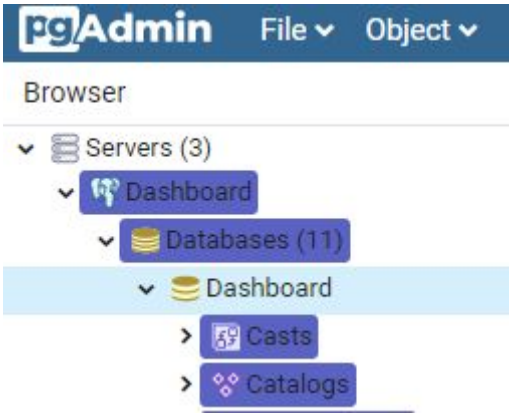


Append the name of the app to the list of INSTALLED_APPS:

```
INSTALLED_APPS = [
    'rest_framework',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'board'
]
```

(In our case, we have appended the name of our app, board at the end of the list)

3. **Connecting to the database**:
   After the execution of the above 2 commands, a hierarchy of files will be created. In this step, we will connect the django app to our PostgreSQL database. For this step, we have assumed that the PostgreSQL has already been installed. Here, for our project, we have created a database named **Dashboard**.

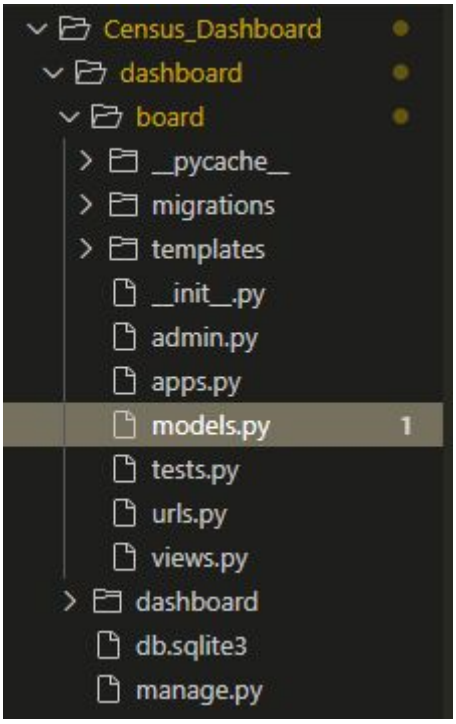


In-order to connect our django app to the Dashboard database, we have to again edit the django project's settings.py file (the same file edited in the previous step). Open the settings.py file and go the section commented as Database. In the designated section, we can see a dictionary named DATABASES. We need to add the information of our database to that dictionary as follows:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'Dashboard',
        'USER': 'postgres',
        'PASSWORD': 'root',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

ENGINE : we have to mention the Django engine for Postgres.
NAME : name of the database.
USER : username for the database.
PASSWORD : password for the database access for the user.
HOST : address of the database. Here since the database is running at the same machine, we have used ' localhost'.
PORT : port number at which the PostgreSQL server is running. By default, it is 5432.

4. **Creating the structure of the table in model.py** :
   So far, we have set-up our django project, connect it to our django app, and connect the database to it. The next part is to create the structure of our database table (schema). For describing our schema, we have to edit the models.py file present in our app directory.



In the models.py file, we add the following lines, which describe our schema in the form of a class:

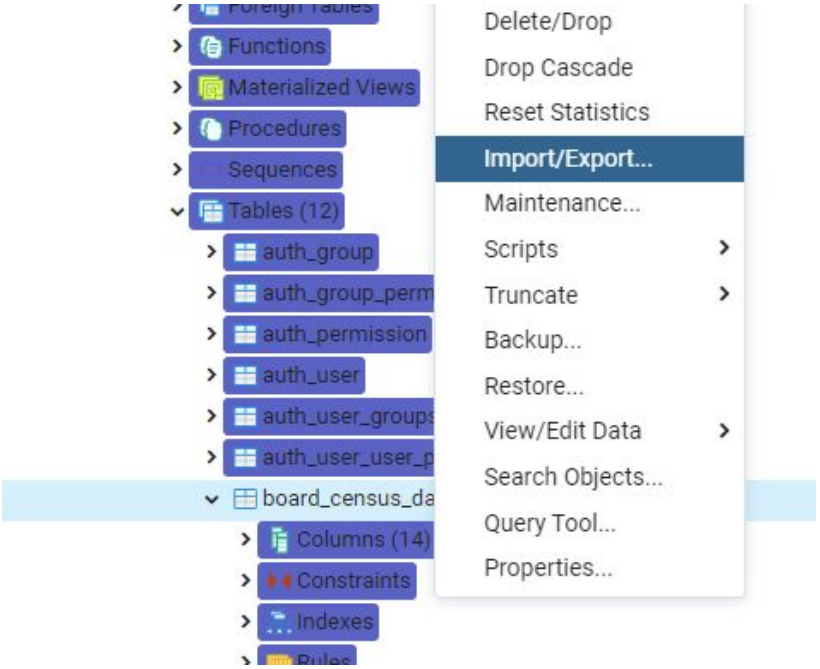
```
class census_data(models.Model):
    country = models.CharField(max_length=25)
    state = models.CharField(max_length=25)
    location = models.CharField(max_length=25)
    pop_total = models.PositiveIntegerField()
    pop_males = models.PositiveIntegerField()
    pop_females = models.PositiveIntegerField()
    lit_total = models.PositiveIntegerField()
    lit_males = models.PositiveIntegerField()
    lit_females = models.PositiveIntegerField()
    sex_ratio = models.DecimalField(max_digits=7, decimal_places=2)
    literacy_rate_total = models.DecimalField(max_digits=7, decimal_places=2)
    literacy_rate_males = models.DecimalField(max_digits=7, decimal_places=2)
    literacy_rate_females = models.DecimalField(max_digits=7, decimal_places=2)
```

