Project - Loan Default

Table of Contents

Description

- General overview of project: background and what we want to solve
- Dataset used (source): https://machinehack.com/hackathons/deloitte_presents_machine_learning_challeng

Dataset Loading and Analysis

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from imblearn.over_sampling import SMOTE
import imblearn
from sklearn.datasets import make_classification, load_iris
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as L
from sklearn.dummy import DummyClassifier
from sklearn.metrics import roc_curve, auc, confusion_matrix, precision_r
from jupyterthemes import jtplot
from sklearn.preprocessing import StandardScaler, MinMaxScaler, FunctionT
```

```
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.calibration import CalibratedClassifierCV, calibration_curve
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

random_state=809
%matplotlib inline
plt.style.use('seaborn-v0_8-whitegrid') # If error reported by this line,
#jtplot.style(theme='onedork')
```

Data Loading

Since the initial dataset has already been seperated into train.csv and test.csv, we just load the data and don't need to seperate the in-sample data and out-of-sample data from train.csv. Here, train.csv data is regarded as in-sample data, while test.csv data is regarded as out-of-sample data

In [2]:
```
df= pd.read_csv("./train.csv")
```

In [3]:
```
df.head(10)
```

Out[3]:

	ID	Loan Amount	Funded Amount	Funded Amount Investor	Term	Batch Enrolled	Interest Rate	Gra
0	65087372	10000	32236	12329.362860	59	BAT2522922	11.135007	
1	1450153	3609	11940	12191.996920	59	BAT1586599	12.237563	
2	1969101	28276	9311	21603.224550	59	BAT2136391	12.545884	
3	6651430	11170	6954	17877.155850	59	BAT2428731	16.731201	
4	14354669	16890	13226	13539.926670	59	BAT5341619	15.008300	
5	50509046	34631	30203	8635.931613	36	BAT4694572	17.246986	
6	32737431	30844	19773	15777.511830	59	BAT4808022	10.731432	
7	63151650	20744	10609	7645.014802	58	BAT2558388	13.993688	
8	4279662	9299	11238	13429.456610	59	BAT5341619	11.178457	
9	4431034	19232	8962	7004.097481	58	BAT2078974	5.520413	

10 rows × 35 columns

Data Cleaning

Clear NaN or NA Values (if any)

In [4]:
```
# Find the count and percentage of missing values
df_na = pd.DataFrame({'Percent': 100*df.isnull().sum()/len(df), 'Count':

# Print columns with null count > 0
df_na[df_na['Count'] > 0]
```

Out[4]:

Percent	Count

In [5]:
```
hasNA = False
for i in df.isna().sum():
    if i != 0:
        hasNA = True
print(hasNA)
```

```
False
```

There's no NA/NaN value in this dataset.

In [6]:
```
df.dtypes
```

Out[6]:
```
ID                                int64
Loan Amount                       int64
Funded Amount                     int64
Funded Amount Investor          float64
Term                              int64
Batch Enrolled                   object
Interest Rate                   float64
Grade                            object
Sub Grade                        object
Home Ownership                   object
Salary                          float64
Verification Status              object
Payment Plan                     object
Loan Title                       object
Debit to Income                 float64
Delinquency - two years           int64
Inquires - six months             int64
Open Account                      int64
Public Record                     int64
Revolving Balance                 int64
Revolving Utilities             float64
Total Accounts                    int64
Initial List Status              object
Total Received Interest         float64
Total Received Late Fee         float64
Recoveries                      float64
Collection Recovery Fee         float64
Collection 12 months Medical      int64
Application Type                 object
Last week Pay                     int64
Accounts Delinquent               int64
Total Collection Amount           int64
Total Current Balance             int64
Total Revolving Credit Limit      int64
Loan Status                       int64
dtype: object
```

First, drop some obvious irrelavant columns. "ID" and "Batch Enrolled" are just identification number of loan, not useful for prediction.

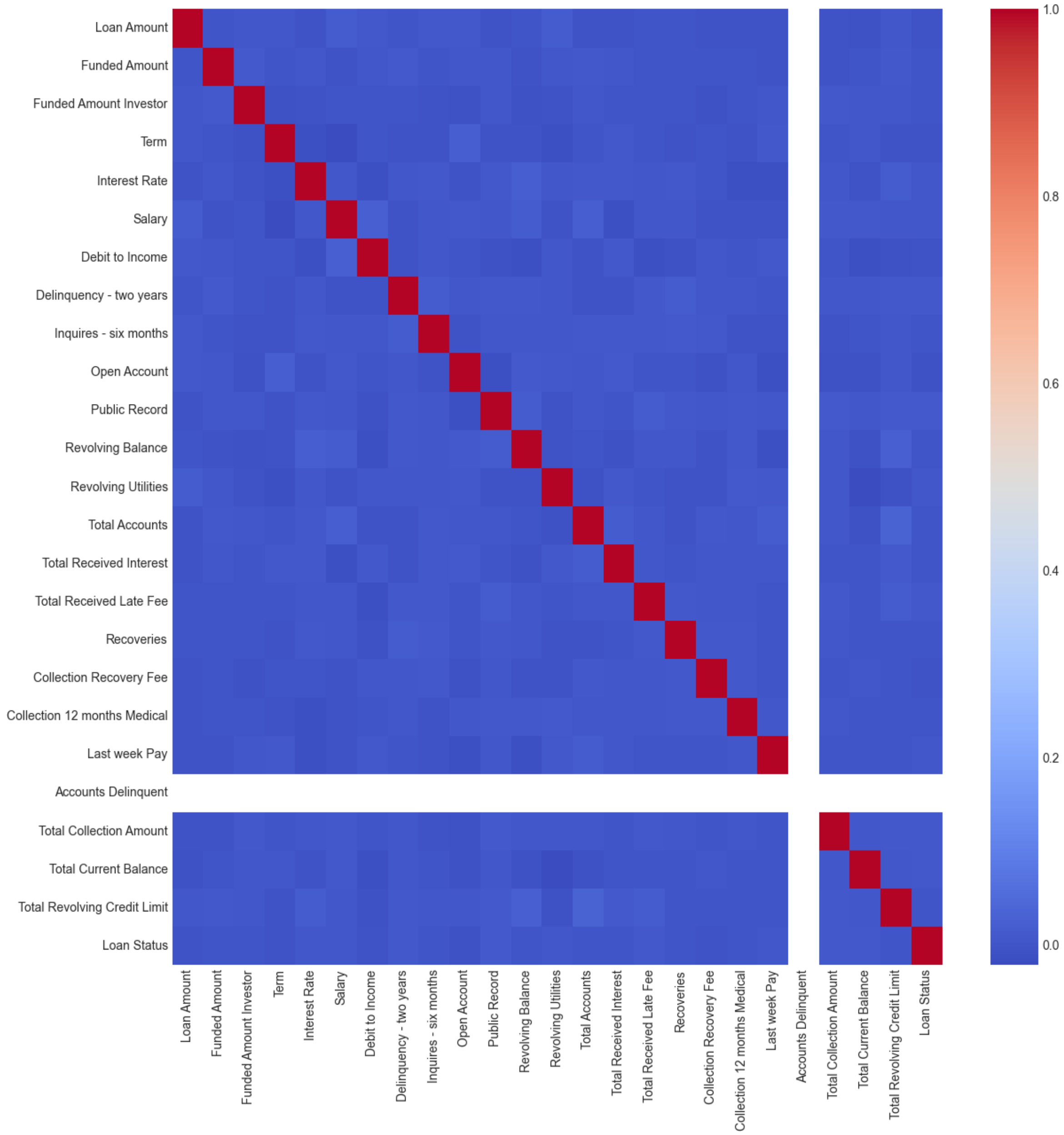
In [7]:
```
df.drop(['ID','Batch Enrolled'],axis=1,inplace=True)
```

Data Distribution

Correlation Analysis

In [8]:
```
corr = df.corr(numeric_only=True )
# Plot the heatmap
plt.figure(figsize=(14,14))
sns.heatmap(corr,
        xticklabels=corr.columns,
        yticklabels=corr.columns,
        cmap='coolwarm')
```

Out[8]: <AxesSubplot: >

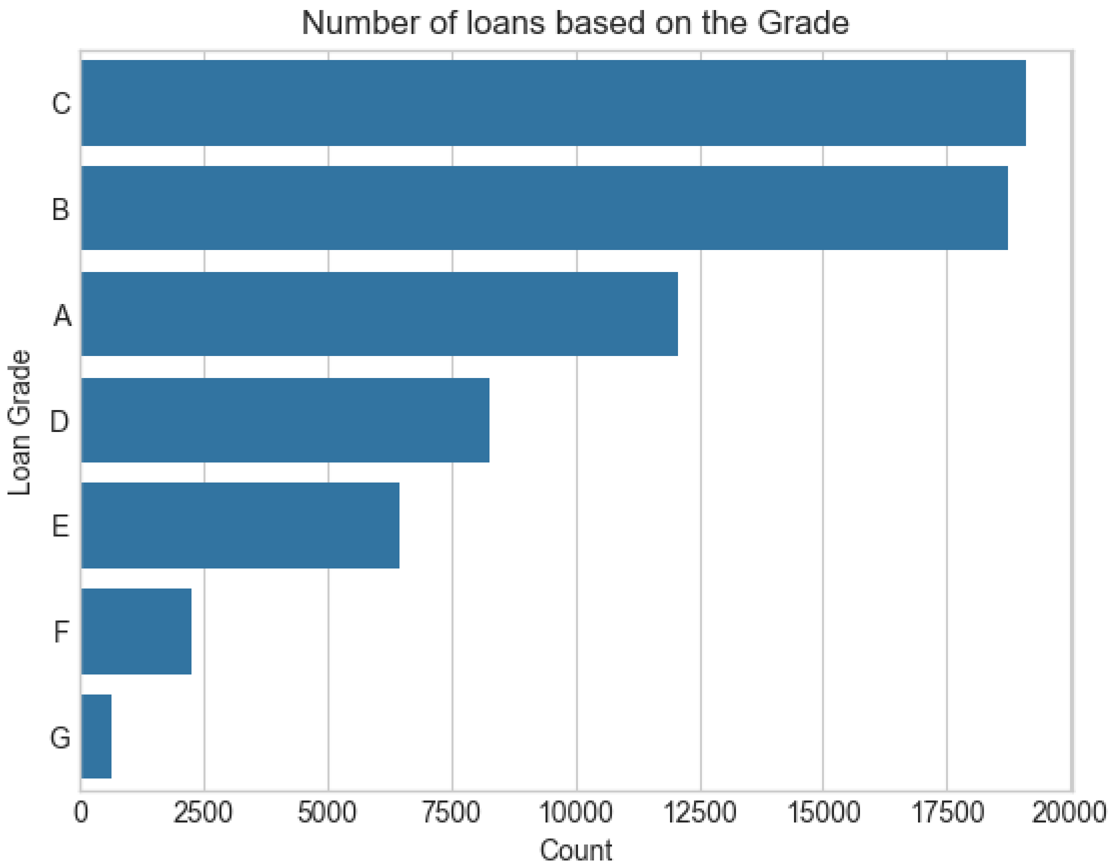


Based on the heatmap, the numeracal values in the dataset does not reveal strong correlation.

Loan Grade

In [9]:
```
grade_vis = df['Grade'].value_counts()

sns.barplot(y=grade_vis.index, x=grade_vis)
plt.title('Number of loans based on the Grade')
plt.ylabel('Loan Grade')
plt.xlabel('Count')
```

Out[9]:
```
Text(0.5, 0, 'Count')
```



Drop columns

In [10]:
```
df.drop(['Term','Verification Status','Payment Plan','Loan Title','Delinq
```

In [11]:
```
df.drop(['Open Account','Total Accounts','Total Received Late Fee','Recov
```

In [12]:
```
df.columns
```

Out[12]:
```
Index(['Loan Amount', 'Funded Amount', 'Funded Amount Investor',
       'Interest Rate', 'Grade', 'Sub Grade', 'Home Ownership', 'Salar
y',
       'Debit to Income', 'Revolving Balance', 'Total Received Interes
t',
       'Total Collection Amount', 'Total Current Balance',
       'Total Revolving Credit Limit', 'Loan Status'],
      dtype='object')
```

Dummy Variables

In [13]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 67463 entries, 0 to 67462
Data columns (total 15 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Loan Amount                   67463 non-null  int64  
 1   Funded Amount                 67463 non-null  int64  
 2   Funded Amount Investor        67463 non-null  float64
 3   Interest Rate                 67463 non-null  float64
 4   Grade                         67463 non-null  object 
 5   Sub Grade                     67463 non-null  object 
 6   Home Ownership                67463 non-null  object 
 7   Salary                        67463 non-null  float64
 8   Debit to Income               67463 non-null  float64
 9   Revolving Balance             67463 non-null  int64  
 10  Total Received Interest       67463 non-null  float64
 11  Total Collection Amount       67463 non-null  int64  
 12  Total Current Balance         67463 non-null  int64  
 13  Total Revolving Credit Limit  67463 non-null  int64  
 14  Loan Status                   67463 non-null  int64  
dtypes: float64(5), int64(7), object(3)
memory usage: 7.7+ MB
```

