VINTAGE ANALYSIS

Vikram Dewangan

Vintage? - You might recognize the term if you're a fan of aged wines or spirits.

When it comes to deciding y (dependent variable) for statistical models in credit risk, Vintage analysis and roll rate analysis comes into picture.

Vintage analysis helps us to decide the time period for which the loan should be tracked to evaluate the loan. As per BASEL, 90+ dpd is the definition of default, and that can be well supported by Vintage analysis.

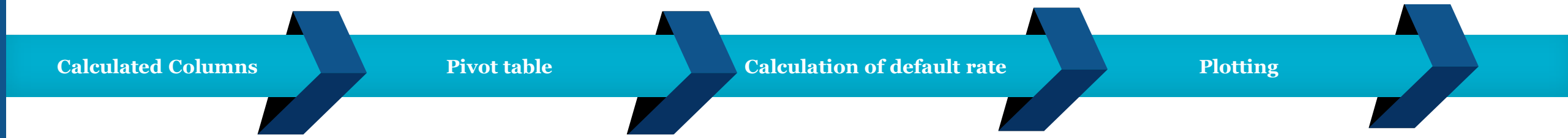
Must have variables in data for vintage analysis:

- Loan Id
- Reporting Month
- Origination Date
- Bucket

Calculated columns:

- Months on Books (MOB)
- Delinquency Status

Data to destiny



- Creating MOB (= Reporting Date – Origination Date) column.
- Creating Delinquency status column to capture the days past due (dpd).

- Create pivot table to capture 30 dpd, 60 dpd, 90 dpd.

- Calculate default rate for each MOB for every default definition.

- Plot default rate against Month on Books for every default definition.

Importing necessary libraries

```
#Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

import warnings

# Suppress all warnings
warnings.filterwarnings("ignore")
```

Reading datasets and checking data type

```
#importing dataset
df = pd.read_csv(
'Vintage and Roll rate analysis.csv'
)

#Lets have a look at the data
df.head()

#Checking data types
df.dtypes
```

Converting date columns to datatime data type

```
#Converting to datetime data type
df['Reporting Month'] = pd.to_datetime(df['Reporting Month'])
df['Origination date'] = pd.to_datetime(df['Origination date'])
```

Creating new columns Month on Books and Delinquency Bucket

```
# Calculating Loan Age/Month on Books
df['MOB'] = df.apply(
lambda row: (row['Reporting Month'].year
- row['Origination date'].year) * 12
+ row['Reporting Month'].month
- row['Origination date'].month,
axis=1)

#Creating new column for Delinquency Status
df['Delinq_Status'] = df['Bucket'].apply(
lambda x: 4 if x >= 4 else x)
```

Pivot tables for calculating different default definition

```
#Pivot table for calculating 30dpd, 60dpd, 90dpd
pivot_table = pd.pivot_table(
    df,
    values='Delinq_Status',
    index=['Loan Id'],
    columns=['MOB'],
    aggfunc='sum',
    fill_value=0
)
```

Find 1st occurrence of 1 in each row for 30 dpd, 2 for 60 dpd, 3 for 90 dpd

```
#Custom function to find 1st occurence
def find_first_one(row, to_find):
    # Iterate over the row to find the index of the first occurrence
    for idx, val in enumerate(row):
        if val == to_find:
            return idx + 1  # Adding 1 because index starts from 0
    return None  # If '1' is not found, return None

# Apply the custom function to each row
pivot_table['30 dpd'] = pivot_table.apply(
find_first_one, args=(1,), axis=1) + 5
pivot_table['60 dpd'] = pivot_table.apply(
find_first_one, args=(2,), axis=1) + 5
pivot_table['90 dpd'] = pivot_table.apply(
find_first_one, args=(3,), axis=1) + 5
```

