```
import pandas as pd
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from yellowbrick.classifier import ConfusionMatrix
from sklearn.metrics import accuracy_score, classification_report
```

```
df=pd.read_csv("/content/credit_risk_dataset.csv")
```

```
df.head()
```

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade	loan_amnt	loan_int_rate	loa
0	22	59000	RENT	123.0	PERSONAL	D	35000	16.02	
1	21	9600	OWN	5.0	EDUCATION	B	1000	11.14	
2	25	9600	MORTGAGE	1.0	MEDICAL	C	5500	12.87	
3	23	65500	RENT	4.0	MEDICAL	C	35000	15.23	
4	24	54400	RENT	8.0	MEDICAL	C	35000	14.27	

```
df.describe()
```

	person_age	person_income	person_emp_length	loan_amnt	loan_int_rate	loan_status	loan_percent_income	cb_person_c
count	32581.000000	3.258100e+04	31686.000000	32581.000000	29465.000000	32581.000000	32581.000000	
mean	27.734600	6.607485e+04	4.789686	9589.371106	11.011695	0.218164	0.170203	
std	6.348078	6.198312e+04	4.142630	6322.086646	3.240459	0.413006	0.106782	
min	20.000000	4.000000e+03	0.000000	500.000000	5.420000	0.000000	0.000000	
25%	23.000000	3.850000e+04	2.000000	5000.000000	7.900000	0.000000	0.090000	
50%	26.000000	5.500000e+04	4.000000	8000.000000	10.990000	0.000000	0.150000	
75%	30.000000	7.920000e+04	7.000000	12200.000000	13.470000	0.000000	0.230000	

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32581 entries, 0 to 32580
Data columns (total 12 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   person_age                  32581 non-null  int64
 1   person_income               32581 non-null  int64
 2   person_home_ownership       32581 non-null  object
 3   person_emp_length           31686 non-null  float64
 4   loan_intent                 32581 non-null  object
 5   loan_grade                  32581 non-null  object
 6   loan_amnt                   32581 non-null  int64
 7   loan_int_rate               29465 non-null  float64
 8   loan_status                 32581 non-null  int64
 9   loan_percent_income         32581 non-null  float64
 10  cb_person_default_on_file   32581 non-null  object
 11  cb_person_cred_hist_length  32581 non-null  int64
dtypes: float64(3), int64(5), object(4)
memory usage: 3.0+ MB
```

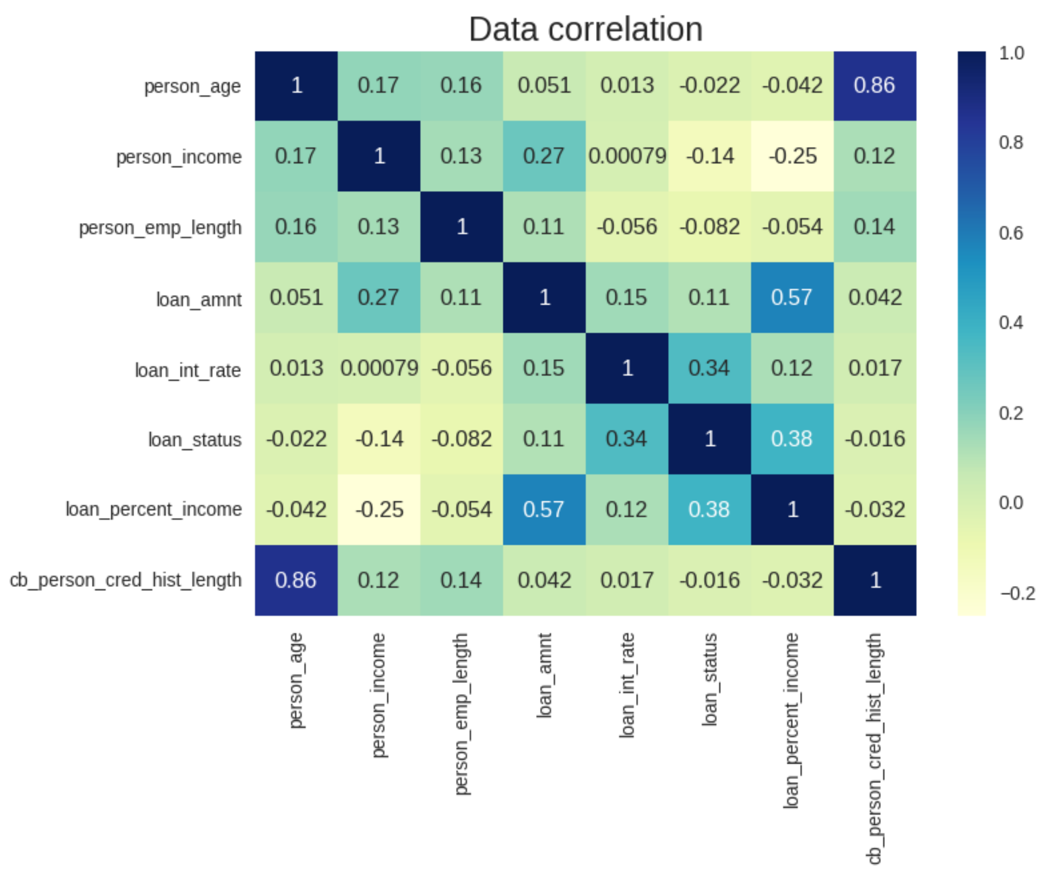
```
missing_values = df.isna().sum()
print(missing_values)
```