In [14]:
```
categorical_variables=['Grade','Sub Grade',"Home Ownership"]
df_categorical=pd.get_dummies(data=df, columns=categorical_variables,pref
```

In [15]:
```
df_categorical.info()
```

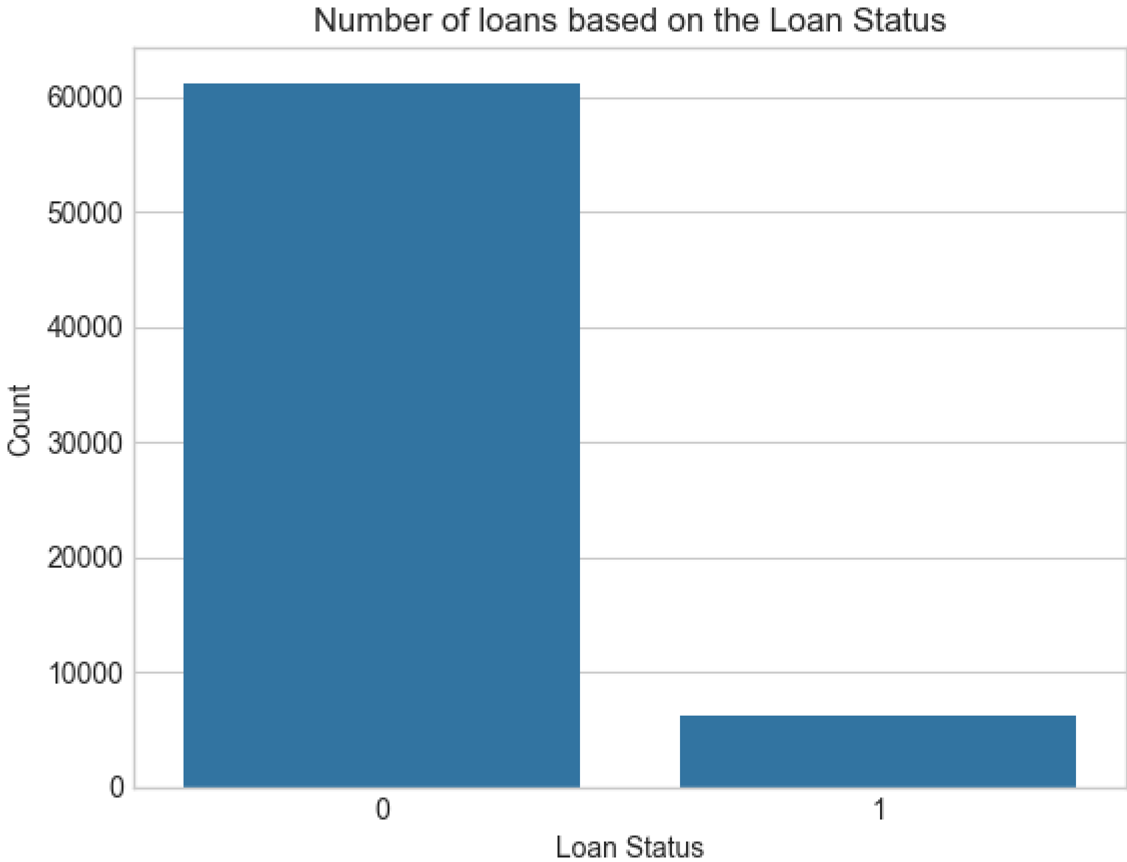
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 67463 entries, 0 to 67462
Data columns (total 54 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Loan Amount                   67463 non-null  int64  
 1   Funded Amount                 67463 non-null  int64  
 2   Funded Amount Investor        67463 non-null  float64
 3   Interest Rate                 67463 non-null  float64
 4   Salary                        67463 non-null  float64
 5   Debit to Income               67463 non-null  float64
 6   Revolving Balance             67463 non-null  int64  
 7   Total Received Interest       67463 non-null  float64
 8   Total Collection Amount       67463 non-null  int64  
 9   Total Current Balance         67463 non-null  int64  
 10  Total Revolving Credit Limit  67463 non-null  int64  
 11  Loan Status                   67463 non-null  int64  
 12  Col_B                         67463 non-null  int8   
 13  Col_C                         67463 non-null  int8   
 14  Col_D                         67463 non-null  int8   
 15  Col_E                         67463 non-null  int8   
 16  Col_F                         67463 non-null  int8   
 17  Col_G                         67463 non-null  int8   
 18  Col_A2                        67463 non-null  int8   
 19  Col_A3                        67463 non-null  int8   
 20  Col_A4                        67463 non-null  int8   
 21  Col_A5                        67463 non-null  int8   
 22  Col_B1                        67463 non-null  int8   
 23  Col_B2                        67463 non-null  int8   
 24  Col_B3                        67463 non-null  int8   
 25  Col_B4                        67463 non-null  int8   
 26  Col_B5                        67463 non-null  int8   
 27  Col_C1                        67463 non-null  int8   
 28  Col_C2                        67463 non-null  int8   
 29  Col_C3                        67463 non-null  int8   
 30  Col_C4                        67463 non-null  int8   
 31  Col_C5                        67463 non-null  int8   
 32  Col_D1                        67463 non-null  int8   
 33  Col_D2                        67463 non-null  int8   
 34  Col_D3                        67463 non-null  int8   
 35  Col_D4                        67463 non-null  int8   
 36  Col_D5                        67463 non-null  int8   
 37  Col_E1                        67463 non-null  int8   
 38  Col_E2                        67463 non-null  int8   
 39  Col_E3                        67463 non-null  int8   
 40  Col_E4                        67463 non-null  int8   
 41  Col_E5                        67463 non-null  int8   
 42  Col_F1                        67463 non-null  int8   
 43  Col_F2                        67463 non-null  int8   
 44  Col_F3                        67463 non-null  int8   
 45  Col_F4                        67463 non-null  int8   
 46  Col_F5                        67463 non-null  int8   
 47  Col_G1                        67463 non-null  int8   
 48  Col_G2                        67463 non-null  int8   
 49  Col_G3                        67463 non-null  int8   
 50  Col_G4                        67463 non-null  int8   
 51  Col_G5                        67463 non-null  int8   
 52  Col_OWN                       67463 non-null  int8   
 53  Col_RENT                      67463 non-null  int8   
```

```
dtypes: float64(5), int64(7), int8(42)
memory usage: 8.9 MB
```

Data Imbalance - SMOTE

In [16]:
```
loan_status_val = df['Loan Status'].value_counts()
sns.barplot(y=loan_status_val, x=loan_status_val.index)
plt.title('Number of loans based on the Loan Status')
plt.ylabel('Count')
plt.xlabel('Loan Status')

print("Number of 0 observations (Loan Non-Defaulted):", loan_status_val[0
print("Number of 1 observations (Loan Defaulted):", loan_status_val[1])
```

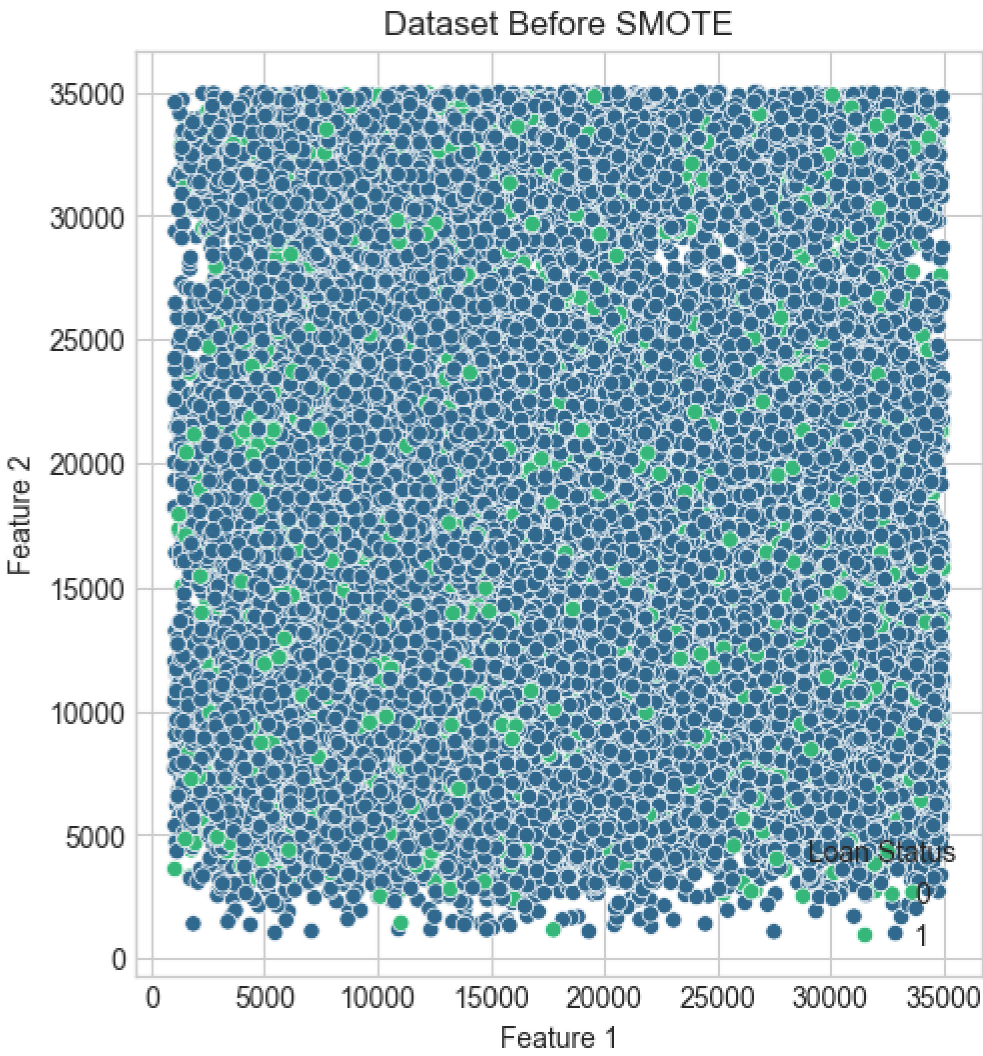
```
Number of 0 observations (Loan Non-Defaulted): 61222
Number of 1 observations (Loan Defaulted): 6241
```



In [17]:
```
X= df_categorical.drop('Loan Status', axis=1)
y= df_categorical['Loan Status']
```

In [18]:
```
# Visualize the dataset before SMOTE
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
sns.scatterplot(x=X.iloc[:, 0], y=X.iloc[:, 1], hue=y, palette='viridis',
plt.title('Dataset Before SMOTE')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
```

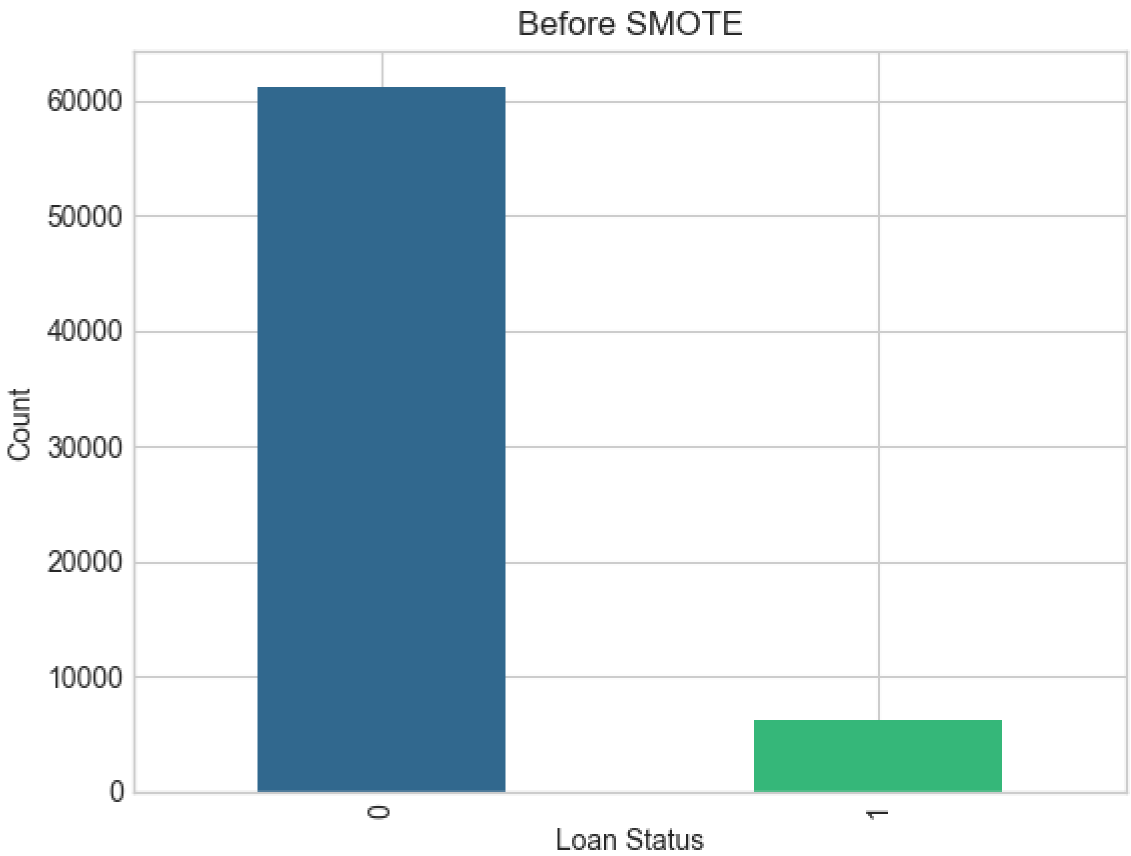
Out[18]:
```
Text(0, 0.5, 'Feature 2')
```



In [23]:

```
# Use matplotlib directly to control colors, if needed
colors = sns.color_palette('viridis', len(y.unique()))  # Assuming y has
y.value_counts().plot(kind='bar', color=colors)
plt.title('Before SMOTE')
plt.xlabel('Loan Status')
plt.ylabel('Count')
```

Out[23]: Text(0, 0.5, 'Count')

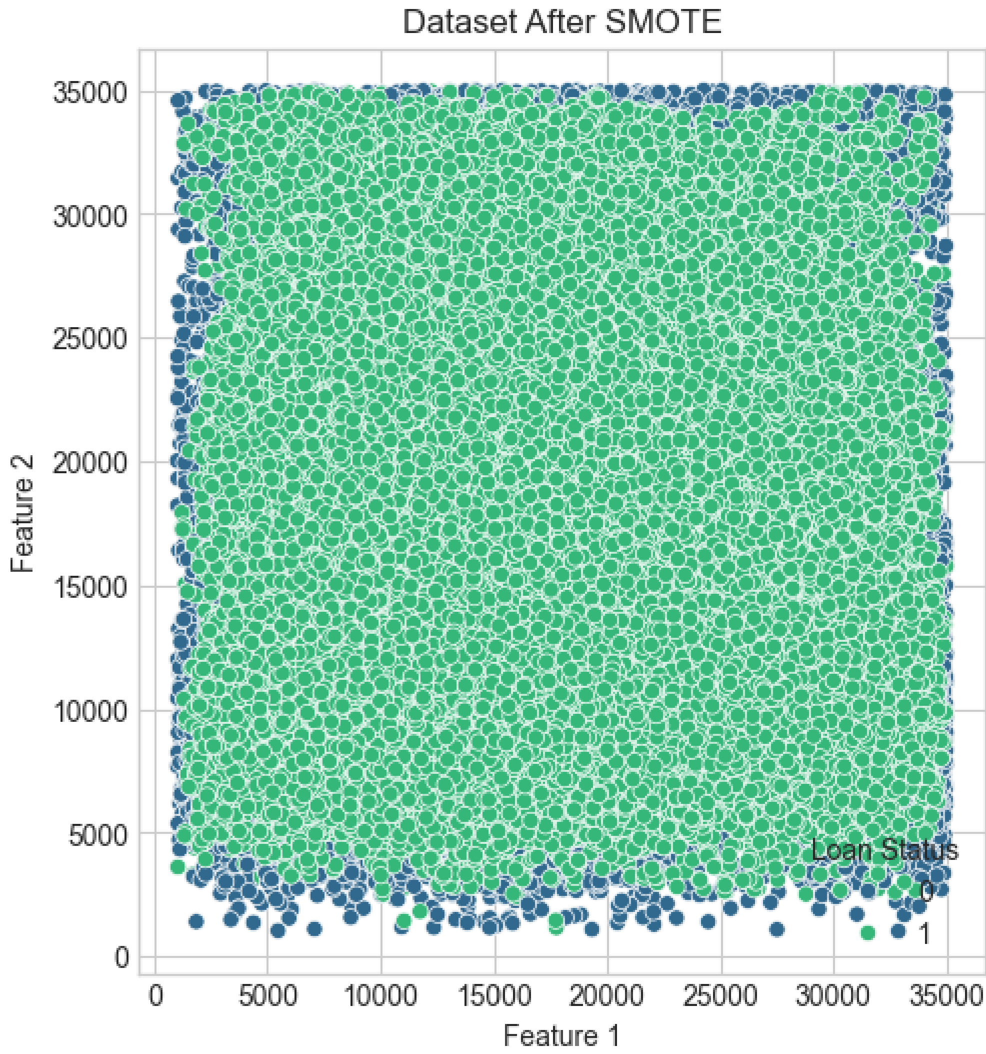


In [24]:
```
# Apply SMOTE
smote = SMOTE(random_state=random_state)
smote.fit(X,y)
X,y=smote.fit_resample(X,y)
```

In [25]:
```
# Feature Scaling
scaler = MinMaxScaler()
x_scaled = scaler.fit_transform(X)
```

In [26]:
```
# Visualize the dataset after SMOTE
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 2)
sns.scatterplot(x=X.iloc[:, 0], y=X.iloc[:, 1], hue=y, palette='viridis',
plt.title('Dataset After SMOTE')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
```

Out[26]:
```
Text(0, 0.5, 'Feature 2')
```



In [27]:

```
# Bar chart for class distribution after SMOTE
plt.figure(figsize=(8, 5))
plt.bar(['Class 0', 'Class 1'], [sum(y == 0), sum(y == 1)], color=['skybl
plt.title('Class Distribution After SMOTE')
plt.xlabel('Class')
plt.ylabel('Count')

# Show the plots
plt.show()
```

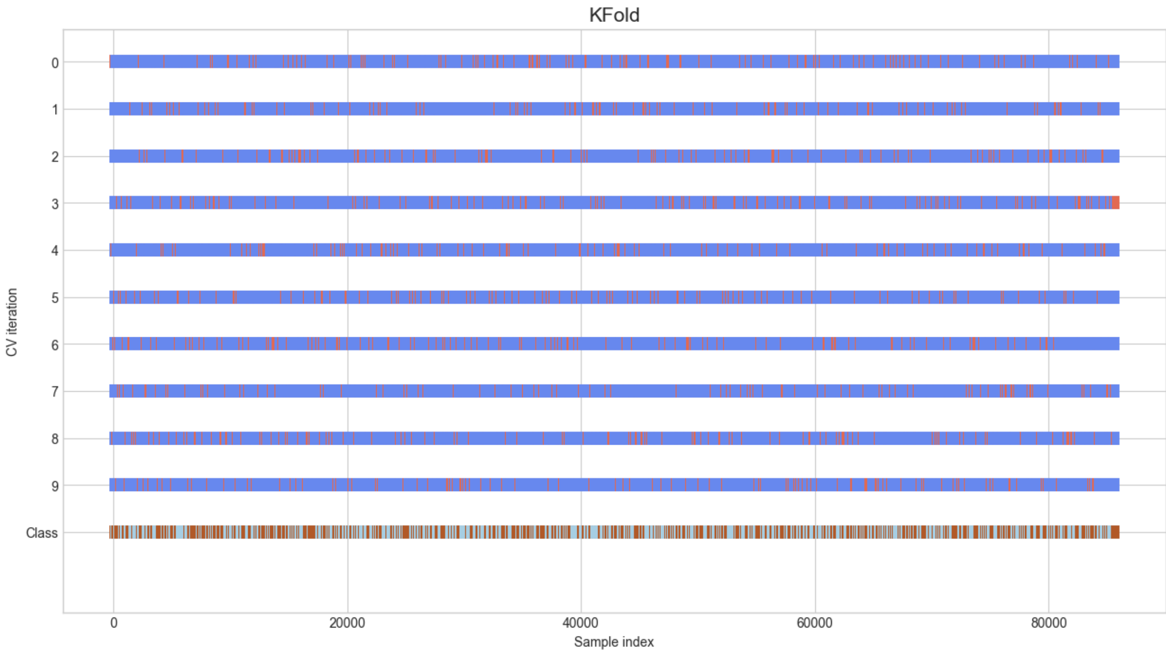
In [28]:
```
# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(x_scaled, y, test_siz
```

Cross validation

In [29]:
```
# visualize the training and testing splits generated by a cross-validati
def plot_cv_indices(cv, X, y, n_splits, lw=10):
    fig, ax = plt.subplots(figsize = (15,8))
    # Generate the training/testing visualizations for each CV split
    for ii, (tr, tt) in enumerate(cv.split(X=X, y=y)):
        # Fill in indices with the training/test groups
        indices = np.array([np.nan] * len(X))
        indices[tt] = 1
        indices[tr] = 0

        # Visualize the results
        ax.scatter(range(len(indices)), [ii + .5] * len(indices),
                   c=indices, marker='_', lw=lw, cmap=plt.cm.coolwarm,
                   vmin=-.2, vmax=1.2)
    # Plot the data classes
    ax.scatter(range(len(X)), [ii + 1.5] * len(X), c=y, marker='_', lw=lw

    # Formatting
    yticklabels = list(range(n_splits)) + ['Class']
    ax.set(yticks=np.arange(n_splits+1) + .5, yticklabels=yticklabels,
        xlabel='Sample index', ylabel="CV iteration",
        ylim=[n_splits+2.2, -.2])
    ax.set_title('{}'.format(type(cv).__name__), fontsize=15)
    return ax
```

In [30]:
```
n_splits = 10
shuffle = True
cv = KFold(n_splits=n_splits, shuffle=shuffle, random_state=random_state)
```

```
plot = plot_cv_indices(cv, X_train, y_train, n_splits)
plt.show()
```

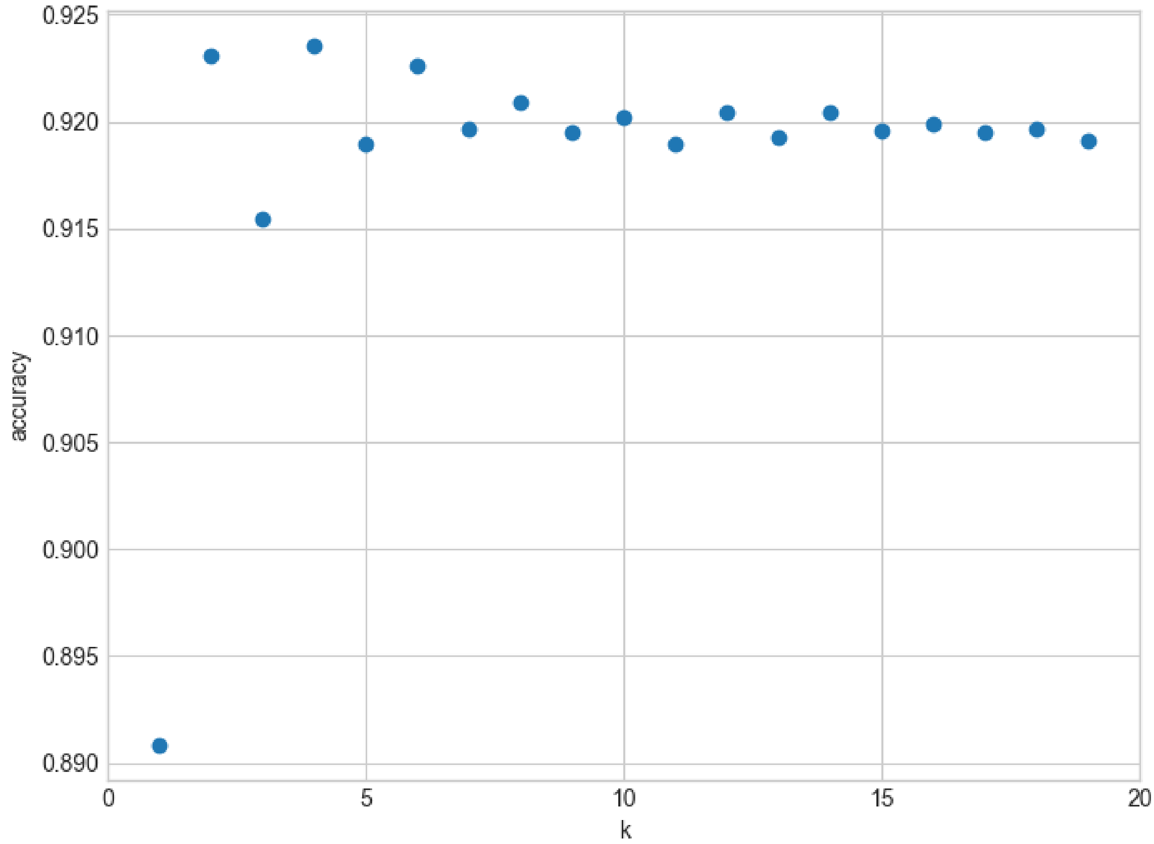


Model Fitting and Evaluation

In [31]:
```
# Compare model evaluation at different values of K by adjusting the valu
k_range = range(1,20)
scores =[]

for k in k_range:
    knn = KNeighborsClassifier(n_neighbors =k)
    knn.fit(X_train,y_train)
    scores.append(knn.score(X_test,y_test))

