DATA INTENSIVE COMPUTING
CREDIT CARD APPROVAL PREDICTION

PROBLEM STATEMENT:

Credit card companies receive a large volume of applications every day, and the process of manually reviewing them can be time-consuming and lead to errors. To resolve this, machine learning algorithms can be applied to predict credit card approvals using historical data. The objective of this project is to develop a predictive model that accurately predicts whether a credit card application will be approved or rejected based on various features such as income, age, employment status, credit score, and other relevant factors.

BACKGROUND:

Credit cards are an essential aspect of the modern financial system, with millions of transactions taking place daily worldwide. With an increase in the number of credit card applicants, credit card issuers must go through a tedious process of manually evaluating applications. Credit card approval prediction involves evaluating the creditworthiness of applicants using machine learning algorithms. The process helps to reduce the manual effort of credit card issuers while providing a quick, efficient, and accurate way of approving or denying credit card applications.

SIGNIFICANCE:

Credit card approval prediction is a significant problem due to its potential impact on both the financial industry and consumers. For credit card issuers, the process of manually evaluating applications can be time-consuming, expensive, and prone to human error. Moreover, automated credit card approval prediction can help identify fraudulent applications and reduce the risk of credit card defaults. For consumers, credit card approval prediction can provide a more transparent way for obtaining credit.

CONTRIBUTION:

The development of an accurate credit card approval prediction model can significantly contribute to the financial industry by reducing the risk of credit card defaults and identifying fraudulent applications. It also improves the efficiency and speed of the credit card application process, leading to better customer satisfaction. The contribution of this project is crucial because it can reduce the risk of credit card issuers and provide a fair and efficient credit application process for consumers.

PHASE 1

DATA SOURCE:

we have gone through different websites for our problem and finally found a best suitable dataset from Kaggle.

CreditCardApprovalPrediction

Initial shape of the dataset

```
df_application = pd.read_csv(application_file)
df_credit = pd.read_csv(credit_file)
print(df_application.shape)
print(df_credit.shape)
```

```
(438557, 18)
(1048575, 3)
```

The Information about the datasets:

```
In [3]: df_application = pd.read_csv(application_file)
        df_credit = pd.read_csv(credit_file)

In [4]: df_application.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 438557 entries, 0 to 438556
Data columns (total 18 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   ID                    438557 non-null  int64
 1   CODE_GENDER           438557 non-null  object
 2   FLAG_OWN_CAR          438557 non-null  object
 3   FLAG_OWN_REALTY       438557 non-null  object
 4   CNT_CHILDREN          438557 non-null  int64
 5   AMT_INCOME_TOTAL      438557 non-null  float64
 6   NAME_INCOME_TYPE      438557 non-null  object
 7   NAME_EDUCATION_TYPE   438557 non-null  object
 8   NAME_FAMILY_STATUS    438557 non-null  object
 9   NAME_HOUSING_TYPE     438557 non-null  object
 10  DAYS_BIRTH            438557 non-null  int64
 11  DAYS_EMPLOYED         438557 non-null  int64
 12  FLAG_MOBIL            438557 non-null  int64
 13  FLAG_WORK_PHONE       438557 non-null  int64
 14  FLAG_PHONE            438557 non-null  int64
 15  FLAG_EMAIL            438557 non-null  int64
 16  OCCUPATION_TYPE       304354 non-null  object
 17  CNT_FAM_MEMBERS       438557 non-null  float64
dtypes: float64(2), int64(8), object(8)
memory usage: 60.2+ MB

In [5]: df_credit.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 3 columns):
 #   Column          Non-Null Count    Dtype
---  ------          --------------    -----
 0   ID              1048575 non-null  int64
 1   MONTHS_BALANCE  1048575 non-null  int64
 2   STATUS          1048575 non-null  object
dtypes: int64(2), object(1)
memory usage: 24.0+ MB
```

DATA CLEANING:

Data cleaning is the process of identifying and correcting or removing errors, inconsistencies, and other problems in a dataset. This typically involves removing duplicate records, filling in missing values, correcting data formatting issues, handling outliers and inconsistencies, and ensuring that the data is accurate, complete, and consistent.

Data cleaning is an important step in the data analysis process because it helps to ensure that the data you are working with is reliable and accurate, which in turn can improve the quality and validity of your analysis.

1. **Removing duplicates:**

1.Removing duplicates

```
[ ]: df_application = df_application.drop_duplicates()
     df_credit = df_credit.drop_duplicates()
     df_application.reset_index(inplace=True).drop(columns=['index'],inplace=True)
     df_credit.reset_index(inplace=True).drop(columns=['index'],inplace=True)
```

After reading the raw data from the files, we first removed all the duplicates from the data so that we will have no repetition.
Duplicates can occur for various reasons, such as data entry errors, system glitches, or data collection methods. This step will help us create single source of truth for our data. Additionally, when the duplicates are removed there will be difference in the index order, we have used reset index to reorder the data.

2. **Adding new columns:**

This will provide us more understanding about the data also this will help us create new inferences on the data.

2.Adding new columns

```
6]: df_application['AGE'] = (df_application['DAYS_BIRTH'] / -365.25).astype(int)

7]: df_application['AGE'].describe()

7]: count    438557.000000
    mean         43.295569
    std          11.459525
    min          20.000000
    25%          34.000000
    50%          42.000000
    75%          53.000000
    max          68.000000
    Name: AGE, dtype: float64

8]: df_application.query('DAYS_EMPLOYED==365243')['OCCUPATION_TYPE'].drop_duplicates()

8]: 7    NaN
    Name: OCCUPATION_TYPE, dtype: object

1]: df_application['TOTAL_EXPERIENCE'] = (df_application['DAYS_EMPLOYED'] / -365.25).astype(int)
```

In this step we have additionally created two new columns based on the existing columns. We have used ‘DAYS_BIRTH’ which had number of days elapsed since birth and created a new column called age, similarly we have ‘DAYS_EMPLOYED’ which was used in calculating total years of

experience. We have used this step because age and total years of experience makes more sense when we are looking at the data than DAYS_BIRTH and DAYS_EMPLOYED.

3. **Filtering out rows based on value:**

When we were adding new columns to the data, we have identified that for 'days_employed' column there is a huge value that did not made much sense to the data, so we have filtered the data using that value and found the number of records in our data with that value.

3.Filtering out rows based on value

```
df_application['DAYS_EMPLOYED'] = df_application['DAYS_EMPLOYED'].where(df_application['DAYS_EMPLOYED']!=365243,0)
```

```
df_application['DAYS_EMPLOYED'].sort_values()
```

```
102186   -17531
102185   -17531
102187   -17531
102188   -17531
102189   -17531
          ...
174975       0
174974       0
174973       0
156723       0
268401       0
Name: DAYS_EMPLOYED, Length: 438557, dtype: int64
```

4. **Filling in missing values:**

Missing values can occur in datasets for many reasons, including data entry errors, data corruption, or incomplete data collection. There are several ways to handle missing values in a dataset. One common method is to fill in the missing values using some form of intuition.

4.Filling in missing values

```
df_application['OCCUPATION_TYPE'].fillna("No Job",inplace=True)
```

```
df_application['OCCUPATION_TYPE'].unique()
```

```
array(['No Job', 'Security staff', 'Sales staff', 'Accountants',
       'Laborers', 'Managers', 'Drivers', 'Core staff',
       'High skill tech staff', 'Cleaning staff', 'Private service staff',
       'Cooking staff', 'Low-skill Laborers', 'Medicine staff',
       'Secretaries', 'Waiters/barmen staff', 'HR staff', 'Realty agents',
       'IT staff'], dtype=object)
```

```
df_application['OCCUPATION_TYPE'].value_counts()
```

```
No Job                   134203
Laborers                  78240
Core staff                43007
Sales staff               41098
Managers                  35487
Drivers                   26090
```

In our data file we had so many rows with null for OCCUPATION_TYPE column, after observing some data rows we have filled the missing values using "No Job" for the null values. After taking the counts by "occupation_type" we had 134203 records were nulls in our data.

5. Conversion of string data to number/data values:

This involves parsing string data and converting it into numerical data, such as integers or floats, by typecasting. We had so many string columns in our data frame, which we further used in encoding. We had made sure that all our string columns were in proper format before applying encoding.

5.Conversion of string data to number/data values

making sure all the columns are strings before applying encoding

```
df_application['CODE_GENDER'] = df_application['CODE_GENDER'].astype(str)
df_application['FLAG_OWN_CAR'] = df_application['FLAG_OWN_CAR'].astype(str)
df_application['FLAG_OWN_REALTY'] = df_application['FLAG_OWN_REALTY'].astype(str)
df_application['NAME_INCOME_TYPE'] = df_application['NAME_INCOME_TYPE'].astype(str)
df_application['NAME_EDUCATION_TYPE'] = df_application['NAME_EDUCATION_TYPE'].astype(str)
df_application['NAME_FAMILY_STATUS'] = df_application['NAME_FAMILY_STATUS'].astype(str)
df_application['NAME_HOUSING_TYPE'] = df_application['NAME_HOUSING_TYPE'].astype(str)
df_application['OCCUPATION_TYPE'] = df_application['OCCUPATION_TYPE'].astype(str)
```

6. Changing categorical columns using encoding:

It is common to encounter categorical data in the form of text or categorical variables. Machine learning algorithms and statistical models require numerical input data, and thus, we need to convert categorical data into numerical data using encoding techniques. We have manually created dictionaries for our categorical data columns for encoding them to numerical types.

6.changing categorical columns using encoding

```
CODE_GENDER_CONVERSION = {'F': 0, 'M': 1}
FLAG_OWN_CAR_CONVERSION = {'N': 0, 'Y': 1}
FLAG_OWN_REALTY_CONVERSION = {'N': 0, 'Y': 1}
NAME_INCOME_TYPE_CONVERSION = {'Commercial associate': 0, 'Pensioner': 1, 'State servant': 2, 'Student': 3, 'Working': 4}
NAME_FAMILY_STATUS_CONVERSION = {'Civil marriage': 0, 'Married': 1, 'Separated': 2, 'Single / not married': 3, 'Widow': 4}
OCCUPATION_TYPE_CONVERSION = {'Accountants': 0, 'Cleaning staff': 1, 'Cooking staff': 2, 'Core staff': 3, 'Drivers': 4, 'HR staff
NAME_HOUSING_TYPE_CONVERSION = {'Co-op apartment': 0, 'House / apartment': 1, 'Municipal apartment': 2, 'Office apartment': 3, 'R
NAME_EDUCATION_TYPE_CONVERSION = {'Academic degree': 0, 'Higher education': 1, 'Incomplete higher': 2, 'Lower secondary': 3, 'Sec
```

```
df_application.replace({'CODE_GENDER' : CODE_GENDER_CONVERSION}, inplace=True)
df_application.replace({'FLAG_OWN_CAR' : FLAG_OWN_CAR_CONVERSION}, inplace=True)
df_application.replace({'FLAG_OWN_REALTY' : FLAG_OWN_REALTY_CONVERSION}, inplace=True)
df_application.replace({'NAME_INCOME_TYPE' : NAME_INCOME_TYPE_CONVERSION}, inplace=True)
df_application.replace({'NAME_FAMILY_STATUS' : NAME_FAMILY_STATUS_CONVERSION}, inplace=True)
df_application.replace({'OCCUPATION_TYPE' : OCCUPATION_TYPE_CONVERSION}, inplace=True)
df_application.replace({'NAME_HOUSING_TYPE' : NAME_HOUSING_TYPE_CONVERSION}, inplace=True)
df_application.replace({'NAME_EDUCATION_TYPE' : NAME_EDUCATION_TYPE_CONVERSION}, inplace=True)
```

