Names	IDs
yahia ashraf	20200636
yahia mahmoud	20201222
hamza abdel hamid	20200162
ziad ibrahim	20200193
omar tarek	20200348

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
```

```
data = pd.read_csv('/content/drive/MyDrive/loan_old.csv')
data
```

	Loan_ID	Gender	Married	Dependents	Education	Income	Coapplicant_Income	Loan_Tenor	Credit_History	Property_Area	Max_Loan_A
0	LP001002	Male	No	0	Graduate	5849	0.0	144.0	1.0	Urban	
1	LP001003	Male	Yes	1	Graduate	4583	1508.0	144.0	1.0	Rural	2
2	LP001005	Male	Yes	0	Graduate	3000	0.0	144.0	1.0	Urban	
3	LP001006	Male	Yes	0	Not Graduate	2583	2358.0	144.0	1.0	Urban	1
4	LP001008	Male	No	0	Graduate	6000	0.0	144.0	1.0	Urban	2
...	...	...	...	...	...	...	...	...	...	...	
609	LP002978	Female	No	0	Graduate	2900	0.0	144.0	1.0	Rural	
610	LP002979	Male	Yes	3+	Graduate	4106	0.0	72.0	1.0	Rural	
611	LP002983	Male	Yes	1	Graduate	8072	240.0	144.0	1.0	Urban	3
612	LP002984	Male	Yes	2	Graduate	7583	0.0	144.0	1.0	Urban	3
613	LP002990	Female	No	0	Graduate	4583	0.0	144.0	0.0	Semiurban	1

Data Analysis

```
data.isna().sum()
```

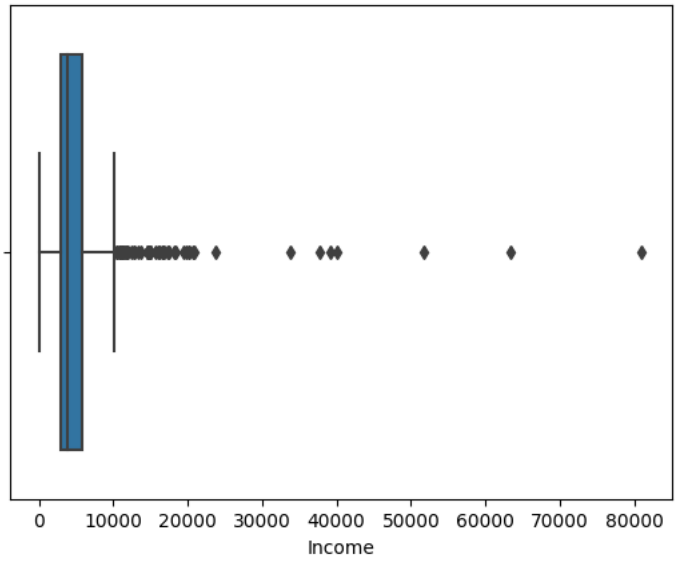
```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Income                0
Coapplicant_Income    0
Loan_Tenor           15
Credit_History       50
Property_Area         0
Max_Loan_Amount      25
Loan_Status           0
dtype: int64
```

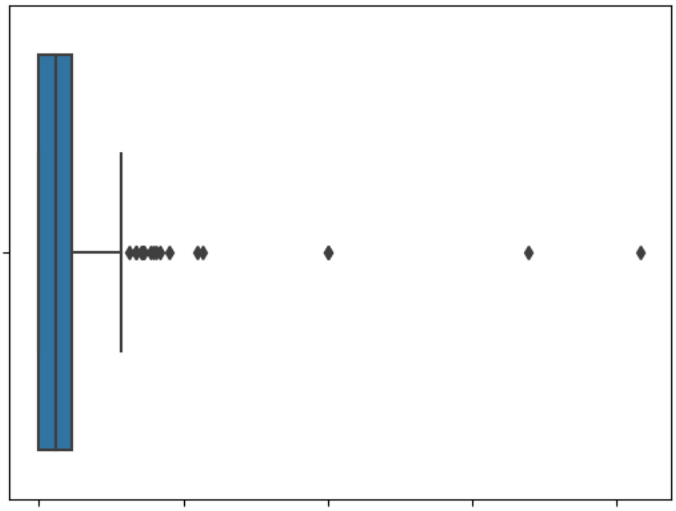
```
data.dtypes
```

