ORACLE

Manage Offensive Behavior Using AI Language, Speech, and Video with Oracle APEX- Application Step by Step Guide

Piotr Kurzynoga, Data Development Specialist, Oracle
Bob Peulen, Data Science and ML Specialist, Oracle

2023 | Version 1.01

Disclaimer

Table of Contents

OCI Pre-requisites:

-Create Object Storage Bucket.

-Provision an Autonomous Database -> Provision an APEX Instance with a workspace.

-Copy the Data Science notebook.

-Create an API key for your OCI user, this will be used for authenticating OCI Services from within APEX.

-Ensure all the correct OCI Policies are in place.

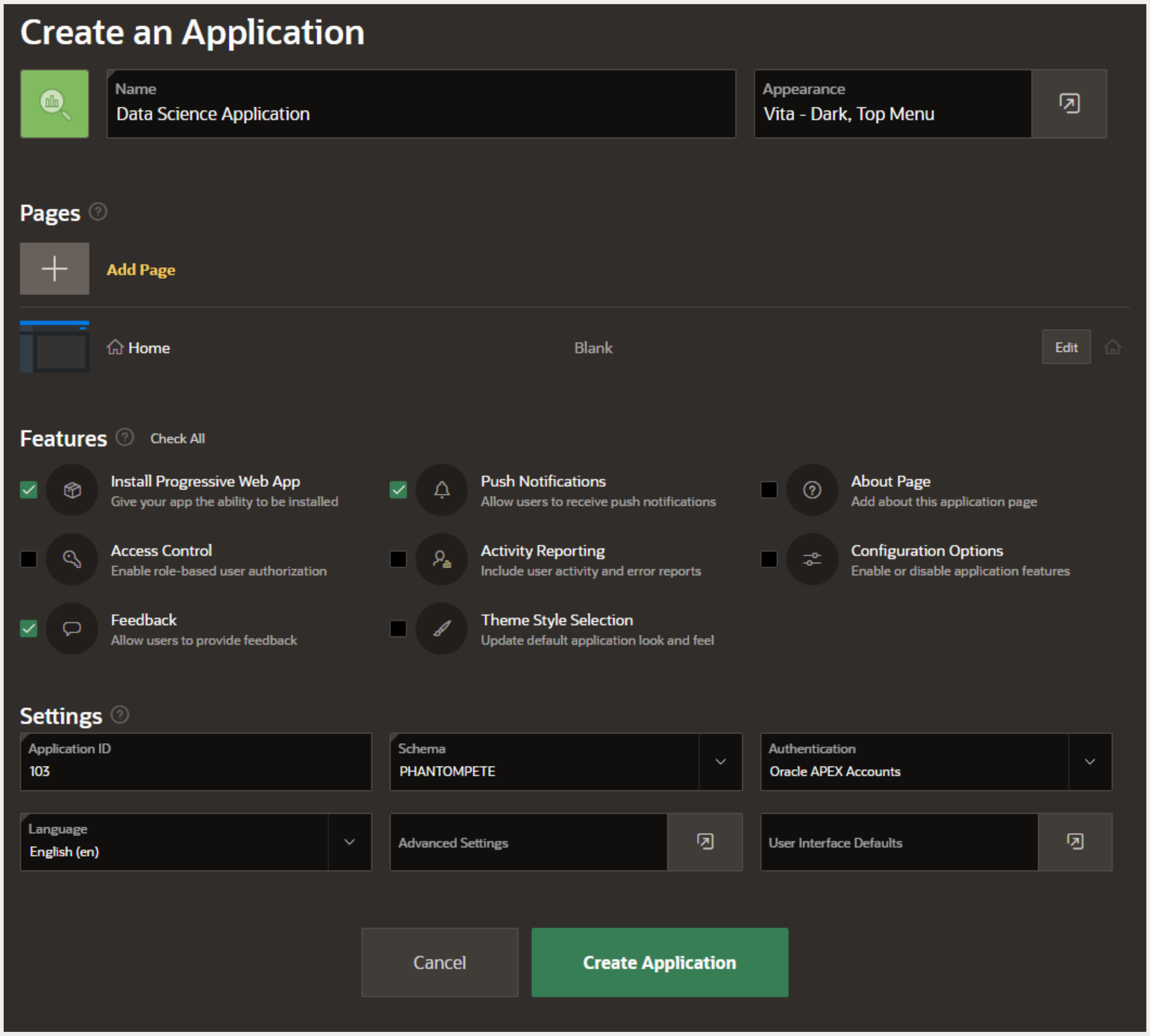
Collateral that can help:

- https://blogs.oracle.com/ai-and-datascience/post/yolov5-models-in-apex-using-oracle-data-science
- https://medium.com/@devpiotrekk/uploading-files-to-oci-object-storage-via-apex-42ad396ec55d

APEX – Integrating Data Science Jobs for Offensive Behavior Analysis

New Application – First Steps

Create a new application, if you're not sure how to get to this step I recommend following the Oracle University Free APEX Learning Path: https://mylearn.oracle.com/learning-path/oracle-apex-foundations/112444


Create an Application

Name: Data Science Application
Appearance: Vita - Dark, Top Menu

Pages
Add Page
Home — Blank — Edit

Features — Check All

- ✓ Install Progressive Web App — Give your app the ability to be installed
- ✓ Push Notifications — Allow users to receive push notifications
- About Page — Add about this application page
- Access Control — Enable role-based user authorization
- Activity Reporting — Include user activity and error reports
- Configuration Options — Enable or disable application features
- ✓ Feedback — Allow users to provide feedback
- Theme Style Selection — Update default application look and feel

Settings

Application ID: 103
Schema: PHANTOMPETE
Authentication: Oracle APEX Accounts
Language: English (en)
Advanced Settings
User Interface Defaults

Application Blueprint – Can be used to re-create the application with same settings.

Authentication - Create Web Credentials

As we will be working with different API calls we need a way to authenticate with our OCI tenancy user, setting up web credentials is an easy way to re-use the credential in multiple REST Data Sources.

1. Navigate to App Builder -> Workspace Utilities -> Web Credentials.

2. Create a new OCI credential using the details obtained from the OCI API Key and Save.

It is important to include the "Static Identifier" which will later be used to create the API request to Object Storage.

Note: Whenever you make a change to the Web Credentials the OCI Private Key needs to be re-added.

Go back to the newly created application and navigate to Shared Components.

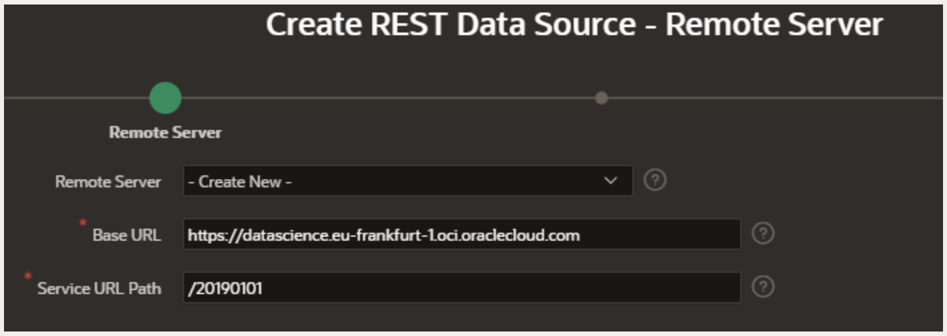
Data Science Integration - Create Data Source

We will now create two data sources, one responsible for creating the Data Science job and one for retrieving its status.

1. In Shared components open REST Data Sources and press Create and next immediately.
2. Leave the REST Data Source type as "Simple HTTP" For the name specify "CreateJobRun", URL Endpoint should follow the format "https://datascience.**eu-frankfurt-1**.oci.oraclecloud.com/**20190101**"
   where "eu-frankfurt-1" is replaced with your OCI Tenancy Region.