max_score = max(scores)
max_k = k_range[scores.index(max_score)]
```

In [32]:
```
plt.figure(figsize = (8,6))
plt.xlabel('k')
plt.ylabel('accuracy')
plt.scatter(k_range, scores)
plt.xticks([0,5,10,15,20])
```

Out[32]:
```
([<matplotlib.axis.XTick at 0x2c825dd3f50>,
  <matplotlib.axis.XTick at 0x2c825db2850>,
  <matplotlib.axis.XTick at 0x2c825dd2c90>,
  <matplotlib.axis.XTick at 0x2c825e0b090>,
  <matplotlib.axis.XTick at 0x2c825e19590>],
 [Text(0, 0, '0'),
  Text(5, 0, '5'),
  Text(10, 0, '10'),
  Text(15, 0, '15'),
  Text(20, 0, '20')])
```



In [33]:
```
# Set the model list
models = []
models.append(('LR', LogisticRegression(random_state=random_state)))
models.append(('KNN', KNeighborsClassifier(n_neighbors=max_k)))
# models.append(('DTC', DecisionTreeClassifier(random_state=random_state)
models.append(('RF', RandomForestClassifier(n_estimators=100, random_stat
models.append(('XGB', XGBClassifier(use_label_encoder=False, eval_metric=
```

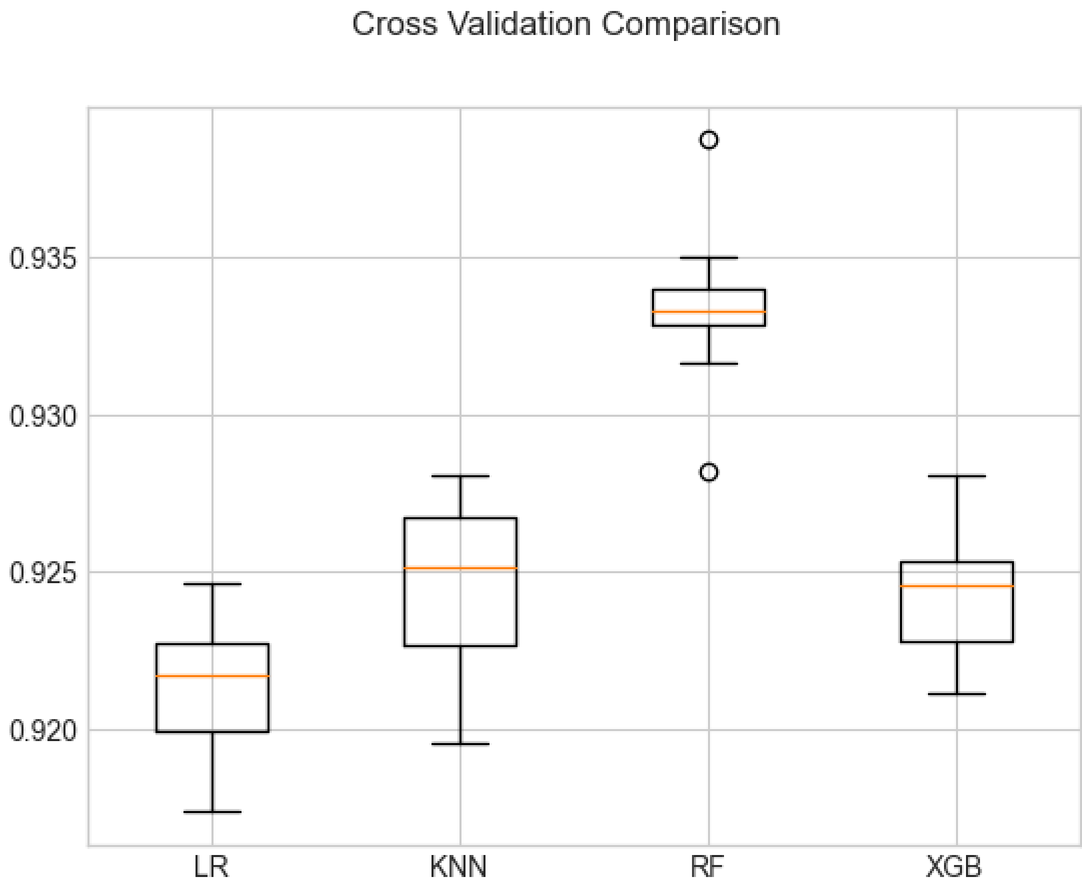
Do Cross Validation using different models to do the comparison

In [34]:
```
# Do cross validation using different models to do the comparison
n_splits = 10
scoring = 'accuracy'
results = []
tags = []
for tag, model in models:
    cv = KFold(n_splits=n_splits, random_state=random_state, shuffle=True
    cv_results = cross_val_score(model, x_scaled, y, cv=cv, scoring=scori
    results += [cv_results]
    tags += [tag]
    print( tag + ": " + f'{cv_results.mean():.4f}' + ' (' + f'{cv_results
```

```
LR: 0.9213 (0.0020)
KNN: 0.9246 (0.0027)
RF: 0.9334 (0.0025)
XGB: 0.9243 (0.0019)
```

In [35]:
```
# Plot the comparison result
fig_cv_compare = plt.figure()
fig_cv_compare.suptitle('Cross Validation Comparison')
ax = fig_cv_compare.add_subplot(111)
```

```
plt.boxplot(results)
ax.set_xticklabels(tags)
plt.show()
```

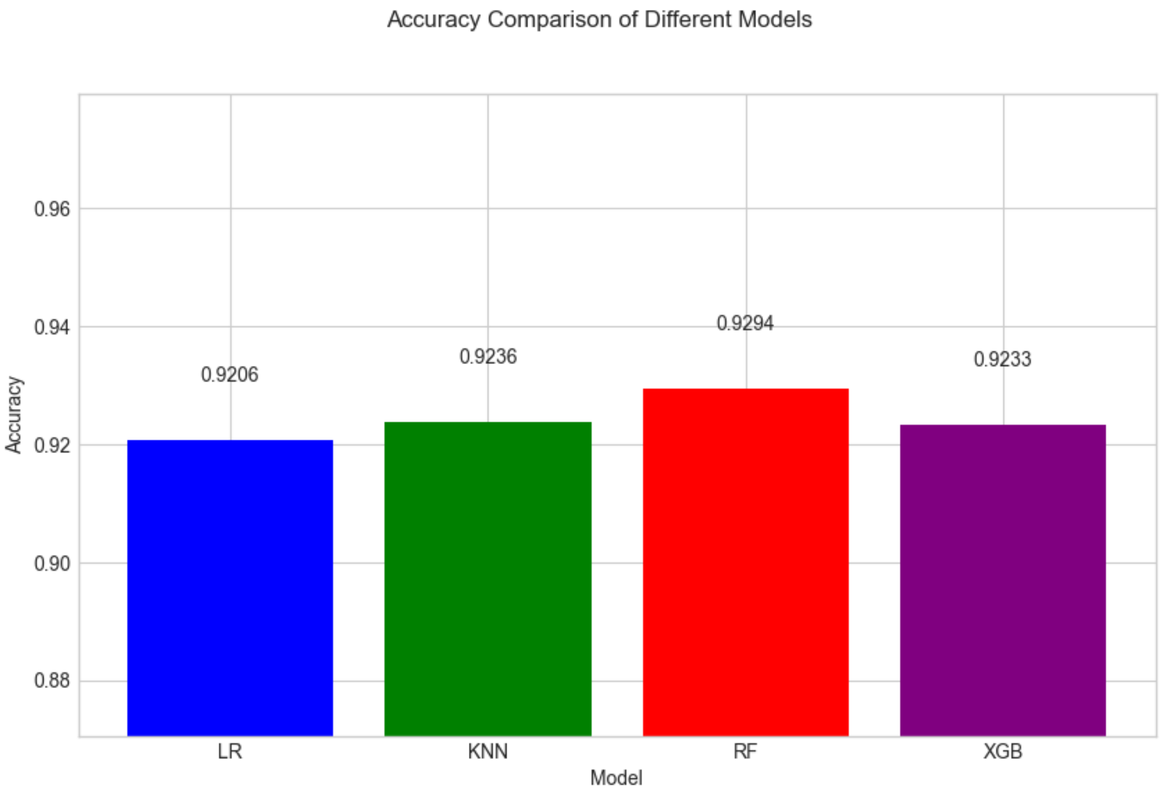


Do Model Accuracy using different models to do the comparison

In [36]:
```
# Dictionary to store accuracy of each model
model_accuracies = {}
acc = []
tags = []
# Iterate over each model, train it, predict on the testing set, and calc
for tag, model in models:
    # Fit the model on the training set
    model.fit(X_train, y_train)

    # Predict on the testing set
    y_pred = model.predict(X_test)

    # Calculate the accuracy and store it
    accuracy = accuracy_score(y_test, y_pred)
    model_accuracies[tag] = accuracy
    acc += [accuracy]
    tags += [tag]
    print(f'{tag} Accuracy: {accuracy:.4f}')
```

```
LR Accuracy: 0.9206
KNN Accuracy: 0.9236
RF Accuracy: 0.9294
XGB Accuracy: 0.9233
```

In [37]:
```
# Plot the comparison result as a bar chart
fig_acc_compare = plt.figure(figsize=(10, 6))
fig_acc_compare.suptitle('Accuracy Comparison of Different Models')
ax = fig_acc_compare.add_subplot(111)

# Create a bar chart
ax.bar(tags, acc, color=['blue', 'green', 'red', 'purple', 'orange'])

ax.set_ylabel('Accuracy')
ax.set_xlabel('Model')
ax.set_ylim(min(acc) - 0.05, max(acc) + 0.05)  # Adjust y-axis limits to

# Add accuracy values on top of the bars
for i, accuracy in enumerate(acc):
    ax.text(i, accuracy + 0.01, f'{accuracy:.4f}', ha='center')

plt.show()
```



Logistic Regression

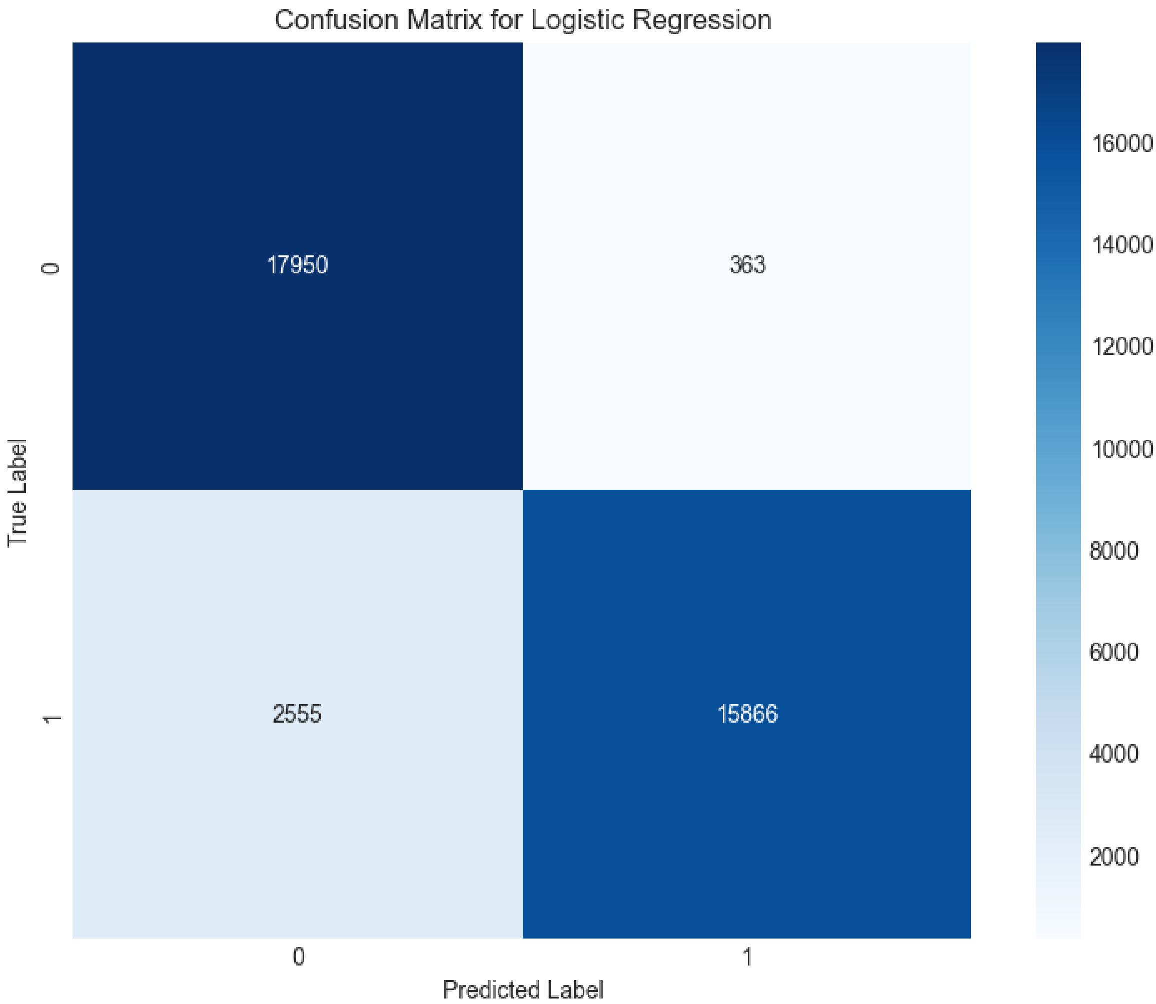
Preparation

In [38]:
```
# Initialize and Fit Logistic Regression Model
logit = LogisticRegression(random_state=random_state)

