Predicting Credit Card Approvals using ML Techniques

Project Description

Commercial banks receive a lot of applications for credit cards. Many of them get rejected for many reasons, like high loan balances, low-income levels, or too many inquiries on an individual’s credit report, for example. Manually analysing these applications is mundane, error-prone, and time-consuming (and time is money!). Fortunately, this task can be automated with the power of machine learning, and pretty much every commercial bank does so nowadays. In this project, we will build an automatic credit card approval predictor using machine learning techniques, just like real banks do.

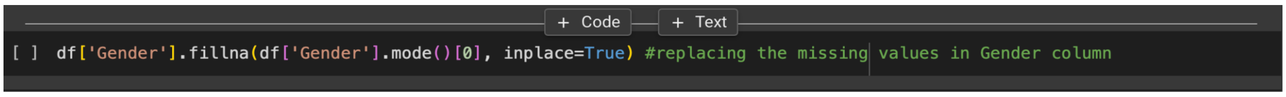
Approach

- We’ll start by loading and viewing the dataset.
- To manipulate data, if there are any missing entries in the dataset.
- To perform exploratory data analysis (EDA) on our dataset.
- To pre-process data before applying machine learning models to the dataset.
- To apply machine learning models that can predict if an individual’s application for a credit card will be accepted or not.

Data analysis approach

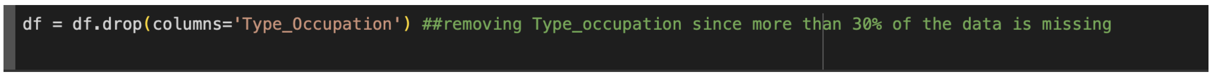
What feature engineering techniques will be relevant to your project?

- I've used an imputation technique to handle missing values.



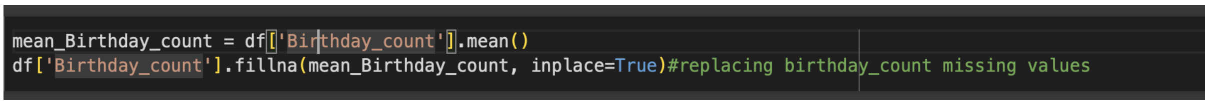
```
[ ]  df['Gender'].fillna(df['Gender'].mode()[0], inplace=True) #replacing the missing values in Gender column
```

- Also dropped column having more than 30% missing data.

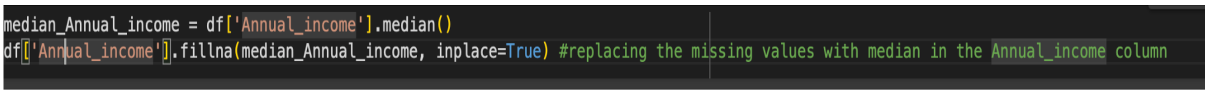


```
df = df.drop(columns='Type_Occupation') ##removing Type_occupation since more than 30% of the data is missing
```

- Used statistical methods to replace missing values

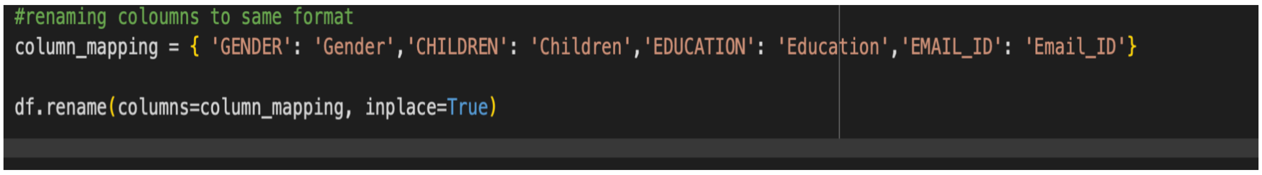


```
mean_Birthday_count = df['Birthday_count'].mean()
df['Birthday_count'].fillna(mean_Birthday_count, inplace=True)#replacing birthday_count missing values
```