The **20190101** simply refers to the API catalog, more here https://docs.oracle.com/en-us/iaas/api/#/en/data-science/20190101/

Ultimately, we should have the format as follows:



Create REST Data Source - Remote Server

Remote Server

Remote Server	- Create New -
* Base URL	https://datascience.eu-frankfurt-1.oci.oraclecloud.com
* Service URL Path	/20190101

3. Press Next until you reach Authentication, toggle the switch and select the Web Credentials you have previously created.

Press Discover and Create REST Data Source (Disregard the errors at this point).

Now we need to add some parameters, open the created operation and edit the GET operation

In the URL Pattern enter “/jobRuns”

Change the HTTP method from GET to POST and paste the following body:

```
{
  "projectId": "#projectId#",
  "compartmentId": "#compartmentId#",
  "jobId": "#jobId#",
  "definedTags": {},
  "displayName": "#displayName#",
  "freeformTags": {},
  "jobConfigurationOverrideDetails": {
    "jobType": "DEFAULT",
    "environmentVariables": {
      "TYPE_OF_ANALYSIS": "#ANALYSIS_TYPE#",
      "YOUTUBE_URL": "#YOUTUBE_URL#",
      "NAMESPACE_NAME" : "#BUCKET_NAMESPACE#",
      "MAIN_BUCKET_NAME": "#BUCKET_NAME#",
      "SCHEMA_NAME": "#SCHEMA_NAME#"}
    }
  }
}
```

Now press Synchronize with Body and confirm with “OK”.

You will now see a list of parameters, these are defined with the hashtags(#) around them.

Add another parameter manually called “Response” with type “Request or Response Body” and “Out” as the direction, this will hold our response from the API call.

Lastly add a “HTTP Header” with Name “Content-Type” and static value “application/json”.

Now press Apply Changes twice.

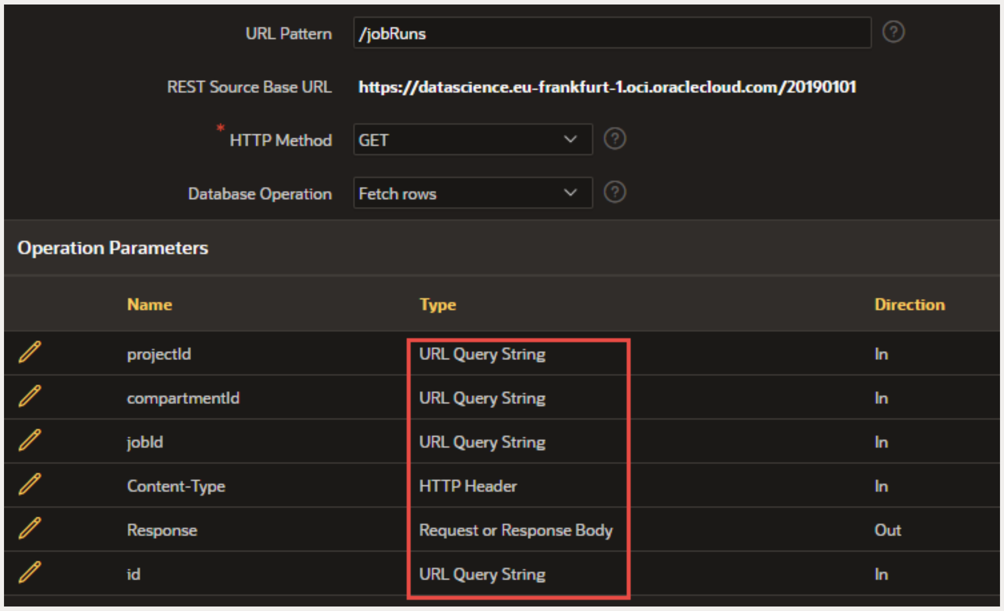
Instead of going through the same process twice we will now press Copy and select our existing CreateJobRun Data Source, provide a New Name for the Copy such as “ListJobRuns” and press copy.

Paste the following body and synchronize with body. This will remove the unwanted parameters from the existing list.

```
{
  "id": "#id#",
  "projectId": "#projectId#",
  "compartmentId": "#compartmentId#",
  "jobId": "#jobId#",
}
```

Change the projectId, compartmentId, jobId, id parameters Type from Request or Response Body to URL Query String

Now edit the POST operation changing it to GET and apply changes twice.



URL Pattern	/jobRuns
REST Source Base URL	https://datascience.eu-frankfurt-1.oci.oraclecloud.com/20190101
* HTTP Method	GET
Database Operation	Fetch rows

Operation Parameters

Name	Type	Direction
projectId	URL Query String	In
compartmentId	URL Query String	In
jobId	URL Query String	In
Content-Type	HTTP Header	In
Response	Request or Response Body	Out
id	URL Query String	In

User Interface

We are now ready to start modelling the UI, go back to the Application and edit the Home page.

Create a Region under Data Science Application with the name “Container”.

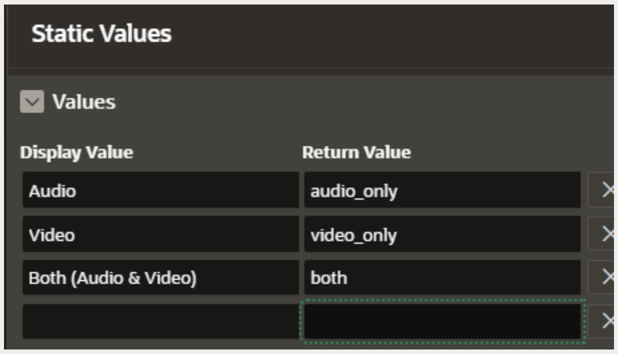
Under “Container” create 6-page items with the following properties:

VIDEO_URL (Text Field)

ALGORITHM (Select List)

Settings -> Page Action on Selection -> Submit Page

For the List of Values choose static, provide the following values and uncheck Display Extra/Null values.



Static Values

Values

Display Value	Return Value
Audio	audio_only
Video	video_only
Both (Audio & Video)	both

PICTURE (File Browse..)

Server-Side Condition -> Type “Item is in colon delimited list” -> Item “ALGORITHM” -> List “video_only:both”

JOBRUN_RESPONSE (Hidden)

LIFECYCLESTATE (Display Only)

Label -> “Lifecycle State”

Settings -> Send On Page Submit (Toggle to Off)

LIFECYCLEDETAILS (Display Only)

Label -> “Lifecycle Details”

Now create 2 buttons with the following properties:

Analyze (Button)

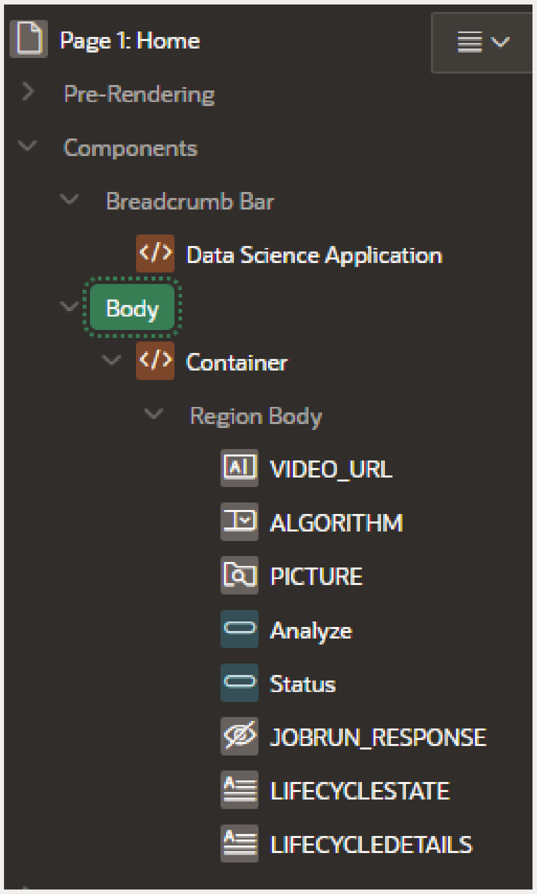
Advanced -> Static ID “ANALYZE”

