Support Vector Machine

Loading the dataset

```
import pandas as pd
```

```
df = pd.read_excel("/content/train_svm.xlsx")
```

```
df1 = pd.read_excel("/content/test_svm.xlsx")
```

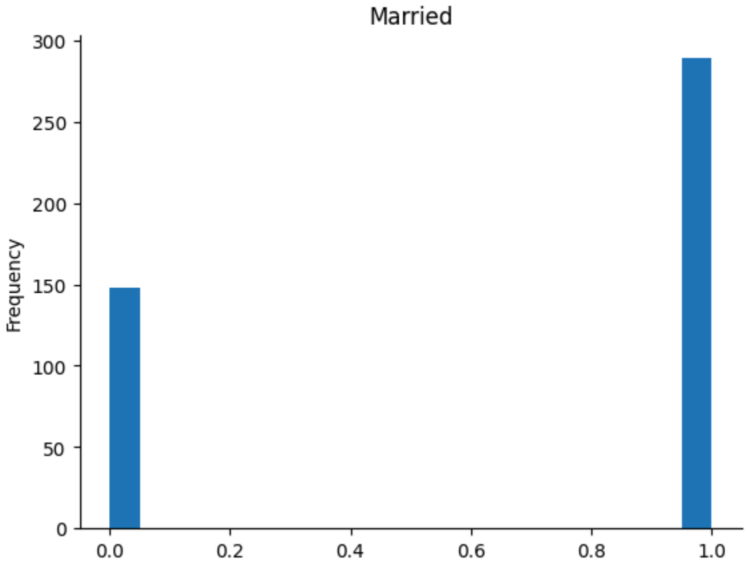
```
df.head()
```

	Loan_ID	Gender	Married	Dependents	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
0	1	1	0	0	0	5849	0	0	360
1	2	1	1	1	0	4583	1508	128	360
2	3	1	1	0	1	3000	0	66	360
3	4	1	1	0	0	2583	2358	120	360
4	5	1	0	0	0	6000	0	141	360

