Task 2

Credit / Home Loans - AutoML vs Bespoke ML

Standard Bank is embracing the digital transformation wave and intends to use new and exciting technologies to give their customers a complete set of services from the convenience of their mobile devices. As Africa's biggest lender by assets, the bank aims to improve the current process in which potential borrowers apply for a home loan. The current process involves loan officers having to manually process home loan applications. This process takes 2 to 3 days to process upon which the applicant will receive communication on whether or not they have been granted the loan for the requested amount. To improve the process Standard Bank wants to make use of machine learning to assess the credit worthiness of an applicant by implementing a model that will predict if the potential borrower will default on his/her loan or not, and do this such that the applicant receives a response immediately after completing their application.

You will be required to follow the data science lifecycle to fulfill the objective. The data science lifecycle (https://www.datascience-pm.com/crisp-dm-2/) includes:

- Business Understanding
- Data Understanding
- Data Preparation
- Modelling
- Evaluation
- Deployment.

You now know the CRoss Industry Standard Process for Data Mining (CRISP-DM), have an idea of the business needs and objectivess, and understand the data. Next is the tedious task of preparing the data for modeling, modeling and evaluating the model. Luckily, just like EDA the first of the two phases can be automated. But also, just like EDA this is not always best.

In this task you will be get a taste of AutoML and Bespoke ML. In the notebook we make use of the library auto-sklearn/autosklearn (https://www.automl.org/automl/auto-sklearn/) for AutoML and sklearn for ML. We will use train one machine for the traditional approach and you will be required to change this model to any of the models that exist in sklearn. The model we will train will be a Logistic Regression. Parts of the data preparation will be omitted for you to do, but we will provide hints to lead you in the right direction.

The data provided can be found in the Resources folder as well as (https://www.kaggle.com/datasets/altruistdelhite04/loan-prediction-problem-dataset).

- train will serve as the historical dataset that the model will be trained on and,
- test will serve as unseen data we will predict on, i.e. new ('future') applicants.

Part One

There are many AutoEDA Python libraries out there which include:

- dtale (https://dtale.readthedocs.io/en/latest/)
- pandas profiling (https://pandas-profiling.ydata.ai/docs/master/index.html)
- autoviz (https://readthedocs.org/projects/autoviz/)
- sweetviz (https://pypi.org/project/sweetviz/)

and many more. In this task we will use Sweetviz.. You may be required to use bespoke EDA methods.

The Home Loans Department manager wants to know the following:

1. An overview of the data. (HINT: Provide the number of records, fields and their data types. Do for both).
2. What data quality issues exist in both train and test? (HINT: Comment any missing values and duplicates)
3. How do the the loan statuses compare? i.e. what is the distrubition of each?
4. How many of the loan applicants have dependents based on the historical dataset?
5. How do the incomes of those who are employed compare to those who are self employed based on the historical dataset?

historical dataset?

6. Are applicants with a credit history more likely to default than those who do not have one?
7. Is there a correlation between the applicant's income and the loan amount they applied for?

Part Two

Run the AutoML section and then fill in code for the traditional ML section for the the omitted cells.

Please note that the notebook you submit must include the analysis you did in Task 2.

Import Libraries

In [2]:

```
!pip install tpot
```

```
Requirement already satisfied: tpot in c:\users\rhydh\anaconda3\lib\site-packages (0.12.2
)
Requirement already satisfied: numpy>=1.16.3 in c:\users\rhydh\anaconda3\lib\site-package
s (from tpot) (1.26.4)
Requirement already satisfied: scipy>=1.3.1 in c:\users\rhydh\anaconda3\lib\site-packages
(from tpot) (1.13.1)
Requirement already satisfied: scikit-learn>=1.4.1 in c:\users\rhydh\anaconda3\lib\site-p
ackages (from tpot) (1.5.0)
Requirement already satisfied: deap>=1.2 in c:\users\rhydh\anaconda3\lib\site-packages (f
rom tpot) (1.4.1)
Requirement already satisfied: update-checker>=0.16 in c:\users\rhydh\anaconda3\lib\site-
packages (from tpot) (0.18.0)
Requirement already satisfied: tqdm>=4.36.1 in c:\users\rhydh\anaconda3\lib\site-packages
(from tpot) (4.65.0)
Requirement already satisfied: stopit>=1.1.1 in c:\users\rhydh\anaconda3\lib\site-package
s (from tpot) (1.1.2)
Requirement already satisfied: pandas>=0.24.2 in c:\users\rhydh\anaconda3\lib\site-packag
es (from tpot) (2.1.4)
Requirement already satisfied: joblib>=0.13.2 in c:\users\rhydh\anaconda3\lib\site-packag
es (from tpot) (1.2.0)
Requirement already satisfied: xgboost>=1.1.0 in c:\users\rhydh\anaconda3\lib\site-packag
es (from tpot) (2.0.3)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\rhydh\anaconda3\lib\sit
e-packages (from pandas>=0.24.2->tpot) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\rhydh\anaconda3\lib\site-packages
(from pandas>=0.24.2->tpot) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in c:\users\rhydh\anaconda3\lib\site-packag
es (from pandas>=0.24.2->tpot) (2023.3)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\rhydh\anaconda3\lib\site-
packages (from scikit-learn>=1.4.1->tpot) (3.5.0)
Requirement already satisfied: colorama in c:\users\rhydh\anaconda3\lib\site-packages (fr
om tqdm>=4.36.1->tpot) (0.4.6)
Requirement already satisfied: requests>=2.3.0 in c:\users\rhydh\anaconda3\lib\site-packa
ges (from update-checker>=0.16->tpot) (2.31.0)
Requirement already satisfied: six>=1.5 in c:\users\rhydh\anaconda3\lib\site-packages (fr
om python-dateutil>=2.8.2->pandas>=0.24.2->tpot) (1.16.0)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\rhydh\anaconda3\lib\s
ite-packages (from requests>=2.3.0->update-checker>=0.16->tpot) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\rhydh\anaconda3\lib\site-packages
(from requests>=2.3.0->update-checker>=0.16->tpot) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\rhydh\anaconda3\lib\site-pa
ckages (from requests>=2.3.0->update-checker>=0.16->tpot) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\rhydh\anaconda3\lib\site-pa
ckages (from requests>=2.3.0->update-checker>=0.16->tpot) (2024.2.2)
```