```
Loan_ID               object
Gender                object
Married               object
Dependents            object
Education             object
Income                 int64
Coapplicant_Income   float64
Loan_Tenor           float64
Credit_History       float64
```

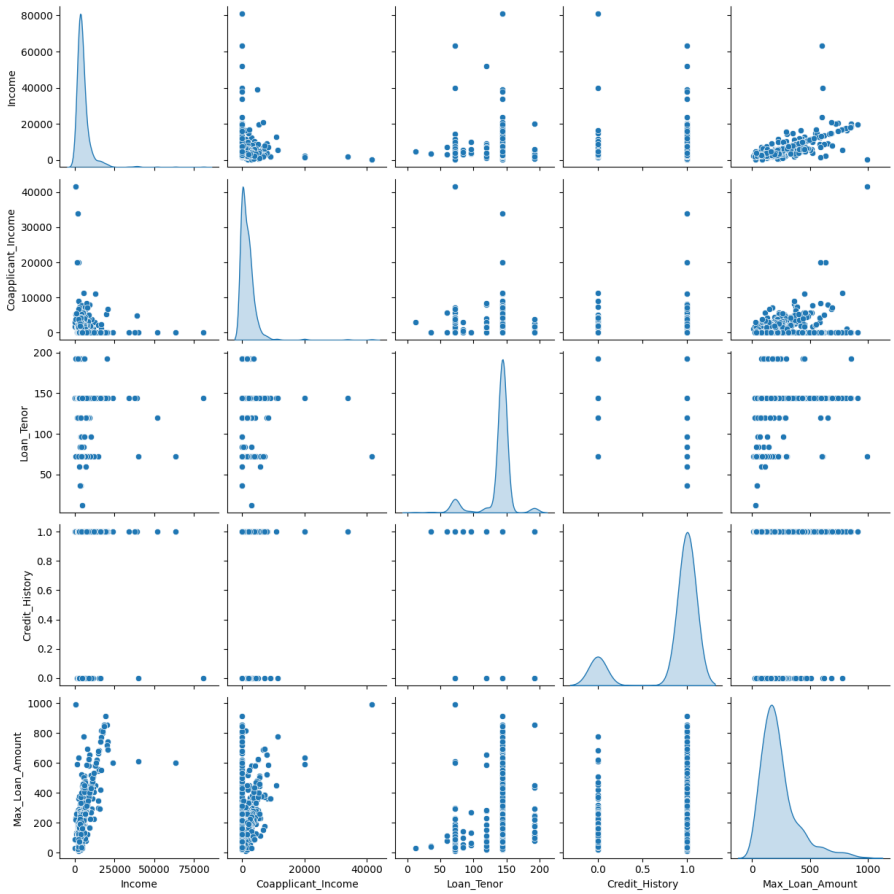
```
Property_Area          object
Max_Loan_Amount       float64
Loan_Status            object
dtype: object
```

```
for col in data:
  if data[col].dtype != 'object':
    sns.boxplot(data = data,x=col)
    plt.show()
```





```
sns.pairplot(data, diag_kind='kde')  # 'kde' for kernel density estimation
plt.show()
```



Preprocessing

```
def sanitize_data(path):
    data = pd.read_csv(path)
    # data.info()

    # dropping rows with missing values
    data.dropna(inplace=True)

    # replacing coapplicant_income with values 0 with the mean value
    data['Coapplicant_Income'] = data['Coapplicant_Income'].replace(0, data[data['Coapplicant_Income'] == 0]['Coapplicant_Income'].mean())

    # removing outliers in the income
    data.drop(data[data['Income']>25000].index,axis=0,inplace=True)

    return data


clean_data = sanitize_data("/content/drive/MyDrive/loan_old.csv")
clean_data
```

	Loan_ID	Gender	Married	Dependents	Education	Income	Coapplicant_Income	Loan_
1	LP001003	Male	Yes	1	Graduate	4583	1508.0	
2	LP001005	Male	Yes	0	Graduate	3000	0.0	
3	LP001006	Male	Yes	0	Not Graduate	2583	2358.0	