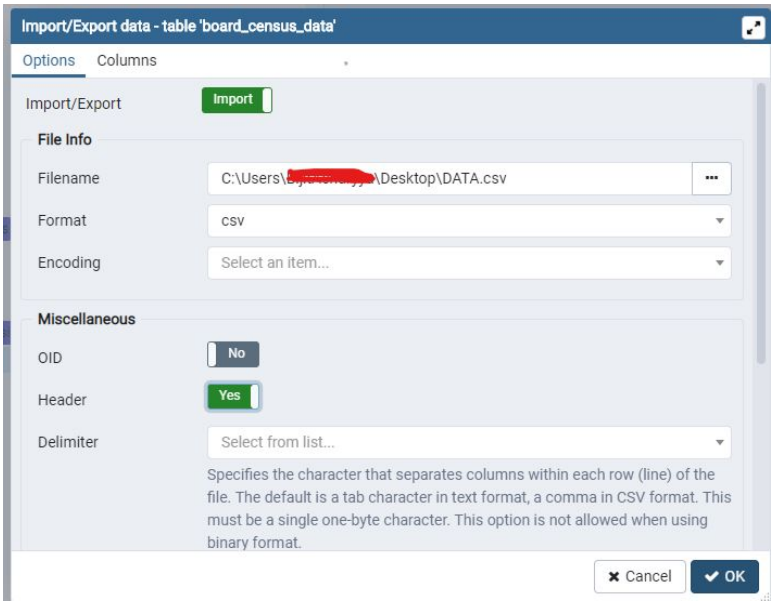
Here we have used the data-types provided by Django to define the schema. Once we have described the schema, in order to create the table in the database, we have to run two django commands:

1. python manage.py makemigrations
2. python manage.py migrate

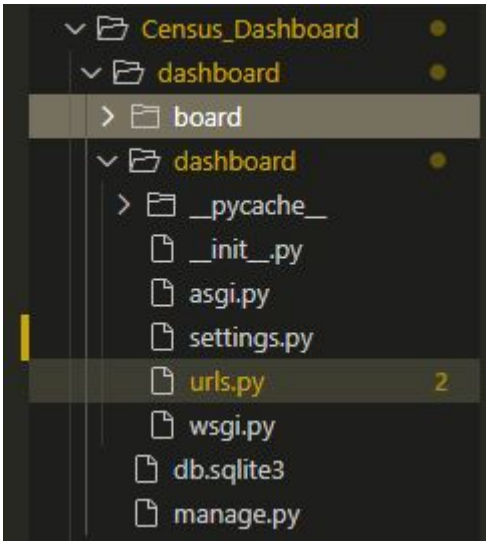
After the execution of the above 2 commands, we can see that the table has been created in our database.

5. **Loading the data into the database** :
   After we have created the database table, now its time to load our data in it. Since our data was in CSV form, we have directly imported the CSV file into our database table.





6. **Setting the URL pattern in the project urls.py and in the app's urls.py** : Next in-order to provide access to our app via the Django server, we have to specify the URL that will provide access to our app. For this purpose, we have to edit the urls.py file in the Django project folder.

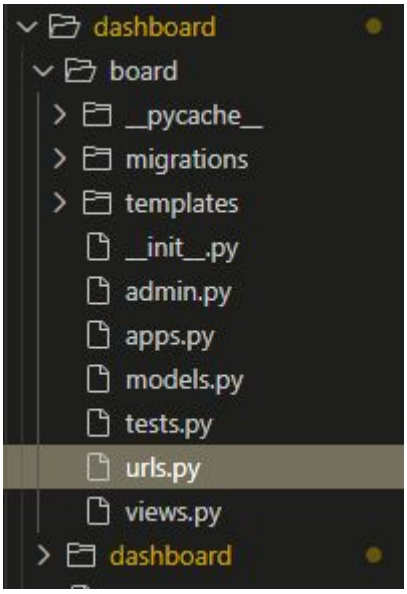


Then in the urls.py, we have to add the following line in the urlpatterns list:

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

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

The second line instructs the Django server that for any url by default, the server must refer to the board.urls file. If 'admin/' part is present in the url, then the server will load the admin site.

Next, we must edit the urls.py file in our django app folder.

