Case Study - 1

In [1]:
```
#importing the required libraries

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
#Reading the csv file

loans_data=pd.read_csv("loans_full_schema.csv")
type(loans_data)
```

Out[2]:
```
pandas.core.frame.DataFrame
```

1. Describe the dataset and any issues with it.

As shown below, the dataset contains 10000 rows and 55 coulmns.

In [3]:
```
#Printing the number of rows of dataset
print(len(loans_data))

#Printing the dimensionality of the dataset
print(loans_data.shape)
```

```
10000
(10000, 55)
```

Displaying all the columns and their data types as shown below.

In [3]:

```
loans_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 55 columns):
emp_title                          9167 non-null object
emp_length                         9183 non-null float64
state                              10000 non-null object
homeownership                      10000 non-null object
annual_income                      10000 non-null float64
verified_income                    10000 non-null object
debt_to_income                     9976 non-null float64
annual_income_joint                1495 non-null float64
verification_income_joint          1455 non-null object
debt_to_income_joint               1495 non-null float64
delinq_2y                          10000 non-null int64
months_since_last_delinq           4342 non-null float64
earliest_credit_line               10000 non-null int64
inquiries_last_12m                 10000 non-null int64
total_credit_lines                 10000 non-null int64
open_credit_lines                  10000 non-null int64
total_credit_limit                 10000 non-null int64
total_credit_utilized              10000 non-null int64
num_collections_last_12m           10000 non-null int64
num_historical_failed_to_pay       10000 non-null int64
months_since_90d_late              2285 non-null float64
current_accounts_delinq            10000 non-null int64
total_collection_amount_ever       10000 non-null int64
current_installment_accounts       10000 non-null int64
accounts_opened_24m                10000 non-null int64
months_since_last_credit_inquiry   8729 non-null float64
num_satisfactory_accounts          10000 non-null int64
num_accounts_120d_past_due         9682 non-null float64
num_accounts_30d_past_due          10000 non-null int64
num_active_debit_accounts          10000 non-null int64
total_debit_limit                  10000 non-null int64
num_total_cc_accounts              10000 non-null int64
num_open_cc_accounts               10000 non-null int64
num_cc_carrying_balance            10000 non-null int64
num_mort_accounts                  10000 non-null int64
account_never_delinq_percent       10000 non-null float64
tax_liens                          10000 non-null int64
public_record_bankrupt             10000 non-null int64
loan_purpose                       10000 non-null object
application_type                   10000 non-null object
loan_amount                        10000 non-null int64
term                               10000 non-null int64
interest_rate                      10000 non-null float64
installment                        10000 non-null float64
grade                              10000 non-null object
sub_grade                          10000 non-null object
issue_month                        10000 non-null object
loan_status                        10000 non-null object
initial_listing_status             10000 non-null object
disbursement_method                10000 non-null object
balance                            10000 non-null float64
paid_total                         10000 non-null float64
paid_principal                     10000 non-null float64
paid_interest                      10000 non-null float64
paid_late_fees                     10000 non-null float64
dtypes: float64(17), int64(25), object(13)
memory usage: 4.2+ MB
```

In [4]:

```
# Displaying the first 5 rows of dataset

loans_data.head()
```

Out[4]:

	emp_title	emp_length	state	homeownership	annual_income	verified_income	debt_to_income	annual_income_joint	verification_
0	global config engineer	3.0	NJ	MORTGAGE	90000.0	Verified	18.01	NaN	
1	warehouse office clerk	10.0	HI	RENT	40000.0	Not Verified	5.04	NaN	
2	assembly	3.0	WI	RENT	40000.0	Source Verified	21.15	NaN	
3	customer service	1.0	PA	RENT	30000.0	Not Verified	10.16	NaN	
4	security supervisor	10.0	CA	RENT	35000.0	Verified	57.96	57000.0	

5 rows × 55 columns

In [6]:
```
# Displaying the last five rows of dataset

loans_data.tail()
```

Out[6]:

	emp_title	emp_length	state	homeownership	annual_income	verified_income	debt_to_income	annual_income_joint	verificati
9995	owner	10.0	TX	RENT	108000.0	Source Verified	22.28	NaN	
9996	director	8.0	PA	MORTGAGE	121000.0	Verified	32.38	NaN	
9997	toolmaker	10.0	CT	MORTGAGE	67000.0	Verified	45.26	107000.0	
9998	manager	1.0	WI	MORTGAGE	80000.0	Source Verified	11.99	NaN	
9999	operations analyst	3.0	CT	RENT	66000.0	Not Verified	20.82	NaN	

5 rows × 55 columns

Displaying the descriptive statistics for all columns as displayed below.

In [7]:
```
loans_data.describe()
```

Out[7]:

	emp_length	annual_income	debt_to_income	annual_income_joint	debt_to_income_joint	delinq_2y	months_since_last_delin
count	9183.000000	1.000000e+04	9976.000000	1.495000e+03	1495.000000	10000.00000	4342.00000
mean	5.930306	7.922215e+04	19.308192	1.279146e+05	19.979304	0.21600	36.76070
std	3.703734	6.473429e+04	15.004851	7.016838e+04	8.054781	0.68366	21.63493
min	0.000000	0.000000e+00	0.000000	1.920000e+04	0.320000	0.00000	1.00000
25%	2.000000	4.500000e+04	11.057500	8.683350e+04	14.160000	0.00000	19.00000
50%	6.000000	6.500000e+04	17.570000	1.130000e+05	19.720000	0.00000	34.00000
75%	10.000000	9.500000e+04	25.002500	1.515455e+05	25.500000	0.00000	53.00000
max	10.000000	2.300000e+06	469.090000	1.100000e+06	39.980000	13.00000	118.00000

8 rows × 42 columns

Issues with respect to the dataset:

1. Detecting the null values:

In [8]:
```
# To detect the missing values which returns True for NA values else False.

loans_data.isna()
```