```
person_age                      0
person_income                   0
person_home_ownership           0
person_emp_length             895
loan_intent                     0
loan_grade                      0
loan_amnt                       0
loan_int_rate                3116
loan_status                     0
loan_percent_income             0
cb_person_default_on_file       0
cb_person_cred_hist_length      0
dtype: int64
```

```
1 df.corr(numeric_only=True)
```

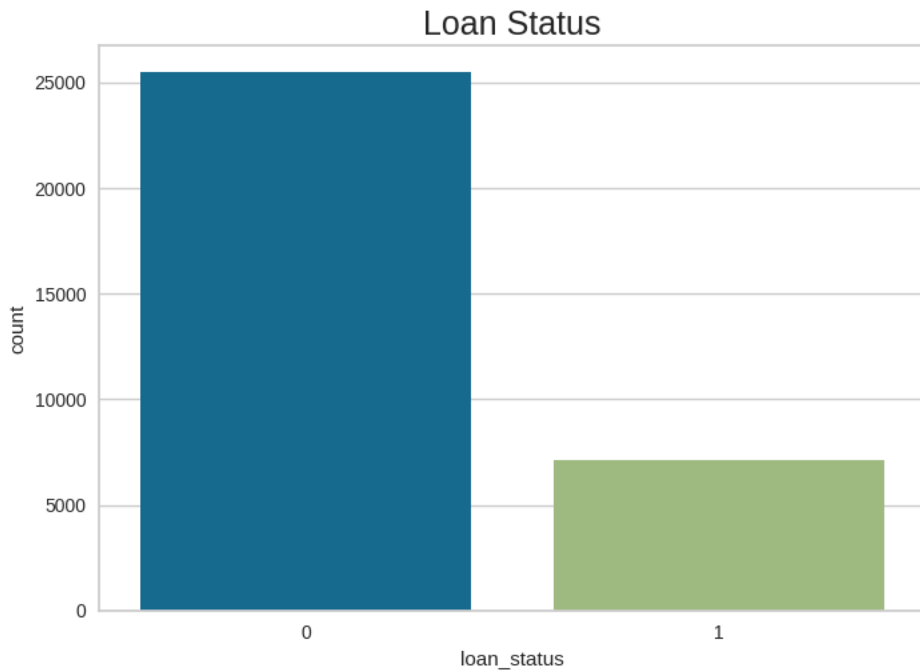
	person_age	person_income	person_emp_length	loan_amnt	loan_int_rate	loan_status	loan_percent_inc
person_age	1.000000	0.173202	0.163106	0.050787	0.012580	-0.021629	-0.042
person_income	0.173202	1.000000	0.134268	0.266820	0.000792	-0.144449	-0.254
person_emp_length	0.163106	0.134268	1.000000	0.113082	-0.056405	-0.082489	-0.05
loan_amnt	0.050787	0.266820	0.113082	1.000000	0.146813	0.105376	0.572
loan_int_rate	0.012580	0.000792	-0.056405	0.146813	1.000000	0.335133	0.120
loan_status	-0.021629	-0.144449	-0.082489	0.105376	0.335133	1.000000	0.379
loan_percent_income	-0.042411	-0.254471	-0.054111	0.572612	0.120314	0.379366	1.000
cb_person_cred_hist_length	0.859133	0.117987	0.144699	0.041967	0.016696	-0.015529	-0.031

