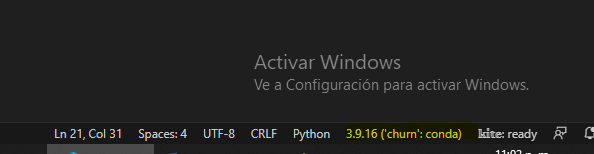
**Creating the streamlit application**

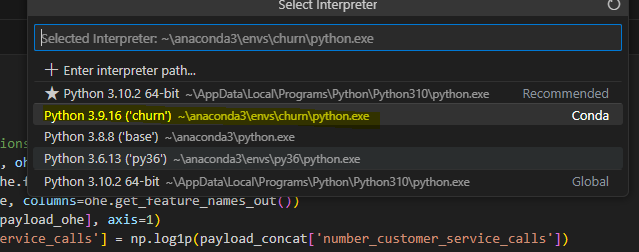
**I - Prerequisites**

If you did not know how to setup visual studio code for making it work for building python applications please follow this guide: <https://code.visualstudio.com/docs/datascience/data-science-tutorial>

For selecting the interpreter you could also click as noted in the figure below in yellow.



Next select the interpreter to use



And that’s it, you made it!.

**II – Make the utils function**

Open the app folder and build this uitility functions before inside the app folder as utils.py following this steps:

* Create a utils.py file
* Copy and paste the previous code of the transformed data as listed below
* Import pandas and numpy libraries
* Save the file

**import** **numpy** **as** **np**

**import** **pandas** **as** **pd**

# just storing all the others transformations to keep the prediction

**def** **transform\_data**(payload, col\_order\_out, mean\_eve\_min, ohe):

payload\_ohe = ohe.transform(payload[ohe.feature\_names\_in\_]).toarray()

payload\_ohe = pd.DataFrame(payload\_ohe, columns=ohe.get\_feature\_names\_out())

payload\_concat = pd.concat([payload, payload\_ohe], axis=**1**)

payload\_concat['log\_number\_customer\_service\_calls'] = np.log1p(payload\_concat['number\_customer\_service\_calls'])

payload\_concat['day\_ratio'] = payload\_concat['total\_day\_charge'] / payload\_concat['total\_day\_minutes']

payload\_concat['eve\_ratio'] = payload\_concat['total\_eve\_charge'] / payload\_concat['total\_eve\_minutes']

payload\_concat['night\_ratio'] = payload\_concat['total\_night\_charge'] / payload\_concat['total\_night\_minutes']

payload\_concat['intl\_ratio'] = payload\_concat['total\_intl\_charge'] / payload\_concat['total\_intl\_minutes']

payload\_concat['unhappy\_customers'] = ((payload\_concat['remaining\_term'] < **5**) &

(payload\_concat['last\_nps\_rating'] <= **7**) &

(payload\_concat.promotions\_offered == 'No')).astype(bool)

payload\_concat['total\_eve\_minutes\_missing'] = payload\_concat.total\_eve\_minutes.isna().astype(bool)