Out[8]:

	emp_title	emp_length	state	homeownership	annual_income	verified_income	debt_to_income	annual_income_joint	verificati
0	False	False	False	False	False	False	False	True	
1	False	False	False	False	False	False	False	True	
2	False	False	False	False	False	False	False	True	
3	False	False	False	False	False	False	False	True	
4	False	False	False	False	False	False	False	False	
...	...	...	...	...	...	...	...	...	
9995	False	False	False	False	False	False	False	True	
9996	False	False	False	False	False	False	False	True	
9997	False	False	False	False	False	False	False	False	
9998	False	False	False	False	False	False	False	True	
9999	False	False	False	False	False	False	False	True	

10000 rows × 55 columns

In [9]:
```
# This prints the total number of missing values for each column.

loans_data.isna().sum()
```

Out[9]:
```
emp_title                           833
emp_length                          817
state                                 0
homeownership                         0
annual_income                         0
verified_income                       0
debt_to_income                       24
annual_income_joint                8505
verification_income_joint          8545
debt_to_income_joint               8505
delinq_2y                             0
months_since_last_delinq           5658
earliest_credit_line                  0
inquiries_last_12m                    0
total_credit_lines                    0
open_credit_lines                     0
total_credit_limit                    0
total_credit_utilized                 0
num_collections_last_12m              0
num_historical_failed_to_pay          0
months_since_90d_late              7715
current_accounts_delinq               0
total_collection_amount_ever          0
current_installment_accounts          0
accounts_opened_24m                   0
months_since_last_credit_inquiry   1271
num_satisfactory_accounts             0
num_accounts_120d_past_due          318
num_accounts_30d_past_due             0
num_active_debit_accounts             0
total_debit_limit                     0
num_total_cc_accounts                 0
num_open_cc_accounts                  0
num_cc_carrying_balance               0
num_mort_accounts                     0
account_never_delinq_percent          0
tax_liens                             0
public_record_bankrupt                0
loan_purpose                          0
application_type                      0
loan_amount                           0
term                                  0
interest_rate                         0
installment                           0
grade                                 0
sub_grade                             0
issue_month                           0
loan_status                           0
initial_listing_status                0
disbursement_method                   0
balance                               0
paid_total                            0
paid_principal                        0
paid_interest                         0
paid_late_fees                        0
dtype: int64
```

In [10]:
```
# Total number of missing values in the entire dataset.

loans_data.isna().sum().sum()
```

Out[10]: 42191

Solutions to fix the issues wrt null values:

In [11]:
```
# Drops the rows containg the null values

clean_loans_data=loans_data.dropna()
clean_loans_data.shape
```

Out[11]: (201, 55)

As shown above, the number of rows has been reduced to 201. Hence, by dropping the rows containing null values can result in loss of necessary information.

In [12]:
```
# The other way to fix this issue is by filling the null values with a fixed value like zero

clean_data=loans_data.fillna(value=0, inplace=True)
```

2. Detecting duplicates:

In [13]:
```
# Apart from missing data, there can also be duplicate rows in a dataframe.

duplicate_loans_data= loans_data[loans_data.duplicated()]
duplicate_loans_data.shape
```

Out[13]: (0, 55)

Solution to fix the issues wrt duplicates:

In [14]:
```
# But if there are any duplicates, it can be removed as shown below.

loans_data.drop_duplicates()
```

Out[14]:

	emp_title	emp_length	state	homeownership	annual_income	verified_income	debt_to_income	annual_income_joint	verificati
0	global config engineer	3.0	NJ	MORTGAGE	90000.0	Verified	18.01	0.0	
1	warehouse office clerk	10.0	HI	RENT	40000.0	Not Verified	5.04	0.0	
2	assembly	3.0	WI	RENT	40000.0	Source Verified	21.15	0.0	
3	customer service	1.0	PA	RENT	30000.0	Not Verified	10.16	0.0	
4	security supervisor	10.0	CA	RENT	35000.0	Verified	57.96	57000.0	
...	...	...	...	...	...	...	...	...	
9995	owner	10.0	TX	RENT	108000.0	Source Verified	22.28	0.0	
9996	director	8.0	PA	MORTGAGE	121000.0	Verified	32.38	0.0	
9997	toolmaker	10.0	CT	MORTGAGE	67000.0	Verified	45.26	107000.0	
9998	manager	1.0	WI	MORTGAGE	80000.0	Source Verified	11.99	0.0	
9999	operations analyst	3.0	CT	RENT	66000.0	Not Verified	20.82	0.0	

10000 rows × 55 columns

3. Another aspect would be detecting outliers in dataset which is an important segment in Exploratory Data Analysis. Outliers can play havoc when we want to apply Machine Learning algorithms for predictions.

2. Generate a minimum of 5 unique visualizations using the data and write a brief description of your observations.

1. Trends of Annual Income and Loan Amounts

The below distribution plot the variation in the distribution of Annual Income and Loan Amounts. We can see from the below graphs that the Annual Income mostly lie between 0 and 500000 and the maximum number of Loan Amount taken is approx 10000.

