pip install fancyimpute

credit\_card.shape

pip install scikit-learn

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

credit\_card = pd.read\_csv('/content/Credit\_card.csv')

credit\_card.head()

credit\_card.describe(include = 'all')

credit\_card.info()

credit\_card.columns

credit\_card.isnull().sum()

# checking if one observation has missing values more than 1

Missing\_I = credit\_card.isnull().sum(axis = 1)

Missing\_I = Missing\_I[Missing\_I > 1]

Missing\_I

# exploring all rows and columns

pd.options.display.max\_columns = None

pd.options.display.max\_rows = None

display(credit\_card)

# visualization of missing values

import missingno as msno

msno.matrix(credit\_card)

# Dropping the null values from target variable

credit\_card.drop(credit\_card[credit\_card['GENDER'].isnull()].index,axis=0, inplace=True)

credit\_card.info()

# Ordinal encoding for number of bedroom, bathroom and furnishing status

# we can also use regular expression too

from sklearn.preprocessing import OrdinalEncoder

Or\_enc = OrdinalEncoder()

credit\_card[["GENDER","Car\_Owner","Propert\_Owner",'EDUCATION','Type\_Income','Marital\_status','Housing\_type','Type\_Occupation']] = Or\_enc.fit\_transform(credit\_card[["GENDER","Car\_Owner", "Propert\_Owner",'EDUCATION','Type\_Income','Marital\_status','Housing\_type','Type\_Occupation']])

credit\_card.head()

# To avoid data leakage from outcome variable splitting the data set

Independent = credit\_card.drop('Ind\_ID',axis=1)

Independent.head()

Dependent = credit\_card['Mobile\_phone'] # creating Y variable only

Dependent.head()

credit\_card.columns

# Imputation using KNN

from fancyimpute import KNN

knn\_imputer = KNN()

Independent\_knn = Independent.copy(deep=True)

Independent\_knn.iloc[:, :] = knn\_imputer.fit\_transform(Independent\_knn)

# Imputation using MICE

from sklearn.impute import IterativeImputer

MICE\_imputer = IterativeImputer()

Independent\_MICE = Independent.copy(deep=True)

Independent\_MICE.iloc[:, :] = MICE\_imputer.fit\_transform(Independent\_MICE)

fig, ax =plt.subplots(1,3)

sns.histplot(credit\_card['Type\_Occupation'],bins=20, color="purple", ax=ax[0])

sns.histplot(Independent\_knn['Type\_Occupation'], color="red", bins=20, ax=ax[1])

sns.histplot(Independent\_MICE['Type\_Occupation'], color="green", bins=20, ax=ax[2])

fig, ax =plt.subplots(1,3)

sns.boxplot(credit\_card['Type\_Occupation'], color="purple", ax=ax[0])

sns.boxplot(Independent\_knn['Type\_Occupation'], color="red", ax=ax[1])

sns.boxplot(Independent\_MICE['Type\_Occupation'], color="green", ax=ax[2])

Independent\_knn.describe() # using describe function to see if there is any discrepency in numerical measuresIndependent\_knn.describe() # using describe function to see if there is any discrepency in numerical measures

Independent\_MICE.describe() # using describe function to see if there is any discrepency in numerical measures

credit\_card\_miss = pd.concat([Independent\_knn, Dependent], axis=1) # concatenating independent and dependent variable

credit\_card\_miss.head(10)

# Check for missing values

missing\_values = X\_vif.isnull().sum()

# Check for infinite values

infinite\_values = X\_vif.isin([np.inf, -np.inf]).sum()

# Example: Removing rows with missing values

X\_vif = X\_vif.dropna()

# Example: Imputing missing values (use a suitable imputation method)

X\_vif.fillna(X\_vif.mean(), inplace=True)

# Example: Replacing infinite values with NaN

X\_vif = X\_vif.replace([np.inf, -np.inf], np.nan)

# Example: Removing rows with infinite values

X\_vif = X\_vif[~X\_vif.isin([np.inf, -np.inf]).any(1)]

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

import pandas as pd

# VIF dataframe

vif\_data = pd.DataFrame()

vif\_data["feature"] = X\_vif.columns

# Calculating VIF for each feature

vif\_data["VIF"] = [variance\_inflation\_factor(X\_vif.values, i) for i in range(len(X\_vif.columns))]

print(vif\_data)

X\_vif = credit\_card.drop(['Ind\_ID','CHILDREN','Mobile\_phone','Family\_Members'], axis=1)

# VIF dataframe

vif\_data = pd.DataFrame()

vif\_data["feature"] = X\_vif.columns

# calculating VIF for each feature

vif\_data["VIF"] = [variance\_inflation\_factor(X\_vif.values, i)

for i in range(len(X\_vif.columns))]

print(vif\_data)

credit\_card.info

import pandas as pd

from sklearn.preprocessing import OrdinalEncoder

from sklearn.feature\_selection import SelectKBest, f\_regression

# Load the data

credit\_card = pd.read\_csv('/content/Credit\_card.csv')

# Remove rows with missing values

credit\_card.dropna(inplace=True)

