Clean Code

In [1]:
```
import os
import pandas as pd
import numpy as np
import math
import warnings
from sklearn import tree
from sklearn.cluster import AgglomerativeClustering
from sklearn.model_selection import train_test_split
from sklearn import metrics

import pandas as pd
import numpy as np
import math
from sklearn.ensemble import RandomForestRegressor
import category_encoders as ce
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,roc_curve, roc_auc_score, auc
import matplotlib.pyplot as plt
import matplotlib
import shap
import _pickle as cPickle
from sklearn import tree
from sklearn.impute import KNNImputer

import os
import pandas as pd
import numpy as np
import math
import warnings
from sklearn import tree
from sklearn.cluster import AgglomerativeClustering
from sklearn.model_selection import train_test_split
from sklearn import metrics

warnings.simplefilter(action='ignore', category=Warning)

# Wczytanie danych
#df = pd.read_csv('/content/drive/My Drive/praca_koncowa/loan_data_2015 2.cs
```

```
/Users/dominikamatusiak/anaconda3/lib/python3.11/site-packages/pandas/core/
arrays/masked.py:60: UserWarning: Pandas requires version '1.3.6' or newer
of 'bottleneck' (version '1.3.5' currently installed).
  from pandas.core import (
```

Cleaning Pipeline

In [2]:
```
# Important parameters
target_name='target'
id_row='id'

time_name='term'
time_report_name='Time'

event_rate_name='BR'
event_name='Bad'
prob_event='PD'
```

```
nonevent_name='Good'
share_name='Share'
variable_name='Variable'
grp_name='GRP'
all_name='All'
logit_name='Logit'
condition_name='Condition'

intercept_name='Intercept'
score_name='Score'
beta_name='Beta'
fbeta_name='FBeta'

gini_train='Gini train'
gini_test='Gini test'
delta_gini='R. Gini'
estimation='Estimation'
pvalue='P-value'
max_pvalue='Max p-value'
max_vif='Max VIF'
max_con_index='Max Con Index'
max_pearson='Max Pearson'
nnegative_betas='N negative betas'
wald_test='Wald test'
degree_free='Degrees of freedom'
std_err='Standard error'
bad_share='Bad share'
good_share='Good share'
INV='Information Value'
PSI='Population Stability Index'
PSI_tar='Population Stability Index for bads'
share_name_test='Share test'
bad_share_test='Bad share test'
type_name='Type'
percent_missing='Missing percent'
count_unique='Number of distinct'
event_value='outstanding_bad'
all_value='outstanding'

event_rate_name_value='BRBal'
nonevent_name_value='Balance good'
share_name_value='Balance share'
mode_name='Mode'
mode_pname='P. mode'
type_name='Type'
```

In [3]:
```
cols_to_exclude=['id', 'member_id','Intercept',
       'outstanding_bad', 'outstanding', 'target', 'term']
cols_to_exclude
```

Out[3]:
```
['id',
 'member_id',
 'Intercept',
 'outstanding_bad',
 'outstanding',
 'target',
 'term']
```

In [8]:
```
df.info()
# Stworzenie zmiennej docelowej 'good_bad' na podstawie kolumny 'loan_status
df['target'] = np.where(df['loan_status'].isin(['Charged Off', 'Default', 'L
                                                'Does not meet the credit
df.drop(columns=['loan_status'], inplace=True)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[8], line 1
----> 1 df.info()
      2 # Stworzenie zmiennej docelowej 'good_bad' na podstawie kolumny 'lo
an_status'
      3 df['target'] = np.where(df['loan_status'].isin(['Charged Off', 'Def
ault', 'Late (31-120 days)',
      4                                                 'Does not meet th
e credit policy. Status:Charged Off']), 1, 0)

NameError: name 'df' is not defined
```

In [ ]:
```
target_counts = df[target_name].value_counts()
count_zeros = target_counts.get(0, 0)
count_ones = target_counts.get(1, 0)

print(f"Liczba zer: {count_zeros}")
print(f"Liczba jedynek: {count_ones}")
```

In [ ]:
```
# cleaning up the emp_length column, assign 0 to NANs, and convert to numer
def emp_length_converter(df, column):
    df[column] = df[column].str.replace(r'\+ years', '', regex=True)
    df[column] = df[column].str.replace(r'< 1 year', '0', regex=True)
    df[column] = df[column].str.replace(r' years', '', regex=True)
    df[column] = df[column].str.replace(r' year', '', regex=True)
    df[column] = df[column].str.replace(r'\+', '', regex=True)  # Handle "10
    df[column] = pd.to_numeric(df[column])
    df[column].fillna(value=0, inplace=True)

emp_length_converter(df, 'emp_length')

print(df['emp_length'].unique())
```

In [ ]:
```
# converting date columns to datetime format and create a new column as a di
def date_columns(df, column):
    today_date = pd.to_datetime('2020-08-01')
    df[column] = pd.to_datetime(df[column], format = "%b-%y")
    df['mths_since_' + column] = round(pd.to_numeric((today_date - df[colum
    df['mths_since_' + column] = df['mths_since_' + column].apply(lambda x:
    df.drop(columns = [column], inplace = True)

date_columns(df, 'earliest_cr_line')
date_columns(df, 'issue_d')
date_columns(df, 'last_pymnt_d')
date_columns(df, 'last_credit_pull_d')

print(df['mths_since_earliest_cr_line'].describe())
print(df['mths_since_issue_d'].describe())
print(df['mths_since_last_pymnt_d'].describe())
print(df['mths_since_last_credit_pull_d'].describe())
```

In [ ]:
```
# function to remove 'months' string from the 'term' column and convert it t
def loan_term_converter(df, column):
    df[column] = pd.to_numeric(df[column].str.replace(' months', ''))

loan_term_converter(df, 'term')
```

Binning - Kod inspirowany (Dr. Karol Przanowski)

In [ ]:
```
# Binning for numerical variables
ncategories_int=4
minimum_share_int=0.01
symbol_missing='Missing'

# Binning for character variables
symbol_other='<OTHERS>'
minimum_share_unique=0.01
ncategories_nom=4

category_order=False

# Dodanie kolumny intercept
df[intercept_name] = 1
df[event_value] = df['loan_amnt'] * df[target_name]
df[all_value] = df['loan_amnt']
df.head()
```

In [ ]:
```
vars = [var for var in list(df) if var not in cols_to_exclude]
vars
```

In [ ]:
```
#splitting into numeric and character variables
varsc=list(df[vars].select_dtypes(include='object'))
varsn=list(df[vars].select_dtypes(include='number'))
```

In [ ]:
```
# Podział na zestawy treningowe i testowe
train, test = train_test_split(df[vars_target_id], random_state=1234, test_s
test, validation = train_test_split(test, random_state=1234, test_size=0.1)
print(train.shape, test.shape, validation.shape)
```

In [ ]:
```
# Binning dla zmiennych numerycznych
labsn = {}

for feature in varsn:
    print(f"Processing feature: {feature}")
    miss_share = train[feature].isnull().sum() / train[feature].shape[0]
    miss_share = 1 - miss_share
    if miss_share <= 0.00001:
        miss_share = 1
    minimum_share = minimum_share_int / miss_share
    if minimum_share > 0.5:
        minimum_share = 0.5
    if minimum_share < minimum_share_int:
        minimum_share = minimum_share_int
    df_two_col = train[[target_name, feature]].dropna(subset=[feature]).copy
    bins = [-np.inf, np.inf]
    clf = tree.DecisionTreeClassifier(
        max_leaf_nodes=ncategories_int,
        min_weight_fraction_leaf=minimum_share,
        random_state=1234)
    clf.fit(df_two_col[feature].values.reshape(-1, 1), df_two_col[target_nar
    thresh = [round(s, 3) for s in clf.tree_.threshold if s != -2]
    bins = bins + thresh
    bins = sorted(bins)
    print(f"Thresholds: {thresh}")
    print(f"Bins: {bins}")
    if train[feature].isnull().sum() / train[feature].shape[0] > minimum_sha
        bins = bins + [symbol_missing]
    labsn[feature] = bins

# Binning dla zmiennych kategorycznych
labsc = {}
```

```
for feature in varsc:
    print(f"Processing feature: {feature}")
    df_two_col1 = pd.DataFrame(train.groupby(feature)[target_name].count() /
    df_two_col2 = pd.DataFrame(train.groupby(feature)[target_name].mean())
    df_two_col = df_two_col2
    df_two_col['share'] = df_two_col1[target_name]
    df_two_col = df_two_col.loc[df_two_col['share'] > minimum_share_unique]
    ncategoriesv = min(ncategories_nom, df_two_col.shape[0])
    if ncategoriesv >= 2:
        cluster = AgglomerativeClustering(n_clusters=ncategoriesv, affinity=
        cluster.fit_predict(df_two_col[[target_name]])
        df_two_col['cluster'] = cluster.labels_.reshape(-1, 1)
    else:
        df_two_col['cluster'] = 0
    bins = df_two_col[['cluster']]
    if df_two_col['share'].sum() < (1 - minimum_share_unique):
        bins.loc[symbol_other] = -1
        bins['cluster'] = bins['cluster'] + 1
    bins = bins.sort_values(by='cluster').reset_index()
    labsc[feature] = bins
    print(f"Clusters: {bins}")

# Tworzenie zmiennych GRP dla zmiennych numerycznych
feature_intervalsn = {}

for feature in varsn:
    intervals = []
    tekst = ''
    for i in range(len(labsn[feature])):
        if i == 0 and labsn[feature][i] == -np.inf and labsn[feature][i+1]
            tekst = feature + ' < ' + str(labsn[feature][i+1])
            intervals = intervals + [tekst]
        if i == 0 and labsn[feature][i] == -np.inf and labsn[feature][i+1] =
            tekst = feature + ' <> ' + symbol_missing
            intervals = intervals + [tekst]
        if i > 0 and labsn[feature][i-1] != -np.inf and labsn[feature][i] ==
            tekst = str(labsn[feature][i-1]) + ' <= ' + feature
            intervals = intervals + [tekst]
        if i > 0 and labsn[feature][i-1] != -np.inf and labsn[feature][i] !=
            tekst = str(labsn[feature][i-1]) + ' <= ' + feature + ' < ' + st
            intervals = intervals + [tekst]
        if labsn[feature][i] == symbol_missing:
            tekst = feature + ' = ' + symbol_missing
            intervals = intervals + [tekst]
    feature_intervalsn[feature] = intervals
    print(f"Intervals for {feature}: {intervals}")

# Tworzenie zmiennych GRP dla zmiennych kategorycznych
feature_intervalsc = {}

for feature in varsc:
    intervals = []
    tekst = ''
    for i in range(labsc[feature].shape[0]):
        if i == 0:
            tekst = labsc[feature][feature][i]
        if i > 0:
            if labsc[feature]['cluster'][i-1] == labsc[feature]['cluster'][i
                tekst = tekst + ', ' + labsc[feature][feature][i]
            else:
                intervals = intervals + [tekst]
                tekst = labsc[feature][feature][i]
        if i + 1 == labsc[feature].shape[0]:
```

```
            intervals = intervals + [tekst]
    feature_intervalsc[feature] = intervals
    print(f"Intervals for {feature}: {intervals}")

# Tworzenie zmiennych GRP w zestawie treningowym
train_grp = train.copy()
nnn = 0

for feature in varsn:
    nnn += 1
    print(nnn, feature)
    def grp(x):
        res = np.NaN
        for i in range(len(labsn[feature])):
            if i == 0 and labsn[feature][i] == -np.inf and labsn[feature][i-
                res = i
            if i == 0 and labsn[feature][i] == -np.inf and labsn[feature][i-
                res = i
            if i > 0 and labsn[feature][i-1] != -np.inf and labsn[feature][i
                res = i - 1
            if i > 0 and labsn[feature][i-1] != -np.inf and labsn[feature][i
                res = i - 1
            if labsn[feature][i] == symbol_missing and math.isnan(x):
                res = i - 1
        return res
    train_grp[feature] = train_grp[feature].apply(grp)

for feature in varsc:
    nnn += 1
    print(nnn, feature)
    def grp(x):
        res = np.NaN
        if type(x) != str:
            x = str('')
        if labsc[feature][feature][0] == symbol_other:
            res = 0
        for i in range(labsc[feature].shape[0]):
            if x == labsc[feature][feature][i] and labsc[feature][feature][i
                res = labsc[feature]['cluster'][i]
        return res
    train_grp[feature] = train_grp[feature].apply(grp)

train_grp.head()
```

In [ ]:

```
Big_scorecard = pd.DataFrame()
sum = train_grp.shape[0]
sum_bad = train_grp[target_name].sum()
sum_good = sum - sum_bad

for feature in varsn:
    fin = pd.DataFrame()
    sss = pd.DataFrame()
    ddd = pd.DataFrame()
    sss = pd.DataFrame(train_grp.groupby(feature).agg({target_name: ['sum',
    sss = pd.DataFrame(sss[target_name])
    sss = sss.reset_index()
    sss = sss.rename(columns={"sum": event_name, "count": all_name, feature
    sss[nonevent_name] = sss[all_name] - sss[event_name]
    sss[event_rate_name] = sss[event_name] / sss[all_name]
    sss[logit_name] = np.log((sss[event_name] + 0.0001) / (sss[nonevent_name
    sss[share_name] = sss[all_name] / sum
    sss[variable_name] = feature
    ddd = pd.DataFrame({grp_name: range(len(feature_intervalsn[feature])), 
    fin = pd.merge(ddd, sss, on=grp_name)
```

```
    fin = fin[[variable_name, condition_name, event_rate_name, share_name, a
    fin = fin.sort_values(by=[event_rate_name], ascending=category_order)
    fin = fin.reset_index(drop=True)
    fin[grp_name] = fin.index
    fin[type_name] = 'INT'
    Big_scorecard = pd.concat([Big_scorecard, fin], ignore_index=True, sort=

for feature in varsc:
    fin = pd.DataFrame()
    sss = pd.DataFrame()
    ddd = pd.DataFrame()
    sss = pd.DataFrame(train_grp.groupby(feature).agg({target_name: ['sum',
    sss = pd.DataFrame(sss[target_name])
    sss = sss.reset_index()
    sss = sss.rename(columns={"sum": event_name, "count": all_name, feature
    sss[nonevent_name] = sss[all_name] - sss[event_name]
    sss[event_rate_name] = sss[event_name] / sss[all_name]
    sss[logit_name] = np.log((sss[event_name] + 0.0001) / (sss[nonevent_name
    sss[share_name] = sss[all_name] / sum
    sss[variable_name] = feature
    ddd = pd.DataFrame({grp_name: range(len(feature_intervalsc[feature])), c
    fin = pd.merge(ddd, sss, on=grp_name)
    fin = fin[[variable_name, condition_name, event_rate_name, share_name, a
    fin = fin.sort_values(by=[event_rate_name], ascending=category_order)
    fin = fin.reset_index(drop=True)
    fin[grp_name] = fin.index
    fin[type_name] = 'NOM'
    Big_scorecard = pd.concat([Big_scorecard, fin], ignore_index=True, sort=

Big_scorecard[bad_share] = Big_scorecard[event_name] / sum_bad
Big_scorecard[good_share] = Big_scorecard[nonevent_name] / sum_good
Big_scorecard[INV] = (Big_scorecard[good_share] - Big_scorecard[bad_share])
Big_scorecard.to_excel('Big_scorecard.xlsx', index=False)
Big_scorecard.head()
```

In [ ]:

```
# Tworzenie grp_train i grp_test
grp_train = train.copy()
grp_test = test.copy()

for feature in varsn:
    sub = pd.DataFrame()
    sub = Big_scorecard[Big_scorecard[variable_name] == feature]
    sub = sub.reset_index()
    def grp(x):
        res = sub[grp_name][0]
        for i in range(sub.shape[0]):
            sl = -np.inf
            sr = np.inf
            fl = sub[condition_name][i].find(' <= ')
            fr = sub[condition_name][i].find(' < ')
            if fl >= 0:
                sl = float(sub[condition_name][i][0:fl])
            if fr >= 0:
                sr = float(sub[condition_name][i][fr+3:])
            fm = sub[condition_name][i].find(' = ' + symbol_missing)
            if fm >= 0 and math.isnan(x):
                res = sub[grp_name][i]
            if fm < 0 and (sl <= x < sr):
                res = sub[grp_name][i]
        return res
    grp_train[feature] = grp_train[feature].apply(grp)
    grp_test[feature] = grp_test[feature].apply(grp)

for feature in varsc:
```

```
    sub = pd.DataFrame()
    sub = Big_scorecard[Big_scorecard[variable_name] == feature]
    sub = sub.reset_index()
    def grp(x):
        res = sub[grp_name][0]
        if type(x) != str:
            x = str('')
        for i in range(sub.shape[0]):
            fo = sub[condition_name][i].find(symbol_other)
            if fo >= 0:
                res = sub[grp_name][i]
        for i in range(sub.shape[0]):
            fo = sub[condition_name][i].find(symbol_other)
            if fo < 0 and sub[condition_name][i].find(x) >= 0:
                res = sub[grp_name][i]
        return res
    grp_train[feature] = grp_train[feature].apply(grp)
    grp_test[feature] = grp_test[feature].apply(grp)

grp_train.head()
```

In [ ]:

```
Big_scorecard_test = pd.DataFrame()
sum = grp_test.shape[0]
sum_bad = grp_test[target_name].sum()

for feature in vars:
    sss = pd.DataFrame()
    sss = pd.DataFrame(grp_test.groupby(feature).agg({target_name: ['sum',
    sss = pd.DataFrame(sss[target_name])
    sss = sss.reset_index()
    sss = sss.rename(columns={"sum": event_name, "count": all_name, feature
    sss[nonevent_name] = sss[all_name] - sss[event_name]
    sss[share_name_test] = sss[all_name] / sum
    sss[bad_share_test] = sss[event_name] / sum_bad
    sss[variable_name] = feature
    sss = sss[[variable_name, grp_name, share_name_test, bad_share_test]]
    Big_scorecard_test = pd.concat([Big_scorecard_test, sss], ignore_index=T

Big_scorecard_test.head()

# Obliczanie PSI INV
Big_scorecard = pd.merge(Big_scorecard, Big_scorecard_test, on=[variable_nar
Big_scorecard[PSI] = (Big_scorecard[share_name] - Big_scorecard[share_name_t
Big_scorecard[PSI_tar] = (Big_scorecard[bad_share] - Big_scorecard[bad_share

Big_scorecard.head()
```

In [ ]:

```
# Obliczanie wartości Gini dla zmiennych
Gini_vars = pd.DataFrame()

for feature in vars:
    sss = pd.DataFrame([feature], columns=[variable_name])
    fpr, tpr, thresholds = metrics.roc_curve(logit_train[target_name], logit
    gini = np.absolute(2 * metrics.auc(fpr, tpr) - 1)
    sss[gini_train] = gini
    fpr, tpr, thresholds = metrics.roc_curve(logit_test[target_name], logit_
    gini = np.absolute(2 * metrics.auc(fpr, tpr) - 1)
    sss[gini_test] = gini
    Gini_vars = pd.concat([Gini_vars, sss], ignore_index=True, sort=False)

Gini_vars[delta_gini] = np.absolute(Gini_vars[gini_train] - Gini_vars[gini_t
Gini_vars[delta_gini] = Gini_vars[delta_gini].fillna(0)
Gini_vars = Gini_vars.sort_values(by=[gini_train], ascending=False)
```

```
Gini_vars = Gini_vars.reset_index(drop=True)

sss = pd.DataFrame(Big_scorecard.groupby(variable_name).agg({INV: ['sum'], P
sss.columns = [INV, PSI, PSI_tar]
sss = sss.reset_index()

pm = train[vars].isnull().sum() / len(train)
stat = pd.DataFrame({'percent_missing': pm}).reset_index()
stat.columns = [variable_name, percent_missing]
stat[count_unique] = 0
for i in range(stat.shape[0]):
    stat.loc[i, count_unique] = len(pd.unique(train[stat.loc[i, variable_nar

Gini_vars = pd.merge(Gini_vars, sss, on=variable_name)
Gini_vars = pd.merge(Gini_vars, stat, on=variable_name)

fin = pd.DataFrame()

for feature in vars:
    row = pd.DataFrame(np.array([[1, 2, 3, 4]]), columns=[variable_name, mod
    row[variable_name] = feature
    ttt = train[[feature]].copy()
    ttt = ttt[ttt[feature].notnull()]
    pm2 = ttt[feature].mode()
    row[mode_name] = pm2[0]
    ttt2 = pd.DataFrame(ttt.groupby(feature)[feature].count() / ttt.shape[0]
    ttt2.columns = ['count']
    ttt2 = ttt2.reset_index()
    ttt2.columns = ['var', 'count']
    ttt2 = ttt2[ttt2['var'] == pm2[0]]
    ttt2 = ttt2.reset_index()
    row[mode_pname] = ttt2.loc[0, 'count']
    row[type_name] = 'INT'
    for f2 in varsc:
        if f2 == feature:
            row[type_name] = 'NOM'
    fin = pd.concat([fin, row], ignore_index=True, sort=False)

Gini_vars = pd.merge(Gini_vars, fin, on=variable_name)

Gini_vars.to_excel('Gini_vars.xlsx', index=False)
Gini_vars.head()
```

In [ ]:
```
# Przetwarzanie wstępne
sub = Gini_vars[
    (Gini_vars[gini_train] > 0.01)
    & (Gini_vars[gini_train] < 0.4) #utrudnienie
    & (Gini_vars[delta_gini] < 1)
    & (Gini_vars[PSI_tar] < 0.5)
    & (Gini_vars[PSI] < 0.5)
].copy()
sub.head()
```

In [ ]:
```
# Funkcja do usuwania jednej najbardziej skorelowanej zmiennej
def remove_highly_correlated(corr_matrix, threshold=0.90):
    upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astyp
    high_corr_pairs = [(column, idx) for column in upper.columns for idx in

    count_dict = {}
    for pair in high_corr_pairs:
        count_dict[pair[0]] = count_dict.get(pair[0], 0) + 1
        count_dict[pair[1]] = count_dict.get(pair[1], 0) + 1
```

```
    if not count_dict:
        return None
    to_remove = max(count_dict, key=count_dict.get)
    return to_remove

# Iteracyjne usuwanie wysoko skorelowanych zmiennych
removed_vars = []
while True:
    corr_matrix = grp_train[keep_list].corr().abs()
    to_remove = remove_highly_correlated(corr_matrix)
    if to_remove is None:
        break
    removed_vars.append(to_remove)
    keep_list.remove(to_remove)

keep_list
```

In [ ]:
```
!pip install xlsxwriter
```

In [ ]:
```
do_not_keep = [var for var in vars if var not in keep_list]
do_not_keep
```

In [ ]:
```
def remove_variables(df, vars_to_remove):
    return df.drop(columns=[var for var in vars_to_remove if var in df.colu

grp_train_clean = remove_variables(grp_train, do_not_keep)
grp_test_clean = remove_variables(grp_test, do_not_keep)

Gini_vars_filtered = Gini_vars[Gini_vars[variable_name].isin(keep_list)]
```