In [3]:

```
!pip install --upgrade scipy
```

```
Requirement already satisfied: scipy in c:\users\rhydh\anaconda3\lib\site-packages (1.13.
1)
Requirement already satisfied: numpy<2.3,>=1.22.4 in c:\users\rhydh\anaconda3\lib\site-pa
ckages (from scipy) (1.26.4)
```

In [4]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sweetviz
from tpot import TPOTClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.impute import SimpleImputer
```

```
C:\Users\rhydh\anaconda3\Lib\site-packages\tpot\builtins\__init__.py:36: UserWarning: Warning: optional dependency `torch` is not available. - skipping import of NN models.
  warnings.warn("Warning: optional dependency `torch` is not available. - skipping import of NN models.")
```

Import Datasets

In [5]:

```
train = pd.read_csv('StandardBank train.csv')
test = pd.read_csv('StandardBank test.csv')
```

Part One

EDA

In [6]:

```
train.head()
```

Out[6]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Lo
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	

In [7]:

```
test.head()
```

Out[7]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Lo
0	LP001015	Male	Yes	0	Graduate	No	5720	0	110.0	
1	LP001022	Male	Yes	1	Graduate	No	3076	1500	126.0	
2	LP001031	Male	Yes	2	Graduate	No	5000	1800	208.0	
3	LP001035	Male	Yes	2	Graduate	No	2340	2546	100.0	
4	LP001051	Male	No	0	Not Graduate	No	3276	0	78.0	

In [8]:

```
# we concat for easy analysis
n = train.shape[0] # we set this to be able to separate the
df = pd.concat([train, test], axis=0)
df.head()
```

Out[8]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Lo
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	

Sweetviz

In [9]:

```
autoEDA = sweetviz.analyze(train)
autoEDA.show_notebook()
```

Your Own EDA

In [10]:

```
df.describe(include='all')
```

Out[10]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmou
count	981	957	978	956	981	926	981.000000	981.000000	954.00000
unique	981	2	2	4	2	2	NaN	NaN	Na
top	LP001002	Male	Yes	0	Graduate	No	NaN	NaN	Na
freq	1	775	631	545	763	807	NaN	NaN	Na
mean	NaN	NaN	NaN	NaN	NaN	NaN	5179.795107	1601.916330	142.51153
std	NaN	NaN	NaN	NaN	NaN	NaN	5695.104533	2718.772806	77.42174
min	NaN	NaN	NaN	NaN	NaN	NaN	0.000000	0.000000	9.00000
25%	NaN	NaN	NaN	NaN	NaN	NaN	2875.000000	0.000000	100.00000
50%	NaN	NaN	NaN	NaN	NaN	NaN	3800.000000	1110.000000	126.00000
75%	NaN	NaN	NaN	NaN	NaN	NaN	5516.000000	2365.000000	162.00000
max	NaN	NaN	NaN	NaN	NaN	NaN	81000.000000	41667.000000	700.00000

In [11]:

```
df.dtypes
```

Out[11]:

```
Loan_ID               object
Gender                object
Married               object
Dependents            object
Education             object
Self_Employed         object
ApplicantIncome        int64
CoapplicantIncome    float64
LoanAmount           float64
Loan_Amount_Term     float64
Credit_History       float64
Property_Area         object
Loan_Status           object
dtype: object
```

In [12]:

```
df.columns
```

Out[12]:

```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

In [13]:

```
train.isna().sum()
```

Out[13]:

```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

In [14]:

```
test.isna().sum()
```

Out[14]:

```
Loan_ID               0
Gender               11
Married               0
Dependents           10
Education             0
Self_Employed        23
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            5
Loan_Amount_Term      6
Credit_History       29
Property_Area         0
dtype: int64
```

In [15]:

```
# Convert float values to 1 or 0
train['Credit_History'] = train['Credit_History'].apply(lambda x: '1' if x >= 1 else '0'
)