```
# separating targets and features
features = ['Gender', 'Married', 'Dependents', 'Education', 'Income', 'Coapplicant_Income', 'Loan_Tenor', 'Credit_History', 'Property_Area']
targets = ['Max_Loan_Amount']

x = clean_data[features]
y = clean_data[targets]

# shuffling and splitting data in testing and training sets
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)

# categorical features are encoded using one-hot encoding
categorical_columns = x.select_dtypes(include=['object']).columns
X_train_encoded = pd.get_dummies(X_train, columns=categorical_columns, drop_first=True)
X_test_encoded = pd.get_dummies(X_test, columns=categorical_columns, drop_first=True)

# numerical features are standardized
numerical_columns = X_train.select_dtypes(include=['float64', 'int64']).columns
scaler = StandardScaler()
X_train_encoded[numerical_columns] = scaler.fit_transform(X_train_encoded[numerical_columns])
X_test_encoded[numerical_columns] = scaler.transform(X_test_encoded[numerical_columns])
```

Linear Regression

```
def linear_train(X_train, y_train):
    model = LinearRegression()
    model.fit(X_train, y_train)
    return model


def linear_predict(model, x):
    predictions = model.predict(x)
    return predictions



def r2_evaluate(yhat, y):
    r2 = r2_score(yhat, y)
    print('r2 score for this linear regression model is', r2)


# load old data
# train model on train part of old data
model = linear_train(X_train_encoded, y_train)
# test model on test part of old data
predictions_test = linear_predict(model, X_test_encoded)
# evaluate the model
r2_evaluate(predictions_test, y_test)
```

```
r2 score for this linear regression model is 0.8830709822697137
```

```
#load new data and preprocess it (encoding and standardization)
new_data = sanitize_data("/content/drive/MyDrive/loan_new.csv")
X_new = new_data[features]
X_new = pd.get_dummies(X_new, columns=categorical_columns, drop_first=True)
X_new[numerical_columns] = scaler.transform(X_new[numerical_columns])
# predict for new data
predictions_new = linear_predict(model, X_new)
print(predictions_new)
```

```
[ 145.18947701]
[ 277.96633746]
[ 322.04823387]
[ 192.79888542]
[ 263.95693209]
[ 193.50102678]
[  70.25978301]
[ 200.01436622]
[ 154.93824526]
[ 152.28259857]
[ 203.75051286]
[ 161.02160319]
[ 102.82685972]
[  73.33248398]
[ 852.86432862]
[ 209.45270863]
[ 122.8891609 ]
[ 213.90920677]
[ 333.99514718]
[ 216.92041333]
[ 387.53679738]
[ 228.62916953]
[ 173.26420761]
[ 123.00379103]
[  67.3350113 ]
[ 132.18695431]
[ 124.05494315]
[ 232.88379239]
[ 140.25403571]
[ 309.09048841]
[  -6.93085992]
[ 179.46963204]
[ 266.64367928]
[ 329.61965579]
[ 196.2360654 ]
[  89.11600248]
[ 171.37606785]
[  53.72891259]
[ 349.09235147]
[ 256.75780122]
[ 345.7548415 ]
[  70.00059912]
[ 345.39244788]
[ 215.57824287]
[  73.40047045]
[ 248.50363799]
[ 168.49334542]
[ 214.75501751]
[ 178.61299625]
[ 287.6290075 ]
[ 225.20449016]]
```

Logistic Regression

```
clean_data = sanitize_data('/content/drive/MyDrive/loan_old.csv')

# separating targets and features
features = ['Gender', 'Married', 'Dependents', 'Education', 'Income', 'Coapplicant_Income', 'Loan_Tenor',
            'Credit_History', 'Property_Area']
targets = ['Loan_Status']

x = clean_data[features]
y = clean_data[targets]

# shuffling and splitting data in testing and training sets
X_train, X_test, y_train, y_test = train_test_split(
    x, y, test_size=0.3, random_state=0)

label_encoder = LabelEncoder()