7. Creating target column:

Our response variable had many constrains that were dependent on other columns, we have used Data cleaning and formatting techniques to create the target column.

```
Status_definition = {'C' : 'Good_Debt', 'X' : 'Good_Debt', '0' : 'Good_Debt', '1' : 'Neutral_Debt', '2' : 'Neutral_Debt', '3' : '
df_credit.replace({'STATUS' : Status_definition}, inplace=True)
```

In the status definition we have created a dictionary, which clearly states whether the debt the applicant is of category “good_debt” or “neutral_debt” or “Bad_debt”.

```
df_credit = df_credit.value_counts(subset=['ID', 'STATUS']).unstack(fill_value=0)
```

```
df_credit.loc[(df_credit['Good_Debt'] > df_credit['Neutral_Debt']), 'CREDIT_APPROVAL_STATUS'] = 1
df_credit.loc[(df_credit['Good_Debt'] > df_credit['Bad_Debt']), 'CREDIT_APPROVAL_STATUS'] = 1
df_credit.loc[(df_credit['Neutral_Debt'] > df_credit['Good_Debt']), 'CREDIT_APPROVAL_STATUS'] = 0
df_credit.loc[(df_credit['Neutral_Debt'] > df_credit['Bad_Debt']), 'CREDIT_APPROVAL_STATUS'] = 1
df_credit.loc[(df_credit['Bad_Debt'] > df_credit['Good_Debt']), 'CREDIT_APPROVAL_STATUS'] = 0
df_credit.loc[(df_credit['Bad_Debt'] > df_credit['Neutral_Debt']), 'CREDIT_APPROVAL_STATUS'] = 0
```

After applying, the debt type to each applicant we then created a column CREDIT_APPROVAL_STATUS (response variable) based on their debt history.

8. **Dropping unwanted columns:**

The columns that do not majorly contribute to the data are removed as part of this step.

8. Dropping unwanted columns

```
df_credit.drop(columns=['Bad_Debt', 'Good_Debt', 'Neutral_Debt'],inplace=True)
```

```
df_credit.head()
```

STATUS ID	CREDIT_APPROVAL_STATUS
5001711	1.0
5001712	1.0
5001713	1.0
5001714	1.0
5001715	1.0

We have created additional columns in the data as part of creating the response variable, these columns will be no longer needed as they were only required temporarily. We have used drop function to drop these additional columns.

9. **Joining multiple datasets:**

The data is accumulated from various sources, all these data must be combined before further proceeding to train the model. We must have a common attribute that will help us join both the datasets.

```
merged_df = pd.merge(df_credit, df_application, on='ID', how='inner')
```

```
merged_df.count()
```

```
ID                          36457
CREDIT_APPROVAL_STATUS      36457
```

This is the final data set that was created by joining the files obtained from the different sources such as applicant portal and credit history portal.

10. **Reset Index:**

This will help us create the index in correct order when we perform manipulation on the data. The manipulation can be of any type such as dropping duplicates or filtering the data based on some value.

10. reset index

```
merged_df = merged_df.reset_index()
merged_df.drop(columns = ['index'],inplace=True)
```

```
df_application = df_application.drop_duplicates()
df_credit = df_credit.drop_duplicates()
df_application.reset_index(inplace=True)
df_application.drop(columns=['index'],inplace=True)
df_credit.reset_index(inplace=True)
df_credit.drop(columns=['index'],inplace=True)
```

EXPLORATORY DATA ANALYSIS:

1.Summary Statistics:

Here, we are summarizing the main statistics of the dataset i.e., calculating different statistics such as count, mean, standard deviation, minimum and maximum values of each column in the dataset.

1.Summary Statistics

```
In [28]: merged_df.describe().T
```

Out[28]:

	count	mean	std	min	25%	50%	75%	max
ID	36457.0	5.078227e+06	41875.240788	5008804.0	5042028.0	5074614.0	5115396.0	5150487.0
CREDIT_APPROVAL_STATUS	36457.0	9.951724e-01	0.069314	0.0	1.0	1.0	1.0	1.0
CODE_GENDER	36457.0	3.298955e-01	0.470181	0.0	0.0	0.0	1.0	1.0
FLAG_OWN_CAR	36457.0	3.797076e-01	0.485321	0.0	0.0	0.0	1.0	1.0
FLAG_OWN_REALTY	36457.0	6.721892e-01	0.469422	0.0	0.0	1.0	1.0	1.0
CNT_CHILDREN	36457.0	4.303152e-01	0.742367	0.0	0.0	0.0	1.0	19.0
AMT_INCOME_TOTAL	36457.0	1.866857e+05	101789.226482	27000.0	121500.0	157500.0	225000.0	1575000.0
NAME_INCOME_TYPE	36457.0	2.398195e+00	1.734032	0.0	1.0	4.0	4.0	4.0
NAME_EDUCATION_TYPE	36457.0	3.097183e+00	1.341292	0.0	1.0	4.0	4.0	4.0
NAME_FAMILY_STATUS	36457.0	1.367885e+00	0.954557	0.0	1.0	1.0	1.0	4.0
NAME_HOUSING_TYPE	36457.0	1.282881e+00	0.951675	0.0	1.0	1.0	1.0	5.0
DAYS_BIRTH	36457.0	-1.597517e+04	4200.549944	-25152.0	-19438.0	-15563.0	-12462.0	-7489.0
DAYS_EMPLOYED	36457.0	-2.200317e+03	2366.796762	-15713.0	-3153.0	-1552.0	-408.0	0.0
FLAG_MOBIL	36457.0	1.000000e+00	0.000000	1.0	1.0	1.0	1.0	1.0
FLAG_WORK_PHONE	36457.0	2.255260e-01	0.417934	0.0	0.0	0.0	0.0	1.0
FLAG_PHONE	36457.0	2.948131e-01	0.455965	0.0	0.0	0.0	1.0	1.0
FLAG_EMAIL	36457.0	8.972214e-02	0.285787	0.0	0.0	0.0	0.0	1.0
OCCUPATION_TYPE	36457.0	1.090169e+01	5.974049	0.0	6.0	10.0	18.0	18.0
CNT_FAM_MEMBERS	36457.0	2.198453e+00	0.911686	1.0	2.0	2.0	3.0	20.0
AGE	36457.0	4.323203e+01	11.503981	20.0	34.0	42.0	53.0	68.0
TOTAL_EXPERIENCE	36457.0	5.612804e+00	6.414407	0.0	1.0	4.0	8.0	43.0

2.Correlation Analysis:

The model can be simplified to improve its performance by removing redundant features, which can be found via correlation analysis. Also, it helps in locating critical characteristics that strongly correlate with the target variable, which is helpful for predictive modeling. The pairwise correlation coefficients between each pair of columns in a dataset are displayed in a correlation matrix that we are creating. The resulting matrix is a square matrix, where each element represents the correlation coefficient between the respective pair of columns.

2.Correlation Analysis

```
In [29]: merged_df.corr().T
```

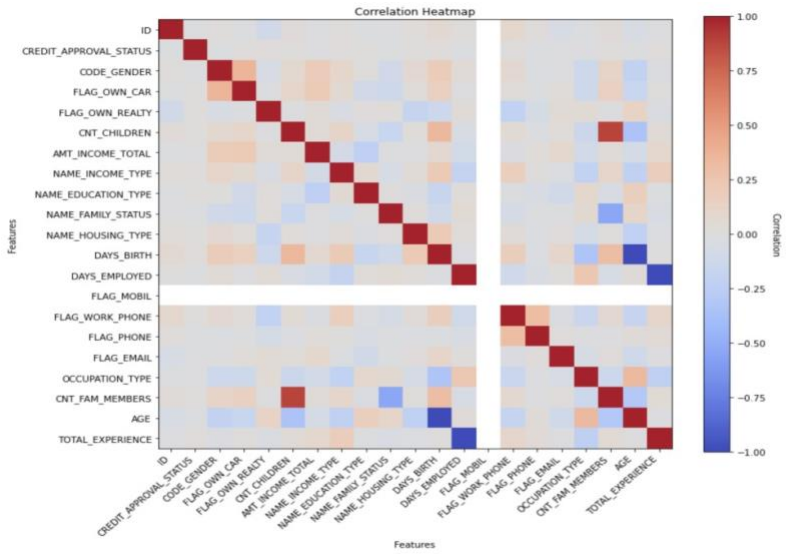
Out[29]:

	ID	CREDIT_APPROVAL_STATUS	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	NAME_INCOME_TYPE	NAME_EDUC
ID	1.000000	-0.008189	0.012022	-0.011163	-0.098851	0.028878	-0.017667	0.023193	
CREDIT_APPROVAL_STATUS	-0.008189	1.000000	-0.010048	0.005568	0.009531	0.005190	-0.004476	0.013940	
CODE_GENDER	0.012022	-0.010048	1.000000	0.361379	-0.050758	0.077690	0.197805	0.105639	
FLAG_OWN_CAR	-0.011163	0.005568	0.361379	1.000000	-0.015185	0.105839	0.215506	0.054817	
FLAG_OWN_REALTY	-0.098851	0.009531	-0.050758	-0.015185	1.000000	-0.000575	0.032719	-0.046543	
CNT_CHILDREN	0.028878	0.005190	0.077690	0.105839	-0.000575	1.000000	0.033691	0.109401	
AMT_INCOME_TOTAL	-0.017667	-0.004476	0.197805	0.215506	0.032719	0.033691	1.000000	-0.072974	
NAME_INCOME_TYPE	0.023193	0.013940	0.105639	0.054817	-0.046543	0.109401	-0.072974	1.000000	
NAME_EDUCATION_TYPE	-0.009211	0.004161	0.005880	-0.101272	0.010997	-0.049823	-0.226931	0.057225	
NAME_FAMILY_STATUS	-0.004336	-0.010054	-0.099289	-0.121716	0.022993	-0.160386	-0.001191	-0.048532	
NAME_HOUSING_TYPE	0.020613	0.002406	0.070301	0.016337	-0.179187	0.023302	-0.006480	0.035694	
DAYS_BIRTH	0.056016	0.015505	0.202352	0.157144	-0.129838	0.339357	0.067908	0.213509	
DAYS_EMPLOYED	0.005745	-0.013357	0.031731	-0.006244	0.033646	-0.043358	-0.087130	-0.192949	
FLAG_MOBIL	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
FLAG_WORK_PHONE	0.079215	0.004443	0.064994	0.021644	-0.207732	0.048091	-0.037746	0.165785	
FLAG_PHONE	0.009879	-0.007909	-0.026833	-0.014019	-0.066601	-0.016291	0.017245	0.006529	
FLAG_EMAIL	-0.046979	-0.000289	-0.003284	0.021750	0.052194	0.015960	0.086681	-0.019567	
OCCUPATION_TYPE	-0.009127	-0.003067	-0.130181	-0.119776	0.048800	-0.134796	-0.091777	-0.204559	
CNT_FAM_MEMBERS	0.026624	0.005178	0.110782	0.151814	-0.005723	0.889114	0.023750	0.109313	
AGE	-0.056346	-0.015486	-0.202286	-0.156825	0.129237	-0.339310	-0.067622	-0.213533	
TOTAL_EXPERIENCE	-0.006937	0.013749	-0.037237	0.001127	-0.031514	0.037226	0.082848	0.185540	

21 rows × 21 columns

3.Heatmap to show the correlation:

we are performing the heatmap for the correlated matrix in the above step to visualize the individual values in a matrix are represented as colors.

