```
import geopandas
import streamlit as st
import pandas    as pd
import numpy     as np
import folium

from datetime import datetime, time

from streamlit_folium import folium_static
from folium.plugins   import MarkerCluster

import plotly.express as px


# ----------------------------------------
# settings
# ----------------------------------------
st.set_page_config( layout='wide' )


# ----------------------------------------
# Helper Functions
# ----------------------------------------
@st.cache( allow_output_mutation=True )
def get_data( path ):
    data = pd.read_csv( path )

    return data


@st.cache( allow_output_mutation=True )
def get_geofile( url ):
    geofile = geopandas.read_file( url )

    return geofile


def set_attributes( data ):
    data['price_m2'] = data['price'] / data['sqft_lot']

    return data


def data_overview( data ):
    f_attributes = st.sidebar.multiselect( 'Enter columns',
data.columns )
    f_zipcode = st.sidebar.multiselect( 'Enter zipcode',
data['zipcode'].unique() )

    st.title( 'Data Overview' )

    if ( f_zipcode != [] ) & ( f_attributes != [] ):
        data = data.loc[data['zipcode'].isin( f_zipcode ),
f_attributes]
```

```
    elif ( f_zipcode != [] ) & ( f_attributes == [] ):
        data = data.loc[data['zipcode'].isin( f_zipcode ), :]

    elif ( f_zipcode == [] ) & ( f_attributes != [] ):
        data = data.loc[:, f_attributes]

    else:
        data = data.copy()

    st.write( data.head() )

    c1, c2 = st.beta_columns((1, 1) )

    # Average metrics
    df1 = data[['id',
'zipcode']].groupby( 'zipcode' ).count().reset_index()
    df2 = data[['price',
'zipcode']].groupby( 'zipcode').mean().reset_index()
    df3 = data[['sqft_living',
'zipcode']].groupby( 'zipcode').mean().reset_index()
    df4 = data[['price_m2',
'zipcode']].groupby( 'zipcode').mean().reset_index()


    # merge
    m1 = pd.merge( df1, df2, on='zipcode', how='inner' )
    m2 = pd.merge( m1, df3, on='zipcode', how='inner' )
    df = pd.merge( m2, df4, on='zipcode', how='inner' )

    df.columns = ['ZIPCODE', 'TOTAL HOUSES', 'PRICE', 'SQRT LIVING',
'PRICe/M2']

    c1.header( 'Average Values' )
    c1.dataframe( df, height=600 )

    # Statistic Descriptive
    num_attributes = data.select_dtypes( include=['int64',
'float64'] )
    media = pd.DataFrame( num_attributes.apply( np.mean ) )
    mediana = pd.DataFrame( num_attributes.apply( np.median ) )
    std = pd.DataFrame( num_attributes.apply( np.std ) )

    max_ = pd.DataFrame( num_attributes.apply( np.max ) )
    min_ = pd.DataFrame( num_attributes.apply( np.min ) )

    df1 = pd.concat([max_, min_, media, mediana, std],
axis=1 ).reset_index()
    df1.columns = ['attributes', 'max', 'min', 'mean', 'median',
'std']

    c2.header( 'Descriptive Analysis' )
    c2.dataframe( df1, height=800 )
```

```
    return None


def region_overview( data, geofile ):
    st.title( 'Region Overview' )

    c1, c2 = st.beta_columns( ( 1, 1 ) )
    c1.header( 'Portfolio Density' )

    df = data.sample( 10 )

    # Base Map - Folium
    density_map = folium.Map( location=[data['lat'].mean(),
data['long'].mean() ],
                              default_zoom_start=15 )

    marker_cluster = MarkerCluster().add_to( density_map )
    for name, row in df.iterrows():
        folium.Marker( [row['lat'], row['long'] ],
            popup='Sold R${0} on: {1}. Features: {2} sqft, {3}
bedrooms, {4} bathrooms, year built: {5}'.format( row['price'],
                          row['date'],
                          row['sqft_living'],
                          row['bedrooms'],
                          row['bathrooms'],

row['yr_built'] ) ).add_to( marker_cluster )


    with c1:
        folium_static( density_map )


    # Region Price Map
    c2.header( 'Price Density' )

    df = data[['price',
'zipcode']].groupby( 'zipcode' ).mean().reset_index()
    df.columns = ['ZIP', 'PRICE']

    geofile = geofile[geofile['ZIP'].isin( df['ZIP'].tolist() )]

    region_price_map = folium.Map( location=[data['lat'].mean(),
                                   data['long'].mean() ],
                                   default_zoom_start=15 )


    region_price_map.choropleth( data = df,
                                 geo_data = geofile,
                                 columns=['ZIP', 'PRICE'],