In [ ]:
```
# Raport zmiennych
startrow = 2

# Zrzut DataFrame do arkusza Excel
writer = pd.ExcelWriter('/content/drive/My Drive/praca_magisterska_2/Variab
Gini_vars_filtered.to_excel(writer, sheet_name=variable_name, startrow=1, i

series = Gini_vars_filtered[variable_name]
max_len = max((series.astype(str).map(len).max(), len(str(series.name)))) +

book = writer.book
sheet = writer.sheets[variable_name]
format1 = book.add_format({'num_format': '0.0%'})
sheet.set_column('B:B', None, format1)
sheet.set_column('C:C', None, format1)
sheet.set_column('D:D', None, format1)
sheet.set_column(0, 0, max_len)
sheet.set_column(1, 1, 15)  # Stała szerokość kolumny zamiast len(gini_trai
sheet.set_column(2, 2, 15)  # Stała szerokość kolumny zamiast len(gini_test)
sheet.set_column(3, 3, 15)  # Stała szerokość kolumny zamiast len(delta_gini
sheet.set_column(4, 4, 15)  # Stała szerokość kolumny zamiast len(INV)
sheet.set_column(5, 5, 15)  # Stała szerokość kolumny zamiast len(PSI)
sheet.set_column(6, 6, 15)  # Stała szerokość kolumny zamiast len(PSI_tar)
sheet.set_column(7, 7, 15)  # Stała szerokość kolumny zamiast len(percent_m
sheet.set_column(8, 8, 15)  # Stała szerokość kolumny zamiast len(count_unic

sss2 = pd.DataFrame(grp_all_clean[time_report_name].value_counts())
ntimes = sss2.shape[0]

# Filtering Big_scorecard to include only features in keep_list
Big_scorecard_filtered = Big_scorecard[Big_scorecard[variable_name].isin(ke
```

```
for feature in keep_list:
    sssgr = pd.DataFrame(grp_all_clean.groupby([time_report_name]).agg({targ
    sssgr = pd.DataFrame(sssgr[target_name])
    sssgr = sssgr.reset_index()
    fin = pd.DataFrame()
    sss = pd.DataFrame()
    ddd = pd.DataFrame()
    sss = pd.DataFrame(grp_all_clean.groupby([feature, time_report_name]).ag
    sss = pd.DataFrame(sss[target_name])
    sss = sss.reset_index()
    sss = sss.rename(columns={"sum": event_name, "count": all_name, feature
    sss = pd.merge(sss, sssgr, on=time_report_name)
    sss[nonevent_name] = sss[all_name] - sss[event_name]
    sss[event_rate_name] = sss[event_name] / sss[all_name]
    sss[share_name] = sss[all_name] / sss["count"]
    sss = sss[[grp_name, time_report_name, event_name, all_name, nonevent_na
    sss = sss.sort_values(by=[grp_name, time_report_name])

    sub = Big_scorecard_filtered[Big_scorecard_filtered[variable_name] == fe
    sub = sub[[grp_name, condition_name, event_rate_name, share_name, all_na
    ncat = sub.shape[0]

    shname = feature[0:31]
    sub.to_excel(writer, sheet_name=shname, startrow=startrow, index_label=I
    sss.to_excel(writer, sheet_name=shname, startrow=startrow, index_label=I
    book = writer.book
    sheet = writer.sheets[shname]
    bold = book.add_format({'bold': True, 'size': 24})
    sheet.write('A1', variable_name + ': ' + feature, bold)
    bold = book.add_format({'bold': True, 'size': 12})
    boldp = book.add_format({'bold': True, 'size': 12, 'num_format': '0.0%'}
    sheet.write_formula('D' + str(startrow + ncat + 2), '=SUM(D4:D' + str(st
    sheet.write_formula('E' + str(startrow + ncat + 2), '=SUM(E4:E' + str(st
    sheet.write_formula('F' + str(startrow + ncat + 2), '=SUM(F4:F' + str(st
    sheet.write_formula('G' + str(startrow + ncat + 2), '=SUM(G4:G' + str(st

    format1 = book.add_format({'num_format': '0.0%'})
    sheet.set_column('C:C', None, format1)
    sheet.set_column('D:D', None, format1)
    sheet.set_column('N:N', None, format1)
    sheet.set_column('O:O', None, format1)

    series = sub[condition_name]
    max_len = max((series.astype(str).map(len).max(), len(str(series.name))
    sheet.set_column(1, 1, max_len)

    # Wykres
    chart = book.add_chart({'type': 'line'})
    chart.set_title({'name': event_rate_name})
    chart.set_x_axis({'name': '=' + shname + '!J3', 'num_font': {'rotation'
    for i in range(ncat):
        chart.add_series({'values': '=' + shname + '!N' + str(4 + i * ntimes
                          'categories': '=' + shname + '!J' + str(4) + ':J'
    chart.set_legend({'position': 'bottom'})
    sheet.insert_chart('A' + str(ncat + 6), chart)

    chart = book.add_chart({'type': 'line'})
    chart.set_title({'name': share_name})
    chart.set_x_axis({'name': '=' + shname + '!J3', 'num_font': {'rotation'
    for i in range(ncat):
        chart.add_series({'values': '=' + shname + '!O' + str(4 + i * ntimes
                          'categories': '=' + shname + '!J' + str(4) + ':J'
    chart.set_legend({'position': 'bottom'})
    sheet.insert_chart('A' + str(ncat + 6 + 18), chart)
```

```
writer.close()
```

In [ ]:
```
import os

output_dir = "/content/drive/My Drive/praca_magisterska_2/"
os.makedirs(output_dir, exist_ok=True)

dataframes = {
    "grp_train_4": grp_train,
    "grp_test_4": grp_test,
    "Big_scorecard": Big_scorecard
     "df_0_1": df
    "Gini_vars": Gini_vars
    "grp_train_clean": grp_train_clean,
    "grp_test_clean": grp_test_clean
    "Gini_vars_filtered": Gini_vars_filtered

}

for df_name, df in dataframes.items():
    output_path = os.path.join(output_dir, f"{df_name}.csv")
    df.to_csv(output_path, index=False)
    print(f"Exported {df_name} to {output_path}")

print("Tabele zapisane")
```

Rozdzial 3 - Analiza RFE

In [4]:
```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, roc_curve, roc_auc_score
from sklearn.utils import resample
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [5]:
```
grp_train_clean = pd.read_csv('grp_train_clean.csv')
grp_test_clean = pd.read_csv('grp_test_clean.csv')
```

In [6]:
```
# Załadowanie przetworzonych zbiorów danych
grp_train_clean = pd.read_csv('grp_train_clean.csv')
grp_test_clean = pd.read_csv('grp_test_clean.csv')

# Zdefiniowanie słownika mapowania
installment_mapping = {0: 3, 1: 2, 2: 1, 3: 0}

# Zastosowanie mapowania do kolumny 'installment'
grp_train_clean['installment'] = grp_train_clean['installment'].map(installr
grp_test_clean['installment'] = grp_test_clean['installment'].map(installmer

# Wyświetlenie pierwszych kilku wierszy zaktualizowanych zbiorów danych
print("Updated grp_train_clean:")
print(grp_train_clean.head())

print("\nUpdated grp_test_clean:")
print(grp_test_clean.head())
```

```
Updated grp_train_clean:
   funded_amnt_inv  installment  grade  emp_length  home_ownership  \
0                1            0      2           1               0
1                2            0      1           1               0
2                3            0      1           1               0
3                1            0      3           1               0
4                1            0      2           1               2

   annual_inc  title  addr_state  dti  delinq_2yrs  ...  total_bal_il  \
0           3      4           2    3            1  ...             0
1           0      1           2    2            3  ...             1
2           3      1           2    3            3  ...             0
3           1      2           2    1            3  ...             0
4           3      4           2    3            3  ...             0

   total_rev_hi_lim  mths_since_earliest_cr_line  \
0                 1                            2
1                 1                            2
2                 3                            3
3                 3                            2
4                 3                            1

   mths_since_last_credit_pull_d  target  term  Intercept  outstanding_bad
\
0                              2       0    60          1                0
1                              2       0    36          1                0
2                              2       0    60          1                0
3                              2       0    36          1                0
4                              2       0    60          1                0

   outstanding        id
0        16000  56230263
1         3975  67339041
2        28000  45092215
3        18000  40350770
4        21600  64008359

[5 rows x 36 columns]

Updated grp_test_clean:
   funded_amnt_inv  installment  grade  emp_length  home_ownership  \
0                3            0      1           3               0
1                1            0      3           1               0
2                2            0      3           1               0
3                2            0      2           0               2
4                2            0      3           0               0

   annual_inc  title  addr_state  dti  delinq_2yrs  ...  total_bal_il  \
0           3      4           4    2            2  ...             0
1           3      2           2    0            3  ...             0
2           2      2           2    1            2  ...             0
3           0      4           4    1            0  ...             0
4           1      4           4    3            3  ...             0

   total_rev_hi_lim  mths_since_earliest_cr_line  \
0                 3                            3
1                 2                            3
2                 2                            3
3                 1                            3
4                 1                            1

   mths_since_last_credit_pull_d  target  term  Intercept  outstanding_bad
\
0                              2       0    60          1                0
```

```
1                              2       0    36          1                0
2                              2       0    36          1                0
3                              2       0    36          1                0
4                              2       0    36          1                0

   outstanding        id
0        30000  54315817
1        12000  39469359
2         6300  40413161
3         8500  41121446
4         9000  52016839

[5 rows x 36 columns]
```

In [7]:
```
selected_features = ['funded_amnt_inv',
 'installment',
 'emp_length',
 'home_ownership',
 'annual_inc',
# 'purpose',
 'title',
 'addr_state',
 'dti',
 'delinq_2yrs',
 'inq_last_6mths',
# 'mths_since_last_delinq',
# 'mths_since_last_record',
# 'open_acc',
 'revol_bal',
 'revol_util',
 'total_acc',
 'initial_list_status',
 'total_pymnt',
 'total_rec_prncp',
 'total_rec_int',
 'last_pymnt_amnt',
 'mths_since_last_major_derog',
 'tot_coll_amt',
 'tot_cur_bal',
# 'il_util',
 'total_rev_hi_lim',
 'mths_since_earliest_cr_line',
# 'mths_since_issue_d',
# 'mths_since_last_pymnt_d',
 'mths_since_last_credit_pull_d']
```

In [8]:
```
X_train = grp_train_clean[selected_features]
y_train = grp_train_clean[target_name]
X_test = grp_test_clean[selected_features]
y_test = grp_test_clean[target_name]

# Undersampling 0
df_majority = grp_train_clean[grp_train_clean[target_name] == 0]
df_minority = grp_train_clean[grp_train_clean[target_name] == 1]

df_majority_downsampled = resample(df_majority,
                                   replace=False,
                                   n_samples=len(df_minority),  # aby liczba
                                   random_state=42)

df_downsampled = pd.concat([df_majority_downsampled, df_minority])

X_train_balanced = df_downsampled[selected_features]
y_train_balanced = df_downsampled[target_name]
```

In [102…

```
def calculate_lift(y_true, y_pred, top_percent=0.1):
    data = pd.DataFrame({'y_true': y_true, 'y_pred': y_pred})
    data = data.sort_values(by='y_pred', ascending=False)
    top_data = data.head(int(top_percent * len(data)))
    lift = top_data['y_true'].sum() / data['y_true'].mean()
    return lift

# Funkcja do obliczania różnych miar dla różnych liczby cech
def calculate_metrics(estimator, X_train, y_train, X_test, y_test, max_featu
    gini_scores_train = []
    gini_scores_test = []
    type_i_errors_train = []
    type_i_errors_test = []
    type_ii_errors_train = []
    type_ii_errors_test = []
    ks_scores_train = []
    ks_scores_test = []
    lift_scores_train = []
    lift_scores_test = []

    for n_selected_features in range(1, max_features + 1):
        sfs = RFE(estimator, n_features_to_select=n_selected_features)
        sfs.fit(X_train, y_train)
        selected_features = sfs.get_support()
        X_train_selected = X_train[:, selected_features]
        X_test_selected = X_test[:, selected_features]

        estimator.fit(X_train_selected, y_train)

        y_train_pred = estimator.predict_proba(X_train_selected)[:, 1]
        y_test_pred = estimator.predict_proba(X_test_selected)[:, 1]

        # GINI
        auc_train = roc_auc_score(y_train, y_train_pred)
        gini_train = 2 * auc_train - 1
        gini_scores_train.append(gini_train)

        auc_test = roc_auc_score(y_test, y_test_pred)
        gini_test = 2 * auc_test - 1
        gini_scores_test.append(gini_test)

        # KS
        fpr_train, tpr_train, _ = roc_curve(y_train, y_train_pred)
        ks_train = max(tpr_train - fpr_train)
        ks_scores_train.append(ks_train)

        fpr_test, tpr_test, _ = roc_curve(y_test, y_test_pred)
        ks_test = max(tpr_test - fpr_test)
        ks_scores_test.append(ks_test)

        # Error Type I i Error Type II
        y_train_pred_class = (y_train_pred >= 0.5).astype(int)
        y_test_pred_class = (y_test_pred >= 0.5).astype(int)

        tn_train, fp_train, fn_train, tp_train = confusion_matrix(y_train, y
        tn_test, fp_test, fn_test, tp_test = confusion_matrix(y_test, y_test

        type_i_error_train = fp_train / (fp_train + tn_train)
        type_ii_error_train = fn_train / (fn_train + tp_train)
        type_i_errors_train.append(type_i_error_train)
        type_ii_errors_train.append(type_ii_error_train)

        type_i_error_test = fp_test / (fp_test + tn_test)
```

```
        type_ii_error_test = fn_test / (fn_test + tp_test)
        type_i_errors_test.append(type_i_error_test)
        type_ii_errors_test.append(type_ii_error_test)

        # Lift
        lift_train = calculate_lift(y_train, y_train_pred)
        lift_test = calculate_lift(y_test, y_test_pred)
        lift_scores_train.append(lift_train)
        lift_scores_test.append(lift_test)

    return (gini_scores_train, gini_scores_test, type_i_errors_train, type_
            type_ii_errors_train, type_ii_errors_test, ks_scores_train, ks_
            lift_scores_train, lift_scores_test)
def calculate_lift(y_true, y_pred, top_percent=0.1):
    data = pd.DataFrame({'y_true': y_true, 'y_pred': y_pred})
    data = data.sort_values(by='y_pred', ascending=False)
    top_data = data.head(int(top_percent * len(data)))
    lift = top_data['y_true'].sum() / data['y_true'].mean()
    return lift
```

definicja funkcji

In [78]:

```
import matplotlib.pyplot as plt

# Funkcja do organizowania wyników w struktury danych
def organize_results(*results, labels):
    organized_results = {
        'gini': {'train': [], 'test': []},
        'type_i_error': {'train': [], 'test': []},
        'type_ii_error': {'train': [], 'test': []},
        'ks': {'train': [], 'test': []}
    }

    for result in results:
        (gini_scores_train, gini_scores_test, type_i_errors_train, type_i_e
         type_ii_errors_train, type_ii_errors_test, ks_scores_train, ks_sco

        organized_results['gini']['train'].append(gini_scores_train)
        organized_results['gini']['test'].append(gini_scores_test)
        organized_results['type_i_error']['train'].append(type_i_errors_tra
        organized_results['type_i_error']['test'].append(type_i_errors_test
        organized_results['type_ii_error']['train'].append(type_ii_errors_t
        organized_results['type_ii_error']['test'].append(type_ii_errors_te
        organized_results['ks']['train'].append(ks_scores_train)
        organized_results['ks']['test'].append(ks_scores_test)

    return organized_results

# Funkcja do generowania tabeli wyników
def generate_results_table(organized_results, labels, num_features):
    table_data = {
        'Model': labels,
        'GINI Score Train': [res[num_features-1] for res in organized_result
        'GINI Score Test': [res[num_features-1] for res in organized_results
        'Type I Error Train': [res[num_features-1] for res in organized_resu
        'Type I Error Test': [res[num_features-1] for res in organized_resul
        'Type II Error Train': [res[num_features-1] for res in organized_res
        'Type II Error Test': [res[num_features-1] for res in organized_resu
        'KS Score Train': [res[num_features-1] for res in organized_results
        'KS Score Test': [res[num_features-1] for res in organized_results[
    }
    results_df = pd.DataFrame(table_data)
    return results_df
```

```
# Funkcja do tworzenia wykresów
def plot_comparison(metric, organized_results, labels, title, dataset='test
    plt.figure(figsize=(14, 8))
    for i, label in enumerate(labels):
        plt.plot(range(1, max_features + 1), organized_results[metric][datas
    plt.title(f'{title} vs Number of Features ({dataset.capitalize()})')
    plt.xlabel('Number of Features')
    plt.ylabel(title)
    plt.legend()
    plt.grid()
    plt.show()
```

Logistic Regression

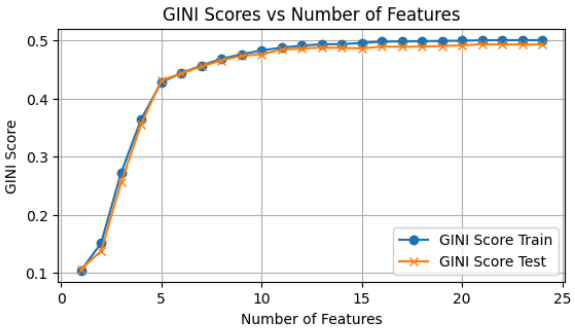
In [96]:

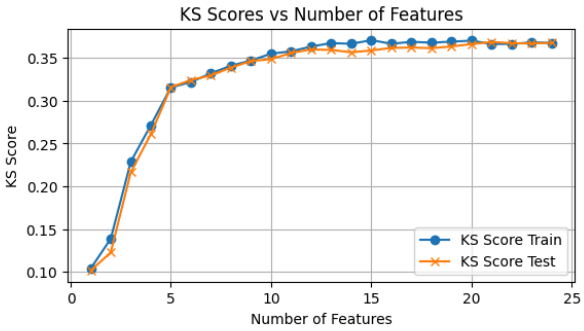
```
# Obliczanie miar dla różnych liczby cech na zbiorze treningowym i testowym
model = LogisticRegression()
max_features = X_train_balanced.shape[1]
(gini_scores_train, gini_scores_test, type_i_errors_train, type_i_errors_tes
 type_ii_errors_train, type_ii_errors_test, ks_scores_train, ks_scores_test,
 lift_scores_train, lift_scores_test) = calculate_metrics(model, X_train_ba


# Wyświetlanie wyników w notebooku
print("GINI, Error Type I, Error Type II, KS and Lift Scores by Number of Fe
display(results_df)

# Wizualizacja wyników
plt.figure(figsize=(14, 8))
plt.subplot(2, 2, 1)
plt.plot(range(1, max_features + 1), gini_scores_train, marker='o', label='(
plt.plot(range(1, max_features + 1), gini_scores_test, marker='x', label='Gi
plt.title('GINI Scores vs Number of Features')
plt.xlabel('Number of Features')
plt.ylabel('GINI Score')
plt.legend()
plt.grid()

plt.subplot(2, 2, 2)
plt.plot(range(1, max_features + 1), ks_scores_train, marker='o', label='KS
plt.plot(range(1, max_features + 1), ks_scores_test, marker='x', label='KS S
plt.title('KS Scores vs Number of Features')
plt.xlabel('Number of Features')
plt.ylabel('KS Score')
plt.legend()
plt.grid()

plt.subplot(2, 2, 3)
plt.plot(range(1, max_features + 1), type_i_errors_train, marker='o', label=
plt.plot(range(1, max_features + 1), type_i_errors_test, marker='x', label=
plt.title('Type I Error vs Number of Features')
plt.xlabel('Number of Features')
plt.ylabel('Error Rate')
plt.legend()
plt.grid()

plt.subplot(2, 2, 4)
plt.plot(range(1, max_features + 1), type_ii_errors_train, marker='o', label
plt.plot(range(1, max_features + 1), type_ii_errors_test, marker='x', label=
plt.title('Type II Error vs Number of Features')
```

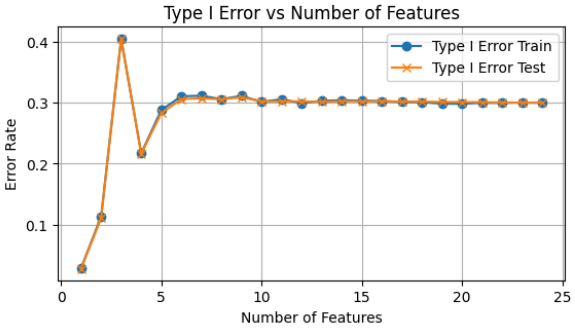
```
plt.xlabel('Number of Features')
plt.ylabel('Error Rate')
plt.legend()
plt.grid()

plt.subplots_adjust(hspace=0.5)
plt.show()

# Osobny wykres dla Lift Score
plt.figure(figsize=(7, 6))
plt.plot(range(1, max_features + 1), lift_scores_train, marker='o', label='L
plt.plot(range(1, max_features + 1), lift_scores_test, marker='x', label='Li
plt.title('Lift Scores vs Number of Features')
plt.xlabel('Number of Features')
plt.ylabel('Lift Score')
plt.legend()
plt.grid()
plt.show()
```

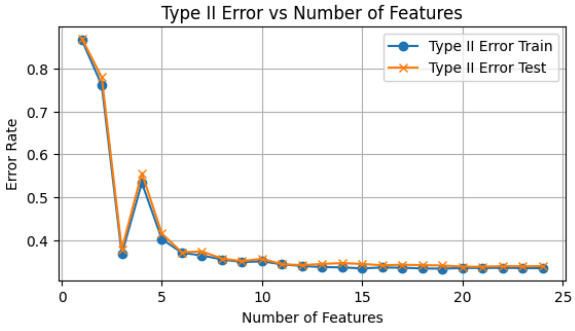
GINI, Error Type I, Error Type II, KS and Lift Scores by Number of Features

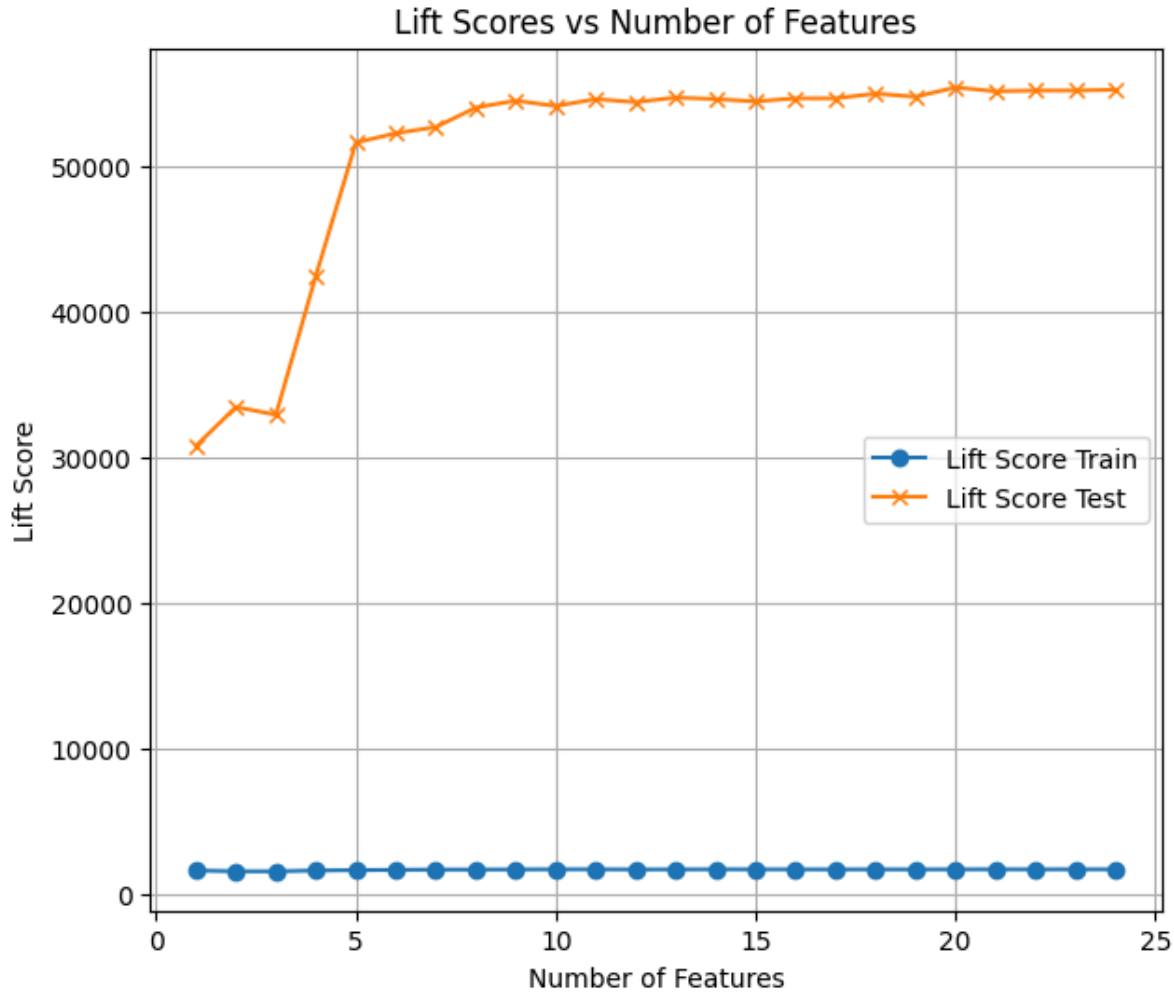
	Number of Features	GINI Score Train	GINI Score Test	Type I Error Train	Type I Error Test	Type II Error Train	Type II Error Test	KS Score Train	KS
0	1	0.103664	0.105961	0.028883	0.028147	0.867351	0.870423	0.103766	0.1
1	2	0.151111	0.137429	0.113607	0.110942	0.761232	0.779577	0.138639	0.1
2	3	0.272285	0.255960	0.403937	0.405219	0.367351	0.378521	0.228712	0.2
3	4	0.364328	0.355552	0.216945	0.216324	0.533590	0.556338	0.271502	0.2
4	5	0.427818	0.430582	0.287762	0.282655	0.402225	0.415141	0.315576	0.3
5	6	0.443629	0.443386	0.310227	0.305518	0.370347	0.371479	0.321994	0.32
6	7	0.456593	0.455063	0.311297	0.307185	0.364142	0.373592	0.332264	0.32
7	8	0.468264	0.465502	0.305520	0.305209	0.354728	0.356690	0.341036	0.33
8	9	0.475884	0.473315	0.311083	0.308832	0.348738	0.351056	0.347240	0.34
9	10	0.482739	0.475999	0.301027	0.301941	0.350449	0.356338	0.355156	0.34
10	11	0.487422	0.483954	0.305520	0.302177	0.343389	0.343662	0.357938	0.35
11	12	0.490918	0.486023	0.299101	0.302009	0.339538	0.341549	0.363500	0.3
12	13	0.493242	0.487524	0.302953	0.301229	0.337184	0.344366	0.367565	0.3
13	14	0.493433	0.487028	0.303808	0.302291	0.336329	0.346479	0.366709	0.3
14	15	0.495564	0.486123	0.303166	0.302029	0.333975	0.344366	0.370988	0.3
15	16	0.497753	0.489307	0.302525	0.302352	0.336115	0.341197	0.366709	0.3
16	17	0.498244	0.488816	0.301027	0.301619	0.335473	0.342606	0.369063	0.3
17	18	0.498571	0.489208	0.300599	0.301599	0.333975	0.341549	0.367993	0.3
18	19	0.498864	0.490376	0.298246	0.300987	0.333333	0.340845	0.369277	0.3
19	20	0.499460	0.491408	0.298032	0.300859	0.335045	0.338028	0.370347	0.3
20	21	0.500131	0.493325	0.299529	0.300563	0.334831	0.337676	0.366496	0.3
21	22	0.500329	0.492943	0.299529	0.300214	0.335259	0.339437	0.366496	0.3
22	23	0.500354	0.492860	0.299957	0.300012	0.334617	0.338732	0.368421	0.3
23	24	0.500350	0.492831	0.299529	0.300052	0.334403	0.339437	0.367779	0.3











Definiowanie funkcji

In [20]:
```
def calculate_scores(estimator, X_train, y_train, X_test, y_test, max_featu
    gini_scores_train = []
    gini_scores_test = []
    type_i_errors_train = []
    type_i_errors_test = []
```

