Contents

Introduction: 2
- Source and nature of dataset: 2
- User requirements: 2
- Challenges and benefits: 2

Approach: 3

Recommendation: 4

Conclusion: 5

Appendices 6
- Appendix 1 6
- Appendix 2 13
- Appendix 3 13

Introduction:

Apex Financial Services have undergone an unprecedented rise in terms of loan applications which are more than 200% over the previous year. Throughout this shift, AFS can launch numerous campaigns online and one of them is creating a strong digital presence. An increase in website traffic and online application requests are now possible through AFS's application of digital marketing strategies. This choice of fortifying its advertising campaign in the field of both search engine optimization for the platforms like Google, Facebook, and LinkedIn is expecting to draw a broad market space and fulfill the need for small business loans since the demand is growing.

Source and nature of dataset:

The database provided is internal and obtained from ASF loan management system which is frequently updated to include the recent applications and approvals. The fit of database for data analytics is found in its comprehensive coverage of the major touch points of loan application processing. Through AFS, the lender obtains an invaluable data on applicant characteristics, lending patterns, and the driving force behind the lending decisions.

User requirements:

- Review application data of loans to reveal tendencies and patterns.
- Determine the likelihood of loan acceptance using applicant characteristics as the base.
- Observe approving loans rates in a period of time and evaluate influence of marketing campaigns.
- Create useful indicators, which are ultimately used to formulate strategies for decision making, as well as resource allocation.

Challenges and benefits:

There can be several challenges to be tackled by the AFS data analysts regarding the development and upkeep of the analytics code.

1. Data Quality:

   Trying to be as correct and complete as possible when creating the dataset to prevent faulty analysis and decision-making.
2. Model Complexity:

   Coping with the intricacy of predictive models by employing the accuracy and interpretability in equilibrium.
3. Scalability:

   Scaling the data analytics infrastructure to handle large volumes of loan application data precisely to get the efficient result.
4. Efficiency:

   Reusable code modules capture development efforts and manage code sustainability.
5. Consistency:

   Standardized coding practices leads to consistent results regardless of where the machine learning pipeline is implemented.
6. Flexibility:

   The modular code structure is designed for additional customization and sustaining the changing company needs as well.

Approach:

1. Code Libraries:

   For data exploration and preparation, libraries such as Pandas and NumPy are used. Pandas is utilized for data manipulation and analysis, while NumPy is employed for numerical operations and calculations. These libraries provide efficient tools for handling and processing large datasets, which is essential for analyzing loan application data.

2. Language Platform:

The choice of Python as the primary language is motivated by its versatility, ease of use, and extensive ecosystem of libraries for data analysis and machine learning. Python's popularity in the data science community also makes it a suitable choice for this project. Jupyter Notebooks or Python scripts can be used as the platform for development, providing an interactive and flexible environment for code execution and documentation.

3. Pseudo Code Design:

   In the design phase, pseudo code is used to outline the steps involved in the analysis process. This pseudo code serves as a blueprint for implementing the solution in Python or other programming languages if needed. The code includes loading the data, cleaning the data by handling missing values, duplicates, and outliers, performing feature engineering to create new features or modify existing ones, and building a classification model to predict loan approval.

4. Code Testing and Maintenance

   To test the code, sample loan data can be used to verify the calculations and ensure the accuracy of the predictions. Excel or other spreadsheet software can be employed for this purpose. Additionally, version control using Git and good commentary in the code are essential for maintaining codebase. Overall, this approach provides a systematic and structured framework for analyzing loan application data and deriving meaningful insights for decision-making at Apex Financial Services.

Recommendation:

To implement a solution for analyzing loan application data for Apex Financial Services, we can follow a structured data analysis lifecycle and framework. These libraries represent the tools of tackling large-scale data sets handling and processing, which stands for analyzing the loan application information.

- The auditing and reviewing the data collecting procedures on a regular basis create a platform for the identification and rectification of all inconsistencies and hiccups that might arise in the process. Adopting data profiling tools can also bring out the more advanced facet of the data and where improvements can be made.