Next steps: View recommended plots

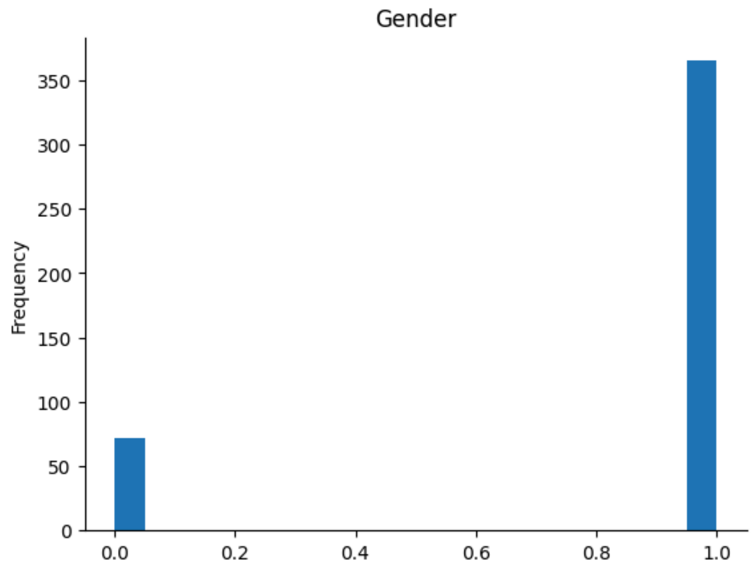
Married

Show code



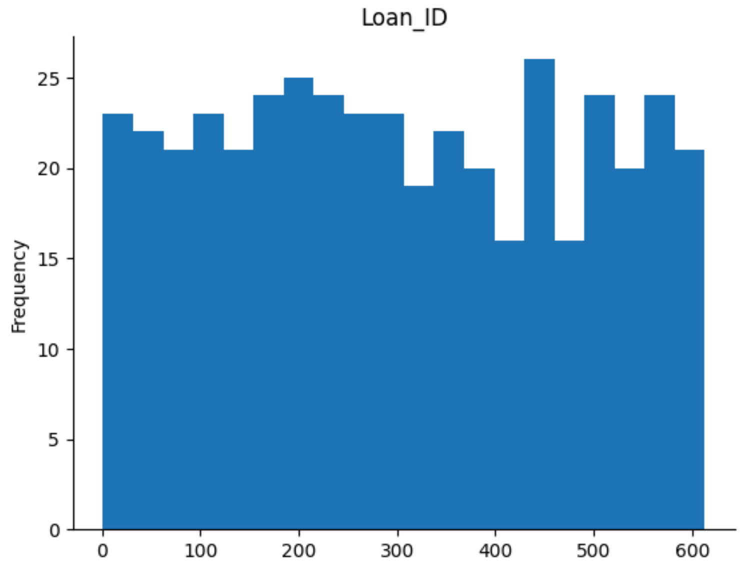
Gender

Show code



Loan_ID

Show code



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 437 entries, 0 to 436
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            437 non-null    int64
 1   Gender             437 non-null    int64
 2   Married            437 non-null    int64
 3   Dependents         437 non-null    int64
 4   Self_Employed      437 non-null    int64
 5   ApplicantIncome    437 non-null    int64
 6   CoapplicantIncome  437 non-null    int64
 7   LoanAmount         437 non-null    int64