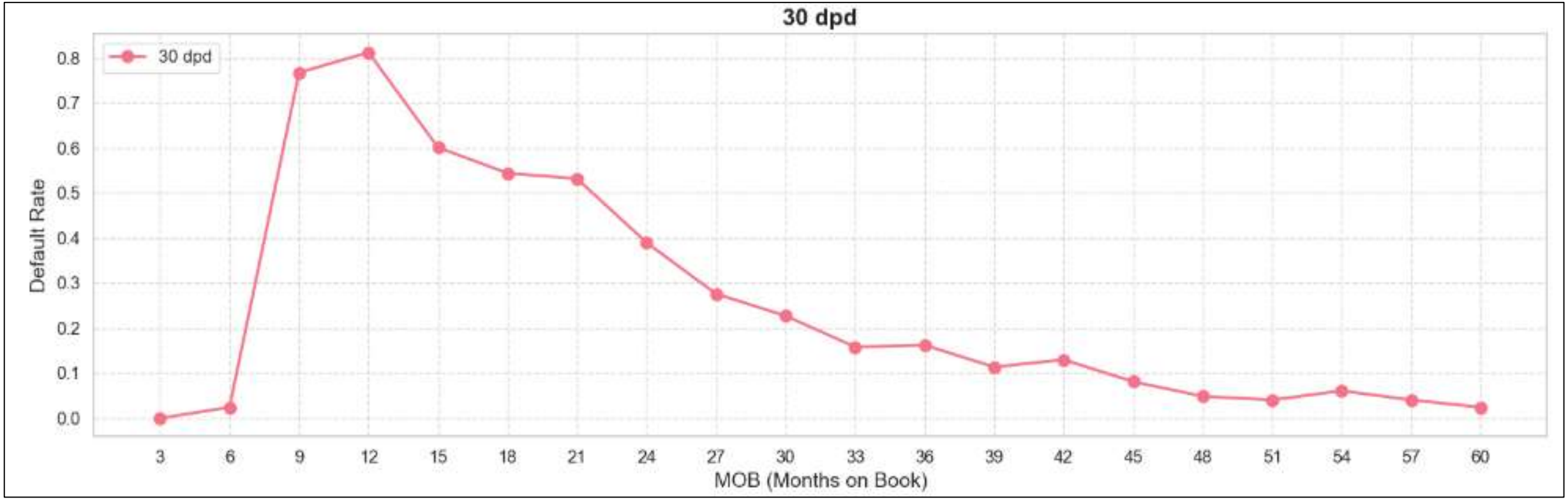
Preparing final data – Calculating default rate

```
# Define the countif function
def countif(column, criteria_values, total_count):
    counts = {}
    for value in criteria_values:
        counts[value] = (column == value).sum()
    total_nan = column.isna().sum()
    return counts, total_count - total_nan

# Criteria values to count
criteria_values = np.arange(3,61,3)

counts_30, total_count_30 = countif(pivot_table['30 dpd'],
criteria_values, len(pivot_table))
counts_60, total_count_60 = countif(pivot_table['60 dpd'],
criteria_values, len(pivot_table))
counts_90, total_count_90 = countif(pivot_table['90 dpd'],
criteria_values, len(pivot_table))
```