In [80]:
```
Amounts = ['annual_income','loan_amount']
for i in Amounts:
    sns.distplot(loans_data[i],color="red",bins=20)
    plt.title("Trends of "+ i, fontsize=15)
    plt.xlabel(i)
    plt.ylabel('count')
    plt.show()
```

```
C:\Python\PythonSoftware\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

2. Percentage of Number of Applicants Statewise Distribution Raking

The below pie chart gives the ranking of top 5 states with highest percentage of number of applicants.

In [16]:

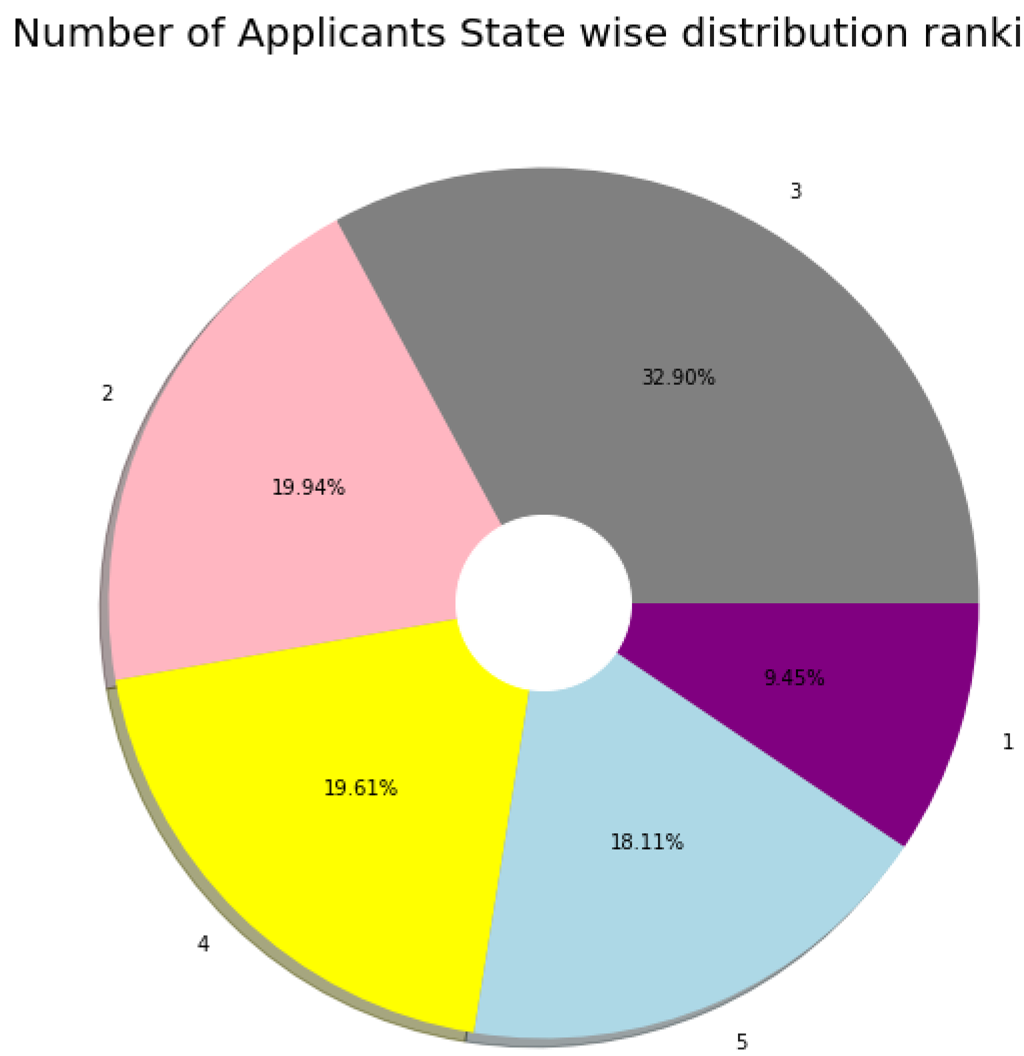
```
# Number of times each state has been mentioned in the dataset

loans_data['state'].value_counts()
```

Out[16]:

```
CA    1330
TX     806
NY     793
FL     732
IL     382
NJ     338
OH     338
GA     334
NC     299
PA     298
VA     261
AZ     255
MD     247
MI     245
MA     237
CO     235
WA     235
CT     181
IN     178
TN     167
MN     159
MO     159
NV     158
SC     145
OR     130
WI     128
AL     122
KY      97
LA      96
KS      89
OK      81
MS      72
AR      70
WV      68
UT      61
NE      56
RI      53
NH      47
NM      43
ID      38
HI      35
AK      33
ME      26
MT      24
DE      24
VT      23
SD      20
WY      19
DC      19
ND      14
Name: state, dtype: int64
```

In [17]:

```
# Percentage of Number of Applicants State wise distribution ranking (Top 5 states)

size=[1330,806,793,732,382]
labels="3","2","4","5","1"
colors=['grey','lightpink','yellow','lightblue','purple']

graph = plt.Circle((0,0), 0.2, color='white')

plt.rcParams['figure.figsize']=(10,10)
plt.pie(size, colors=colors, labels=labels, shadow=True, autopct="%.2f%%")
plt.title("Number of Applicants State wise distribution ranking", fontsize= 20)
p=plt.gcf()
p.gca().add_artist(graph)
plt.show()
```



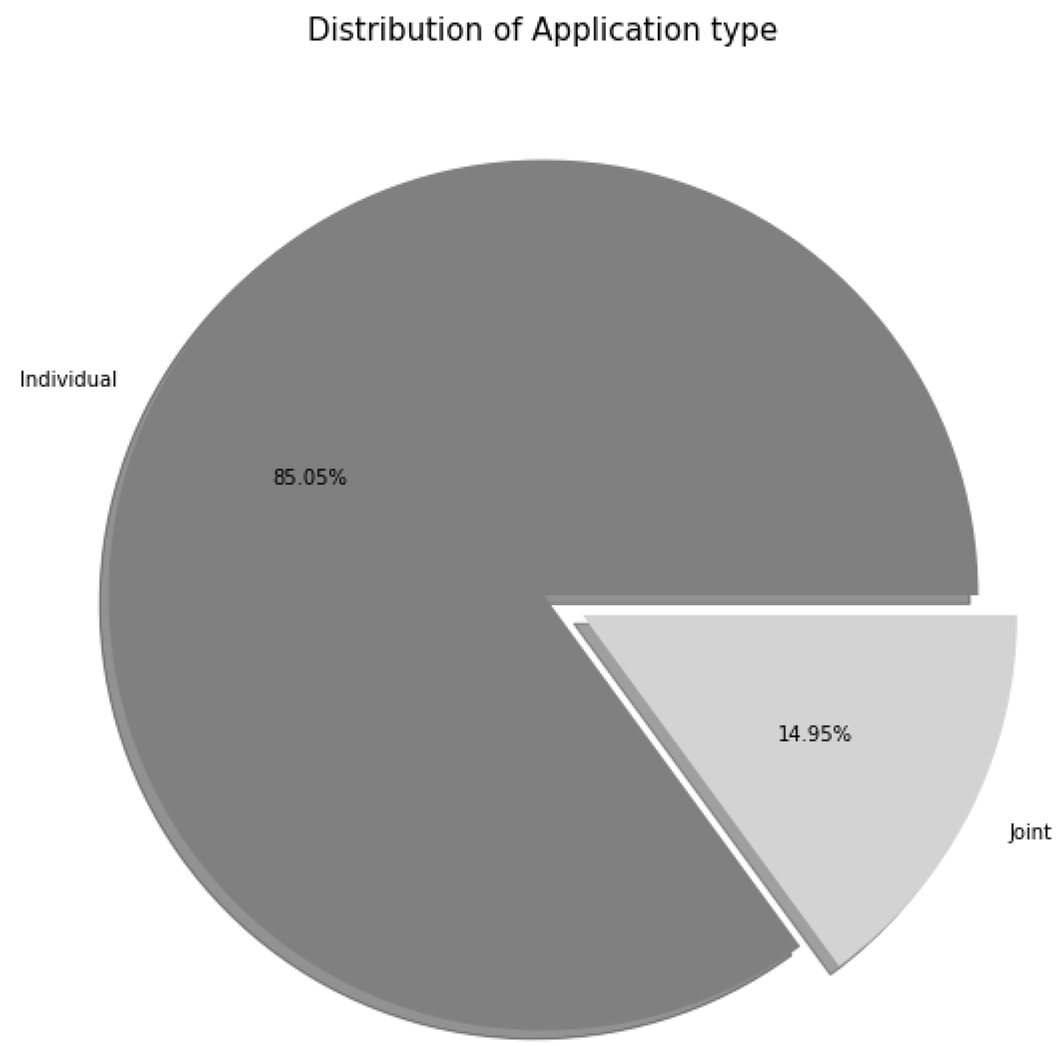
3. Distribution of Application Types

The below pie chart shows the percentage of distribution of application types i.e Individual and Joint application types. From below graph, we can say that maximum percentage of application type is Individual application type.

In [18]:
```
# Distribution of Application type