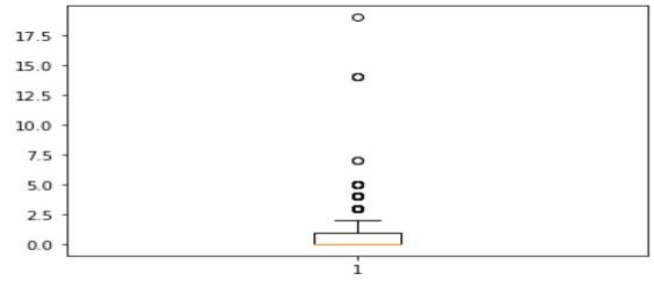


4.Outlier Detection:

As part of Outlier Detection, we identify the data that is significantly different from most of the data. This can occur because of data entry errors or measurement errors. We have used to boxplot from matplotlib to identify the outliers in our data.

```
plt.boxplot(merged_df['CNT_CHILDREN'])
```

```
{'whiskers': [<matplotlib.lines.Line2D at 0x28522844e20>,
  <matplotlib.lines.Line2D at 0x28522855130>],
 'caps': [<matplotlib.lines.Line2D at 0x285228554c0>,
  <matplotlib.lines.Line2D at 0x285228556d0>],
 'boxes': [<matplotlib.lines.Line2D at 0x28522844b50>],
 'medians': [<matplotlib.lines.Line2D at 0x285228559a0>],
 'fliers': [<matplotlib.lines.Line2D at 0x28522855c70>],
 'means': []}
```



In the above boxplot there seems to be a smaller number of applicants that have more than 15 children. So, we further observed the data.

Removing the outliers from the cnt_children column

```
merged_df[merged_df['CNT_CHILDREN']>=17.5]
```

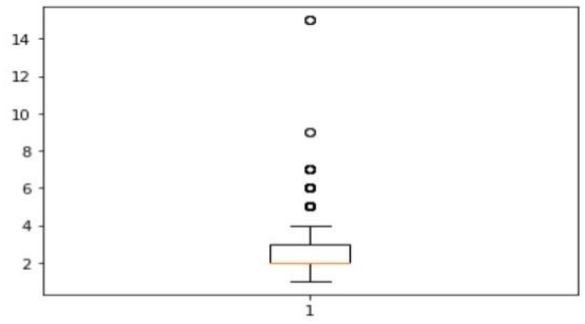
	ID	CREDIT_APPROVAL_STATUS	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN
24753	5105054	1.0	0	0	1	19

```
merged_df = merged_df[merged_df['CNT_CHILDREN']<17.5]
```

We have removed the data with records more than 17 children as there is only one row and it did not much impact the data.

```
plt.boxplot(merged_df['CNT_FAM_MEMBERS'])
```

```
{'whiskers': [<matplotlib.lines.Line2D at 0x285228c1430>,
  <matplotlib.lines.Line2D at 0x285228c1700>],
 'caps': [<matplotlib.lines.Line2D at 0x285228c19d0>,
  <matplotlib.lines.Line2D at 0x285228c1ca0>],
 'boxes': [<matplotlib.lines.Line2D at 0x285228c1160>],
 'medians': [<matplotlib.lines.Line2D at 0x285228c1f70>],
 'fliers': [<matplotlib.lines.Line2D at 0x285228cc280>],
 'means': []}
```




```
merged_df[merged_df['CNT_FAM_MEMBERS']>=12]
```

	ID	CREDIT_APPROVAL_STATUS	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	NAMI
14132	5061207	1.0	1	1	1	14	225000.0	
14133	5061210	1.0	1	1	1	14	225000.0	
14134	5061211	1.0	1	1	1	14	225000.0	

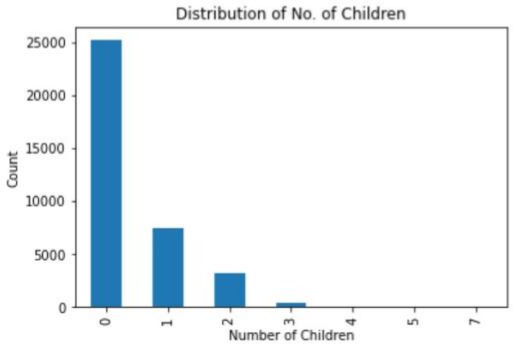
```
merged_df = merged_df[merged_df['CNT_FAM_MEMBERS']<12]
```

Similarly, we have created a box plot for CNT_FAM_MEMBERS column, we have 3 records with applicants' family members greater than 12, so we removed these 3 columns from our data.

5. Using bar chart to find distribution of categorical data:

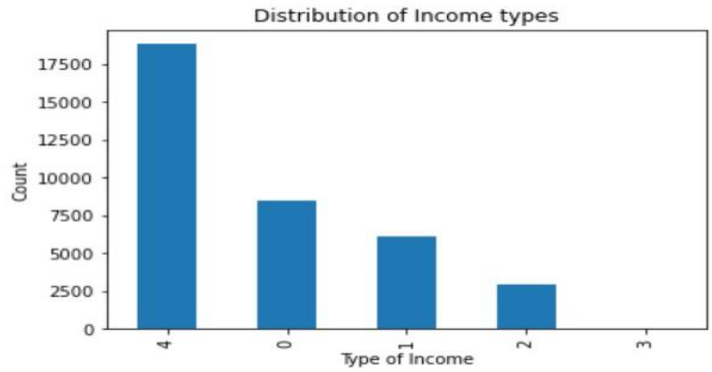
The distribution of data based on a category can be obtained by using a bar chart.

```
children_counts = merged_df['CNT_CHILDREN'].value_counts()
children_counts.plot(kind='bar')

plt.title('Distribution of No. of Children')

plt.xlabel('Number of Children')
plt.ylabel('Count')

plt.xticks(range(len(children_counts)), children_counts.index)

plt.show()
print("From the bar chart we have observed that there are compartively very less number of families with 3 or more children")
```



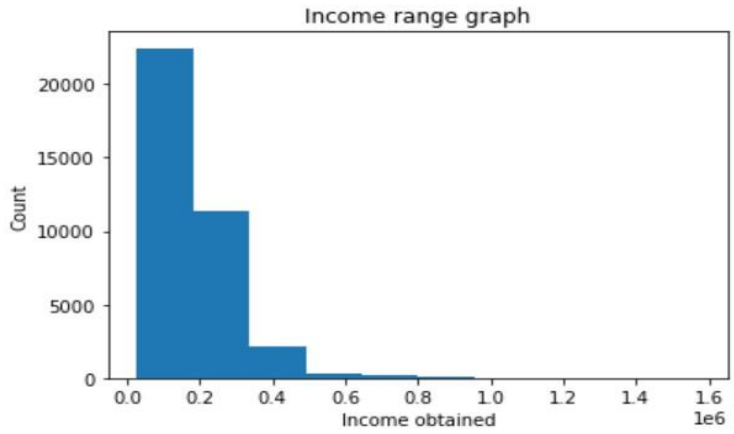
From the bar chart we have observed that there are comparatively very a smaller number of families with 3 or more children

```
income_type_counts = merged_df['NAME_INCOME_TYPE'].value_counts()
income_type_counts.plot(kind='bar')

plt.title('Distribution of Income types')

plt.xlabel('Type of Income')
plt.ylabel('Count')

plt.xticks(range(len(income_type_counts)), income_type_counts.index)

plt.show()
```



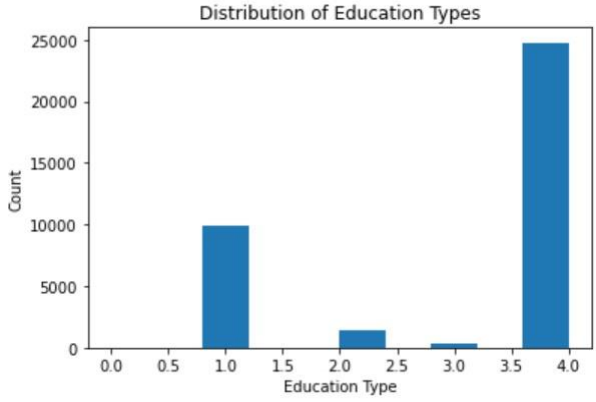
6. Histogram to find the frequency of observations:

```
plt.hist(merged_df['AMT_INCOME_TOTAL'])
plt.title('Income range graph')
plt.xlabel('Income obtained')
plt.ylabel('Count')

plt.show()
```



This histogram is used to show the distribution of income among the applicants.

```
plt.title('Distribution of Education Types')

plt.xlabel('Education Type')
plt.ylabel('Count')

plt.show()

print(NAME_EDUCATION_TYPE_CONVERSION)
print("Applicants have mostly completed their secondary school, also some applicants have even comple
```

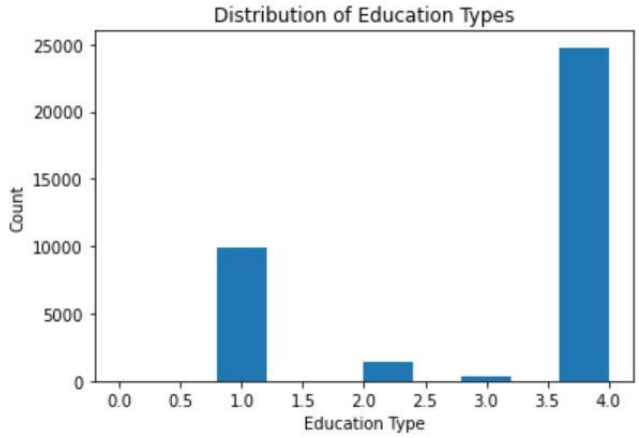


This histogram is used to show the distribution of type of education among the applicants.

7. Using skew to find the skewness of the data:

skew refers to the degree of asymmetry in the distribution of the data. A histogram is said to be skewed if it is not symmetric around its central point.
A positive skewness coefficient indicates a right-skewed distribution, a negative skewness coefficient indicates a left-skewed distribution, and a skewness coefficient of zero indicates a symmetric distribution.

```
plt.title('Distribution of Education Types')

plt.xlabel('Education Type')
plt.ylabel('Count')

plt.show()

print(NAME_EDUCATION_TYPE_CONVERSION)
print("Applicants have mostly completed their secondary school, also some applicants have even comple
```



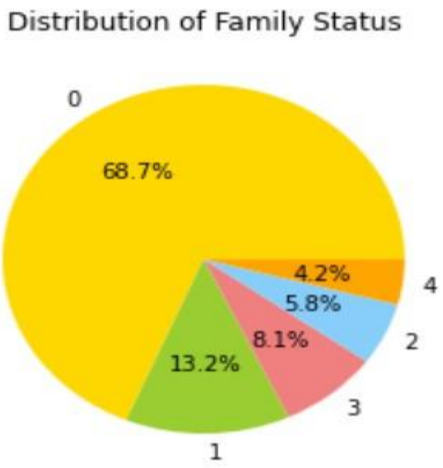

```
skew = skew(merged_df['OCCUPATION_TYPE'])
ation_skew)
skewness value is negative for occupation type. \nThis indicates that the distribution of data is negatively skewed, with a l
```

```
-0.146216574987524
```

The skewness value is negative for occupation type.
This indicates that the distribution of data is negatively skewed, with a long tail towards the left. This means that most of the data is concentrated towards the right side of the distribution.

8. PIE Chart to find proportion of applicants:

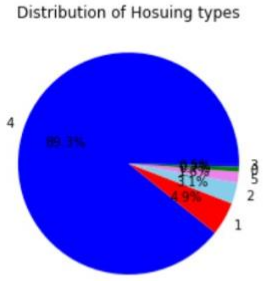
A pie chart is a circular graph that is divided into slices to represent the relative sizes of different categories or values. Each slice represents a proportion of the whole, and the total of all the slices equals 100%.




```
Around 70% of our applicatns are married
```

In the family status, around 70% of the applicants had Civil marriage.