# Convert dtype to object
train['Credit_History'] = train['Credit_History'].astype('object')

# Display the modified DataFrame
print(train)
```

```
      Loan_ID  Gender Married Dependents     Education Self_Employed  \
0    LP001002    Male      No          0      Graduate            No
1    LP001003    Male     Yes          1      Graduate            No
2    LP001005    Male     Yes          0      Graduate           Yes
3    LP001006    Male     Yes          0  Not Graduate            No
4    LP001008    Male      No          0      Graduate            No
..        ...     ...     ...        ...           ...           ...
609  LP002978  Female      No          0      Graduate            No
610  LP002979    Male     Yes         3+      Graduate            No
611  LP002983    Male     Yes          1      Graduate            No
612  LP002984    Male     Yes          2      Graduate            No
613  LP002990  Female      No          0      Graduate           Yes

     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0               5849                0.0         NaN             360.0
1               4583             1508.0       128.0             360.0
2               3000                0.0        66.0             360.0
3               2583             2358.0       120.0             360.0
4               6000                0.0       141.0             360.0
..               ...                ...         ...               ...
609             2900                0.0        71.0             360.0
610             4106                0.0        40.0             180.0
611             8072              240.0       253.0             360.0
```

```
612             7583                0.0       187.0          360.0
613             4583                0.0       133.0          360.0

    Credit_History Property_Area Loan_Status
0                1         Urban           Y
1                1         Rural           N
2                1         Urban           Y
3                1         Urban           Y
4                1         Urban           Y
..             ...           ...         ...
609              1         Rural           Y
610              1         Rural           Y
611              1         Urban           Y
612              1         Urban           Y
613              0     Semiurban           N

[614 rows x 13 columns]
```

In [16]:

```
# Convert float values to 1 or 0
test['Credit_History'] = test['Credit_History'].apply(lambda x: '1' if x >= 1 else '0')

# Convert dtype to object
test['Credit_History'] = test['Credit_History'].astype('object')

# Display the modified DataFrame
print(test)
```

```
      Loan_ID Gender Married Dependents     Education Self_Employed  \
0    LP001015   Male     Yes          0      Graduate            No
1    LP001022   Male     Yes          1      Graduate            No
2    LP001031   Male     Yes          2      Graduate            No
3    LP001035   Male     Yes          2      Graduate            No
4    LP001051   Male      No          0  Not Graduate            No
..        ...    ...     ...        ...           ...           ...
362  LP002971   Male     Yes         3+  Not Graduate           Yes
363  LP002975   Male     Yes          0      Graduate            No
364  LP002980   Male      No          0      Graduate            No
365  LP002986   Male     Yes          0      Graduate            No
366  LP002989   Male      No          0      Graduate           Yes

     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0               5720                  0       110.0             360.0
1               3076               1500       126.0             360.0
2               5000               1800       208.0             360.0
3               2340               2546       100.0             360.0
4               3276                  0        78.0             360.0
..               ...                ...         ...               ...
362             4009               1777       113.0             360.0
363             4158                709       115.0             360.0
364             3250               1993       126.0             360.0
365             5000               2393       158.0             360.0
366             9200                  0        98.0             180.0

    Credit_History Property_Area
0                1         Urban
1                1         Urban
2                1         Urban
3                0         Urban
4                1         Urban
..             ...           ...
362              1         Urban
363              1         Urban
364              0     Semiurban
365              1         Rural
366              1         Rural

[367 rows x 12 columns]
```

In [17]:

```
# Initialize SimpleImputer for numeric columns
numeric_imputer = SimpleImputer(strategy='mean')

# Initialize SimpleImputer for object columns
object_imputer = SimpleImputer(strategy='most_frequent')

# Loop through columns
for col in train.columns:
    if train[col].dtype == 'object':
        # Impute object columns with most frequent value
        train[col] = object_imputer.fit_transform(train[[col]])[:, 0]  # Extracting the
imputed values from the 2D array
    elif train[col].dtype in ['int64', 'float64']:  # Adjust as per your specific numeri
c types
        # Impute numeric columns with mean
        train[col] = numeric_imputer.fit_transform(train[[col]])[:, 0]  # Extracting the
imputed values from the 2D array

# Display the imputed DataFrame
print(train)
```

```
      Loan_ID  Gender Married Dependents     Education Self_Employed  \
0    LP001002    Male      No          0      Graduate            No
1    LP001003    Male     Yes          1      Graduate            No
2    LP001005    Male     Yes          0      Graduate           Yes
3    LP001006    Male     Yes          0  Not Graduate            No
4    LP001008    Male      No          0      Graduate            No
..        ...     ...     ...        ...           ...           ...
609  LP002978  Female      No          0      Graduate            No
610  LP002979    Male     Yes         3+      Graduate            No
611  LP002983    Male     Yes          1      Graduate            No
612  LP002984    Male     Yes          2      Graduate            No
613  LP002990  Female      No          0      Graduate           Yes

     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             5849.0                0.0  146.412162             360.0
1             4583.0             1508.0  128.000000             360.0
2             3000.0                0.0   66.000000             360.0
3             2583.0             2358.0  120.000000             360.0
4             6000.0                0.0  141.000000             360.0
..               ...                ...         ...               ...
609           2900.0                0.0   71.000000             360.0
610           4106.0                0.0   40.000000             180.0
611           8072.0              240.0  253.000000             360.0
612           7583.0                0.0  187.000000             360.0
613           4583.0                0.0  133.000000             360.0