Status (Button)

Behavior -> Redirect to URL -> Target “javascript:fn_click_trigger();”

Advanced -> Static ID “STATUS”

This is how the page layout should look like now:



Page 1: Home
- Pre-Rendering
- Components
  - Breadcrumb Bar
    - Data Science Application
  - Body
    - Container
      - Region Body
        - VIDEO_URL
        - ALGORITHM
        - PICTURE
        - Analyze
        - Status
        - JOBRUN_RESPONSE
        - LIFECYCLESTATE
        - LIFECYCLEDETAILS

Processing

Go to the Processing Tab and within Processing create 4 processes

Name “JobProcess” of type “Execution Chain”

Server-side Condition -> When Button Pressed -> Analyze

Name “UploadPictureSQL” of type “Execute Code” (Select JobProcess as Execution Chain)

Server-side Condition -> Type “Item is NOT NULL” -> Item “PICTURE”

Code:

```
declare
    l_request_url varchar(32000);
    l_content_length number;
    l_response clob;
    upload_failed_exception exception;
    l_request_object blob;
    l_request_filename varchar2(500);
begin
    select blob_content, filename into l_request_object, l_request_filename
from apex_application_temp_files where name = :PICTURE;
    l_request_url := 'https://objectstorage.eu-frankfurt-
1.oraclecloud.com/n/namespace/b/bucket_name/o/folder_name /' ||
apex_util.url_encode(l_request_filename);
    l_response := apex_web_service.make_rest_request(
        p_url => l_request_url,
        p_http_method => 'PUT',
        p_body_blob => l_request_object,
        p_credential_static_id => 'OCI_API' --Update with your Web
Credentials
    );
end;
```

Name “CreateJobRun” of type “Invoke API” & “REST Source” as CreateJobRun with POST operation (Select JobProcess as Execution Chain)

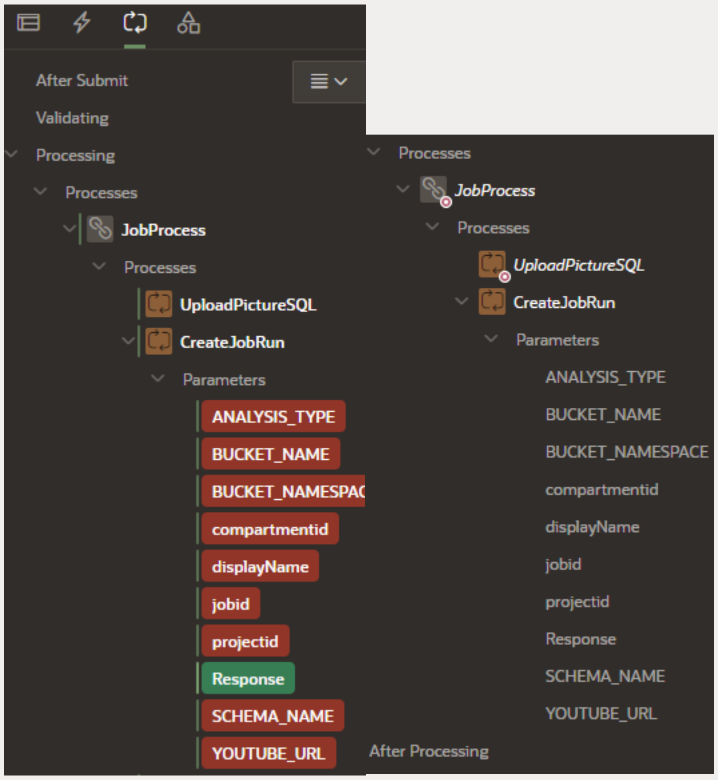
Parameters (Configure Data Science Parameters as required, you should obtain those from **Lab 3**):

ANALYSIS_TYPE (Item -> ALGORITHM)

YOUTUBE_URL (Item -> VIDEO_URL)

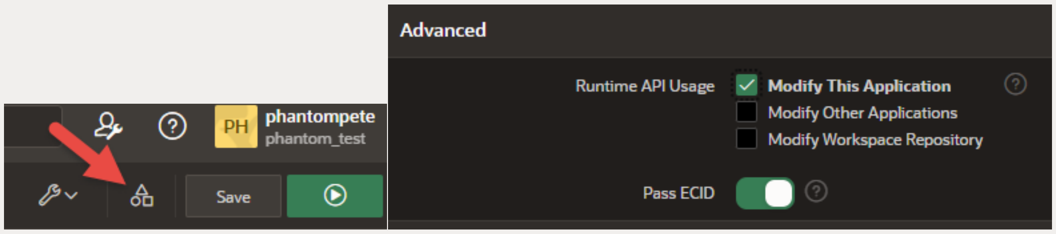
Response (Item -> JOBRUN_RESPONSE)

The result should look as follows:



Now go to shared components either from the application home page or by using the shortcut in the editor go to Security -> Security Attributes -> Advanced and toggle "Modify This Application".

This allows some of our background code to modify the Display Only fields in our application when doing processing.



Advanced

Runtime API Usage: [✓ Modify This Application] Modify Other Applications Modify Workspace Repository

Pass ECID: On

Improving the look and feel

Go to the Rendering tab and select the “Data Science Application” Breadcrumb Bar, in the options go to Appearance -> Template Options and choose Style as Stacked Featured

Now select the “Container” region. In the Region options go to -> Appearance -> Template Options and adjust the following:

Select Remove Body Padding

Change Header to Hidden

Change style to Stack Region

Press OK & Save.

Now select the PICTURE item and adjust the Item options.

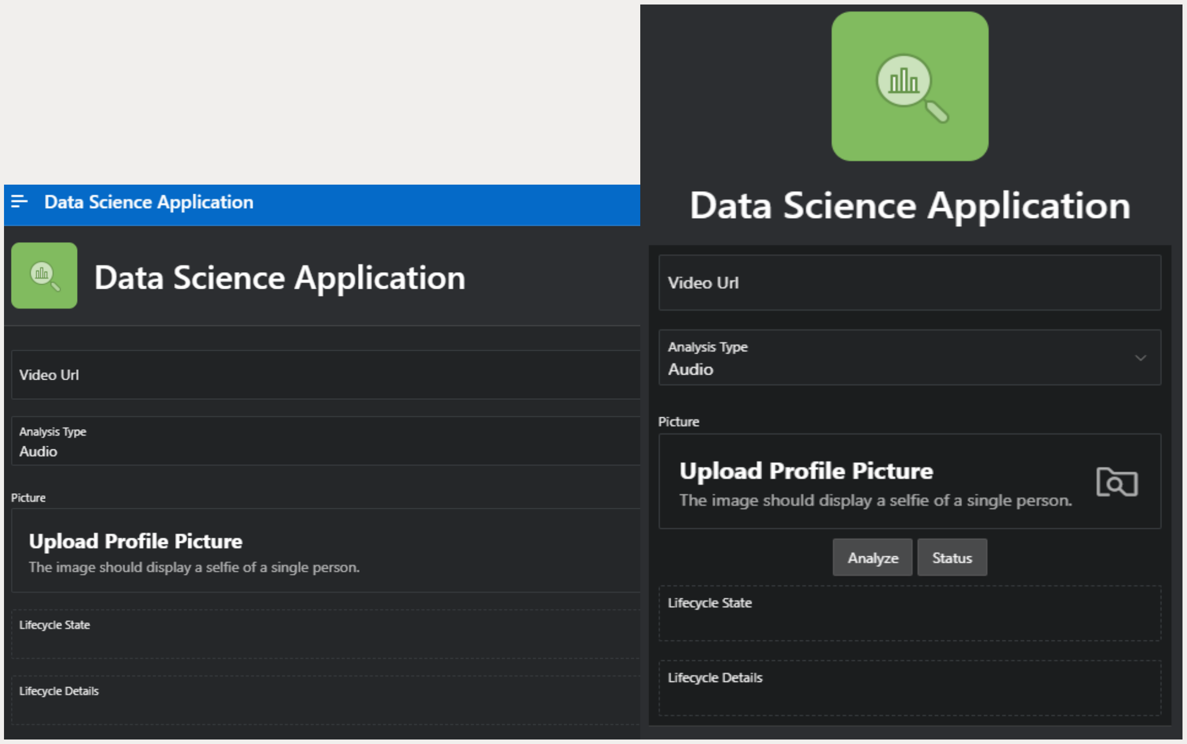
Settings -> Display as Inline Dropzone

Dropzone Title “Upload Profile Picture”

Dropzone Description insert “The image should display a selfie of a single person.”