                                 key_on='feature.properties.ZIP',
                                 fill_color='YlOrRd',
                                 fill_opacity = 0.7,
                                 line_opacity = 0.2,
```

```
                                    legend_name='AVG PRICE' )

    with c2:
        folium_static( region_price_map )


    return None


def set_commercial( data ):
    st.sidebar.title( 'Commercial Options' )
    st.title( 'Commercial Attributes' )

    # ---------- Average Price per year built
    # setup filters
    min_year_built = int( data['yr_built'].min() )
    max_year_built = int( data['yr_built'].max() )

    st.sidebar.subheader( 'Select Max Year Built' )
    f_year_built = st.sidebar.slider( 'Year Built', min_year_built,
max_year_built, min_year_built )

    st.header( 'Average price per year built' )

    # get data
    data['date'] = pd.to_datetime( data['date'] ).dt.strftime( '%Y-
%m-%d' )

    df = data.loc[data['yr_built'] < f_year_built]
    df = df[['yr_built',
'price']].groupby( 'yr_built' ).mean().reset_index()

    fig = px.line( df, x='yr_built', y='price' )
    st.plotly_chart( fig, use_container_width=True )


    # ---------- Average Price per day
    st.header( 'Average Price per day' )
    st.sidebar.subheader( 'Select Max Date' )

    # setup filters
    min_date = datetime.strptime( data['date'].min(), '%Y-%m-%d' )
    max_date = datetime.strptime( data['date'].max(), '%Y-%m-%d' )

    f_date = st.sidebar.slider( 'Date', min_date, max_date,
min_date )

    # filter data
    data['date'] = pd.to_datetime( data['date'] )
    df = data[data['date'] < f_date]
    df = df[['date',
'price']].groupby( 'date' ).mean().reset_index()

    fig = px.line( df, x='date', y='price' )
```

```
    st.plotly_chart( fig, use_container_width=True )

    # ---------- Histogram -----------
    st.header( 'Price Distribuition' )
    st.sidebar.subheader( 'Select Max Price' )

    # filters
    min_price = int( data['price'].min() )
    max_price = int( data['price'].max() )
    avg_price = int( data['price'].mean() )

    f_price = st.sidebar.slider( 'Price', min_price, max_price,
avg_price )

    df = data[data['price'] < f_price]

    fig = px.histogram( df, x='price', nbins=50 )
    st.plotly_chart( fig, use_container_width=True )

    return None


def set_phisical( data ):
    st.sidebar.title( 'Attributes Options' )
    st.title( 'House Attributes' )

    # filters
    f_bedrooms = st.sidebar.selectbox( 'Max number of bedrooms',

sorted( set( data['bedrooms'].unique() ) ) )
    f_bathrooms = st.sidebar.selectbox( 'Max number of bath',

sorted( set( data['bathrooms'].unique() ) ) )

    c1, c2 = st.beta_columns( 2 )

    # Houses per bedrooms
    c1.header( 'Houses per bedrooms' )
    df = data[data['bedrooms'] < f_bedrooms]
    fig = px.histogram( df, x='bedrooms', nbins=19 )
    c1.plotly_chart( fig, use_containder_width=True )

    # Houses per bathrooms
    c2.header( 'Houses per bathrooms' )
    df = data[data['bathrooms'] < f_bathrooms]
    fig = px.histogram( df, x='bathrooms', nbins=10 )
    c2.plotly_chart( fig, use_containder_width=True )

    # filters
    f_floors = st.sidebar.selectbox('Max number of floors',
sorted( set( data['floors'].unique() ) ) )
    f_waterview = st.sidebar.checkbox('Only House with Water View' )

    c1, c2 = st.beta_columns( 2 )
```

```
    # Houses per floors
    c1.header( 'Houses per floors' )
    df = data[data['floors'] < f_floors]
    fig = px.histogram( df, x='floors', nbins=19 )
    c1.plotly_chart( fig, use_containder_width=True )

    # Houses per water view
    if f_waterview:
        df = data[data['waterfront'] == 1]
    else:
        df = data.copy()

    fig = px.histogram( df, x='waterfront', nbins=10 )
    c2.header( 'Houses per water view' )
    c2.plotly_chart( fig, use_containder_width=True )

    return None


if __name__ == "__main__":
    # ETL
    path = '../kc_house_data.csv'
    url='https://opendata.arcgis.com/datasets/
83fc2e72903343aabff6de8cb445b81c_2.geojson'

    # load data
    data = get_data( path )
    geofile = get_geofile( url )

    # transform data
    data = set_attributes( data )

    data_overview( data )

    region_overview( data, geofile )

    set_commercial( data )

    set_phisical( data )
```