- The other libraries like TensorFlow or PyTorch should be employed for building study deeper and complicated machine learning models. Thus, the depth of learning machines that libraries provide could contribute to more accurate predictions of loan granting.
- In staff training and support, the Apex Financial Services must supply the staff with training in the new tools and techniques employed for the data analysis and modeling.
- To ensure the regulation and ethics standards of personal data be implemented by these techniques such as data anonymization. It is going to provide customer privacy. Furthermore, the business should have regular audits and checks performed of the data processing and analysis for them to comply with the law and ethical guidelines.
- The core motivation for which is the usage of Python due to its versatility, ease of use and machine learning as well as data analysis are all supported by an extensive ecosystem of libraries. The choice of python is reasonable just due to its well-known reputation as a data science software by many data science experts and stakeholders.
- The diagrammatic process in the design stage is the use of pseudo-code which is used to map out the steps which are to be taken during analysis. Finally, classification model will be used to predict loan approval.

Conclusion:

The analytics team at AFS help the company use the loans application data quite effectively hence applicable for both business growth and improvement of customer satisfaction. AFS works on fulfilling the needs of users that finds solutions to difficulties and conforms to the regulations and ethics to lead the fast changing financial services industry. Embracing the user needs attitude, the company handles the problems and lives up to the regulatory and ethical standards, hence the AFS remains at the peak of the innovation in the financial services industry.

Appendices

Appendix 1

```
!pip install tabula-py pandas
import tabula
import pandas as pd
import fitz
from matplotlib.backends.backend_pdf import PdfPages
import matplotlib.pyplot as plt
```

```
Collecting tabula-py
  Downloading tabula_py-2.9.0-py3-none-any.whl (12.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.0/12.0 MB 39.1 MB/s eta 0:00:00
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (1.5.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from tabula-py) (1.25.2)
Requirement already satisfied: distro in /usr/lib/python3/dist-packages (from tabula-py) (1.7.0)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2023.4)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas) (1.16.0)
Installing collected packages: tabula-py
Successfully installed tabula-py-2.9.0
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
pdf_path = '/content/drive/MyDrive/APEX_Loans_Database_Table.pdf'
excel_path = '/content/drive/MyDrive/APEX Loan Data.xlsx'
```

```
# loan databe
df_loan_data = pd.read_excel(excel_path)
df_loan_data.head()
```

	Loan_ID	Gender	Married	Dependents	Graduate	Self_Employed	ApplicantIncome	CoapplicantIncome	L
0	2284	1	0	0	0	0	3902	1666.0	
1	2287	2	0	0	1	0	1500	1800.0	
2	2288	1	1	2	0	0	2889	0.0	
3	2296	1	0	0	0	0	2755	0.0	
4	2297	1	0	0	1	0	2500	20000.0	

Next steps: Generate code with df_loan_data   View recommended plots

```
[ ] # loan_database

    #data from pdf tfile as table of dataframe
    dfs = tabula.read_pdf(pdf_path, pages='all', multiple_tables=False)

    columns = ['Loan_ID', 'Gender', 'Married', 'Dependents', 'Graduate', 'Self_Employed',
               'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term',
               'Credit_History', 'Property_Area', 'Loan_Status']
    #list of df
    selected_dfs = []

    #specific columns to DataFrame
    for df in dfs:
        selected_dfs.append(df[columns])
    df_loan_database = pd.concat(selected_dfs, ignore_index=True)
    df_loan_database.head()
```

	Loan_ID	Gender	Married	Dependents	Graduate	Self_Employed	ApplicantIncome	CoapplicantIr
0	1002	1	0	0	1	0	5849	
1	1003	1	1	1	1	0	4583	1!
2	1005	1	1	0	1	1	3000	
3	1006	1	1	0	0	0	2583	2:
4	1008	1	0	0	1	0	6000	