#Fit the model
logit.fit(X_train, y_train)
```

Out[38]:
```
▾ LogisticRegression
LogisticRegression(random_state=809)
```

Confusion Matrix

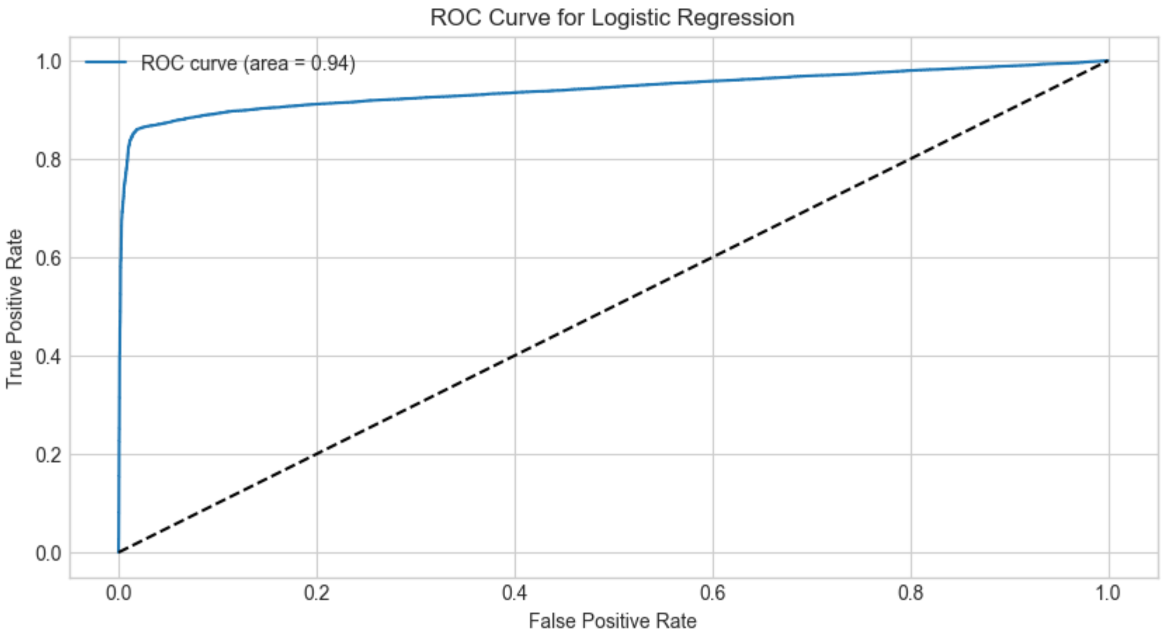
In [39]:
```
# Predictions
y_pred = logit.predict(X_test)

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Visualization of the Confusion Matrix
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', square=True)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for Logistic Regression')
plt.show()
```

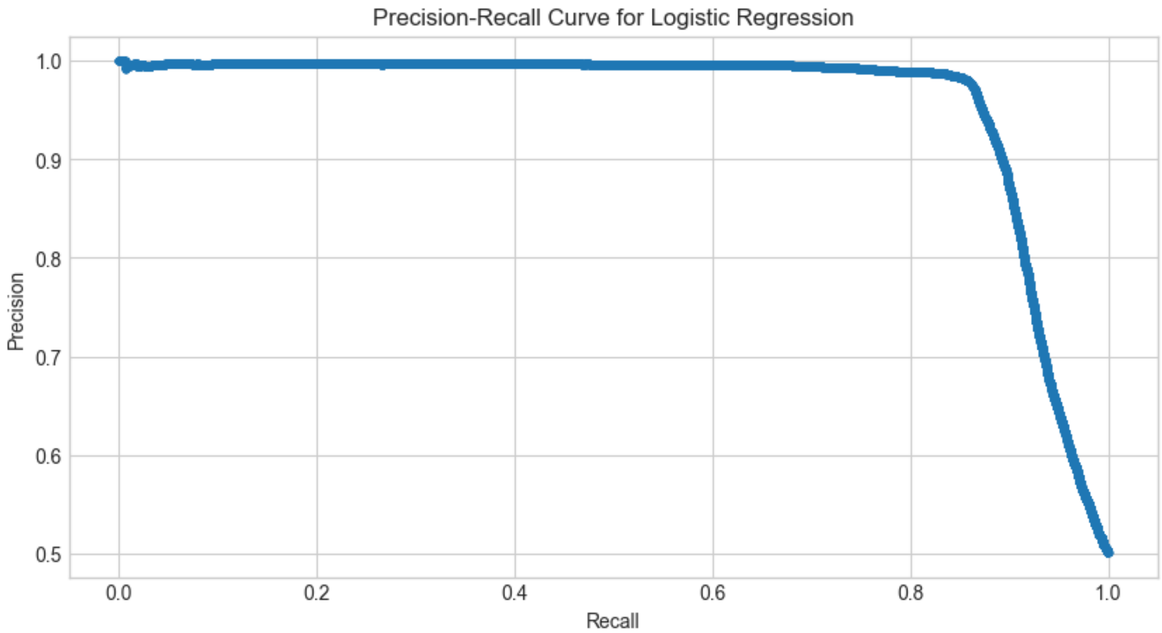


ROC Curve & Precision-Recall

In [40]:
```
# ROC Curve
y_pred_proba = logit.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
roc_auc = roc_auc_score(y_test, y_pred_proba)

plt.figure(figsize=(10, 5))
plt.plot(fpr, tpr, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Logistic Regression')
plt.legend(loc='best')
plt.show()

# Precision-Recall Curve
precision, recall, _ = precision_recall_curve(y_test, y_pred_proba)

plt.figure(figsize=(10, 5))
plt.plot(recall, precision, marker='.')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve for Logistic Regression')
plt.show()
```

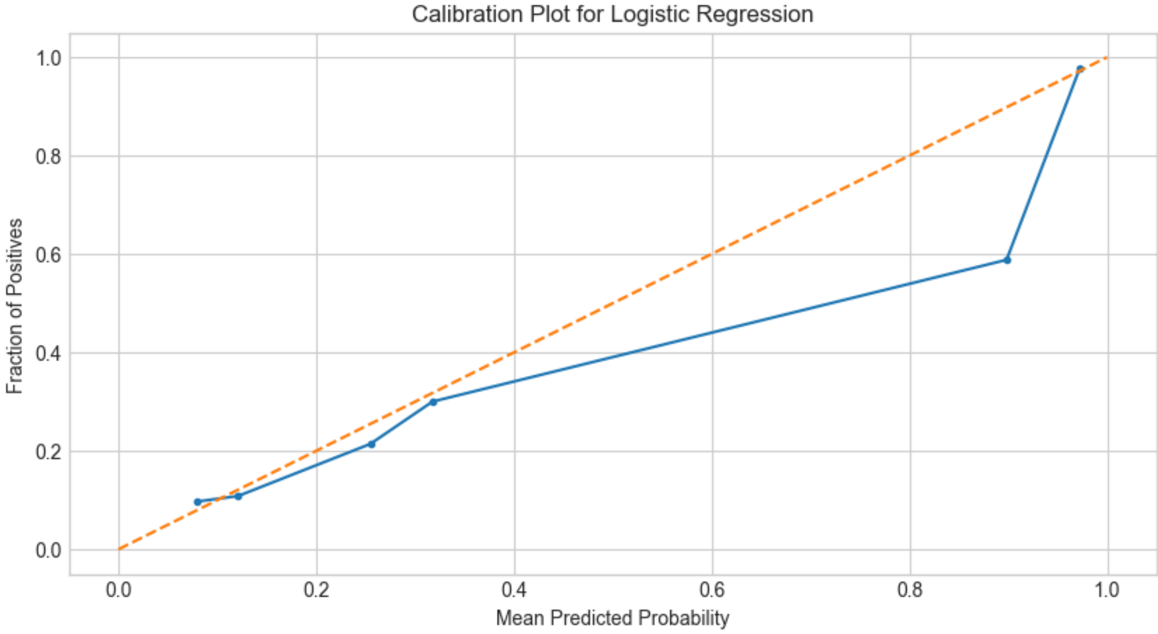




Calibration Plot

In [41]:
```
# Calibration Plot
prob_true, prob_pred = calibration_curve(y_test, y_pred_proba, n_bins=10)
```

```
plt.figure(figsize=(10, 5))
plt.plot(prob_pred, prob_true, marker='.')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('Mean Predicted Probability')
plt.ylabel('Fraction of Positives')
plt.title('Calibration Plot for Logistic Regression')
plt.show()
```



KNN

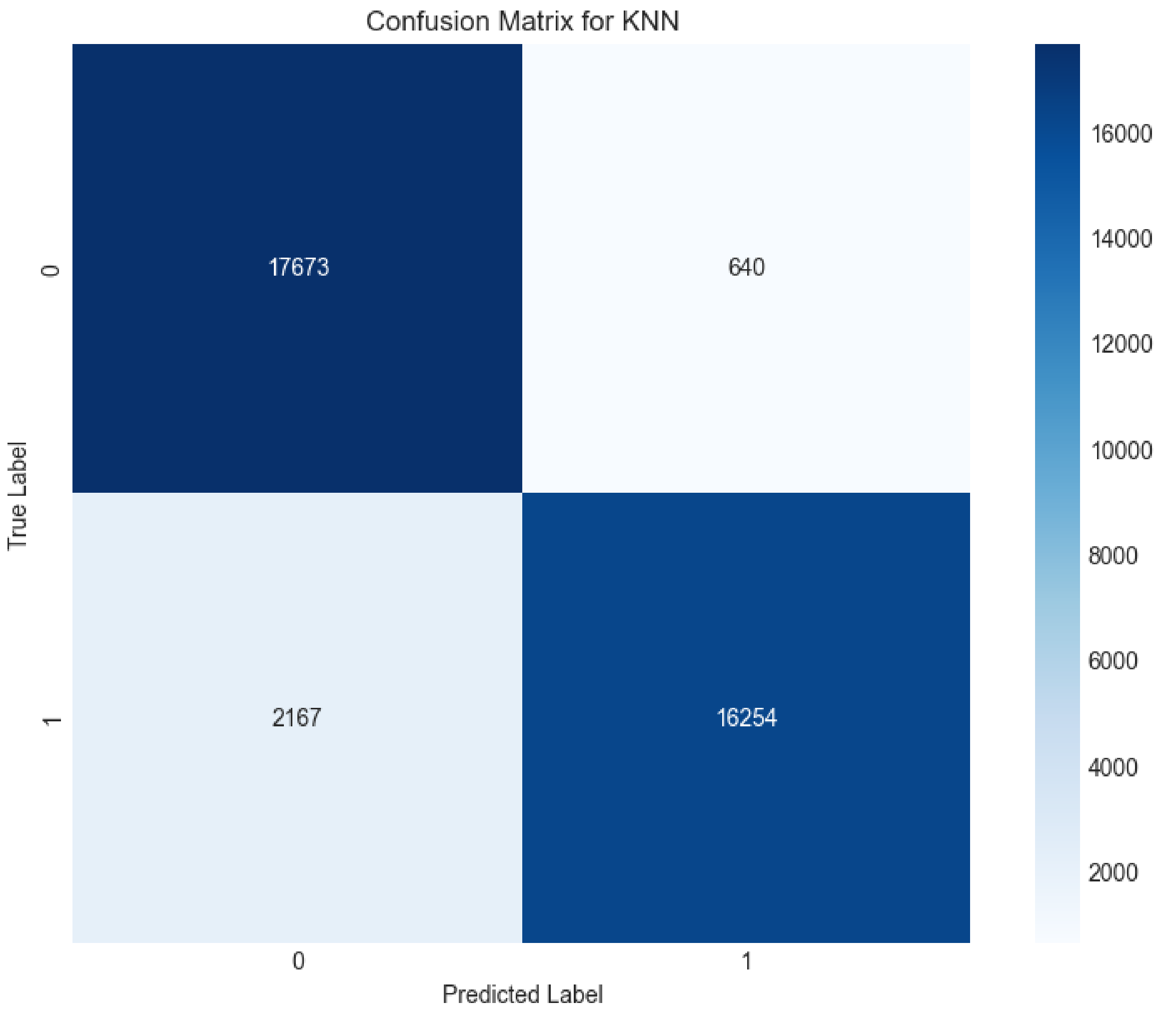
Preparation

In [42]:
```
# Initialize the KNN classifier, choosing an appropriate number of neighb
knn = KNeighborsClassifier(n_neighbors=max_k)

