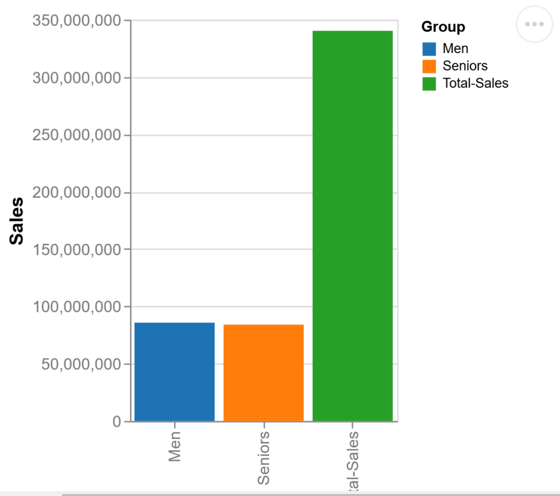
```python
# Sales Analysis (Assignment-2)
# Soumen-Dey: Dec-2024
# Lic: Freeware
# -------------------------------

import pandas as pd
from io import StringIO

import json
from io import FileIO
from json import loads, dumps

from datetime import datetime
from pydrive2.auth import GoogleAuth
from pydrive2.drive import GoogleDrive

from google.colab import drive
from google.colab import auth
from oauth2client.client import GoogleCredentials

# import csv
from os import system, name
from decimal import Decimal, InvalidOperation
from time import sleep
from tabulate import tabulate
import matplotlib.pyplot as plt
import seaborn as sns
import altair as alt

import calendar

# ---------------- CODE ----------------------------------------

# Authenticate and create the PyDrive client.
# This part is crucial for accessing files in your Google Drive
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

fName= 'AusApparalSales4thQrt2020.csv'
folderId = '1gOAnUVtJn-xuFkiRr_rvExSKsqdndPyy'

pd.options.mode.copy_on_write = True

grpSorted = pd.DataFrame()
grpTable = ''

def read_file( id_file ) :
      # file_list = drive.ListFile({'q': "title='MyAiFile.txt' and trashed=false"}).GetList()
      title = "title='" + id_file + "' and trashed=false"
      print ('File reading: ', title)

      # #Get the file from the list as per API Spec
      file_list = drive.ListFile({'q': title}).GetList()

      if len(file_list) > 0:
          file = file_list[0]
          file_content = file.GetContentString()

          return file_content
      else:
        print("File : ", id_file , " not found")

# print table
def display_tabular(content):
  global grpTable

  jsonData = json.loads(content)
  grpTable = tabulate(jsonData, headers="keys", tablefmt="grid")

  return grpTable



# Group the data
def group_by_sales(df_local):
      global grpSorted

      n_df = df_local[['Group', 'Sales']]
```

```
        df_grp = n_df.groupby(['Group'],sort=False).sum()

        # Sort the group by Descending
        grpSorted = df_grp.sort_values('Sales', ascending = False)
        grpSorted = grpSorted.reset_index() #set index if needed

        maxSalesRow = (grpSorted.iloc[0])
        minSalesRow = (grpSorted.iloc[len(grpSorted)-1])

        json_data_minmax = []

        json_data_minmax.append(
          {
            "Group":  maxSalesRow['Group'],
            "Sales": str(maxSalesRow['Sales']),
            "Type" : "Highest"
          }
        )
        json_data_minmax.append(
          {
            "Group":  minSalesRow['Group'],
            "Sales": str(minSalesRow['Sales']),
            "Type" : "Lowest"
          }
        )
        json_data_minmax.append(
          {
            "Group": "Total-Sales",
            "Sales": str(df_grp.sum().get(key = 'Sales'))
          }
        )

        return json_data_minmax


  # Group the data by State
  def group_by_sales_state(df_local):
        global grpSorted

        n_df = df_local[['State', 'Sales']]

        df_grp = n_df.groupby(['State'],sort=False).sum()

        # Sort the group by Descending
        grpSorted = df_grp.sort_values('Sales', ascending = False)
        grpSorted = grpSorted.reset_index() #set index if needed

        maxSalesRow = (grpSorted.iloc[0])
        minSalesRow = (grpSorted.iloc[len(grpSorted)-1])

        json_data_minmax = []

        json_data_minmax.append(
          {
            "State":  maxSalesRow['State'],
            "Sales": str(maxSalesRow['Sales']),
            "Type" : "Highest"
          }
        )
        json_data_minmax.append(
          {
            "State":  minSalesRow['State'],
            "Sales": str(minSalesRow['Sales']),
            "Type" : "Lowest"
          }
        )
        json_data_minmax.append(
          {
            "State": "Total-Sales",
            "Sales": str(df_grp.sum().get(key = 'Sales'))
          }
        )

        return json_data_minmax