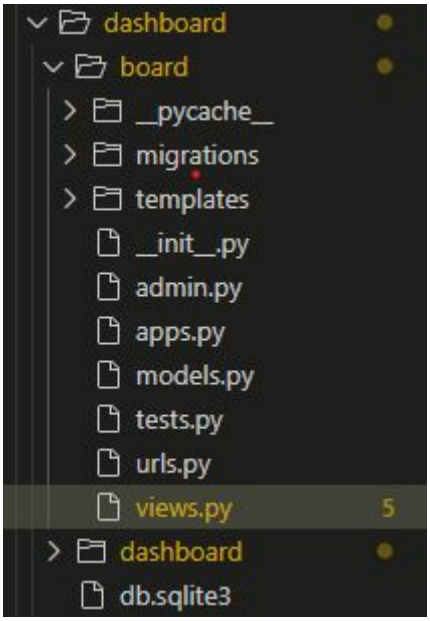


If the urls.py file is not present, create it in the app folder and add the following lines.

```
from django.urls import path
from django.conf.urls import url
from . import views

urlpatterns = [
    path('', views.get_homebase_data),
    url(r'^select_page',views.select_page),
]
```

These 2 lines map the URLs to the functions present in the views.py file. We are gonna define the functions in the next step. By default, the server will call the get_homebase_data() and if 'select_page' is present in the url then the select_page() will be called. Both the functions are defined in the views.py file in the next part.

7. **Define the functions in views.py :**
   We have to now define the functions that will retrieve the data from the database and calculate the necessary values and forward the data to the front-end.



In the views.py, we have defined a total of 3 functions to handle all the needs of our app.
Two of the functions are directly mapped with the help of urls.py (in the previous step). So we are gonna define each of the 3 functions in the following section :

A. **Get_homebase_data ()** :

- Arguments : 2 (request and id), the purpose of the id argument will be demonstrated at the end of the function.

```
def get_homebase_data(request,id=0):
```

-> Initialize the variables:

```
    population=0
    literates=0
    literacy_rate_set=[]
    states=[]
    state_set_data=[]
    literate_total_data = []
```

-> Retrieval of the data from the database :

```
    with connection.cursor() as cursor:
        cursor.execute('select state, sum(lit_total) from board_census_data group
by state')
        queryset = cursor.fetchall()
```

```
        cursor.execute('select state, sum(pop_males), sum(pop_females) from
board_census_data group by state')
        state_set=cursor.fetchall()

        cursor.execute('select sum(pop_total) from board_census_data')
        population=cursor.fetchall()

        cursor.execute('select sum(lit_total) from board_census_data')
        literates=cursor.fetchall()

        cursor.execute('select state, (cast(total_literates_male as
float)/cast(total_population_male as float) * 100) as male_lr,\
                        (cast(total_literates_female as
float)/cast(total_population_female as float) * 100) as female_lr\
                        from(select state, sum(lit_males) as total_literates_male,
sum(lit_females) as total_literates_female,\
                                sum(pop_males) as total_population_male,
sum(pop_females) as total_population_female\
                                from board_census_data group by state) as foo')
        literacy_rate_fetch=cursor.fetchall()
```

In this part of the function, we are just executing queries to retrieve the values from the database table and retrieve the results. In this part, we are retrieving the following values:

- The total population: population
- State name + literary population (the number of literates) : queryset
- State name + State male population + State female population : state_set
- The total literate population: literates
- The literacy rates (gender-wise) as per state : literacy_rate_fetch

Then we calculate the overall literacy rate :

```
    literacy_rate=round((literates[0][0]/population[0][0])*100,2)
```

Next, we will create a dictionary of the literacy rates state-wise:

```
for state in literacy_rate_fetch:
      literacy_rate_set.append({'state':state[0],'male':round(state[1],2),'female'
:round(state[2],2)})
```

Another dictionary for population gender-wise state-wise and the name of states (to populate the drop-down menu):

```
    for state in state_set:
        state_set_data.append({"location":state[0],"population_male":state[1],
                    "population_female":state[2]})
        states.append(state[0])
```

Another dictionary for total population state-wise:

```
    for row in queryset:
        literate_total_data.append({"location":row[0],"population":row[1]})
```

And finally we have the return statement:

```
 if id==0:
        return render(request, 'dashboard_census.html',{ 'literate_total_data':
 literate_total_data,'no_of_states':len(states), 'state_data':state_set_data,
'states':states, 'population':population[0][0], 'literates':literates[0][0],
'literacy_rate':literacy_rate, 'state_lr':literacy_rate_set})
    else:
        return JsonResponse({'literate_total_data': literate_total_data,
'no_of_states': len(states), 'state_data': state_set_data, 'states': states,
'population': population[0][0], 'literates': literates[0][0], 'literacy_rate':
literacy_rate, 'state_lr': literacy_rate_set})
```

Now here we have the usage of the id argument. We can call this function from 2 locations and the value of id help us to identify the location and accordingly we return data. As we can see we have 2 return statements according to the value of the id variable. When the value of id is 0 (zero), the function returns the rendered Html page (dashboard_census.html) along with the dictionary data.

Whereas if the value of id is other than zero, we will simply send data in the form of JSON. This case scenario was specifically designed for ajax calls.