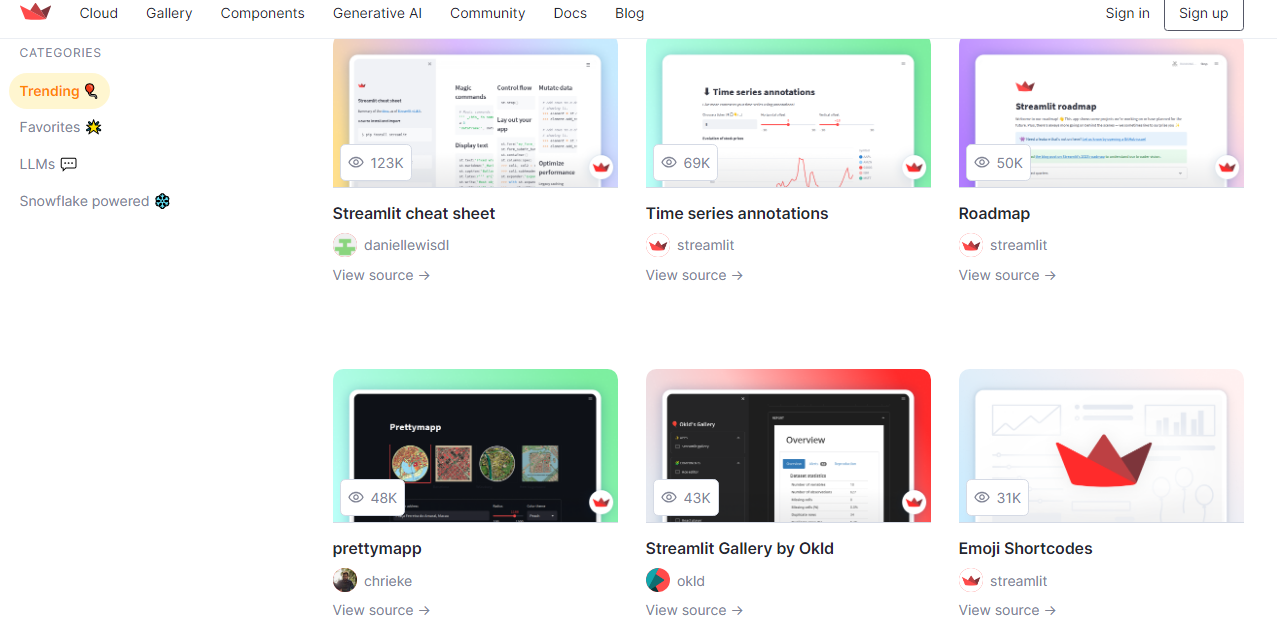
payload\_concat.total\_eve\_minutes.fillna(mean\_eve\_min, inplace=True)

payload\_transformed = payload\_concat[col\_order\_out]

**return** payload\_transformed

**III – Write the Streamlit App**

¿So what is streamlit and its use?



Long story short, Streamlit is a easy way to write applications, based in python and is oriented to datascience so its good to test applications and make them usable. If you want to do more please go to the streamlit website

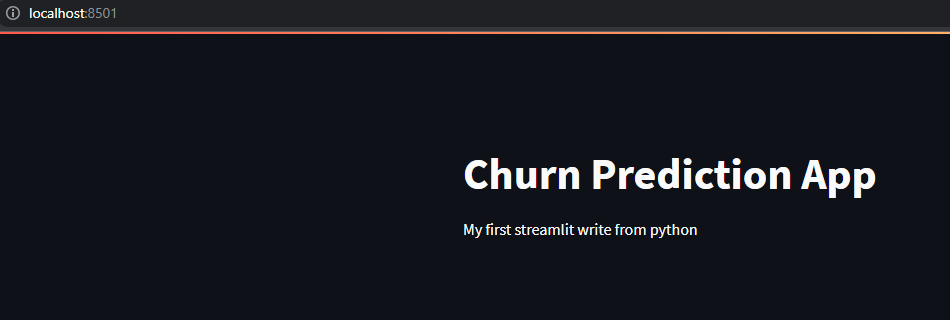
* Install streamlit via pip… pip install streamlit
* After installing streamlit open a file named app.py and write the code below

**import** **streamlit** **as** **st**

st.title('Churn Prediction App')

