The code of this visualization project has been written in Pycharm (Python) using the Dash Library. To make use of the visualization tool the attached code need to be copied in Pycharm (Python). In line 10 of the code the attached data file needs to be loaded in.

After the code can be run by pressing the run button in the top right corner. After running the code, a link will pop up, and you open that link. A dash visualization is opened and you can make use of the visualization tool. Have Fun!!

The libraries used are plotly, pandas, numpy and dash. Plotly is used for rendering the charts, pandas is used to load the data, numpy is used to alter and compute certain data necessary for well visualization. Everything is encoded and implemented by ourselves.

**The best way to run the code is cloning our repository:**

[**https://github.com/E-Nikq/jbi100**](https://github.com/E-Nikq/jbi100)

**Then you should run the python file named “tester\_vis.py”, note that it may take some time to load.**

Code:

import plotly.graph\_objects as go  
from dash import Dash, dcc, html, Input, Output, State, dash\_table, callback\_context  
import pandas as pd  
import plotly.express as px  
import plotly.figure\_factory as ff  
import numpy as np  
from dash.dependencies import Input, Output  
from dash.exceptions import PreventUpdate  
  
df = pd.read\_excel('used\_data.xlsx')  
  
app = Dash(\_\_name\_\_)  
  
def most\_frequent\_credit\_score(x):  
 if not x.empty:  
 value\_counts = x.value\_counts()  
 if not value\_counts.empty:  
 return value\_counts.idxmax()  
 return None  
  
avg\_debt\_per\_occupation\_age = df.groupby(['Occupation', 'Credit\_History\_Age']).agg({  
 'Outstanding\_Debt': 'mean',  
 'Age': 'mean',  
 'Annual\_Income\_Code2': 'mean',  
 'Credit\_Utilization\_Ratio': 'mean',  
 'Credit\_Score': most\_frequent\_credit\_score  
}).reset\_index()  
  
avg\_debt\_per\_occupation\_age = avg\_debt\_per\_occupation\_age.round({'Outstanding\_Debt': 0,  
 'Age': 0,  
 'Annual\_Income\_Code2': 0,  
 'Credit\_Utilization\_Ratio': 0,  
 'Credit\_Score': 0})  
  
avg\_income\_per\_occupation = df.groupby('Occupation')['Annual\_Income\_Code2'].mean().reset\_index()  
avg\_debt\_per\_occupation = df.groupby('Occupation')['Outstanding\_Debt'].mean().reset\_index()  
avg\_income\_per\_occupation.columns = ['Occupation', 'Avg\_Annual\_Income']  
avg\_debt\_per\_occupation.columns = ['Occupation', 'Avg\_Outstanding\_Debt']  
  
min\_age = int(df['Credit\_History\_Age'].min())  
max\_age = min(33, int(df['Credit\_History\_Age'].max()))  
  
colors = {  
 'background': '#111111',  
 'text': '#7FDBFF'  
}  
  
# Step 2: Calculate Credit Score Probability  
credit\_score\_probabilities = df.groupby('Occupation')['Credit\_Score'].apply(lambda x: (x == 'Poor').mean()).reset\_index()  
credit\_score\_probabilities.columns = ['Occupation', 'Credit\_Score\_Probability']  
  
# Step 3: Add the Columns to DataFrame  
df = pd.merge(df, avg\_income\_per\_occupation, on='Occupation', how='left')  
df = pd.merge(df, avg\_debt\_per\_occupation, on='Occupation', how='left')  
df = pd.merge(df, credit\_score\_probabilities, on='Occupation', how='left')  
  
scatter\_fig = px.scatter(  
 df,  
 x='Avg\_Annual\_Income',  
 y='Avg\_Outstanding\_Debt',  
 size='Credit\_Score\_Probability',  
 color='Occupation',  
 labels={'Avg\_Annual\_Income': 'Average Annual Income in $', 'Avg\_Outstanding\_Debt': 'Average Outstanding Debt in $'},  
 title='Scatter Plot with Credit Score Probability',  
 hover\_data={'Credit\_Score\_Probability': ':.2%'},  
 size\_max=250  
)  
  
scatter\_fig.update\_layout(  
 plot\_bgcolor=colors['background'],  
 paper\_bgcolor=colors['background'],  
 font\_color=colors['text'],  
 title='Scatter Plot with Credit Score Probability',  
 annotations=[  
 {  
 'x': 0.5,  
 'y': 1.07,  
 'xref': 'paper',  
 'yref': 'paper',  
 'text': 'Bubble size is proportional to the probability of the credit score being \'Poor\'',  
 'showarrow': False,  
 'font': {'size': 14, 'color': colors['text']}  
 }  
 ],  
 height=800  
)  
  