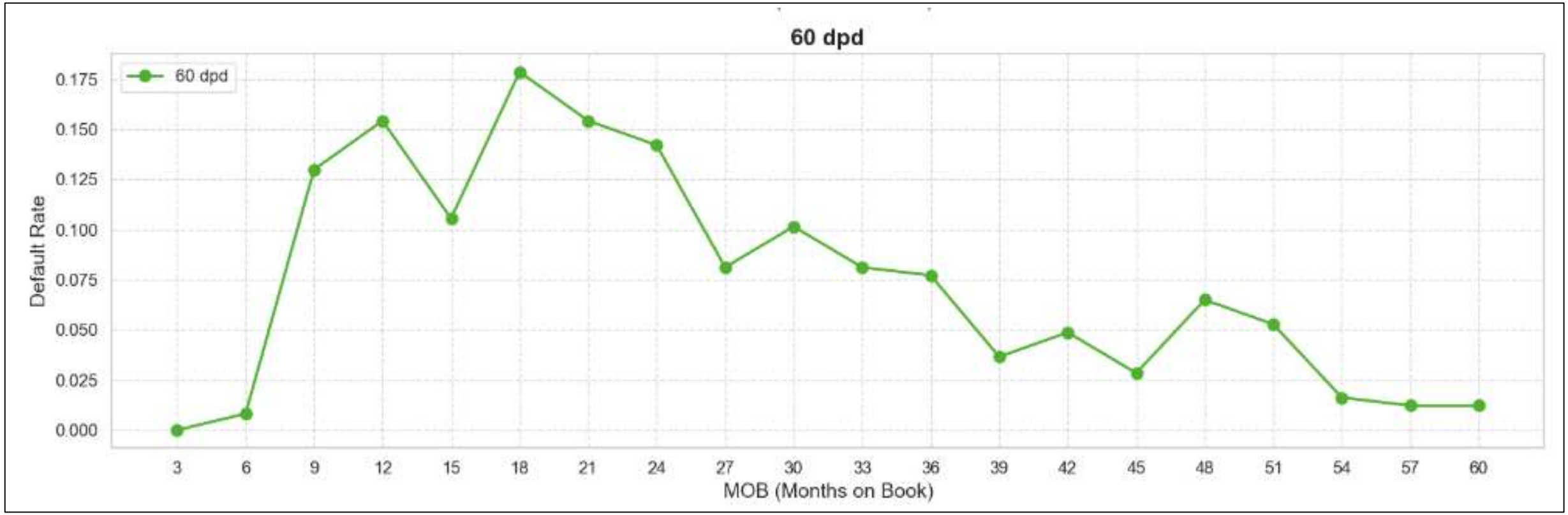
```
data = {
    '30 dpd': counts_30,
    '60 dpd': counts_60,
    '90 dpd': counts_90
}

# Convert data to DataFrame
df_final = pd.DataFrame(data)

#Total no of A/Cs
total_accounts = pivot_table.shape[0]
total_accounts

df_final = (df_final/total_accounts)*100
```

Plotting graphs for 30 dpd, 60 dpd, 90 dpd

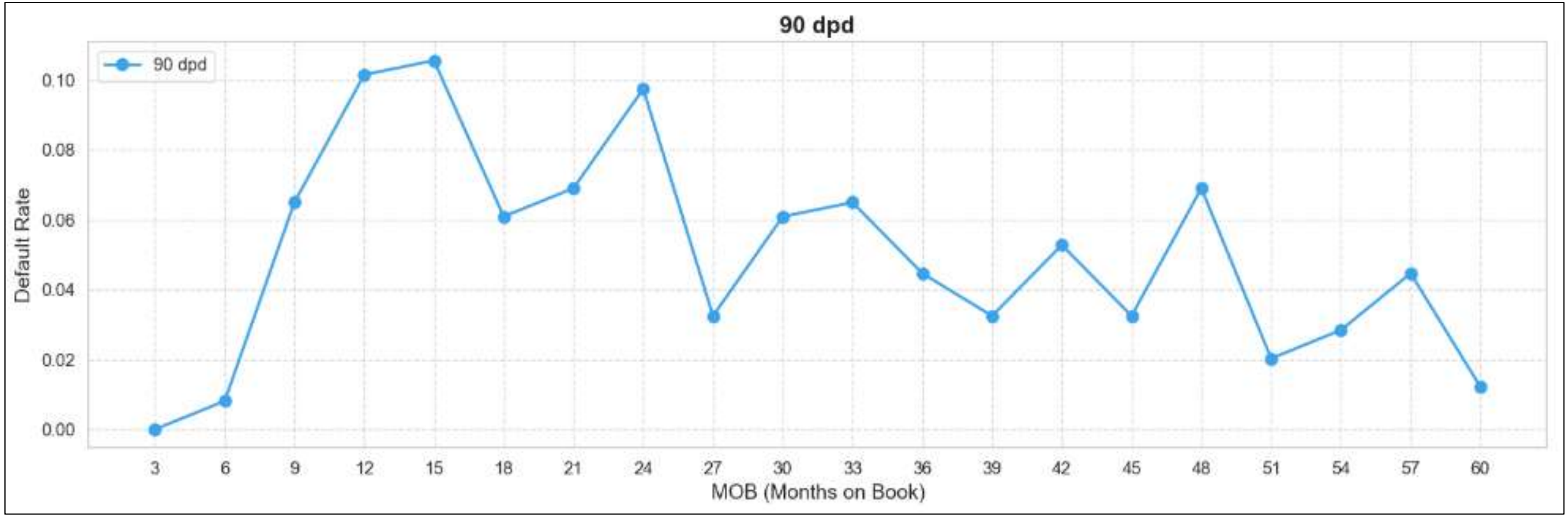
```
# Set the style and color palette
sns.set(style="whitegrid")
palette = sns.color_palette("husl", 3)

# Create subplots
fig, ax = plt.subplots(3, 1, figsize=(15, 15))

# List of columns and titles
columns = df_final.columns
titles = df_final.columns

# Plot data on each subplot
for i in range(len(columns)):
    ax[i].plot(df_final.index, df_final[columns[i]].values, label=columns[i], marker='o',
               color=palette[i], linestyle='-', linewidth=2, markersize=8)
    ax[i].set_xticks(criteria_values)
    ax[i].set_title(titles[i], fontsize=16, fontweight='bold')
    ax[i].legend(loc='upper left', fontsize=12)
    ax[i].set_xlabel('MOB (Months on Book)', fontsize=14)
    ax[i].set_ylabel('Default Rate', fontsize=14)
    ax[i].tick_params(axis='both', which='major', labelsize=12)
    ax[i].grid(True, linestyle='--', alpha=0.7)


# Adjust layout to make room for the main title
plt.tight_layout(rect=[0, 0, 1, 0.96])

# Show the plot
plt.show()
```