Next steps: Generate code with df_loan_database View recommended plots

```
#datatype of loan_database
print('Loand database')
print()
print(df_loan_database.dtypes)
#datatype of loand_data
print()
print('Loand data')
print()
print(df_loan_data.dtypes)
```

```
Loand database

Loan_ID               int64
Gender                int64
Married               int64
Dependents            int64
Graduate              int64
Self_Employed         int64
ApplicantIncome       int64
CoapplicantIncome   float64
LoanAmount            int64
Loan_Amount_Term      int64
Credit_History        int64
```

```
#preprocessing libraries
import numpy as np
from scipy import stats
from sklearn.preprocessing import StandardScaler
```

```
#for Loan database
# Missing Values
df_loan_database = df_loan_database.dropna()
#  Duplicates
df_loan_database = df_loan_database.drop_duplicates()
# Outliers
df_loan_database = df_loan_database[(np.abs(stats.zscore(df_loan_database.select_dt
```

```
print(df_loan_database.head())
```

```
   Loan_ID  Gender  Married  Dependents  Graduate  Self_Employed  \
0     1002       1        0           0         1              0
1     1003       1        1           1         1              0
2     1005       1        1           0         1              1
3     1006       1        1           0         0              0
4     1008       1        0           0         1              0

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             5849                0.0         128               360
1             4583             1508.0         128               360
2             3000                0.0          66               360
3             2583             2358.0         120               360
4             6000                0.0         141               360

   Credit_History  Property_Area Loan_Status
0               1              1           Y
1               1              3           N
2               1              1           Y
3               1              1           Y
```

```
#for loan data
# Missing Values
df_loan_data = df_loan_data.dropna()
# Duplicates
df_loan_data = df_loan_data.drop_duplicates()
# Outliers
df_loan_data = df_loan_data[(np.abs(stats.zscore(df_loan_data.select_dtypes(include=['number'])))
print(df_loan_data.head())
```

```
   Loan_ID  Gender  Married  Dependents  Graduate  Self_Employed  \
0     2284       1        0           0         0              0
1     2287       2        0           0         1              0
2     2288       1        1           2         0              0
3     2296       1        0           0         0              0
5     2300       2        0           0         0              0

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             3902             1666.0         109               333
1             1500             1800.0         103               333
2             2889                0.0          45               180
3             2755                0.0          65               300
5             1963                0.0          53               333

   Credit_History  Property_Area Loan_Status
0               1              3           Y
1               0              2           N
2               0              1           N
3               1              3           N
5               1              2           Y
```

Descriptive anlysis on loan data

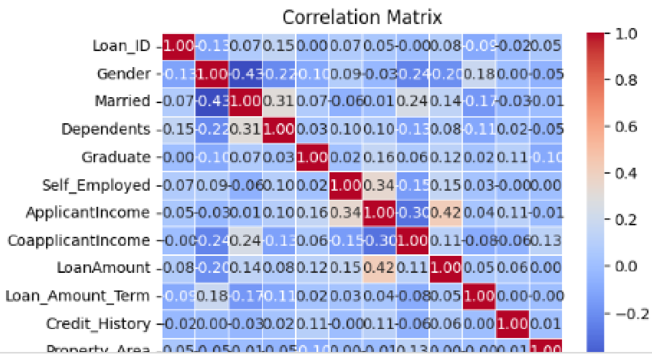
```
[ ] # Statistics summary
    summary_stats = df_loan_data.describe()
    print("Summary Statistics:")
    print(summary_stats)
```

```
Summary Statistics:
           Loan_ID      Gender    Married  Dependents    Graduate  \
count   230.000000  230.000000  230.00000  230.000000  230.000000
mean   2538.795652    1.195652    0.63913    0.726087    0.730435
std     303.166314    0.397567    0.48130    0.984029    0.444702
min    1900.000000    1.000000    0.00000    0.000000    0.000000
25%    2369.250000    1.000000    0.00000    0.000000    0.000000
50%    2550.000000    1.000000    1.00000    0.000000    1.000000
75%    2777.750000    1.000000    1.00000    1.000000    1.000000
max    2990.000000    2.000000    1.00000    3.000000    1.000000

       Self_Employed  ApplicantIncome  CoapplicantIncome  LoanAmount  \
count     230.000000       230.000000         230.000000  230.000000
mean        0.139130      4772.643478        1314.482261  142.769565
std         0.346837      3423.599585        1492.973907   69.560968
min         0.000000       210.000000           0.000000    9.000000
25%         0.000000      2751.250000           0.000000  100.000000
50%         0.000000      3675.500000        1252.500000  128.000000
75%         0.000000      5696.500000        2139.500000  170.250000
max         1.000000     20667.000000        8333.000000  405.000000

       Loan_Amount_Term  Credit_History  Property_Area
count        230.000000      230.000000     230.000000
mean         325.682609        0.752174       2.086957
std           47.422832        0.432692       0.782619
min          180.000000        0.000000       1.000000
25%          333.000000        1.000000       1.000000
50%          333.000000        1.000000       2.000000
75%          333.000000        1.000000       3.000000
max          480.000000        1.000000       3.000000
```