Categorical\_names = ['Credit\_Mix', 'Occupation', 'Payment\_of\_Min\_Amount', 'Month', 'Credit\_Score',  
 'Payment\_Behaviour']  
Numeric\_names = ["Age", "Amount\_invested\_monthly", "Changed\_Credit\_Limit", "Interest\_Rate",  
 "Monthly\_Inhand\_Salary",  
 "Num\_Bank\_Accounts", "Num\_Credit\_Card", "Num\_Credit\_Inquiries", "Num\_of\_Delayed\_Payment",  
 "Num\_of\_Loan",  
 "Outstanding\_Debt", "Delay\_from\_due\_date", "Credit\_Utilization\_Ratio", "Credit\_History\_Age",  
 "Total\_EMI\_per\_month", "Monthly\_Balance", "Annual\_Income"]  
Aggregate\_functions = ["mean","count","min","max","median","std"]  
  
df\_Num = df[Numeric\_names]  
corr = df\_Num.corr()  
corr = np.round(corr,2)  
  
fig = px.imshow(corr, labels=dict(x="Numeric Attributes", y="Numeric Attributes", color='Correlation'),  
 x=Numeric\_names,  
 y=Numeric\_names,  
 text\_auto=True,  
 color\_continuous\_scale='reds',  
 color\_continuous\_midpoint=0)  
fig.update\_layout(  
 title="Correlation Heat Map",  
 width=1200, height=800,  
 transition\_duration=500,  
 plot\_bgcolor=colors['background'],  
 paper\_bgcolor=colors['background'],  
 font\_color=colors['text']  
)  
fig.update\_xaxes(tickangle=30, tickfont=dict(size=15))  
fig.update\_yaxes(tickfont=dict(size=15))  
fig.update\_traces(textfont=dict(size=12),  
 hovertemplate='%{x}<br>%{y}<br>Correlation: %{z:.2f}')  
  
app.layout = html.Div(style={'backgroundColor': colors['background'], 'height': '100vh'}, children=[  
 html.Div(style={'backgroundColor': colors['background']}, children=[  
 dcc.Graph(id='Correlation Heat Map',  
 style={'width': '100%', 'Align': 'center'},  
 figure=fig)  
 ]),  
  
 html.Div(style={'flexgrow':1,'backgroundColor': colors['background']}, children=[  
 html.H1(  
 children='Radar Chart',  
 style={'textAlign': 'left',  
 'color': colors['text'],  
 'margin-left': '40%'}  
 ),  
  
 html.Div(style={  
 'textAlign': 'center',  
 'color': colors['text']  
 }),  
  
 dcc.Graph(  
 id='Radar Chart',  
 style={'width': '100%', 'Align': 'center'},  
 clickData={'points': [{'theta': 'Num\_of\_Loan'}]},  
 ),  
  
 html.Div(style={'position': 'absolute', 'top': '850px', 'left': '10px', 'backgroundColor': colors['background'],'width': '10%'},  
 children=[  
 html.Label("Select the attribute:", style={'color': colors["text"]}),  
 dcc.Dropdown(options=[{'label': name, 'value': name} for name in Categorical\_names],  
 value='Occupation',  
 id='DropdownAttribute'  
 ),  
 html.Br(),  
 html.Label("Select additional attributes:", style={'color': colors["text"]}),  
 dcc.Dropdown(options=[{'label': name, 'value': name} for name in Numeric\_names],  
 value=['Num\_of\_Loan', 'Num\_Bank\_Accounts'],  
 id='Multi',  
 multi=True)  
 ])  
 ]),  
  