B. **statebase_page()** :

Next, we have the statebase_page function, whose main aim is to retrieve information of a particular state and forward them to the front-end to be displayed. The same steps (explained in the above function) are repeated in this function but for a particular state, that is supplied with the help of the request argument. Since this function can be called by an ajax call, the function returns a JSON object.

```
def statebase_page(request):
    population = 0
    literates = 0

    location=request.GET['value']

    with connection.cursor() as cursor:
        query = "select location, lit_total from board_census_data where state='"
+location+"';"
        cursor.execute(query)
        queryset = cursor.fetchall()

        query = "select location, pop_males, pop_females from board_census_data
where state='" +location+"';"
        cursor.execute(query)
        state_set = cursor.fetchall()

        cursor.execute("select sum(pop_total) from board_census_data where
state='"+location+"' group by state;")
        population = cursor.fetchall()
```

```
        cursor.execute("select sum(lit_total) from board_census_data where
state='"+location+"' group by state;")
        literates = cursor.fetchall()

        cursor.execute("select location, (cast(total_literates_male as
float)/cast(total_population_male as float) * 100) as male_lr,\
                        (cast(total_literates_female as
float)/cast(total_population_female as float) * 100) as female_lr\
                        from(select location, lit_males as total_literates_male,
lit_females as total_literates_female,\
                                pop_males as total_population_male, pop_females as
total_population_female\
                                from board_census_data where state='"+location+"') as
foo")
        literacy_rate_fetch = cursor.fetchall()

    literacy_rate = round((literates[0][0]/population[0][0])*100, 2)
    literacy_rate_set = []
    for state in literacy_rate_fetch:
        literacy_rate_set.append({'state': state[0], 'male': round(
            state[1], 2), 'female': round(state[2], 2)})

    states = []
    state_set_data = []
    for state in state_set:
        state_set_data.append(
            {"location": state[0], "population_male": state[1],
"population_female": state[2]})
        states.append(state[0])

    literate_total_data = []

    for row in queryset:
        literate_total_data.append({"location": row[0], "population": row[1]})

    return JsonResponse({'literate_total_data': literate_total_data,
'no_of_states': len(states), 'state_data': state_set_data, 'states': states,
'population': population[0][0], 'literates': literates[0][0], 'literacy_rate':
literacy_rate, 'state_lr': literacy_rate_set})
```

The function differs from the previous one in one aspect:

```
location=request.GET['value']
```

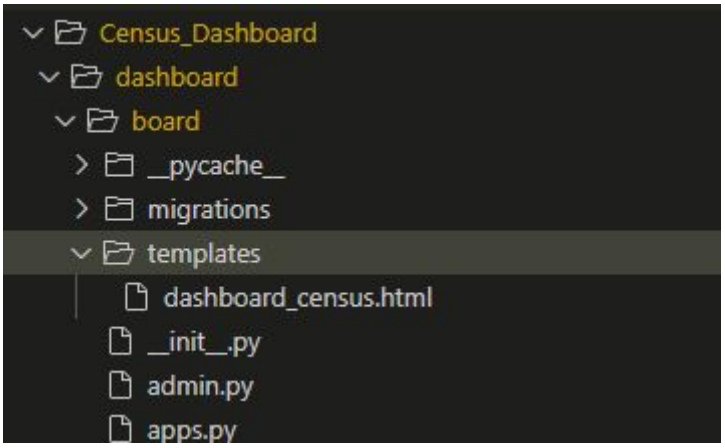
In the above line, we are retrieving the location (state) name from the request argument that was sent from the front-end via a GET method.

C. **select_page()** :

This is the third and final function that we have defined in our views.py file. This function receives the ajax call request and then calls the appropriate functions according to the content of the request sent by the front-end.

```
def select_page(request):
    if request.method == 'GET':
        print(request.GET)
        value = request.GET['value']
        if value=="overview":
            return get_homebase_data(request,1)
        else:
            return statebase_page(request)
```

8. **Structuring the Html file :**
   First create a folder named templates and inside it a html file. We have named our html file : dashboard_census.html



   Getting started with the html file, in the head part we will do the required imports and the title.

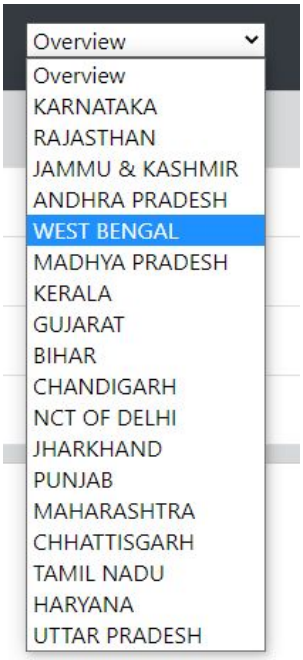
```
    <head>
        <meta charset="UTF-8">

        <!--JQuery-->
        <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></scr
ipt>
        <!-- Latest compiled and minified JavaScript -->
        <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"

integrity="sha384-Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIc
PD7Txa"
            crossorigin="anonymous"></script>

        <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.c
ss"
```

```
integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT
2MZw1T" crossorigin="anonymous">
        <!--amcharts-->
        <script src="//cdn.amcharts.com/lib/4/core.js"></script>
        <script src="//cdn.amcharts.com/lib/4/charts.js"></script>
        <script
src="https://cdn.amcharts.com/lib/4/themes/animated.js"></script>

        <title>RSAC_dashboard</title>
    </head>
```

In the above part, we have imports for the 4 components:

- Jquery
- Amcharts
- Javascript
- Bootstrap

Moving on to the body part of the html file, we have divided it into 3 parts. The first part of the html file is the navigation bar part that remains constant. The nav-bar consists of 2 parts: a header and a drop-down list.




```
 <nav class="navbar navbar-expand-lg navbar-dark navbar-dark
bg-dark">
            <h1 style="color: #ffffff" >RSAC Census
Dashboard</h1>

        <div class="navbar color3 mx-auto">
            <select class="mx-auto" id="drop-down-states"
onchange="stateChange(this);">
                <option value="overview"
selected>Overview</option>
                {% for result in states%}
                    <option
value='{{result}}'>{{result}}</option>
                {% endfor %}
            </select>
            <br>
        </div>
        </nav>
```

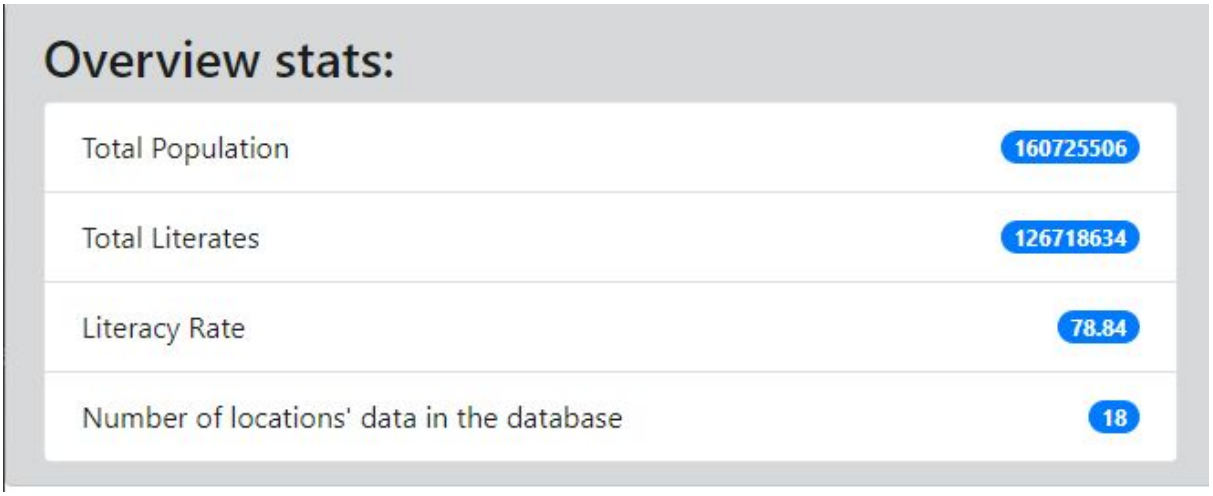
The drop-down list is dynamically populated from the data provided by the function (in views.py). Here we have also set to call a function

stateChange(this), that gets called when the value of the drop-down list gets changed. The function is explained in the next step.

The 2nd and 3rd part of the html file is kind-of dynamic in the sense that its contents changes according to the value selected by the user in the drop-down list.

The 2nd part of the html file is the overview stats part, that displays the overall statistics of the location. In this part, we have only displayed a certain number of informations :

- Total Population
- Total Literates
- Literacy Rate
- Number of locations' data in the database




```
        <div class="alert alert-dark">
        <h3>Overview stats:</h3>
            <div>
                <ul class="list-group">
                    <li class="list-group-item d-flex justify-content-between
align-items-center">
                        Total Population
                        <span class="badge badge-primary badge-pill"
id="total_population">{{population}}</span>
                    </li>
                    <li class="list-group-item d-flex justify-content-between
align-items-center">
                        Total Literates
                        <span class="badge badge-primary badge-pill"
id="total_literates">{{literates}}</span>
                    </li>
                    <li class="list-group-item d-flex justify-content-between
align-items-center">
                        Literacy Rate
```

```
                        <span class="badge badge-primary badge-pill"
id="total_literacy_rate">{{literacy_rate}}</span>
                    </li>
                    <li class="list-group-item d-flex justify-content-between
align-items-center">
                        Number of locations' data in the database
                        <span class="badge badge-primary badge-pill"
id="total_location">{{no_of_states}}</span>
                    </li>
                </ul>
            </div>
        </div>
```