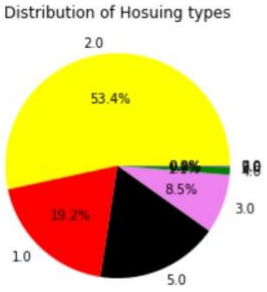
```
plt.pie(merged_df['NAME_HOUSING_TYPE'].value_counts(), labels=merged_df['NAME_HOUSING_TYPE'].unique(), colors=colors, autopct

plt.title('Distribution of Hosuing types')
plt.show()
print(NAME_HOUSING_TYPE_CONVERSION)
print("Around 90% of our applicatns are living in rented houses")
```




```
{'Co-op apartment': 0, 'House / apartment': 1, 'Municipal apartment': 2, 'Office apartment': 3, 'Rented apartment': 4, 'With
parents': 5}
Around 90% of our applicatns are living in rented houses
```

Around 90% of our applicants are living in rented houses.

```
colors = ['Yellow', 'red', 'Black', 'violet', 'green','blue']

plt.pie(merged_df['CNT_FAM_MEMBERS'].value_counts(), labels=merged_df['CNT_FAM_MEMBERS'].unique(), colors=colors, autopct='%1

plt.title('Distribution of Hosuing types')
plt.show()

print("Around 54% of our applicatns are having 2 children")
```

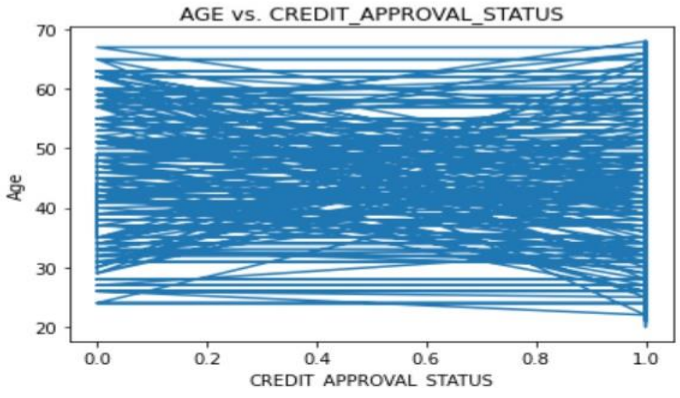


Around 54% of our applicants are having 2 children.

9. <u>Line Graph to see the impact of one column on the other:</u>

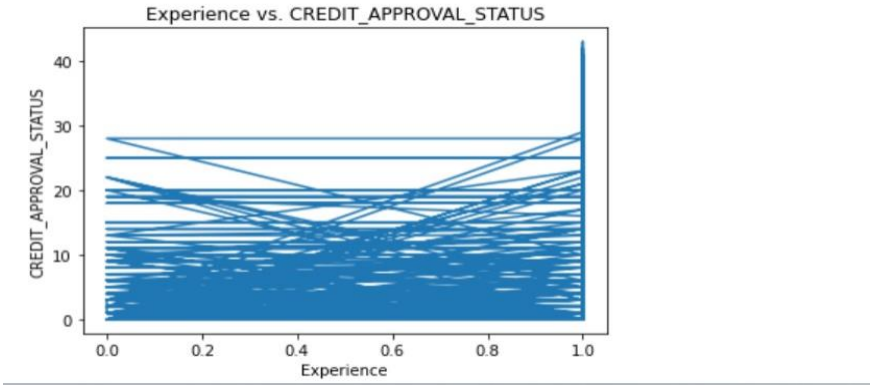
A line graph is used to compare the impact of a data column on the other column, the columns can either be related or not related.

```
x = merged_df['CREDIT_APPROVAL_STATUS']
y = merged_df['AGE']
plt.plot(x, y)

# Add title and labels
plt.title('AGE vs. CREDIT_APPROVAL_STATUS')
plt.xlabel('CREDIT_APPROVAL_STATUS')
plt.ylabel('Age')

plt.show()

print("Approval is not that much impacted by age")
```



There is not much impact of age on “credit_approval_status”.

```
x = merged_df['CREDIT_APPROVAL_STATUS']
y = merged_df['TOTAL_EXPERIENCE']
plt.plot(x, y)

# Add title and labels
plt.title('Experience vs. CREDIT_APPROVAL_STATUS')
plt.xlabel('Experience')
plt.ylabel('CREDIT_APPROVAL_STATUS')

plt.show()

print("With increase in experience the Income have steadily raised")
```



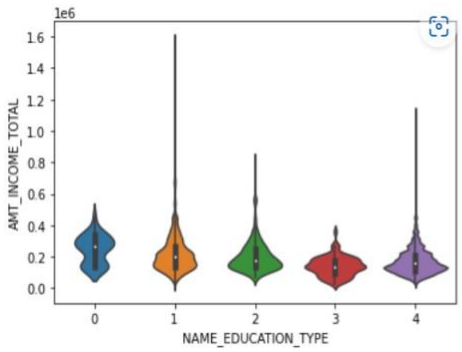
With increase in experience the approval had impact and people with more experience mostly got approval.

10. Violin graph to visualize distribution of numerical data across various categories:

Violin plot is a type of data visualization that displays the distribution of numeric data across different levels or categories of a categorical variable.

```
sns.violinplot(x='NAME_EDUCATION_TYPE', y='AMT_INCOME_TOTAL', data=merged_df)

print(NAME_EDUCATION_TYPE_CONVERSION)
print("\n \nThe applicants who have registered and not capable of completing higher have more salaries compared with other ca
```

```
{'Academic degree': 0, 'Higher education': 1, 'Incomplete higher': 2, 'Lower secondary': 3, 'Secondary / secondary special':
4}

The applicants who have registered and not capable of completing higher have more salaries compared with other categories
```



PHASE 2

NORMALIZING THE DATA:

```
from sklearn.preprocessing import StandardScaler

# Scaling numerical columns
numerical_cols = ['CNT_CHILDREN', 'AMT_INCOME_TOTAL', 'DAYS_BIRTH',
                  'DAYS_EMPLOYED', 'CNT_FAM_MEMBERS', 'AGE', 'TOTAL_EXPERIENCE']
scaler = StandardScaler()
merged_df[numerical_cols] = scaler.fit_transform(merged_df[numerical_cols])
```

LIBRARIES USED FOR REGRESSION AND VISUALIZATIONS:

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

from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score,recall_score
from sklearn.metrics import confusion_matrix, classification_report, PrecisionRecallDisplay, RocCurveDisplay
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
```

SPLITTING THE DATA INTO TEST AND TRAIN DATA:

```
X_train, X_test, y_train, y_test = train_test_split(merged_df.drop(columns=['ID','CREDIT_APPROVAL_STATUS'])
                                                    , merged_df['CREDIT_APPROVAL_STATUS']
                                                    , test_size=0.20
                                                    , random_state=90)
```

ALGORITHMS:

The Algorithms used for the problem are:

1. Logistic Regression
2. Decision Trees
3. Random Forest
4. Support Vector Machine
5. Gradient Boosting Machines

LOGISTIC REGRESSION:

Logistic Regression can be used for the classification problem where we must predict the probability of a binary outcome variable. In our case, the target column is "CREDIT_APPROVAL_STATUS" which is a binary variable indicating whether a credit application is approved or not. Logistic regression can be used to predict the probability of credit approval based on the given features such as "CODE_GENDER", "FLAG_OWN_CAR", "CNT_CHILDREN", "AMT_INCOME_TOTAL", "DAYS_BIRTH", etc. Therefore, logistic regression is a suitable algorithm for our problem.

In credit risk analysis, logistic regression is frequently employed and has a track record of success in forecasting credit card defaults and other related outcomes. Financial organizations can reduce the risk of losses from default by utilizing logistic regression to help them decide which credit card applications to approve.

CODE FOR LOGISTIC REGRESSION:

Logistic Regression

```
logit = LogisticRegression(max_iter=10000).fit(X_train, y_train)
```

```
logit.predict(X_test)
```

```
array([1., 1., 1., ..., 1., 1., 1.])
```

```
y_pred = logit.predict(X_test)
score = accuracy_score(y_test, y_pred)
```

```
accuracies.update({'Logistic Regression':score})
print(accuracies['Logistic Regression'])
```

```
0.9954738718968591
```

DECISION TREES:

Decision Trees are supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation.

Decision trees are a good technique for credit approval status prediction because they can handle both numerical and categorical data, like the features in the given dataset. Decision trees work by splitting the data based on the features that provide the most information gain, or the most useful information for predicting the target variable.

In the case of credit approval status prediction, there may be complex relationships between different features and the target variable, such as a borrower's income, education, employment history, and family status. Decision trees can identify these relationships and create rules to predict whether an individual's credit application will be approved or not.

Additionally, decision trees are easy to interpret, as they can be visualized as a flowchart or tree structure, which can help explain the factors that influence a credit decision. This can be useful for regulatory compliance, as well as for developing fair and transparent lending practices.

CODE FOR DECISION TREES:

Decision Trees

```
tree = DecisionTreeClassifier()
```

```
tree.fit(X_train, y_train)
```

```
▸ DecisionTreeClassifier
```

```
y_pred = tree.predict(X_test)
```

```
score = accuracy_score(y_test, y_pred)
accuracies.update({'DecisionTreeClassifier':score})
print(accuracies['DecisionTreeClassifier'])
```

```
0.9935536963379509
```

SUPPORT VECTOR MACHINE (SVM):

Support Vector Machine (SVM) is a classification algorithm that tries to find the best separating A Support Vector Machine (SVM) is a type of machine learning algorithm that can be used for classification, regression, and other tasks.

In simple terms, an SVM works by finding the best possible line or boundary that can separate different classes of data points in a high-dimensional space. This line or boundary is called a "hyperplane", and the SVM aims to maximize the distance between the hyperplane and the closest data points from each class.

The data points that are closest to the hyperplane are called "support vectors", and they play a crucial role in determining the position and orientation of the hyperplane. Once the hyperplane has been found, the SVM can use it to make predictions on new data points, by determining which side of the hyperplane they belong to.

In the context of credit approval status, SVM can be used to predict whether an applicant's credit will be approved or not based on the given features. By identifying the support vectors, you can gain insight into the key features that contribute to credit approval decisions. For example, if a particular occupation type is often associated with support vectors, this may suggest that the applicant's job is a critical factor in determining whether they are approved for credit.

SVM can be a good choice for credit approval status prediction as it can handle non-linear boundaries and works well with high-dimensional data. However, the performance of SVM can be sensitive to the choice of kernel and hyperparameters, so it is important to experiment with different configurations to find the optimal model.

CODE FOR SUPPORT VECTOR MACHINE (SVM):

Support vector machine

```
svm = SVC(kernel='linear', probability=True)
```

```
svm.fit(X_train, y_train)
```

▸ SVC

```
y_pred = svm.predict(X_test)
```

```
score = accuracy_score(y_test, y_pred)
accuracies.update({'Support Vector Machine':score})
print(accuracies['Support Vector Machine'])
```

```
0.9954738718968591
```

RANDOM FOREST:

Random forest is a machine learning algorithm that can be used for both classification and regression tasks. It is an ensemble learning method that combines multiple decision trees to create a more accurate and robust model.

In a random forest, a set of decision trees are generated using a random subset of the available features and data points. Each decision tree independently predicts the outcome of the target variable, and the final prediction is determined by taking the average or majority vote of the individual decision trees.

The randomness in the selection of features and data points helps to reduce overfitting and improve the generalization of the model.

Random forest is a suitable algorithm for the credit approval problem for several reasons:

Robustness to noise and outliers: Credit approval datasets often have missing values, outliers, and noisy data. Random forest is robust to these data issues, as it builds multiple trees on different subsets of the data and features, reducing the impact of noisy data.

Handling of imbalanced data: Credit approval datasets often have imbalanced classes, with more rejected than approved applications. Random forest can handle imbalanced datasets by adjusting the weights of the samples, ensuring that the model learns from both classes.

Feature selection and interpretation: Credit approval datasets may have many features, and it can be challenging to identify which features are essential for deciding. Random forest provides a measure of feature importance, which can help in feature selection and interpretation of the results.

CODE FOR RANDOM FOREST:

Random Forest