```
1 dataplot = sns.heatmap(df.corr(numeric_only=True), cmap='YlGnBu', annot=True)
2 plt.title('Data correlation', fontsize=18)
3 plt.show()
```



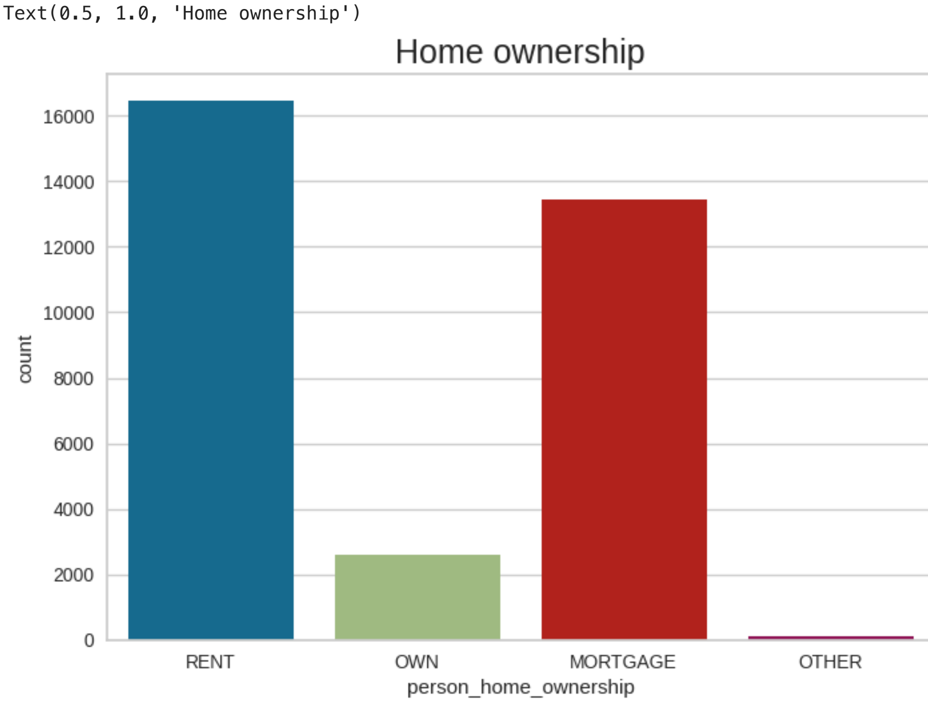

```
1 sns.countplot(x=df['loan_status'])
2 plt.title('Loan Status', fontsize=18)
```

```
Text(0.5, 1.0, 'Loan Status')
```



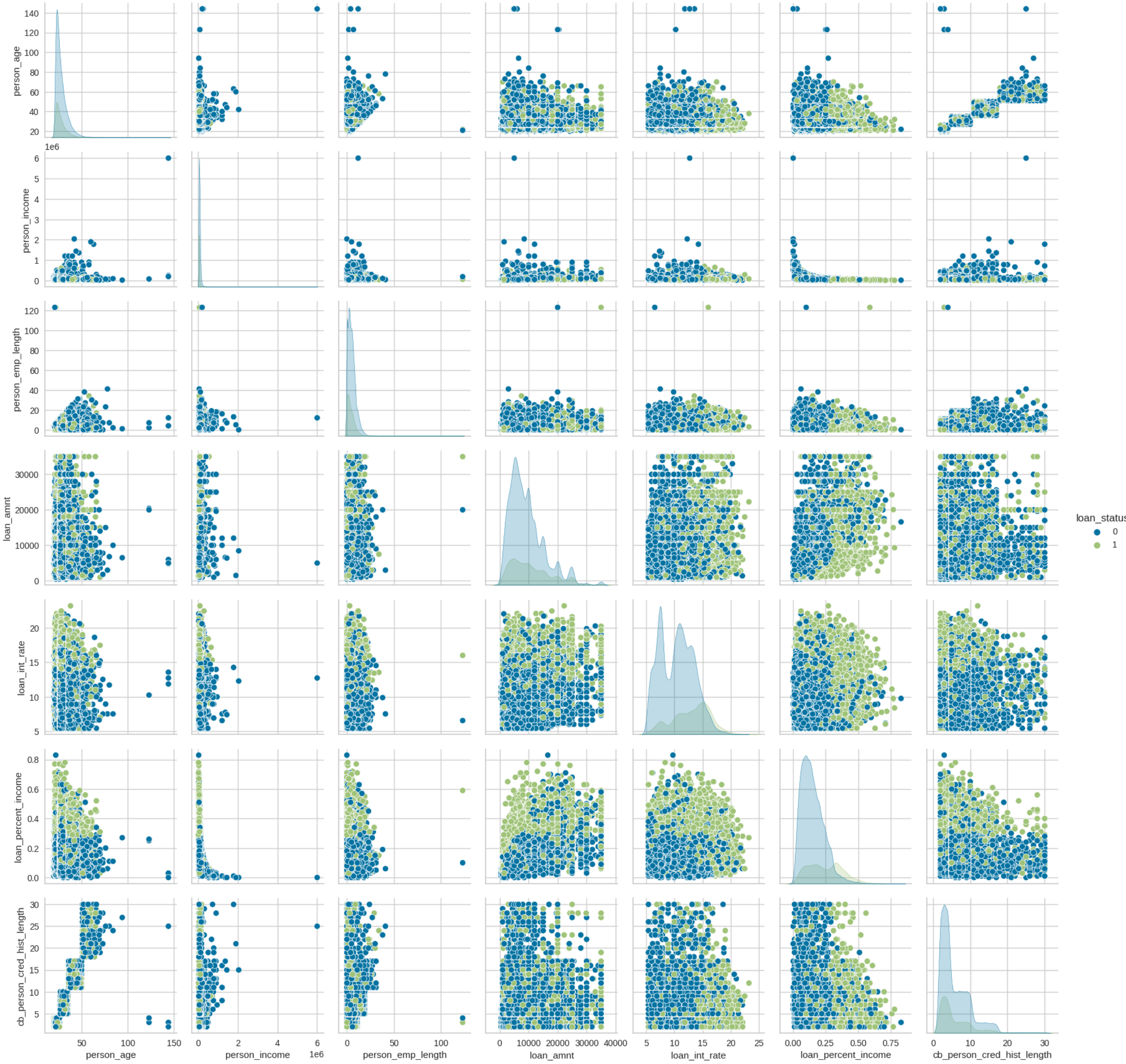