Default rate is highest at 12 MOB when 30 dpd is the default definition.



Default rate is highest at 18 MOB when 60 dpd is the default definition.



Default rate is highest at 15 MOB when 90 dpd is the default definition.

ROLL RATE ANALYSIS

Roll Rate analysis helps us to decide default definition. Definition of default is the dpd bucket beyond which the chances of cure is very low and chances of turning out worst in more.

Must have variables in data for roll rate analysis:

- Loan Id
- Reporting Month
- Bucket

Calculated columns:

- Source Bucket
- Destination Bucket

Data to destiny

- Filter all the rows where we have 12 reporting months.
- Take snapshot and performance date and bucket corresponding to that.
- Create transition matrix and calculate roll forward and roll backward rate.
- Plot roll backward rates against each bucket and select default definition.

Importing necessary libraries

```
#Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

import warnings

# Suppress all warnings
warnings.filterwarnings("ignore")
```

Reading datasets and checking data type

```
#importing dataset
df_roll = pd.read_csv('Roll Rate Data.csv')

#Lets have a look at data
df_roll.head()

#Checking data types
df.dtypes
```

Keep all the Loan Id for which there are 12 reporting month

```
# Calculate the count of unique
#reporting months for each account
df_roll['count'] = df_roll.groupby(
'Loan Id')['Reporting Month'].transform('nunique')

# Filter the DataFrame to keep only those accounts
#with exactly 12 unique reporting months
filtered_df = df_roll[df_roll['count'] == 12]
```

Converting Reporting column to data time data type and taking snapshot date and performance date

```
#Converting to datatime datatype
filtered_df['Reporting Month'] = pd.to_datetime(
                        df_roll['Reporting Month'])

#Snapshot Date
Source_Month = filtered_df['Reporting Month'].min()

#Performance Date
Destination_Month = pd.to_datetime(
filtered_df['Reporting Month'].min())
+ pd.DateOffset(months=11)
```

Preparing final data – Combining Loan Id, Source Bucket and Destination Bucket

```
df_sourceNdestination = pd.DataFrame(
    {
        'Load_Id' : filtered_df['Loan Id'].unique(),
        'Source_Bucket' : filtered_df[filtered_df['Reporting Month'] ==
        Source_Month]['Bucket'].values,
        'Destination_Bucket' : filtered_df[filtered_df['Reporting Month'] ==
        Destination_Month]['Bucket'].values
    }
)
```