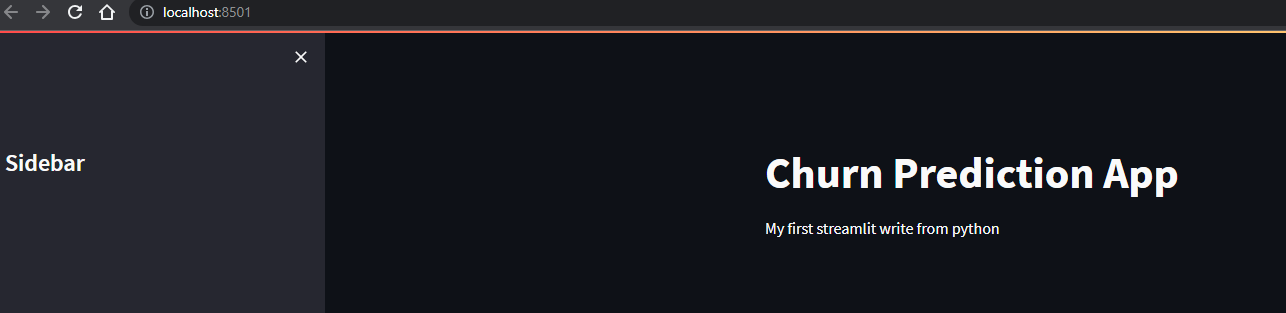
st.write('My first streamlit write from python')

* Next open the terminal (on the virtual environment) and run the app as the next step command below
* streamlit run app.py



Modify again the streamlit app.py to make a sidebar

st.sidebar.title('Sidebar')



**IV – Import Dependencies**

* Next paste the code below just to have the imports in order to use

**import** **streamlit** **as** **st**

**import** **pandas** **as** **pd**

**import** **os**

**import** **joblib**

**import** **json**

**from** **utils** **import** transform\_data

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

st.title('Churn Prediction App')

st.write('My first streamlit write from python')

st.sidebar.title('Sidebar')

* Rerun the app

**V – Load the Schema in JSON**

* Add some lines of code to test the schema

st.sidebar.title('Sidebar')

# load the schema

**with** open(os.path.join('schema.json'), 'r') **as** f: # reload the schema (testing it)

schema = json.load(f)

st.write(schema)



* Do the same and add the colum\_info to check the values in the window too.

#st.write(schema)

column\_order\_in = list(schema['col\_info'].keys())[**1**:-**1**] # extract column order

column\_order\_out = list(schema['transformed\_columns']) # get the output order

st.write(column\_order\_in)



* Now test the column\_order\_out to see as we saw with te column\_order\_in

**VI – Make the sidebar**

* Write the next lines of code that does the following
  + Extract over the schema the information of min and max values
  + Loop over the values and extract max, min and calculate the mean
  + Create the slidebars over the left side bar

#st.write(column\_order\_in)

st.sidebar.info('Customer Churn Features for Prediction')

options = {}

**for** col, col\_properties **in** schema['col\_info'].items():

**if** col == 'id':

**continue**

**if** col == 'churn':

**continue**

**if** col\_properties['dtype'] == 'int64' **or** col\_properties['dtype'] == 'float64':

min\_, max\_ = col\_properties['values']

dtype = col\_properties['dtype']

mean\_ = (min\_ + max\_)//**2**

# if dtype=='int64':

mean\_ = int(mean\_)

options[col] = st.sidebar.slider(col, int(min\_), int(max\_), value=mean\_)

st.write(options)

* Create the categorical features using the code below

# if dtype=='int64':

mean\_ = int(mean\_)