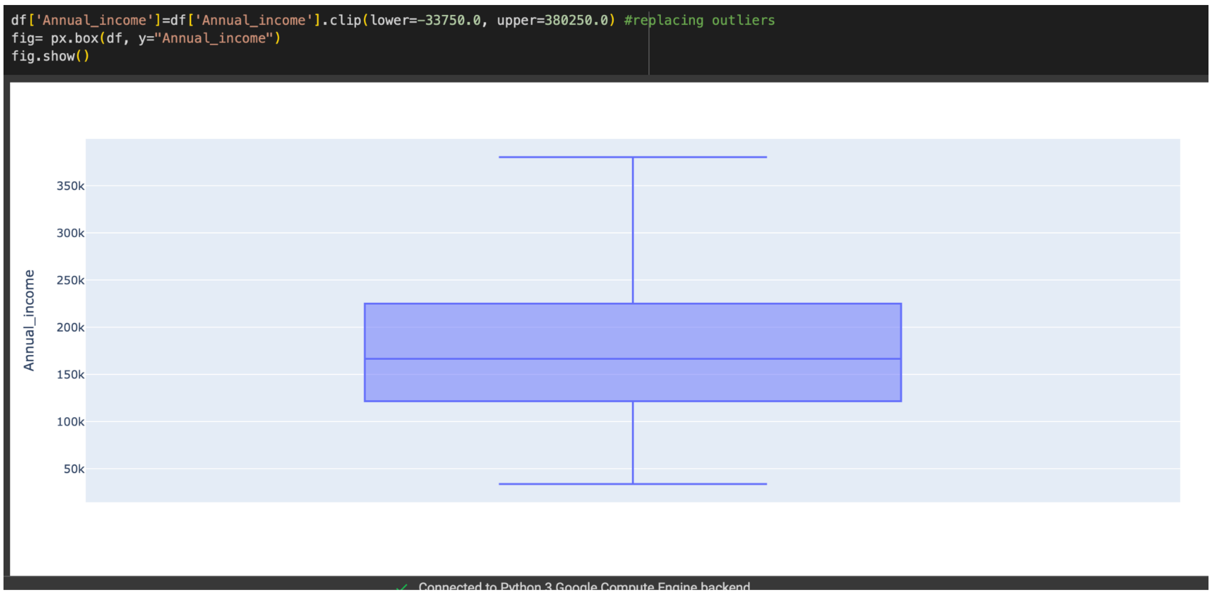
```
median_Annual_income = df['Annual_income'].median()
df['Annual_income'].fillna(median_Annual_income, inplace=True) #replacing the missing values with median in the Annual_income column
```

- Renamed column names to same format



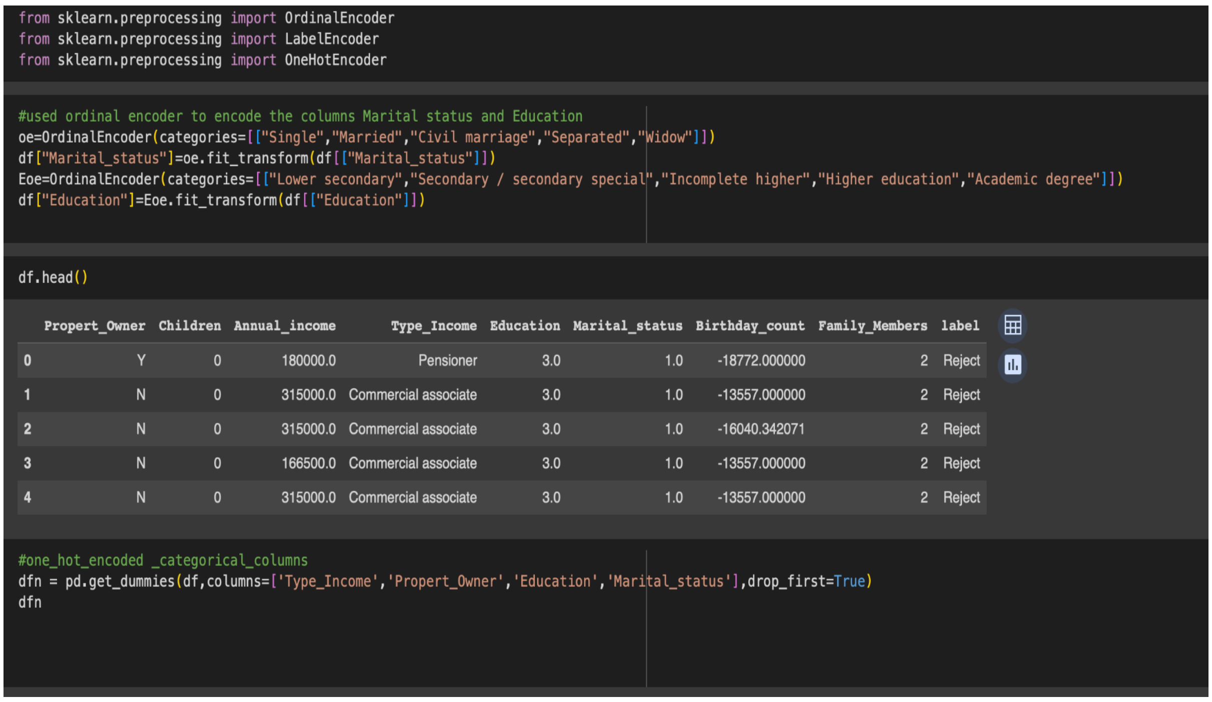
```
#renaming coloumns to same format
column_mapping = { 'GENDER': 'Gender','CHILDREN': 'Children','EDUCATION': 'Education','EMAIL_ID': 'Email_ID'}

df.rename(columns=column_mapping, inplace=True)
```

