In [246…

```
#Open CSV File
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
df = pd.read_csv('rellika_one_acre.csv')
df.head()
```

Out[246…

| | contract_reference | status | start_date | end_date | next_contract_paymen |
|---|---|---|---|---|---|
| **0** | abc0001 | Completed | 2022-05-20T08:23:51.406303 | 2022-05-20T08:23:51.406303 | 2022-05-20T08:23 |
| **1** | abc0002 | Active | 2022-05-25T13:28:49.873746 | NaN | 2022-08-31T18:34 |
| **2** | abc0003 | Active | 2022-05-31T10:02:23.158972 | NaN | 2022-07-22T01:1 |
| **3** | abc0004 | Active | 2022-07-05T11:49:03.801563 | NaN | 2022-08-18T13:34 |
| **4** | abc0005 | Active | 2022-05-31T06:31:25.977374 | NaN | 2022-07-04T13:30 |

# Task 1: Using Python read the provided dataset and derive the following metrics:

PAR(Portfolio at Risk) Status. Assuming the Finance request came in today(the day you receive the exercise), assign each client a PAR Status based on their repayment progression. This should be as a new column labeled ‘PAR status’ in the dataset.

In [247…

```
# Write a function to calculate how many days ago or in the future a date is fro
def is_date_in_future(date):
    today = pd.to_datetime('today')
    if pd.to_datetime(date) > today:
        return 'On Time'
    # between 0 and 7 days late
    elif (today - pd.to_datetime(date)).days <= 7:
        return 'PAR0-7'
    # between 8 and 30 days late
    elif (today - pd.to_datetime(date)).days <= 30:
        return 'PAR8-30'
    # Between 31 and 90 days late
    elif (today - pd.to_datetime(date)).days <= 90:
        return 'PAR31-90'
    # More than 90 days late
    else:
        return 'PAR90+'
```

In [248…

```
# Check column for future dates and create new column using apply function
# For each row in the column, apply the function
# I decide to name the new column 'par_status' for for uniformity with the other
```

```
df['par_status'] = df['next_contract_payment_due_date'].apply(is_date_in_future)
df.head()
```

Out[248…

|   | contract_reference | status | start_date | end_date | next_contract_paymen |
|---|---|---|---|---|---|
| **0** | abc0001 | Completed | 2022-05-20T08:23:51.406303 | 2022-05-20T08:23:51.406303 | 2022-05-20T08:2 |
| **1** | abc0002 | Active | 2022-05-25T13:28:49.873746 | NaN | 2022-08-31T18:3 |
| **2** | abc0003 | Active | 2022-05-31T10:02:23.158972 | NaN | 2022-07-22T01:1 |
| **3** | abc0004 | Active | 2022-07-05T11:49:03.801563 | NaN | 2022-08-18T13:3 |
| **4** | abc0005 | Active | 2022-05-31T06:31:25.977374 | NaN | 2022-07-04T13:3 |

Current Collection Rate which is derived by taking the Cumulative Amount Paid divided by (Expected Cumulative Amount Paid - Deposit)

In [249…

```
# Create a new column for the number of days late and name it 'current_collectio
df['current_collection_rate'] = df['cumulative_amount_paid']/(df['expected_cumul
# Check for NaN values and inf values
df[df['current_collection_rate'].isnull() | df['current_collection_rate'].isin([
```

Out[249…

|   | contract_reference | status | start_date | end_date | next_contract_payr |
|---|---|---|---|---|---|
| **0** | abc0001 | Completed | 2022-05-20T08:23:51.406303 | 2022-05-20T08:23:51.406303 | 2022-05-20T0 |
| **5** | abc0006 | Completed | 2022-06-09T06:40:08.391773 | 2022-06-09T06:40:08.391773 | 2022-06-09T0 |
| **7** | abc0008 | Completed | 2022-05-28T08:29:58.400869 | 2022-05-28T08:29:58.400869 | 2022-05-28T0 |
| **8** | abc0009 | Completed | 2022-05-20T07:54:21.755012 | 2022-05-20T07:54:21.755012 | 2022-05-20T |
| **12** | abc0013 | Active | 2022-09-12T11:57:45.18464 | NaN | 2022-10-06T1 |
| **...** | ... | ... | ... | ... |  |
| **984** | abc0985 | Completed | 2022-02-17T08:12:57.526295 | 2022-02-17T08:12:57.526295 | 2022-02-17T |
| **989** | abc0990 | Completed | 2022-02-14T07:55:28.134987 | 2022-02-14T07:55:28.134987 | 2022-02-14T0 |
| **991** | abc0992 | Completed | 2022-03-03T05:10:42.192882 | 2022-03-03T05:10:42.192882 | 2022-03-03T |
| **995** | abc0996 | Completed | 2022-02-22T10:37:49.307274 | 2022-02-22T10:37:49.307274 | 2022-02-22T1 |
| **999** | abc1000 | Completed | 2022-02-18T07:21:30.171788 | 2022-02-18T07:21:30.171788 | 2022-02-18T |