File Types “.jpg”

Before and after the modifications:



Data Science Application

Data Science Application

Video Url

Analysis Type
Audio

Picture

Upload Profile Picture
The image should display a selfie of a single person.

Lifecycle State

Lifecycle Details

Data Science Application

Video Url

Analysis Type
Audio

Picture

Upload Profile Picture
The image should display a selfie of a single person.

Lifecycle State

Lifecycle Details

Time to fetch the status

In the Processing tab create a new “Execute Code” process under AJAX Callback and name it “GetStatus”.

Add a Server-side Condition:

Type: “Item is NOT NULL”

Item: “JOBRUN_RESPONSE”

Use the following code:

```
DECLARE
    l_params apex_exec.t_parameters;
    j apex_json.t_values;
    lifecyclestate varchar2(50);
    lifecyclestatus varchar2(250);
BEGIN
    apex_json.parse(j, :JOBRUN_RESPONSE);
    apex_exec.add_parameter( l_params, 'id',
apex_json.get_varchar2(p_path=>'id',p0=> 3,p_values=>j) );
    apex_exec.add_parameter( l_params,
'jobId',apex_json.get_varchar2(p_path=>'jobId',p0=> 3,p_values=>j) );
    apex_exec.add_parameter( l_params,
'compartmentId',apex_json.get_varchar2(p_path=>'compartmentId',p0=>
3,p_values=>j) );
    apex_exec.add_parameter( l_params,
'projectId',apex_json.get_varchar2(p_path=>'projectId',p0=> 3,p_values=>j) );

    apex_exec.execute_rest_source(
        p_static_id        => 'listjobruns', --Get this static ID from the REST
Data Source
        p_operation        => 'GET',
        p_parameters       => l_params );

     apex_json.parse(j, apex_exec.get_parameter_clob(l_params,'Response'));
     apex_json.open_object;

     lifecyclestate := apex_json.get_varchar2(p_path=>'[%d].lifecycleState',p0 =>
1,p_values=>j);
     lifecyclestatus := apex_json.get_varchar2(p_path=>'[%d].lifecycleDetails',p0
=> 1,p_values=>j);

     apex_json.write( p_name  => 'lifeState', p_value => lifecyclestate);
     apex_json.write( p_name  => 'lifeStatus', p_value => lifecyclestatus);

     apex_json.close_object;
     :LIFECYCLESTATE := lifecyclestate; --Persist update of status.

     IF (lifecyclestate = 'SUCCEEDED') THEN
         :JOBRUN_RESPONSE := apex_json.get_varchar2(p_path=>'id',p0=>
3,p_values=>j);
     END IF;

END;
```

Now in the Rendering tab select the root entry (Home) and copy the following in the options properties JavaScript Function and Global Variable Declaration:

```
var myInterval; //Stores the interval variable

function fn_click_trigger() {
    myInterval = setInterval(fn_click, 5000); //Refresh every 5 seconds
    apex.item('ANALYZE').hide();
}

function fn_click_stop() {
    clearInterval(myInterval);
}

function fn_click() {
    // Call Ajax process to execute some PL/SQL code
    apex.server.process('GetStatus' //PL/SQL process name
        , { x01: $v('JOBRUN_RESPONSE') }
        , {
            dataType: 'json'
            //, async : false
            , success: function (pData) {
                //Store the PL/SQL result in a Page item
                apex.item("LIFECYCLESTATE").setValue(pData.lifeState);
                apex.item("LIFECYCLEDETAILS").setValue(pData.lifeStatus);
            }
        }
    ).done(function (pData) {
        // Do the post processing here, next set of code post PL/SQL call
        if (pData.lifeState == "SUCCEEDED") {
            fn_click_stop();
            apex.submit('STATUS');
            apex.item('ANALYZE').show();
        }

        if (pData.lifeState == "FAILED") {
            fn_click_stop();
            apex.item('ANALYZE').show();
        }
    }
    );
}
```

Now we're able to call our Process via an asynchronous call through AJAX, which means we can continue our work as usual while the magic happens in the background!

Test Run

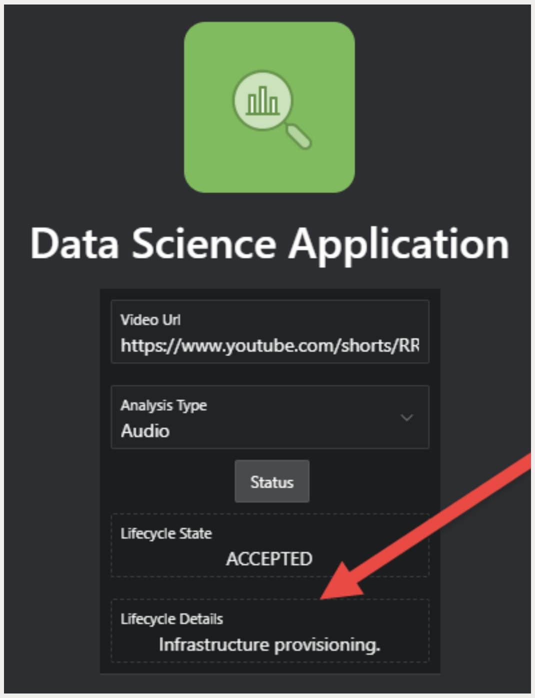
Now we can test the setup, for the test let's use the following video and paste it in the Video URL:

https://www.youtube.com/embed/epU3ipIBSqA?autoplay=1&mute=1

Now select the analysis type and press Analyze, this will trigger the Data Science Job.

To see how our job is progressing press Status, this will call the JavaScript wrapper that triggers the Process in the background and updates our Lifecycle State and Lifecycle Details.

Note: The Analyze button disappears until the Lifecycle State does not go into SUCCEEDED or FAILED.



Data Science Application

Video Url: https://www.youtube.com/shorts/RR

Analysis Type: Audio

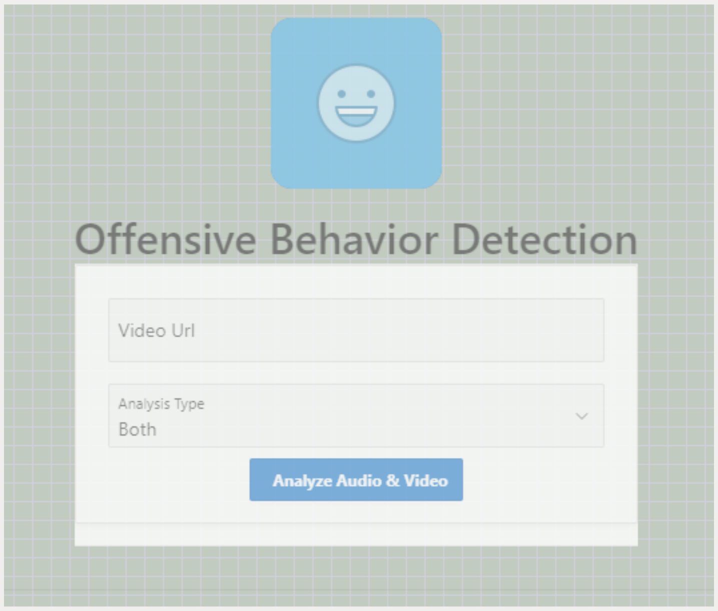
Lifecycle State: ACCEPTED

Lifecycle Details: Infrastructure provisioning.