- **Replaced Outliers**



```
df['Annual_income']=df['Annual_income'].clip(lower=-33750.0, upper=380250.0) #replacing outliers
fig= px.box(df, y="Annual_income")
fig.show()
```

- **Did ordinal and one hot encoding to convert categorical variables to binary vectors**



```
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
```

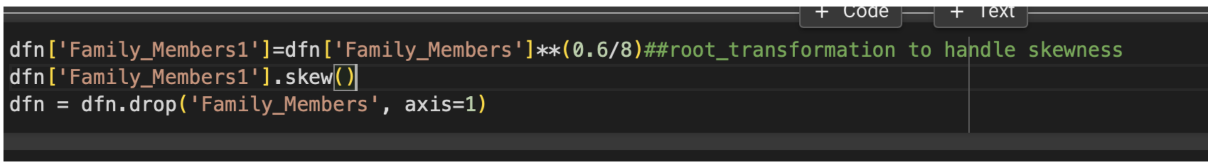
```
#used ordinal encoder to encode the columns Marital status and Education
oe=OrdinalEncoder(categories=[["Single","Married","Civil marriage","Separated","Widow"]])
df["Marital_status"]=oe.fit_transform(df[["Marital_status"]])
Eoe=OrdinalEncoder(categories=[["Lower secondary","Secondary / secondary special","Incomplete higher","Higher education","Academic degree"]])
df["Education"]=Eoe.fit_transform(df[["Education"]])
```

```
df.head()
```

```
   Propert_Owner  Children  Annual_income           Type_Income  Education  Marital_status  Birthday_count  Family_Members   label
0              Y         0       180000.0             Pensioner        3.0             1.0   -18772.000000               2  Reject
1              N         0       315000.0  Commercial associate        3.0             1.0   -13557.000000               2  Reject
2              N         0       315000.0  Commercial associate        3.0             1.0   -16040.342071               2  Reject
3              N         0       166500.0  Commercial associate        3.0             1.0   -13557.000000               2  Reject
4              N         0       315000.0  Commercial associate        3.0             1.0   -13557.000000               2  Reject
```