# Create the features

features = ['Ind\_ID','CHILDREN','Mobile\_phone','Family\_Members']

# Create the target variable

target = 'Ind\_ID'

# Preprocess the categorical features using OrdinalEncoder

encoder = OrdinalEncoder()

credit\_card[features] = encoder.fit\_transform(credit\_card[features])

# Create the SelectKBest object

selector = SelectKBest(score\_func=f\_regression, k='all')

# Fit the selector to the data

selector.fit(credit\_card[features],credit\_card[target])

# Get the selected features

selected\_features\_mask = selector.get\_support()

# Print the selected features

selected\_features = [features[i] for i, selected in enumerate(selected\_features\_mask) if selected]

print(selected\_features)

import pandas as pd

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error, r2\_score

# Load the dataset into a pandas DataFrame

credit\_card = pd.read\_csv('/content/Credit\_card.csv')

print(credit\_card.columns)

credit\_card.dropna(subset=['Annual\_income'],inplace=True)

# Separate the features('CHILDREN','Family\_Members','Annual\_income') and target variable(phone)

X = credit\_card[['CHILDREN','Family\_Members','Annual\_income']]

y = credit\_card['Phone']

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize a linear regression model

model = LinearRegression()

# Train the model on the training data

model.fit(X\_train,y\_train)

# Make predictions on the testing data

y\_pred = model.predict(X\_test)

# Evaluate the model performance

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

# Print the model's coefficients and evaluation metrics

print("Coefficients:", model.coef\_)

print("Intercept:", model.intercept\_)

print("Mean Squared Error:", mse)

print("R^2 Score:", r2)

import pandas as pd

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score

# Load the dataset into a pandas DataFrame

credit\_card = pd.read\_csv('/content/Credit\_card.csv')

print(credit\_card.columns)

credit\_card.dropna(subset=['Annual\_income'],inplace=True)

# Separate the features('CHILDREN','Family\_Members','Annual\_income') and target variable(phone)

X = credit\_card[['CHILDREN','Family\_Members','Annual\_income']]

y = credit\_card['Phone']

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize a logistic regression model

model = LogisticRegression()

# Train the model on the training data

model.fit(X\_train, y\_train)

# Make predictions on the testing data

y\_pred = model.predict(X\_test)

# Evaluate the model performance

accuracy = accuracy\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred)

recall = recall\_score(y\_test, y\_pred)

# Print the model's coefficients and evaluation metrics

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall:", recall)

print("Coefficients:", model.coef\_)

print("Intercept:", model.intercept\_)

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score

# Load the dataset into a pandas DataFrame

credit\_card = pd.read\_csv('/content/Credit\_card.csv')

print(credit\_card.columns)

credit\_card.dropna(subset=['Annual\_income'],inplace=True)

# Separate the features('CHILDREN','Family\_Members','Annual\_income') and target variable(phone)

X = credit\_card[['CHILDREN','Family\_Members','Annual\_income']]

y = credit\_card['Phone']

# Split the data into training and testing sets

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()

# Train the model on the training data

model.fit(X\_train, y\_train)

# Make predictions on the testing data

y\_pred = model.predict(X\_test)

# Evaluate the model performance

accuracy = accuracy\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred)

recall = recall\_score(y\_test, y\_pred)

# Print the evaluation metrics

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall:", recall)

import pandas as pd

import numpy as np

from sklearn.ensemble import RandomForestRegressor

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

# Load the dataset into a pandas DataFrame

credit\_card = pd.read\_csv('/content/Credit\_card.csv')

print(credit\_card.columns)

credit\_card.dropna(subset=['Annual\_income'],inplace=True)

# Separate the features('CHILDREN','Family\_Members','Annual\_income') and target variable(phone)

X = credit\_card[['CHILDREN','Family\_Members','Annual\_income']]

y = credit\_card['Phone']

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize a Random Forest regression model

model = RandomForestRegressor()

# Train the model on the training data

model.fit(X\_train, y\_train)

# Make predictions on the testing data

y\_pred = model.predict(X\_test)

# Calculate evaluation metrics

mse = mean\_squared\_error(y\_test, y\_pred)

mae = mean\_absolute\_error(y\_test, y\_pred)

# Print the evaluation metrics

print("Mean Squared Error:", mse)

print("Mean Absolute Error:", mae)

print('RMSE :',np.sqrt(mse))

print(np.mean(y) \* 0.1)

import pandas as pd

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model\_selection import cross\_val\_score

# Load the dataset into a pandas DataFrame

credit\_card = pd.read\_csv('/content/Credit\_card.csv')

print(credit\_card.columns)

credit\_card.dropna(subset=['Annual\_income'],inplace=True)

# Separate the features('CHILDREN','Family\_Members','Annual\_income') and target variable(phone)

X = credit\_card[['CHILDREN','Family\_Members','Annual\_income']]

y = credit\_card['Phone']

# Initialize a KNN classifier

model = KNeighborsClassifier()

# Perform 10-fold cross-validation

scores = cross\_val\_score(model, X, y, cv=10)

# Print the cross-validation scores

print("Cross-Validation Scores:", scores)

print("Average Score:", scores.mean())