Preparing Transition Matrix

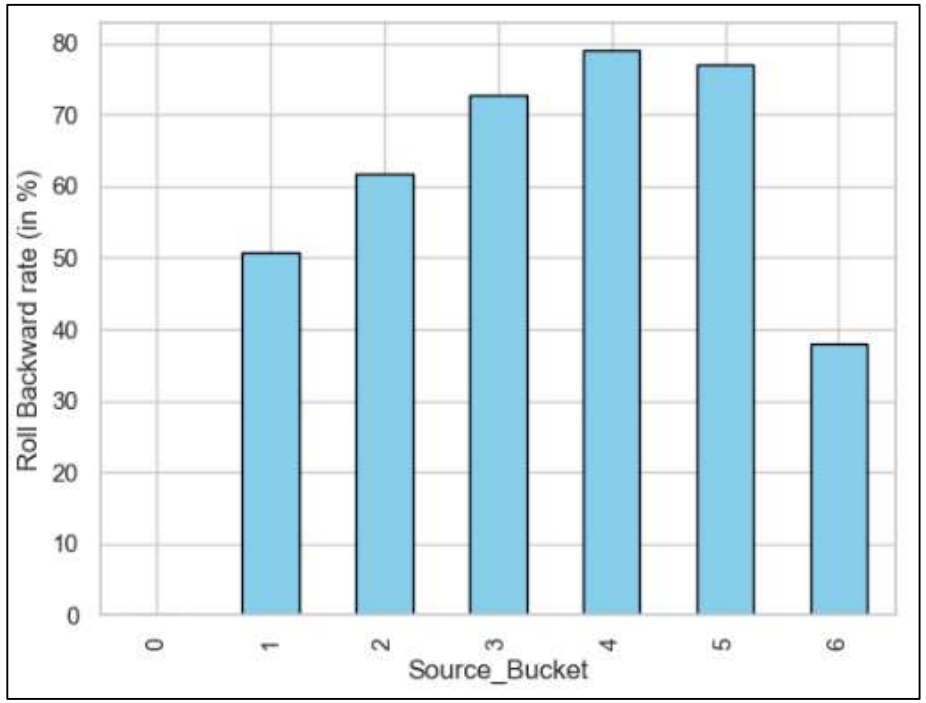
```
#Transition Matrix
pivot_TM = pd.pivot_table(
    df_sourceNdestination,
    values='Load_Id',
    index=['Source_Bucket'],
    columns=['Destination_Bucket'],
    aggfunc='count',
    fill_value=0
)
```

Calculating and Plotting Roll forward and Roll Backward rate

```
#Adding a column for Total no. of A/Cs in each bucket
pivot_TM['Total_Accounts'] = pivot_TM.sum(axis = 1)

# Calculate roll back(upgrade) column
rollback_values = []
rollforward_values = []
for index, row in pivot_TM.iterrows():
    rollback_values.append(sum(row[:index]) / row['Total_Accounts'])
    rollforward_values.append(sum(row[index+1:pivot_TM.shape[0]]) /
    row['Total_Accounts'])

# Add rollforward and rollbackward column column to DataFrame
pivot_TM['Roll_Back (Upgardes)'] = rollback_values
pivot_TM['Roll_Back (Upgardes)'] =
(pivot_TM['Roll_Back (Upgardes)'] * 100).map("{:.2f}%".format)
pivot_TM['Roll_Forward (Downgrades)'] = rollforward_values
pivot_TM['Roll_Forward (Downgrades)'] =
(pivot_TM['Roll_Forward (Downgrades)'] * 100).map("{:.2f}%".format)
```

```
#plot the roll backward rate for each bucket
pivot_TM['Roll_Back (Upgardes)'].str.rstrip('%').astype(float).plot(
kind='bar', color='skyblue', edgecolor='black')
plt.ylabel('Roll Backward rate (in %)')
```

Result of roll rate analysis



Destination_Bucket / Source_Bucket	0	1	2	3	4	5	6	Total_Accounts	Roll_Back (Upgardes)	Roll_Forward (Downgrades)
0	13025	339	73	33	20	9	34	13533	0.00%	3.75%
1	373	226	63	28	11	11	23	735	50.75%	18.50%
2	72	49	27	16	8	5	19	196	61.73%	24.49%
3	56	20	12	12	2	5	14	121	72.73%	17.36%
4	50	15	4	2	0	3	16	90	78.89%	21.11%
5	32	8	4	0	3	2	12	61	77.05%	19.67%
6	33	7	6	3	3	3	90	145	37.93%	0.00%

We will take 150+ dpd (bucket 6) as definition of default as chances of curing of accounts after that is very low.