107 rows × 16 columns

In [250…

```
# All NaN values seem to be because of both the expected cumulative amount paid
# Replace NaN values with 0
df['current_collection_rate'] = df['current_collection_rate'].fillna(0)
# Replace inf values with 0
df['current_collection_rate'] = df['current_collection_rate'].replace([np.inf, -
df.head()
```

Out[250…

| | contract_reference | status | start_date | end_date | next_contract_paymen |
|---|---|---|---|---|---|
| **0** | abc0001 | Completed | 2022-05-20T08:23:51.406303 | 2022-05-20T08:23:51.406303 | 2022-05-20T08:23 |
| **1** | abc0002 | Active | 2022-05-25T13:28:49.873746 | NaN | 2022-08-31T18:3 |
| **2** | abc0003 | Active | 2022-05-31T10:02:23.158972 | NaN | 2022-07-22T01:1 |
| **3** | abc0004 | Active | 2022-07-05T11:49:03.801563 | NaN | 2022-08-18T13:3 |
| **4** | abc0005 | Active | 2022-05-31T06:31:25.977374 | NaN | 2022-07-04T13:30 |

Derive each client’s total amount in arrears which is the expected amount to have been paid at this time minus what has been paid.

In [251…

```
# client arrears status
# If par_status is not 'On Time' create a new column called 'client_arrears' and
# If par_status is 'On Time' create a new column called 'client_arrears' and set
df['client_arrears'] = np.where(df['par_status'] != 'On Time', df['expected_cumu
# If 'client_arrears' is less than 0 set it to 0. This means that the client has
df['client_arrears'] = np.where(df['client_arrears'] < 0, 0, df['client_arrears'
df.head()
```

Out[251…

| | contract_reference | status | start_date | end_date | next_contract_paymen |
|---|---|---|---|---|---|
| **0** | abc0001 | Completed | 2022-05-20T08:23:51.406303 | 2022-05-20T08:23:51.406303 | 2022-05-20T08:23 |
| **1** | abc0002 | Active | 2022-05-25T13:28:49.873746 | NaN | 2022-08-31T18:3 |
| **2** | abc0003 | Active | 2022-05-31T10:02:23.158972 | NaN | 2022-07-22T01:1 |
| **3** | abc0004 | Active | 2022-07-05T11:49:03.801563 | NaN | 2022-08-18T13:3 |
| **4** | abc0005 | Active | 2022-05-31T06:31:25.977374 | NaN | 2022-07-04T13:30 |

Payment Progression for each client. This is cumulative amount paid divided by the nominal contract value

In [252…

```
# Create a new column called 'payment_progression' and set it to cumulative amou
df['payment_progression'] = df['cumulative_amount_paid']/df['nominal_contract_va
# If 'payment_progression' is NaN or inf set it to 0. This NaN and inf values ar
```

```
df['payment_progression'] = df['payment_progression'].fillna(0)
df['payment_progression'] = df['payment_progression'].replace([np.inf, -np.inf],
df.head()
```

Out[252…

| | contract_reference | status | start_date | end_date | next_contract_paymen |
|---|---|---|---|---|---|
| **0** | abc0001 | Completed | 2022-05-20T08:23:51.406303 | 2022-05-20T08:23:51.406303 | 2022-05-20T08:23 |
| **1** | abc0002 | Active | 2022-05-25T13:28:49.873746 | NaN | 2022-08-31T18:34 |
| **2** | abc0003 | Active | 2022-05-31T10:02:23.158972 | NaN | 2022-07-22T01:1 |
| **3** | abc0004 | Active | 2022-07-05T11:49:03.801563 | NaN | 2022-08-18T13:34 |
| **4** | abc0005 | Active | 2022-05-31T06:31:25.977374 | NaN | 2022-07-04T13:30 |

Derive loan type from ‘name’ column: using the ‘name’ column, create a new column called Loan Type. Any entry in the name column that contains ‘Individual’ is an Individual Loan, any entry that contains ‘Group’ is a Group Loan, any entry that contains ‘Paygo’ is a Paygo Loan and any entry that contains ‘Cash’ is a Cash Sale