```
rfc = RandomForestClassifier()
```

```
rfc.fit(X_train, y_train)
```

```
▸ RandomForestClassifier
```

```
y_pred = rfc.predict(X_test)
```

```
score = accuracy_score(y_test, y_pred)
accuracies.update({'RandomForestClassifier':score})
print(accuracies['RandomForestClassifier'])
```

```
0.9943766287203402
```

GRADIENT BOOSTING MACHINES (GBM):

Gradient Boosting Machines is a machine learning algorithm that works by iteratively training weak models, usually decision trees, and combining them into a strong predictive model.

The idea behind GBM is to improve the accuracy of predictions by focusing on the samples that are misclassified by the previous models. At each iteration, the model calculates the difference between the predicted and actual value for each sample and trains a new model to minimize this difference.

Gradient Boosting Machines is a good technique for the problem of credit approval status prediction because it is a powerful ensemble machine learning technique that can handle both numeric and categorical data. GBMs are capable of handling complex relationships between variables, which is essential in a credit approval scenario where various factors may affect the approval status.

GBMs are also known for their ability to handle imbalanced datasets, which is often the case with credit approval datasets where the number of approved applications is usually much higher than the number of denied applications.

Furthermore, GBMs have shown to have high accuracy and predictive power in various real-world scenarios, making it a popular choice for many machine learning practitioners. Therefore, it is a suitable technique for the given problem of credit approval status prediction.

CODE FOR GRADIENT BOOSTING MACHINES (GBM):

Gradient Boosting Machines (GBM)

```
gb = GradientBoostingClassifier()
```

```
gb.fit(X_train, y_train)
```

```
▸ GradientBoostingClassifier
```

```
y_pred = gb.predict(X_test)
```

```
score = accuracy_score(y_test, y_pred)
accuracies.update({'Gradient Boosting Machine':score})
print(accuracies['Gradient Boosting Machine'])
```

```
0.9942394733232752
```

VISUALIZATIONS:

CONFUSION MATRIX:

Confusion matrix is a good visualization for this problem as it helps you to evaluate the performance of your model by showing the number of true positives, false positives, true negatives, and false negatives.

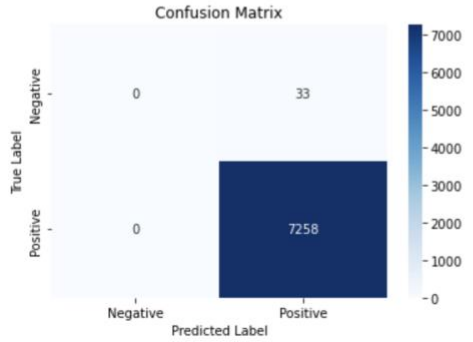
The number of true positives (cases where the model accurately predicts that a credit card application should be approved), true negatives (cases where the model accurately predicts that a credit card application should be rejected), false positives (cases where the model predicts that a credit card application should be approved but it should actually be rejected), and false negatives are displayed in a confusion matrix for credit card approval prediction. (i.e., cases where the model predicts that a credit card application should be rejected but it should be approved).

APPLYING CONFUSION MATRIX ON OUR ALGORITHMS

LOGISTIC REGRESSION:

Creating a confusion matrix

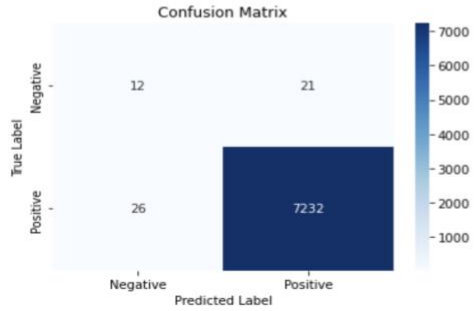
```
# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = cm.ravel()

# Plot confusion matrix as heatmap
labels = ['Negative', 'Positive']
sns.heatmap(cm, annot=True, cmap='Blues', xticklabels=labels, yticklabels=labels, fmt='d')
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



DECISION TREES:

Creating a confusion matrix

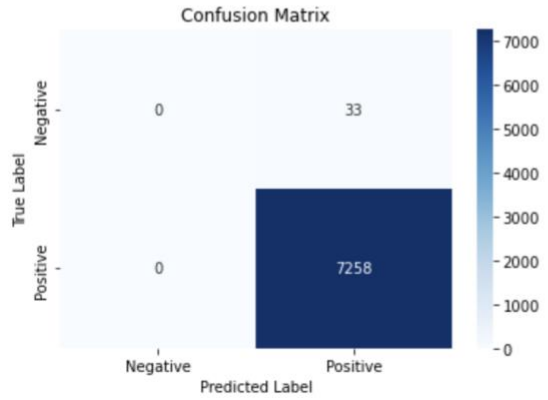
```
# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = cm.ravel()

# Plot confusion matrix as heatmap
labels = ['Negative', 'Positive']
sns.heatmap(cm, annot=True, cmap='Blues', xticklabels=labels, yticklabels=labels, fmt='d')
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



SUPPORT VECTOR MACHINE (SVM):

Creating a confusion matrix

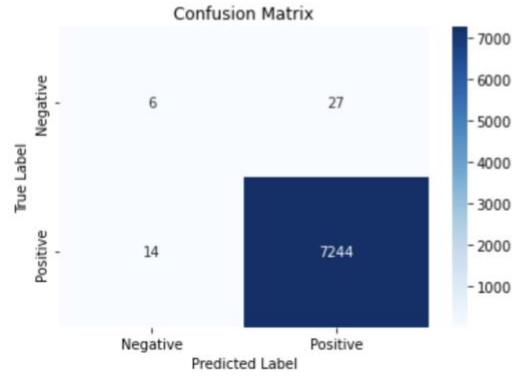
```
# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = cm.ravel()

# Plot confusion matrix as heatmap
labels = ['Negative', 'Positive']
sns.heatmap(cm, annot=True, cmap='Blues', xticklabels=labels, yticklabels=labels, fmt='d')
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



RANDOM FOREST:

Creating a confusion matrix

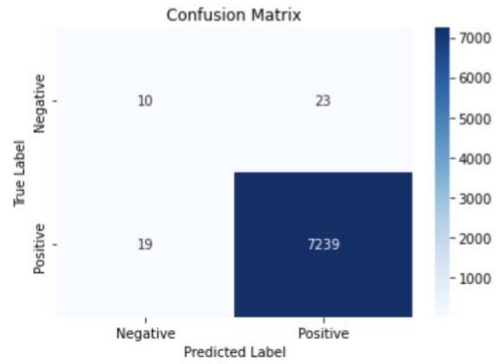
```
# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = cm.ravel()

# Plot confusion matrix as heatmap
labels = ['Negative', 'Positive']
sns.heatmap(cm, annot=True, cmap='Blues', xticklabels=labels, yticklabels=labels, fmt='d')
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



GRADIENT BOOSTING MACHINES (GBM):

Creating a confusion matrix

```
# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = cm.ravel()

# Plot confusion matrix as heatmap
labels = ['Negative', 'Positive']
sns.heatmap(cm, annot=True, cmap='Blues', xticklabels=labels, yticklabels=labels, fmt='d')
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



PRECISION-RECALL CURVE:

Precision-Recall Curve is a good visualization for the classification problem with imbalanced classes, like the one you have, where the positive class (approved credit) is much smaller than the negative class (rejected credit). In such problems, accuracy can be a misleading metric, as the classifier may have high accuracy by simply predicting the majority class all the time. In such cases, Precision and Recall can provide a more accurate evaluation of the classifier's performance.

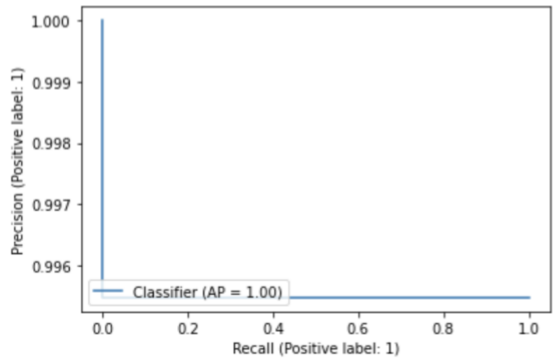
Precision-Recall Curve plots the trade-off between Precision and Recall for different thresholds of the classifier's output probability. It helps to visualize the classifier's performance on different levels of recall and precision, which is important in imbalanced classification problems. It can also help you to choose the threshold value that balances between precision and recall, depending on your specific needs.

CALCULATING PRECISION RECALL ON ALL OUR MODELS

LOGISTIC REGRESSION:

```
#calculating precision and reall
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

print('Precision: ',precision)
print('Recall: ',recall)

#Plotting Precision-Recall Curve
disp = PrecisionRecallDisplay.from_predictions(y_test, y_pred)
```

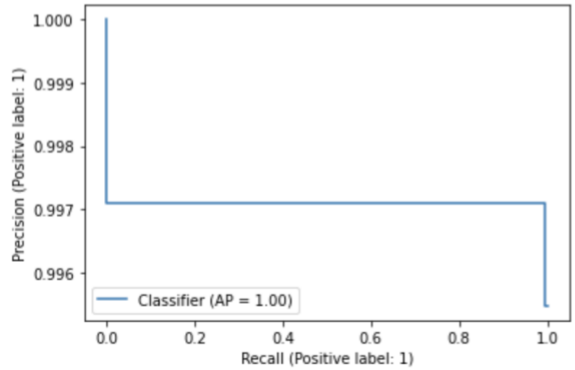
```
Precision:  0.9954738718968591
Recall:  1.0
```



DECISION TREES:

```
#calculating precision and reall
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

print('Precision: ',precision)
print('Recall: ',recall)

#Plotting Precision-Recall Curve
disp = PrecisionRecallDisplay.from_predictions(y_test, y_pred)
```

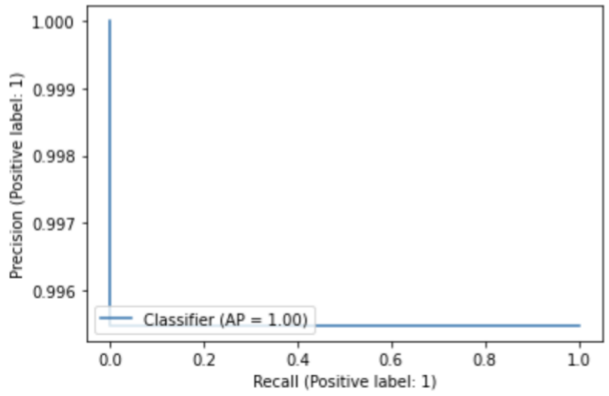
```
Precision:  0.9971046463532331
Recall:  0.9964177459355195
```



SUPPORT VECTOR MACHINE (SVM):

```
: #calculating precision and reall
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

print('Precision: ',precision)
print('Recall: ',recall)

#Plotting Precision-Recall Curve
disp = PrecisionRecallDisplay.from_predictions(y_test, y_pred)
```