```
    type_ii_errors_train = []
    type_ii_errors_test = []
    ks_scores_train = []
    ks_scores_test = []

    for n_selected_features in range(1, max_features + 1):
        sfs = RFE(estimator, n_features_to_select=n_selected_features)
        sfs.fit(X_train, y_train)
        selected_features = sfs.get_support()
        X_train_selected = X_train[:, selected_features]
        X_test_selected = X_test[:, selected_features]

        estimator.fit(X_train_selected, y_train)

        y_train_pred = estimator.predict_proba(X_train_selected)[:, 1]
        y_test_pred = estimator.predict_proba(X_test_selected)[:, 1]

        # GINI
        auc_train = roc_auc_score(y_train, y_train_pred)
        gini_train = 2 * auc_train - 1
        gini_scores_train.append(gini_train)

        auc_test = roc_auc_score(y_test, y_test_pred)
        gini_test = 2 * auc_test - 1
        gini_scores_test.append(gini_test)

        # Error Type I i Error Type II
        y_train_pred_class = (y_train_pred >= 0.5).astype(int)
        y_test_pred_class = (y_test_pred >= 0.5).astype(int)

        tn_train, fp_train, fn_train, tp_train = confusion_matrix(y_train, y
        tn_test, fp_test, fn_test, tp_test = confusion_matrix(y_test, y_test

        type_i_error_train = fp_train / (fp_train + tn_train)
        type_ii_error_train = fn_train / (fn_train + tp_train)
        type_i_errors_train.append(type_i_error_train)
        type_ii_errors_train.append(type_ii_error_train)

        type_i_error_test = fp_test / (fp_test + tn_test)
        type_ii_error_test = fn_test / (fn_test + tp_test)
        type_i_errors_test.append(type_i_error_test)
        type_ii_errors_test.append(type_ii_error_test)

        # KS
        fpr_train, tpr_train, _ = roc_curve(y_train, y_train_pred)
        ks_train = max(tpr_train - fpr_train)
        ks_scores_train.append(ks_train)

        fpr_test, tpr_test, _ = roc_curve(y_test, y_test_pred)
        ks_test = max(tpr_test - fpr_test)
        ks_scores_test.append(ks_test)

    return (gini_scores_train, gini_scores_test, type_i_errors_train, type_
            type_ii_errors_train, type_ii_errors_test, ks_scores_train, ks_s
```

definiowanie funkcji RFE

In [ ]:
```
from sklearn.linear_model import Ridge, LogisticRegression
from sklearn.metrics import roc_auc_score, confusion_matrix, roc_curve
from sklearn.feature_selection import RFE
import matplotlib.pyplot as plt
```

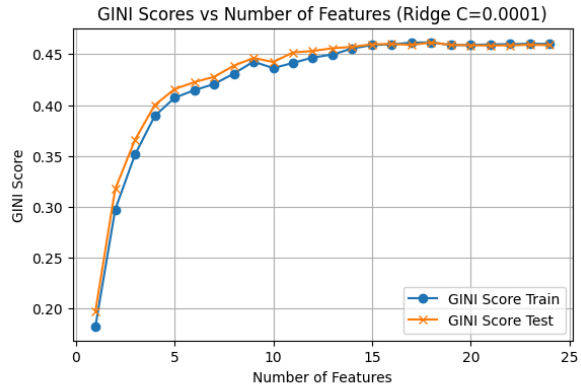
```
import pandas as pd
import numpy as np

def calculate_scores(estimator, X_train, y_train, X_test, y_test, max_featu
    gini_scores_train = []
    gini_scores_test = []
    type_i_errors_train = []
    type_i_errors_test = []
    type_ii_errors_train = []
    type_ii_errors_test = []
    ks_scores_train = []
    ks_scores_test = []

    for n_selected_features in range(1, max_features + 1):
        sfs = RFE(estimator, n_features_to_select=n_selected_features)
        sfs.fit(X_train, y_train)
        selected_features = sfs.get_support()
        X_train_selected = X_train[:, selected_features]
        X_test_selected = X_test[:, selected_features]

        estimator.fit(X_train_selected, y_train)

        if hasattr(estimator, "predict_proba"):
            y_train_pred = estimator.predict_proba(X_train_selected)[:, 1]
            y_test_pred = estimator.predict_proba(X_test_selected)[:, 1]
        else:
            y_train_pred = estimator.predict(X_train_selected)
            y_test_pred = estimator.predict(X_test_selected)

        # GINI
        auc_train = roc_auc_score(y_train, y_train_pred)
        gini_train = 2 * auc_train - 1
        gini_scores_train.append(gini_train)

        auc_test = roc_auc_score(y_test, y_test_pred)
        gini_test = 2 * auc_test - 1
        gini_scores_test.append(gini_test)

        # KS
        fpr_train, tpr_train, _ = roc_curve(y_train, y_train_pred)
        ks_train = max(tpr_train - fpr_train)
        ks_scores_train.append(ks_train)

        fpr_test, tpr_test, _ = roc_curve(y_test, y_test_pred)
        ks_test = max(tpr_test - fpr_test)
        ks_scores_test.append(ks_test)

        # Error Type I i Error Type II
        y_train_pred_class = (y_train_pred >= 0.5).astype(int)
        y_test_pred_class = (y_test_pred >= 0.5).astype(int)

        tn_train, fp_train, fn_train, tp_train = confusion_matrix(y_train, y
        tn_test, fp_test, fn_test, tp_test = confusion_matrix(y_test, y_test

        type_i_error_train = fp_train / (fp_train + tn_train)
        type_ii_error_train = fn_train / (fn_train + tp_train)
        type_i_errors_train.append(type_i_error_train)
        type_ii_errors_train.append(type_ii_error_train)

        type_i_error_test = fp_test / (fp_test + tn_test)
        type_ii_error_test = fn_test / (fn_test + tp_test)
        type_i_errors_test.append(type_i_error_test)
        type_ii_errors_test.append(type_ii_error_test)
```

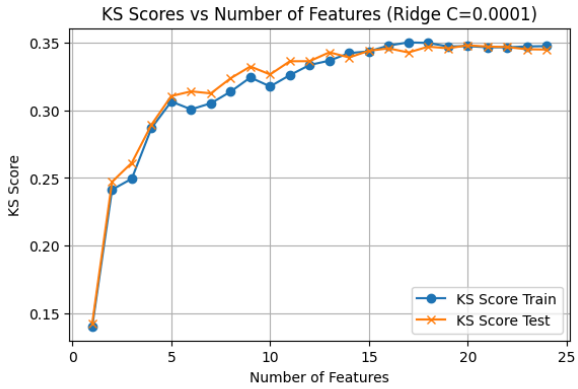
```
    return (gini_scores_train, gini_scores_test, type_i_errors_train, type_
            type_ii_errors_train, type_ii_errors_test, ks_scores_train, ks_

def plot_results(results, model_name, max_features):
    gini_scores_train, gini_scores_test, type_i_errors_train, type_i_errors_

    results_df = pd.DataFrame({
        'Number of Features': range(1, max_features + 1),
        'GINI Score Train': gini_scores_train,
        'GINI Score Test': gini_scores_test,
        'Type I Error Train': type_i_errors_train,
        'Type I Error Test': type_i_errors_test,
        'Type II Error Train': type_ii_errors_train,
        'Type II Error Test': type_ii_errors_test,
        'KS Score Train': ks_scores_train,
        'KS Score Test': ks_scores_test,
    })

    display(results_df)

    plt.figure(figsize=(14, 10))
    plt.subplot(2, 2, 1)
    plt.plot(range(1, max_features + 1), gini_scores_train, marker='o', labe
    plt.plot(range(1, max_features + 1), gini_scores_test, marker='x', labe
    plt.title(f'GINI Scores vs Number of Features ({model_name})')
    plt.xlabel('Number of Features')
    plt.ylabel('GINI Score')
    plt.legend()
    plt.grid()

    plt.subplot(2, 2, 2)
    plt.plot(range(1, max_features + 1), ks_scores_train, marker='o', label=
    plt.plot(range(1, max_features + 1), ks_scores_test, marker='x', label=
    plt.title(f'KS Scores vs Number of Features ({model_name})')
    plt.xlabel('Number of Features')
    plt.ylabel('KS Score')
    plt.legend()
    plt.grid()

    plt.subplot(2, 2, 3)
    plt.plot(range(1, max_features + 1), type_i_errors_train, marker='o', la
    plt.plot(range(1, max_features + 1), type_i_errors_test, marker='x', lab
    plt.title(f'Type I Error vs Number of Features ({model_name})')
    plt.xlabel('Number of Features')
    plt.ylabel('Error Rate')
    plt.legend()
    plt.grid()

    plt.subplot(2, 2, 4)
    plt.plot(range(1, max_features + 1), type_ii_errors_train, marker='o',
    plt.plot(range(1, max_features + 1), type_ii_errors_test, marker='x', la
    plt.title(f'Type II Error vs Number of Features ({model_name})')
    plt.xlabel('Number of Features')
    plt.ylabel('Error Rate')
    plt.legend()
    plt.grid()

    plt.subplots_adjust(hspace=0.5)  # Zwiększenie odstępów między wierszam
    plt.show()

# Liczba wybranych cech
max_features = X_train_balanced.shape[1]
```

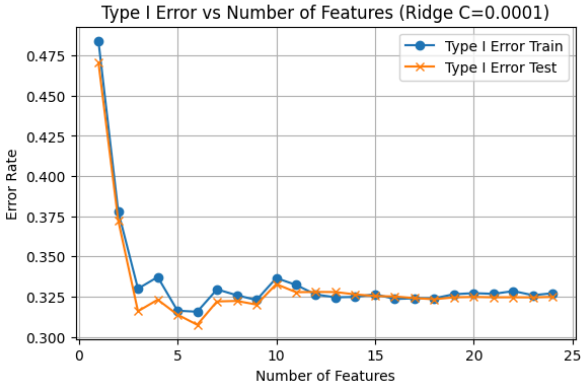
estymator ridge

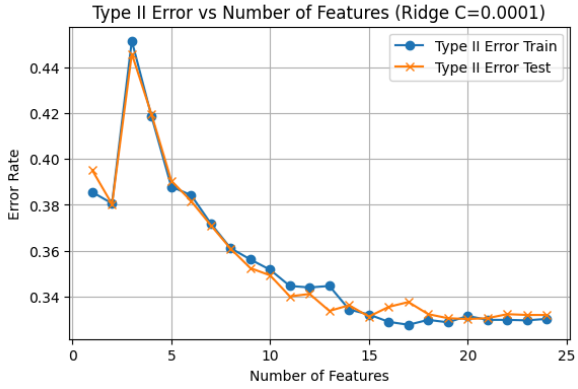
In [93]:
```
models_ridge = [
    ('Ridge C=0.0001', Ridge(alpha=10000)),
    ('Ridge C=10000', Ridge(alpha=0.0001)),
    ('Logistic Regression', LogisticRegression())
]

# Ewaluacja i wizualizacja wyników dla różnych modeli Ridge
results_ridge_c00001 = calculate_scores(models_ridge[0][1], X_train_balanced
results_ridge_c10000 = calculate_scores(models_ridge[1][1], X_train_balanced
results_logistic_regression = calculate_scores(models_ridge[2][1], X_train_b

for name, model in models_ridge:
    results = calculate_scores(model, X_train_balanced.values, y_train_balan
    plot_results(results, name, max_features)

# Organizowanie wyników dla wszystkich modeli
labels = [
    'Ridge C=0.0001', 'Ridge C=10000', 'Logistic Regression'
]

all_results = [
    results_ridge_c00001, results_ridge_c10000, results_logistic_regression
]

organized_results = organize_results(*all_results, labels=labels)

# Tworzenie wykresów dla każdej metryki dla zbioru testowego
plot_comparison('gini', organized_results, labels, 'GINI Score', dataset='te
plot_comparison('type_i_error', organized_results, labels, 'Type I Error Rat
plot_comparison('type_ii_error', organized_results, labels, 'Type II Error R
plot_comparison('ks', organized_results, labels, 'KS Score', dataset='test')