In [253…

```
# Derive loan type from name
# Search for ‘Individual’, ‘Group’, ‘Cash’ and ‘Paygo’ in the name column and cr
df['loan_type'] = np.where(df['name'].str.contains('Individual'), 'Individual',
df.head()
```

Out[253…

| | contract_reference | status | start_date | end_date | next_contract_paymen |
|---|---|---|---|---|---|
| **0** | abc0001 | Completed | 2022-05-20T08:23:51.406303 | 2022-05-20T08:23:51.406303 | 2022-05-20T08:23 |
| **1** | abc0002 | Active | 2022-05-25T13:28:49.873746 | NaN | 2022-08-31T18:34 |
| **2** | abc0003 | Active | 2022-05-31T10:02:23.158972 | NaN | 2022-07-22T01:1 |
| **3** | abc0004 | Active | 2022-07-05T11:49:03.801563 | NaN | 2022-08-18T13:34 |
| **4** | abc0005 | Active | 2022-05-31T06:31:25.977374 | NaN | 2022-07-04T13:30 |

## Writing the dataframe into a MySQL database table

Uncomment the below code to write the dataframe into a MySQL database table

Install the following packages `sqlalchemy` and `pymysql` before running the following code. Remember to change `psw` and `db` to your own databse and password

In [254…

```
from sqlalchemy import create_engine
psw = 'krellika'
```

```
db = 'DataAnalysis'
engine = create_engine('mysql+pymysql://root:'+psw+'@localhost/'+db)
```

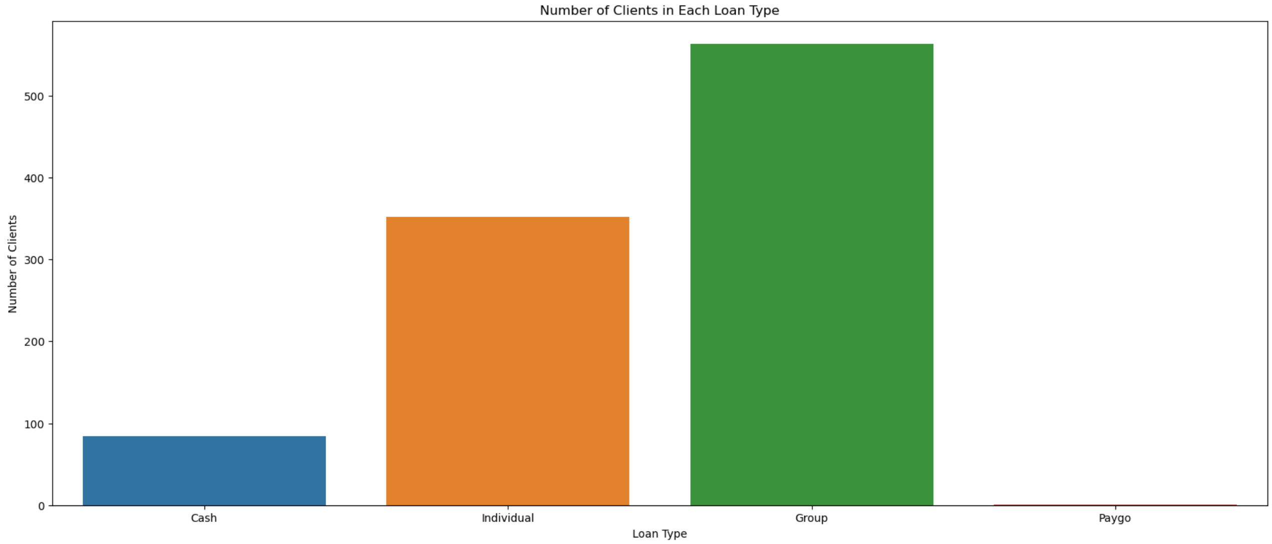
In [255…

```
# df_to_mysql = pd.read_csv('rellika_one_acre.csv')
# df_to_mysql.to_sql('rellika_one_acre', con=engine, if_exists='replace', index=
```