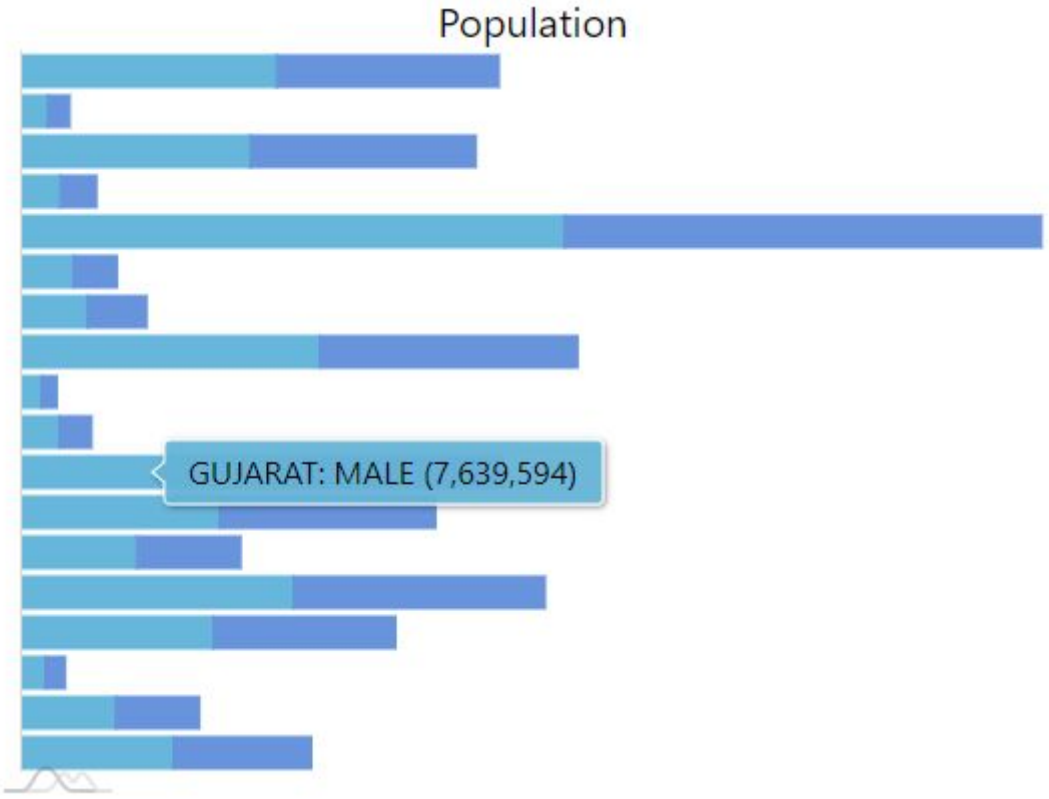
How these values are updated are referred to in the later part of this step. The 3rd part of the html file, is the part that presents the information on population, literates population and literacy rates gender-wise of the constituent locations of the selected state/country.

For the html part, we define the divisions to hold the charts as follows :

```
        <div class="container">
            <h3>Information Breakdown:</h3>
            <div class="row">

                <div class='col' >
                    <div id="horizontal-bar-chart-states" style="width:
100%;"></div>
                </div>
                <div class='col'>
                    <div id="pie-chart" style="width: 85%; height: 600px;"></div>
                </div>

            </div>

            <div class="row">
                <div class='col'>
                    <div id="nested_pie_chart" style="width: 100%; height:
400px;"></div>
                </div>
            </div>

        </div>
```

Now we have to define the javascript part that creates the charts and put them in the corresponding divisions. We have 3 charts : one for population distribution gender-wise within the selected location (country/state), one for literate population distribution gender-wise within the selected location and one for literacy rate distribution gender-wise within the selected location.

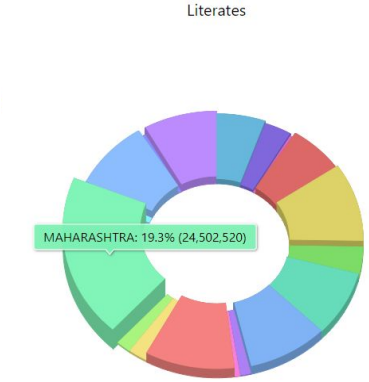
- Population distribution gender-wise (vertical-stacked- bar-chart) :




```
 am4core.useTheme(am4themes_animated);
                var chart01 = am4core.create("horizontal-bar-chart-states",
am4charts.XYChart);
                chart01.data = {{ state_data|safe }};
                var title = chart01.titles.create();
                title.text = "Population";
                title.fontSize = 20;

                var cellSize = 20;
                chart01.events.on("datavalidated", function (ev) {
                        // Get objects of interest
                        var chart = ev.target;
                        var categoryAxis = chart.yAxes.getIndex(0);
                        // Calculate how we need to adjust chart height
                        var adjustHeight = chart.data.length * cellSize -
categoryAxis.pixelHeight;
                        // get current chart height
                        var targetHeight = chart.pixelHeight + adjustHeight;
                        // Set it on chart's container
```