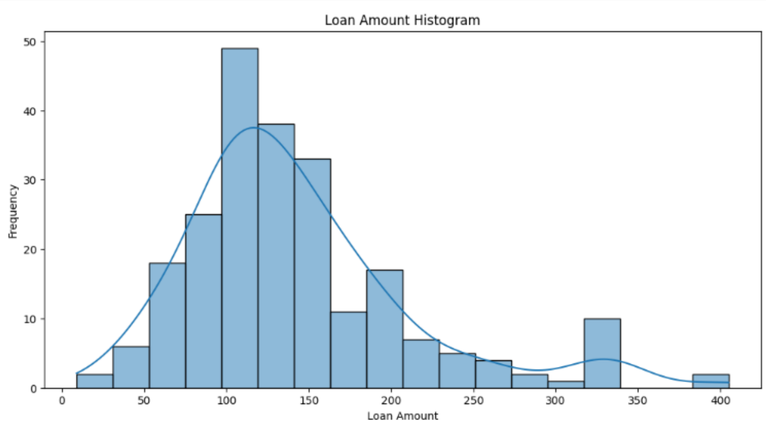
```
Frequency distribution for Loan_Status
Y    157
N     73
Name: Loan_Status, dtype: int64
```

```python
# Correlation Analysis
correlation_matrix = df_loan_data.corr()
print("\nCorrelation Matrix:")
import seaborn as sns
# Plot Matrix
plt.figure(figsize=(6, 4))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title("Correlation Matrix")
plt.show()
```

```
<ipython-input-70-140fb881eee3>:2: FutureWarning: The default value of numeric_only in DataFrame.c
  correlation_matrix = df_loan_data.corr()

Correlation Matrix:
```




```python
#plot data
plt.figure(figsize=(12, 6))
sns.histplot(df_loan_data['LoanAmount'], kde=True)
plt.title('Loan Amount Histogram')
plt.xlabel('Loan Amount')
plt.ylabel('Frequency')
plt.grid(False)
plt.show()
```

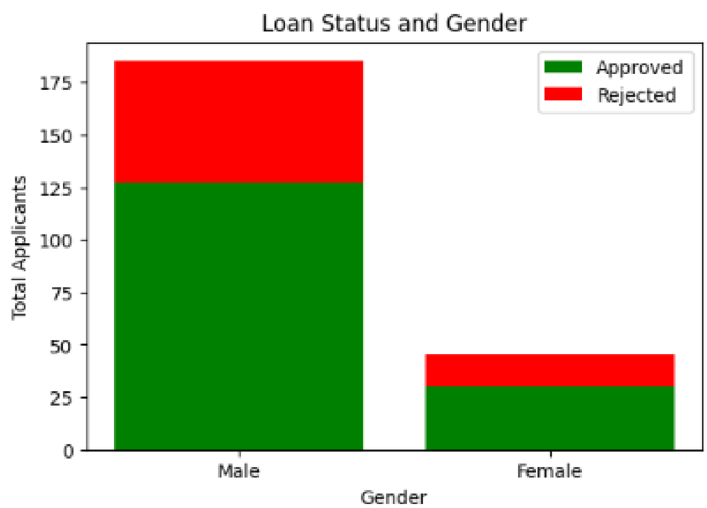


Mentioned Exploratory Data Analysis

```
#total amount loaned
total_amount_loaned = df_loan_data['LoanAmount'].sum()
print("Total amount loaned by AFS:", total_amount_loaned)

# average loaned amount
average_loan_amount = df_loan_data['LoanAmount'].mean()
print("Average amount loaned:", average_loan_amount)

# average term loaned
average_loan_term = df_loan_data['Loan_Amount_Term'].mean()
print("Average of loan term:", average_loan_term)

# Approved and Rejected
approved_applicants = df_loan_data[df_loan_data['Loan_Status'] == 'Y']
rejected_applicants = df_loan_data[df_loan_data['Loan_Status'] == 'N']
#let 1 for male and 2 for female
approved_gender_count = approved_applicants['Gender'].map({1: 'Male', 2: 'Female'}).value_counts()
rejected_gender_count = rejected_applicants['Gender'].map({1: 'Male', 2: 'Female'}).value_counts()

plt.figure(figsize=(6, 4))
plt.bar(approved_gender_count.index, approved_gender_count.values, label='Approved', color='green')
plt.bar(rejected_gender_count.index, rejected_gender_count.values, bottom=approved_gender_count.values, label='Rejected', color='red')
plt.xlabel('Gender')
plt.ylabel('Total Applicants')
plt.title('Loan Status and Gender')
plt.legend()
plt.show()
```

```
Total amount loaned by AFS: 32837
Average amount loaned: 142.76956521739132
Average of loan term: 325.68260869565216
```