## Relationship between the number of clients and the type of loan taken

In [256…

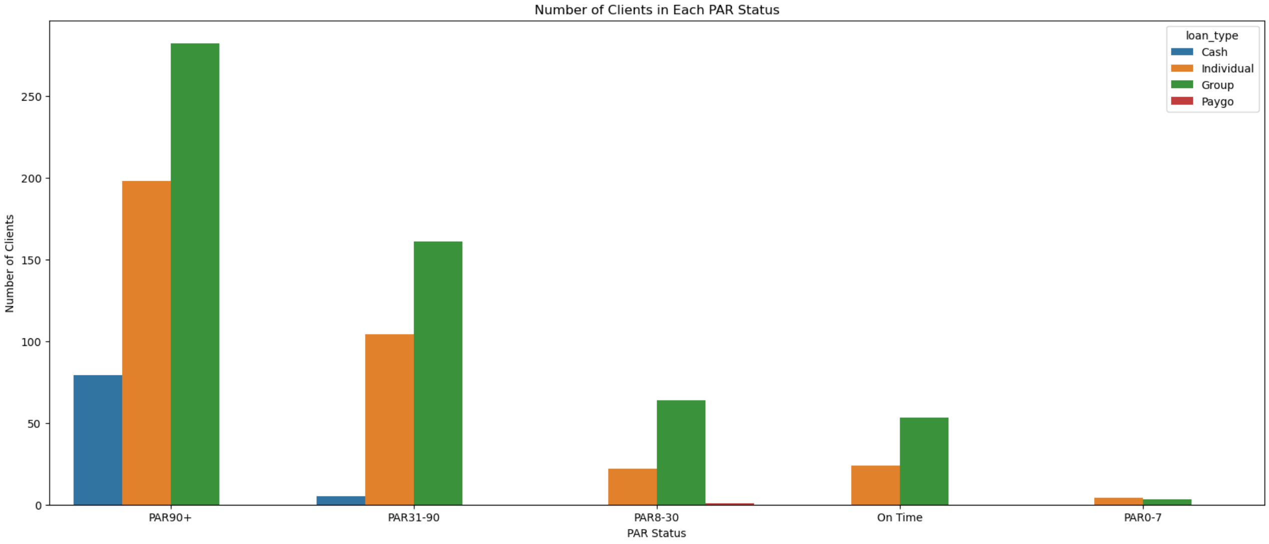
```
# Plot the number of clients in each loan type using matplotlib countplot
plt.figure(figsize=(20,8))
sns.countplot(x='loan_type', data=df)
plt.title('Number of Clients in Each Loan Type')
plt.xlabel('Loan Type')
plt.ylabel('Number of Clients')
plt.show()
```



The plot shows that the majority of clients are in the Group loan type. The number of clients in the Individual loan type is the second highest. It is advisable to focus on the Group loan type as it has the highest number of clients. Fewer resources should be allocated to the Paygo loan type as it has the lowest number of clients.

In [257…

```
# Group by month and count the number of clients in each par_status
# Plot the results using matplotlib countplot
plt.figure(figsize=(20,8))
sns.countplot(x='par_status', data=df, hue='loan_type')
plt.title('Number of Clients in Each PAR Status')
plt.xlabel('PAR Status')
plt.ylabel('Number of Clients')
plt.show()
```