# categorical data are encoded
for col in X_train.columns:
    if X_train[col].dtype == 'object':
        X_train[col] = label_encoder.fit_transform(X_train[col])

for col in y_train.columns:
    if y_train[col].dtype == 'object':
        y_train[col] = label_encoder.fit_transform(y_train[col])

for col in X_test.columns:
    if X_test[col].dtype == 'object':
        X_test[col] = label_encoder.fit_transform(X_test[col])

for col in y_test.columns:
    if y_test[col].dtype == 'object':
        y_test[col] = label_encoder.fit_transform(y_test[col])

# numerical features are standardized
numerical_columns = X_train.select_dtypes(include=['float64', 'int64']).columns
scaler = StandardScaler()
X_train[numerical_columns] = scaler.fit_transform(X_train[numerical_columns])
X_test[numerical_columns] = scaler.transform(X_test[numerical_columns])


def get_z(x, w, b):
    return np.dot(x, w) + b


def get_sigmoid(z):
    return 1/(1+np.exp(-z))


def get_cost(y, predictions):
    m = len(y)
    cost = (-1/m) * np.sum(-y * np.log(predictions) -
                           (1-y)*np.log(1-predictions))
    return cost


def logistic_regression(x, y):

    x = x.to_numpy()
    y = y.to_numpy()

    m, features = x.shape[0], x.shape[1]
    # fx1
    w = np.zeros((features, 1))
    b = 0

    # gradient descent
    alpha = 0.001
    max_iterations = 100

    for i in range(max_iterations):
        # mxf . fx1 = mx1
        z = get_z(x, w, b)
        predictions = get_sigmoid(z)

        # fxm . mx1 = fx1
        dw = (1/m) * np.dot(x.T, predictions - y)
        db = (1/m) * np.sum(predictions - y)

        new_w = w - alpha * dw
```

```
            new_b = b - alpha * db
            w = new_w
            b = new_b

    return w, b


def predict(x, w, b):
    # x . w
    # mxf . fx1

    z_hat = get_z(x, w, b)
    sigmoid_hat = get_sigmoid(z_hat)
    threshold = 0.5
    y_hat = (sigmoid_hat > threshold).astype(int)
    return y_hat


def model_accuracy(y, y_hat):
    m = y.shape[0]
    wrong_predictions = np.sum(np.abs(y - y_hat))
    return (m - wrong_predictions) / m


w, b = logistic_regression(X_train, y_train)

predictions = predict(X_test, w, b)

accuracy = model_accuracy(predictions, y_test)

print("model accuracy is " + str(accuracy), end="")
```

```
model accuracy is Loan_Status    0.75974
dtype: float64
```

```
#load new data and preprocess it (encoding and standardization)
new_data = sanitize_data("/content/drive/MyDrive/loan_new.csv")
new_data.drop(columns='Loan_ID',inplace = True)
X_new = new_data

scaler = StandardScaler()
for col in X_new.columns:
    if X_new[col].dtype != 'object':
        X_new[col] = scaler.fit_transform(X_new[[col]])

for col in X_new.columns:
  if X_new[col].dtype == 'object':
    X_new[col] = label_encoder.fit_transform(X_new[col])

X_new
```

	Gender	Married	Dependents	Education	Income	Coapplicant_Income	Loan_Tenor	C
0	1	1	0	0	0.431977	-0.657881	0.252027	
1	1	1	1	0	-0.516183	-0.015406	0.252027	
2	1	1	2	0	0.173779	0.113089	0.252027	
4	1	0	0	1	-0.444461	-0.657881	0.252027	
5	1	1	0	1	-0.842875	0.807820	0.252027	
...	...	...	...	...	...	...	...	
361	1	1	1	0	-0.805580	0.270282	0.252027	
362	1	1	3	1	-0.181601	0.103238	0.252027	
363	1	1	0	0	-0.128169	-0.354204	0.252027	
365	1	1	0	0	0.173779	0.367081	0.252027	
366	1	0	0	0	1.679934	-0.657881	-2.875550	

313 rows × 9 columns

```
# predict for new data
predictions_new = predict(X_new, w, b)
print(predictions_new)
```

```
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[0]
[0]
[1]
[1]
[0]
[1]
[0]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[0]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[0]
[1]
[1]
[1]
[1]
[0]
[1]
[1]
[1]
[1]
[0]
[0]
[1]
```