    Credit_History Property_Area Loan_Status
0                1         Urban           Y
1                1         Rural           N
2                1         Urban           Y
3                1         Urban           Y
4                1         Urban           Y
..             ...           ...         ...
609              1         Rural           Y
610              1         Rural           Y
611              1         Urban           Y
612              1         Urban           Y
613              0     Semiurban           N

[614 rows x 13 columns]
```

In [18]:

```
# Loop through columns
for col in test.columns:
    if test[col].dtype == 'object':
        # Impute object columns with most frequent value
        test[col] = object_imputer.fit_transform(test[[col]])[:, 0]  # Extracting the imp
uted values from the 2D array
    elif test[col].dtype in ['int64', 'float64']:  # Adjust as per your specific numeric
types
```

```
        # Impute numeric columns with mean
        test[col] = numeric_imputer.fit_transform(test[[col]])[:, 0]  # Extracting the im
puted values from the 2D array

# Display the imputed DataFrame
print(test)
```

```
      Loan_ID Gender Married Dependents     Education Self_Employed  \
0    LP001015   Male     Yes          0      Graduate            No   
1    LP001022   Male     Yes          1      Graduate            No   
2    LP001031   Male     Yes          2      Graduate            No   
3    LP001035   Male     Yes          2      Graduate            No   
4    LP001051   Male      No          0  Not Graduate            No   
..        ...    ...     ...        ...           ...           ...   
362  LP002971   Male     Yes         3+  Not Graduate           Yes   
363  LP002975   Male     Yes          0      Graduate            No   
364  LP002980   Male      No          0      Graduate            No   
365  LP002986   Male     Yes          0      Graduate            No   
366  LP002989   Male      No          0      Graduate           Yes   

     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             5720.0                0.0       110.0             360.0   
1             3076.0             1500.0       126.0             360.0   
2             5000.0             1800.0       208.0             360.0   
3             2340.0             2546.0       100.0             360.0   
4             3276.0                0.0        78.0             360.0   
..               ...                ...         ...               ...   
362           4009.0             1777.0       113.0             360.0   
363           4158.0              709.0       115.0             360.0   
364           3250.0             1993.0       126.0             360.0   
365           5000.0             2393.0       158.0             360.0   
366           9200.0                0.0        98.0             180.0   

    Credit_History Property_Area  
0                1         Urban  
1                1         Urban  
2                1         Urban  
3                0         Urban  
4                1         Urban  
..             ...           ...  
362              1         Urban  
363              1         Urban  
364              0     Semiurban  
365              1         Rural  
366              1         Rural  

[367 rows x 12 columns]
```

In [19]:

```
train['Credit_History'] = train['Credit_History'].astype('int64')
test['Credit_History'] = test['Credit_History'].astype('int64')
```

In [20]:

```
train['Dependents'] = train['Dependents'].replace({'3+':3, '2':2, '1': 1, '0': 0}).astyp
e('int64')
```

In [21]:

```
test['Dependents'] = test['Dependents'].replace({'3+':3, '2':2, '1': 1, '0': 0}).astype(
'int64')
```

In [22]:

```
train['Dependents']
```

Out[22]:

```
0      0
1      1
2      0
```

```
3      0
4      0
      ..
609    0
610    3
611    1
612    2
613    0
Name: Dependents, Length: 614, dtype: int64
```

In [23]:

```
train['Loan_Status'].value_counts()
```

Out[23]:

```
Loan_Status
Y    422
N    192
Name: count, dtype: int64
```

In [24]:

```
train[train['Dependents']>0].value_counts().sum()
```

Out[24]:

```
254
```

In [25]:

```
# Convert 'Yes' and 'No' to 1 and 0 in Self_Employed column
train['Self_Employed'] = train['Self_Employed'].replace({'Yes': 1, 'No': 0})

# Separate data for employed and self-employed individuals
employed_data = train[train['Self_Employed'] == 0]
self_employed_data = train[train['Self_Employed'] == 1]

# Calculate summary statistics for each group
employed_mean_income = employed_data['ApplicantIncome'].mean()
self_employed_mean_income = self_employed_data['ApplicantIncome'].mean()

employed_median_income = employed_data['ApplicantIncome'].median()
self_employed_median_income = self_employed_data['ApplicantIncome'].median()

print(f"Employed Mean Income: {employed_mean_income}")
print(f"Self-Employed Mean Income: {self_employed_mean_income}")

print(f"Employed Median Income: {employed_median_income}")
print(f"Self-Employed Median Income: {self_employed_median_income}")
```