```
1 sns.countplot(x=df['person_home_ownership'])
2 plt.title('Home ownership', fontsize=18)
```

```
Text(0.5, 1.0, 'Home ownership')
```




```
1 sns.pairplot(df, hue='loan_status')
```

```
<seaborn.axisgrid.PairGrid at 0x7aae3f5dd060>
```

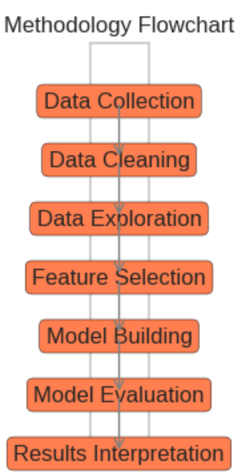



```
df.isnull().sum()
#Filling missing values with mean:
df.loc[df['loan_int_rate'].isnull(), 'loan_int_rate'] = df['loan_int_rate'].median()
df.loc[df['person_emp_length'].isnull(), 'person_emp_length'] = df['person_emp_length'].median()
df.isnull().sum()
```

```
person_age                    0
person_income                 0
person_home_ownership         0
person_emp_length             0
loan_intent                   0
loan_grade                    0
loan_amnt                     0
loan_int_rate                 0
loan_status                   0
loan_percent_income           0
cb_person_default_on_file     0
cb_person_cred_hist_length    0
dtype: int64
```

```
df['person_age'].max()

#Assuming individuals with age > 90 to be errors
df = df.loc[df['person_age'] < 90]
df['person_emp_length'].max()

#Employment cannot be greater than the individual's age (accounting for childhood)
df = df.loc[df['person_emp_length'] < df['person_age'] - 10]
```

```
import matplotlib.pyplot as plt
# Create the approach flowchart
fig, ax = plt.subplots(figsize=(6, 4))
steps = [
    "Data Collection",
    "Data Cleaning",
    "Data Exploration",
    "Feature Selection",
    "Model Building",
    "Model Evaluation",
    "Results Interpretation"
]
positions = list(range(len(steps)))
step_coordinates = {}
for i, step in enumerate(steps):
    x = 0.5
    y = len(steps) - 1 - i
    step_coordinates[step] = (x, y)
    ax.text(x, y, step, ha='center', va='center', fontsize=12, bbox=dict(boxstyle='round,pad=0.3', facecolor='coral'))
for i in range(len(steps) - 1):
    ax.annotate("", xy=step_coordinates[steps[i]], xytext=step_coordinates[steps[i+1]], arrowprops=dict(arrowstyle='<-', line
ax.set_xlim(0, 1)
ax.set_ylim(0, len(steps))
ax.set_xticks([])
ax.set_yticks([])
ax.set_aspect('equal')
plt.title("Methodology Flowchart")
plt.show()
```



Data Processing and Encoding

```
# Creating groups
df['income_group'] = pd.cut(df['person_income'], bins= [0, 25000, 50000, 75000, 100000, float('inf')], labels=['low', 'l-midc
df['income_group']
df['loan_amnt_group'] = pd.cut(df['loan_amnt'], bins= [0, 10000, 15000, float('inf')],labels=['small', 'medium', 'large'])
df['loan_amnt_group']
df['loan_to_income'] = df['loan_amnt'] / df['person_income']
df['loan_to_income']
```