# Tworzenie wykresów dla każdej metryki dla zbioru treningowego
plot_comparison('gini', organized_results, labels, 'GINI Score', dataset='tr
plot_comparison('type_i_error', organized_results, labels, 'Type I Error Rat
plot_comparison('type_ii_error', organized_results, labels, 'Type II Error R
plot_comparison('ks', organized_results, labels, 'KS Score', dataset='train
```

	Number of Features	GINI Score Train	GINI Score Test	Type I Error Train	Type I Error Test	Type II Error Train	Type II Error Test	KS Score Train	KS S
0	1	0.182726	0.197628	0.484168	0.470710	0.385537	0.395070	0.140565	0.14
1	2	0.297346	0.317435	0.378049	0.372514	0.380616	0.380282	0.241335	0.24
2	3	0.351256	0.365817	0.329910	0.315864	0.451433	0.445775	0.249465	0.2
3	4	0.389151	0.399926	0.337184	0.323064	0.418485	0.419366	0.286906	0.28
4	5	0.407067	0.415612	0.316217	0.313780	0.387890	0.390493	0.306376	0.3
5	6	0.414548	0.422430	0.315576	0.307447	0.384253	0.381690	0.300599	0.31
6	7	0.420600	0.427685	0.329482	0.321975	0.371844	0.371127	0.305092	0.31
7	8	0.430807	0.438538	0.325845	0.322290	0.361147	0.360915	0.313650	0.32
8	9	0.442661	0.446269	0.322850	0.320018	0.356226	0.352465	0.324134	0.33
9	10	0.436245	0.442273	0.336543	0.332737	0.351733	0.349296	0.317501	0.32
10	11	0.441403	0.451662	0.332264	0.327736	0.344673	0.340141	0.325845	0.3
11	12	0.446674	0.453099	0.326273	0.327890	0.344031	0.341197	0.333333	0.33
12	13	0.449562	0.455759	0.324561	0.327816	0.344673	0.333803	0.336543	0.34
13	14	0.455623	0.457132	0.324775	0.326337	0.334403	0.336268	0.342105	0.33
14	15	0.458832	0.459681	0.326487	0.325369	0.332264	0.331338	0.343389	0.34
15	16	0.459612	0.459949	0.323492	0.325067	0.329054	0.335563	0.347668	0.34
16	17	0.461275	0.459031	0.323920	0.324032	0.327771	0.337676	0.350021	0.34
17	18	0.461542	0.461492	0.323920	0.323339	0.329910	0.332394	0.349593	0.34
18	19	0.458826	0.459044	0.326487	0.324529	0.328840	0.330634	0.347026	0.34
19	20	0.459075	0.458625	0.327129	0.324771	0.331622	0.330282	0.347240	0.34
20	21	0.459528	0.458372	0.326701	0.324549	0.329910	0.330634	0.346384	0.34
21	22	0.459838	0.458661	0.328412	0.324569	0.329910	0.332394	0.346384	0.34
22	23	0.460147	0.459194	0.325845	0.324502	0.329696	0.332042	0.347026	0.34
23	24	0.460049	0.458822	0.327129	0.324805	0.330338	0.332042	0.347240	0.34

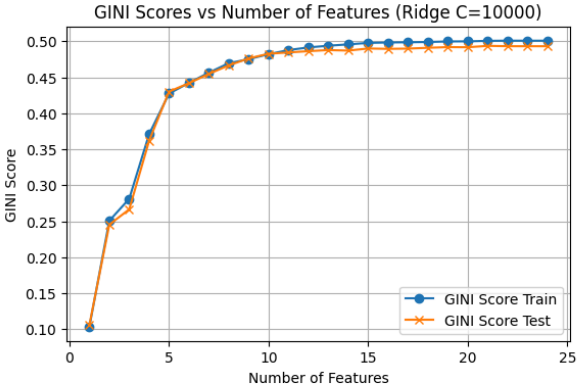

GINI Scores vs Number of Features (Ridge C=0.0001)
GINI Score
Number of Features
GINI Score Train
GINI Score Test

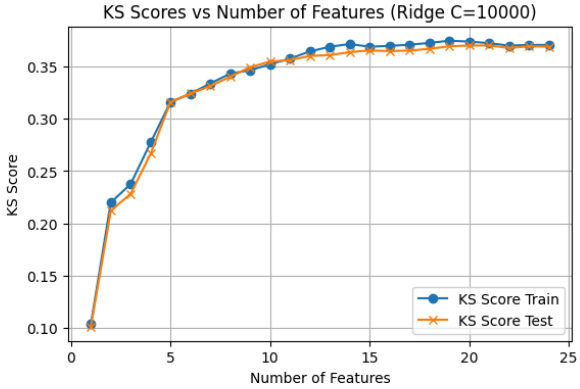

KS Scores vs Number of Features (Ridge C=0.0001)
KS Score
Number of Features
KS Score Train
KS Score Test

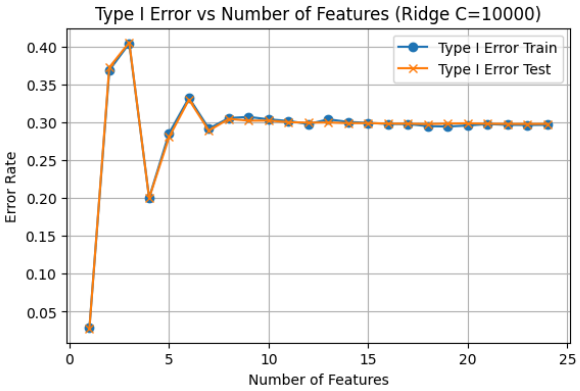

Type I Error vs Number of Features (Ridge C=0.0001)
Error Rate
Number of Features
Type I Error Train
Type I Error Test

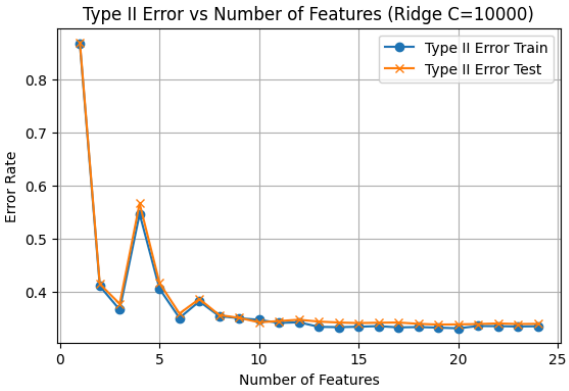

Type II Error vs Number of Features (Ridge C=0.0001)
Error Rate
Number of Features
Type II Error Train
Type II Error Test

	Number of Features	GINI Score Train	GINI Score Test	Type I Error Train	Type I Error Test	Type II Error Train	Type II Error Test	KS Score Train	KS
0	1	0.103664	0.105961	0.028883	0.028147	0.867351	0.870423	0.103766	0.1
1	2	0.250711	0.245400	0.368635	0.372380	0.411425	0.415493	0.219940	0.2
2	3	0.281289	0.266578	0.403937	0.405300	0.367137	0.378169	0.237698	0.2
3	4	0.371106	0.361942	0.200257	0.199867	0.547497	0.567254	0.277279	0.2
4	5	0.427525	0.430218	0.285409	0.280376	0.406504	0.417254	0.316003	0.3
5	6	0.442254	0.441917	0.332264	0.329793	0.351519	0.359859	0.323920	0.3
6	7	0.456594	0.454826	0.291613	0.288947	0.382756	0.387324	0.333761	0.3
7	8	0.469617	0.466658	0.305520	0.304469	0.354942	0.357042	0.343175	0.34
8	9	0.475307	0.476311	0.307018	0.302062	0.351091	0.351761	0.346384	0.34
9	10	0.482176	0.482806	0.303808	0.302224	0.348096	0.343310	0.352161	0.35
10	11	0.487943	0.484599	0.301669	0.300032	0.342105	0.345775	0.357510	0.35
11	12	0.491583	0.486378	0.297604	0.299461	0.343389	0.348239	0.364356	0.36
12	13	0.493951	0.487916	0.303808	0.299676	0.334831	0.344718	0.368849	0.36
13	14	0.495803	0.487179	0.300385	0.298802	0.334189	0.342958	0.371416	0.3
14	15	0.497849	0.490116	0.299101	0.298869	0.335045	0.341901	0.368849	0.36
15	16	0.498391	0.489585	0.297818	0.298210	0.336115	0.342606	0.369705	0.3
16	17	0.498740	0.489953	0.297604	0.298143	0.333761	0.342958	0.370774	0.3
17	18	0.499088	0.491054	0.294822	0.297538	0.334403	0.340493	0.372486	0.36
18	19	0.499850	0.492032	0.294395	0.298258	0.333333	0.339085	0.374626	0.36
19	20	0.500008	0.491853	0.295678	0.298587	0.332050	0.339437	0.373770	0.36
20	21	0.500719	0.493730	0.297604	0.298009	0.336329	0.340141	0.372272	0.3
21	22	0.500943	0.493305	0.296534	0.297807	0.336115	0.341197	0.369919	0.3
22	23	0.500905	0.493261	0.296320	0.297726	0.335473	0.340141	0.370561	0.3
23	24	0.500901	0.493240	0.296534	0.297827	0.335901	0.340493	0.370561	0.36

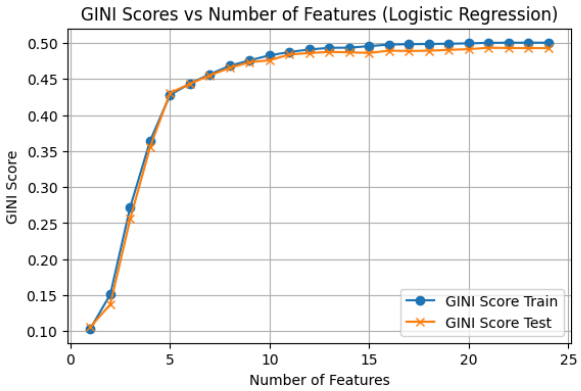

GINI Scores vs Number of Features (Ridge C=10000)
GINI Score
Number of Features
GINI Score Train
GINI Score Test

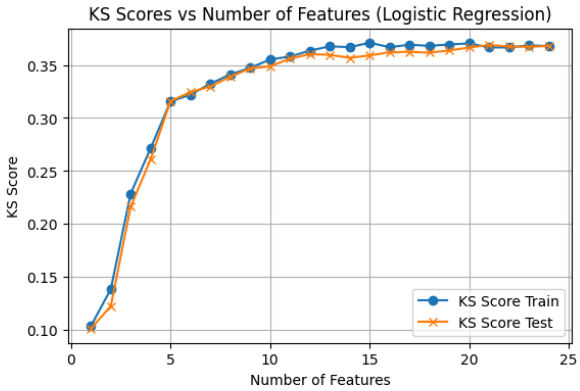