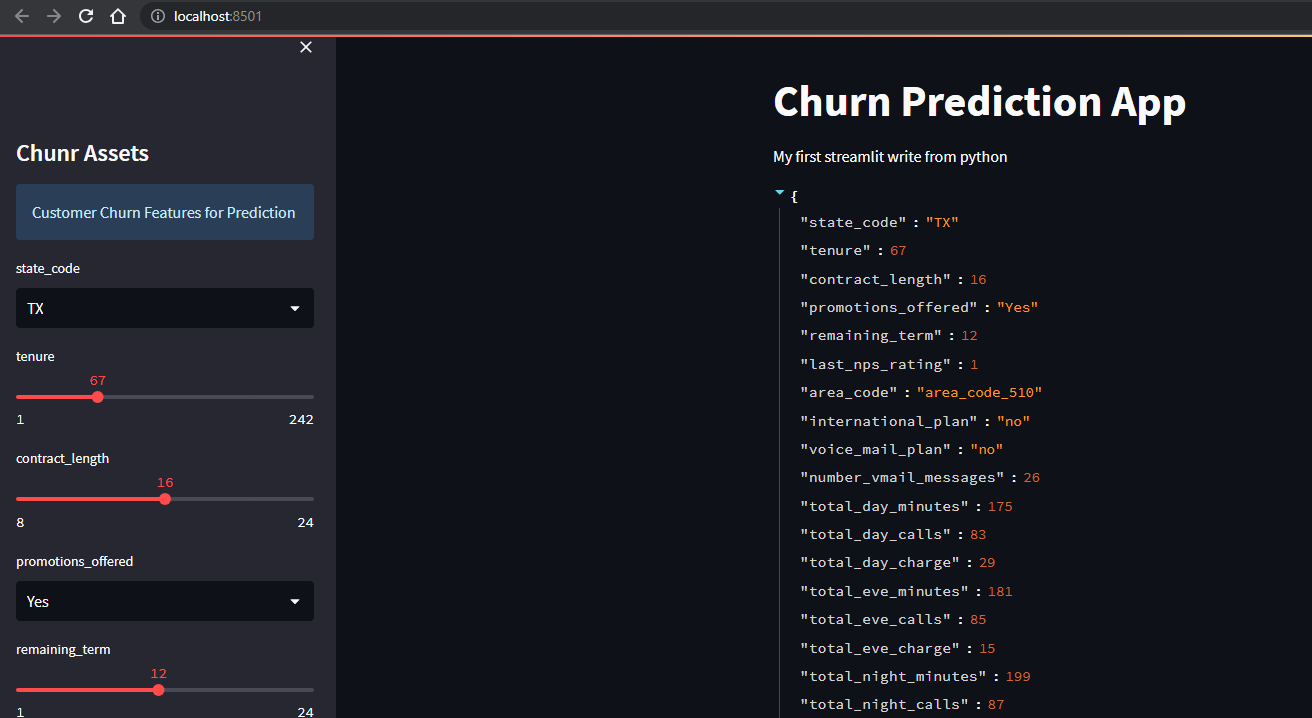
options[col] = st.sidebar.slider(col, int(min\_), int(max\_), value=mean\_)

**if** col\_properties['dtype']=='object':

options[col] = st.sidebar.selectbox(col, col\_properties['values'])

st.write(options)

Your dashboard will be taking this form as it shows the categorical and numerical data



**VII – Load the model and the encoder**

* Continue improving the dashboard now loading the model and the one hot encoder.

**if** col\_properties['dtype']=='object':

options[col] = st.sidebar.selectbox(col, col\_properties['values'])

# Load the model

**with** open(os.path.join('..', 'models', 'experiment\_1', 'xg.joblib'), 'rb') **as** f:

model = joblib.load(f)

# Load the encoder

**with** open(os.path.join('..', 'models', 'experiment\_1', 'ohe.joblib'), 'rb') **as** f:

ohe = joblib.load(f)

# mean eve minutes

mev = **200.29**

**VIII – Build the predictor**

* We will build a predict button to test the scoring, let’s do it.

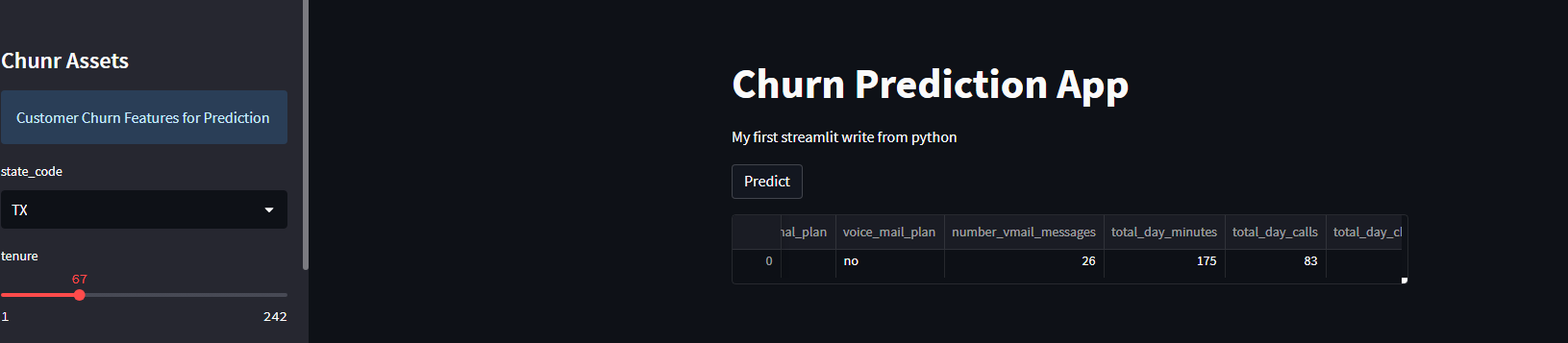
**if** st.button('Predict'):