```
1        0.104167
2        0.572917
3        0.534351
4        0.643382
5        0.252525
           ...
32576    0.109434
32577    0.146875
```

```
32578    0.460526
32579    0.100000
32580    0.154167
Name: loan_to_income, Length: 32573, dtype: float64
```

```
#Spliting dataset in to Training set and Test set
y_credit = df['loan_status']
X_credit = df.drop(['loan_status'], axis=1)
X_credit.columns
label_encode_cols = ['person_home_ownership', 'loan_intent', 'loan_grade', 'cb_person_default_on_file', 'income_group', 'loar
label_encoder = LabelEncoder()

for col in label_encode_cols:
  X_credit[col] = label_encoder.fit_transform(X_credit[col])

X_credit = pd.get_dummies(X_credit, columns=label_encode_cols)
X_credit.head(1)
scaler = StandardScaler()
X_credit = scaler.fit_transform(X_credit)
X_credit[0]
X_training, X_test, y_training, y_test = train_test_split(X_credit, y_credit, test_size= 0.2, random_state=0)
X_training.shape, y_training.shape
```

```
((26058, 35), (26058,))
```

Modelling

```
#Naive Bayes
from sklearn.naive_bayes import GaussianNB
naive_bayer = GaussianNB()
naive_bayer.fit(X_training, y_training)
predict_NB = naive_bayer.predict(X_test)
```

```
#Decision Tree
from sklearn.tree import DecisionTreeClassifier
decision_tree = DecisionTreeClassifier(criterion='entropy', random_state = 0)
decision_tree.fit(X_training, y_training)
predict_decision_tree = decision_tree.predict(X_test)
```

```
#Random Forest
from sklearn.ensemble import RandomForestClassifier
random_forest = RandomForestClassifier(n_estimators=200, criterion='entropy', random_state =0)
random_forest.fit(X_training, y_training)
predict_random_forest = random_forest.predict(X_test)
```

```
#Nearest Neighbors
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=20)
knn.fit(X_training, y_training)
predict_knn = knn.predict(X_test)
```

```
#Logistic Regression
from sklearn.linear_model import LogisticRegression
logistic = LogisticRegression(random_state=1)
logistic.fit(X_training, y_training)
predict_logistic = logistic.predict(X_test)
logistic.intercept_
```

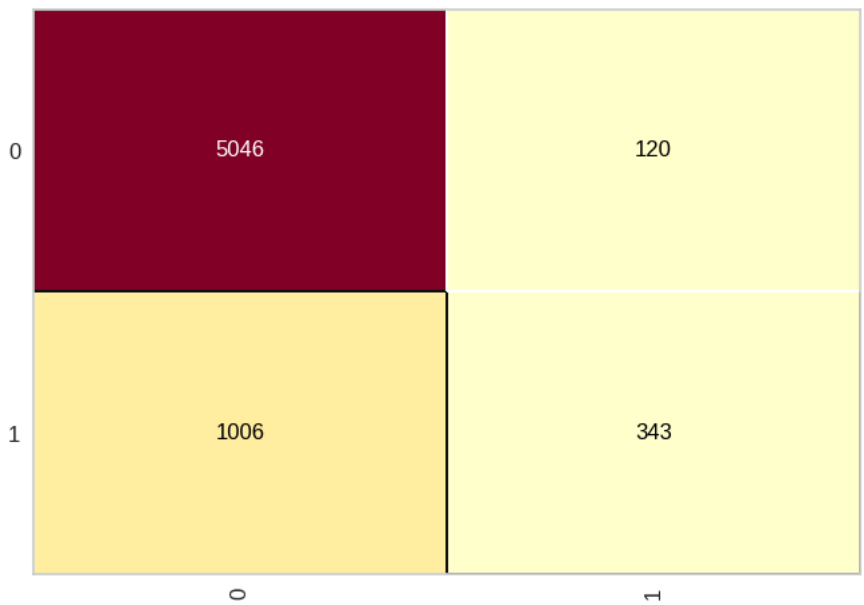
```
array([-1.91329537])
```