loans_data['application_type'].value_counts()
```

Out[18]:
```
individual    8505
joint         1495
Name: application_type, dtype: int64
```

In [19]:
```
size=[8505,1495]
labels="Individual", "Joint"
colors= ["grey","lightgrey"]
explode= [0,0.1]
plt.rcParams['figure.figsize']= (10,10)
plt.pie(size, colors=colors, explode=explode, labels=labels, shadow=True, autopct='%.2f%%')
plt.title('Distribution of Application type', fontsize=15)
plt.axis('off')
plt.show()
```

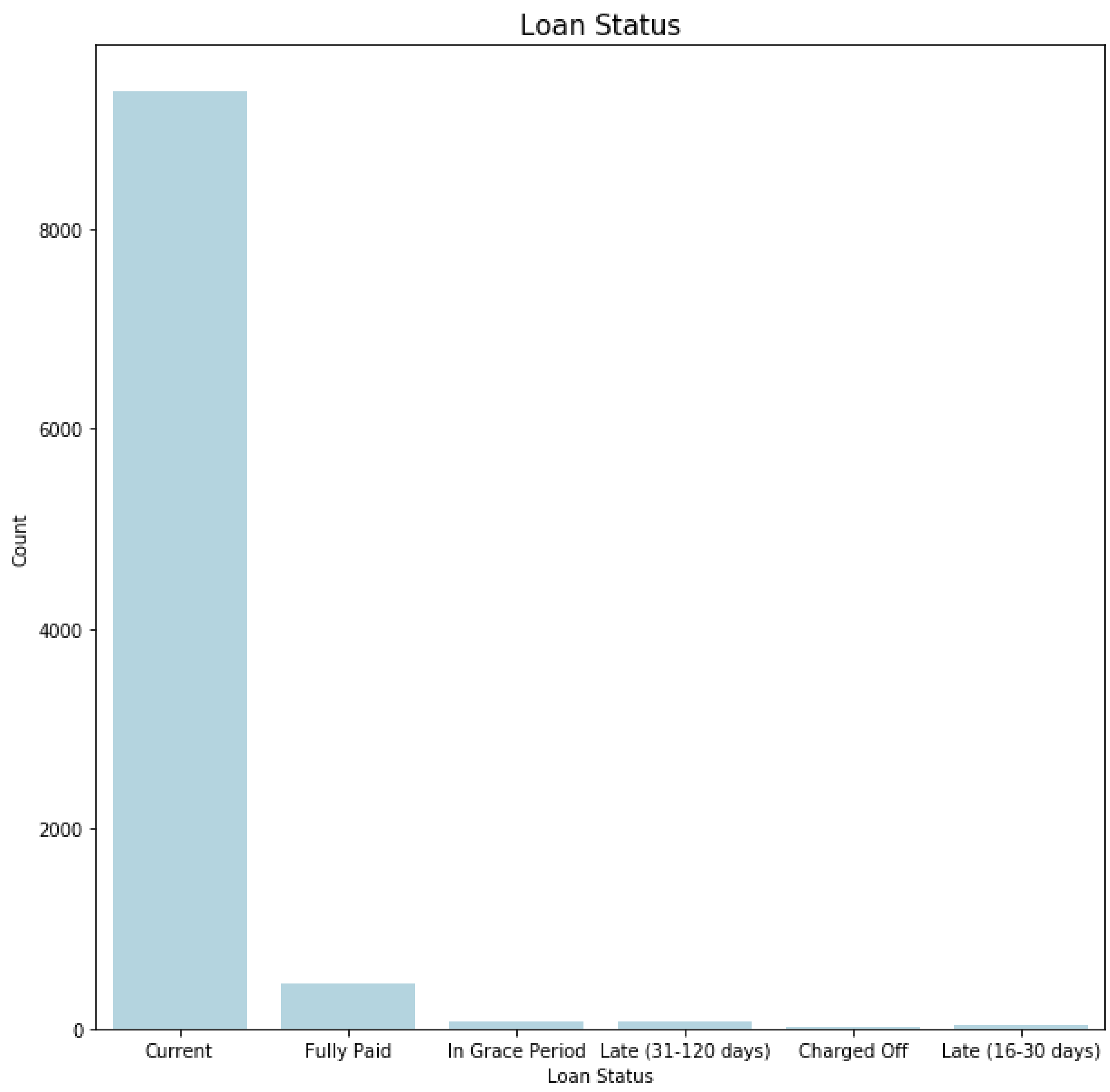


4. Count of Loan Status

The below bar graph gives us the count of each loan status. And from the below graph, we can say that current loan status has the highest count.

In [20]:
```
# Loan status

sns.countplot(loans_data['loan_status'],color="lightblue")
plt.title("Loan Status", fontsize=15)
plt.xlabel("Loan Status")
plt.ylabel("Count")
plt.show()
```