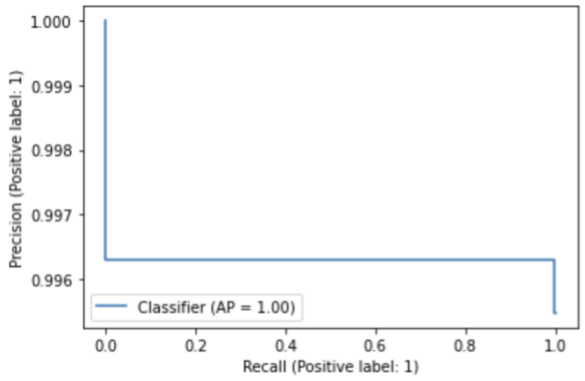
```
Precision:  0.9954738718968591
Recall:  1.0
```



RANDOM FOREST:

```
#calculating precision and reall
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

print('Precision: ',precision)
print('Recall: ',recall)

#Plotting Precision-Recall Curve
disp = PrecisionRecallDisplay.from_predictions(y_test, y_pred)
```

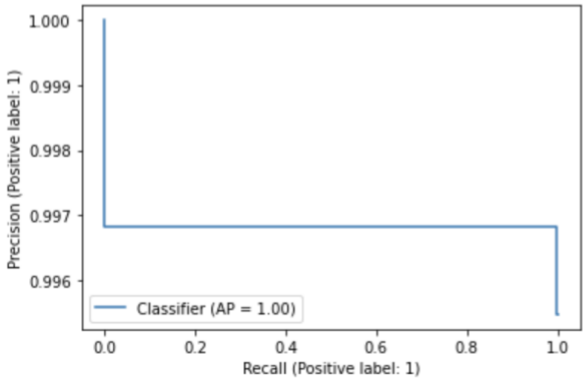
```
Precision:  0.9962866180717921
Recall:  0.9980710939652797
```



GRADIENT BOOSTING MACHINES (GBM):

```
#calculating precision and reall
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

print('Precision: ',precision)
print('Recall: ',recall)

#Plotting Precision-Recall Curve
disp = PrecisionRecallDisplay.from_predictions(y_test, y_pred)
```

```
Precision:  0.9968328284219223
Recall:  0.9973821989528796
```



PAIR PLOT:

Using **pair plot** to visualize y_test and y_pred can help to see the relationship between the actual and predicted values, and to identify any patterns or trends in the model's performance. It can also help to identify any potential issues with the model, such as overfitting or underfitting. Additionally, pair plot can help to identify any potential interactions between different features that may be affecting the model's performance.

Overall, pair plot is a good visualization tool to gain insights into the relationship between multiple variables in a dataset and can be useful in evaluating the performance of a classification model.

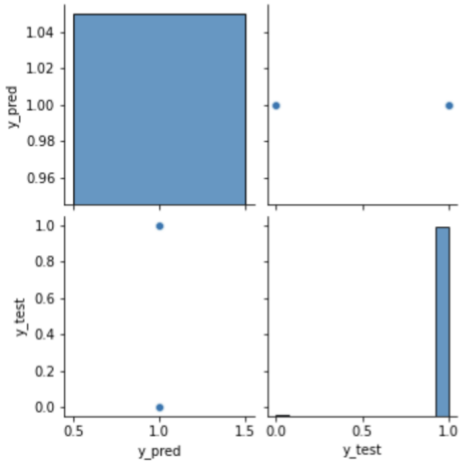
CREATING PAIRPLOT FOR ALL OUR MODELS

LOGISTIC REGRESSION:

Pairplot

```
graph_df = pd.DataFrame({'y_pred': y_pred, 'y_test': y_test})
sns.pairplot(graph_df)
```

```
<seaborn.axisgrid.PairGrid at 0x7fce787de0d0>
```

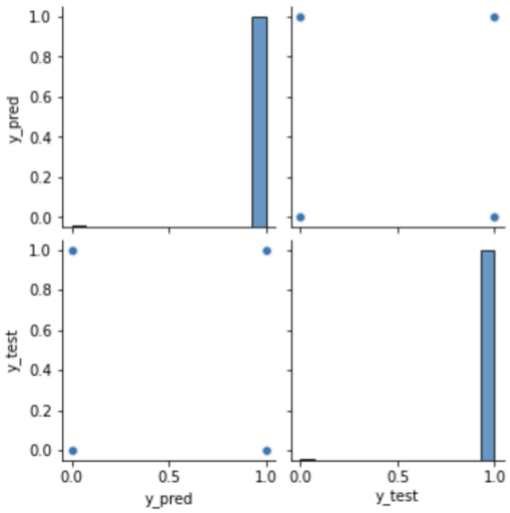


DECISION TREES:

Pairplot

```
graph_df = pd.DataFrame({'y_pred': y_pred, 'y_test': y_test})
sns.pairplot(graph_df)
```

```
<seaborn.axisgrid.PairGrid at 0x7fce5be53be0>
```

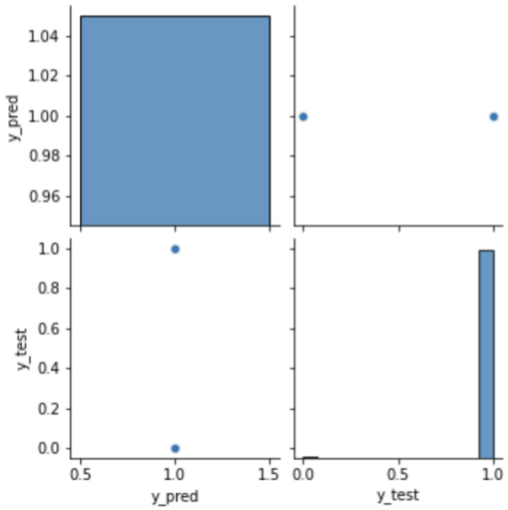


SUPPORT VECTOR MACHINE (SVM):

Pairplot

```
graph_df = pd.DataFrame({'y_pred': y_pred, 'y_test': y_test})
sns.pairplot(graph_df)
```

```
<seaborn.axisgrid.PairGrid at 0x7fce5bc5de50>
```

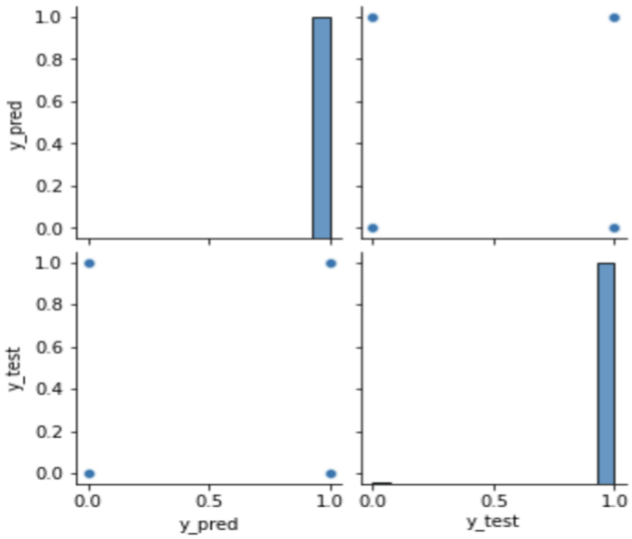


RANDOM FOREST:

Pairplot

```
graph_df = pd.DataFrame({'y_pred': y_pred, 'y_test': y_test})
sns.pairplot(graph_df)
```

```
<seaborn.axisgrid.PairGrid at 0x7fce9aee17c0>
```

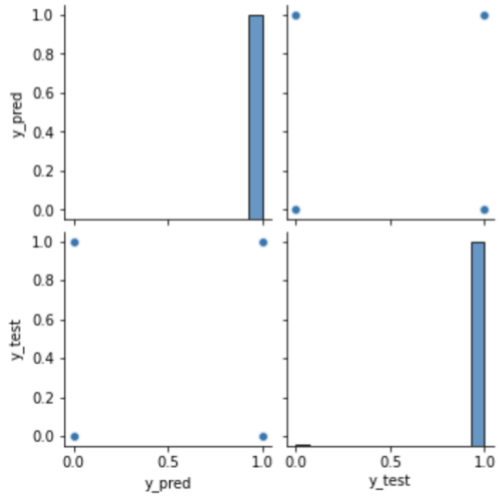


GRADIENT BOOSTING MACHINES (GBM):

Pairplot

```
graph_df = pd.DataFrame({'y_pred': y_pred, 'y_test': y_test})
sns.pairplot(graph_df)
```

```
<seaborn.axisgrid.PairGrid at 0x7fce9a9e3190>
```



CLASSIFICATION REPORT:

A **classification report** is a good visualization for y_pred and y_test because it provides a detailed evaluation of the performance of the classifier in terms of precision, recall, F1-score, and support for each class. It also gives an overall accuracy score, as well as macro-averaged and weighted-average scores for precision, recall, and F1-score.

This information is valuable because it helps to assess the quality of the predictions made by the classifier and to identify which classes may be more difficult to predict accurately. It also provides a summary of the overall performance of the classifier that can be used to compare it with other classifiers and to determine if further optimization is needed.

CREATING CLASSIFICATIONS REPORT FOR ALL THE MODELS

LOGISTIC REGRESSION:

Creating a classification report to compare y_test and y_pred

```
print(classification_report(y_test, y_pred, zero_division=1))
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.00      0.00        33
         1.0       1.00      1.00      1.00      7258

    accuracy                           1.00      7291
   macro avg       1.00      0.50      0.50      7291
weighted avg       1.00      1.00      0.99      7291
```

DECISION TREES:

Creating a classification report to compare y_test and y_pred

```
print(classification_report(y_test, y_pred, zero_division=1))
```

```
              precision    recall  f1-score   support

         0.0       0.32      0.36      0.34        33
         1.0       1.00      1.00      1.00      7258

    accuracy                           0.99      7291
   macro avg       0.66      0.68      0.67      7291
weighted avg       0.99      0.99      0.99      7291
```

SUPPORT VECTOR MACHINE (SVM):

Creating a classification report to compare y_test and y_pred

```
print(classification_report(y_test, y_pred, zero_division=1))
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.00      0.00        33
         1.0       1.00      1.00      1.00      7258

    accuracy                           1.00      7291
   macro avg       1.00      0.50      0.50      7291
weighted avg       1.00      1.00      0.99      7291
```

RANDOM FOREST:

Creating a classification report to compare y_test and y_pred

```
print(classification_report(y_test, y_pred, zero_division=1))
```

```
              precision    recall  f1-score   support

         0.0       0.30      0.18      0.23        33
         1.0       1.00      1.00      1.00      7258

    accuracy                           0.99      7291
   macro avg       0.65      0.59      0.61      7291
weighted avg       0.99      0.99      0.99      7291
```

GRADIENT BOOSTING MACHINES (GBM):

Creating a classification report to compare y_test and y_pred

```
print(classification_report(y_test, y_pred, zero_division=1))
```

```
              precision    recall  f1-score   support

         0.0       0.34      0.30      0.32        33
         1.0       1.00      1.00      1.00      7258

    accuracy                           0.99      7291
   macro avg       0.67      0.65      0.66      7291
weighted avg       0.99      0.99      0.99      7291
```

ROCCURVEDISPLAY:

The **RocCurveDisplay** plot is a better visualization technique because it provides a comprehensive view of the performance of the model. It displays the true positive rate (sensitivity) against the false positive rate (1-specificity) for different thresholds of the predicted probabilities.