```
#SVM
from sklearn.svm import SVC
svm = SVC(kernel='rbf', random_state=1, C=2)
svm.fit(X_training, y_training)
predict_svm = svm.predict(X_test)
```

Valuation Metrics

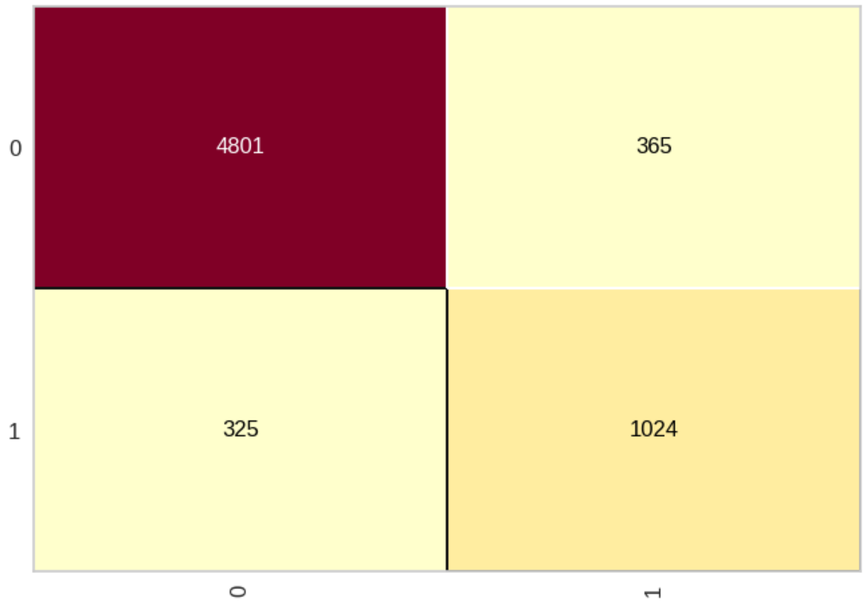
```
#Naive Bayes
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
cm = ConfusionMatrix(naive_bayer)
cm.fit(X_training, y_training)
cm.score(X_test, y_test)
# Calculate accuracy
accuracy = accuracy_score(y_test, predict_NB)

# Calculate precision
precision = precision_score(y_test, predict_NB)

# Calculate recall
recall = recall_score(y_test, predict_NB)

# Calculate F1-score
f1 = f1_score(y_test, predict_NB)

print("\033[1mNaive Bayes\033[0m")
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")
```

Naive Bayes
Accuracy: 0.83
Precision: 0.74
Recall: 0.25
F1-Score: 0.38



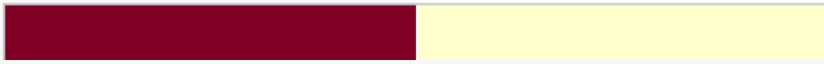

```
#Decision Tree
cm = ConfusionMatrix(decision_tree)
cm.fit(X_training, y_training)
cm.score(X_test, y_test)
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Calculate accuracy
accuracy = accuracy_score(y_test, predict_decision_tree)

# Calculate precision
precision = precision_score(y_test, predict_decision_tree)

# Calculate recall
recall = recall_score(y_test, predict_decision_tree)

# Calculate F1-score
f1 = f1_score(y_test, predict_decision_tree)

print("\033[1mDecision Tree\033[0m")
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")
```

```
Decision Tree
Accuracy: 0.89
Precision: 0.74
Recall: 0.76
F1-Score: 0.75
```



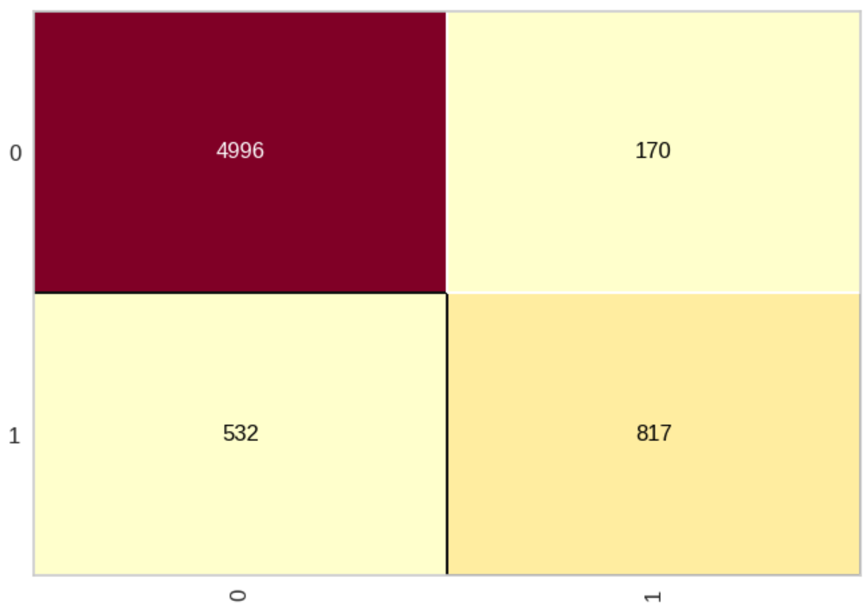