```
C:\Python\PythonSoftware\lib\site-packages\seaborn\_decorators.py:43: FutureWarning: Pass the following var
iable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passin
g other arguments without an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```



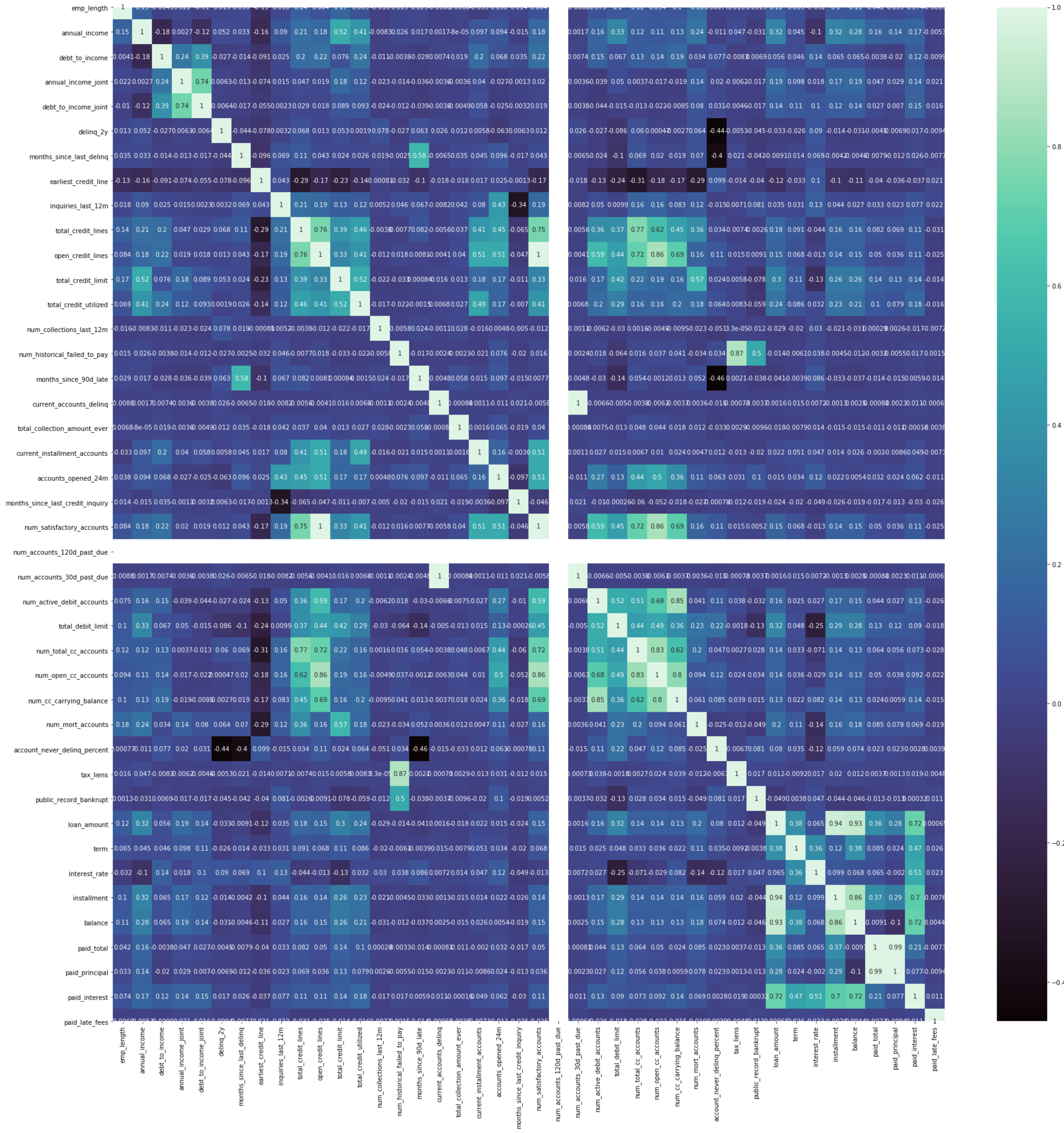
5. Correlation Matrix for all variables in the dataset

The below Correlation Matrix gives the correlation of all variables in the dataset.

In [21]:

```
#Correlation Matrix: Correlation of all variables

fig, ax = plt.subplots(figsize=(30,30))
sns.heatmap(loans_data.corr(), annot=True,cmap="mako")
```

Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x25a7273db08>

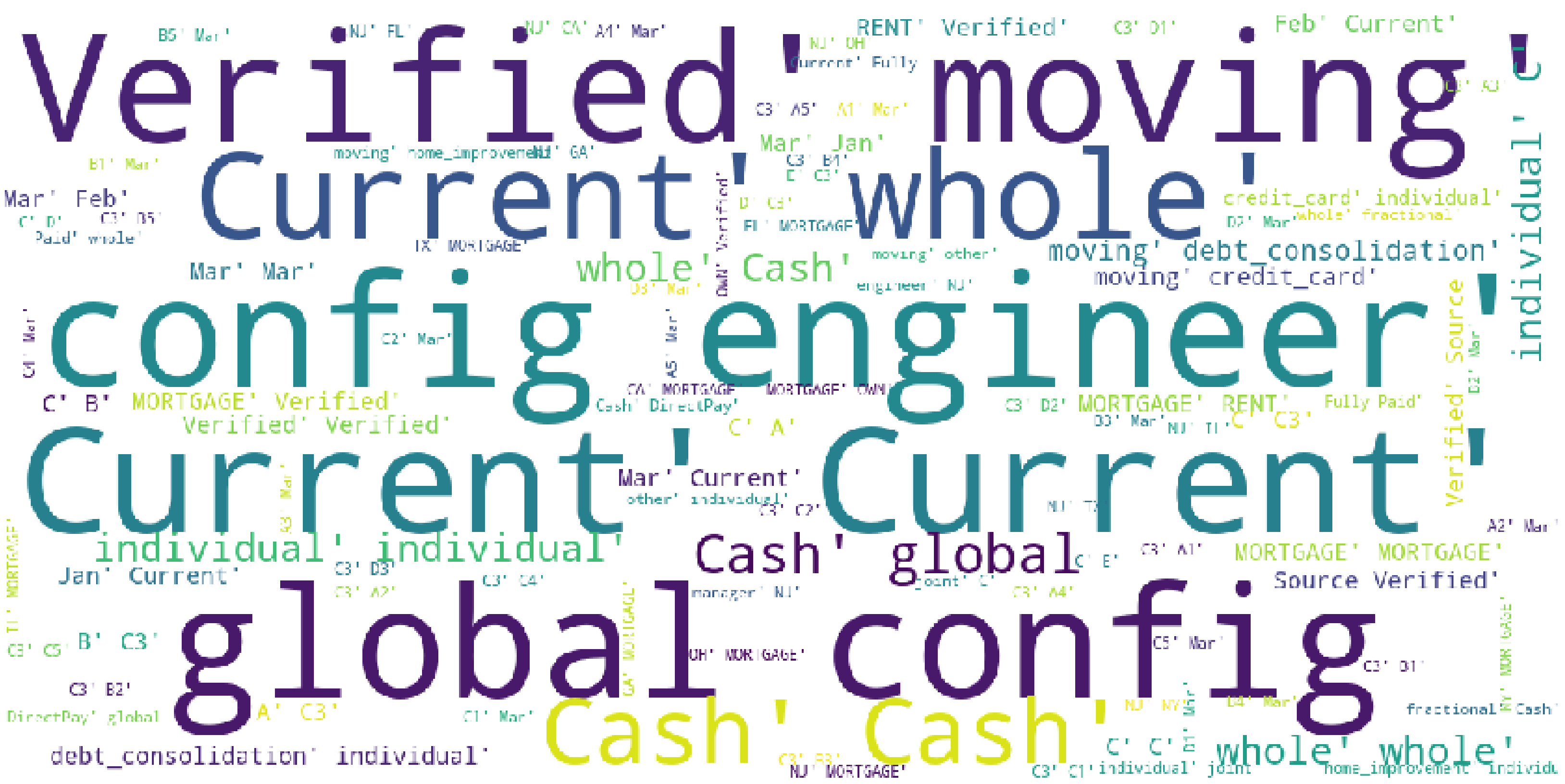


6. Word Cloud

The below word cloud displays the most prominent or frequent words in the entire dataset.

In [22]:

```
#Word Cloud

from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
stopwords = set(STOPWORDS)
import json
import numpy as np

with open('loans_full_schema.json', errors="ignore") as f:
    data = json.load(f)

# get the data in json format
text = []
for row in data:
    if (row != ""):
        text.append(row)

while('' in text) :
    text.remove('')

# text = np.delete(text['type'], 1, 0)
# print(text)


def show_wordcloud(data, title = None):
    wordcloud = WordCloud(
        background_color='white',
        stopwords=stopwords,
        max_words=2000,
        max_font_size=40,
        scale=3,
        random_state=1 # chosen at random by flipping a coin; it was heads
    ).generate(str(data))

    fig = plt.figure(1, figsize=(15, 15))
    plt.axis('off')
    if title:
        fig.suptitle(title, fontsize=10)
        fig.subplots_adjust(top=2.3)

    plt.imshow(wordcloud)
    plt.show()

show_wordcloud(text)
```



7. Dashboard of Loans Trends:

- The below dashboard gives the background information for different states which is displaying the common information for the selected states in the dashboard.

- The dashboard has 4 graphs included.

- The first graph displays the loan amounts range in each state. The second graph displays the range of interest rates for each state. The third graph displays the range of annual income in each state. Finally, the fourth graph displays the debt range for each state.

- Have to apply the below graphs to multiple states to see the changes in the trends of the bar graphs.