 html.Div(style={'margin-top': '0px', 'backgroundColor': colors["background"], 'padding-left': '10px','width': '100%'},  
 children=[  
 html.Div(style={'width': '80%', 'color': colors['text'], 'backgroundColor':colors["background"]},children=[  
 dcc.RadioItems(id='distribution',  
 options=[{'label': 'Hist', 'value': 'Hist'}, {'label': 'None', 'value': 'None'}],  
 value='None', inline=True,  
 style={'color': colors["text"], 'width': '50%', 'display': 'inline-block','float': 'center'}  
 ),  
 dcc.RadioItems(id='X\_axis\_type',  
 options=[{'label': 'Linear', 'value': 'linear'}, {'label': 'Log', 'value': 'log'}],  
 value='linear',  
 style={'color': colors["text"], 'width': '50%',  
 'display': 'inline-block',  
 'float': 'right'},  
 inline=True,  
 )]),  
 dcc.Graph(id="Add\_chart")  
 ]),  
 html.Div(style={'backgroundColor': 'black', 'color': 'white', "margin-top": "0px"}, children=[  
 html.H1(  
 children='Bar Chart',  
 style={'textAlign': 'center',  
 'color': colors['text']}  
 ),  
 dcc.Location(id='url', refresh=False),  
 html.Div([  
 html.Label("Select Age of Credit History:", style={'color': '#FF6347'}),  
 dcc.Slider(  
 id='credit-age-slider',  
 min=min\_age,  
 max=max\_age,  
 step=1,  
 marks={i: str(i) for i in range(min\_age, max\_age + 1)},  
 value=min\_age,  
 ),  
 html.Label("Select Occupation:", style={'color': '#FF6347', 'margin-top': '20px'}),  
 dcc.Dropdown(  
 id='occupation-dropdown',  
 options=[{'label': 'All', 'value': 'all'}] + [{'label': occ, 'value': occ} for occ in  
 df['Occupation'].unique()],  
 value=['all'],  
 multi=True,  
 style={'color': 'black', 'width': '150px', 'backgroundColor': '#144E78'}  
 # Set background color to a navy blue shade  
 ),  
 ]),  
 dcc.Graph(id='credit-score-graph'),  
 html.Div(id='page-content', style={'margin': 'auto', 'width': '50%'}),  
 ]),  
 html.Div(style={'flexgrow':1,'backgroundColor': colors['background']}, children=[  
  
 html.H1(  
 children='Scatter Plot',  
 style={'textAlign': 'center',  
 'color': colors['text'] }  
 ),  
 html.Button(  
 id='reset-scatter-button',  
 children='Reset Scatter Plot',  
 n\_clicks=0,  
 style={'margin-top': '5px'}  
 ),  
 dcc.Graph(  
 id='Scatter Plot',  
 style={'width': '100%', 'Align': 'center'},  
 figure=scatter\_fig  
 )  
  
 ])  
  
])  
  
# Existing callback for bar chart  
@app.callback(  
 Output('credit-score-graph', 'figure'),  
 [Input('credit-age-slider', 'value'),  
 Input('occupation-dropdown', 'value')]  
)  
def update\_graph(selected\_age, selected\_occupation):  
 if 'all' in selected\_occupation:  
 filtered\_df = avg\_debt\_per\_occupation\_age[avg\_debt\_per\_occupation\_age['Credit\_History\_Age'] == selected\_age]  
 else:  
 filtered\_df = avg\_debt\_per\_occupation\_age[  
 (avg\_debt\_per\_occupation\_age['Credit\_History\_Age'] == selected\_age) &  
 (avg\_debt\_per\_occupation\_age['Occupation'].isin(selected\_occupation))  
 ]  
  
 figure = {  
 'data': [  
 {  
 'x': filtered\_df['Occupation'],  
 'y': filtered\_df['Outstanding\_Debt'],  
 'type': 'bar',  
 'name': 'Average Debt per Occupation',  
 'marker': {'color': filtered\_df['Annual\_Income\_Code2'], 'colorbar': {'title': 'Average Annual Income'}},  
 },  
 ],  
 'layout': {  
 'title': f'Average Outstanding Debt by Occupation (Age of Credit History: {selected\_age} Years)',  
 'xaxis': {'title': 'Occupation', 'categoryorder': 'total ascending'},  
 'yaxis': {'title': 'Average Outstanding Debt'},  
 'clickmode': 'event+select',  
 'plot\_bgcolor': 'black',  
 'paper\_bgcolor': 'black',  
 'legend': {'title': {'text': 'Average Annual Income'}, 'bgcolor': 'black', 'bordercolor': 'white', 'borderwidth': 1, 'font': {'color': 'white'}},  
 }  
 }  
  
 return figure  
  
# Existing callback for selected data table  
@app.callback(  
 Output('page-content', 'children'),  
 [Input('credit-score-graph', 'clickData'),  
 Input('credit-age-slider', 'value')]  
)  
def display\_selected\_data\_new(chart\_click\_data, selected\_age):  
 if chart\_click\_data:  
 selected\_occupation\_index = chart\_click\_data['points'][0]['pointIndex']  
 selected\_occupation = chart\_click\_data['points'][0]['x']  
 selected\_data = avg\_debt\_per\_occupation\_age[  
 (avg\_debt\_per\_occupation\_age['Occupation'] == selected\_occupation) &  
 (avg\_debt\_per\_occupation\_age['Credit\_History\_Age'] == selected\_age)  
 ]  
  