# Fit the model
knn.fit(X_train, y_train)
```

Out[42]:
```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=4)
```

Confusion Matrix

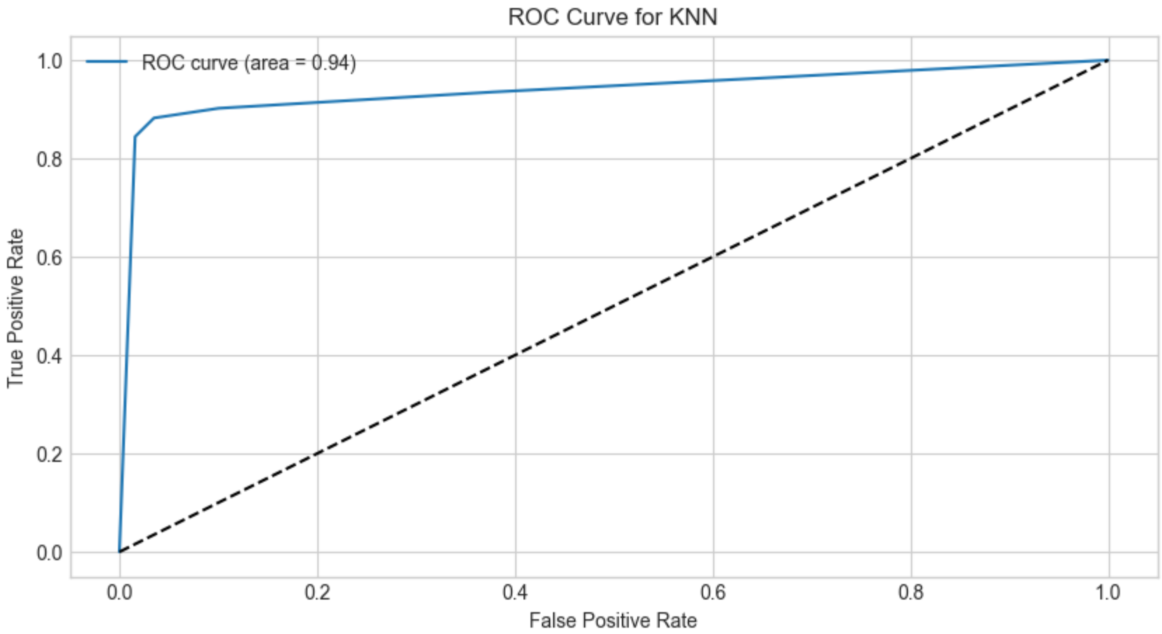
In [43]:
```
# Predictions
y_pred = knn.predict(X_test)

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Visualization
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', square=True)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for KNN')
plt.show()
```

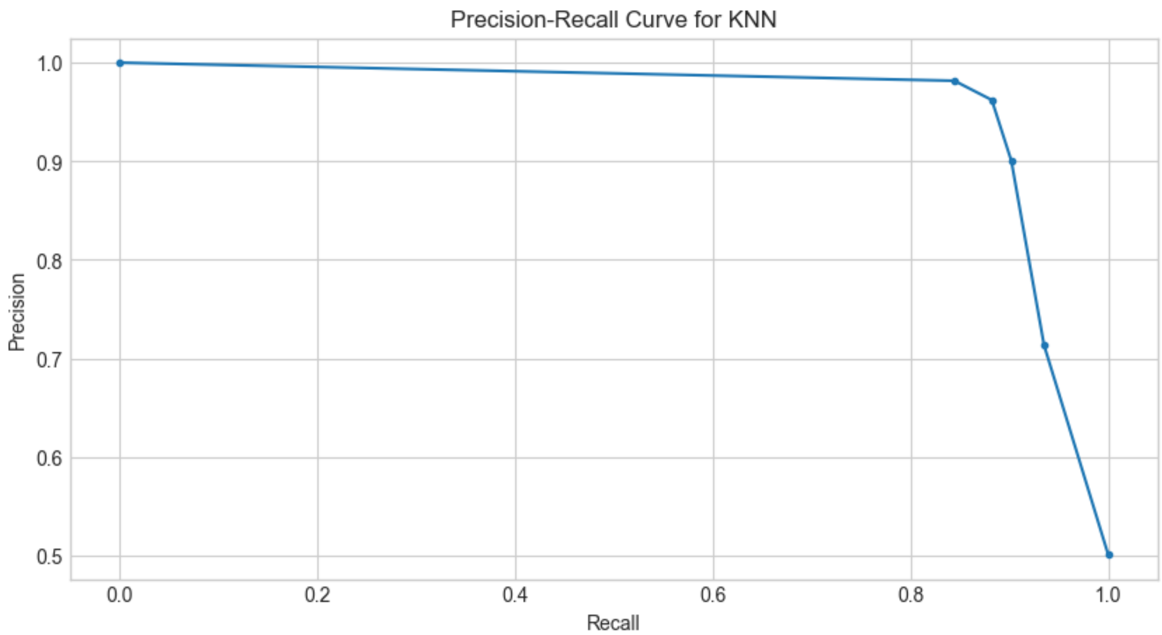


ROC Curve & Precision-Recall

In [44]:
```
# ROC Curve
y_pred_proba = knn.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
roc_auc = roc_auc_score(y_test, y_pred_proba)

plt.figure(figsize=(10, 5))
plt.plot(fpr, tpr, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for KNN')
plt.legend(loc='best')

# Precision-Recall Curve
precision, recall, _ = precision_recall_curve(y_test, y_pred_proba)

plt.figure(figsize=(10, 5))
plt.plot(recall, precision, marker='.')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve for KNN')
```

Out[44]:
```
Text(0.5, 1.0, 'Precision-Recall Curve for KNN')
```

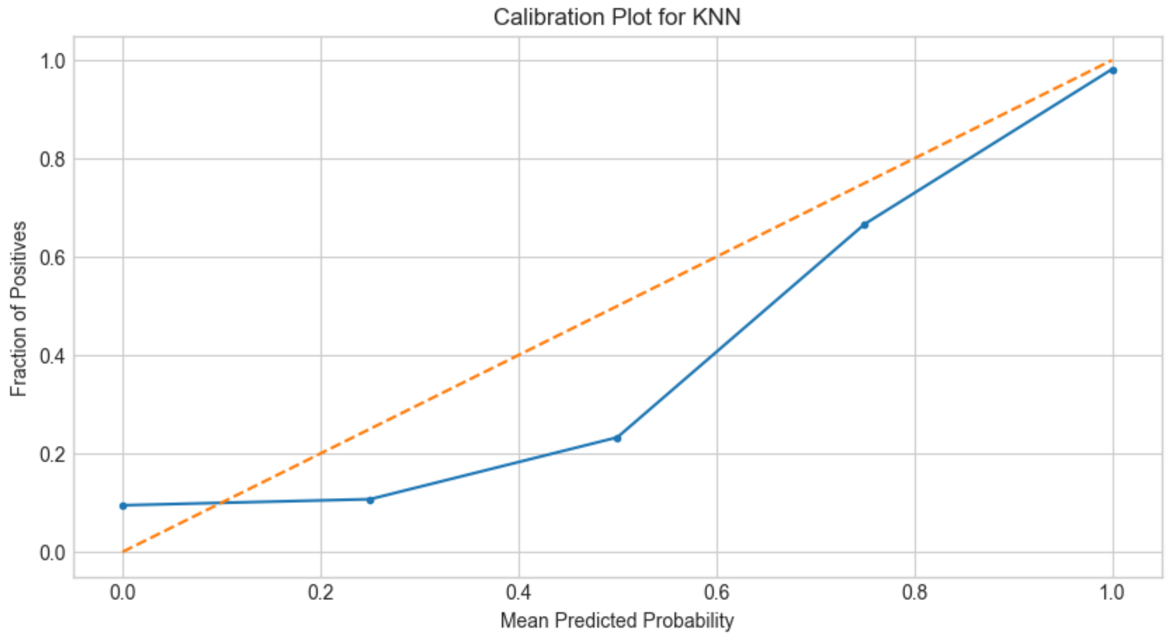




Calibration plot

In [45]:
```
# Calibration plot
prob_true, prob_pred = calibration_curve(y_test, y_pred_proba, n_bins=10)

plt.figure(figsize=(10, 5))
plt.plot(prob_pred, prob_true, marker='.')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('Mean Predicted Probability')
plt.ylabel('Fraction of Positives')
plt.title('Calibration Plot for KNN')
```

Out[45]:
```
Text(0.5, 1.0, 'Calibration Plot for KNN')
```



Random Forest

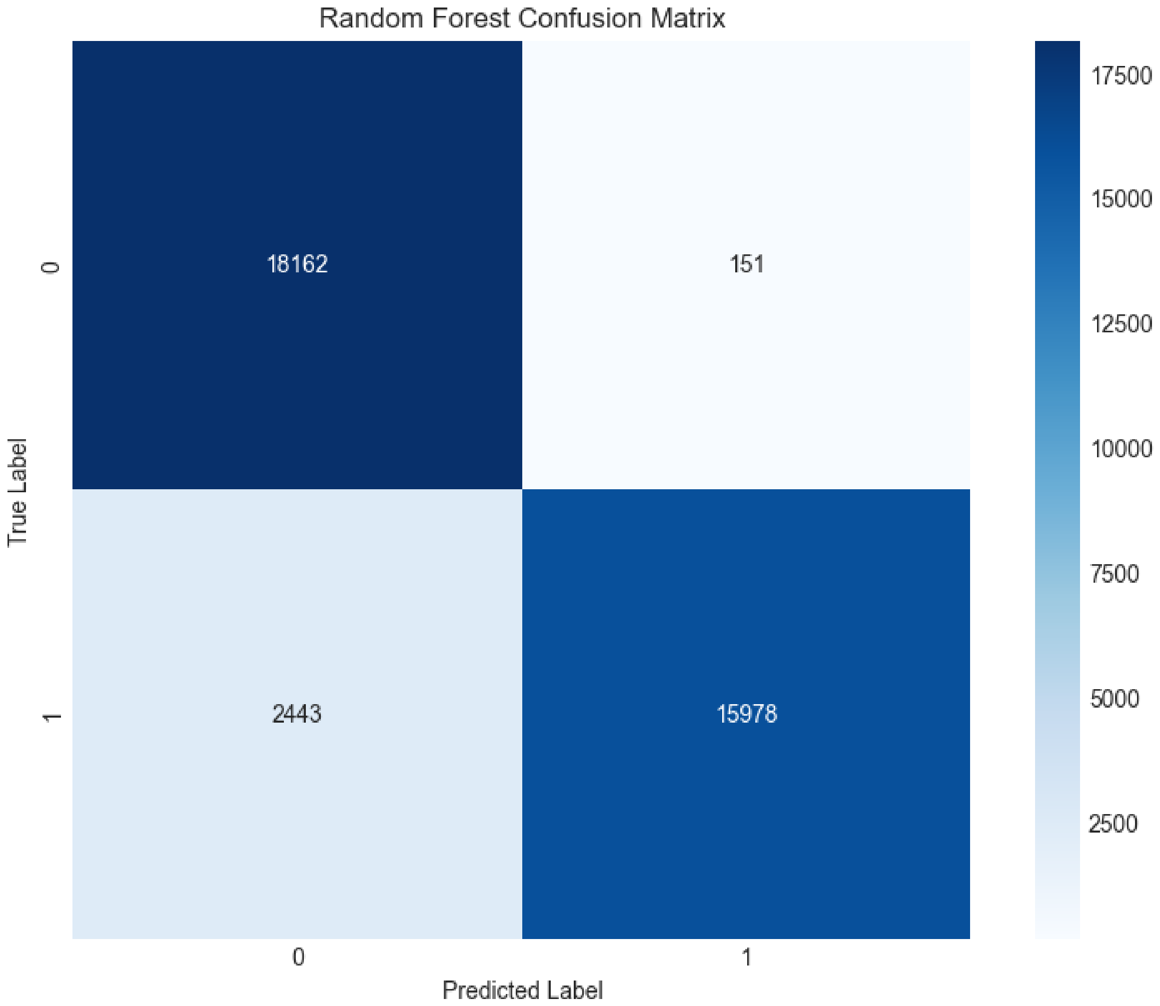
Preparation

In [46]:
```
# Initialize the Random Forest classifier
rf = RandomForestClassifier(n_estimators=100, random_state=random_state)

# Fit the model
rf.fit(X_train, y_train)
```

Out[46]:
```
▾ RandomForestClassifier
RandomForestClassifier(random_state=809)
```

Confusion Matrix

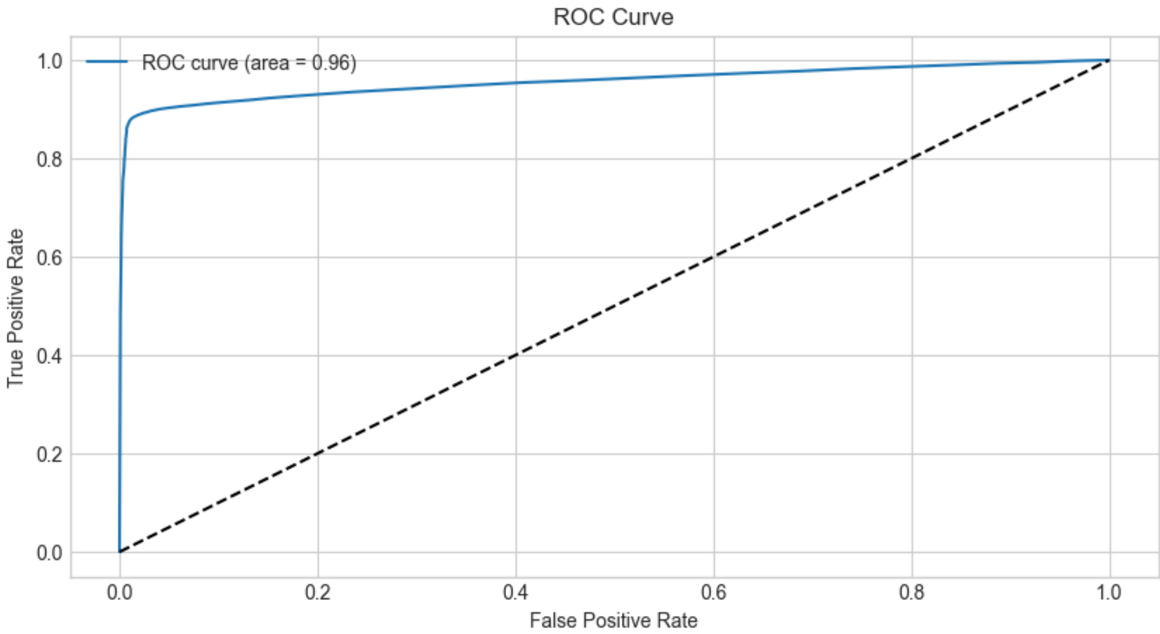
In [47]:
```
# Predictions
y_pred = rf.predict(X_test)