In [ ]:

```
import dash
from dash.dependencies import Input, Output
import dash_core_components as dcc
import dash_html_components as html
import pandas as pd
import plotly.express as px
global data
data = loans_data


#assets_external_path='/style.css'
app = dash.Dash(__name__)
server = app.server

global dict_products
def create_dict_list_of_product():
    dictlist = []
    unique_list = loans_data.state.unique()
    for state in unique_list:
        dictlist.append({'label': state, 'value': state})
    return dictlist

def dict_product_list(dict_list):
    product_list = []
    for dict in dict_list:
        product_list.append(dict.get('value'))
    return product_list


dict_products = create_dict_list_of_product()



app.layout = html.Div([
    html.Div([
        html.H1('Loans Trend Dashboard'),
        html.H2('Choose a State'),
        dcc.Dropdown(
            id='state-dropdown',
            options=[{'label':'Delaware','value':'Delaware'},
{'label':'Pennsylvania','value':'Pennsylvania'},
{'label':'New Jersey','value':'New Jersey'},

{'label':'Georgia','value':'GA'},
{'label':'Connecticut','value':'CT'},
{'label':'Massachusetts','value':'MA'},
{'label':'Maryland','value':'MD'},
{'label':'South Carolina','value':'SC'},
{'label':'New Hampshire','value':'NH'},
{'label':'Virginia','value':'VA'},
{'label':'New York','value':'NY'},
{'label':'North Carolina','value':'NC'},
{'label':'Rhode Island','value':'RI'},
{'label':'Vermont','value':'VT'},
{'label':'Kentucky','value':'KY'},
{'label':'Tennessee','value':'TN'},
{'label':'Ohio','value':'OH'},
{'label':'Louisiana','value':'LA'},
{'label':'Indiana','value':'IN'},
{'label':'Mississippi','value':'MS'},
{'label':'Illinois','value':'IL'},
{'label':'Alabama','value':'AL'},
{'label':'Maine','value':'ME'},
{'label':'Missouri','value':'MO'},
{'label':'Arkansas','value':'AR'},
{'label':'Michigan','value':'MI'},
{'label':'Florida','value':'FL'},
{'label':'Texas','value':'TX'},
{'label':'Iowa','value':'IA'},
{'label':'Wisconsin','value':'WI'},
{'label':'California','value':'CA'},
{'label':'Minnesota','value':'MN'},
{'label':'Oregon','value':'OR'},
{'label':'Kansas','value':'Kansas'},
{'label':'West Virginia','value':'WV'},
{'label':'Nevada','value':'NV'},
{'label':'Nebraska','value':'NE'},
{'label':'Colorado','value':'CO'},
{'label':'North Dakota','value':'ND'},
{'label':'South Dakota','value':'SD'},
{'label':'Montana','value':'MT'},
{'label':'Washington','value':'WA'},
{'label':'Idaho','value':'ID'},
{'label':'Wyoming','value':'WY'},
{'label':'Utah','value':'UT'},
{'label':'Oklahoma','value':'OK'},
{'label':'New Mexico','value':'NM'},
{'label':'Arizona','value':'AZ'},
{'label':'Alaska','value':'AK'},
{'label':'Hawaii','value':'HI'}],
```

```
            multi=True,
            value = ["GA"],
            searchable = True,
        ),

    ], style={'width': '40%', 'display': 'inline-block'}),
    html.Div([
        html.H2('Background Information of Selected States'),
        html.Table(id='my-table'),
        html.P(''),
    ], style={'width': '55%', 'float': 'right', 'display': 'inline-block'}),
    html.Div([
        html.H2('Counts of Loan Amounts  '),
        dcc.Graph(id='loanamount-graph'),
        html.P('')
    ], style={'width': '50%',  'display': 'inline-block'}),

    html.Div([
        html.H2('Counts of Interest Rates'),
        dcc.Graph(id='intrate-graph'),
        html.P('')
    ], style={'width': '50%',  'display': 'inline-block'}),

html.Div([
    html.H2('Counts of Annual Income'),
    dcc.Graph(id='other-graph'),
    html.P('')
], style={'width': '50%',  'display': 'inline-block'}),

html.Div([
    html.H2('Counts of Debts to Income'),
    dcc.Graph(id='multiple-graph'),
    html.P('')
], style={'width': '50%',  'display': 'inline-block'}),



])

@app.callback(Output('my-table', 'children'), [Input('state-dropdown', 'value')])
def generate_table(selected_dropdown_value, max_rows=5):

    df_filter = data[(data['state'].isin(selected_dropdown_value))]


    return [html.Tr([html.Th(col) for col in df_filter.columns])] + [html.Tr([
        html.Td(df_filter.iloc[i][col]) for col in df_filter.columns])
        for i in range(min(len(df_filter), max_rows))]


@app.callback(Output('loanamount-graph', 'figure'), [Input('state-dropdown', 'value')])

def update_graph(selected_dropdown_value):

    fig = loans_data.loc[(loans_data['state'].isin(selected_dropdown_value))]

    fig1 = px.bar(fig, x="state", y ='loan_amount')

    return fig1
@app.callback(Output('intrate-graph', 'figure'), [Input('state-dropdown', 'value')])

def update_graph(selected_dropdown_value):

    fig = loans_data.loc[(loans_data['state'].isin(selected_dropdown_value))]

    fig1 = px.bar(fig, x="state", y ='interest_rate')

    return fig1

@app.callback(Output('other-graph', 'figure'), [Input('state-dropdown', 'value')])

def update_graph(selected_dropdown_value):

    fig = loans_data.loc[(loans_data['state'].isin(selected_dropdown_value))]

    fig1 = px.bar(fig, x="state", y ='annual_income')

    return fig1

@app.callback(Output('multiple-graph', 'figure'), [Input('state-dropdown', 'value')])

def update_graph(selected_dropdown_value):

    fig = loans_data.loc[(loans_data['state'].isin(selected_dropdown_value))]

    fig1 = px.bar(fig, x="state", y ='debt_to_income')

    return fig1
```

```
if __name__ == '__main__':
    app.run_server(debug=False)
```

```
Dash is running on http://127.0.0.1:8050/ (http://127.0.0.1:8050/)

 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off

 * Running on http://127.0.0.1:8050/ (http://127.0.0.1:8050/) (Press CTRL+C to quit)
127.0.0.1 - - [02/Nov/2021 00:03:28] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [02/Nov/2021 00:03:28] "GET /_dash-layout HTTP/1.1" 200 -
127.0.0.1 - - [02/Nov/2021 00:03:28] "GET /_dash-dependencies HTTP/1.1" 200 -
127.0.0.1 - - [02/Nov/2021 00:03:29] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [02/Nov/2021 00:03:30] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [02/Nov/2021 00:03:30] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [02/Nov/2021 00:03:30] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [02/Nov/2021 00:03:30] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [02/Nov/2021 00:03:34] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [02/Nov/2021 00:03:34] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [02/Nov/2021 00:03:35] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [02/Nov/2021 00:03:35] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [02/Nov/2021 00:03:35] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [02/Nov/2021 00:03:36] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [02/Nov/2021 00:03:36] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [02/Nov/2021 00:03:36] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [02/Nov/2021 00:03:36] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [02/Nov/2021 00:03:36] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [02/Nov/2021 00:03:38] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [02/Nov/2021 00:03:38] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [02/Nov/2021 00:03:38] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [02/Nov/2021 00:03:38] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [02/Nov/2021 00:03:38] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [02/Nov/2021 00:03:40] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [02/Nov/2021 00:03:40] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [02/Nov/2021 00:03:40] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [02/Nov/2021 00:03:40] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [02/Nov/2021 00:03:40] "POST /_dash-update-component HTTP/1.1" 200 -
```

3. Create a feature set and create a model which predicts interest rate using at least 2 algorithms. Describe any data cleansing that must be performed and analysis when examining the data.