 8   Loan_Amount_Term   437 non-null    int64
 9   Property_Area      437 non-null    int64
 10  Loan_Status        437 non-null    int64
dtypes: int64(11)
memory usage: 37.7 KB
```

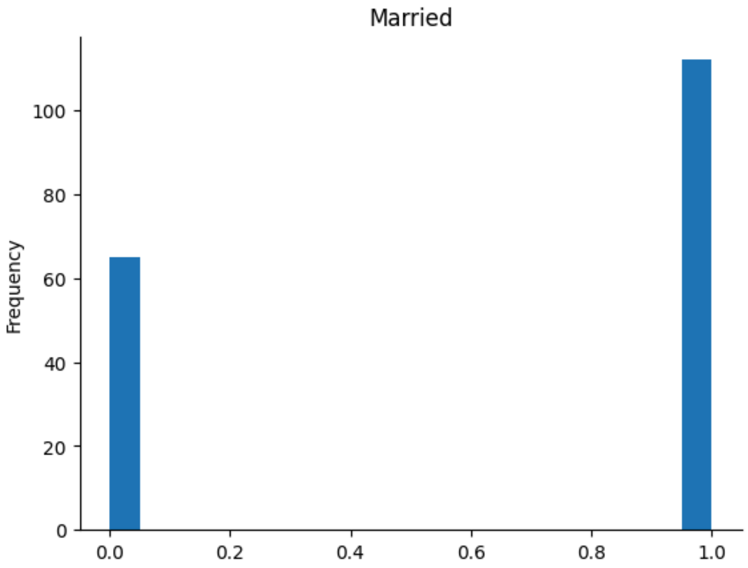
```
df1.head()
```

	Loan_ID	Gender	Married	Dependents	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
0	7	1	1	0	0	2333	1516.0	95	360
1	8	1	1	3	0	3036	2504.0	158	360
2	13	1	1	2	0	3073	8106.0	200	360
3	20	1	1	0	1	2600	3500.0	115	0
4	22	1	1	1	0	5955	5625.0	315	360

Next steps: View recommended plots

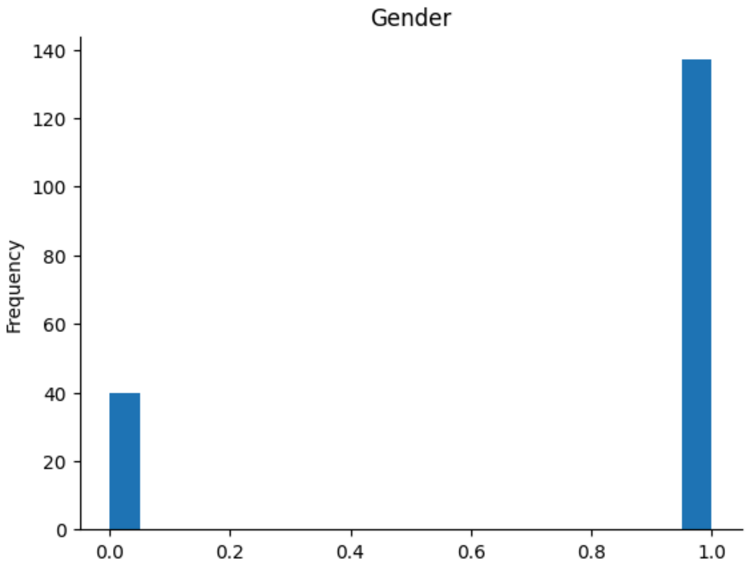
Married

Show code



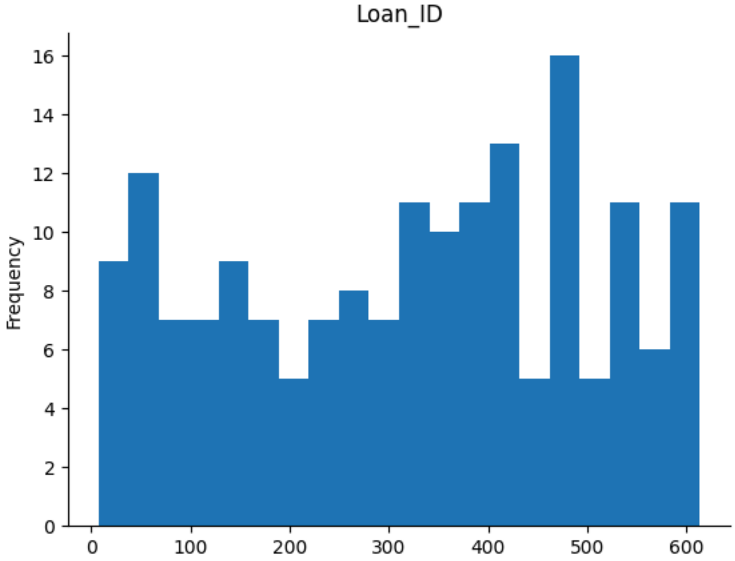
Gender

Show code



Loan_ID

Show code