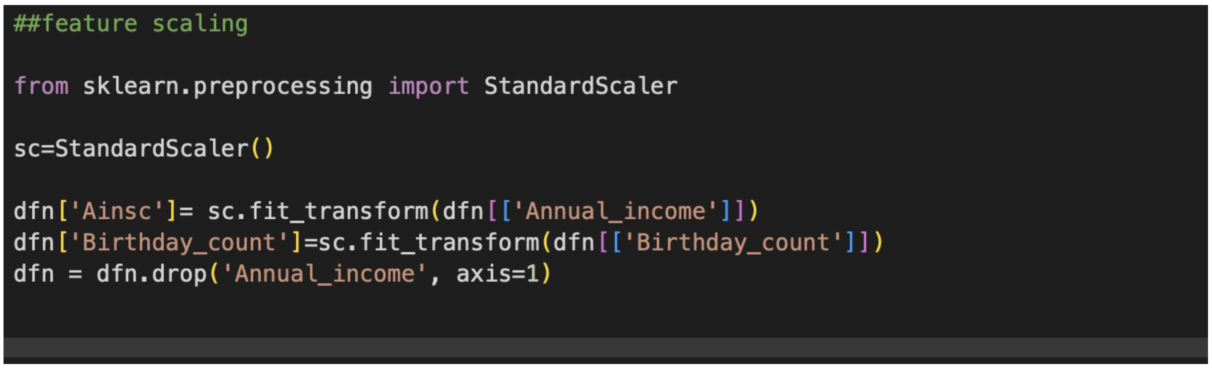
```
#one_hot_encoded _categorical_columns
dfn = pd.get_dummies(df,columns=['Type_Income','Propert_Owner','Education','Marital_status'],drop_first=True)
dfn
```

- **Did Feature transformation to handle skewed distribution.**



```
dfn['Family_Members1']=dfn['Family_Members']**(0.6/8)##root_transformation to handle skewness
dfn['Family_Members1'].skew()
dfn = dfn.drop('Family_Members', axis=1)
```

- **Scaled numeric Features using standardisation technique**



```
##feature scaling

from sklearn.preprocessing import StandardScaler

sc=StandardScaler()

dfn['Ainsc']= sc.fit_transform(dfn[['Annual_income']])
dfn['Birthday_count']=sc.fit_transform(dfn[['Birthday_count']])
dfn = dfn.drop('Annual_income', axis=1)
```

Identify important patterns in your data using the EDA approach to justify your findings.

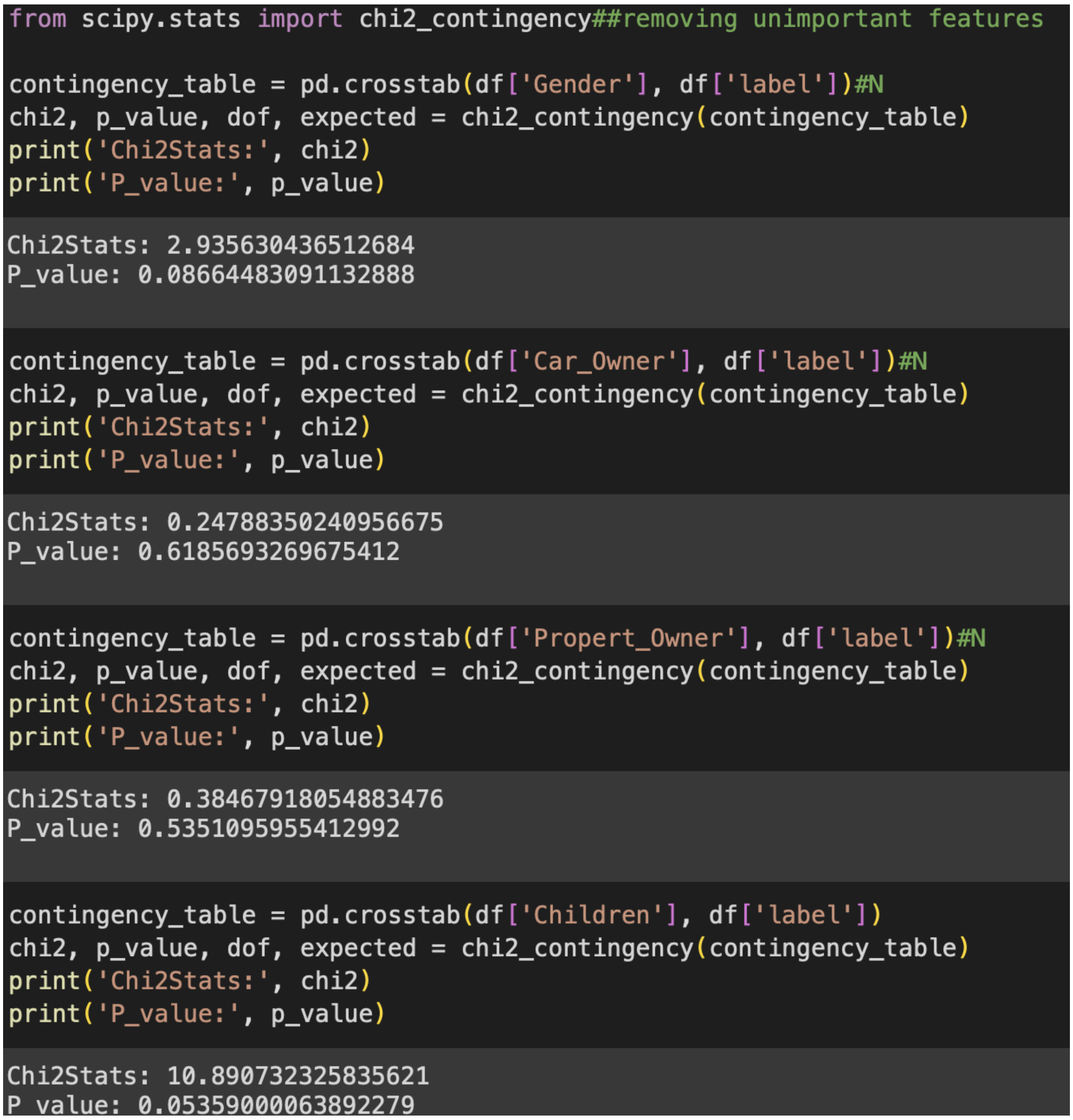
The data set contains 1548 rows with 18 columns. There were 2 columns with dtype: float 64 and 8 columns with dtype: int64 and 8 columns with dtype as object.