```
#Random Forest
cm = ConfusionMatrix(random_forest)
cm.fit(X_training, y_training)
cm.score(X_test, y_test)
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Calculate accuracy
accuracy = accuracy_score(y_test, predict_random_forest)

# Calculate precision
precision = precision_score(y_test, predict_random_forest)

# Calculate recall
recall = recall_score(y_test, predict_random_forest)

# Calculate F1-score
f1 = f1_score(y_test, predict_random_forest)

print("\033[1mRandom Forest\033[0m")
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")
```

```
Random Forest
Accuracy: 0.94
Precision: 0.97
Recall: 0.72
F1-Score: 0.82
```



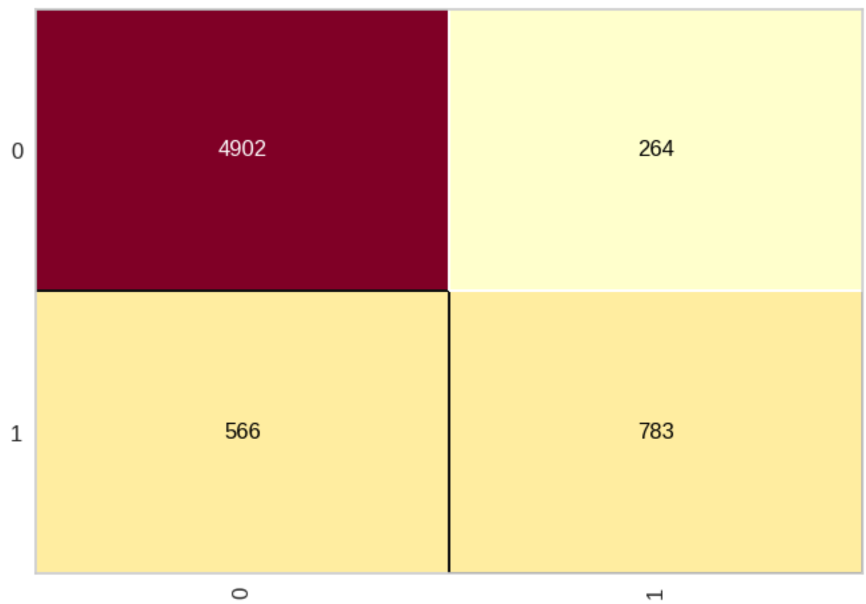
```
#Nearest Neighbors
cm = ConfusionMatrix(knn)
cm.fit(X_training, y_training)
cm.score(X_test, y_test)
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Calculate accuracy
accuracy = accuracy_score(y_test, predict_knn)

# Calculate precision
precision = precision_score(y_test, predict_knn)

# Calculate recall
recall = recall_score(y_test, predict_knn)

# Calculate F1-score
f1 = f1_score(y_test, predict_knn)

print("\033[1mNearest Neighbors\033[0m")
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")
```

```
Nearest Neighbors
Accuracy: 0.89
Precision: 0.83
Recall: 0.61
F1-Score: 0.70
```



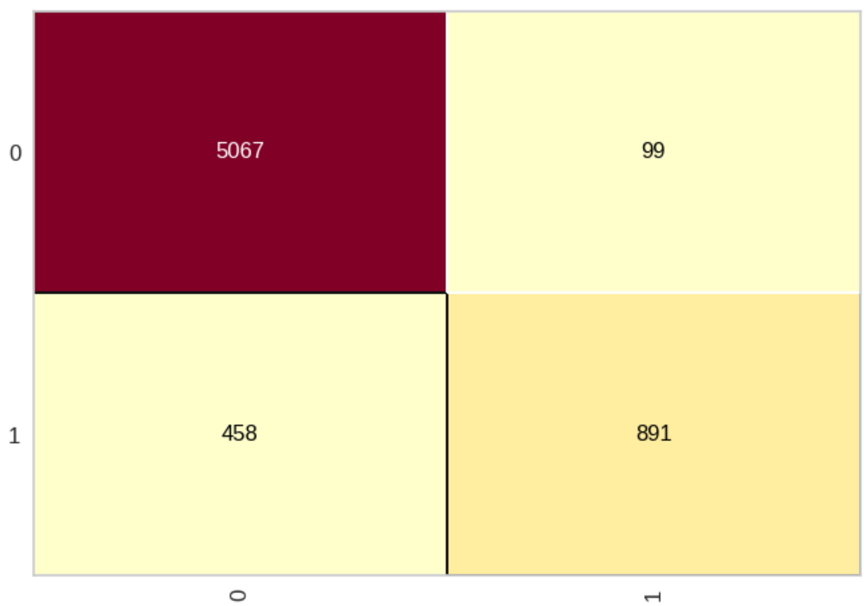
```
#Logistic Regression
cm = ConfusionMatrix(logistic)
cm.fit(X_training, y_training)
cm.score(X_test, y_test)
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Calculate accuracy
accuracy = accuracy_score(y_test, predict_logistic)