 if not selected\_data.empty:  
 selected\_data = selected\_data.iloc[0]  
  
 table\_data = [  
 {'Metric': 'Average Outstanding Debt in $', 'Value': selected\_data['Outstanding\_Debt']},  
 {'Metric': 'Average Age in Years', 'Value': selected\_data['Age']},  
 {'Metric': 'Average Annual Income in $', 'Value': selected\_data['Annual\_Income\_Code2']},  
 {'Metric': 'Credit Utilization Ratio', 'Value': selected\_data['Credit\_Utilization\_Ratio']},  
 {'Metric': 'Credit Score', 'Value': selected\_data['Credit\_Score']},  
 ]  
  
 return html.Div([  
 html.H2(f"Occupation: {selected\_occupation} - Age of credit history: {selected\_age} Years", style={'color': '#FF6347'}),  
 html.P("Only data corresponding to the specified occupation and the specified 'Age of Credit \n"  
 "History' are being used for the calculations", style={'margin-bottom': '20px'}),  
 dash\_table.DataTable(  
 id='selected-data-table',  
 columns=[{'name': 'Metric', 'id': 'Metric'}, {'name': 'Value', 'id': 'Value'}],  
 data=table\_data,  
 style\_table={'overflowX': 'auto'},  
 style\_header={'backgroundColor': 'rgb(30, 30, 30)', 'color': 'white'},  
 style\_cell={'backgroundColor': 'rgb(50, 50, 50)', 'color': 'white'},  
 )  
 ])  
  
 return html.Div()  
  
# Existing callback for set\_multi\_attributes\_options  
@app.callback(  
 Output('Multi', 'options'),  
 [Input('Correlation Heat Map', 'clickData')]  
)  
def set\_multi\_attributes\_options(selected\_data):  
 if selected\_data is None or not selected\_data['points']:  
 return [{'label': i, 'value' : i} for i in Numeric\_names]  
 else:  
 Remove = [selected\_data['points'][0]['x'], selected\_data['points'][0]['y']]  
 Numeric\_names1 = Numeric\_names.copy()  
 for remove in Remove:  
 if remove in Numeric\_names1:  
 Numeric\_names1.remove(remove)  
 return [{'label': i, 'value': i} for i in Numeric\_names1]  
  
# Existing callback for set\_multi\_attributes\_values  
@app.callback(  
 Output('Multi', 'value'),  
 [Input('Multi', 'options'),  
 State('Multi', 'value')]  
)  
def set\_multi\_attributes\_values(Available\_options, Current\_values):  
 Av\_Op\_values = [option['value'] for option in Available\_options]  
 return [value for value in Av\_Op\_values if value in Current\_values]  
  
# Existing callback for update\_RadarChart  
  
  
# Existing callback for set\_dist\_plots  
@app.callback(  
 Output('Radar Chart', "clickData"),  
 Input("Radar Chart", "clickData")  
)  
def set\_dist\_plots(Current\_option):  
 return Current\_option  
  
# Existing callback for Update\_add\_charts  
@app.callback(  
 Output('Radar Chart', 'figure'),  
 [Input('DropdownAttribute', 'value'),  
 Input('Multi', 'value'),  
 Input('Correlation Heat Map', 'clickData'),  
 Input("Add\_chart",'hoverData')  
 ]  
)  
def update\_RadarChart(DropdownAttribute, Multi, selected\_data, hoverData):  
 if callback\_context.triggered\_id == 'Add\_chart':  
 hoverDatavalue = hoverData["points"][0]["curveNumber"]  
 else:  
 hoverDatavalue = range(len(df[DropdownAttribute].unique()))  
  
 if selected\_data is None or not selected\_data['points']:  
 MultidropdownAttribute = Multi  
 else:  
 MultidropdownAttribute = [selected\_data['points'][0]['x'], selected\_data['points'][0]['y']]  
 for multi in Multi:  
 if multi not in MultidropdownAttribute:  
 MultidropdownAttribute.append(multi)  
  
 if type(hoverDatavalue) == int:  
 Unique\_Attributes = df[DropdownAttribute].unique()  
 Unique\_Attribute = [(Unique\_Attributes[hoverDatavalue])]  
 else:  
 Unique\_Attributes = df[DropdownAttribute].unique()  
 Unique\_Attribute = Unique\_Attributes[hoverDatavalue]  
  