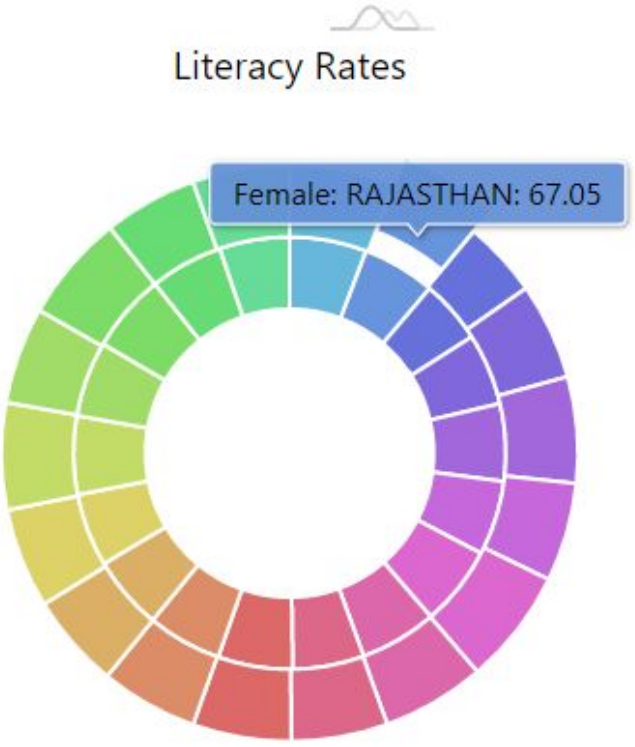
```
                        chart.svgContainer.htmlElement.style.height =
targetHeight + "px";
                    });

                // Create axes
                var categoryAxis = chart01.yAxes.push(new
am4charts.CategoryAxis());
                categoryAxis.dataFields.category = "location";
                categoryAxis.renderer.grid.template.opacity = 0;
                categoryAxis.renderer.labels.template.disabled = true;
                categoryAxis.renderer.minGridDistance = 300;

                var valueAxis = chart01.xAxes.push(new
am4charts.ValueAxis());
                valueAxis.renderer.labels.template.disabled = true;
                valueAxis.renderer.grid.template.opacity = 0;

                // Create series
                function createSeries01(name) {
                    var series = chart01.series.push(new
am4charts.ColumnSeries());
                    series.dataFields.valueX = "population_male";
                    series.dataFields.categoryY = "location";
                    series.stacked = true;
                    series.name = name;
                    series.columns.template.tooltipText = "{categoryY}:
{name} ({valueX.value})";

                }

                function createSeries02(name) {
                        var series = chart01.series.push(new
am4charts.ColumnSeries());
                        series.dataFields.valueX = "population_female";
                        series.dataFields.categoryY = "location";
                        series.stacked = true;
                        series.name = name;
                        series.columns.template.tooltipText = "{categoryY}:
{name} ({valueX.value})";
                    }

                createSeries01("MALE");
                createSeries02("FEMALE");
```

- Literate population distribution gender-wise (3D pie chart) :



```
    am4core.useTheme(am4themes_animated);
                var
pie_chart=am4core.create("pie-chart",am4charts.PieChart3D);
                pie_chart.hiddenState.properties.opacity = 0;
                var title = pie_chart.titles.create();
                title.text = "Literates";
                title.fontSize = 20;

                pie_chart.data={{literate_total_data|safe}};
                pie_chart.innerRadius = am4core.percent(40);
                pie_chart.depth = 120;

                // Add and configure Series
                var pieSeries = pie_chart.series.push(new
am4charts.PieSeries3D());
                pieSeries.dataFields.value = "population";
                pieSeries.dataFields.depthValue = "population";
                pieSeries.dataFields.category = "location";
                pieSeries.slices.template.cornerRadius = 5;
                pieSeries.colors.step = 3;
                // Disable ticks and labels
                pieSeries.labels.template.disabled = true;
                pieSeries.ticks.template.disabled = true;
```

- Literacy rate distribution gender-wise (Nested pie-chart) :



```
                am4core.useTheme(am4themes_animated);
                var chart02 = am4core.create("nested_pie_chart",
am4charts.PieChart);
                chart02.innerRadius = am4core.percent(40);
                // Add data
                chart02.data ={{state_lr|safe}} ;
                var title = chart02.titles.create();
                title.text = "Literacy Rates";
                title.fontSize = 20;

                // Add and configure Series
                var pieSeries = chart02.series.push(new am4charts.PieSeries());
                pieSeries.dataFields.value = "male";
                pieSeries.dataFields.category = "state";
                pieSeries.slices.template.stroke = am4core.color("#fff");
                pieSeries.slices.template.strokeWidth = 2;
                pieSeries.slices.template.strokeOpacity = 1;

                // Disabling labels and ticks on inner circle
                pieSeries.labels.template.disabled = true;

                // Disable sliding out of slices

pieSeries.slices.template.states.getKey("hover").properties.shiftRadius = 0;
                pieSeries.slices.template.states.getKey("hover").properties.scale =
0.9;

                // Add second series
                var pieSeries2 = chart02.series.push(new am4charts.PieSeries());
                pieSeries2.dataFields.value = "female";
                pieSeries2.dataFields.category = "state";
                pieSeries2.slices.template.stroke = am4core.color("#fff");
                pieSeries2.slices.template.strokeWidth = 2;
                pieSeries2.slices.template.strokeOpacity = 1;

pieSeries2.slices.template.states.getKey("hover").properties.shiftRadius = 0;
                pieSeries2.slices.template.states.getKey("hover").properties.scale
= 1.1;

                pieSeries.slices.template.tooltipText = "Male: {category}:
{value}";
                pieSeries2.slices.template.tooltipText = "Female: {category}:
{value}";
                // Disabling labels and ticks on inner circle
                pieSeries2.labels.template.disabled = true;
```

9. The ajax calls for asynchronous updates:

Whenever the selected item in the drop-down list changes, a function stateChange() is called which in-turn sets up an ajax call that ultimately updates the stats value and graphs in the dashboard page.

```
function stateChange(selectObj){
                var idx=selectObj.selectedIndex;
                var option=selectObj.options[idx].value;
                $.ajax({
                    type:"GET",
                    url: "select_page",
                    data:{
                        'value':option,
                        'index':idx
                    },
                    dataType: 'json',
                    success:  function(result){

document.getElementById("total_population").innerHTML =
result['population'];
                        document.getElementById("total_literates").innerHTML
= result['literates'];

document.getElementById("total_literacy_rate").innerHTML =
result['literacy_rate'];
                        document.getElementById("total_location").innerHTML
= result['no_of_states'];

                        chart01.data=result['state_data'];
                        pie_chart.data = result['literate_total_data'] ;
                        chart02.data = result['state_lr'] ;
                    }
                });
            }
```

In the ajax call, we define the structure of the call, by type, we define the type of method we need to use to send the data to the server. By url, we specify the url to which we need the page to send the data. Next is the data, which is a data-structure containing the data we need to send to the server. By dataType we mention the type of data we are expecting from the server. And finally the success part of the call, specifies the steps that must be executed in case the server successfully responds. In our case we update the values in our page and the graphs.