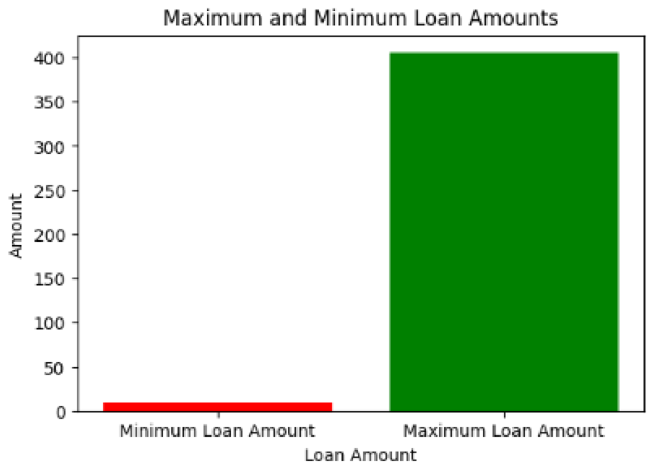

Average amount loaned: 142.76956521739132
Average of loan term: 325.68260869565216




```
[ ] # minimum and maximum loan
max_loan_amount = df_loan_data['LoanAmount'].max()
min_loan_amount = df_loan_data['LoanAmount'].min()
print("Maximum Loan Amount:", max_loan_amount)
print("Minimum Loan Amount:", min_loan_amount)

# plot loan amount
loan_amounts = [min_loan_amount, max_loan_amount]
labels = ['Minimum Loan Amount', 'Maximum Loan Amount']

plt.figure(figsize=(6, 4))
plt.bar(labels, loan_amounts, color=['red', 'green'])
plt.xlabel('Loan Amount')
plt.ylabel('Amount')
```

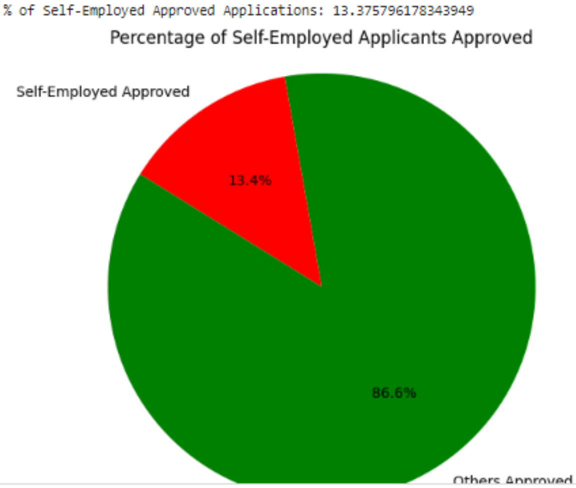
```python
print("Minimum Loan Amount:", min_loan_amount)

# plot loan amount
loan_amounts = [min_loan_amount, max_loan_amount]
labels = ['Minimum Loan Amount', 'Maximum Loan Amount']

plt.figure(figsize=(6, 4))
plt.bar(labels, loan_amounts, color=['red', 'green'])
plt.xlabel('Loan Amount')
plt.ylabel('Amount')
plt.title('Maximum and Minimum Loan Amounts')
plt.show()
```

```
Maximum Loan Amount: 405
Minimum Loan Amount: 9
```




```python
print(" % of Self-Employed Approved Applications:", percentage_self_employed_approved)

# ploting
labels = ['Self-Employed Approved', 'Others Approved']
sizes = [percentage_self_employed_approved, 100 - percentage_self_employed_approved]
colors = ['red', 'green']
plt.figure(figsize=(8, 6))
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=100)
plt.axis('equal')
plt.title('Percentage of Self-Employed Applicants Approved')
plt.show()
```

```
 % of Self-Employed Approved Applications: 13.375796178343949
```



```
# All main applicants income
income_stats = df_loan_data['ApplicantIncome'].describe()
average_income = income_stats['mean']
std_dev_income = income_stats['std']
print("Average Income:", average_income)
print("Standard Deviation of Income:", std_dev_income)
```