KS Scores vs Number of Features (Ridge C=10000)
KS Score
Number of Features
KS Score Train
KS Score Test

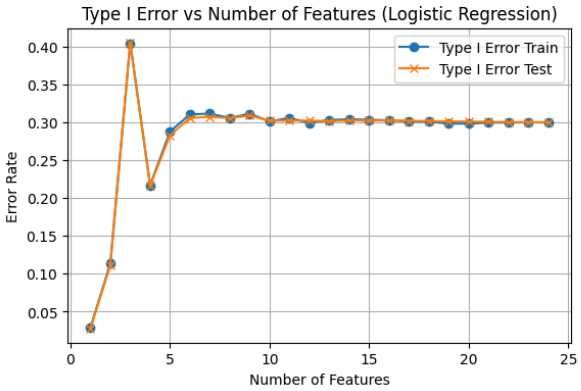

Type I Error vs Number of Features (Ridge C=10000)
Error Rate
Number of Features
Type I Error Train
Type I Error Test

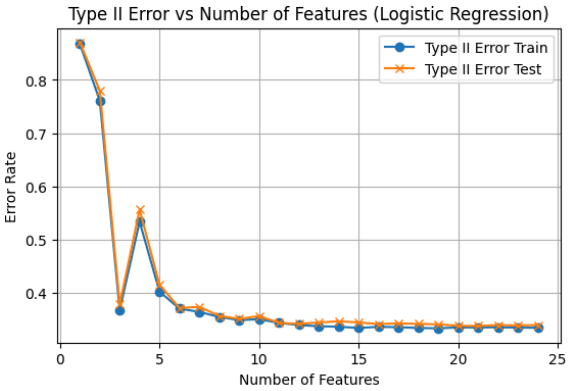

Type II Error vs Number of Features (Ridge C=10000)
Error Rate
Number of Features
Type II Error Train
Type II Error Test

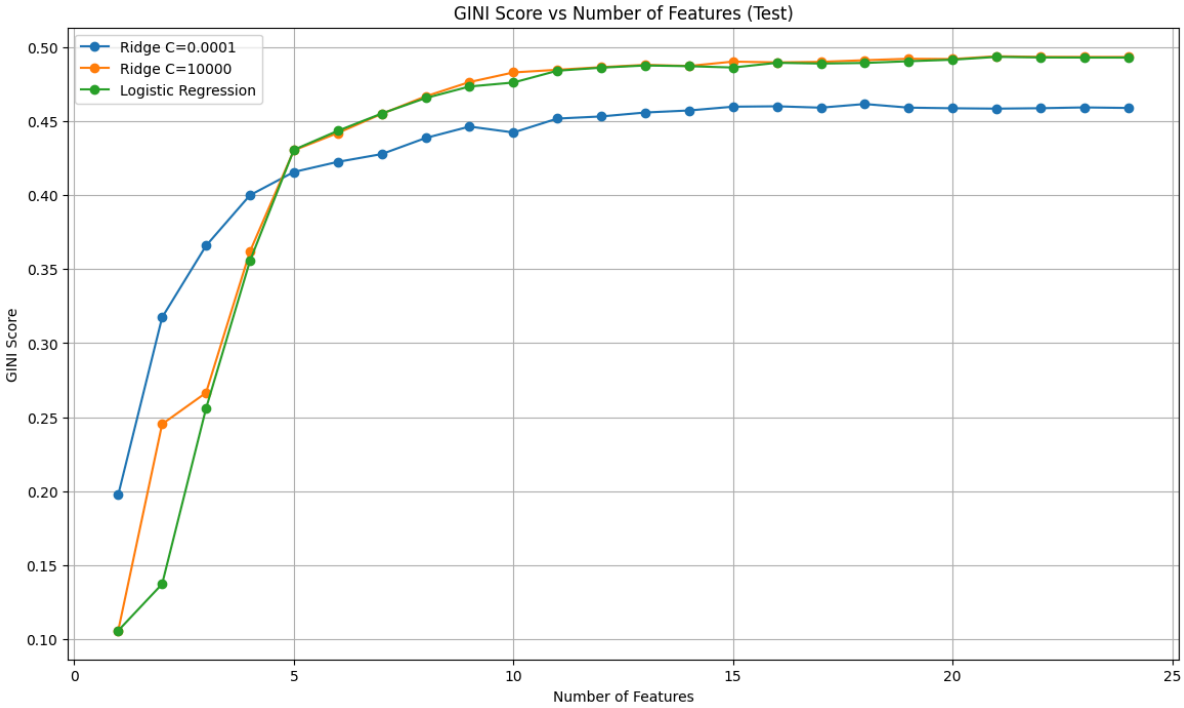
	Number of Features	GINI Score Train	GINI Score Test	Type I Error Train	Type I Error Test	Type II Error Train	Type II Error Test	KS Score Train	KS
0	1	0.103664	0.105961	0.028883	0.028147	0.867351	0.870423	0.103766	0.1
1	2	0.151111	0.137429	0.113607	0.110942	0.761232	0.779577	0.138639	0.1
2	3	0.272285	0.255960	0.403937	0.405219	0.367351	0.378521	0.228712	0.2
3	4	0.364328	0.355552	0.216945	0.216324	0.533590	0.556338	0.271502	0.2
4	5	0.427818	0.430582	0.287762	0.282655	0.402225	0.415141	0.315576	0.3
5	6	0.443629	0.443386	0.310227	0.305518	0.370347	0.371479	0.321994	0.32
6	7	0.456593	0.455063	0.311297	0.307185	0.364142	0.373592	0.332264	0.32
7	8	0.468264	0.465502	0.305520	0.305209	0.354728	0.356690	0.341036	0.33
8	9	0.475884	0.473315	0.311083	0.308832	0.348738	0.351056	0.347240	0.34
9	10	0.482739	0.475999	0.301027	0.301941	0.350449	0.356338	0.355156	0.34
10	11	0.487422	0.483954	0.305520	0.302177	0.343389	0.343662	0.357938	0.35
11	12	0.490918	0.486023	0.299101	0.302009	0.339538	0.341549	0.363500	0.3
12	13	0.493242	0.487524	0.302953	0.301229	0.337184	0.344366	0.367565	0.3
13	14	0.493433	0.487028	0.303808	0.302291	0.336329	0.346479	0.366709	0.35
14	15	0.495564	0.486123	0.303166	0.302029	0.333975	0.344366	0.370988	0.35
15	16	0.497753	0.489307	0.302525	0.302352	0.336115	0.341197	0.366709	0.3
16	17	0.498244	0.488816	0.301027	0.301619	0.335473	0.342606	0.369063	0.3
17	18	0.498571	0.489208	0.300599	0.301599	0.333975	0.341549	0.367993	0.3
18	19	0.498864	0.490376	0.298246	0.300987	0.333333	0.340845	0.369277	0.36
19	20	0.499460	0.491408	0.298032	0.300859	0.335045	0.338028	0.370347	0.36
20	21	0.500131	0.493325	0.299529	0.300563	0.334831	0.337676	0.366496	0.36
21	22	0.500329	0.492943	0.299529	0.300214	0.335259	0.339437	0.366496	0.36
22	23	0.500354	0.492860	0.299957	0.300012	0.334617	0.338732	0.368421	0.36
23	24	0.500350	0.492831	0.299529	0.300052	0.334403	0.339437	0.367779	0.36

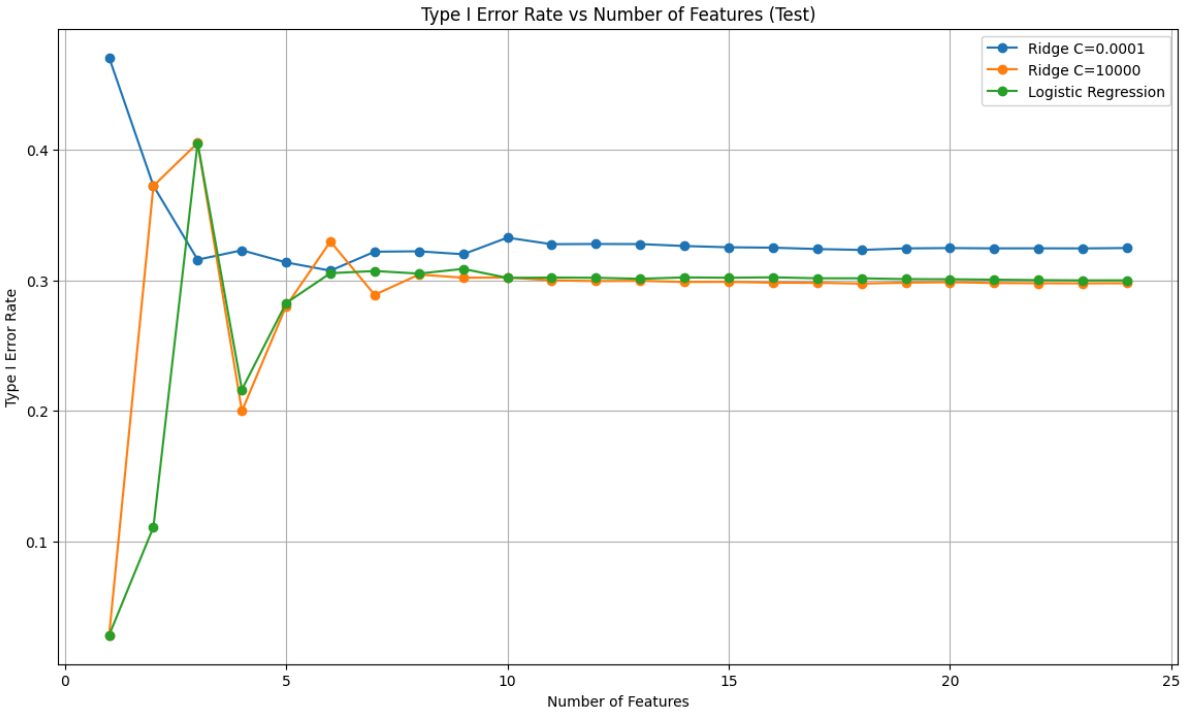

GINI Scores vs Number of Features (Logistic Regression)
GINI Score
Number of Features
GINI Score Train
GINI Score Test

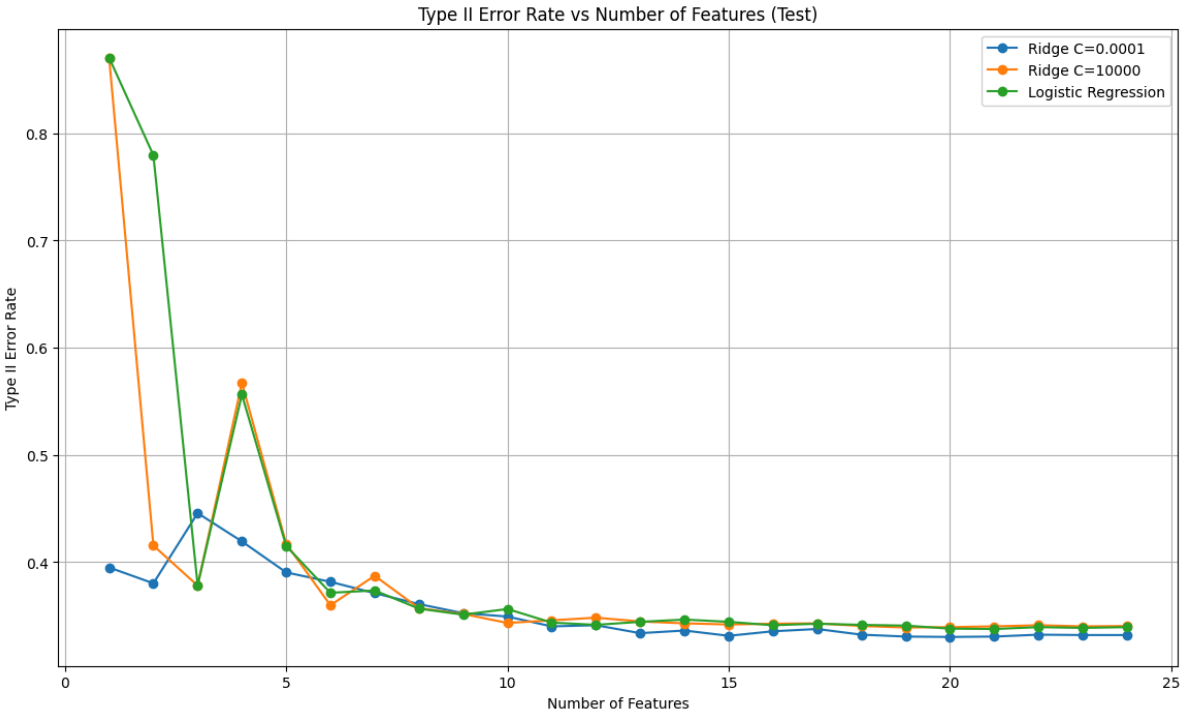

KS Scores vs Number of Features (Logistic Regression)
KS Score
Number of Features
KS Score Train
KS Score Test

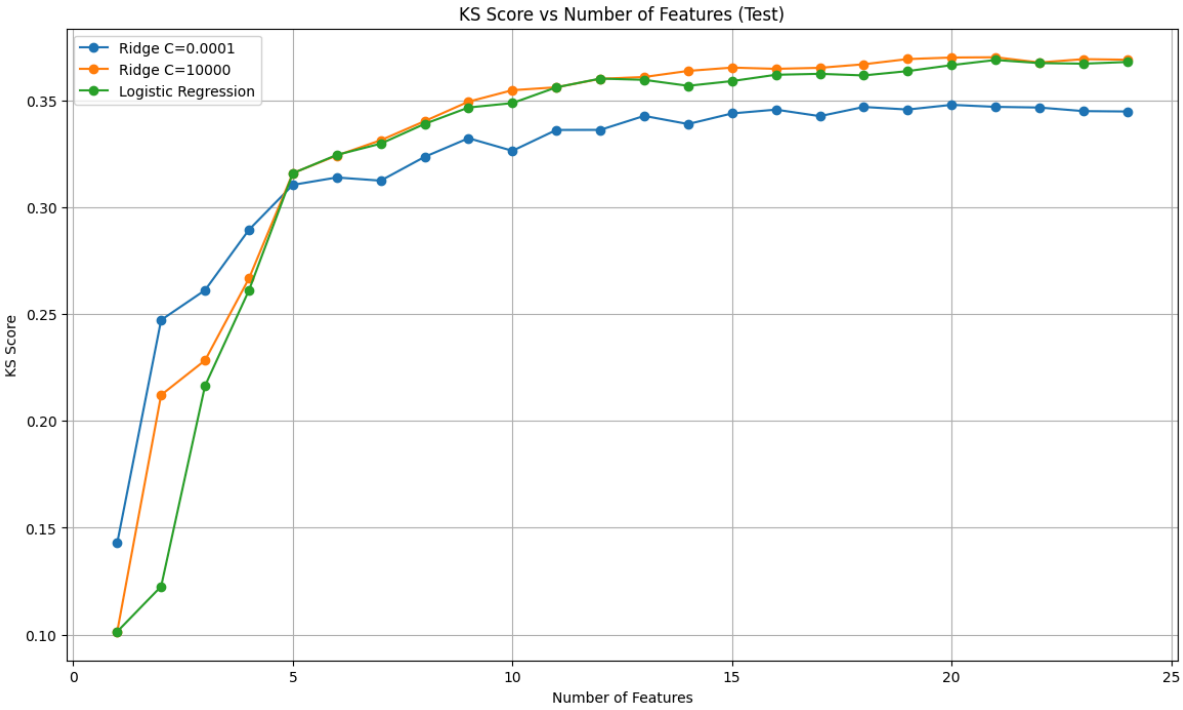

Type I Error vs Number of Features (Logistic Regression)
Error Rate
Number of Features
Type I Error Train
Type I Error Test

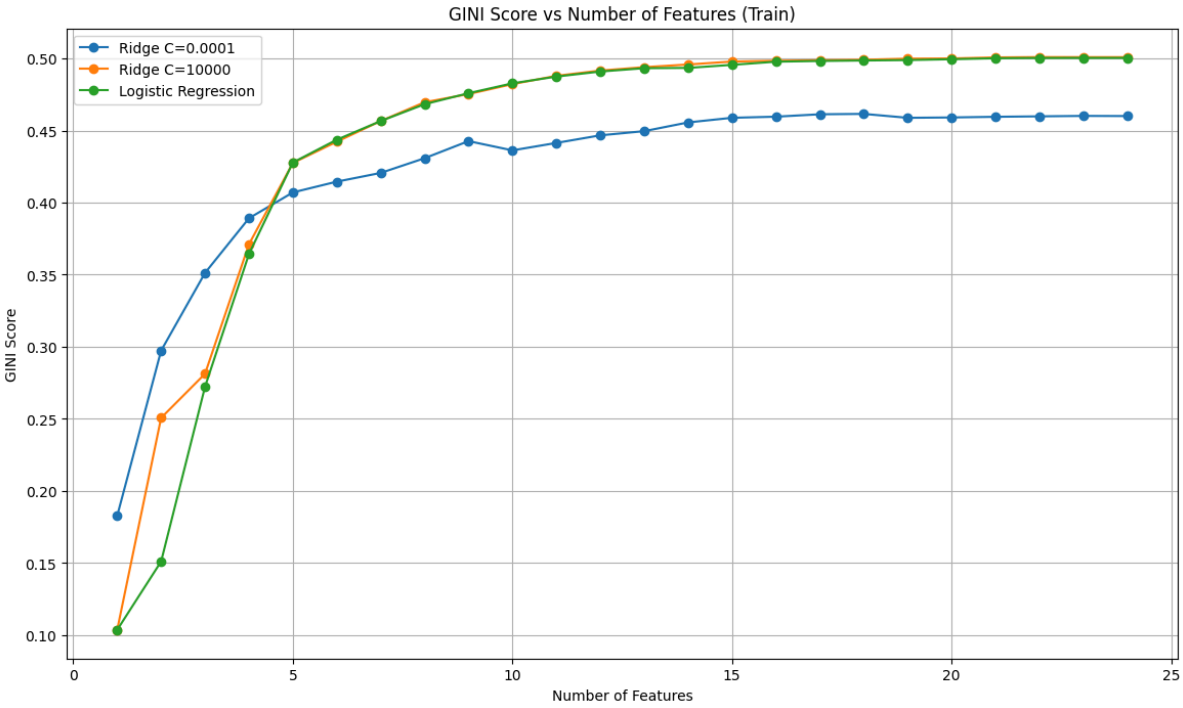

Type II Error vs Number of Features (Logistic Regression)
Error Rate
Number of Features
Type II Error Train
Type II Error Test

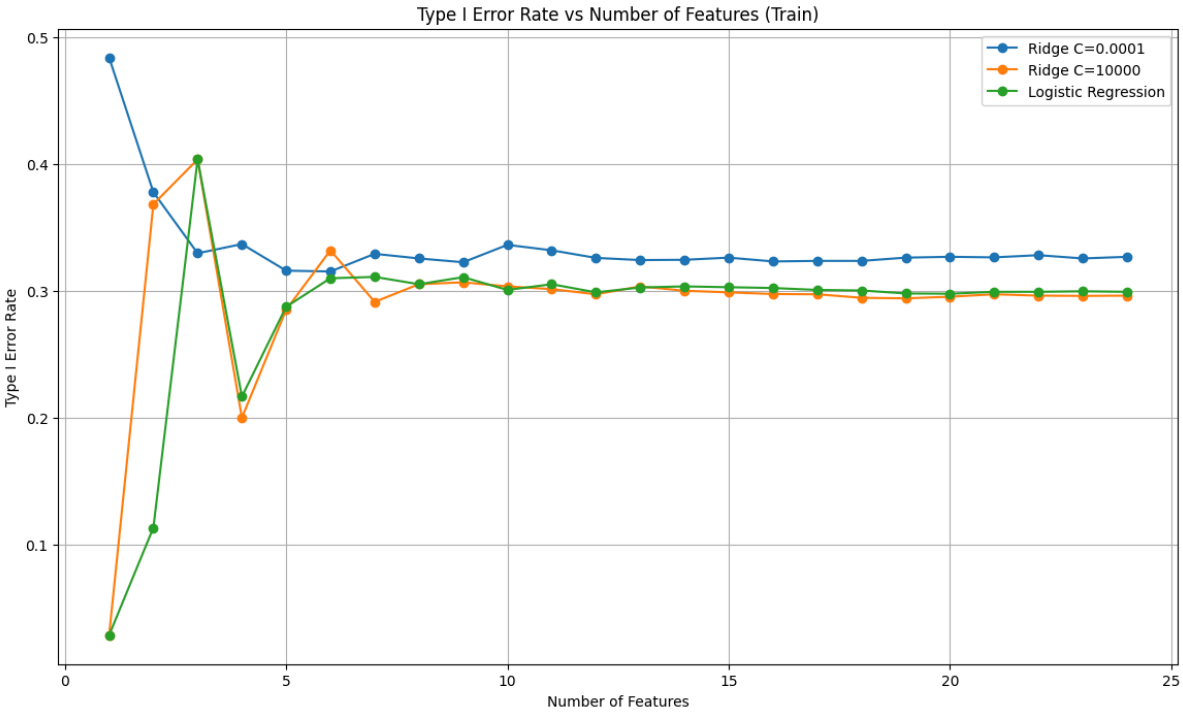

GINI Score vs Number of Features (Test)
GINI Score
Number of Features
Ridge C=0.0001
Ridge C=10000
Logistic Regression

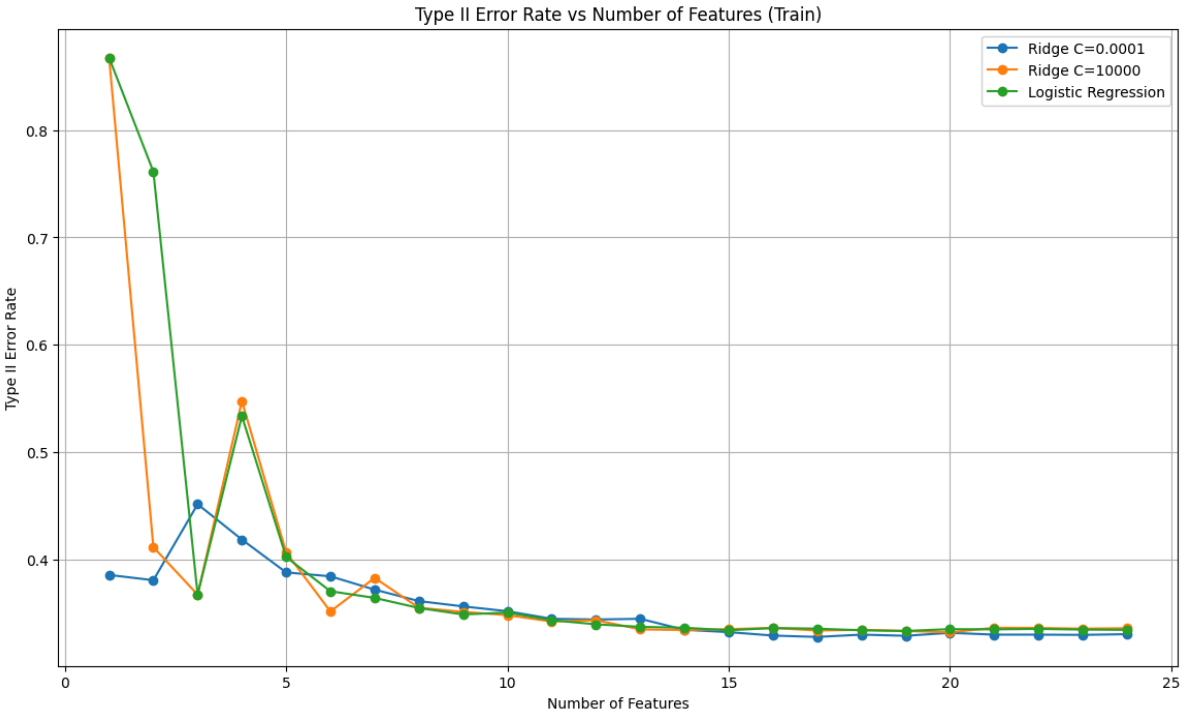

Type I Error Rate vs Number of Features (Test)
Ridge C=0.0001
Ridge C=10000
Logistic Regression
Type I Error Rate
Number of Features
0.1
0.2
0.3
0.4
0
5
10
15
20
25

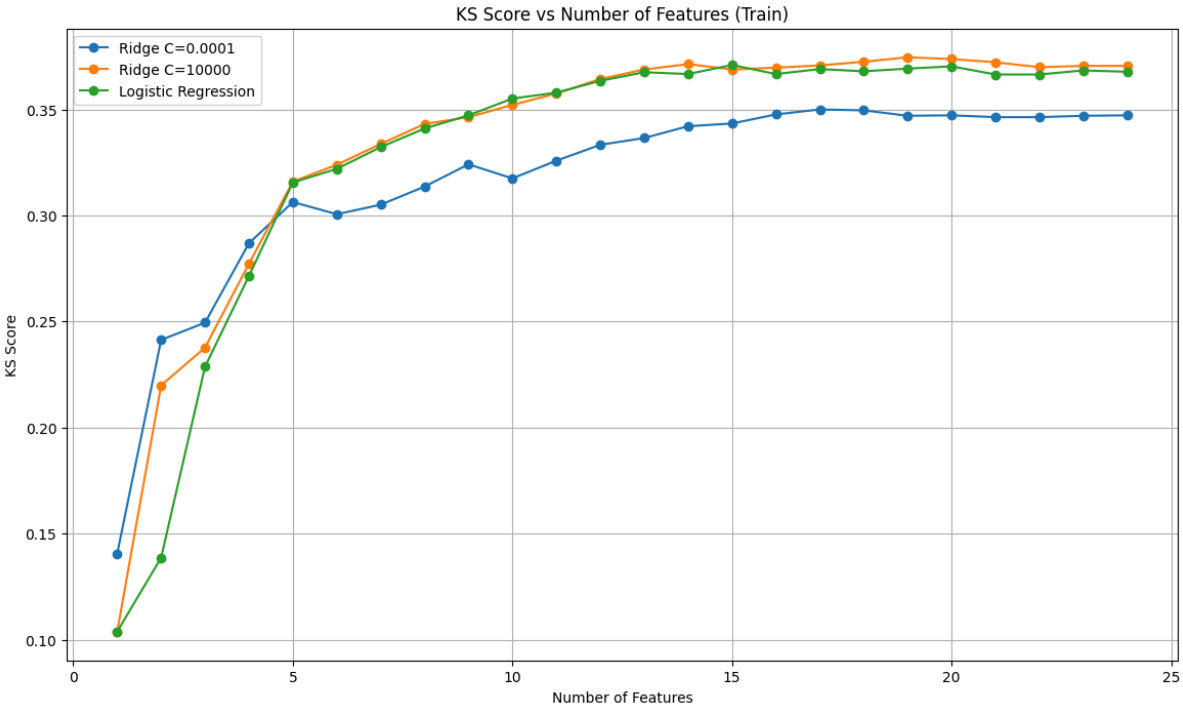

Type II Error Rate vs Number of Features (Test)
Ridge C=0.0001
Ridge C=10000
Logistic Regression
Type II Error Rate
Number of Features
0.4
0.5
0.6
0.7
0.8
0
5
10
15
20
25


KS Score vs Number of Features (Test)
Ridge C=0.0001
Ridge C=10000
Logistic Regression
KS Score
Number of Features
0.10
0.15
0.20
0.25
0.30
0.35
0
5
10
15
20
25


GINI Score vs Number of Features (Train)
Ridge C=0.0001
Ridge C=10000
Logistic Regression
GINI Score
Number of Features
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50
0
5
10
15
20
25


Type I Error Rate vs Number of Features (Train)
Ridge C=0.0001
Ridge C=10000
Logistic Regression
Type I Error Rate
Number of Features
0.5
0.4
0.3
0.2
0.1
0
5
10
15
20
25

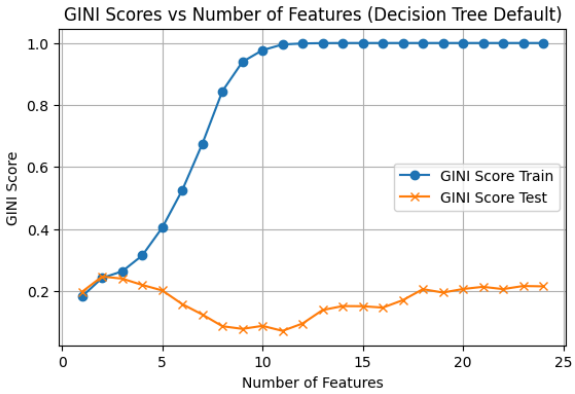

Type II Error Rate vs Number of Features (Train)
Ridge C=0.0001
Ridge C=10000
Logistic Regression
Type II Error Rate
Number of Features
0.8
0.7
0.6
0.5
0.4
0
5
10
15
20
25

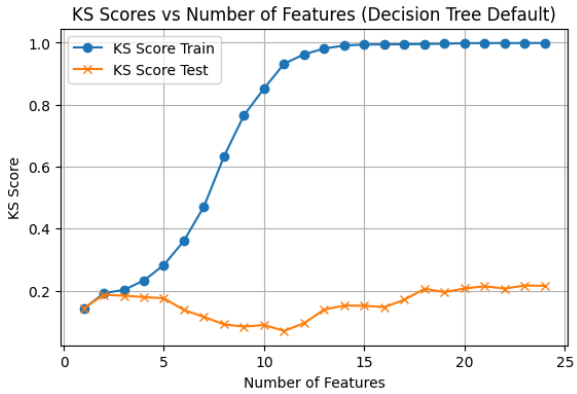


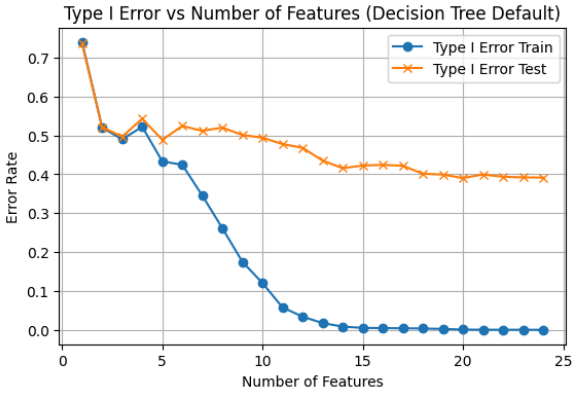
estymator drzewa decyzyjnego

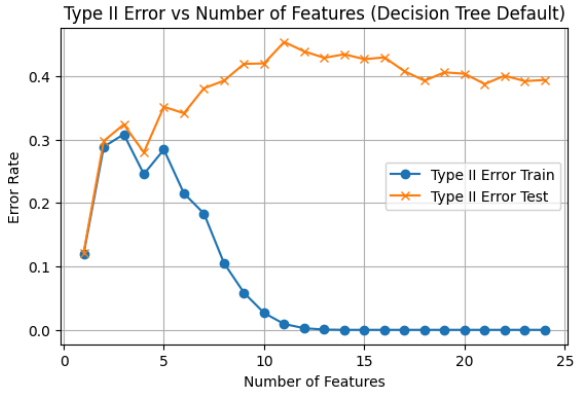
In [97]:
```
# Ewaluacja i wizualizacja wyników dla różnych modeli
models = [
    ('Decision Tree Default', DecisionTreeClassifier()),
    ('Decision Tree Depth 5', DecisionTreeClassifier(max_depth=5)),
    ('Decision Tree Depth 10', DecisionTreeClassifier(max_depth=10)),
    ('Decision Tree min_samples_split=10', DecisionTreeClassifier(min_sample
    ('Decision Tree min_samples_leaf=5', DecisionTreeClassifier(min_samples_
    ('Decision Tree max_features=log2', DecisionTreeClassifier(max_features=
]