Linear Regression Model

In [23]:

```
from sklearn.preprocessing import LabelEncoder
import matplotlib.pylab as plt
import numpy as np
from scipy import sparse
from sklearn.datasets import make_classification, make_blobs, load_boston
from sklearn.decomposition import PCA
from sklearn.model_selection import ShuffleSplit, train_test_split
from sklearn import metrics
from sklearn.model_selection import learning_curve
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import GradientBoostingRegressor
from pprint import pprint
import pandas as pd
import urllib
import seaborn


# Converting the string type of data to numeric data type

loans_data=loans_data._convert(numeric=True)
loans_data.head(2)
```

Out[23]:

	emp_title	emp_length	state	homeownership	annual_income	verified_income	debt_to_income	annual_income_joint	verification_i
0	NaN	3.0	NJ	MORTGAGE	90000.0	Verified	18.01	0.0	
1	NaN	10.0	HI	RENT	40000.0	Not Verified	5.04	0.0	

2 rows × 55 columns

In [24]:
```
# Loading the interest rate into y variable

y=loans_data.interest_rate.values

# Performing Data Cleaning by dropping interest_rate and all the non-numeric columns

del loans_data['interest_rate']

loans_data=loans_data.drop(['emp_title','state','homeownership','verified_income','verification_income_joint
```

In [31]:
```
print(y)
```

```
[14.07 12.61 17.09 ... 23.88  5.32 10.91]
```

In [73]:
```
# Loading the features/dataset values into the variable x

x=loans_data.values
```

In [74]:
```
# Now training the test split

X_train, X_test, y_train, y_test = train_test_split(x,y)

# Fitting the Linear Regression model to the training set

linr=LinearRegression().fit(X_train, y_train)

# Printing the parameters we have learned

print ("Coefficients (theta_1..theta_n)")
print (linr.coef_)
print ("Y Intercept(theta0)")
print (linr.intercept_)

print ("R-squared for Train: %.2f" %linr.score(X_train, y_train))
print ("R-squared for Test: %.2f" %linr.score(X_test, y_test))
```

```
Coefficients (theta_1..theta_n)
[-1.15835931e-02 -1.69511496e-06  1.36740349e-02 -1.21070922e-06
  1.10813407e-02  2.46766010e-01 -1.04401761e-03  9.02511964e-03
  1.01456201e-01 -1.63528437e-02  2.69312562e-01 -5.04483665e-07
  3.86877085e-06  8.73926199e-01 -4.22616203e-02  2.69029390e-03
  7.51921255e-07 -7.08906011e-06 -1.29833554e-03  1.26454000e-01
 -1.33521731e-02 -2.99284284e-01  2.90286906e-10  8.12883094e-11
 -2.06623653e-02 -1.76818561e-05 -1.99084386e-02 -3.96734305e-02
  1.83322760e-01 -1.28061963e-01 -2.47644670e-02  4.16296060e-02
 -8.38879743e-02 -1.02310878e-03  3.38886450e-01  3.07062240e-02
 -8.79498828e-05  2.06200428e+01 -2.06200397e+01 -2.06147796e+01
 -2.05935936e+01]
Y Intercept(theta0)
-17.572922211128926
R-squared for Train: 0.66
R-squared for Test: 0.68
```

In [76]:
```
# Fitting the Linear regression model with normalize=True to the training set

linr=LinearRegression(normalize=True).fit(X_train, y_train)

# Getting the parameters we have learned

print ("Coefficients (theta_1..theta_n)")
print (linr.coef_)
print ("Y Intercept(theta0)")
print (linr.intercept_)

print ("R-squared for Train: %.2f" %linr.score(X_train, y_train))
print ("R-squared for Test: %.2f" %linr.score(X_test, y_test))
```

```
Coefficients (theta_1..theta_n)
[-1.15835931e-02 -1.69511496e-06  1.36740349e-02 -1.21070922e-06
  1.10813407e-02  2.46766010e-01 -1.04401761e-03  9.02511964e-03
  1.01456201e-01 -1.63528437e-02  2.69312562e-01 -5.04483665e-07
  3.86877085e-06  8.73926199e-01 -1.30336204e-02  2.69029390e-03
  3.30765033e-07 -7.08906010e-06 -1.29833555e-03  1.26454000e-01
 -1.33521731e-02 -2.99284284e-01  7.29341991e-06 -2.93320045e-06
 -2.06623653e-02 -1.76818561e-05 -1.99084386e-02 -3.96734305e-02
  1.83322760e-01 -1.28061963e-01 -2.47644670e-02  1.24016061e-02
 -1.13115974e-01 -1.02310878e-03  3.38886450e-01  3.07062240e-02
 -8.79498828e-05  2.06200426e+01 -2.06200395e+01 -2.06147793e+01
 -2.05935934e+01]
Y Intercept(theta0)
-17.57292220912076
R-squared for Train: 0.66
R-squared for Test: 0.68
```

The data cleansing to be performed are remove the missing or null values. Drop the interest rate after assigning the values to Y variable and delete all the non-numeric values from the dataset.Remove the special characters if there are any. Finally, convert all the values in the dataset to numberic to make sure that every value is numeric type. The mentioned steps have been shown above.

4. Visualize the test results and propose enhancements to the model, what would you do if you had more time. Also describe assumptions you made and your approach.

In the above models, the R-squared value on the test set is about 72%, which is not great but understandable considering the data must be much more sophisticated than a straight line. The only other thing we can do with this regressor is to normalize the data before training so that all values are in the same range from 0 to 1. If I had more time, I would explore more sophisticated regressors and convert the non-numeric/string data types to numeric values while building the model.

In [1]:
```

```

In [ ]:
```

```