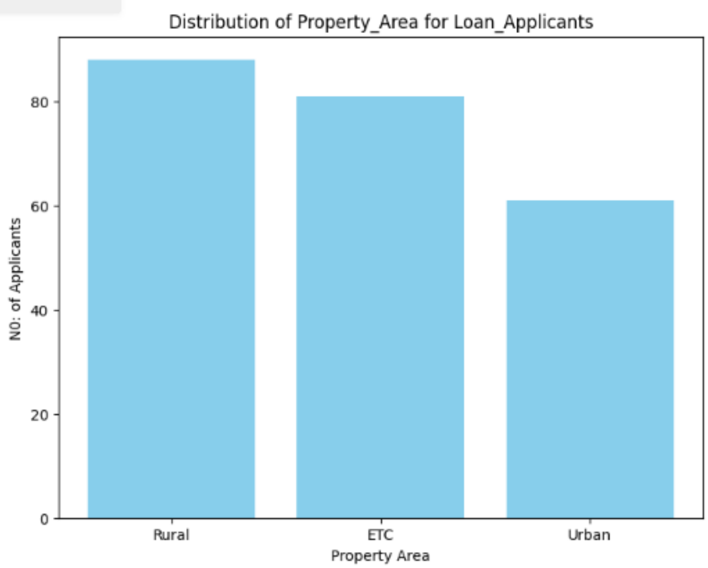
```
Average Income: 4772.643478260869
Standard Deviation of Income: 3423.59958477604
```

```
# top 10 application
top_ten_applicants = df_loan_data.nlargest(10, 'LoanAmount')[['Loan_ID', 'LoanAmount']]
print("\nTop Ten Applicants by Loan Amount:")
print(top_ten_applicants)
```

```
Top Ten Applicants by Loan Amount:
     Loan_ID  LoanAmount
65      2386         405
158     2699         400
20      1922         333
41      1990         333
68      2393         333
70      2401         333
112     2533         333
157     2697         333
183     2778         333
184     2784         333
```

```
# Distribution  properties: all loan applicants
property_distribution = df_loan_data['Property_Area'].map({1: 'Urban', 2: 'Rural', 3: 'ETC'}).value_counts()
plt.figure(figsize=(8, 6))
plt.bar(property_distribution.index, property_distribution.values, color='skyblue')
plt.xlabel('Property Area')
plt.ylabel('N0: of Applicants')
plt.title('Distribution of Property_Area for Loan_Applicants')
plt.show()
```


```
plt.title('Distribution of Property_Area for Loan_Applicants')
plt.show()
```

Appendix 2

```
pdf_path = '/content/drive/MyDrive/APEX_Loans_Database_Table.pdf'
excel_path = '/content/drive/MyDrive/APEX Loan Data.xlsx'
```

```
# loan databe
df_loan_data = pd.read_excel(excel_path)
df_loan_data.head()
```

	Loan_ID	Gender	Married	Dependents	Graduate	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	2284	1	0	0	0	0	3902	1666.0	109	333	1	3	Y
1	2287	2	0	0	1	0	1500	1800.0	103	333	0	2	N
2	2288	1	1	2	0	0	2889	0.0	45	180	0	1	N
3	2296	1	0	0	0	0	2755	0.0	65	300	1	3	N
4	2297	1	0	0	1	0	2500	20000.0	103	333	1	2	Y

Appendix 3

```
# Load Data
import pandas as pd
loan_data = pd.read_excel('loan_data.xlsx')

# Data Cleaning
loan_data_cleaned = loan_data.dropna()
loan_data_cleaned = loan_data_cleaned.drop_duplicates()
loan_data_cleaned = loan_data_cleaned[(np.abs(stats.zscore(loan_data_cleaned.select_dtypes(include=['number']))) < 3).all(axis=1)]

# Feature Engineering
# new features
loan_data_cleaned['TotalIncome'] = loan_data_cleaned['ApplicantIncome'] + loan_data_cleaned['CoapplicantIncome']

# Data Analysis & Prediction model
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
X = loan_data_cleaned[['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term']]
y = loan_data_cleaned['Loan_Status']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LogisticRegression()
model.fit(X_train, y_train)
predictions = model.predict(X_test)

# Evaluation
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, predictions)
print("Accuracy:", accuracy)
```