data = pd.Series(options).to\_frame().T

data = data[column\_order\_in]

st.write(data)

* After reloading the app and press Predict we should see something like this.



* Now impor to make some final transformations.

**if** st.button('Predict'):

data = pd.Series(options).to\_frame().T

data = data[column\_order\_in]

# check data types

**for** col, col\_properties **in** schema['col\_info'].items():

**if** col=='id':

**continue**

**if** col=='churn':

**continue**

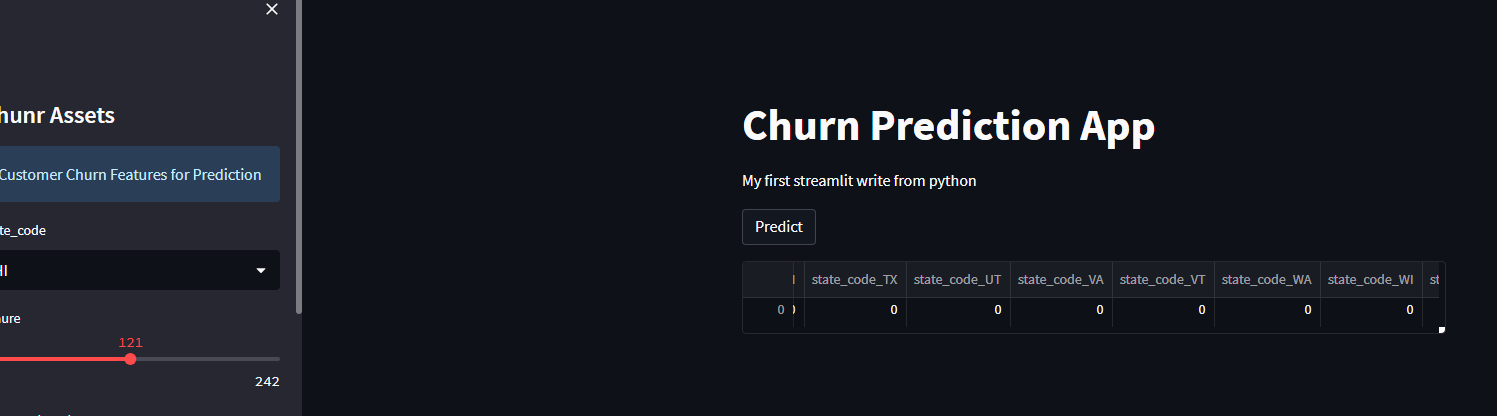
dtype = col\_properties['dtype']

data[col] = data[col].astype(dtype)

data = transform\_data(data, column\_order\_out, mev, ohe)

st.write(data)

* And updating the dashboard looks like this.