for name, model in models:
    results = calculate_scores(model, X_train_balanced.values, y_train_balan
    plot_results(results, name, max_features)
```

	Number of Features	GINI Score Train	GINI Score Test	Type I Error Train	Type I Error Test	Type II Error Train	Type II Error Test	KS Score Train	KS
0	1	0.182726	0.197628	0.739837	0.735355	0.119598	0.121831	0.140565	0.14
1	2	0.242209	0.246788	0.520111	0.521304	0.288404	0.297535	0.191485	0.18
2	3	0.263545	0.240019	0.489944	0.496686	0.308087	0.323592	0.201968	0.18
3	4	0.314601	0.219677	0.522251	0.542688	0.245400	0.279577	0.232349	0.17
4	5	0.404570	0.201903	0.433462	0.488740	0.283911	0.351408	0.282627	0.1
5	6	0.525766	0.157717	0.424476	0.524208	0.215019	0.341197	0.360505	0.13
6	7	0.675062	0.124218	0.345315	0.511637	0.183141	0.380634	0.471545	0.1
7	8	0.844733	0.086525	0.261660	0.520067	0.105049	0.392606	0.633291	0.09
8	9	0.939217	0.078045	0.175439	0.500941	0.058194	0.419014	0.766367	0.08
9	10	0.976881	0.087747	0.120881	0.494151	0.026530	0.419718	0.852589	0.0
10	11	0.995277	0.071574	0.058408	0.477715	0.009200	0.453521	0.932392	0.0
11	12	0.998596	0.095344	0.034660	0.468115	0.002567	0.438732	0.962773	0.09
12	13	0.999651	0.138795	0.018186	0.435336	0.000428	0.428521	0.981386	0.13
13	14	0.999915	0.151535	0.009200	0.415827	0.000000	0.434155	0.990800	0.15
14	15	0.999967	0.151073	0.005777	0.422563	0.000000	0.426408	0.994223	0.15
15	16	0.999971	0.146576	0.005349	0.424298	0.000000	0.429225	0.994651	0.14
16	17	0.999976	0.170549	0.004921	0.422106	0.000000	0.407394	0.995079	0.17
17	18	0.999982	0.206054	0.004279	0.401569	0.000000	0.392606	0.995721	0.20
18	19	0.999990	0.195217	0.003209	0.399458	0.000000	0.405634	0.996791	0.19
19	20	0.999998	0.206567	0.001498	0.390356	0.000000	0.403521	0.998502	0.20
20	21	0.999999	0.213644	0.001070	0.399673	0.000000	0.387324	0.998930	0.21
21	22	0.999999	0.206234	0.001070	0.393771	0.000000	0.400352	0.998930	0.20
22	23	0.999999	0.216231	0.001070	0.392245	0.000000	0.391901	0.998930	0.21
23	24	0.999999	0.215131	0.000856	0.391660	0.000000	0.393662	0.999144	0.2

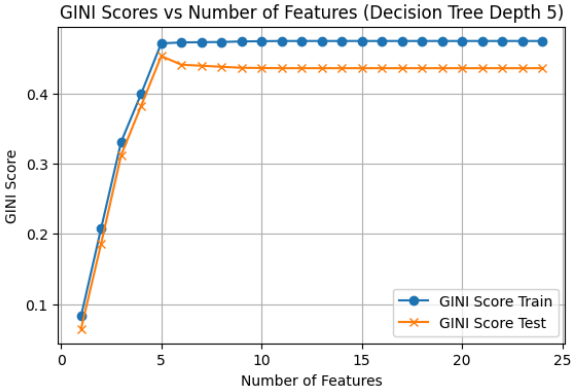

GINI Scores vs Number of Features (Decision Tree Default)
GINI Score
Number of Features
GINI Score Train
GINI Score Test

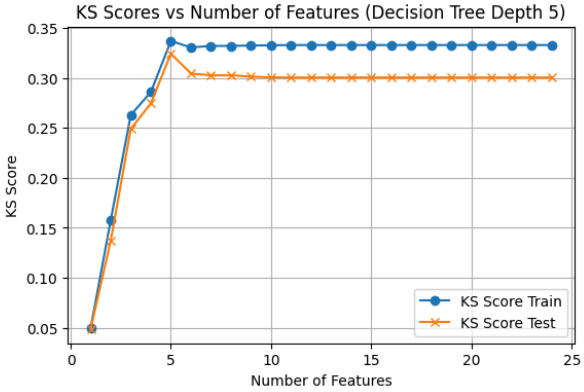

KS Scores vs Number of Features (Decision Tree Default)
KS Score
Number of Features
KS Score Train
KS Score Test

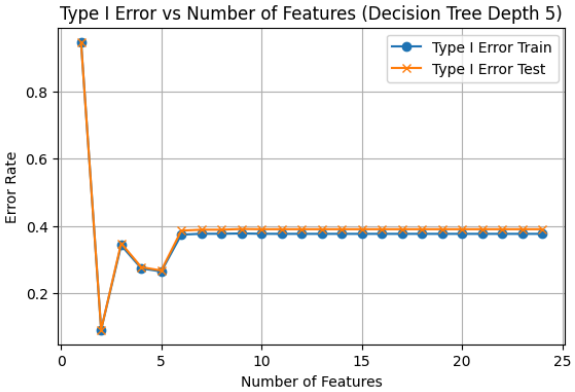

Type I Error vs Number of Features (Decision Tree Default)
Error Rate
Number of Features
Type I Error Train
Type I Error Test

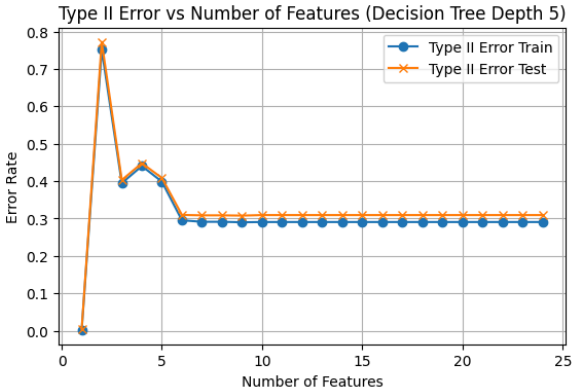

Type II Error vs Number of Features (Decision Tree Default)
Error Rate
Number of Features
Type II Error Train
Type II Error Test

	Number of Features	GINI Score Train	GINI Score Test	Type I Error Train	Type I Error Test	Type II Error Train	Type II Error Test	KS Score Train	KS S
0	1	0.082794	0.064626	0.948010	0.947208	0.002139	0.003873	0.049850	0.04
1	2	0.207699	0.185997	0.090929	0.090660	0.751391	0.772183	0.157681	0.1
2	3	0.330959	0.311975	0.342533	0.348313	0.394523	0.402817	0.262944	0.2
3	4	0.399485	0.382381	0.273855	0.277559	0.440308	0.447887	0.285837	0.27
4	5	0.471031	0.452973	0.264442	0.267273	0.398588	0.408451	0.336970	0.32
5	6	0.472321	0.440682	0.374198	0.386161	0.295250	0.309507	0.330552	0.30
6	7	0.472722	0.439220	0.376765	0.388890	0.291185	0.308451	0.332050	0.30
7	8	0.472946	0.437717	0.376765	0.388890	0.291185	0.308451	0.332050	0.30
8	9	0.473763	0.436302	0.377621	0.391001	0.289902	0.307746	0.332478	0.30
9	10	0.474096	0.435942	0.376765	0.390316	0.290543	0.309155	0.332691	0.30
10	11	0.474381	0.435709	0.376551	0.390396	0.290543	0.309155	0.332905	0.30
11	12	0.474381	0.435709	0.376551	0.390396	0.290543	0.309155	0.332905	0.30
12	13	0.474381	0.435709	0.376551	0.390396	0.290543	0.309155	0.332905	0.30
13	14	0.474381	0.435709	0.376551	0.390396	0.290543	0.309155	0.332905	0.30
14	15	0.474381	0.435709	0.376551	0.390396	0.290543	0.309155	0.332905	0.30
15	16	0.474381	0.435709	0.376551	0.390396	0.290543	0.309155	0.332905	0.30
16	17	0.474381	0.435709	0.376551	0.390396	0.290543	0.309155	0.332905	0.30
17	18	0.474381	0.435709	0.376551	0.390396	0.290543	0.309155	0.332905	0.30
18	19	0.474381	0.435709	0.376551	0.390396	0.290543	0.309155	0.332905	0.30
19	20	0.474381	0.435709	0.376551	0.390396	0.290543	0.309155	0.332905	0.30
20	21	0.474381	0.435709	0.376551	0.390396	0.290543	0.309155	0.332905	0.30
21	22	0.474381	0.435709	0.376551	0.390396	0.290543	0.309155	0.332905	0.30
22	23	0.474381	0.435709	0.376551	0.390396	0.290543	0.309155	0.332905	0.30
23	24	0.474381	0.435709	0.376551	0.390396	0.290543	0.309155	0.332905	0.30

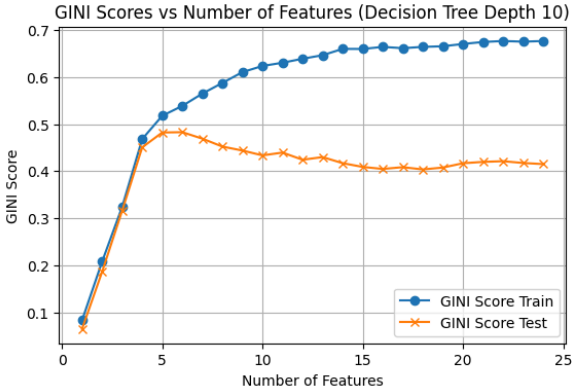

GINI Scores vs Number of Features (Decision Tree Depth 5)
GINI Score
Number of Features
GINI Score Train
GINI Score Test

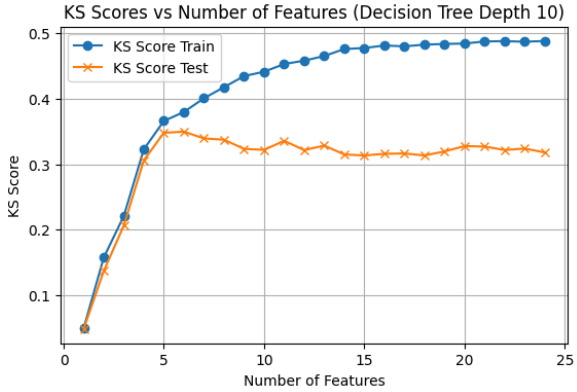

KS Scores vs Number of Features (Decision Tree Depth 5)
KS Score
Number of Features
KS Score Train
KS Score Test

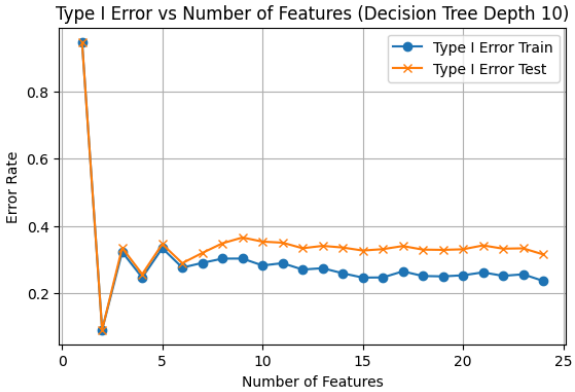

Type I Error vs Number of Features (Decision Tree Depth 5)
Error Rate
Number of Features
Type I Error Train
Type I Error Test

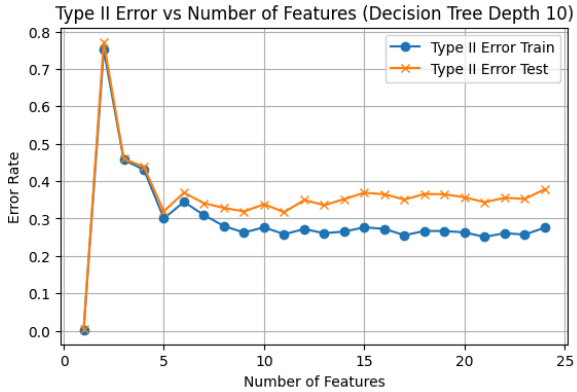

Type II Error vs Number of Features (Decision Tree Depth 5)
Error Rate
Number of Features
Type II Error Train
Type II Error Test

	Number of Features	GINI Score Train	GINI Score Test	Type I Error Train	Type I Error Test	Type II Error Train	Type II Error Test	KS Score Train	KS
0	1	0.082794	0.064626	0.948010	0.947208	0.002139	0.003873	0.049850	0.0
1	2	0.207699	0.185997	0.090929	0.090660	0.751391	0.772183	0.157681	0.1
2	3	0.324201	0.316112	0.322422	0.334653	0.456568	0.459507	0.221010	0.2
3	4	0.468387	0.450781	0.246256	0.256491	0.430894	0.438380	0.322850	0.3
4	5	0.517930	0.482318	0.333761	0.345786	0.300171	0.319366	0.366068	0.3
5	6	0.538891	0.482986	0.275995	0.289283	0.344245	0.369014	0.379760	0.3
6	7	0.565076	0.469228	0.290116	0.319749	0.309157	0.340845	0.400727	0.3
7	8	0.587753	0.452372	0.302739	0.348340	0.279632	0.328169	0.417629	0.3
8	9	0.610757	0.443952	0.302953	0.365684	0.262516	0.319014	0.434531	0.3
9	10	0.623845	0.433637	0.282413	0.353362	0.276423	0.338028	0.441164	0.3
10	11	0.630729	0.439629	0.289260	0.350216	0.257809	0.316901	0.452931	0.3
11	12	0.639208	0.424367	0.270218	0.333645	0.271716	0.349296	0.458066	0.3
12	13	0.646757	0.429989	0.274283	0.340717	0.260591	0.335915	0.465126	0.3
13	14	0.660211	0.417055	0.259093	0.335823	0.265083	0.351761	0.475824	0.3
14	15	0.660033	0.409097	0.246256	0.326815	0.276423	0.369014	0.477321	0.3
15	16	0.664447	0.404800	0.246684	0.330909	0.272144	0.364789	0.481172	0.3
16	17	0.661678	0.408573	0.264869	0.340240	0.255242	0.350704	0.479889	0.3
17	18	0.664435	0.403726	0.250963	0.329571	0.266581	0.365493	0.482456	0.3
18	19	0.665762	0.407785	0.249893	0.328811	0.266581	0.364789	0.483526	0.3
19	20	0.670639	0.416898	0.253102	0.330505	0.262730	0.356690	0.484168	0.3
20	21	0.674876	0.419795	0.261874	0.342176	0.250749	0.343662	0.487377	0.3
21	22	0.676939	0.421266	0.251391	0.332368	0.260804	0.355282	0.487805	0.3
22	23	0.675586	0.417482	0.256098	0.333389	0.256953	0.352465	0.486949	0.3
23	24	0.676896	0.415114	0.236414	0.315299	0.275567	0.378169	0.488019	0.3

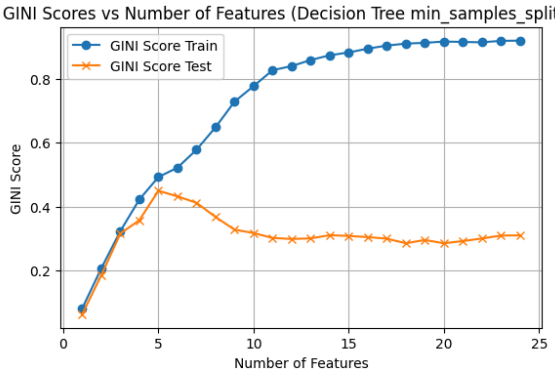

GINI Scores vs Number of Features (Decision Tree Depth 10)
GINI Score
Number of Features
GINI Score Train
GINI Score Test

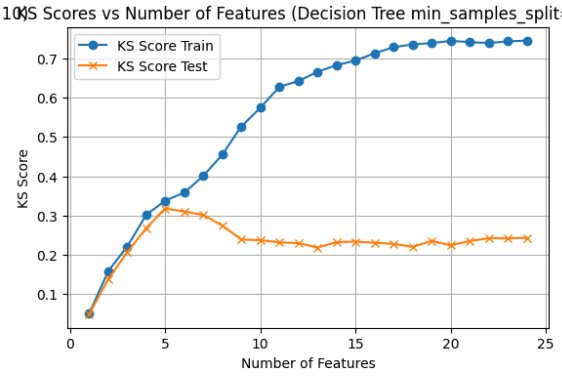

KS Scores vs Number of Features (Decision Tree Depth 10)
KS Score
Number of Features
KS Score Train
KS Score Test

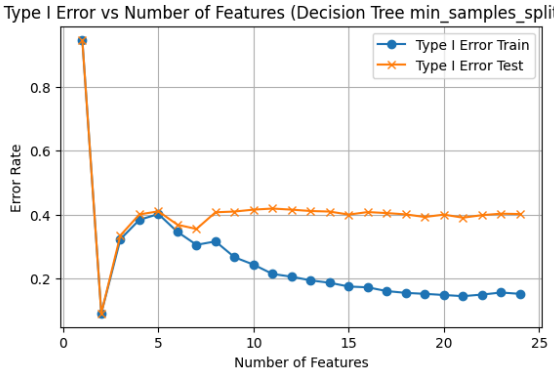

Type I Error vs Number of Features (Decision Tree Depth 10)
Error Rate
Number of Features
Type I Error Train
Type I Error Test

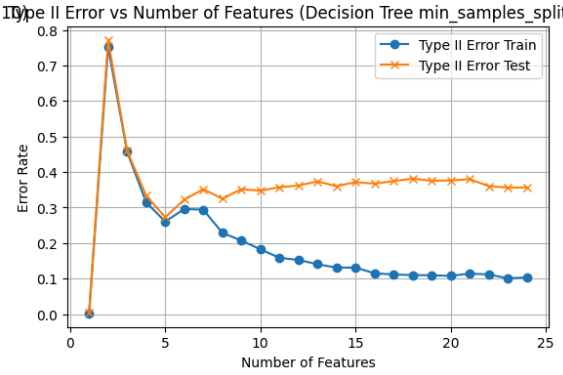

Type II Error vs Number of Features (Decision Tree Depth 10)
Error Rate
Number of Features
Type II Error Train
Type II Error Test

	Number of Features	GINI Score Train	GINI Score Test	Type I Error Train	Type I Error Test	Type II Error Train	Type II Error Test	KS Score Train	KS S
0	1	0.082794	0.064626	0.948010	0.947208	0.002139	0.003873	0.049850	0.04
1	2	0.207699	0.185997	0.090929	0.090660	0.751391	0.772183	0.157681	0.1
2	3	0.323868	0.316164	0.322636	0.334660	0.456568	0.459507	0.220796	0.2
3	4	0.422473	0.357821	0.383825	0.400695	0.314720	0.333451	0.301455	0.26
4	5	0.492513	0.449738	0.401797	0.410308	0.260377	0.272887	0.337826	0.3
5	6	0.521748	0.432265	0.345956	0.368306	0.295892	0.322183	0.358151	0.3
6	7	0.577622	0.412225	0.305306	0.355110	0.293967	0.351761	0.400727	0.3
7	8	0.647807	0.367861	0.316003	0.407384	0.228926	0.324648	0.455071	0.2
8	9	0.727906	0.328355	0.266367	0.409690	0.206889	0.351761	0.526744	0.23
9	10	0.776402	0.317669	0.243261	0.415680	0.182499	0.347535	0.574240	0.23
10	11	0.826123	0.302408	0.213950	0.419034	0.158323	0.357042	0.627728	0.23
11	12	0.838922	0.299126	0.205392	0.415619	0.152332	0.361972	0.642276	0.22
12	13	0.857820	0.301073	0.193838	0.411310	0.140351	0.373592	0.665811	0.21
13	14	0.872709	0.310994	0.186350	0.409596	0.130723	0.360211	0.682927	0.23
14	15	0.881543	0.308597	0.174583	0.400117	0.130937	0.371479	0.694480	0.23
15	16	0.893411	0.305122	0.172015	0.408076	0.114891	0.367254	0.713094	0.23
16	17	0.902944	0.300164	0.160248	0.404816	0.111254	0.374296	0.728498	0.22
17	18	0.909134	0.285789	0.154685	0.400782	0.109756	0.381338	0.735558	0.2
18	19	0.911584	0.296015	0.151690	0.392588	0.109114	0.375352	0.739196	0.23
19	20	0.915543	0.285734	0.148053	0.400104	0.107403	0.376056	0.744544	0.22
20	21	0.914301	0.292681	0.144416	0.390524	0.114035	0.379930	0.741549	0.23
21	22	0.913180	0.300717	0.149337	0.398342	0.111468	0.360211	0.739196	0.24
22	23	0.917703	0.309817	0.155755	0.402940	0.100770	0.355986	0.743475	0.24
23	24	0.918321	0.310428	0.151690	0.401858	0.103124	0.356690	0.745186	0.24

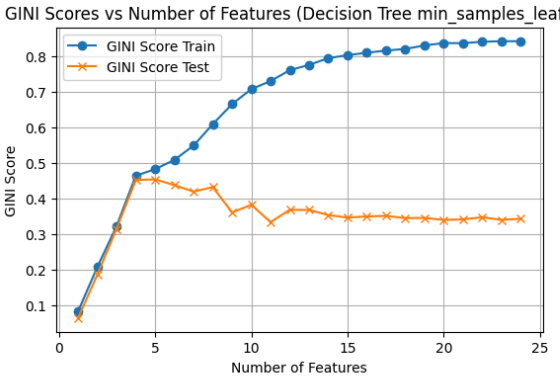

GINI Scores vs Number of Features (Decision Tree min_samples_split=10)
GINI Score Train
GINI Score Test
GINI Score
Number of Features

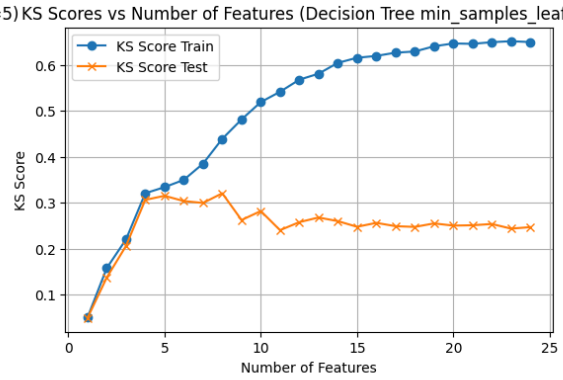

KS Scores vs Number of Features (Decision Tree min_samples_split=10)
KS Score Train
KS Score Test
KS Score
Number of Features

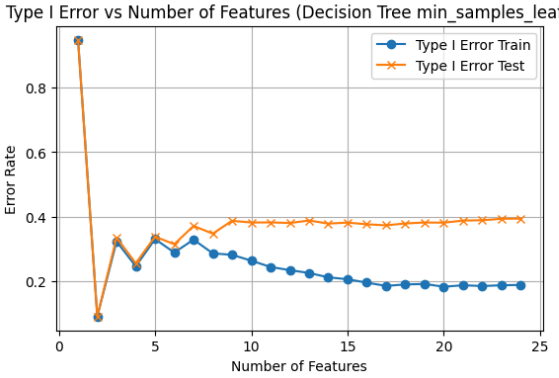

Type I Error vs Number of Features (Decision Tree min_samples_split=10)
Type I Error Train
Type I Error Test
Error Rate
Number of Features

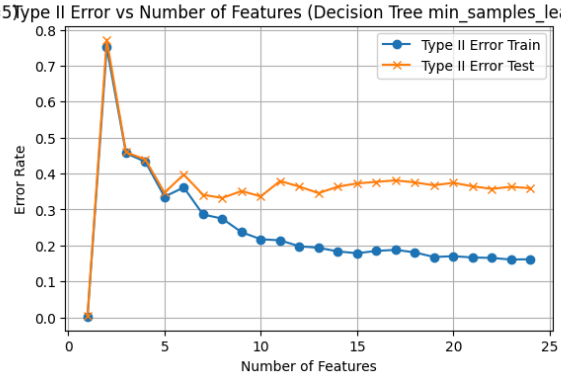

Type II Error vs Number of Features (Decision Tree min_samples_split=10)
Type II Error Train
Type II Error Test
Error Rate
Number of Features

	Number of Features	GINI Score Train	GINI Score Test	Type I Error Train	Type I Error Test	Type II Error Train	Type II Error Test	KS Score Train	KS
0	1	0.082794	0.064626	0.948010	0.947208	0.002139	0.003873	0.049850	0.0
1	2	0.207380	0.185605	0.091356	0.090888	0.751177	0.772183	0.157467	0.1
2	3	0.322345	0.315110	0.323492	0.335285	0.456568	0.459507	0.219940	0.2
3	4	0.464058	0.452040	0.246470	0.255785	0.433034	0.439085	0.320496	0.3
4	5	0.482434	0.453857	0.330766	0.338324	0.335473	0.346831	0.333761	0.3
5	6	0.508348	0.437773	0.289046	0.314815	0.361361	0.397183	0.349593	0.3
6	7	0.548673	0.419469	0.329054	0.371398	0.286050	0.340845	0.384895	0.2
7	8	0.609001	0.432135	0.286478	0.347560	0.274497	0.332042	0.439024	0.3
8	9	0.665253	0.360989	0.281985	0.387405	0.236842	0.351408	0.481172	0.2
9	10	0.707371	0.382410	0.263800	0.382033	0.217159	0.336268	0.519042	0.2
10	11	0.729955	0.333662	0.244330	0.382363	0.214163	0.379225	0.541506	0.2
11	12	0.760608	0.368425	0.234489	0.380467	0.197689	0.363732	0.567822	0.2
12	13	0.775434	0.367907	0.225717	0.388272	0.193410	0.345423	0.580873	0.2
13	14	0.794375	0.353552	0.212238	0.378739	0.183355	0.363380	0.604407	0.2
14	15	0.802630	0.346563	0.206247	0.381771	0.178220	0.372535	0.615533	0.2
15	16	0.809684	0.349746	0.195764	0.376534	0.184638	0.376761	0.619598	0.2
16	17	0.815603	0.351264	0.185708	0.373274	0.187634	0.380986	0.626658	0.2
17	18	0.820383	0.345042	0.190201	0.379156	0.180573	0.375352	0.629226	0.2
18	19	0.830073	0.345473	0.191699	0.382114	0.167736	0.367254	0.640565	0.2
19	20	0.836742	0.339784	0.183141	0.381536	0.170090	0.374296	0.646769	0.2
20	21	0.836387	0.341517	0.187634	0.388070	0.166453	0.364437	0.645914	0.2
21	22	0.840421	0.347333	0.185280	0.389119	0.165169	0.357394	0.649551	0.2
22	23	0.842035	0.340061	0.187634	0.393549	0.160676	0.363028	0.651690	0.2
23	24	0.841785	0.343137	0.188703	0.394194	0.161532	0.359155	0.649765	0.2

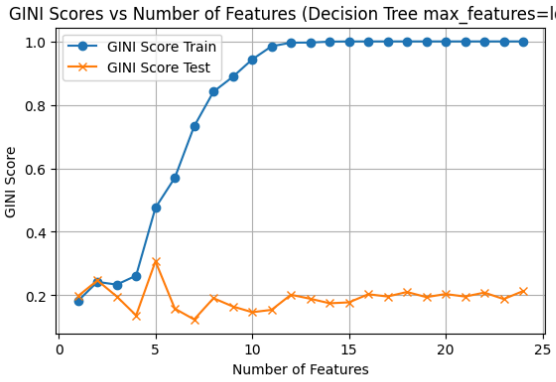

GINI Scores vs Number of Features (Decision Tree min_samples_leaf=5)
GINI Score Train
GINI Score Test
GINI Score
Number of Features

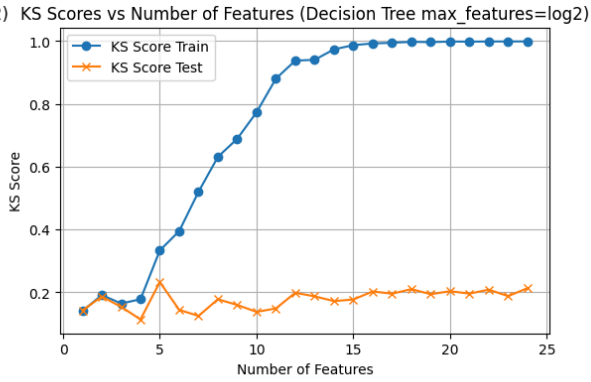

KS Scores vs Number of Features (Decision Tree min_samples_leaf=5)
KS Score Train
KS Score Test
KS Score
Number of Features

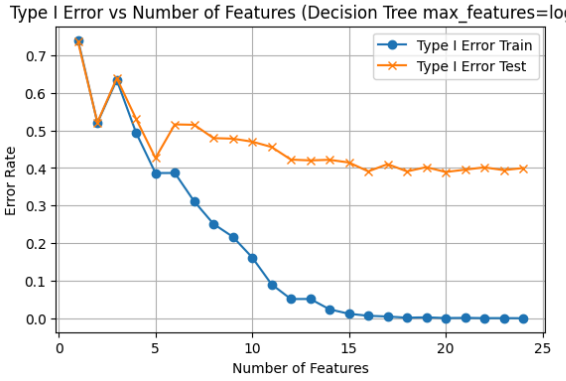

Type I Error vs Number of Features (Decision Tree min_samples_leaf=5)
Type I Error Train
Type I Error Test
Error Rate
Number of Features

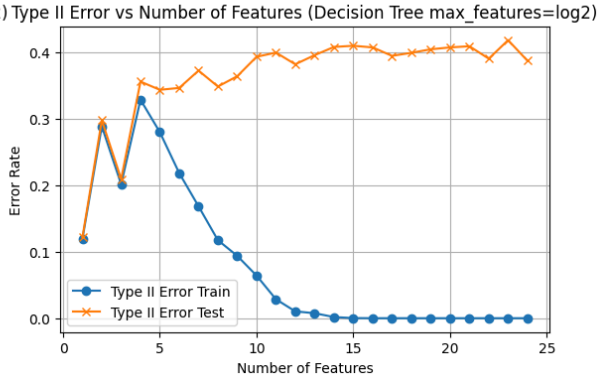

Type II Error vs Number of Features (Decision Tree min_samples_leaf=5)
Type II Error Train
Type II Error Test
Error Rate
Number of Features

	Number of Features	GINI Score Train	GINI Score Test	Type I Error Train	Type I Error Test	Type II Error Train	Type II Error Test	KS Score Train	KS S
0	1	0.182726	0.197628	0.739837	0.735355	0.119598	0.121831	0.140565	0.14
1	2	0.242209	0.246788	0.520111	0.521304	0.288404	0.297535	0.191485	0.18
2	3	0.233600	0.196186	0.634146	0.638679	0.201326	0.209155	0.164527	0.15
3	4	0.261972	0.136682	0.493582	0.530608	0.328412	0.355634	0.178006	0.1
4	5	0.475853	0.307150	0.386607	0.426496	0.279632	0.343310	0.333761	0.23
5	6	0.570206	0.157878	0.386821	0.515932	0.218228	0.346127	0.394951	0.14
6	7	0.732542	0.123883	0.311510	0.514897	0.168164	0.372183	0.520325	0.12
7	8	0.841568	0.190796	0.251391	0.479463	0.117244	0.348239	0.631365	0.17
8	9	0.888510	0.164800	0.217159	0.478051	0.093710	0.363732	0.689131	0.15
9	10	0.943093	0.146697	0.162388	0.470058	0.063971	0.392958	0.773641	0.13
10	11	0.984645	0.154061	0.090929	0.455914	0.028455	0.399296	0.880616	0.14
11	12	0.995996	0.201627	0.051776	0.422362	0.010270	0.381690	0.937955	0.19
12	13	0.996349	0.189127	0.051990	0.420305	0.007488	0.395423	0.940522	0.18
13	14	0.999312	0.174837	0.024390	0.421871	0.001712	0.407746	0.973898	0.17
14	15	0.999835	0.177684	0.012623	0.414644	0.000214	0.409507	0.987163	0.17
15	16	0.999947	0.203247	0.007274	0.391102	0.000000	0.407042	0.992726	0.20
16	17	0.999971	0.196143	0.005349	0.410322	0.000000	0.394366	0.994651	0.19
17	18	0.999995	0.210424	0.002139	0.391035	0.000000	0.398944	0.997861	0.21
18	19	0.999990	0.194226	0.003209	0.402322	0.000000	0.404225	0.996791	0.19
19	20	0.999998	0.203937	0.001284	0.389381	0.000000	0.407042	0.998716	0.20
20	21	0.999997	0.195949	0.001712	0.395673	0.000000	0.408803	0.998288	0.19
21	22	0.999999	0.208452	0.001070	0.401757	0.000000	0.390141	0.998930	0.20
22	23	0.999999	0.188085	0.001070	0.394363	0.000000	0.417958	0.998930	0.18
23	24	0.999999	0.213234	0.000856	0.399962	0.000000	0.386972	0.999144	0.2





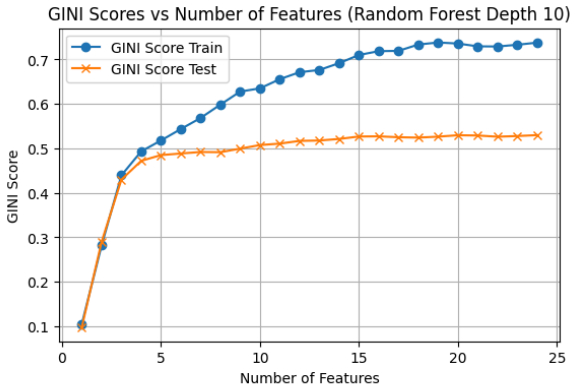


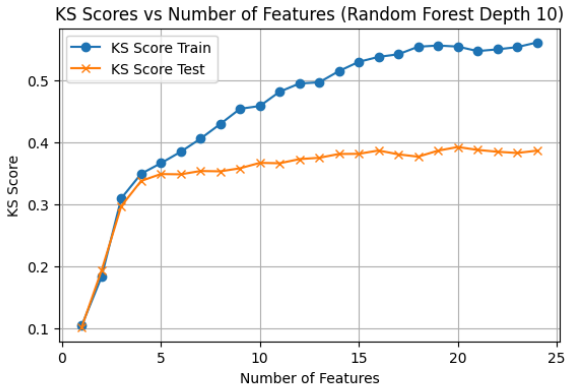


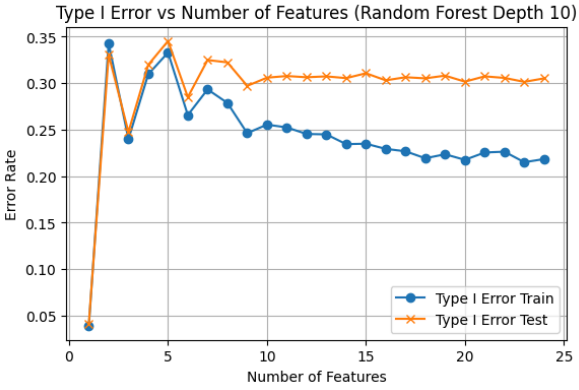
estymator lasu losowego

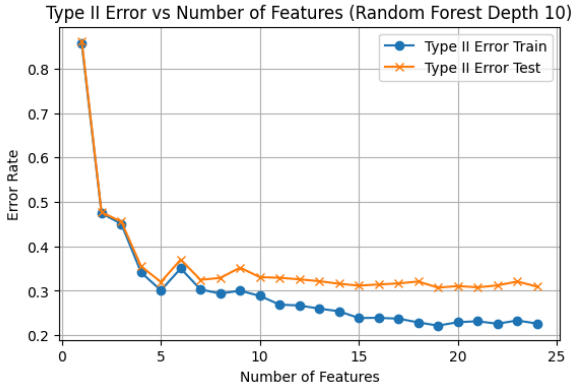
In [98]:
```
from sklearn.ensemble import RandomForestClassifier