```
Employed Mean Income: 5098.678571428572
Self-Employed Mean Income: 7380.817073170731
Employed Median Income: 3698.0
Self-Employed Median Income: 5809.0
```

In [26]:

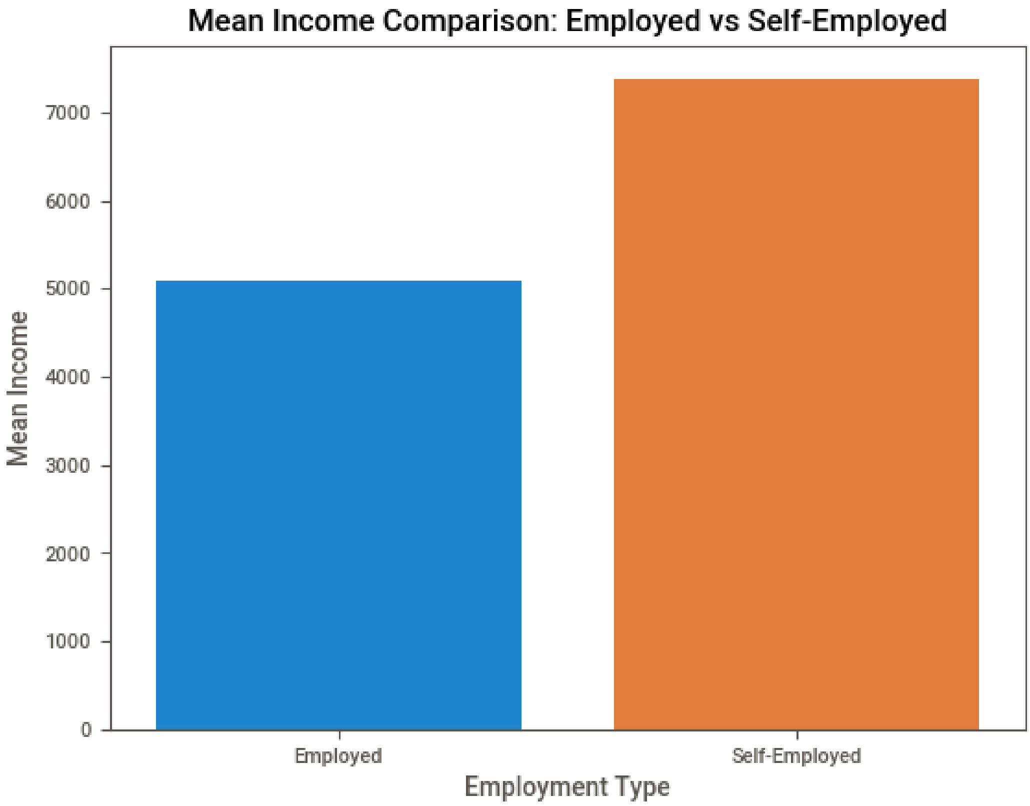
```
import seaborn as sns
import matplotlib.pyplot as plt

# Plotting with Seaborn
sns.barplot(x=['Employed', 'Self-Employed'], y=[employed_mean_income, self_employed_mean
_income])
plt.xlabel('Employment Type')
plt.ylabel('Mean Income')
plt.title('Mean Income Comparison: Employed vs Self-Employed')
plt.show()

#Self-Employed People earn more in terms of average (mean) income
```

```
C:\Users\rhydh\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1765: FutureWarning: uniqu
e with argument that is not not a Series, Index, ExtensionArray, or np.ndarray is depreca
```

```
e with argument that is not not a Series, Index, ExtensionArray, or np.ndarray is depreca
ted and will raise in a future version.
  order = pd.unique(vector)
```



In [27]:

```
train[['Credit_History','Loan_Status']].value_counts()
```

Out[27]:

```
Credit_History  Loan_Status
1               Y              378
                N               97
0               N               95
                Y               44
Name: count, dtype: int64
```

In [28]:

```
from scipy.stats import chi2_contingency

# Create the contingency table
observed = [
    [378, 97],
    [44, 95]
]

# Perform chi-square test
chi2, p, dof, expected = chi2_contingency(observed)

# Print results
print(f"Chi-square value: {chi2}")
print(f"P-value: {p}")
print(f"Degrees of freedom: {dof}")
print("Expected frequencies table:")
print(expected)

'''
If the p-value is greater than or equal to your significance level,
you fail to reject the null hypothesis, indicating no significant association between Cre
dit_History and Loan_Status.
'''
```

```
Chi-square value: 112.69526773505117
P-value: 2.516222405860599e-26
Degrees of freedom: 1
```

```
Degrees of freedom: 1
Expected frequencies table:
[[326.46579805 148.53420195]
 [ 95.53420195  43.46579805]]
```

Out[28]:

```
'\nIf the p-value is greater than or equal to your significance level, \nyou fail to reje
ct the null hypothesis, indicating no significant association between Credit_History and
Loan_Status.\n'
```

In [29]:

```
merged_df = pd.concat([train, test], axis=0)
merged_df.head()
```

Out[29]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Lo
0	LP001002	Male	No	0	Graduate	0	5849.0	0.0	146.412162	
1	LP001003	Male	Yes	1	Graduate	0	4583.0	1508.0	128.000000	
2	LP001005	Male	Yes	0	Graduate	1	3000.0	0.0	66.000000	
3	LP001006	Male	Yes	0	Not Graduate	0	2583.0	2358.0	120.000000	
4	LP001008	Male	No	0	Graduate	0	6000.0	0.0	141.000000	

In [30]:

```
merged_df[['ApplicantIncome','LoanAmount']].corr()

#so there is 55% +ve correlation
```

Out[30]:

	ApplicantIncome	LoanAmount
ApplicantIncome	1.000000	0.547036
LoanAmount	0.547036	1.000000

Your anwers:

1. **Overview done using .describe(include='all')**
2. **Imputed missing values using SimpleImputer (mean for float/int 64 dtype and most_frequent for object dtype)**
3. **422 approved loans vs 192 rejected**
4. **254 loan applicants have dependents**
5. **On an average, Self-Employed people earn more than Employed people**
6. **No significant association between Credit_History and Loan_Status**
7. **there is 55% +ve correlation between 'ApplicantIncome' & 'LoanAmount'**

In [31]:

```
train.dtypes
```

Out[31]:

```
Loan_ID               object
Gender                object
Married               object
Dependents             int64
Education             object
Self_Employed          int64
ApplicantIncome      float64
CoapplicantIncome    float64
```

```
LoanAmount           float64
Loan_Amount_Term     float64
Credit_History         int64
Property_Area         object
Loan_Status           object
dtype: object
```

In [32]:

```
from sklearn.preprocessing import LabelEncoder

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Iterate over each column
for col in train.columns:
    if train[col].dtype == 'object':  # Check if the column is of object type
        train[col] = label_encoder.fit_transform(train[col].astype(str))

# Display the encoded DataFrame
print(train)
```

```
     Loan_ID  Gender  Married  Dependents  Education  Self_Employed  \
0          0       1        0           0          0              0   
1          1       1        1           1          0              0   
2          2       1        1           0          0              1   
3          3       1        1           0          1              0   
4          4       1        0           0          0              0   
..       ...     ...      ...         ...        ...            ...   
609      609       0        0           0          0              0   
610      610       1        1           3          0              0   
611      611       1        1           1          0              0   
612      612       1        1           2          0              0   
613      613       0        0           0          0              1   

     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             5849.0                0.0  146.412162             360.0   
1             4583.0             1508.0  128.000000             360.0   
2             3000.0                0.0   66.000000             360.0   
3             2583.0             2358.0  120.000000             360.0   
4             6000.0                0.0  141.000000             360.0   
..               ...                ...         ...               ...   
609           2900.0                0.0   71.000000             360.0   
610           4106.0                0.0   40.000000             180.0   
611           8072.0              240.0  253.000000             360.0   
612           7583.0                0.0  187.000000             360.0   
613           4583.0                0.0  133.000000             360.0   

     Credit_History  Property_Area  Loan_Status  
0                 1              2            1  
1                 1              0            0  
2                 1              2            1  
3                 1              2            1  
4                 1              2            1  
..              ...            ...          ...  
609               1              0            1  
610               1              0            1  
611               1              2            1  
612               1              2            1  
613               0              1            0  

[614 rows x 13 columns]
```

In [33]:

```
from sklearn.preprocessing import LabelEncoder

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Iterate over each column
```

```
for col in test.columns:
    if test[col].dtype == 'object':  # Check if the column is of object type
        test[col] = label_encoder.fit_transform(test[col].astype(str))

# Display the encoded DataFrame
print(test)
```

```
     Loan_ID  Gender  Married  Dependents  Education  Self_Employed  \
0          0       1        1           0          0              0
1          1       1        1           1          0              0
2          2       1        1           2          0              0
3          3       1        1           2          0              0
4          4       1        0           0          1              0
..       ...     ...      ...         ...        ...            ...
362      362       1        1           3          1              1
363      363       1        1           0          0              0
364      364       1        0           0          0              0
365      365       1        1           0          0              0
366      366       1        0           0          0              1

     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             5720.0                0.0       110.0             360.0
1             3076.0             1500.0       126.0             360.0
2             5000.0             1800.0       208.0             360.0
3             2340.0             2546.0       100.0             360.0
4             3276.0                0.0        78.0             360.0
..               ...                ...         ...               ...
362           4009.0             1777.0       113.0             360.0
363           4158.0              709.0       115.0             360.0
364           3250.0             1993.0       126.0             360.0
365           5000.0             2393.0       158.0             360.0
366           9200.0                0.0        98.0             180.0

     Credit_History  Property_Area
0                 1              2
1                 1              2
2                 1              2
3                 0              2
4                 1              2
..              ...            ...
362               1              2
363               1              2
364               0              1
365               1              0
366               1              0

[367 rows x 12 columns]
```

In [34]:

```
train.dtypes    #Finally every feature is in numeric format
```

Out[34]:

```
Loan_ID                int32
Gender                 int32
Married                int32
Dependents             int64
Education              int32
Self_Employed          int64
ApplicantIncome      float64
CoapplicantIncome    float64
LoanAmount           float64
Loan_Amount_Term     float64
Credit_History         int64
Property_Area          int32
Loan_Status            int32
dtype: object
```

Part Two

Auto ML wth tpot

In [35]:

```
# Matrix of features

X = train[['Gender',
'Married',
'Dependents',
'Education',
'Self_Employed',
'ApplicantIncome',
'CoapplicantIncome',
'LoanAmount',
'Loan_Amount_Term',
'Credit_History',
'Property_Area']]


# label encode target
y = train['Loan_Status']


# # train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
)
```

In [37]:

```
# train
autoML = TPOTClassifier(generations=5, population_size=50, verbosity=2, n_jobs=-1)
autoML.fit(X_train, y_train)

# predict
predictions_autoML = autoML.predict(X_test)
```

```
Generation 1 - Current best internal CV score: 0.7697175840032981

Generation 2 - Current best internal CV score: 0.7697175840032981

Generation 3 - Current best internal CV score: 0.7697175840032981

Generation 4 - Current best internal CV score: 0.7697175840032981

Generation 5 - Current best internal CV score: 0.7697175840032981

Best pipeline: ExtraTreesClassifier(input_matrix, bootstrap=True, criterion=entropy, max_
features=0.8500000000000001, min_samples_leaf=20, min_samples_split=13, n_estimators=100)
```

In [39]:

```
print('Model Accuracy:', accuracy_score(predictions_autoML, y_test))
```

```
Model Accuracy: 0.7723577235772358
```

In [40]:

```
print(confusion_matrix(predictions_autoML, y_test))
```

```
[[22  7]
 [21 73]]
```

Bespoke ML sklearn

Data Preparation

In [41]:

In [41]:

```
# Matrix of features

X = train[['Gender',
'Married',
'Dependents',
'Education',
'Self_Employed',
'ApplicantIncome',
'CoapplicantIncome',
'LoanAmount',
'Loan_Amount_Term',
'Credit_History',
'Property_Area']]

# label encode target
y = train['Loan_Status']

# # train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
)
```

In [42]:

```
# some classifiers you can pick from (remember to import)
import sklearn
classifiers = sklearn.utils.all_estimators(type_filter=None)
for name, class_ in classifiers:
    if hasattr(class_, 'predict_proba'):
        print(name)
```

```
AdaBoostClassifier
BaggingClassifier
BayesianGaussianMixture
BernoulliNB
CalibratedClassifierCV
CategoricalNB
ClassifierChain
ComplementNB
DecisionTreeClassifier
DummyClassifier
ExtraTreeClassifier
ExtraTreesClassifier
FixedThresholdClassifier
GaussianMixture
GaussianNB
GaussianProcessClassifier
GradientBoostingClassifier
GridSearchCV
HistGradientBoostingClassifier
KNeighborsClassifier
LabelPropagation
LabelSpreading
LinearDiscriminantAnalysis
LogisticRegression
LogisticRegressionCV
MLPClassifier
MultiOutputClassifier
MultinomialNB
NuSVC
OneVsRestClassifier
Pipeline
QuadraticDiscriminantAnalysis
RFE
RFECV
RadiusNeighborsClassifier
RandomForestClassifier
RandomizedSearchCV
SGDClassifier
SVC
SelfTrainingClassifier
StackingClassifier
```

TunedThresholdClassifierCV
VotingClassifier

In [48]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import AdaBoostClassifier, BaggingClassifier, RandomForestClassifie
r, ExtraTreesClassifier, GradientBoostingClassifier, HistGradientBoostingClassifier, Stac
kingClassifier, VotingClassifier
from sklearn.naive_bayes import BernoulliNB, CategoricalNB, ComplementNB, GaussianNB, Mul
tinomialNB
from sklearn.mixture import BayesianGaussianMixture, GaussianMixture
from sklearn.tree import DecisionTreeClassifier, ExtraTreeClassifier
from sklearn.dummy import DummyClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.svm import SVC, NuSVC
from sklearn.neighbors import KNeighborsClassifier, RadiusNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis, QuadraticDiscrimina
ntAnalysis
from sklearn.multioutput import MultiOutputClassifier, ClassifierChain
from sklearn.semi_supervised import LabelPropagation, LabelSpreading, SelfTrainingClassif
ier
from sklearn.feature_selection import RFE, RFECV
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

# List of models to evaluate
classifiers = {
    "LogisticRegression": LogisticRegression(),
    "AdaBoostClassifier": AdaBoostClassifier(),
    "BaggingClassifier": BaggingClassifier(),
    "RandomForestClassifier": RandomForestClassifier(),
    "ExtraTreesClassifier": ExtraTreesClassifier(),
    "GradientBoostingClassifier": GradientBoostingClassifier(),
    "HistGradientBoostingClassifier": HistGradientBoostingClassifier(),
    "BernoulliNB": BernoulliNB(),
    "GaussianNB": GaussianNB(),
    "BayesianGaussianMixture": BayesianGaussianMixture(),
    "GaussianMixture": GaussianMixture(),
    "DecisionTreeClassifier": DecisionTreeClassifier(),
    "ExtraTreeClassifier": ExtraTreeClassifier(),
    "DummyClassifier": DummyClassifier(),
    "SVC": SVC(),
    "NuSVC": NuSVC(),
    "KNeighborsClassifier": KNeighborsClassifier(),
    "MLPClassifier": MLPClassifier(),
    "GaussianProcessClassifier": GaussianProcessClassifier(),
    "LinearDiscriminantAnalysis": LinearDiscriminantAnalysis(),
    "QuadraticDiscriminantAnalysis": QuadraticDiscriminantAnalysis(),
    "SelfTrainingClassifier": SelfTrainingClassifier(LogisticRegression()),
    "RFE": RFE(LogisticRegression()),
    "RFECV": RFECV(LogisticRegression()),
    "StackingClassifier": StackingClassifier(estimators=[('lr', LogisticRegression()), ('
rf', RandomForestClassifier())]),
    "VotingClassifier": VotingClassifier(estimators=[('lr', LogisticRegression()), ('rf',
RandomForestClassifier())])
}

# Example data loading and splitting (replace with your actual dataset)
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
)

# Function to evaluate models
```

```python
def evaluate_models(classifiers, X_train, y_train, X_test, y_test):
    results = []
    for name, clf in classifiers.items():
        try:
            clf.fit(X_train, y_train)
            predictions = clf.predict(X_test)
            accuracy = accuracy_score(y_test, predictions)
            results.append((name, accuracy))
        except Exception as e:
            print(f"{name} failed to run: {e}")

    # Sort results by accuracy in descending order and get the top 3
    results.sort(key=lambda x: x[1], reverse=True)
    return results[:3]

# Evaluate and display top 3 models
top_models = evaluate_models(classifiers, X_train, y_train, X_test, y_test)

# Print the top 3 models
print("\nTop 3 Models:")
for name, accuracy in top_models:
    print(f"{name}: Accuracy = {accuracy:.4f}")
```

```
C:\Users\rhydh\anaconda3\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527: Futu
reWarning: The SAMME.R algorithm (the default) is deprecated and will be removed in 1.6.
Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
C:\Users\rhydh\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1426: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks tha
n available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS
=4.
  warnings.warn(
C:\Users\rhydh\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1426: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks tha
n available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS
=4.
  warnings.warn(
C:\Users\rhydh\anaconda3\Lib\site-packages\sklearn\neural_network\_multilayer_perceptron.
py:690: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and th
e optimization hasn't converged yet.
  warnings.warn(
C:\Users\rhydh\anaconda3\Lib\site-packages\sklearn\discriminant_analysis.py:949: UserWarn
ing: Variables are collinear
  warnings.warn("Variables are collinear")
C:\Users\rhydh\anaconda3\Lib\site-packages\sklearn\semi_supervised\_self_training.py:227:
UserWarning: y contains no unlabeled samples
  warnings.warn("y contains no unlabeled samples", UserWarning)
```

```
Top 3 Models:
HistGradientBoostingClassifier: Accuracy = 0.9150
GradientBoostingClassifier: Accuracy = 0.9100
BaggingClassifier: Accuracy = 0.8850
```