* Build the predictions using the code below

data = transform\_data(data, column\_order\_out, mev, ohe)

# Render Predictions

pred = model.predict(data)

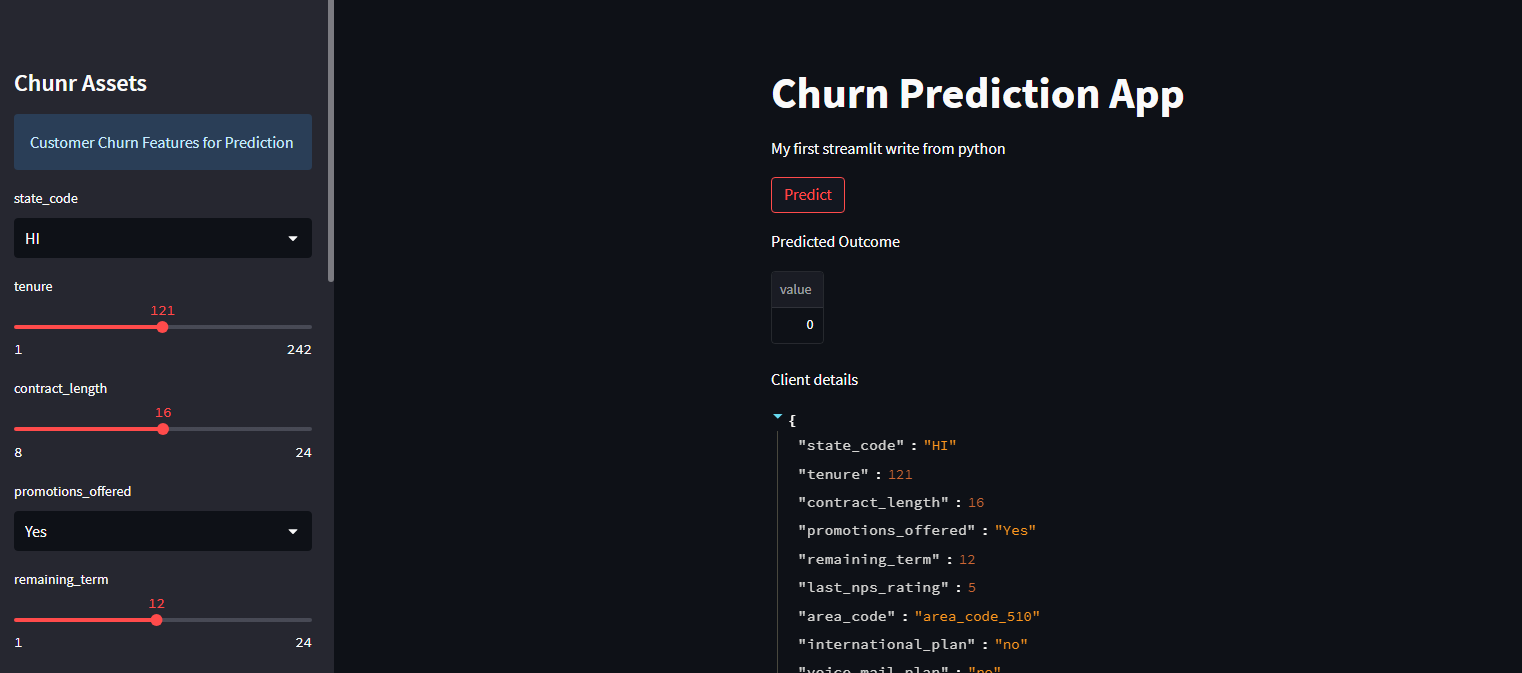
st.write('Predicted Outcome')

st.write(pred)

st.write('Cleint details')

st.write(options)

* Finally if you press predict we will see this



* Now we will add predictions to a csv to save historical data, this is very valuable to make the model measuremets of drifting if they will do.