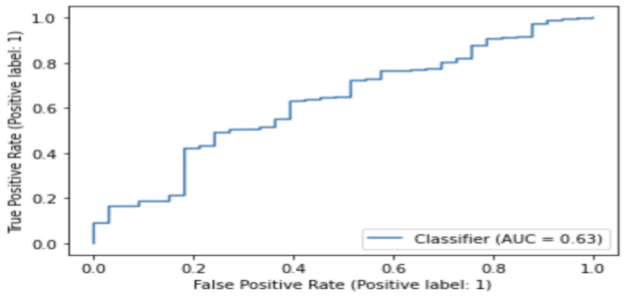
The plot helps in determining the optimal threshold for the classification model. If the curve is closer to the upper left corner of the plot, it suggests better model performance as it means the model is achieving higher true positive rates while keeping the false positive rate low. Additionally, it also helps in comparing the performance of different models by comparing their ROC curves. Overall, the RocCurveDisplay provides an intuitive way to visualize the performance of the classification model and helps in identifying the tradeoff between sensitivity and specificity for different thresholds.

CREATING ROC CURVE FOR ALL THE MODELS

LOGISTIC REGRESSION:

auc - roc_curve

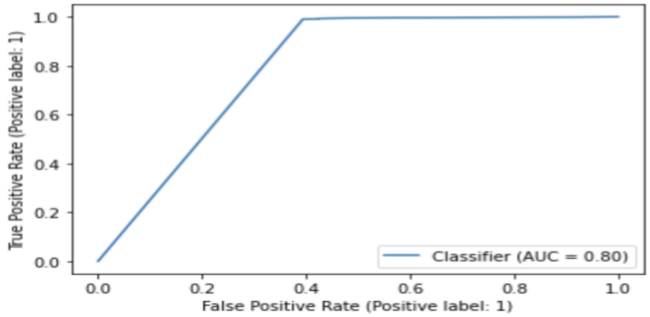
```
y_pred = logit.predict_proba(X_test)[:, 1]
roc_display = RocCurveDisplay.from_predictions(y_test, y_pred)
```



DECISION TREES:

auc - roc_curve

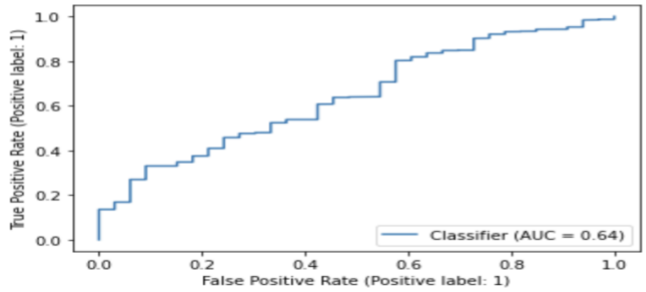
```
y_pred = tree.predict_proba(X_test)[:, 1]
roc_display = RocCurveDisplay.from_predictions(y_test, y_pred)
```



SUPPORT VECTOR MACHINE (SVM):

auc - roc_curve

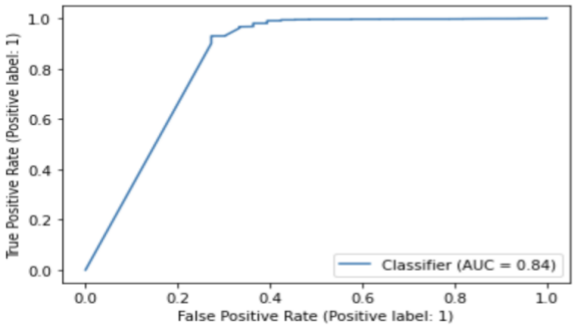
```
y_pred = svm.predict_proba(X_test)[:, 1]
roc_display = RocCurveDisplay.from_predictions(y_test, y_pred)
```



RANDOM FOREST:

auc - roc_curve

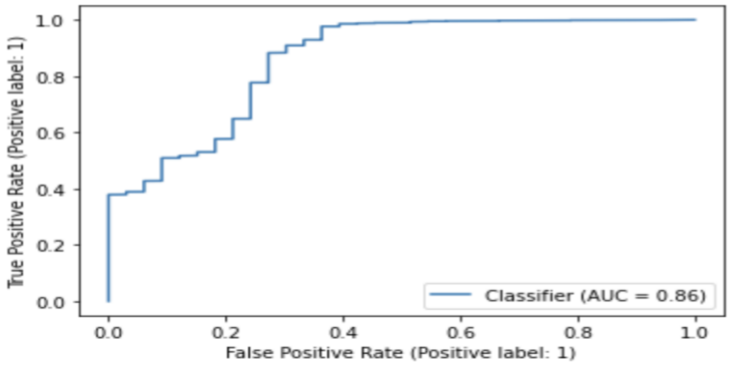
```
y_pred = rfc.predict_proba(X_test)[:, 1]
roc_display = RocCurveDisplay.from_predictions(y_test, y_pred)
```



GRADIENT BOOSTING MACHINES (GBM):

auc - roc_curve

```
y_pred = gb.predict_proba(X_test)[:, 1]
roc_display = RocCurveDisplay.from_predictions(y_test, y_pred)
```



ACCURACY COMPARISION:

Regression method	Accuracy
Logistic Regression	0.9954738718968591
DecisionTreeClassifier	0.9935536963379509
Support Vector Machine	0.9954738718968591
RandomForestClassifier	0.9943766287203402
Gradient Boosting Machine	0.9942394733232752

After evaluating the performance of all the models using accuracy metrics, it has been observed that all the models have similar accuracy scores on the given data. This suggests that any of these models

can be used to make predictions with comparable accuracy, and the choice of the model depends on other factors such as interpretability, computation time, or other specific requirements of the problem.

In other words, the models have similar predictive power on the given data, and selecting one model over the other is a matter of considering other factors that may affect the practical implementation and usability of the model.

PHASE 3

We provided a detailed overview of our website development process using Python Flask framework and the deployment of a Logistic Regression model. Our aim was to create a user-friendly website that incorporates machine learning capabilities to predict certain outcomes based on user input. We evaluated the performance of multiple models and ultimately selected the Logistic Regression model due to its superior accuracy. This report outlines the steps we took, the challenges we encountered, and the final outcomes achieved.

Website Development

Python Flask

We chose Python Flask, a lightweight web framework, for building our website. Flask offers flexibility and ease of use, making it an ideal choice for our project.

User Interface

Our website features a user-friendly interface with 21 different input fields to capture relevant information from the user. These inputs serve as features for our Logistic Regression model to make predictions.

Model Evaluation

Model Selection

We experimented with multiple machine learning algorithms, including Logistic Regression, Decision Trees, Random Forest, Support Vector Machines, and Neural Networks. We trained and evaluated each model using appropriate performance metrics.

Accuracy Comparison

We compared the accuracies of all the models to determine the best-performing one. Logistic Regression exhibited the highest accuracy among the models, making it the ideal choice for our prediction task.

Model Deployment

Model Training

We trained the Logistic Regression model using a labeled dataset, which served as the basis for learning patterns and making predictions.

Serialization

After training, we serialized the trained model using the pickle library in Python, resulting in the generation of a model.pkl file. This file contains the necessary information to make predictions using the trained Logistic Regression model.

Flask Integration

We integrated the model.pkl file into our Flask application. This allowed us to load the serialized model and use it for making predictions based on the user inputs obtained through the website.

Prediction Process

When a user submits the input data through the website, the Flask application retrieves the user's inputs and feeds them into the Logistic Regression model. The model then predicts the desired outcome based on the learned patterns and returns the result to the user.

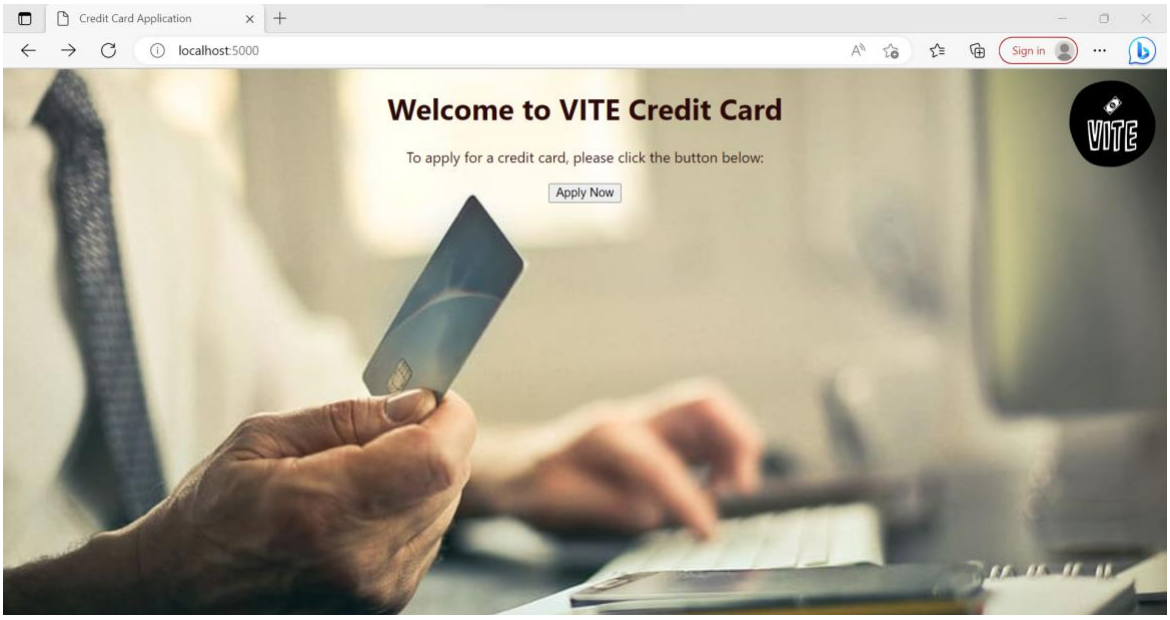
Website Hosting

The website is currently hosted on a local machine (localhost) and can be accessed by running the Python script.

Homepage

The home page of the VITE Credit Card Application Page is a simple and straightforward page that provides users with the information they need to apply for a VITE credit card. The page includes the following elements:

- A header that displays the VITE logo and the title of the page.
- A brief introduction that explains what the page is for.
- A call-to-action button that invites users to apply for a credit card.