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Visualization
plt.figure(figsize=(10, 7))  # Adjusts the figure size for better readabi
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', square=True)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Random Forest Confusion Matrix')
plt.show()
```

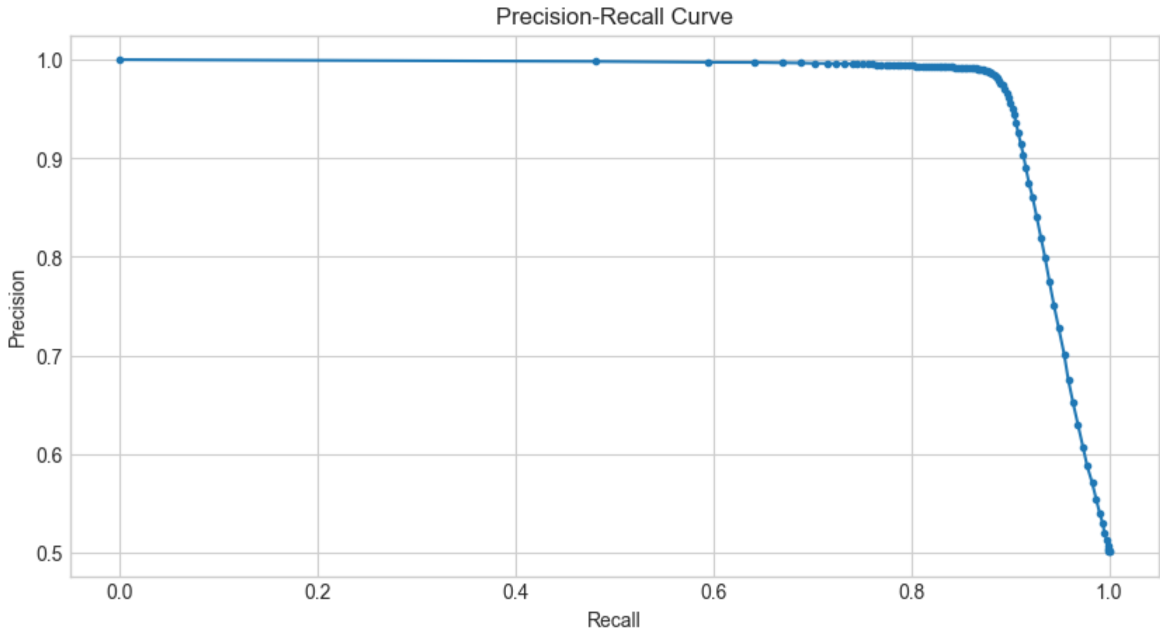


ROC Curve & Precision-Recall

In [48]:
```
# ROC Curve
y_pred_proba = rf.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
roc_auc = roc_auc_score(y_test, y_pred_proba)

plt.figure(figsize=(10, 5))
plt.plot(fpr, tpr, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='best')

# Precision-Recall Curve
precision, recall, _ = precision_recall_curve(y_test, y_pred_proba)

plt.figure(figsize=(10, 5))
plt.plot(recall, precision, marker='.')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
```

Out[48]:
```
Text(0.5, 1.0, 'Precision-Recall Curve')
```

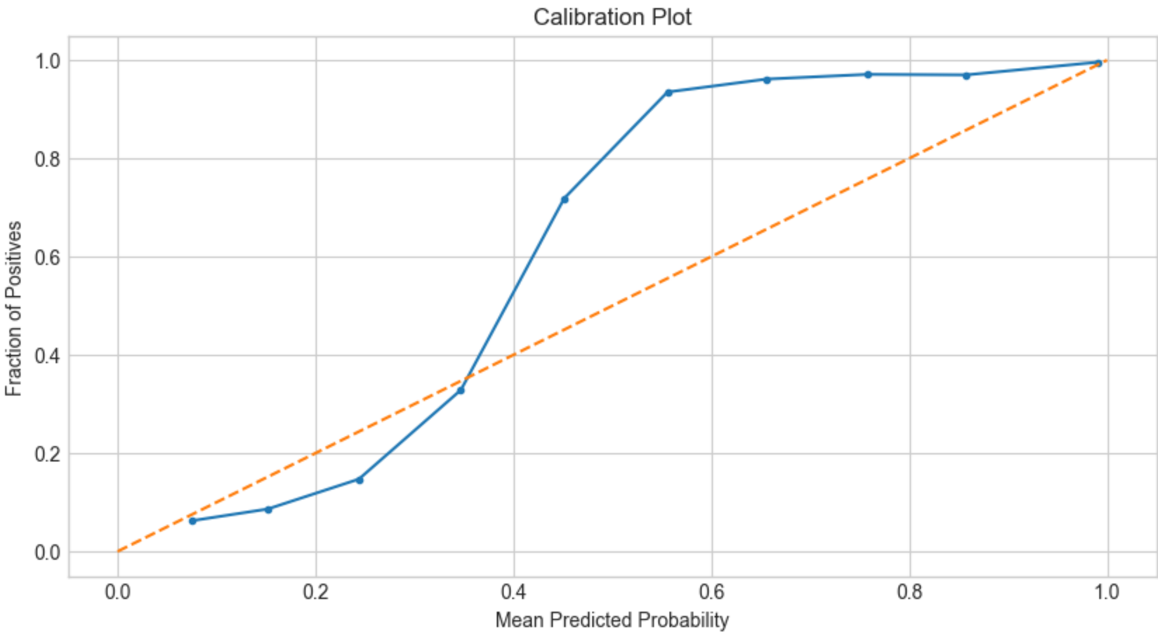




Calibration Plot

In [49]:
```
# Calibration plot
prob_true, prob_pred = calibration_curve(y_test, y_pred_proba, n_bins=10)

plt.figure(figsize=(10, 5))
plt.plot(prob_pred, prob_true, marker='.')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('Mean Predicted Probability')
plt.ylabel('Fraction of Positives')
plt.title('Calibration Plot')
```

Out[49]:
```
Text(0.5, 1.0, 'Calibration Plot')
```



Xtreme Gradient Boosting (XGBoost)

Preparation

In [50]:
```
# Initialize XGBoost Classifier
xgb_model = XGBClassifier(use_label_encoder=False, eval_metric='logloss',

# Fit the model
xgb_model.fit(X_train, y_train)
```

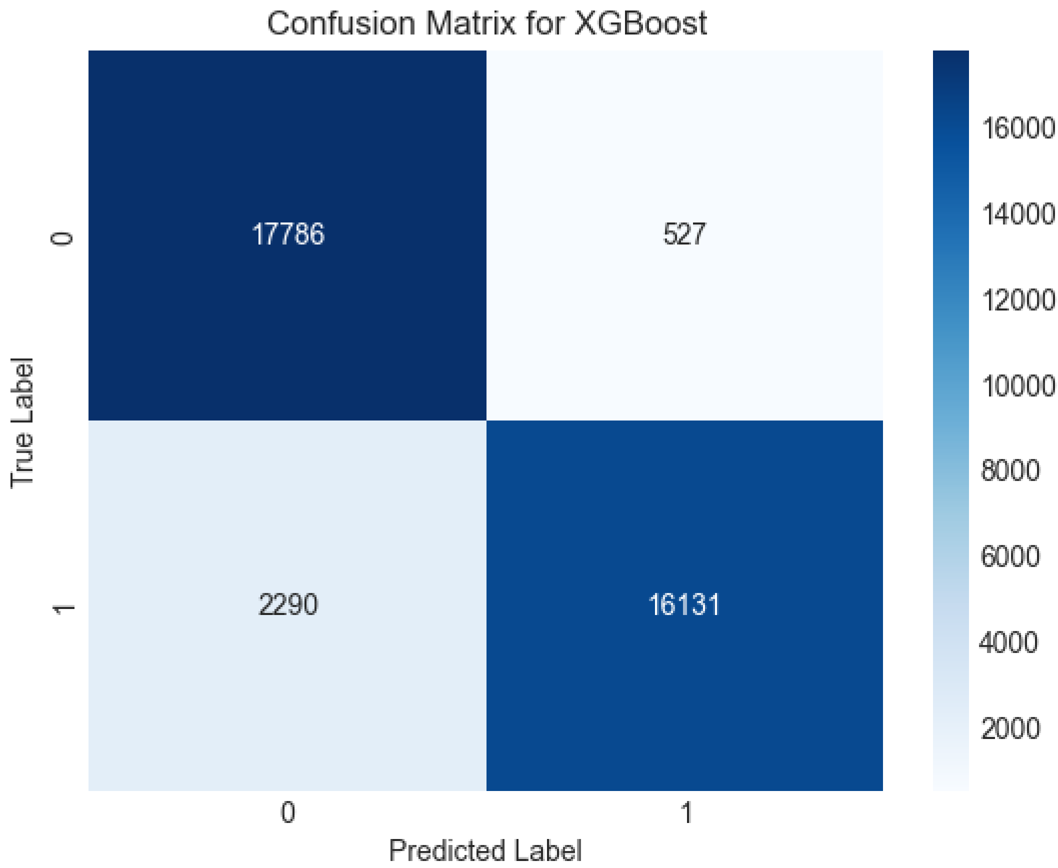
Out[50]:
```
▾ XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_
rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None, max_cat_threshold
=None,
```

Confusion Matrix

In [51]:
```
# Predictions
y_pred = xgb_model.predict(X_test)

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Visualization
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues")
plt.xlabel('Predicted Label')
```

```
plt.ylabel('True Label')
plt.title('Confusion Matrix for XGBoost')
```

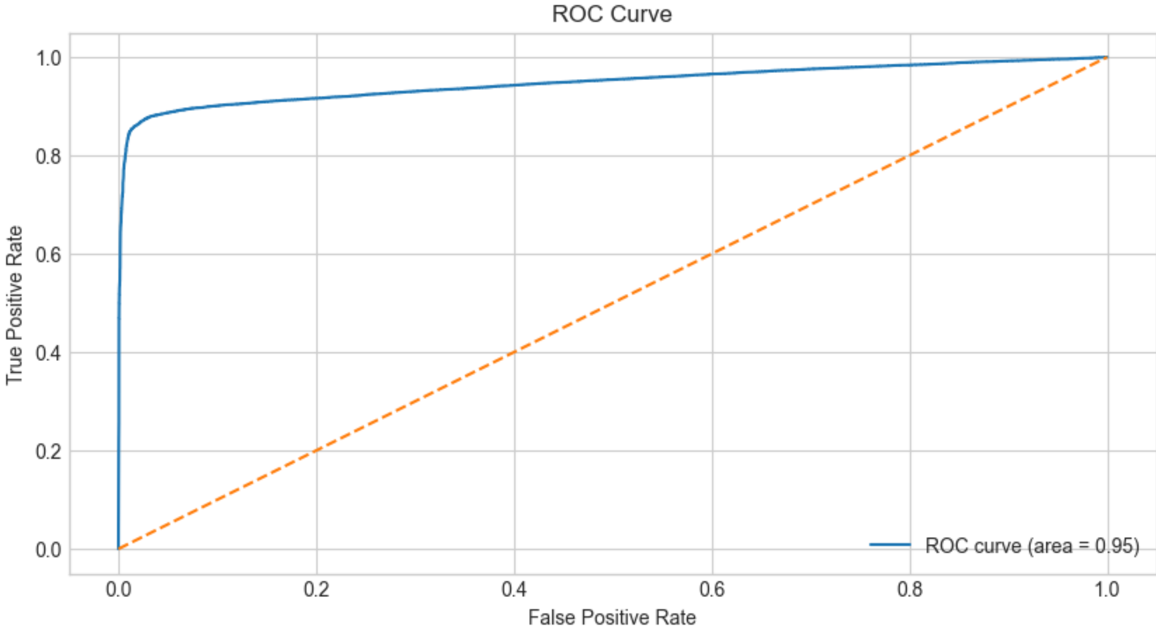
Out[51]: Text(0.5, 1.0, 'Confusion Matrix for XGBoost')

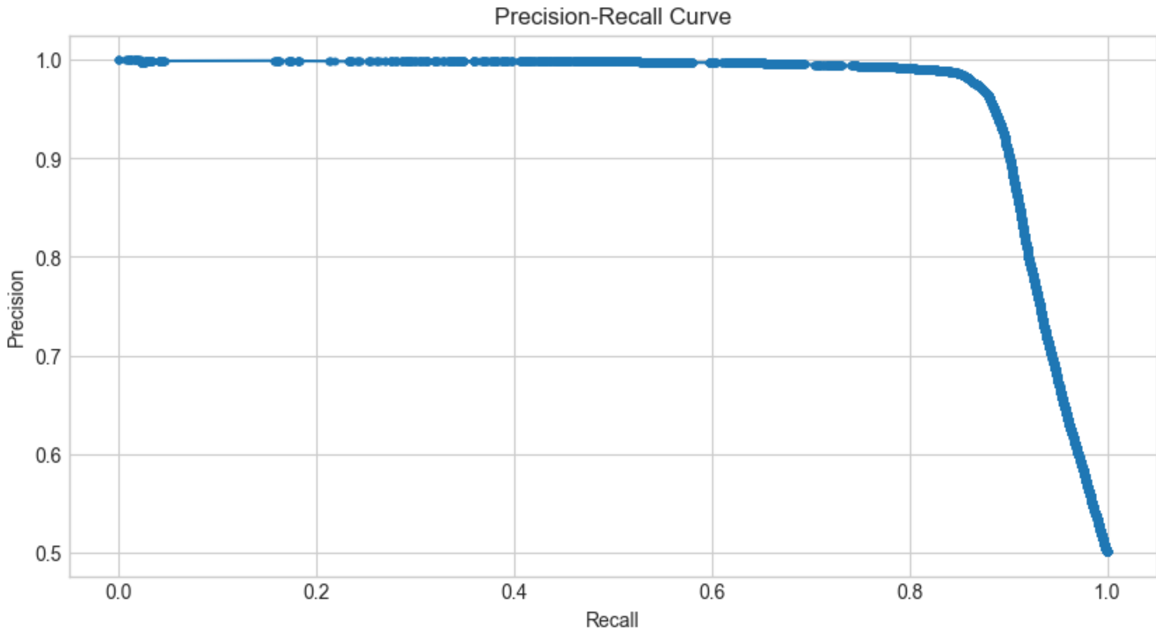


ROC Curve & Precision-Recall

In [52]:
```
# ROC Curve
y_pred_proba = xgb_model.predict_proba(X_test)[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(10, 5))
plt.plot(fpr, tpr, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")