# Save Historical data

**try**:

hist = pd.Series(options).to\_frame().T

hist['prediciton'] = pred

**if** os.path.isfile('historical.csv'):

hist.to\_csv('historical.csv', mode='a', header=False, index=False)

**else**:

hist.to\_csv('historical.csv', header=True, index=False)

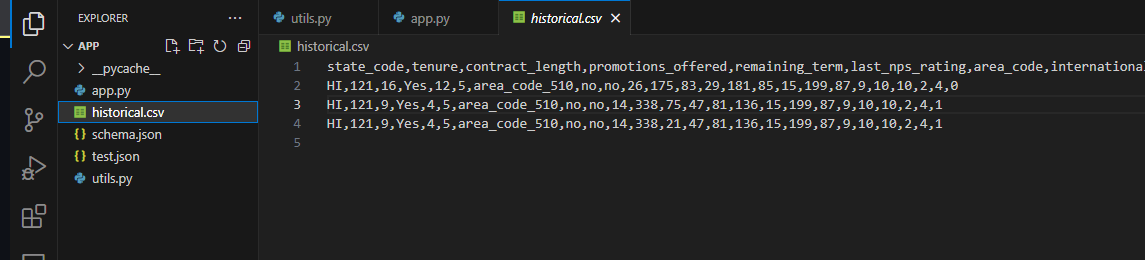
**except** **Exception** **as** e:

st.write(e)

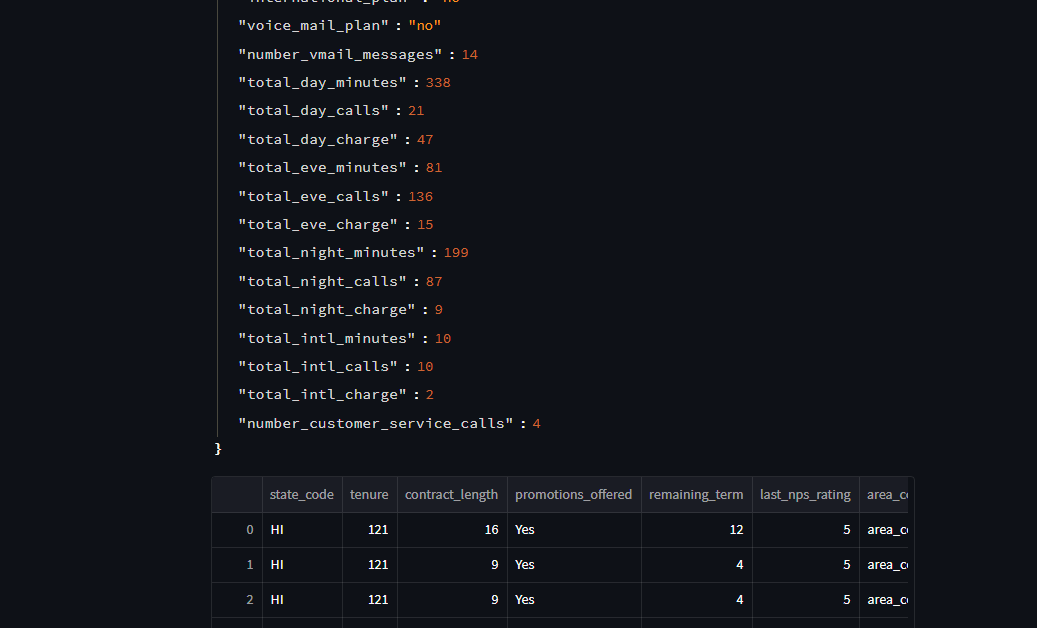
**pass**

st.write(pd.read\_csv('historical.csv'))

* And if you rerun again your window it will show as:



* Where our last value is if the client will churn or not



* Last but not least, let’s modify the app a little to display a beautiful graph of the churners predictions

# Save Historical data

**try**:

hist = pd.Series(options).to\_frame().T

hist['prediction'] = pred

**if** os.path.isfile('historical.csv'):

hist.to\_csv('historical.csv', mode='a', header=False, index=False)

**else**:

hist.to\_csv('historical.csv', header=True, index=False)

**except** **Exception** **as** e:

st.write(e)

**pass**

st.header('Historical Outcomes')

**if** os.path.isfile('historical.csv'):

hist = pd.read\_csv('historical.csv')

st.dataframe(hist)

fig, ax = plt.subplots()