 Mean\_Multi\_All = df[MultidropdownAttribute].mean()  
 Std\_Multi\_Att = df[MultidropdownAttribute].std()  
  
  
 fig = go.Figure()  
 for name\_uni in Unique\_Attribute:  
 Mean\_Multi\_AttData = df.loc[df[DropdownAttribute] == name\_uni, MultidropdownAttribute].mean()  
 Z\_value\_uni = (Mean\_Multi\_AttData - Mean\_Multi\_All) / Std\_Multi\_Att  
  
 fig.add\_trace(go.Scatterpolar(  
 r=Z\_value\_uni,  
 theta=pd.Series(MultidropdownAttribute),  
 name=name\_uni,  
 fill='toself'  
 ))  
  
 fig.update\_layout(showlegend=True,  
 polar=dict(radialaxis=dict(visible=True)),  
 plot\_bgcolor=colors['background'],  
 paper\_bgcolor=colors['background'],  
 font\_color=colors['text'],  
 transition\_duration=500,  
 )  
  
 return fig  
  
@app.callback(  
 Output('Add\_chart', 'figure'),  
 [Input('DropdownAttribute', "value"),  
 Input('X\_axis\_type', "value"),  
 Input("distribution", "value"),  
 Input('Radar Chart', "clickData")  
 ])  
def Update\_add\_charts(DropdownAttribute, X\_axis\_type, Hist\_rug\_none, clickData\_Radar):  
 clickedAttribute = clickData\_Radar['points'][0]['theta']  
  
 if Hist\_rug\_none == "Hist":  
 Hist = True  
 else:  
 Hist = False  
  
 hist\_data = []  
 for uni\_name in df[DropdownAttribute].unique():  
 Data\_uni\_att = df[df[DropdownAttribute] == uni\_name][clickedAttribute]  
 hist\_data.append(Data\_uni\_att)  
  
 fig = ff.create\_distplot(hist\_data, group\_labels=df[DropdownAttribute].unique(), show\_hist=Hist,  
 show\_rug=False, curve\_type='normal')  
  
 fig.update\_xaxes(title=clickedAttribute,  
 type='linear' if X\_axis\_type == 'linear' else 'log')  
  
 fig.update\_yaxes(title="Frequency")  
  
 fig.update\_layout(transition\_duration=500,  
 plot\_bgcolor=colors['background'],  
 paper\_bgcolor=colors['background'],  
 font\_color=colors['text'],  
 title="Distribution Chart")  
 return fig  
  
  
# New callback for bar chart interaction with scatter plot  
@app.callback(  
 Output('Scatter Plot', 'figure'),  
 [Input('credit-score-graph', 'clickData'),  
 Input('reset-scatter-button', 'n\_clicks')]  
)  
def update\_scatter\_plot(selected\_bar, reset\_button\_clicks):  
 ctx = callback\_context  
 if not ctx.triggered\_id:  
 raise PreventUpdate  
  
 trigger\_id = ctx.triggered\_id.split('.')[0]  
  
 if trigger\_id == 'reset-scatter-button' or not selected\_bar:  
 # Reset the scatter plot to show all data points  
 scatter\_figure = scatter\_fig  
 else:  
 selected\_occupation = selected\_bar['points'][0]['x']  
 filtered\_df = df[df['Occupation'] == selected\_occupation]  
  
 scatter\_figure = px.scatter(  
 filtered\_df,  
 x='Avg\_Annual\_Income',  
 y='Avg\_Outstanding\_Debt',  
 size='Credit\_Score\_Probability',  
 color='Occupation',  
 labels={'Avg\_Annual\_Income': 'Average Annual Income in $', 'Avg\_Outstanding\_Debt': 'Average Outstanding Debt in $'},  
 title=f'Scatter Plot for {selected\_occupation}',  
 hover\_data={'Credit\_Score\_Probability': ':.2%'},  
 size\_max=250  
 )  
  
 scatter\_figure.update\_layout(  
 plot\_bgcolor=colors['background'],  
 paper\_bgcolor=colors['background'],  
 font\_color=colors['text'],  
 title=f'Scatter Plot for {selected\_occupation}',  
 annotations=[  
 {  
 'x': 0.5,  
 'y': 1.07,  
 'xref': 'paper',  
 'yref': 'paper',  
 'text': 'Bubble size is proportional to the probability of the credit score being \'Poor\'',  
 'showarrow': False,  
 'font': {'size': 14, 'color': colors['text']}  
 }  
 ],  
 height=800  
 )  
  
 return scatter\_figure  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run\_server(debug=True)