If the status is "SUCCEEDED" you can then observe the results from your table, the results have been pushed to.

APEX Improvements – Further improvements to our application

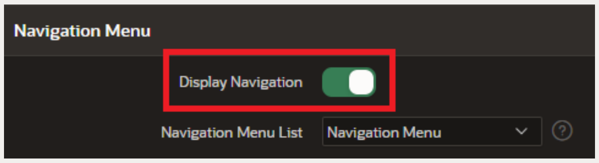
How to improve the UI (CSS, Universal Theme etc.)



Go back to the Home Page root entry in the Rendering tab and paste the following Inline CSS:

```
.t-Body{
background-color: #99a996;
opacity: 1;
background-image:  linear-gradient(#b4b4c2 0.7000000000000001px, transparent
0.7000000000000001px), linear-gradient(to right, #b4b4c2 0.7000000000000001px, #99a996
0.7000000000000001px);
background-size: 14px 14px;
}

#t_Body_title{
background-color: #99a996;
opacity: 0.8;
background-image:  linear-gradient(#b4b4c2 0.7000000000000001px, transparent
0.7000000000000001px), linear-gradient(to right, #b4b4c2 0.7000000000000001px, #99a996
0.7000000000000001px);
background-size: 14px 14px;
}
```

To make our application look even better let's remove the navigation, to do this:
Toggle (Off) Display Navigation in Shared Components -> User Interface -> User Interface Attributes

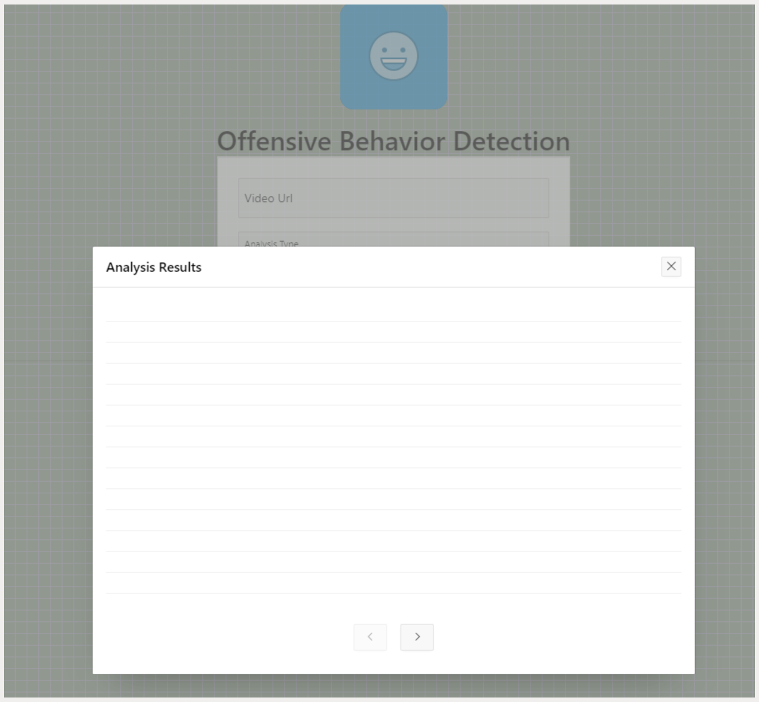

Navigation Menu

Display Navigation

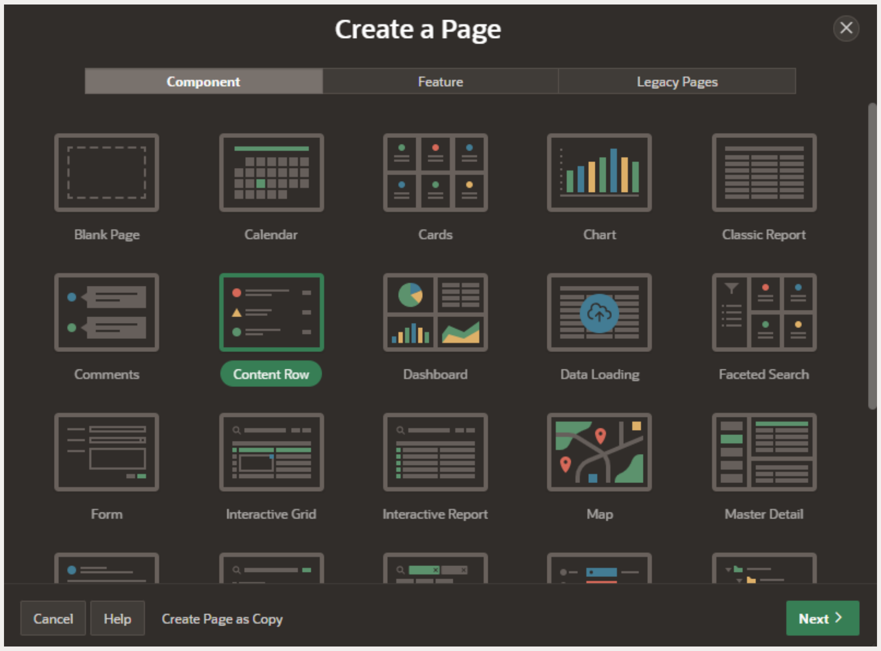
Navigation Menu List: Navigation Menu

Displaying & enchancing the data received from analysis (Navigating between pages and relaying data)

This is where we will handle the data that is returned from the Data Science job once it completes it's analysis.

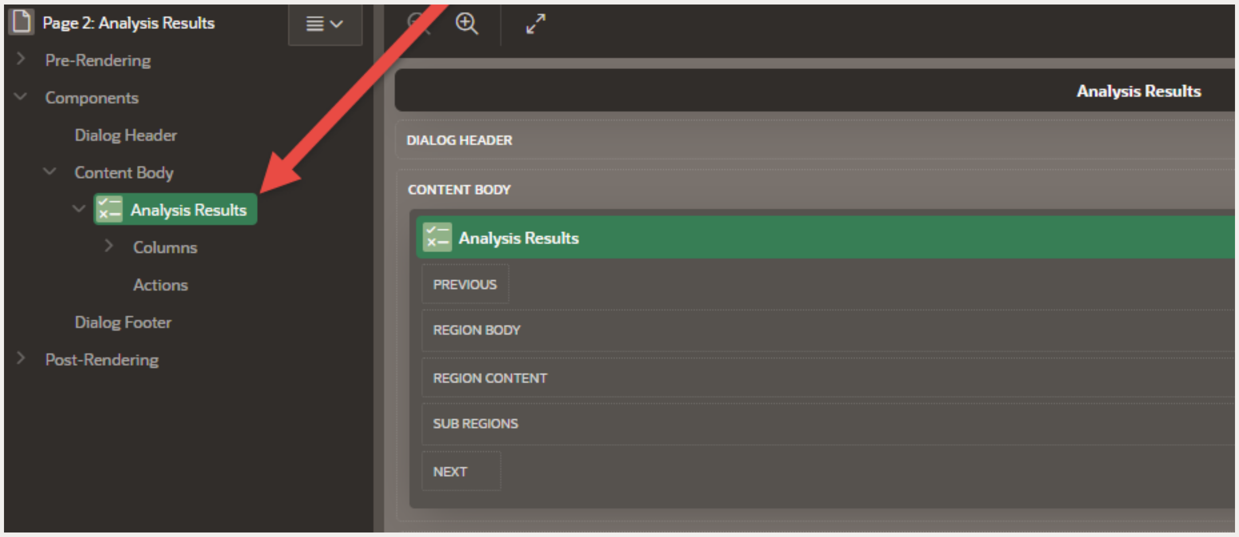


Press "Create Page" and select "Content Row"