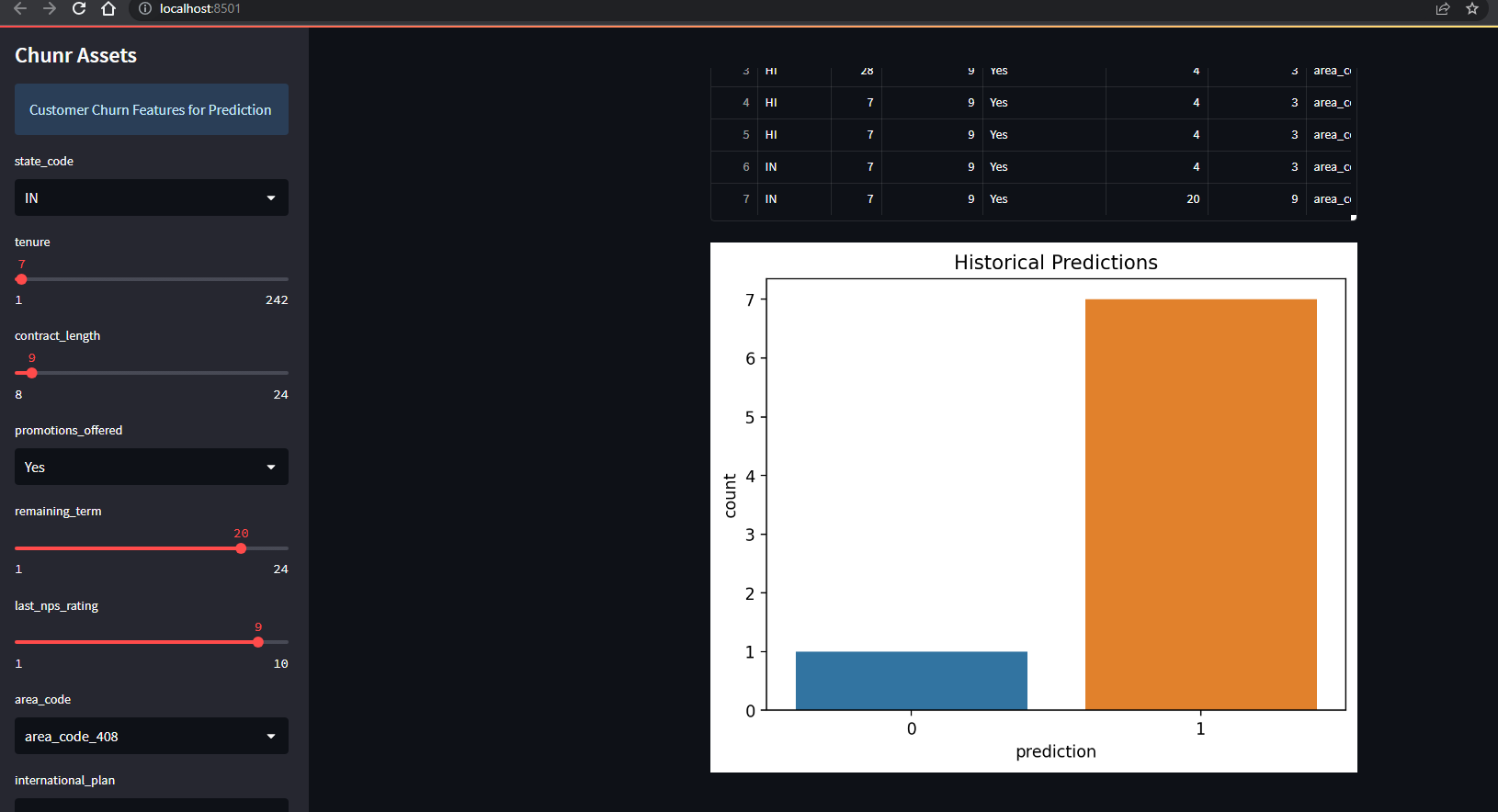
sns.countplot(x='prediction', data=hist, ax=ax).set\_title('Historical Predictions')

st.pyplot(fig)

**else**:

st.write('No historical data')

* Finally your dashboard will look like this



Today we have completed a full model creation and deployment, let’