In [ ]:

```
'''
### Top 3 Models - Summary

1. **HistGradientBoostingClassifier**
   - **Accuracy**: 91.50%
   - **Description**: Optimized for large datasets using histograms to bin features, maki
ng training faster and more efficient.
   - **Interpretation**: Highest accuracy, indicating excellent class distinction and eff
icient memory management.

2. **GradientBoostingClassifier**
   - **Accuracy**: 91.00%
   - **Description**: Sequentially builds decision trees, correcting errors from previous
trees to improve performance.
   - **Interpretation**: High accuracy, slightly less than HistGradientBoosting due to la
ck of histogram binning but still highly effective.
```

```
3. **BaggingClassifier**
   - **Accuracy**: 88.50%
   - **Description**: Trains multiple models on different data subsets and combines predi
ctions, reducing variance and overfitting.
   - **Interpretation**: Robust performance with good accuracy, providing a stable basel
ine through ensemble learning.
'''
```

In [50]:

```
# Cross-validation

from sklearn.model_selection import cross_val_score

# Define the top models based on the previous results
top_classifiers = {
    "HistGradientBoostingClassifier": HistGradientBoostingClassifier(),
    "GradientBoostingClassifier": GradientBoostingClassifier(),
    "BaggingClassifier": BaggingClassifier()
}

# Number of cross-validation folds
cv_folds = 5

# Function to perform cross-validation and print results
def cross_validate_models(classifiers, X, y, cv_folds):
    results = []
    for name, clf in classifiers.items():
        try:
            # Perform cross-validation
            cv_scores = cross_val_score(clf, X, y, cv=cv_folds, scoring='accuracy')
            mean_accuracy = np.mean(cv_scores)
            std_dev = np.std(cv_scores)
            results.append((name, mean_accuracy, std_dev))
            print(f"{name}: Mean Accuracy = {mean_accuracy:.4f}, Std Dev = {std_dev:.4f}
")
        except Exception as e:
            print(f"{name} failed to run: {e}")

    # Sort results by mean accuracy in descending order
    results.sort(key=lambda x: x[1], reverse=True)
    return results

# Perform cross-validation and display results
cv_results = cross_validate_models(top_classifiers, X, y, cv_folds)

# Print the cross-validation results
print("\nCross-Validation Results:")
for name, mean_accuracy, std_dev in cv_results:
    print(f"{name}: Mean Accuracy = {mean_accuracy:.4f}, Std Dev = {std_dev:.4f}")
```

```
HistGradientBoostingClassifier: Mean Accuracy = 0.9090, Std Dev = 0.0218
GradientBoostingClassifier: Mean Accuracy = 0.9020, Std Dev = 0.0194
BaggingClassifier: Mean Accuracy = 0.8870, Std Dev = 0.0223

Cross-Validation Results:
HistGradientBoostingClassifier: Mean Accuracy = 0.9090, Std Dev = 0.0218
GradientBoostingClassifier: Mean Accuracy = 0.9020, Std Dev = 0.0194
BaggingClassifier: Mean Accuracy = 0.8870, Std Dev = 0.0223
```

In [ ]:

```
'''
Overall Insights:
HistGradientBoostingClassifier is the top-performing model with the highest mean accuracy
and a low standard deviation, indicating both high performance and stability.
GradientBoostingClassifier also performs very well, slightly below the HistGradientBoosti
ngClassifier, and has the lowest standard deviation, indicating extremely consistent perf
ormance.
BaggingClassifier performs well but is slightly behind the two boosting methods. It still
shows good accuracy and stability, making it a reliable choice as well.
```

```
'''
```