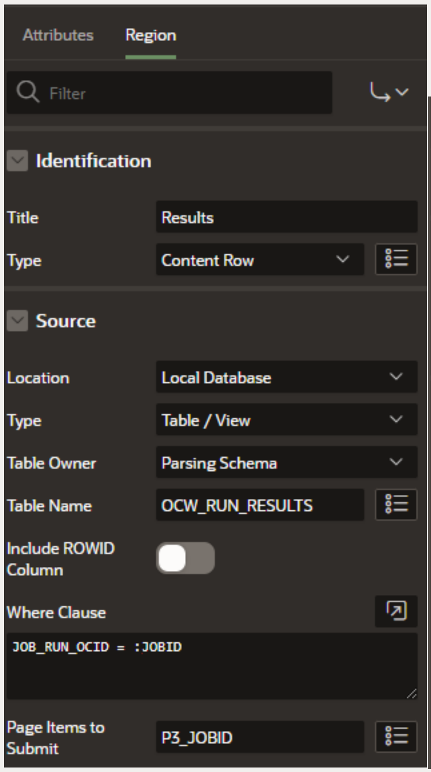


Give it a name "Analysis Results" and select Page Mode as "Modal Dialog", then press Create Page.

From the Rendering tab choose the Analysis Results "Content Row" region.



Configure the Attribute and Region parameters as follows, feel free to experiment with the Title, Description etc:

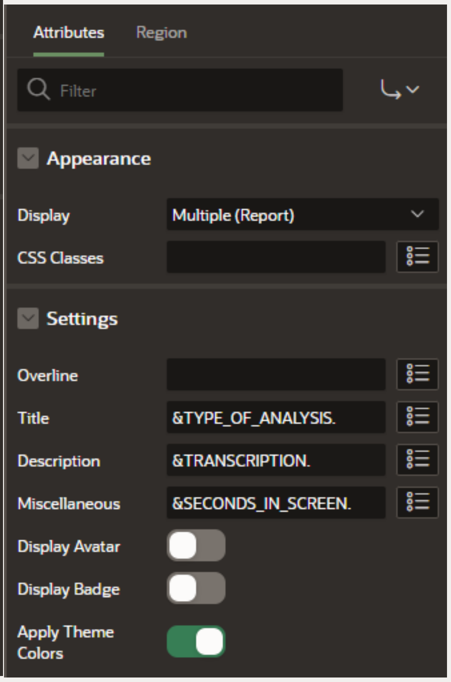


Identification

Title	Results
Type	Content Row

Source

Location	Local Database
Type	Table / View
Table Owner	Parsing Schema
Table Name	OCW_RUN_RESULTS
Include ROWID Column	Off
Where Clause	JOB_RUN_OCID = :JOBID
Page Items to Submit	P3_JOBID



Appearance

Display	Multiple (Report)
CSS Classes	

Settings

Overline	
Title	&TYPE_OF_ANALYSIS.
Description	&TRANSCRIPTION.
Miscellaneous	&SECONDS_IN_SCREEN.
Display Avatar	Off
Display Badge	Off
Apply Theme Colors	On

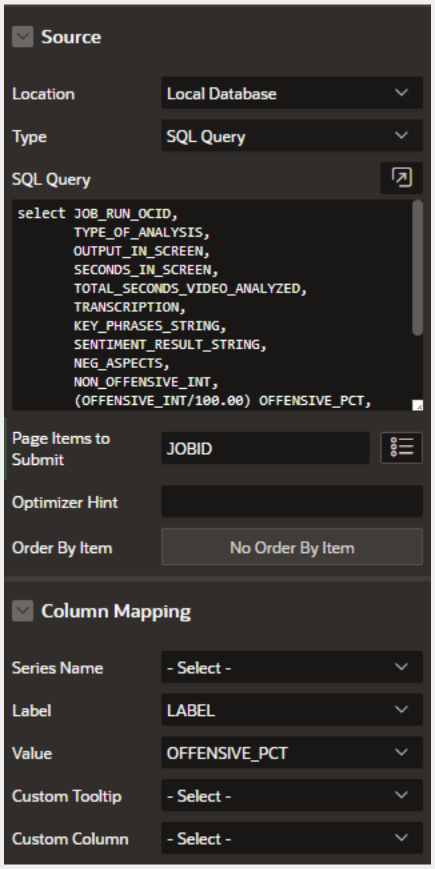
Create a "JOBID" Page item.

Now create a region of type Chart.

Title: "Offensive Indicators"

Attributes: Type "Pie"

Now configure the series as follows:



Source

Location: Local Database

Type: SQL Query

SQL Query

```
select JOB_RUN_OCID,
       TYPE_OF_ANALYSIS,
       OUTPUT_IN_SCREEN,
       SECONDS_IN_SCREEN,
       TOTAL_SECONDS_VIDEO_ANALYZED,
       TRANSCRIPTION,
       KEY_PHRASES_STRING,
       SENTIMENT_RESULT_STRING,
       NEG_ASPECTS,
       NON_OFFENSIVE_INT,
       (OFFENSIVE_INT/100.00) OFFENSIVE_PCT,
```

Page Items to Submit: JOBID

Optimizer Hint:

Order By Item: No Order By Item

Column Mapping

Series Name: - Select -

Label: LABEL

Value: OFFENSIVE_PCT

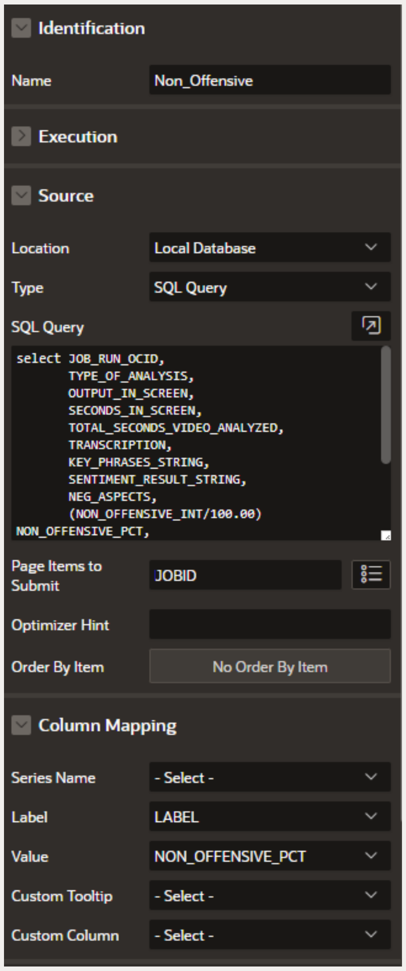
Custom Tooltip: - Select -

Custom Column: - Select -

Code for displaying Offensive percentage:

```
select JOB_RUN_OCID,
       TYPE_OF_ANALYSIS,
       OUTPUT_IN_SCREEN,
       SECONDS_IN_SCREEN,
       TOTAL_SECONDS_VIDEO_ANALYZED,
       TRANSCRIPTION,
       KEY_PHRASES_STRING,
       SENTIMENT_RESULT_STRING,
       NEG_ASPECTS,
       NON_OFFENSIVE_INT,
       (OFFENSIVE_INT/100.00) OFFENSIVE_PCT,
       NON_HATE_INT,
       HATE_INT,
       (select 'Offensive' from dual) LABEL
  from OCW_RUN_RESULTS
 where JOB_RUN_OCID = :JOBID
```

Then create a second series as follows:



Identification

Name: Non_Offensive

Execution

Source

Location: Local Database

Type: SQL Query

SQL Query

```
select JOB_RUN_OCID,
       TYPE_OF_ANALYSIS,
       OUTPUT_IN_SCREEN,
       SECONDS_IN_SCREEN,
       TOTAL_SECONDS_VIDEO_ANALYZED,
       TRANSCRIPTION,
       KEY_PHRASES_STRING,
       SENTIMENT_RESULT_STRING,
       NEG_ASPECTS,
       (NON_OFFENSIVE_INT/100.00)
NON_OFFENSIVE_PCT,
```

Page Items to Submit: JOBID

Optimizer Hint:

Order By Item: No Order By Item

Column Mapping

Series Name: - Select -

Label: LABEL

Value: NON_OFFENSIVE_PCT

Custom Tooltip: - Select -

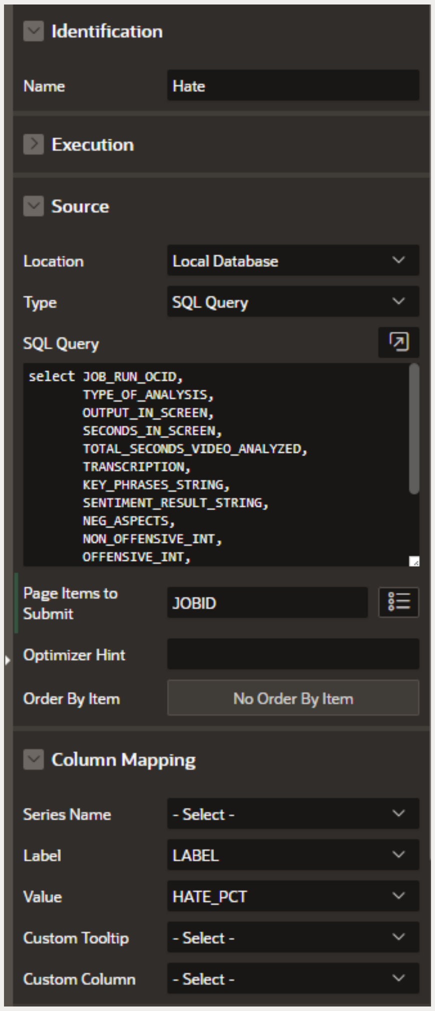
Custom Column: - Select -

Code for displaying Non Offensive percentage:

```
select JOB_RUN_OCID,
       TYPE_OF_ANALYSIS,
       OUTPUT_IN_SCREEN,
       SECONDS_IN_SCREEN,
       TOTAL_SECONDS_VIDEO_ANALYZED,
       TRANSCRIPTION,
       KEY_PHRASES_STRING,
       SENTIMENT_RESULT_STRING,
       NEG_ASPECTS,
       (NON_OFFENSIVE_INT/100.00) NON_OFFENSIVE_PCT,
       (OFFENSIVE_INT/100.00) OFFENSIVE_PCT,
       NON_HATE_INT,
       HATE_INT,
       (select 'Non Offensive' from dual) LABEL
  from OCW_RUN_RESULTS
 where JOB_RUN_OCID = :JOBID
```

Now right-click and duplicate the "Offensive Indicators" chart from the Rendering tab.

Adjust the Chart Title to "Hate Indicators" and configure the series as follows:



Identification

Name	Hate

Execution

Source

Location	Local Database
Type	SQL Query

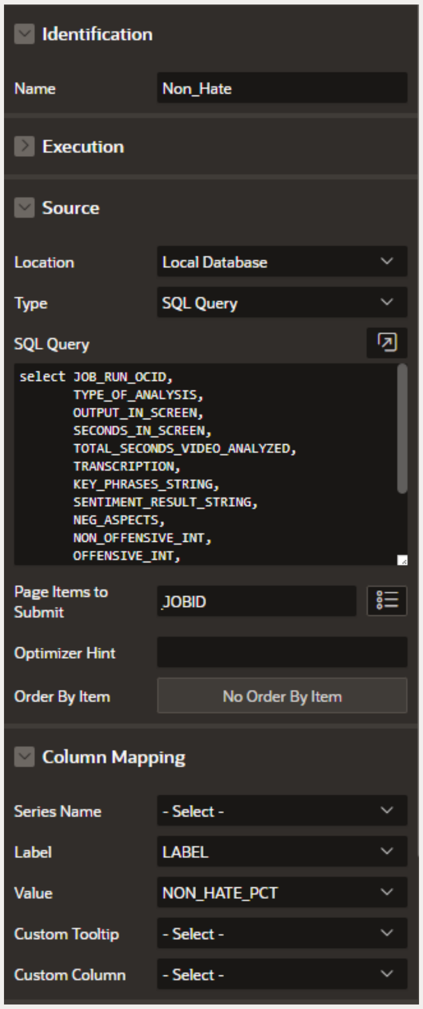
SQL Query

```
select JOB_RUN_OCID,
       TYPE_OF_ANALYSIS,
       OUTPUT_IN_SCREEN,
       SECONDS_IN_SCREEN,
       TOTAL_SECONDS_VIDEO_ANALYZED,
       TRANSCRIPTION,
       KEY_PHRASES_STRING,
       SENTIMENT_RESULT_STRING,
       NEG_ASPECTS,
       NON_OFFENSIVE_INT,
       OFFENSIVE_INT,
```

Page Items to Submit	JOBID
Optimizer Hint	
Order By Item	No Order By Item

Column Mapping

Series Name	- Select -
Label	LABEL
Value	HATE_PCT
Custom Tooltip	- Select -
Custom Column	- Select -



Identification

Name	Non_Hate

Execution

Source

Location	Local Database
Type	SQL Query

SQL Query

```
select JOB_RUN_OCID,
       TYPE_OF_ANALYSIS,
       OUTPUT_IN_SCREEN,
       SECONDS_IN_SCREEN,
       TOTAL_SECONDS_VIDEO_ANALYZED,
       TRANSCRIPTION,
       KEY_PHRASES_STRING,
       SENTIMENT_RESULT_STRING,
       NEG_ASPECTS,
       NON_OFFENSIVE_INT,
       OFFENSIVE_INT,
```

Page Items to Submit	JOBID
Optimizer Hint	
Order By Item	No Order By Item

Column Mapping

Series Name	- Select -
Label	LABEL
Value	NON_HATE_PCT
Custom Tooltip	- Select -
Custom Column	- Select -

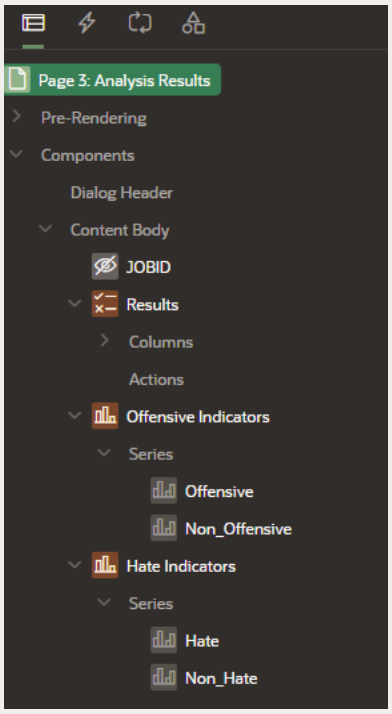
Code for displaying Hate percentage:

```
select JOB_RUN_OCID,
       TYPE_OF_ANALYSIS,
       OUTPUT_IN_SCREEN,
       SECONDS_IN_SCREEN,
       TOTAL_SECONDS_VIDEO_ANALYZED,
       TRANSCRIPTION,
       KEY_PHRASES_STRING,
       SENTIMENT_RESULT_STRING,
       NEG_ASPECTS,
       NON_OFFENSIVE_INT,
       OFFENSIVE_INT,
       NON_HATE_INT,
       (HATE_INT/100.00) HATE_PCT,
       (select 'Hate' from dual) LABEL
  from OCW_RUN_RESULTS
  where JOB_RUN_OCID = :JOBID
```