# Precision-Recall Curve
precision, recall, _ = precision_recall_curve(y_test, y_pred_proba)

plt.figure(figsize=(10, 5))
plt.plot(recall, precision, marker='.')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
```

Out[52]: Text(0.5, 1.0, 'Precision-Recall Curve')





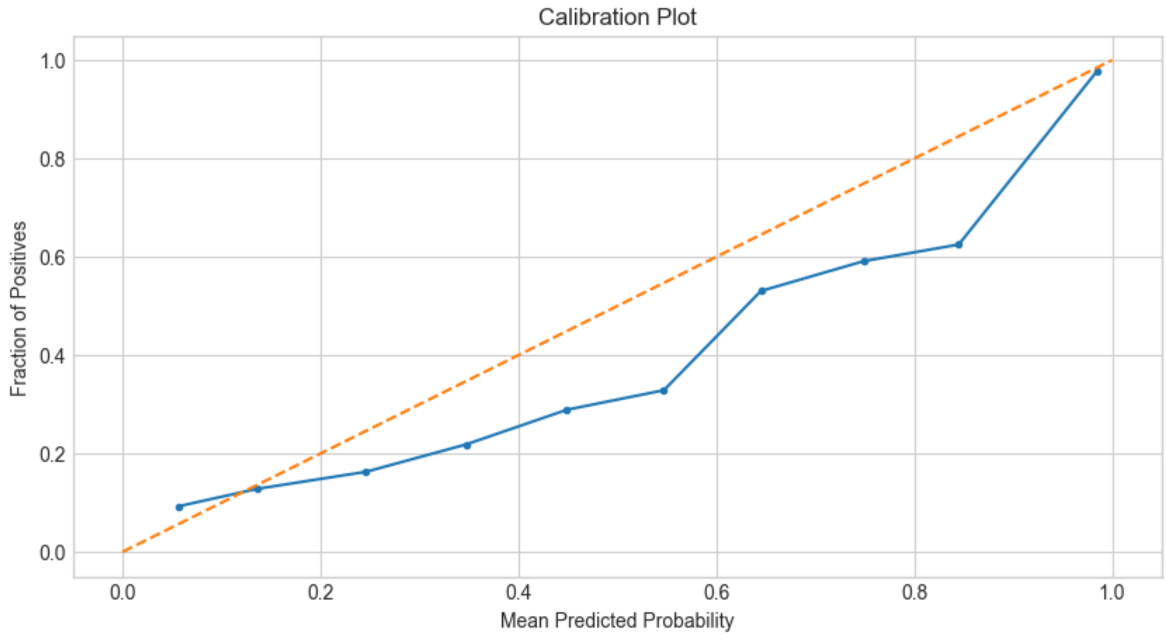
Calibration Plot

In [53]:

```
# Calibrated probabilities
calibrated = CalibratedClassifierCV(xgb_model, method='sigmoid', cv='pref
calibrated.fit(X_train, y_train)
prob_pos = calibrated.predict_proba(X_test)[:, 1]

# Calibration curve
prob_true, prob_pred = calibration_curve(y_test, prob_pos, n_bins=10)

plt.figure(figsize=(10, 5))
plt.plot(prob_pred, prob_true, marker='.')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('Mean Predicted Probability')
plt.ylabel('Fraction of Positives')
plt.title('Calibration Plot')
```

Out[53]:

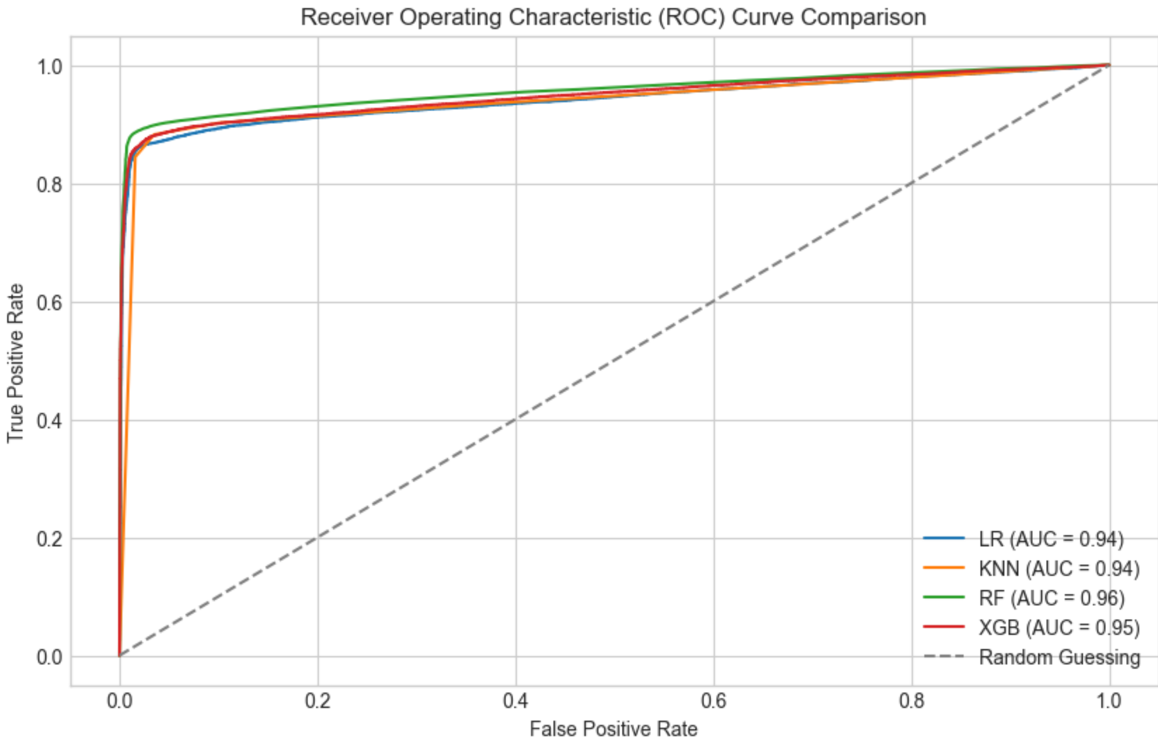
```
Text(0.5, 1.0, 'Calibration Plot')
```



Compare ROC curve

In [54]:

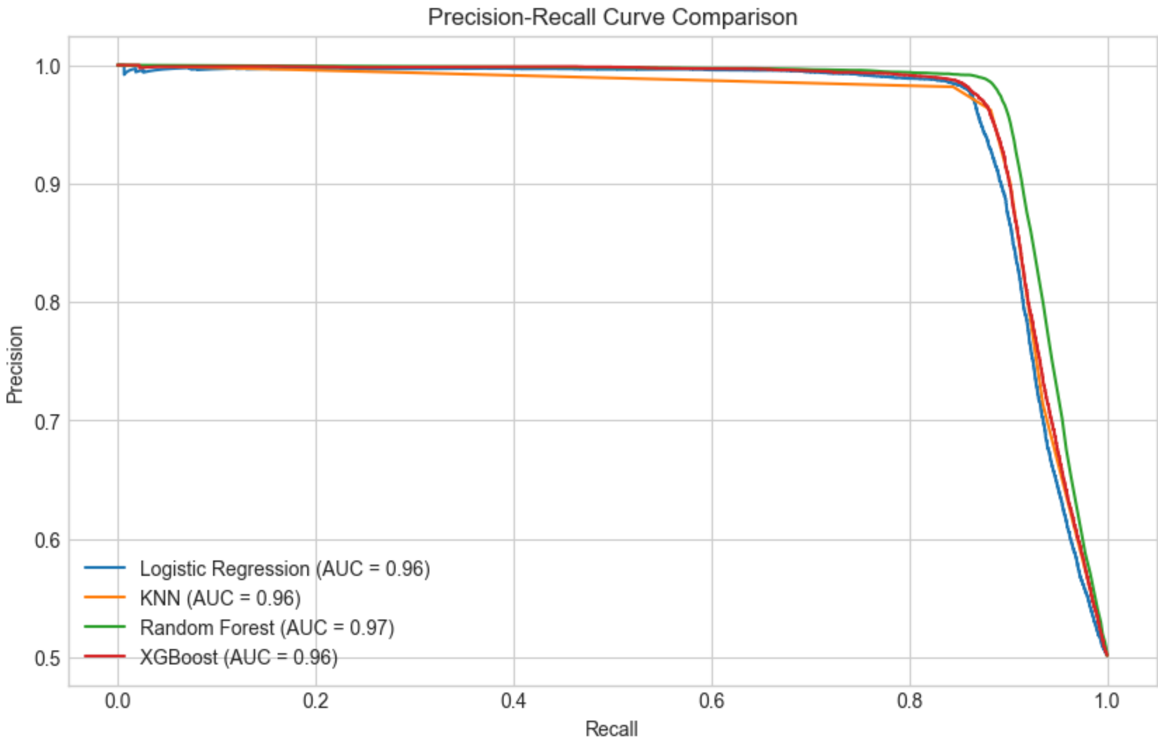
```
# Initialize models for ROC curve comparison
models_roc = [('LR', LogisticRegression(random_state=random_state)),
              ('KNN', KNeighborsClassifier(n_neighbors=max_k)),
              ('RF', RandomForestClassifier(n_estimators=100, random_stat
              ('XGB', XGBClassifier(use_label_encoder=False, eval_metric=

# Plot ROC curve for each model
plt.figure(figsize=(10, 6))
for name, model in models_roc:
    model.fit(X_train, y_train)  # Ensure models are trained before calcu
    y_pred_proba = model.predict_proba(X_test)[:, 1]
    fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'{name} (AUC = {roc_auc:.2f})')

# Plot the random guessing line
plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random Gues

# Customize the plot
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve Comparison')
plt.legend(loc='lower right')

# Show the plot
plt.show()
```



Compare Precision-Recall curve

In [55]:

```
# Initialize models for Precision-Recall curve comparison
models_pr = [('Logistic Regression', logit), ('KNN', knn), ('Random Fores

# Plot Precision-Recall curve for each model
plt.figure(figsize=(10, 6))
for name, model in models_pr:
    model.fit(X_train, y_train)  # Ensure models are trained before calcu
    y_pred_proba = model.predict_proba(X_test)[:, 1]
    precision, recall, _ = precision_recall_curve(y_test, y_pred_proba)
    pr_auc = auc(recall, precision)
    plt.plot(recall, precision, label=f'{name} (AUC = {pr_auc:.2f})')

# Customize the plot
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve Comparison')
plt.legend()

# Show the plot
plt.show()
```


Precision-Recall Curve Comparison
Precision
Recall
1.0
0.9
0.8
0.7
0.6
0.5
0.0
0.2
0.4
0.6
0.8
1.0
Logistic Regression (AUC = 0.96)
KNN (AUC = 0.96)
Random Forest (AUC = 0.97)
XGBoost (AUC = 0.96)