  # Group the data by State -> Group
  def group_by_sales_state_demographic(df_local):
        global grpSorted

        n_df = df_local[['State','Group','Sales']]

        df_grp = n_df.groupby(['State','Group'],sort=False).sum()
```

```
      # Sort the group by Descending
      # grpSorted = df_grp.sort_values('State','Group','Sales', ascending = False)
      # grpSorted = grpSorted.reset_index() #set index if needed

      print (df_grp)

      # maxSalesRow = (grpSorted.iloc[0])
      # minSalesRow = (grpSorted.iloc[len(grpSorted)-1])

      # json_data_minmax = []

      # json_data_minmax.append(
      #   {
      #     "State":  maxSalesRow['State'],
      #     "Sales": str(maxSalesRow['Sales']),
      #     "Type" : "Highest"
      #   }
      # )
      # json_data_minmax.append(
      #   {
      #     "State":  minSalesRow['State'],
      #     "Sales": str(minSalesRow['Sales']),
      #     "Type" : "Lowest"
      #   }
      # )
      # json_data_minmax.append(
      #   {
      #     "State": "Total-Sales",
      #     "Sales": str(df_grp.sum().get(key = 'Sales'))
      #   }
      # )


      # return json_data_minmax


# Group the data by Month
def group_by_month(df_local):
      global grpSorted

      df_date = df[['Date', 'Sales']]
      df_date['Month'] = pd.to_datetime(df_date['Date']).dt.month
      df_grp = df_date.groupby(['Month']).sum()

      # Sort the group by Descending
      grpSorted = df_grp.sort_values('Sales', ascending = False)
      grpSorted = grpSorted.reset_index() #set index if needed

      # convert month index to str month
      grpSorted['Month'] = grpSorted['Month'].apply(lambda x: calendar.month_abbr[x])

      maxSalesRow = (grpSorted.iloc[0])
      minSalesRow = (grpSorted.iloc[len(grpSorted)-1])

      json_data_minmax = []

      json_data_minmax.append(
        {
          "Month":  str(maxSalesRow['Month']),
          "Sales": str(maxSalesRow['Sales']),
          "Type" : "Highest"
        }
      )
      json_data_minmax.append(
        {
          "Month":  str(minSalesRow['Month']),
          "Sales": str(minSalesRow['Sales']),
          "Type" : "Lowest"
        }
      )
      json_data_minmax.append(
        {
          "Month": "Total-Sales",
          "Sales": str(df_grp.sum().get(key = 'Sales'))
        }
      )

      return json_data_minmax

# Group the data by Week
def group_by_week(df_local):
      global grpSorted
      global grpSorted

      n_df = df_local[['Date', 'Sales']]
      n_df['Week'] = pd.to_datetime(n_df['Date']).dt.isocalendar().week

      df_grp = n_df.groupby(['Week'],sort=False).sum()

      # Sort the group by Descending
      grpSorted = df_grp.sort_values('Sales', ascending = True)
      grpSorted = grpSorted.reset_index() #set index if needed

      maxSalesRow = (grpSorted.iloc[0])
      minSalesRow = (grpSorted.iloc[len(grpSorted)-1])

      json_data_minmax = []

      json_data_minmax.append(
        {
          "Week":  str(maxSalesRow['Week']),
          "Sales": str(maxSalesRow['Sales']),
          "Type" : "Highest"
        }
      )
      json_data_minmax.append(
        {
          "Week":  str(minSalesRow['Week']),
          "Sales": str(minSalesRow['Sales']),
          "Type" : "Lowest"
        }
      )
      json_data_minmax.append(
        {
          "Week": "Total-Sales",
          "Sales": str(df_grp.sum().get(key = 'Sales'))
        }
      )

      return json_data_minmax

def group_by_PickTimeInADay(df_local):
      n_df = df_local[['Date', 'Time', 'Sales']]
      df_grp = n_df.groupby(['Date','Time'],sort=False).sum()
      ng = df_grp.groupby(['Date'], sort=False)['Sales'].nlargest(1)
      df = pd.DataFrame(ng)

      return df


def dfToJson(df_local):
      result = df_local.to_json(orient="split")
      parsed = loads(result)
      strJson = dumps(parsed)

      return strJson

# --- altair chart theme
def my_custom_theme():
    return {
        "config": {
            "view": {"continuousWidth": 400, "continuousHeight": 300},