Code for displaying Non Hate percentage:

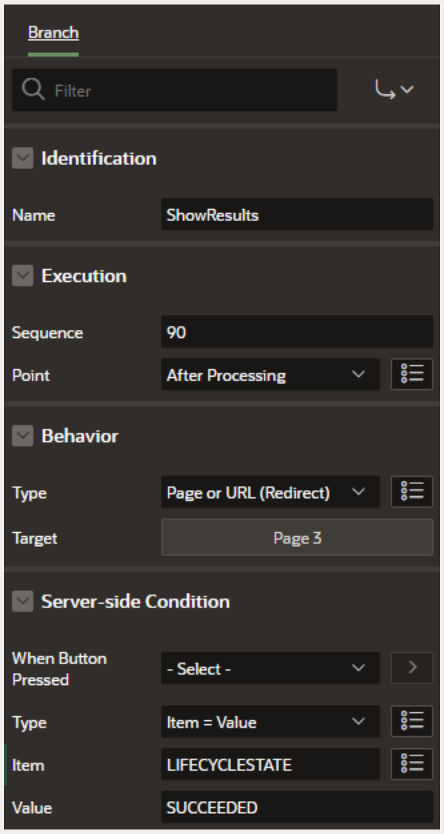
```
select JOB_RUN_OCID,
       TYPE_OF_ANALYSIS,
       OUTPUT_IN_SCREEN,
       SECONDS_IN_SCREEN,
       TOTAL_SECONDS_VIDEO_ANALYZED,
       TRANSCRIPTION,
       KEY_PHRASES_STRING,
       SENTIMENT_RESULT_STRING,
       NEG_ASPECTS,
       NON_OFFENSIVE_INT,
       OFFENSIVE_INT,
       (NON_HATE_INT/100.00) NON_HATE_PCT,
       (HATE_INT/100.00) HATE_PCT,
       (select 'Non Hate' from dual) LABEL
  from OCW_RUN_RESULTS
  where JOB_RUN_OCID = :JOBID
```

Now our Analysis Results page is completely created, you can continue to modify the look & feel of the charts and other elements of this page to your liking.



Now let's tell APEX it needs to open this page once we get a "SUCCEEDED" status in our Data Science Job and display the actual results.

For this navigate back to the home page, open the processing tab and create a branch in the After Processing, configure it as follows:



Identification

Name	ShowResults

Execution

Sequence	90
Point	After Processing

Behavior

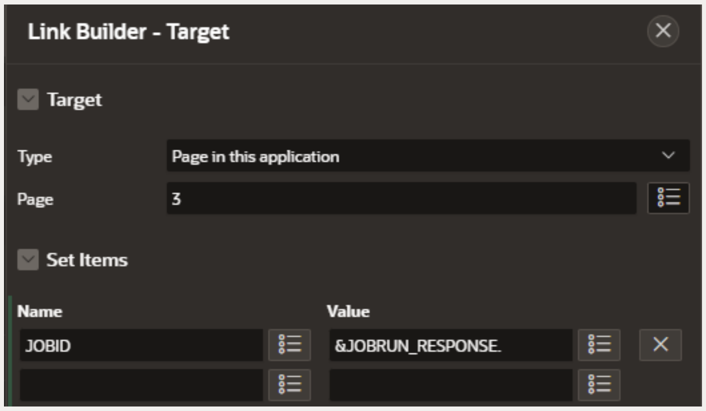
Type	Page or URL (Redirect)
Target	Page 3

Server-side Condition

When Button Pressed	- Select -
Type	Item = Value
Item	LIFECYCLESTATE
Value	SUCCEEDED

In the Behavior section choose type as "Page or URL (Redirect), select the target as a "Page in this application" and choose the created "Analysis Results" page as visible below.

Ensure that Set Items is configured to pass the "&JOBRUN_RESPONSE" parameter.



Link Builder - Target

Target

Type	Page in this application
Page	3

Set Items

Name	Value
JOBID	&JOBRUN_RESPONSE.

Once the data science job will be in the state succeeded our analysis results modal dialog will show up with the charts!

Previewing the Video in APEX

Create another region called Video_Preview of type "URL" with following Attributes:

URL - &VIDEO_URL.

IFrame Attributes: allowfullscreen width="100%" height="560"

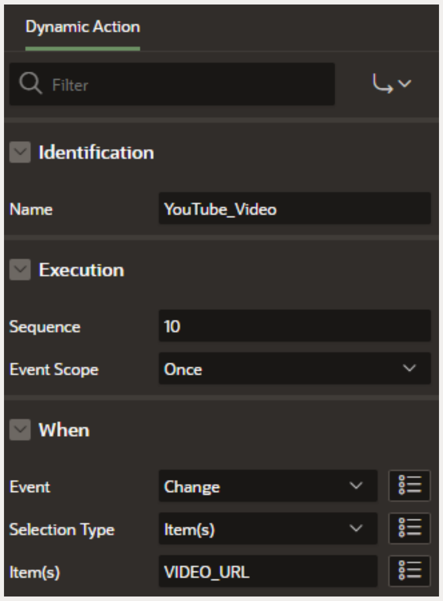
In the Advanced region options pass a Static ID as "PREVIEW"

Server-side Condition

Type – Item is NOT NULL and NOT ZERO

Item : VIDEO_URL

Now right click the VIDEO_URL page item and "Create Dynamic Action", configure it as follows:



Dynamic Action

Identification

Name	YouTube_Video

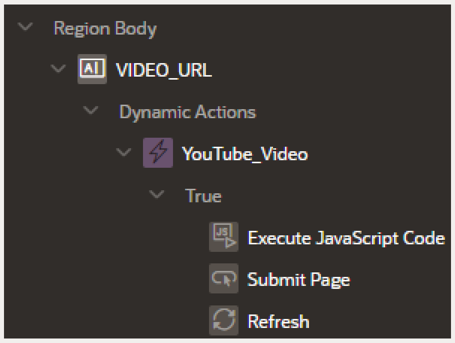
Execution

Sequence	10
Event Scope	Once

When

Event	Change
Selection Type	Item(s)
Item(s)	VIDEO_URL

Within the True Event, create three actions



```
Region Body
  VIDEO_URL
    Dynamic Actions
      YouTube_Video
        True
          Execute JavaScript Code
          Submit Page
          Refresh
```

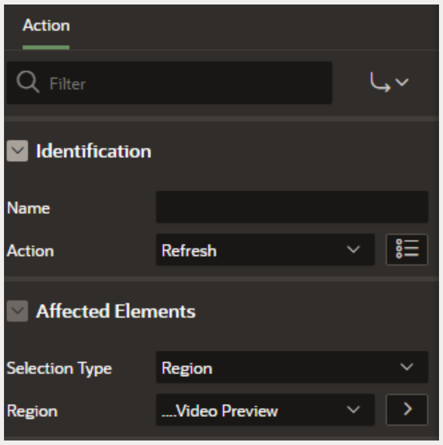
The first action “Execute JavaScript Code” having the following code:

```
var video=apex.items.VIDEO_URL.value;

video = video.replace('/shorts/', '/embed/');

video = video +'?autoplay=1&mute=1';

apex.item("VIDEO_URL").setValue(video);
```

This JS Script is aimed at YouTube shorts as it will modify their URL. You can adjust it for any type of YouTube video by modifying it.

Second action “Submit Page” is responsible for sending the URL to the Video Preview component.

Lastly the third action “Refresh”, aimed at refreshing the Video Preview region.



Name	
Action	Refresh
Selection Type	Region
Region	....Video Preview

Now you can preview your video inside APEX!

Now go back to the Livelab and follow **Lab5.**