10. **Deploying it using the apache server:**
    Finally, we now need to deploy our dashboard app. For the deployment purpose, we have 2 options:
    1. Using django server
    2. Using other server

The django server is appropriate for development purposes. But for deployment purposes it is generally not preferred. In our project, we will be using the apache server to deploy our app.

a. Download the apache server file from the following website (https://www.apachelounge.com/download/). This will download a zipped file. Unzip the downloaded file, which will yield 1 folder (Apache24) and 2 files.
b. In C drive, create a folder named 'apache24' and copy the Apache24 folder from the unzipped folder to the apache24 folder.
c. The next step is adding the apache folder to the system path. Also add an environment variable (MOD_WSGI_APACHE_ROOT_DIR C:\apache24\Apache24).
d. After it is added to the system's path, now we can install the apache server in our system using the following command:

httpd.exe -k install

e. Next in-order to link our django app to the apache server, we need to first edit the content of the settings.py in our django project folder. In the file, add/edit the following lines:

```
DEBUG = False
ALLOWED_HOSTS = ["localhost","*","127.0.0.1"]
WSGI_APPLICATION = 'dashboard.wsgi_windows.application'

STATIC_URL = '/static/'
STATIC_ROOT=os.path.join(BASE_DIR, 'static')
```

f. After editing, run the following command:

```
python manage.py collectstatic
```

This will create the static folder in the django project directory.

g. Next we need to install the following package in our system:

```
pip install mod_wsgi
```

After installing the package, we execute the following command:

```
mod_wsgi-express module-config
```

The output is of the following format:

```
LoadFile
"c:/users/myuser/appdata/local/programs/python/python37/python37.dll"
LoadModule wsgi_module
"c:/users/myuser/appdata/local/programs/python/python37/lib/site-packages/mod_wsgi/server/mod_wsgi.cp37-win_amd64.pyd"
WSGIPythonHome
"c:/users/myuser/appdata/local/programs/python/python37"
```

h. In the django project folder, add a file : wsgi_windows.py and add the following lines of code:

```
import os
```

```
import sys
import site
from django.core.wsgi import get_wsgi_application

# Add the app's directory to the PYTHONPATH
sys.path.append('C:/Users/Admin/Desktop/charts_django/rsac_dashboard/Census_
Dashboard/dashboard')
sys.path.append('C:/Users/Admin/Desktop/charts_django/rsac_dashboard/Census_
Dashboard/dashboard/board')

os.environ['DJANGO_SETTINGS_MODULE'] = 'dashboard.settings'
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'dashboard.settings')

application = get_wsgi_application()
```

i. The output of step 'g' is appended to the httpd.conf file available at the location:
C: > apache24 > Apache24 > conf > httpd.conf
After appending the lines, we need to uncomment and add the following lines:

```
Define SRVROOT "c:/apache24/Apache24"
ServerName localhost
Include conf/extra/httpd-vhosts.conf
```

j. Next we need to edit the httpd-vhosts.conf file available at location:
C: > apache24 > Apache24 > conf > extra > httpd-vhosts.conf

Add the following lines of code:

```
<VirtualHost *:80>
    ServerName localhost
    ServerAlias localhost

    WSGIScriptAlias /
"C:\Users\Admin\Desktop\charts_django\rsac_dashboard\Census_Dashboard\dashbo
ard\dashboard\wsgi_windows.py"

ErrorLog
"C:\Users\Admin\Desktop\charts_django\rsac_dashboard\Census_Dashboard\dashbo
ard\dashboard\dashboard.error.log"
CustomLog
"C:\Users\Admin\Desktop\charts_django\rsac_dashboard\Census_Dashboard\dashbo
ard\dashboard\dashboard.access.log" combined

<Directory
"C:\Users\Admin\Desktop\charts_django\rsac_dashboard\Census_Dashboard\dashbo
ard\dashboard">
        <Files wsgi_windows.py>
            Require all granted
        </Files>
</Directory>
```

```
Alias /static
"C:\Users\Admin\Desktop\charts_django\rsac_dashboard\Census_Dashboard\dashbo
ard\static"

<Directory
"C:\Users\Admin\Desktop\charts_django\rsac_dashboard\Census_Dashboard\dashbo
ard\static">
    Require all granted
</Directory>
</VirtualHost>
```

The paths in the above lines of codes must be modified according to the location of the django app in the system.

k. Then in-order to load the latest changes, we need to reload the apache server using the following command:
(to start the apache server, httpd.exe -k start)
httpd.exe -k restart

l. Finally go to the website and watch your dashboard in action.