There were 4 columns with missing data out of which 1 column had more than 30% data missing.

There were few columns with outliers which had to be corrected statistically

Feature Selection

Used **Chi2** to find the relevance of the columns with the output variable and dropped the column where the criterion was not met.



```
from scipy.stats import chi2_contingency##removing unimportant features

contingency_table = pd.crosstab(df['Gender'], df['label'])#N
chi2, p_value, dof, expected = chi2_contingency(contingency_table)
print('Chi2Stats:', chi2)
print('P_value:', p_value)
```

```
Chi2Stats: 2.935630436512684
P_value: 0.08664483091132888
```

```
contingency_table = pd.crosstab(df['Car_Owner'], df['label'])#N
chi2, p_value, dof, expected = chi2_contingency(contingency_table)
print('Chi2Stats:', chi2)
print('P_value:', p_value)
```

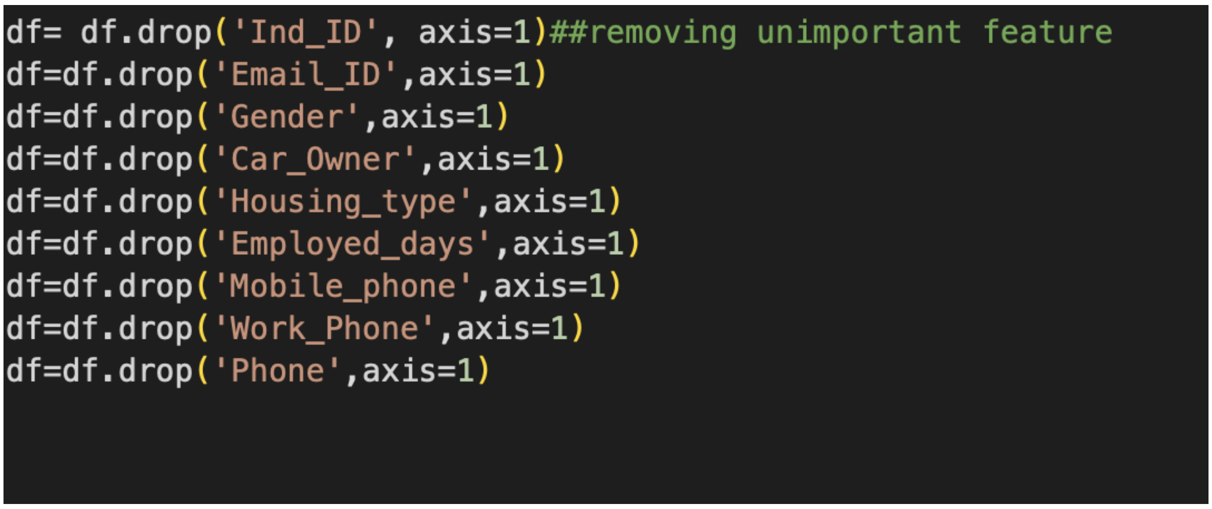
```
Chi2Stats: 0.24788350240956675
P_value: 0.6185693269675412
```

```
contingency_table = pd.crosstab(df['Propert_Owner'], df['label'])#N
chi2, p_value, dof, expected = chi2_contingency(contingency_table)
print('Chi2Stats:', chi2)
print('P_value:', p_value)
```

```
Chi2Stats: 0.38467918054883476
P_value: 0.5351095955412992
```

```
contingency_table = pd.crosstab(df['Children'], df['label'])
chi2, p_value, dof, expected = chi2_contingency(contingency_table)
print('Chi2Stats:', chi2)
print('P_value:', p_value)
```

```
Chi2Stats: 10.890732325835621
P_value: 0.05359000063892279
```



```
df= df.drop('Ind_ID', axis=1)##removing unimportant feature
df=df.drop('Email_ID',axis=1)
df=df.drop('Gender',axis=1)
df=df.drop('Car_Owner',axis=1)
df=df.drop('Housing_type',axis=1)
df=df.drop('Employed_days',axis=1)
df=df.drop('Mobile_phone',axis=1)
df=df.drop('Work_Phone',axis=1)
df=df.drop('Phone',axis=1)
```