models_rf = [
#     ('Random Forest Default', RandomForestClassifier()),
    ('Random Forest Depth 10', RandomForestClassifier(max_depth=10)),
    ('Random Forest n_estimators=200', RandomForestClassifier(n_estimators=2
    ('Random Forest min_samples_split=5', RandomForestClassifier(min_samples
    ('Random Forest min_samples_leaf=3', RandomForestClassifier(min_samples_
]

for name, model in models_rf:
    results = calculate_scores(model, X_train_balanced.values, y_train_balan
    plot_results(results, name, max_features)
```

	Number of Features	GINI Score Train	GINI Score Test	Type I Error Train	Type I Error Test	Type II Error Train	Type II Error Test	KS Score Train	KS S
0	1	0.105424	0.098180	0.039367	0.041659	0.857082	0.862324	0.103766	0.10
1	2	0.282004	0.290269	0.342961	0.330472	0.474540	0.476761	0.182499	0.19
2	3	0.439504	0.429832	0.239837	0.247570	0.450578	0.455634	0.309585	0.29
3	4	0.493292	0.472011	0.309585	0.319696	0.341249	0.354930	0.349166	0.33
4	5	0.517838	0.484913	0.332905	0.345315	0.300813	0.319718	0.366282	0.34
5	6	0.543482	0.488497	0.265725	0.285001	0.350235	0.370070	0.384681	0.34
6	7	0.568395	0.491893	0.293325	0.324899	0.302953	0.324648	0.405862	0.35
7	8	0.598454	0.491412	0.278776	0.322358	0.293539	0.328873	0.429611	0.35
8	9	0.627773	0.499497	0.246256	0.297142	0.300171	0.352113	0.454215	0.35
9	10	0.635009	0.507460	0.255456	0.305921	0.288404	0.330634	0.458708	0.36
10	11	0.655646	0.510740	0.252460	0.307696	0.268721	0.329225	0.481814	0.36
11	12	0.671311	0.516837	0.245400	0.306372	0.266795	0.325704	0.495079	0.37
12	13	0.676359	0.517713	0.244972	0.307319	0.259735	0.321479	0.497005	0.37
13	14	0.691711	0.521206	0.234489	0.305424	0.253744	0.315845	0.515190	0.38
14	15	0.710043	0.526720	0.234917	0.310654	0.238554	0.311620	0.530167	0.38
15	16	0.718654	0.527083	0.229354	0.303071	0.238982	0.314085	0.538083	0.38
16	17	0.719326	0.525078	0.226786	0.306392	0.237056	0.316549	0.542362	0.38
17	18	0.733533	0.524259	0.219298	0.305202	0.228712	0.320775	0.554557	0.37
18	19	0.738189	0.526334	0.223577	0.308106	0.221438	0.307042	0.556483	0.38
19	20	0.735784	0.529594	0.217587	0.301646	0.229140	0.310563	0.554771	0.39
20	21	0.729510	0.528982	0.225503	0.307340	0.231279	0.307746	0.547497	0.38
21	22	0.729121	0.526257	0.226359	0.305672	0.225717	0.311972	0.550278	0.38
22	23	0.733033	0.527782	0.215019	0.301417	0.232991	0.320775	0.553915	0.38
23	24	0.737808	0.529826	0.218442	0.305182	0.226145	0.309507	0.561404	0.38

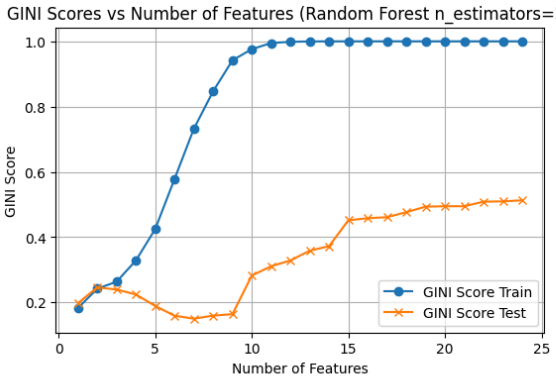

GINI Scores vs Number of Features (Random Forest Depth 10)
GINI Score Train
GINI Score Test
GINI Score
Number of Features

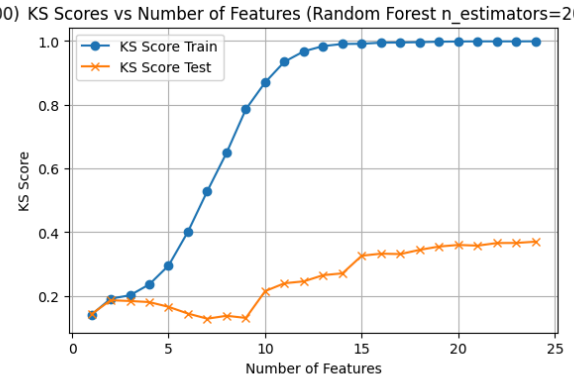

KS Scores vs Number of Features (Random Forest Depth 10)
KS Score Train
KS Score Test
KS Score
Number of Features

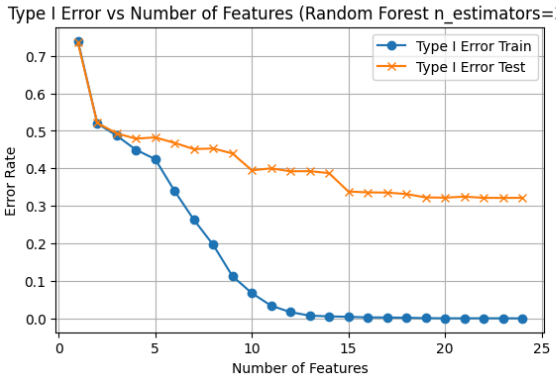

Type I Error vs Number of Features (Random Forest Depth 10)
Type I Error Train
Type I Error Test
Error Rate
Number of Features

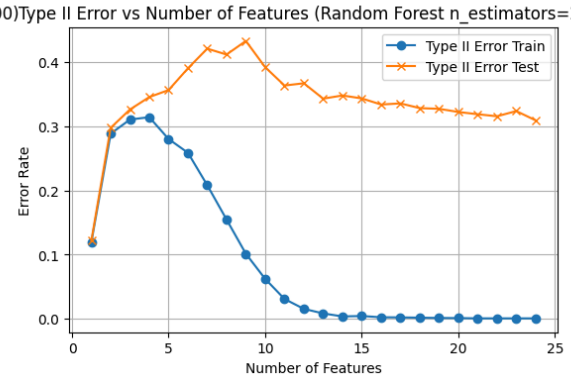

Type II Error vs Number of Features (Random Forest Depth 10)
Type II Error Train
Type II Error Test
Error Rate
Number of Features

	Number of Features	GINI Score Train	GINI Score Test	Type I Error Train	Type I Error Test	Type II Error Train	Type II Error Test	KS Score Train	KS
0	1	0.182726	0.197628	0.739837	0.735355	0.119598	0.121831	0.140565	0.14
1	2	0.242149	0.246167	0.520111	0.521304	0.288404	0.297535	0.191485	0.18
2	3	0.263435	0.239590	0.487591	0.493506	0.310441	0.326056	0.201968	0.18
3	4	0.328562	0.224497	0.449722	0.479301	0.314078	0.345423	0.236200	0.18
4	5	0.424565	0.189175	0.424690	0.482602	0.280060	0.356338	0.295464	0.16
5	6	0.578027	0.159394	0.339966	0.468129	0.258665	0.390141	0.401369	0.14
6	7	0.731787	0.150291	0.263158	0.451410	0.208173	0.420775	0.528669	0.12
7	8	0.846955	0.159636	0.196192	0.453124	0.154685	0.411620	0.649123	0.13
8	9	0.941745	0.164033	0.112537	0.439840	0.101412	0.432746	0.786050	0.13
9	10	0.975910	0.282366	0.067394	0.394833	0.062259	0.391901	0.870347	0.2
10	11	0.994551	0.310833	0.033804	0.400097	0.031023	0.363380	0.935173	0.23
11	12	0.998448	0.328507	0.016688	0.392272	0.015618	0.366901	0.967694	0.24
12	13	0.999651	0.358553	0.007274	0.392628	0.008558	0.342958	0.984168	0.26
13	14	0.999906	0.372282	0.005349	0.387371	0.003851	0.347887	0.990800	0.27
14	15	0.999913	0.451650	0.004065	0.338277	0.004493	0.343310	0.991442	0.32
15	16	0.999974	0.457990	0.002567	0.335971	0.002567	0.333803	0.994865	0.33
16	17	0.999978	0.461155	0.002353	0.335467	0.002353	0.335563	0.995293	0.33
17	18	0.999986	0.476559	0.001712	0.331648	0.001926	0.328169	0.996363	0.34
18	19	0.999995	0.493429	0.000642	0.322620	0.001498	0.327113	0.997861	0.3
19	20	0.999998	0.494765	0.000214	0.321914	0.001284	0.322183	0.998502	0.36
20	21	0.999999	0.494686	0.000214	0.324690	0.000856	0.318662	0.998930	0.35
21	22	0.999999	0.508776	0.000214	0.321497	0.000856	0.315493	0.998930	0.3
22	23	0.999999	0.509557	0.000214	0.321269	0.000856	0.323592	0.998930	0.3
23	24	0.999999	0.512913	0.000000	0.321538	0.000856	0.309155	0.999144	0.37

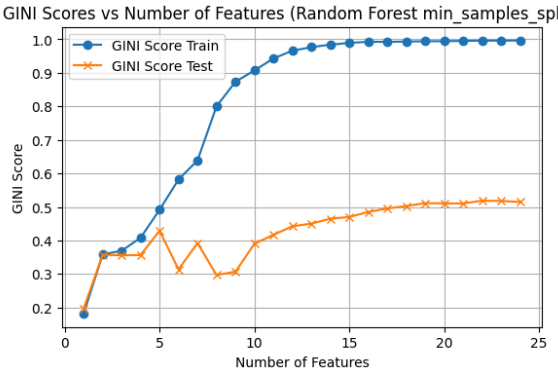

GINI Scores vs Number of Features (Random Forest n_estimators=200)
GINI Score
Number of Features
GINI Score Train
GINI Score Test

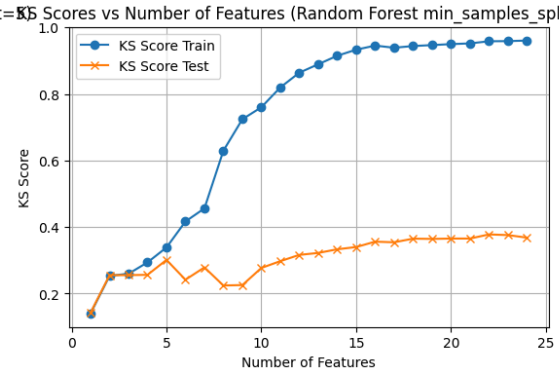

KS Scores vs Number of Features (Random Forest n_estimators=200)
KS Score
Number of Features
KS Score Train
KS Score Test

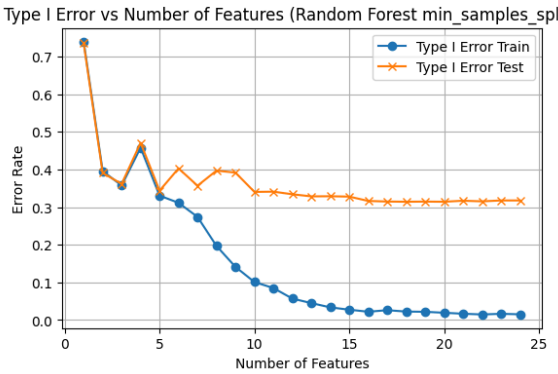

Type I Error vs Number of Features (Random Forest n_estimators=200)
Error Rate
Number of Features
Type I Error Train
Type I Error Test

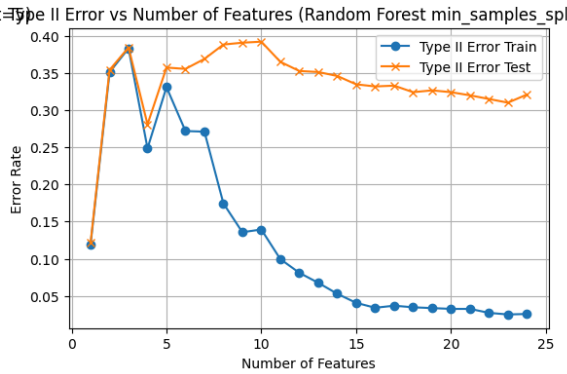

Type II Error vs Number of Features (Random Forest n_estimators=200)
Error Rate
Number of Features
Type II Error Train
Type II Error Test

	Number of Features	GINI Score Train	GINI Score Test	Type I Error Train	Type I Error Test	Type II Error Train	Type II Error Test	KS Score Train	KS S
0	1	0.182726	0.197628	0.739837	0.735355	0.119598	0.121831	0.140565	0.14
1	2	0.358357	0.357450	0.395807	0.390853	0.350877	0.353873	0.253316	0.25
2	3	0.369297	0.355785	0.357510	0.361718	0.382970	0.384155	0.259735	0.25
3	4	0.408220	0.356576	0.458066	0.470945	0.248395	0.280634	0.293753	0.25
4	5	0.491699	0.429152	0.330552	0.343769	0.330766	0.357042	0.338682	0.3
5	6	0.581972	0.313711	0.311510	0.403001	0.271716	0.355282	0.416774	0.2
6	7	0.638757	0.392306	0.274069	0.356938	0.270860	0.368662	0.456140	0.2
7	8	0.800042	0.297103	0.196834	0.396998	0.174369	0.388028	0.629653	0.2
8	9	0.873520	0.306860	0.141207	0.391774	0.135644	0.390493	0.724219	0.22
9	10	0.907012	0.390252	0.100984	0.340468	0.139281	0.391901	0.760163	0.2
10	11	0.942976	0.416791	0.084938	0.341114	0.099487	0.364789	0.819855	0.2
11	12	0.966031	0.442343	0.056911	0.334196	0.081087	0.352465	0.864142	0.3
12	13	0.976690	0.450346	0.044929	0.328603	0.067608	0.351056	0.889602	0.3
13	14	0.984079	0.464958	0.033590	0.328932	0.052846	0.345775	0.915062	0.3
14	15	0.989256	0.470261	0.027386	0.327971	0.040650	0.334507	0.933462	0.33
15	16	0.992763	0.485155	0.022037	0.316496	0.034018	0.331690	0.945657	0.3
16	17	0.992434	0.495485	0.026102	0.314855	0.036799	0.332746	0.939238	0.3
17	18	0.993376	0.502695	0.022465	0.314344	0.034660	0.323944	0.944373	0.3
18	19	0.994129	0.511389	0.021823	0.314734	0.033590	0.326408	0.947154	0.3
19	20	0.994513	0.510711	0.019255	0.314331	0.032520	0.323944	0.950150	0.3
20	21	0.995296	0.510418	0.016688	0.317067	0.032520	0.319718	0.952289	0.36
21	22	0.995934	0.518783	0.014763	0.315380	0.027172	0.314789	0.958708	0.37
22	23	0.996146	0.518065	0.016474	0.317786	0.025032	0.309859	0.959136	0.3
23	24	0.996372	0.514902	0.015190	0.317699	0.025674	0.320775	0.960847	0.3

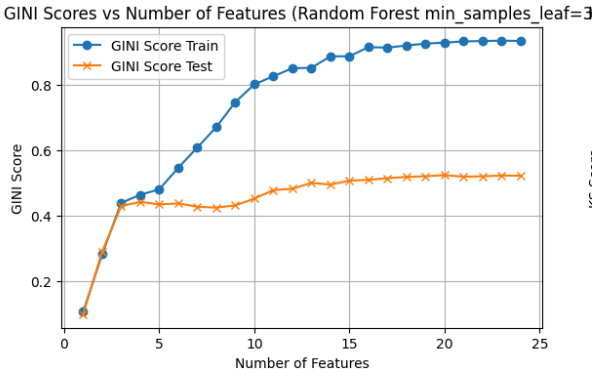

GINI Scores vs Number of Features (Random Forest min_samples_split=5)
GINI Score Train
GINI Score Test
GINI Score
Number of Features

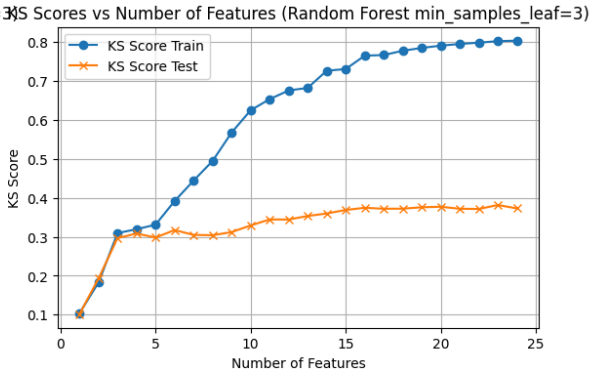

KS Scores vs Number of Features (Random Forest min_samples_split=5)
KS Score Train
KS Score Test
KS Score
Number of Features

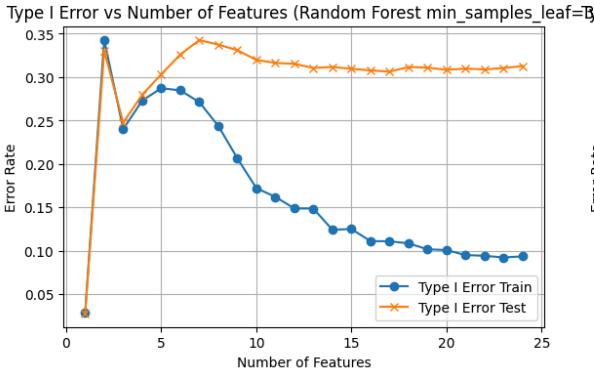

Type I Error vs Number of Features (Random Forest min_samples_split=5)
Type I Error Train
Type I Error Test
Error Rate
Number of Features

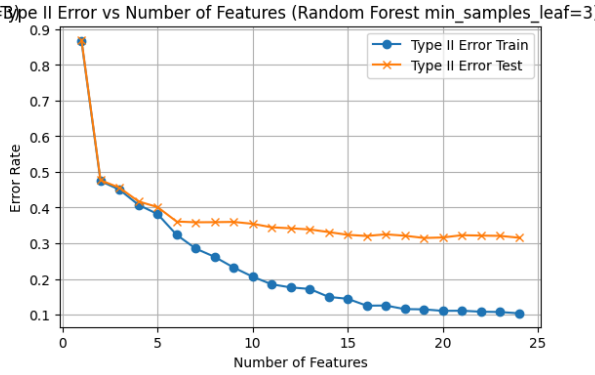

Type II Error vs Number of Features (Random Forest min_samples_split=5)
Type II Error Train
Type II Error Test
Error Rate
Number of Features

	Number of Features	GINI Score Train	GINI Score Test	Type I Error Train	Type I Error Test	Type II Error Train	Type II Error Test	KS Score Train	KS S
0	1	0.105424	0.098180	0.028883	0.028147	0.867351	0.870423	0.103766	0.10
1	2	0.282004	0.290274	0.342961	0.330472	0.474540	0.476761	0.182499	0.19
2	3	0.439374	0.429593	0.240265	0.247657	0.450150	0.455634	0.309585	0.29
3	4	0.464065	0.442208	0.273000	0.279791	0.407574	0.417254	0.319427	0.30
4	5	0.480938	0.434520	0.287334	0.303266	0.381900	0.401408	0.331194	0.29
5	6	0.546749	0.437646	0.284767	0.326028	0.323706	0.360915	0.391742	0.31
6	7	0.608876	0.427535	0.271288	0.342640	0.285195	0.358803	0.444373	0.30
7	8	0.671699	0.424647	0.243902	0.337477	0.261874	0.359155	0.494865	0.30
8	9	0.748390	0.431454	0.206675	0.331063	0.231921	0.359859	0.567822	0.31
9	10	0.802779	0.452450	0.172229	0.319897	0.206247	0.354577	0.624947	0.32
10	11	0.827514	0.478482	0.162174	0.316375	0.184852	0.344718	0.653616	0.34
11	12	0.852215	0.482544	0.148909	0.315373	0.176080	0.341901	0.675866	0.34
12	13	0.853393	0.500787	0.148695	0.310600	0.171801	0.338732	0.682499	0.35
13	14	0.888592	0.495306	0.124305	0.311729	0.149551	0.331338	0.726573	0.35
14	15	0.888900	0.507535	0.125160	0.309632	0.144202	0.323592	0.730852	0.36
15	16	0.916786	0.509784	0.111254	0.307851	0.124947	0.320423	0.765297	0.37
16	17	0.915468	0.515020	0.111040	0.306304	0.125374	0.325000	0.766795	0.37
17	18	0.921738	0.518811	0.108686	0.311649	0.115319	0.321127	0.777920	0.37
18	19	0.927884	0.520889	0.101840	0.310997	0.114677	0.314789	0.785195	0.37
19	20	0.930523	0.524376	0.100984	0.308462	0.110398	0.316197	0.790971	0.37
20	21	0.934522	0.519583	0.095208	0.309827	0.110826	0.322535	0.795250	0.37
21	22	0.935572	0.520680	0.094352	0.308919	0.108045	0.321479	0.798674	0.37
22	23	0.936765	0.523195	0.092426	0.310506	0.107189	0.320775	0.802525	0.38
23	24	0.935825	0.522681	0.093710	0.312812	0.103552	0.315493	0.803808	0.37





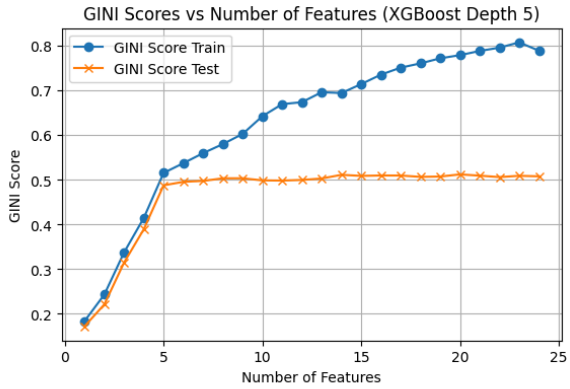


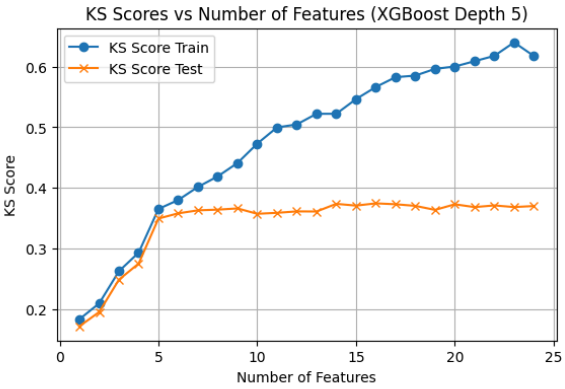


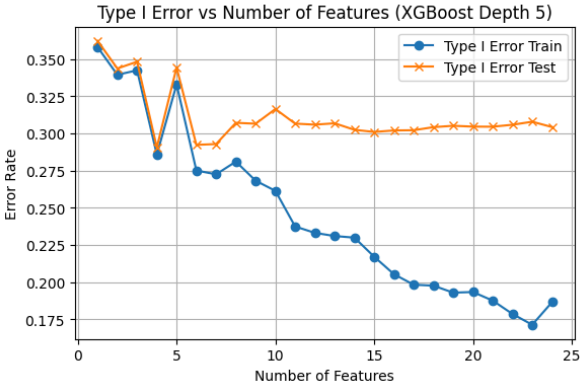
estymator XGBoost

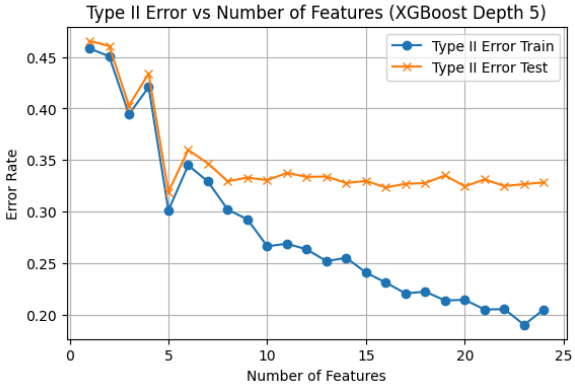
In [99]:
```
from xgboost import XGBClassifier

models_xgb = [
#     ('XGBoost Default', XGBClassifier(use_label_encoder=False, eval_metric=
    ('XGBoost Depth 5', XGBClassifier(max_depth=5, use_label_encoder=False,
    ('XGBoost n_estimators=200', XGBClassifier(n_estimators=200, use_label_e
    ('XGBoost learning_rate=0.1', XGBClassifier(learning_rate=0.1, use_label
    ('XGBoost colsample_bytree=0.8', XGBClassifier(colsample_bytree=0.8, use
]