            "mark": {"color": "steelblue"},
            "axis": {
                "labelFontSize": 12,
                "titleFontSize": 14,
                "labelColor": "gray",
                "titleColor": "black"
            },
            "range": {
                "category": ["#1f77b4", "#ff7f0e", "#2ca02c", "#d62728", "#9467bd"]
            }
        }
    }

# Register and enable the custom theme
alt.themes.register('my_custom_theme', my_custom_theme)
alt.themes.enable('my_custom_theme')


# ---------------------------------
records = read_file(fName)
# --------------------------------------------------------

df = pd.read_csv(StringIO(records))
```

```
    #Process the data
    # Data Wrangling Actions

    #identify group with hightest sale (Group + Sales)
    result_byGrp_list = group_by_sales(df)
    json_str = json.dumps(result_byGrp_list)
    grpTable = display_tabular(json_str)

    #B1
    print ("\n\n---------------------[Section - 1: GROUP]--------------------------")
    print ('Analysis :')
    print ('======================================================================')

    print("Group wise sale",  end="                      |           ")
    print("Plots:")
    print(grpSorted,                    end="                |           ")
    print("\n\nGroup -(Highest & Lowest):\n",grpTable, end="   -  ")

    print ("\n")
    df = pd.DataFrame(result_byGrp_list)


    colors = ['#170c3b', '#fa8d0b']
    # Convert 'Sales' column to numeric type
    df['Sales'] = pd.to_numeric(df['Sales'])

    # Create the bar chart
    alt.Chart(df).mark_bar().encode(
        x='Group',
        y='Sales',
        color='Group'
    ).properties(
         width=200,
         height=300
    )
    #--------------------------------[End Section -1]-------------------------------
```

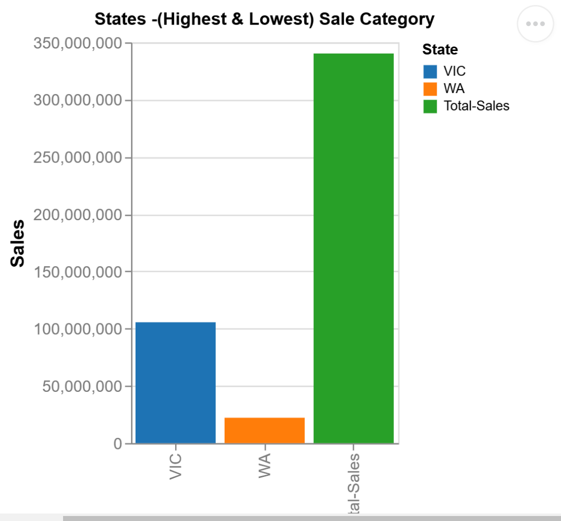
```
File reading:  title='AusApparalSales4thQrt2020.csv' and trashed=false


---------------------[Section - 1: GROUP]--------------------------
Analysis :
=======================================================================
Group wise sale                      |           Plots:
      Group     Sales
0       Men  85750000
1     Women  85442500
2      Kids  85072500
3   Seniors  84037500                |

Group -(Highest & Lowest):
 +-------------+-----------+---------+
| Group       |     Sales | Type    |
+=============+===========+=========+
| Men         |  85750000 | Highest |
+-------------+-----------+---------+
| Seniors     |  84037500 | Lowest  |
+-------------+-----------+---------+
| Total-Sales | 340302500 |         |
+-------------+-----------+---------+    -
```



```
print ("\n\n---------------------[Section - 2: STATE]--------------------------")

print("\nStates -(Highest & Lowest):")
df = pd.read_csv(StringIO(records))
result = group_by_sales_state(df)

print("\nState wise sales",  end="                   |           ")
print("Plots:")
print(grpSorted,                      end="                   |           ")

print("\n\nState - Demographic (High - Low) >>")
group_by_sales_state_demographic(df)

json_str = json.dumps(result)
grpTable1 = display_tabular(json_str)
print(grpTable1)


# ---- Create Chart --------
df = pd.DataFrame(result)
df = df[['State', 'Sales']]

# Convert 'Sales' column to numeric type
df['Sales'] = pd.to_numeric(df['Sales'])

# Create the bar chart
alt.Chart(df).mark_bar().encode(
    x='State',
    y='Sales',
    color='State'
).properties(
     title = "States -(Highest & Lowest) Sale Category",
     width=200,
     height=300
)
```

```
# dfgrp = pd.DataFrame(grpSorted)
# alt.Chart(dfgrp).mark_bar().encode(
#     x='State',
#     y='Sales'
# )

#--------------------------------[End Section -2]--------------------------------
```

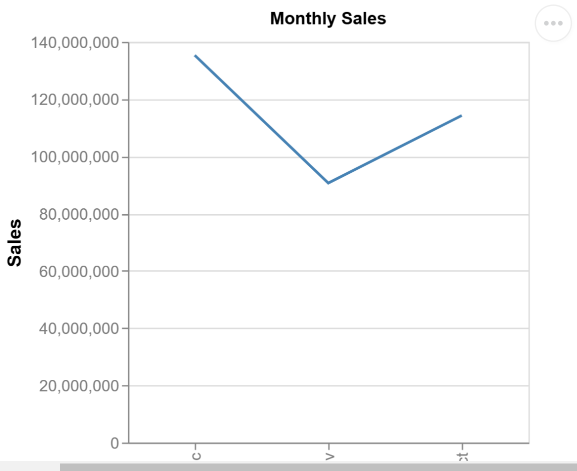
```
---------------------[Section - 2: STATE]---------------------------

States -(Highest & Lowest):

State wise sales                   |           Plots:
  State      Sales
0   VIC  105565000
1   NSW   74970000
2    SA   58857500
3   QLD   33417500
4   TAS   22760000
5    NT   22580000
6    WA   22152500                 |

State - Demographic (High - Low) >>
                  Sales
State Group
WA    Kids      5625000
      Men       5752500
      Women     5262500
      Seniors   5512500
NT    Kids      5700000
      Men       5762500
      Women     5652500
      Seniors   5465000
SA    Kids     14515000
      Men      14655000
      Women    14970000
      Seniors  14717500
VIC   Kids     26360000
      Men      26407500
      Women    26482500
      Seniors  26315000
QLD   Kids      8510000
      Men       8392500
      Women     8325000
      Seniors   8190000
NSW   Kids     18587500
      Men      19022500
      Women    19172500
      Seniors  18187500
TAS   Kids      5775000
      Men       5757500
      Women     5577500
      Seniors   5650000
+-------------+-----------+---------+
| State       |     Sales | Type    |
+=============+===========+=========+
| VIC         | 105565000 | Highest |
+-------------+-----------+---------+
| WA          |  22152500 | Lowest  |
+-------------+-----------+---------+
| Total-Sales | 340302500 |         |
+-------------+-----------+---------+
```



```
print ("\n\n---------------------[Section - 3: MONTH]---------------------------------")
print ("\nMonthly sales:")

df = pd.read_csv(StringIO(records))
result_byGrp_month = group_by_month(df)
json_str = json.dumps(result_byGrp_month)
```

```
grpSorted = grpSorted.sort_values(by=['Sales'], ascending=False)
grpSorted = pd.DataFrame(grpSorted, columns = ['Month', 'Sales'])
print (grpSorted)

tableStr = display_tabular(json_str)
print("\nMonth -(Highest & Lowest):\n",tableStr, end="   -  ")
print ("\n")

alt.Chart(grpSorted).mark_line().encode(
    x='Month',
    y='Sales'
).properties(
     title = "Monthly Sales",
     width=300,
     height=300
)
#--------------------------------[End Section -3 --------------------------------
```

```
---------------------[Section - 3: MONTH]----------------------------------

Monthly sales:
  Month      Sales
0   Dec  135330000
1   Oct  114290000
2   Nov   90682500

Month -(Highest & Lowest):
 +-------------+-----------+---------+
| Month       |     Sales | Type    |
+=============+===========+=========+
| Dec         | 135330000 | Highest |
+-------------+-----------+---------+
| Nov         |  90682500 | Lowest  |
+-------------+-----------+---------+
| Total-Sales | 340302500 |         |
+-------------+-----------+---------+   -
```



```
print ("\n\n---------------------[Section - 4: WEEK]----------------------------")
print ("\nWeekly sales:")

df = pd.read_csv(StringIO(records))
result_byGrp_week = group_by_week(df)

grpSorted = grpSorted.sort_values(by=['Week'], ascending=True)
grpSorted = grpSorted.reset_index() #set index if needed
grpSorted = pd.DataFrame(grpSorted, columns = ['Week', 'Sales'])
print (grpSorted)

json_str = json.dumps(result_byGrp_week)
tableStr = display_tabular(json_str)
print("\nWeek -(Highest & Lowest):\n",tableStr, end="   -   ")
print ("\n")

import altair as alt
alt.Chart(grpSorted).mark_line().encode(
    x='Week',
        y='Sales'
    ).properties(
        title = "Weekly Sales",
        width=300,
        height=300
    )
    #--------------------------------[End Section -4] --------------------------------
```

```
---------------------[Section - 4: WEEK]-----------------------------

Weekly sales:
    Week     Sales
0     40  15045000
1     41  27002500
2     42  26640000
3     43  26815000
4     44  21807500
5     45  20865000
6     46  21172500
7     47  21112500
8     48  21477500
9     49  29622500
10    50  31525000
11    51  31655000
12    52  31770000
13    53  13792500

Week -(Highest & Lowest):
 +-------------+-----------+---------+
| Week        |     Sales | Type    |
+=============+===========+=========+
| 53          |  13792500 | Highest |
```