# Calculate precision
precision = precision_score(y_test, predict_logistic)

# Calculate recall
recall = recall_score(y_test, predict_logistic)

# Calculate F1-score
f1 = f1_score(y_test, predict_logistic)

print("\033[1mLogistic Regression\033[0m")
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")
```

```
Logistic Regression
Accuracy: 0.87
Precision: 0.75
Recall: 0.58
F1-Score: 0.65
```



```
#SVM
cm = ConfusionMatrix(svm)
cm.fit(X_training, y_training)
cm.score(X_test, y_test)

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Calculate accuracy
accuracy_svm = accuracy_score(y_test, predict_svm)

# Calculate precision
precision_svm = precision_score(y_test, predict_svm)

# Calculate recall
recall_svm = recall_score(y_test, predict_svm)

# Calculate F1-score
f1_svm = f1_score(y_test, predict_svm)

print("\033[1mSVM\033[0m")
print(f"Accuracy: {accuracy_svm:.2f}")
print(f"Precision: {precision_svm:.2f}")
print(f"Recall: {recall_svm:.2f}")
print(f"F1-Score: {f1_svm:.2f}")
```

```
SVM
Accuracy: 0.91
Precision: 0.90
Recall: 0.66
F1-Score: 0.76
```



Evaluation of Models

```
from datetime import datetime
from sklearn.model_selection import cross_val_score, KFold
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Define a function to evaluate and print model performance
def evaluate_model(model, X_train, X_test, y_train, y_test):
    # Time to Train
    start_time = datetime.now()
    model.fit(X_train, y_train)
    end_time = datetime.now()
    time_to_train = end_time - start_time
    print(f"Time to Train: {time_to_train}")

    # Time to Test
    start_time = datetime.now()
    y_pred = model.predict(X_test)
    end_time = datetime.now()
    time_to_test = end_time - start_time
    print(f"Time to Test: {time_to_test}")
```

```

    # Stability using cross-validation
    num_folds = 5
    kf = KFold(n_splits=num_folds, shuffle=True, random_state=0)
    cross_val_results = cross_val_score(model, X_train, y_train, cv=kf, scoring='accuracy')
    print(f"Cross-validation results: {cross_val_results}")
    print(f"Mean accuracy: {cross_val_results.mean()}")
    print(f"Standard deviation of accuracy: {cross_val_results.std()}")

    # Performance Metrics on Test Set
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    print(f"Accuracy: {accuracy:.2f}")
    print(f"Precision: {precision:.2f}")
    print(f"Recall: {recall:.2f}")
    print(f"F1-Score: {f1:.2f}")

# Apply the function for each model
models = [naive_bayer, decision_tree, random_forest, knn, logistic, svm]
for model in models:
    print(f"\nEvaluating {model.__class__.__name__}:")
    evaluate_model(model, X_training, X_test, y_training, y_test)
```

```
Precision: 0.74
Recall: 0.25
F1-Score: 0.38

Evaluating DecisionTreeClassifier:
Time to Train: 0:00:00.274620
Time to Test: 0:00:00.002521
Cross-validation results: [0.8862241  0.89850345 0.88718342 0.88140472 0.89195932]
Mean accuracy: 0.8890550024573143
Standard deviation of accuracy: 0.005792269139852778
Accuracy: 0.89
Precision: 0.74
Recall: 0.76
F1-Score: 0.75

Evaluating RandomForestClassifier:
Time to Train: 0:00:07.496751
Time to Test: 0:00:00.334772
Cross-validation results: [0.93016117 0.93937068 0.92689946 0.93033967 0.9337939 ]
Mean accuracy: 0.932112975194306
Standard deviation of accuracy: 0.004234022129317153
Accuracy: 0.94
Precision: 0.97
Recall: 0.72
F1-Score: 0.82

Evaluating KNeighborsClassifier:
Time to Train: 0:00:00.004485
Time to Test: 0:00:01.256882
Cross-validation results: [0.88660783 0.89140445 0.8823868  0.87622337 0.88677797]
Mean accuracy: 0.8846800844721148
```