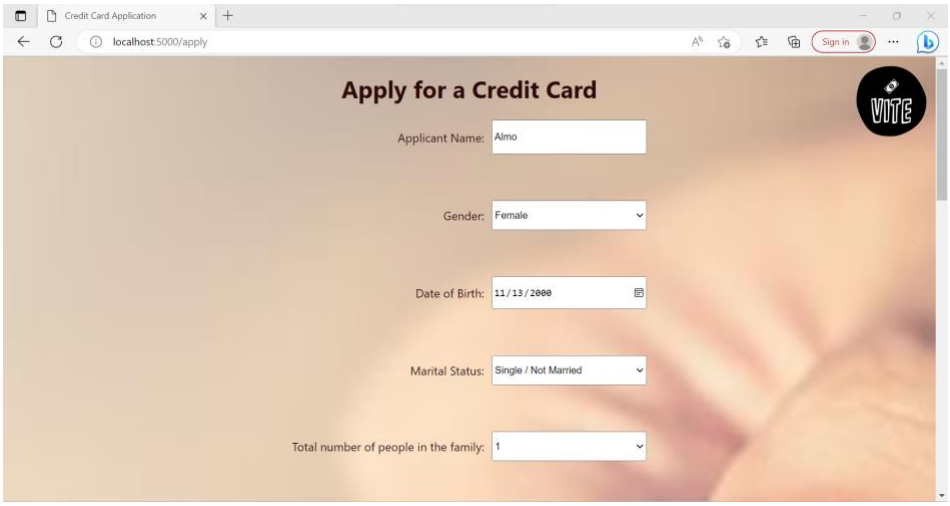
Application portal

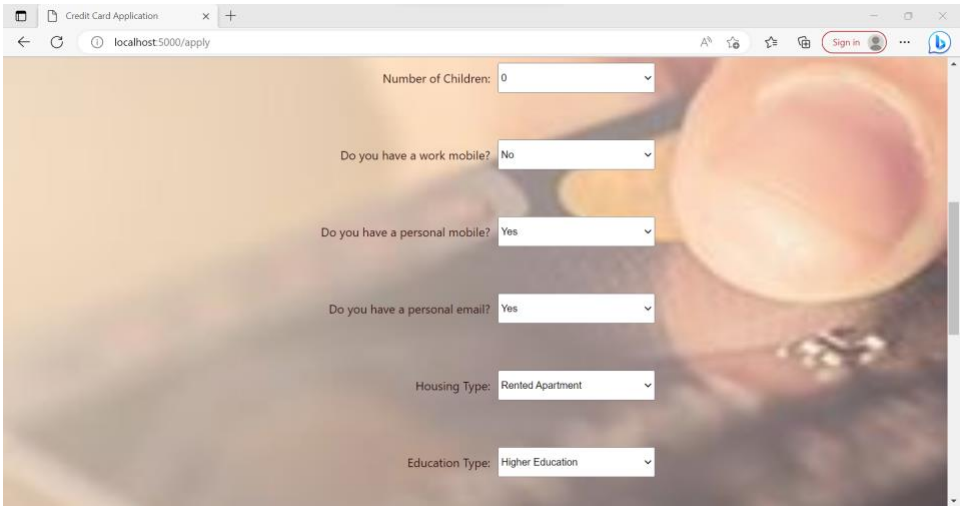
The credit card application form will collect various details from the user, such as their name, date of birth, family status, income, and employment. These details will be used as inputs to a model that will analyze the user's credit history. The credit history is a number that represents the user's creditworthiness, and it is used to determine whether to approve a credit card application.

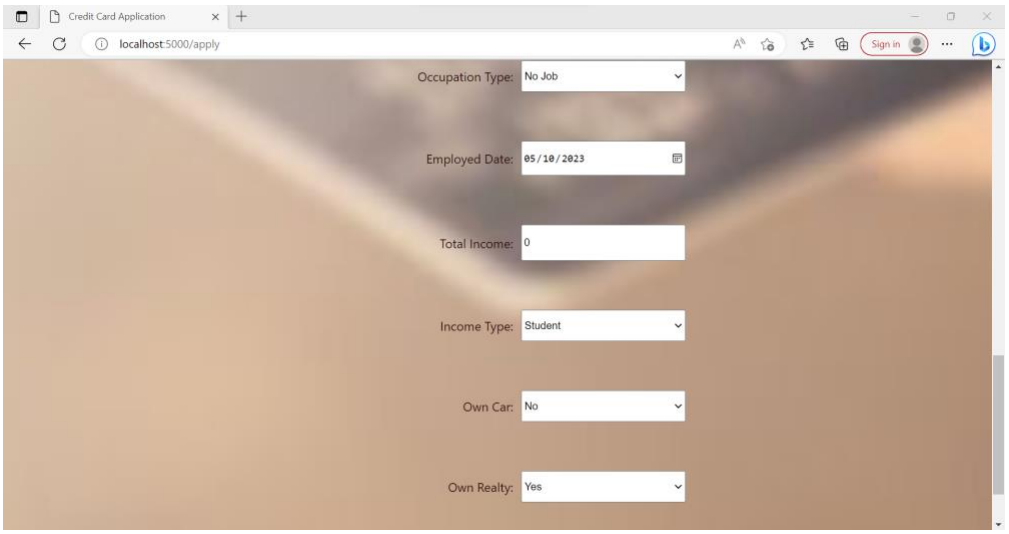
The model will use a variety of factors to calculate the user's credit history, including their payment history, debt-to-income ratio. The model will also consider the user's age, employment status, and other factors.

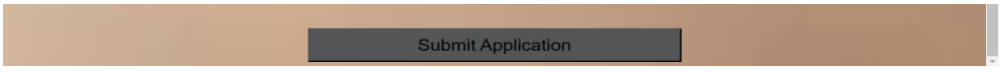
Once the model has analyzed the user's credit history, it will use the score to determine whether to approve the credit card application. If the user's credit history is good enough, it will approve the application and issue the user a credit card. If the user's credit history is too bad, it will deny the application.

The credit card application form is an important part of the credit card application process. The form collects the information that is needed to calculate the user's credit history, and it is used to determine whether to approve a credit card application.


Credit Card Application
localhost:5000/apply
Sign in
Apply for a Credit Card
VITE
Applicant Name: Almo
Gender: Female
Date of Birth: 11/13/2000
Marital Status: Single / Not Married
Total number of people in the family: 1


Credit Card Application
localhost:5000/apply
Sign in
Number of Children: 0
Do you have a work mobile? No
Do you have a personal mobile? Yes
Do you have a personal email? Yes
Housing Type: Rented Apartment
Education Type: Higher Education

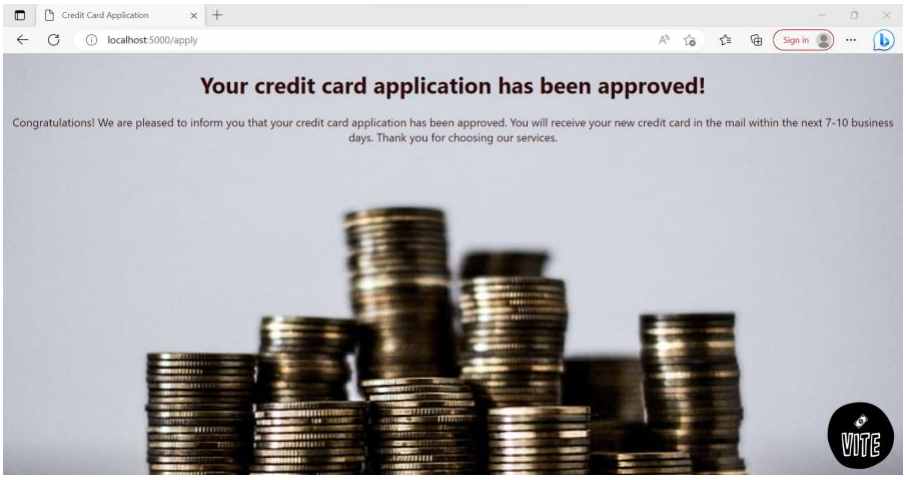

Credit Card Application
localhost:5000/apply
Sign in
Occupation Type: No Job
Employed Date: 05/10/2023
Total Income: 0
Income Type: Student
Own Car: No
Own Realty: Yes


Submit Application

On Approval

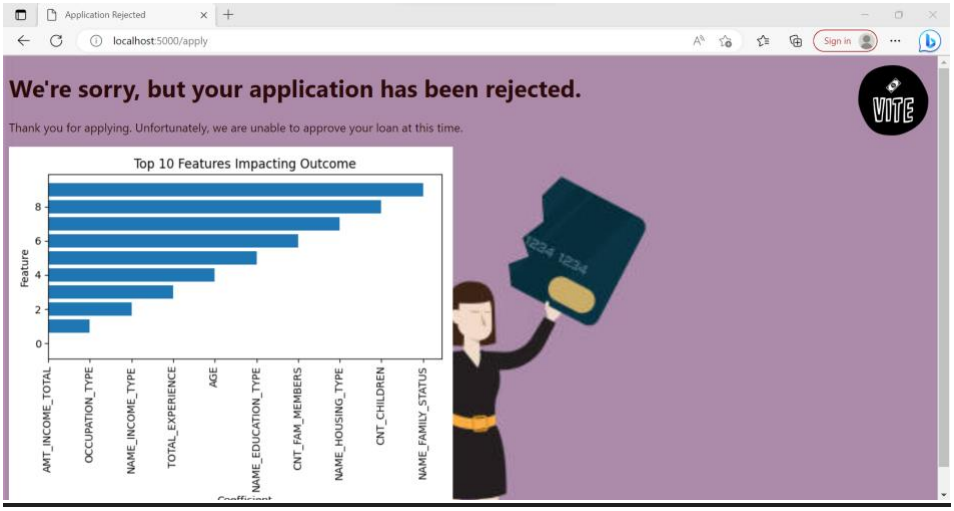
Once the credit card application is approved, the user will be redirected to a webpage that displays an approved message. The message will include the following information:

- A congratulations message.
- Confirmation that the application has been approved.
- An estimated time frame for when the user will receive their new credit card in the mail.
- A thank you message.

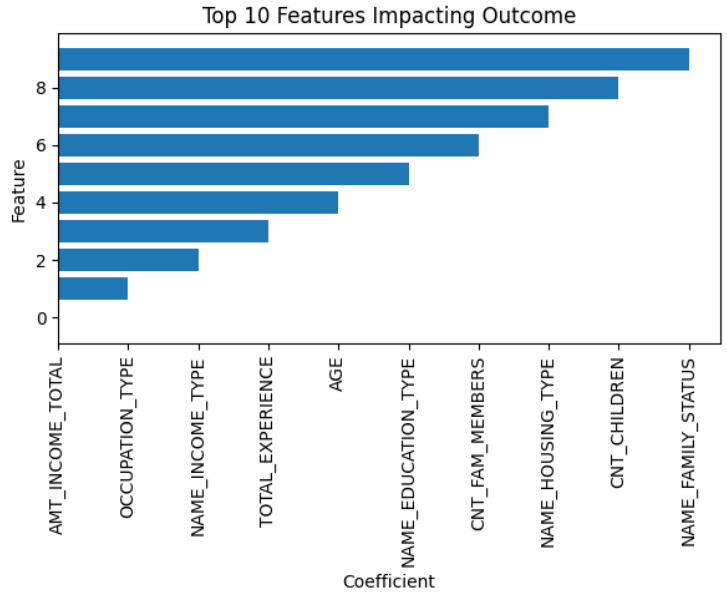


On Rejection

Once the credit card application is rejected, the user will be redirected to a webpage that displays *"we're sorry, but your application has been rejected. Thank you for applying. Unfortunately, we are unable to approve your loan currently."*



Visualization



The visualization component developed for the rejection page of our credit card application website provides users with valuable insights into the parameters that impacted their application rejection. The clear and intuitive presentation of the parameters and their relative importance helps users understand the decision-making process and make more informed choices. Developing this visualization involved careful analysis of the dataset and assigning appropriate weights to the parameters. The visualization enhances the user experience and supports users in their credit card application journey by providing actionable information for improvement or alternative options.

REFERENCES:

- https://pandas.pydata.org/
- https://seaborn.pydata.org/
- https://matplotlib.org/
- https://numpy.org/
- https://matplotlib.org/stable/gallery/images_contours_and_fields/image_annotated_heatmap.html
- https://seaborn.pydata.org/generated/seaborn.violinplot.html
- https://www.interviewqs.com/ddi-code-snippets/nan-replace-zero
- train_test_split
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
- Logistic Regression
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
- Accuracy score
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html
- Confusion matrix
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html
- Decision Tree Classifier
https://www.datacamp.com/tutorial/decision-tree-classification-python
- Precision and Recall
https://www.askpython.com/python/examples/precision-and-recall-in-python
- Support vector machine
https://scikit-learn.org/stable/modules/svm.html
- Pair plot
https://seaborn.pydata.org/tutorial/regression.html
- Random Forest
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
- Classification report
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html
- Gradient Boosting Machines
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html
- ROC Curve
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.RocCurveDisplay.html

- https://medium.com/geekculture/part-2-end-to-end-machine-learning-model-deployment-using-flask-a73c977221ee
- https://www.geeksforgeeks.org/deploy-machine-learning-model-using-flask/