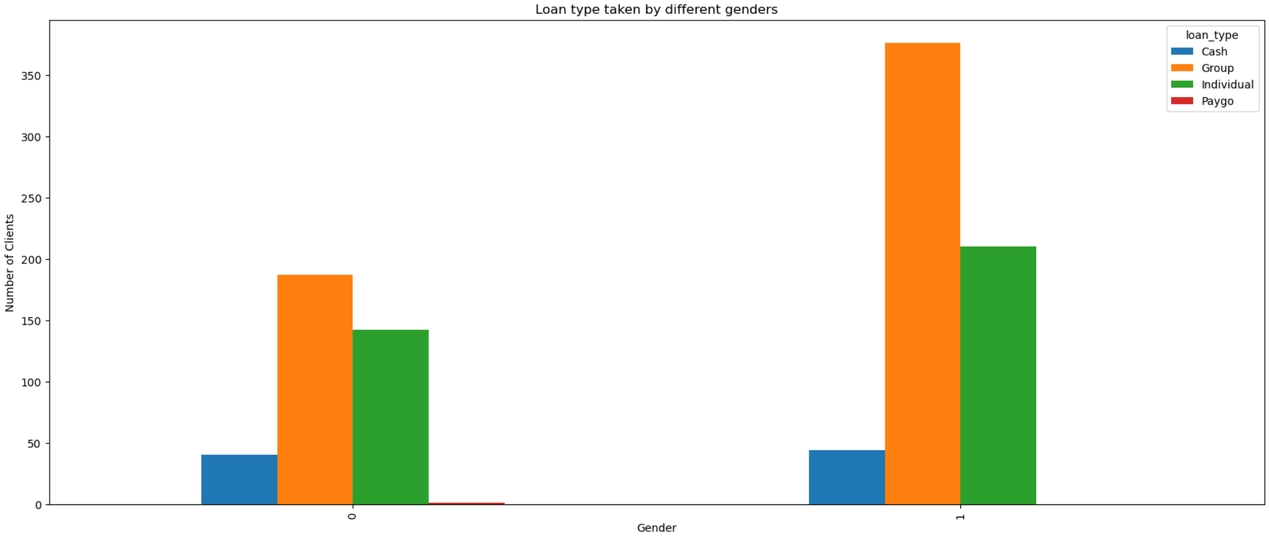


From the plot, it can be seen that a lot of people tend to pay late than the time they had indicated to pay the loan. This is seen across all the loan types. It can be seen that most people who take cash loans they tend to pay late. The plot shows that none of the people who borrowed loan on cash paid 'On Time" or even within past 7 days after due date

In [258…

```
# Plot the results using matplotlib countplot
plt.figure(figsize=(20,8))
df.groupby('gender')['loan_type'].value_counts().unstack().plot(kind='bar', stac
plt.title('Loan type taken by different genders')
plt.xlabel('Gender')
plt.ylabel('Number of Clients')
plt.show()
```

```
<Figure size 2000x800 with 0 Axes>
```

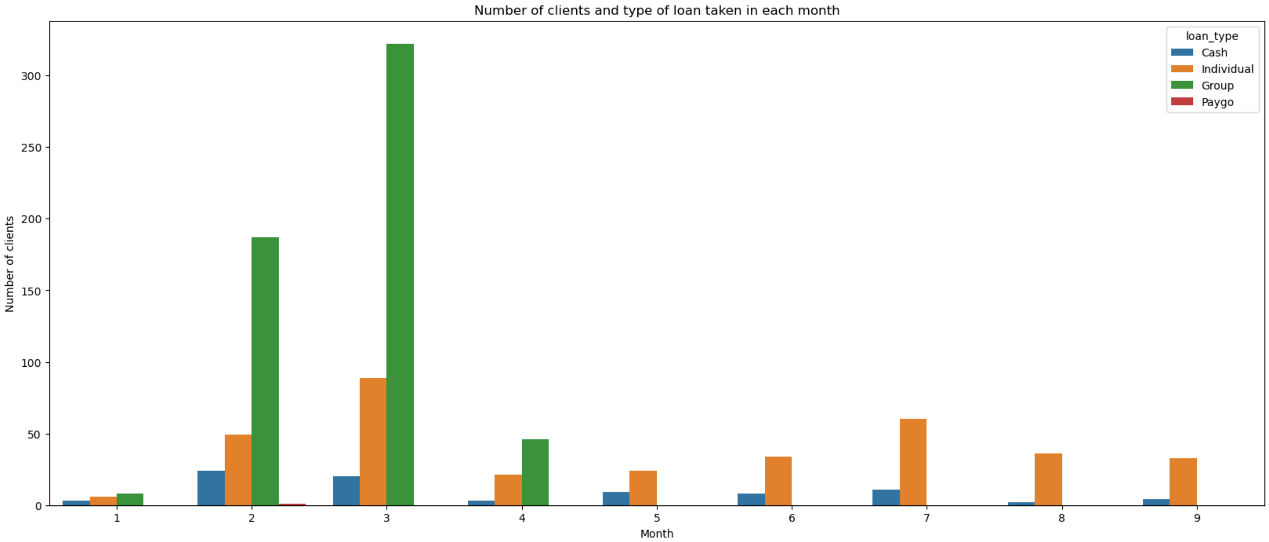


Gender 1 tends to get more loan than gender 0. It is advisable to try to research why gender 0 shys away from the loan. If the business could increase make gender 0 more interested in loan, they could raise yearly revenue.

In [259…

```
# Get month from next_contract_payment_due_date and create a new column called '
df['month'] = pd.DatetimeIndex(df['start_date']).month
```

```
# Group by month and count the number of clients in each par_status
# Plot the results using matplotlib countplot
plt.figure(figsize=(20,8))
sns.countplot(x='month', data=df, hue='loan_type')
plt.title('Number of clients and type of loan taken in each month')
plt.xlabel('Month')
plt.ylabel('Number of clients')
plt.show()
```



The plot shows that majority of clients tend to borrow more on February and March. This is also the time mostly groups borrow. It is advisable to focus on these months as they are the months with the highest number of clients.