What method will you use for machine learning based predictions for credit card approval?

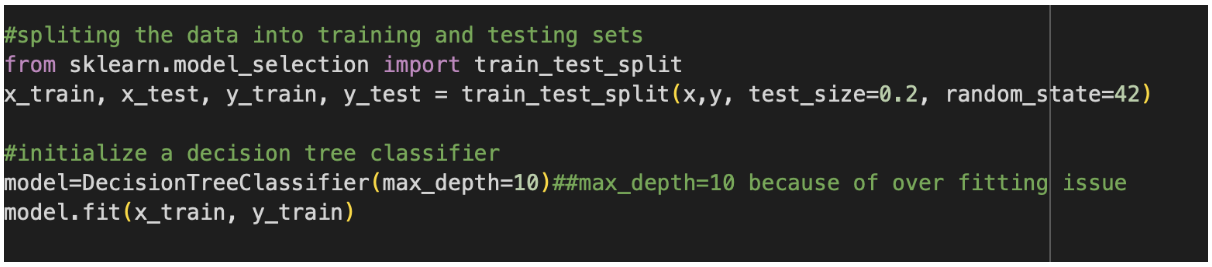
1. Please justify the most appropriate model.

   In this project I've used 5 of the most popular ML algorithms i.e **Logistic Regression, Decision Tree, Random Forest, Gradient Boost, KNN** to test and train the data.

   In my findings the **DecisionTree** and **RandomForest** performed best with 93% train Accuracy and 90% Test Accuracy.