```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 177 entries, 0 to 176
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            177 non-null    int64  
 1   Gender             177 non-null    int64  
 2   Married            177 non-null    int64  
 3   Dependents         177 non-null    int64  
 4   Self_Employed      177 non-null    int64  
 5   ApplicantIncome    177 non-null    int64  
 6   CoapplicantIncome  177 non-null    float64
 7   LoanAmount         177 non-null    int64  
 8   Loan_Amount_Term   177 non-null    int64  
 9   Property_Area      177 non-null    int64  
 10  Loan_Status        177 non-null    int64  
dtypes: float64(1), int64(10)
memory usage: 15.3 KB
```

Splitting the data our target variable is Loan_status

```
x_train = df.iloc[: , :-1]
```

```
x_train.head()
```

	Loan_ID	Gender	Married	Dependents	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
0	1	1	0	0	0	5849	0	0	360
1	2	1	1	1	0	4583	1508	128	360
2	3	1	1	0	1	3000	0	66	360
3	4	1	1	0	0	2583	2358	120	360
4	5	1	0	0	0	6000	0	141	360

Next steps: View recommended plots

```
y_train = df['Loan_Status']
```

```
y_train.head()
```

```
0    1
1    0
2    1
3    1
4    1
Name: Loan_Status, dtype: int64
```

```
y_train.shape
```

```
(437,)
```

```
x_train.shape
```

```
(437, 10)
```

```
x_test = df1.iloc[: , :-1]
```

```
x_test.shape
```

```
(177, 10)
```

```
y_test = df1['Loan_Status']
```

```
y_test.shape
```

```
(177,)
```

Generating the Model

```
#Import svm model
from sklearn import svm
from sklearn.model_selection import GridSearchCV
```

```
#Create a svm Classifier
clf = svm.SVC(kernel='linear') # Linear Kernel
```

```
#Train the model using the training sets
clf.fit(x_train, y_train)
```

```
▾       SVC
SVC(kernel='linear')
```

```
m1 = svm.SVC()
param_grid={'C': [1,10,100,1000,10000], 'gamma': [1,0.1,0.01,0.001,0.0001], 'kernel': ['rbf']}

grid = GridSearchCV(m1, param_grid, refit=True, verbose=1, cv=15)
grid_search=grid.fit(x_train,y_train)
```

```
Fitting 15 folds for each of 25 candidates, totalling 375 fits
```

```
print(grid_search.best_params_)
```

```
{'C': 1, 'gamma': 1, 'kernel': 'rbf'}
```

```
accuracy=grid_search.best_score_*100  #this is validation accuracy
print("Accuracy for training with tuning is : %0.2f" %(accuracy))
```

```
Accuracy for training with tuning is : 68.43
```

```
m1.fit(x_train,y_train)
y_pred=m1.predict(x_test)
```

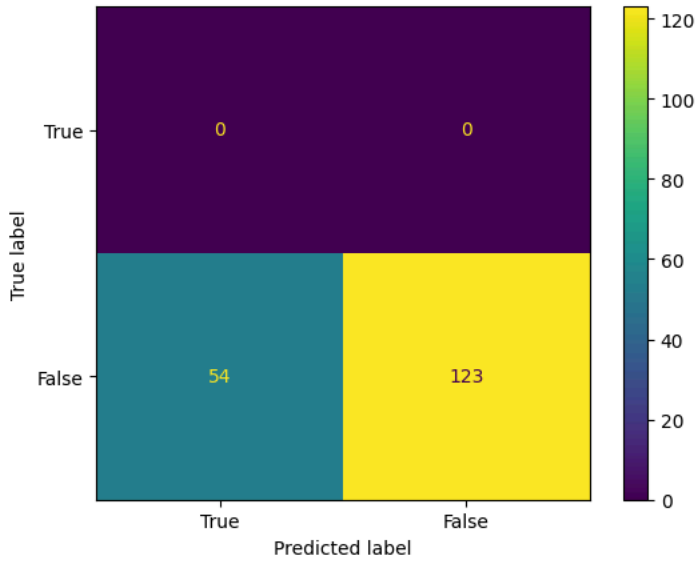
```
from sklearn.metrics import classification_report
cf=classification_report(y_pred,y_test)
print(cf)
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         0
           1       1.00      0.69      0.82       177

    accuracy                           0.69       177
   macro avg       0.50      0.35      0.41       177
weighted avg       1.00      0.69      0.82       177

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Recall and F-s
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Recall and F-s
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Recall and F-s
  _warn_prf(average, modifier, msg_start, len(result))
```

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm=confusion_matrix(y_pred,y_test)
print(cm)
```

```
[[  0   0]
```

```
cm_display= ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=[True, False])
cm_display.plot()
plt.show()
```



Training SVM withg kernel = Linear is much better approach

```
#Predict the response for test dataset
y_pred = clf.predict(x_test)
```

Evaluating the Model

```
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics

# Model Accuracy: how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7062146892655368
```

```
# Model Precision: what percentage of positive tuples are labeled as such?
print("Precision:",metrics.precision_score(y_test, y_pred))

# Model Recall: what percentage of positive tuples are labelled as such?
print("Recall:",metrics.recall_score(y_test, y_pred))
```

```
Precision: 0.7052023121387283
```