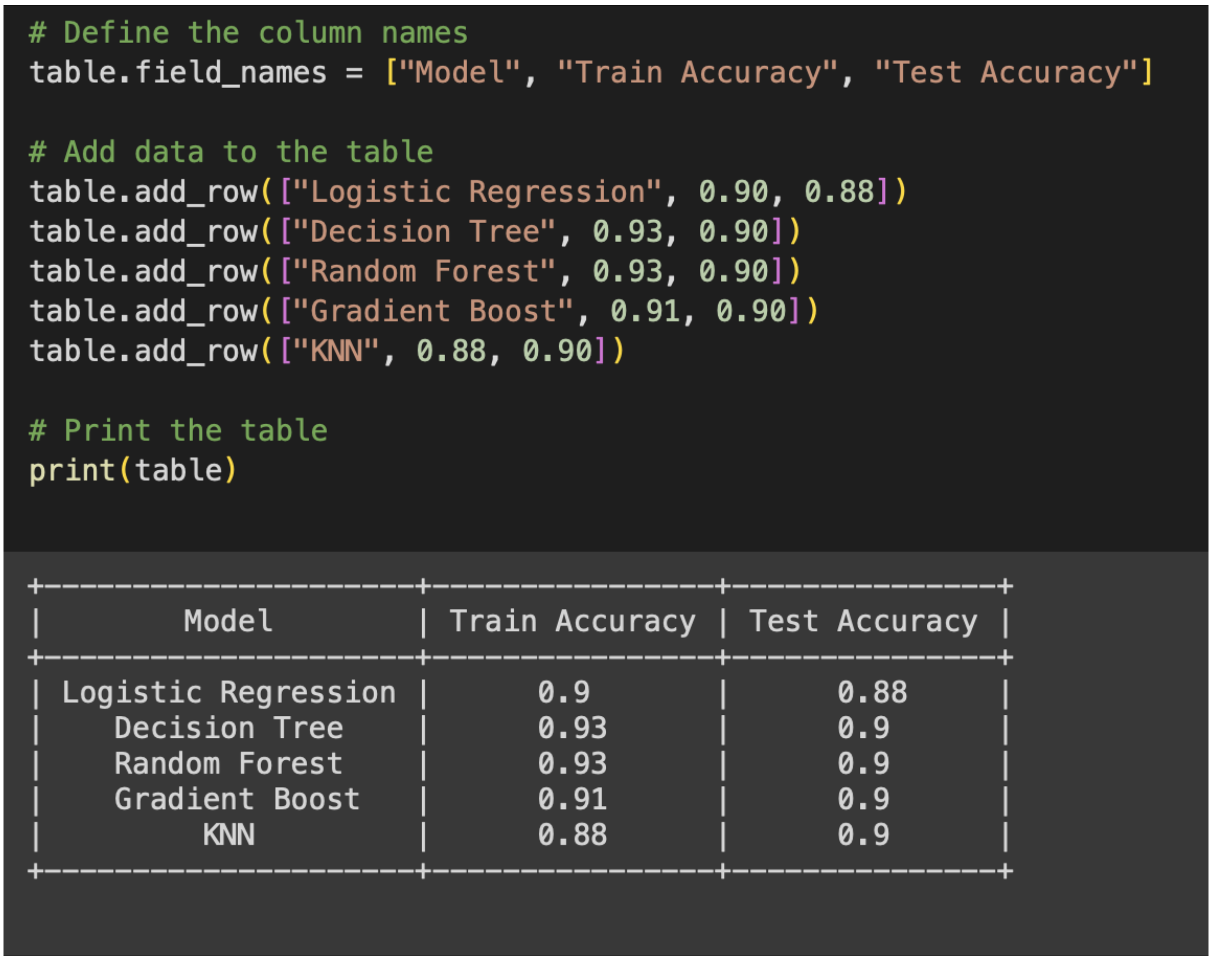
2. Please perform necessary steps required to improve the accuracy of your model.

   Used pruning technique to improve the result in the Tree based models.



```
#spliting the data into training and testing sets
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2, random_state=42)

#initialize a decision tree classifier
model=DecisionTreeClassifier(max_depth=10)##max_depth=10 because of over fitting issue
model.fit(x_train, y_train)
```

Please compare all models (at least 4 models).



```
# Define the column names
table.field_names = ["Model", "Train Accuracy", "Test Accuracy"]

# Add data to the table
table.add_row(["Logistic Regression", 0.90, 0.88])
table.add_row(["Decision Tree", 0.93, 0.90])
table.add_row(["Random Forest", 0.93, 0.90])
table.add_row(["Gradient Boost", 0.91, 0.90])
table.add_row(["KNN", 0.88, 0.90])

# Print the table
print(table)
```

```
+---------------------+----------------+---------------+
|        Model        | Train Accuracy | Test Accuracy |
+---------------------+----------------+---------------+
| Logistic Regression |      0.9       |      0.88     |
|    Decision Tree    |      0.93      |      0.9      |
|    Random Forest    |      0.93      |      0.9      |
|    Gradient Boost   |      0.91      |      0.9      |
|         KNN         |      0.88      |      0.9      |
+---------------------+----------------+---------------+
```

Conclusion

While building this credit card approval predictor model, I tackled some of the most widely-known pre-processing steps such as **scaling, label encoding,** and **missing value imputation.**
I then finished with some machine learning model to predict if a person's application for a credit card would get approved or not given some information about that person.
During the analysis I found out that Tree based models are best for predicting credit card approval.