for name, model in models_xgb:
    results = calculate_scores(model, X_train_balanced.values, y_train_balan
    plot_results(results, name, max_features)
```

	Number of Features	GINI Score Train	GINI Score Test	Type I Error Train	Type I Error Test	Type II Error Train	Type II Error Test	KS Score Train	KS
0	1	0.183783	0.172150	0.358151	0.362357	0.458066	0.465493	0.183783	0.1
1	2	0.243229	0.221147	0.339538	0.343762	0.450578	0.460563	0.209884	0.1
2	3	0.337693	0.314922	0.342533	0.348313	0.394523	0.402817	0.262944	0.2
3	4	0.415071	0.389660	0.285837	0.290668	0.420625	0.433803	0.293539	0.2
4	5	0.515731	0.487963	0.333119	0.344448	0.301455	0.319366	0.365426	0.3
5	6	0.537549	0.495756	0.275139	0.292395	0.345101	0.359859	0.380188	0.3
6	7	0.560043	0.497834	0.272572	0.292933	0.329268	0.346831	0.401797	0.3
7	8	0.580309	0.503688	0.281130	0.307219	0.302097	0.329577	0.419127	0.3
8	9	0.603088	0.503720	0.268293	0.306627	0.292683	0.333099	0.441164	0.3
9	10	0.642791	0.499041	0.261446	0.316496	0.266581	0.330634	0.473042	0.3
10	11	0.669795	0.498570	0.237484	0.306681	0.269148	0.337676	0.499786	0.3
11	12	0.674245	0.500041	0.233205	0.305988	0.263586	0.333803	0.504707	0.3
12	13	0.696529	0.503237	0.231065	0.306903	0.252246	0.334155	0.522251	0.3
13	14	0.694712	0.511459	0.229996	0.302533	0.255456	0.327817	0.522251	0.3
14	15	0.714677	0.509132	0.217159	0.301068	0.241121	0.329930	0.546427	0.3
15	16	0.735938	0.509996	0.205392	0.302096	0.231493	0.323592	0.566324	0.3
16	17	0.751585	0.509896	0.198331	0.302237	0.221010	0.327113	0.582585	0.3
17	18	0.761132	0.506908	0.197689	0.304308	0.222721	0.327817	0.585152	0.3
18	19	0.772479	0.507655	0.192982	0.305229	0.214163	0.335211	0.596277	0.3
19	20	0.779586	0.512419	0.193410	0.304664	0.214805	0.324648	0.600128	0.3
20	21	0.788987	0.509646	0.187634	0.304610	0.205392	0.331338	0.608900	0.3
21	22	0.796357	0.506355	0.178648	0.305928	0.205819	0.325000	0.617672	0.3
22	23	0.808215	0.509308	0.171374	0.307958	0.190629	0.326761	0.639923	0.3
23	24	0.789473	0.508079	0.186564	0.304375	0.205392	0.328521	0.617458	0.3

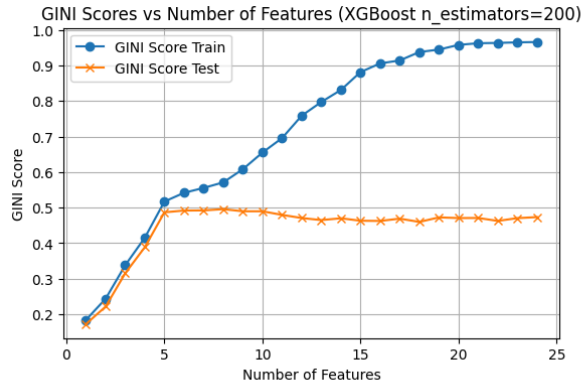

GINI Scores vs Number of Features (XGBoost Depth 5)
GINI Score Train
GINI Score Test
GINI Score
Number of Features

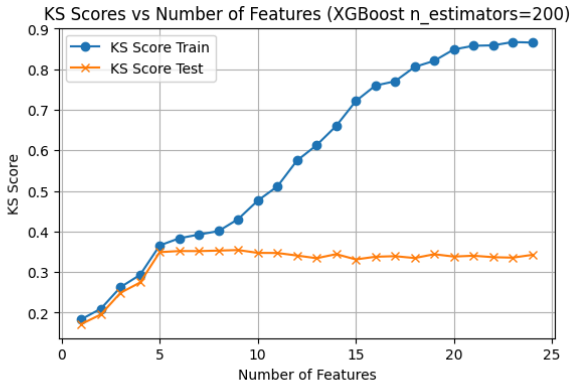

KS Scores vs Number of Features (XGBoost Depth 5)
KS Score Train
KS Score Test
KS Score
Number of Features

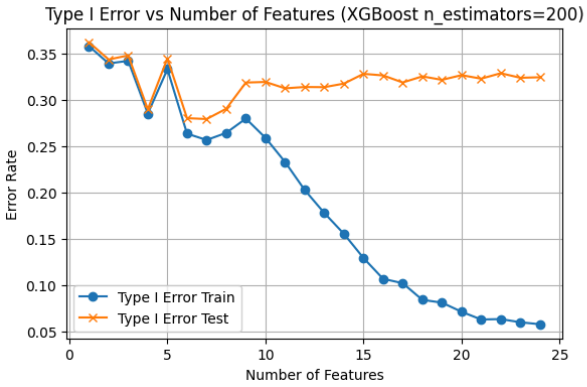

Type I Error vs Number of Features (XGBoost Depth 5)
Type I Error Train
Type I Error Test
Error Rate
Number of Features

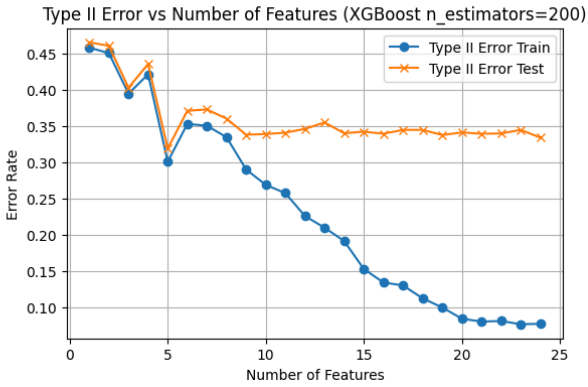

Type II Error vs Number of Features (XGBoost Depth 5)
Type II Error Train
Type II Error Test
Error Rate
Number of Features

	Number of Features	GINI Score Train	GINI Score Test	Type I Error Train	Type I Error Test	Type II Error Train	Type II Error Test	KS Score Train	KS
0	1	0.183783	0.172150	0.358151	0.362357	0.458066	0.465493	0.183783	0.1
1	2	0.243229	0.221147	0.339538	0.343762	0.450578	0.460563	0.209884	0.19
2	3	0.337694	0.314904	0.342319	0.348172	0.394737	0.402817	0.262944	0.2
3	4	0.415235	0.388542	0.285409	0.290076	0.421267	0.435563	0.293325	0.27
4	5	0.517137	0.486726	0.333333	0.344966	0.300813	0.319014	0.365854	0.3
5	6	0.541683	0.491716	0.263800	0.280503	0.353017	0.371127	0.383398	0.35
6	7	0.555480	0.492111	0.256953	0.279576	0.350449	0.372887	0.392597	0.35
7	8	0.570988	0.495619	0.264656	0.290641	0.334617	0.360211	0.401155	0.35
8	9	0.608089	0.489674	0.280060	0.318916	0.290116	0.338028	0.430680	0.35
9	10	0.655073	0.489637	0.259307	0.319561	0.268935	0.339085	0.476252	0.34
10	11	0.695858	0.479465	0.233205	0.312697	0.257595	0.340845	0.511125	0.3
11	12	0.758876	0.470688	0.203038	0.314129	0.225931	0.346127	0.574882	0.34
12	13	0.797502	0.464846	0.178220	0.313955	0.209884	0.354930	0.612751	0.33
13	14	0.831320	0.469947	0.155755	0.317780	0.191485	0.340141	0.660248	0.34
14	15	0.881728	0.463033	0.129653	0.328314	0.152974	0.342254	0.721438	0.3
15	16	0.906114	0.462430	0.107189	0.326620	0.134360	0.339437	0.759735	0.33
16	17	0.914504	0.468936	0.102482	0.319003	0.130295	0.344718	0.770218	0.33
17	18	0.938162	0.459686	0.084510	0.325578	0.112537	0.344718	0.805520	0.33
18	19	0.945150	0.472444	0.081515	0.321833	0.099914	0.337676	0.820710	0.34
19	20	0.958304	0.470608	0.071673	0.326895	0.084510	0.341197	0.848738	0.3
20	21	0.962978	0.470827	0.063329	0.323111	0.080445	0.339437	0.857724	0.34
21	22	0.963865	0.462530	0.063757	0.328966	0.081301	0.339789	0.858793	0.33
22	23	0.965465	0.470365	0.060334	0.324059	0.076808	0.344718	0.866923	0.33
23	24	0.966331	0.473222	0.058194	0.324630	0.077450	0.334155	0.865640	0.34

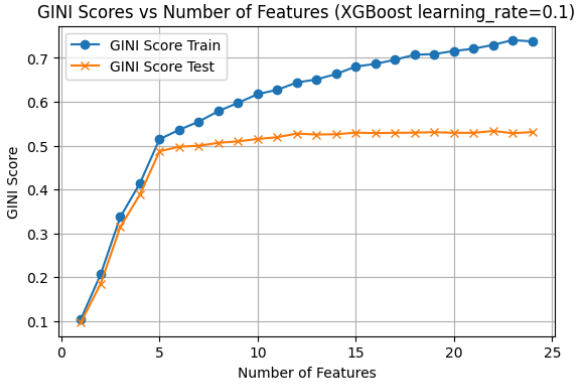

GINI Scores vs Number of Features (XGBoost n_estimators=200)
GINI Score Train
GINI Score Test
GINI Score
Number of Features

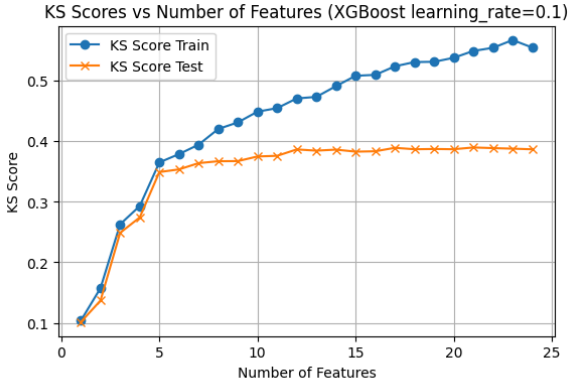

KS Scores vs Number of Features (XGBoost n_estimators=200)
KS Score Train
KS Score Test
KS Score
Number of Features

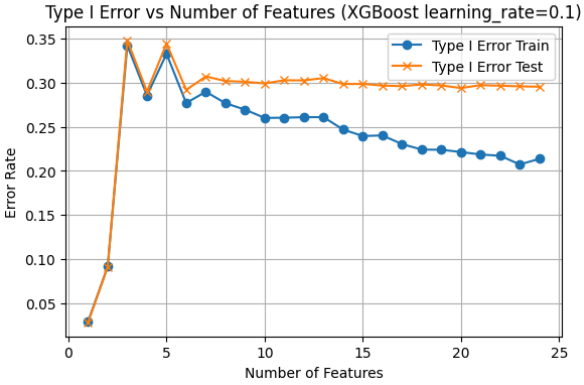

Type I Error vs Number of Features (XGBoost n_estimators=200)
Type I Error Train
Type I Error Test
Error Rate
Number of Features

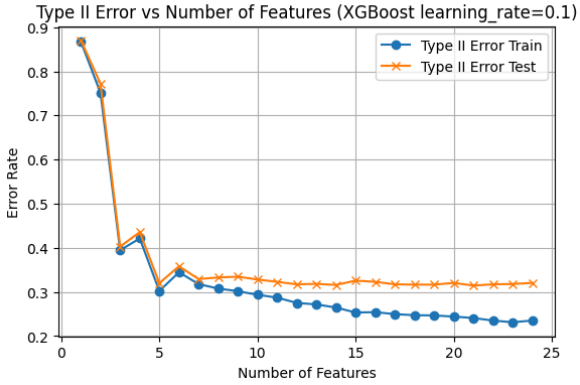

Type II Error vs Number of Features (XGBoost n_estimators=200)
Type II Error Train
Type II Error Test
Error Rate
Number of Features

	Number of Features	GINI Score Train	GINI Score Test	Type I Error Train	Type I Error Test	Type II Error Train	Type II Error Test	KS Score Train	KS
0	1	0.105424	0.098180	0.028883	0.028147	0.867351	0.870423	0.103766	0.1
1	2	0.207372	0.185687	0.091356	0.090888	0.751177	0.772183	0.157467	0.13
2	3	0.337471	0.314746	0.342533	0.348313	0.394523	0.402817	0.262944	0.24
3	4	0.414253	0.388819	0.285623	0.290170	0.421267	0.435563	0.293111	0.27
4	5	0.514805	0.487311	0.332905	0.344408	0.302097	0.320070	0.365212	0.3
5	6	0.536010	0.497459	0.277065	0.291999	0.344673	0.358099	0.378905	0.35
6	7	0.554898	0.499930	0.289902	0.307232	0.317715	0.329225	0.394523	0.36
7	8	0.578680	0.506423	0.276851	0.302036	0.307659	0.332746	0.420411	0.3
8	9	0.597672	0.509792	0.269576	0.301222	0.301669	0.334507	0.431322	0.36
9	10	0.617394	0.515177	0.260163	0.299300	0.293539	0.328169	0.449080	0.37
10	11	0.627377	0.519257	0.260377	0.302916	0.287120	0.322887	0.454857	0.37
11	12	0.643886	0.527536	0.261018	0.302755	0.275139	0.317254	0.470903	0.38
12	13	0.651011	0.525334	0.261018	0.305377	0.271716	0.318310	0.473470	0.3
13	14	0.663234	0.526127	0.246898	0.298506	0.264656	0.315845	0.491228	0.3
14	15	0.680330	0.529605	0.239623	0.298842	0.253316	0.325704	0.508344	0.38
15	16	0.686314	0.528569	0.240265	0.296879	0.254172	0.322887	0.509842	0.38
16	17	0.695797	0.529315	0.230424	0.296274	0.249679	0.317254	0.523962	0.38
17	18	0.707102	0.529662	0.224219	0.298150	0.247326	0.316549	0.531023	0.3
18	19	0.708670	0.530757	0.224005	0.297034	0.246684	0.316549	0.531451	0.38
19	20	0.715784	0.529284	0.221438	0.294069	0.244116	0.320423	0.538083	0.38
20	21	0.720840	0.529272	0.218656	0.297384	0.240907	0.314437	0.549422	0.38
21	22	0.729660	0.533430	0.217159	0.296758	0.235131	0.317254	0.554557	0.3
22	23	0.740662	0.528298	0.207317	0.295992	0.231279	0.317958	0.567394	0.38
23	24	0.737469	0.531078	0.213736	0.295569	0.235131	0.320423	0.554985	0.38

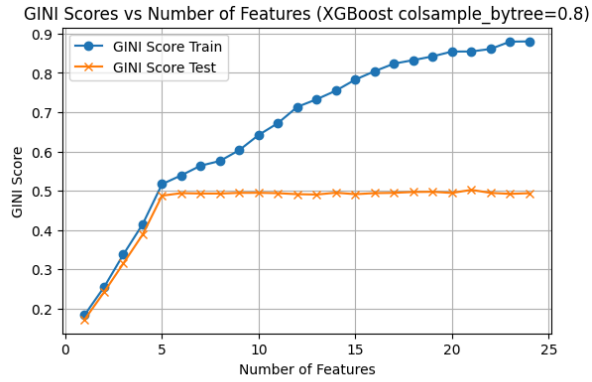

GINI Scores vs Number of Features (XGBoost learning_rate=0.1)
GINI Score Train
GINI Score Test
GINI Score
Number of Features

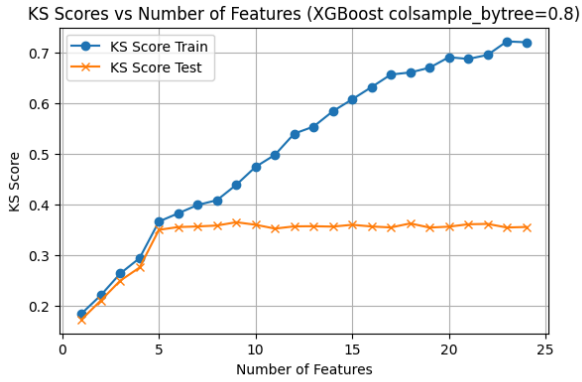

KS Scores vs Number of Features (XGBoost learning_rate=0.1)
KS Score Train
KS Score Test
KS Score
Number of Features

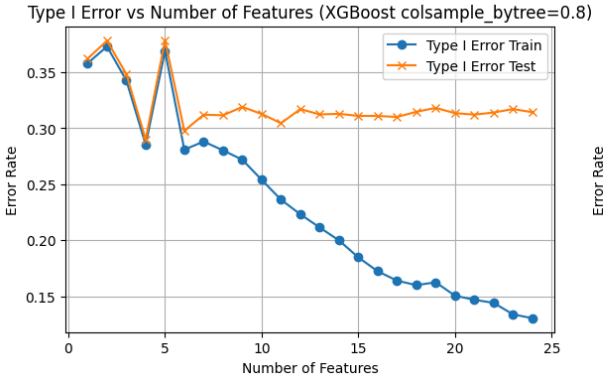

Type I Error vs Number of Features (XGBoost learning_rate=0.1)
Type I Error Train
Type I Error Test
Error Rate
Number of Features

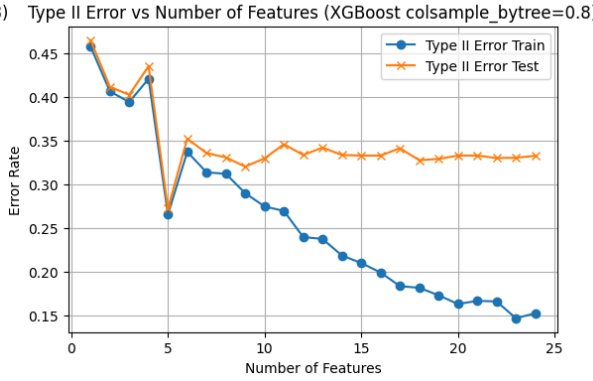

Type II Error vs Number of Features (XGBoost learning_rate=0.1)
Type II Error Train
Type II Error Test
Error Rate
Number of Features

	Number of Features	GINI Score Train	GINI Score Test	Type I Error Train	Type I Error Test	Type II Error Train	Type II Error Test	KS Score Train	KS
0	1	0.183783	0.172150	0.358151	0.362357	0.458066	0.465493	0.183783	0.1
1	2	0.254154	0.241409	0.373128	0.378262	0.406718	0.411972	0.220154	0.2
2	3	0.337469	0.314744	0.342747	0.348401	0.394523	0.402817	0.262944	0.2
3	4	0.414893	0.388487	0.285409	0.290036	0.421267	0.435563	0.293325	0.2
4	5	0.516698	0.487369	0.369063	0.378766	0.265511	0.272183	0.365640	0.3
5	6	0.539260	0.493878	0.280916	0.297821	0.337398	0.352465	0.381900	0.3
6	7	0.563824	0.492916	0.288190	0.312140	0.313864	0.335915	0.398374	0.3
7	8	0.575776	0.492782	0.280274	0.311608	0.312152	0.330986	0.407788	0.3
8	9	0.603480	0.494620	0.271930	0.319077	0.289902	0.320423	0.438169	0.3
9	10	0.641985	0.494729	0.253958	0.312825	0.274711	0.329930	0.473684	0.3
10	11	0.672301	0.493743	0.236200	0.304563	0.269576	0.346127	0.497433	0.3
11	12	0.713551	0.491188	0.223149	0.317141	0.239623	0.333803	0.539367	0.3
12	13	0.732810	0.490293	0.211382	0.312415	0.237484	0.342254	0.553273	0.3
13	14	0.754942	0.494987	0.199829	0.312933	0.218442	0.333803	0.583654	0.3
14	15	0.783022	0.491184	0.184638	0.311064	0.209884	0.333099	0.607189	0.3
15	16	0.804441	0.493914	0.172229	0.311044	0.198973	0.333099	0.631365	0.3
16	17	0.823670	0.494596	0.163885	0.310136	0.183355	0.341549	0.656183	0.3
17	18	0.832617	0.496733	0.159820	0.314721	0.181429	0.327817	0.660248	0.3
18	19	0.842190	0.497279	0.162388	0.318237	0.172871	0.329225	0.669662	0.3
19	20	0.854456	0.494329	0.150407	0.313538	0.162816	0.333099	0.690415	0.3
20	21	0.854586	0.502168	0.146769	0.312220	0.166453	0.333099	0.686992	0.3
21	22	0.861106	0.494449	0.143988	0.314096	0.165811	0.330282	0.694908	0.3
22	23	0.879670	0.492055	0.133718	0.317181	0.146555	0.330634	0.721652	0.3
23	24	0.880293	0.493639	0.130295	0.314392	0.152118	0.332746	0.719940	0.3







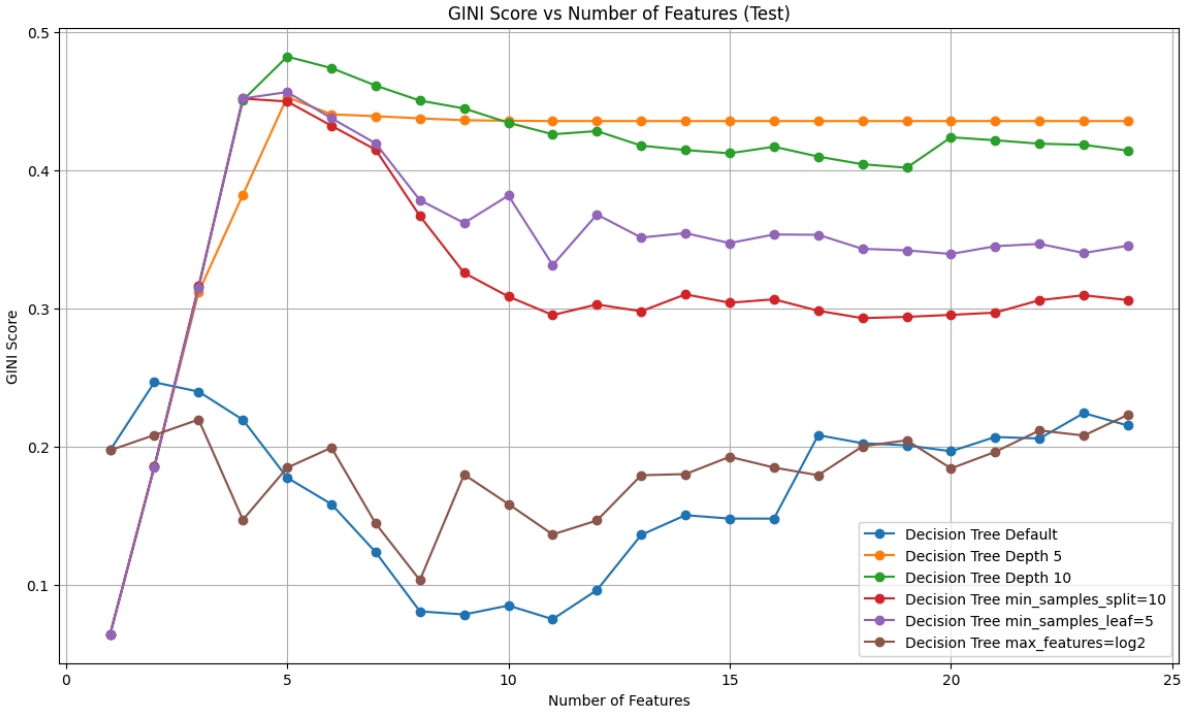


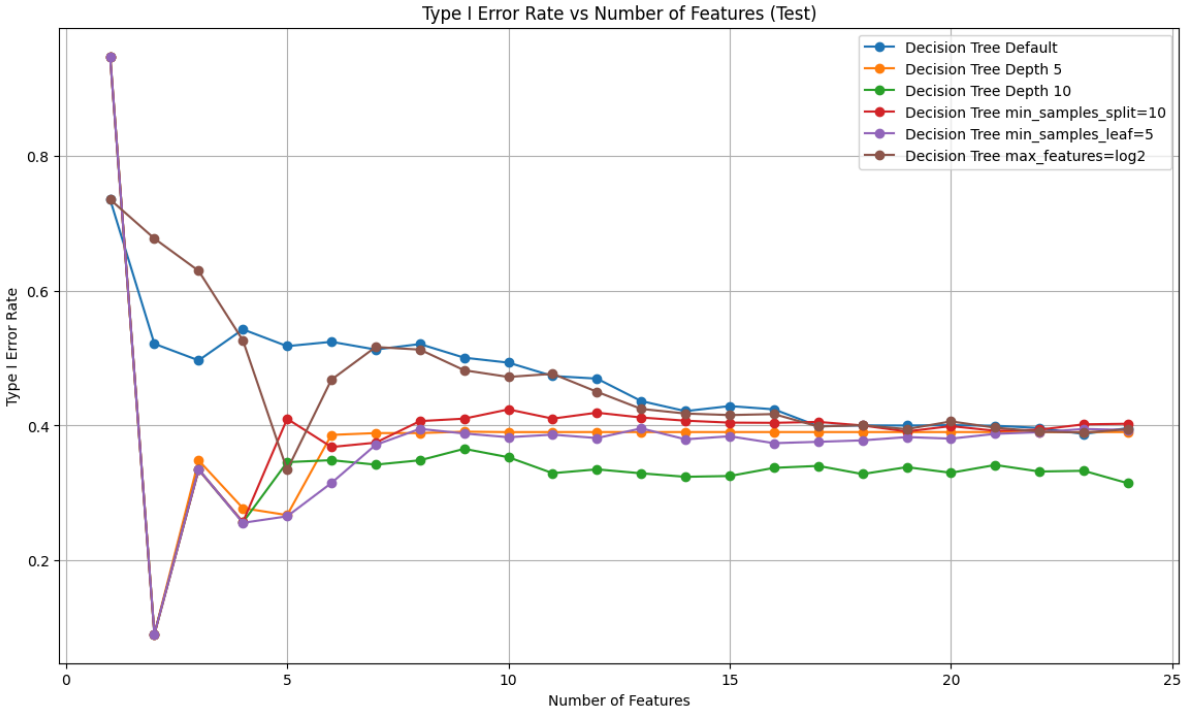
Model comparison within model type

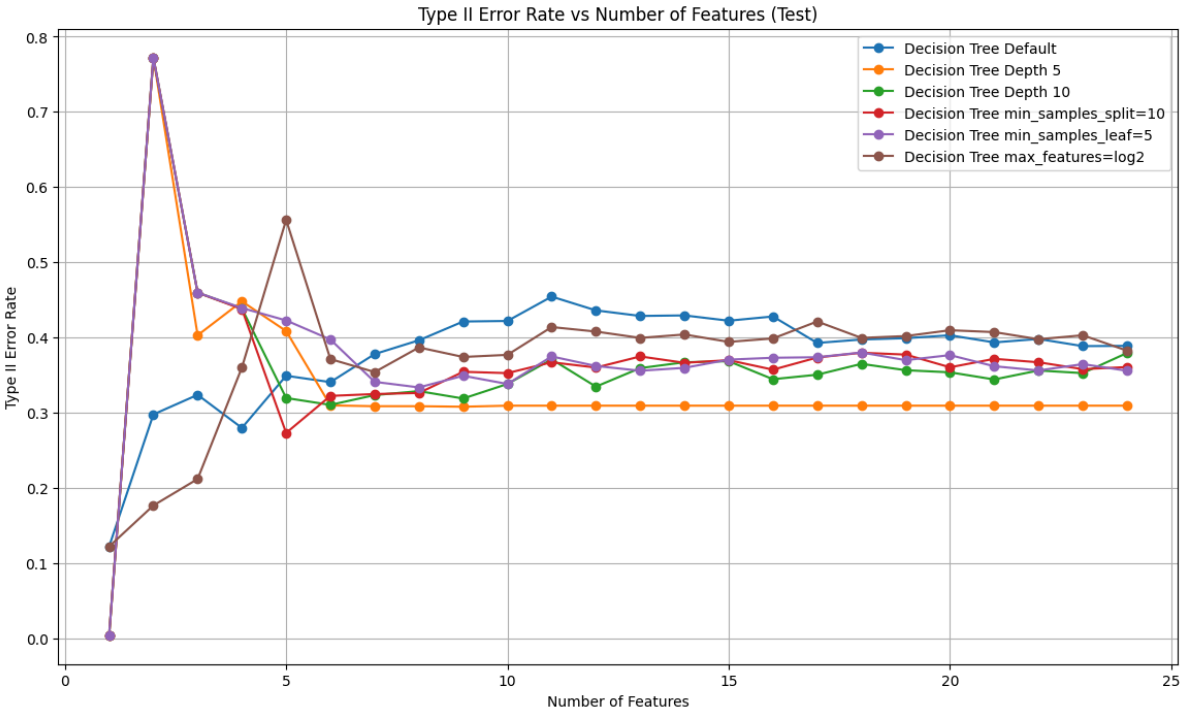
Drzewo

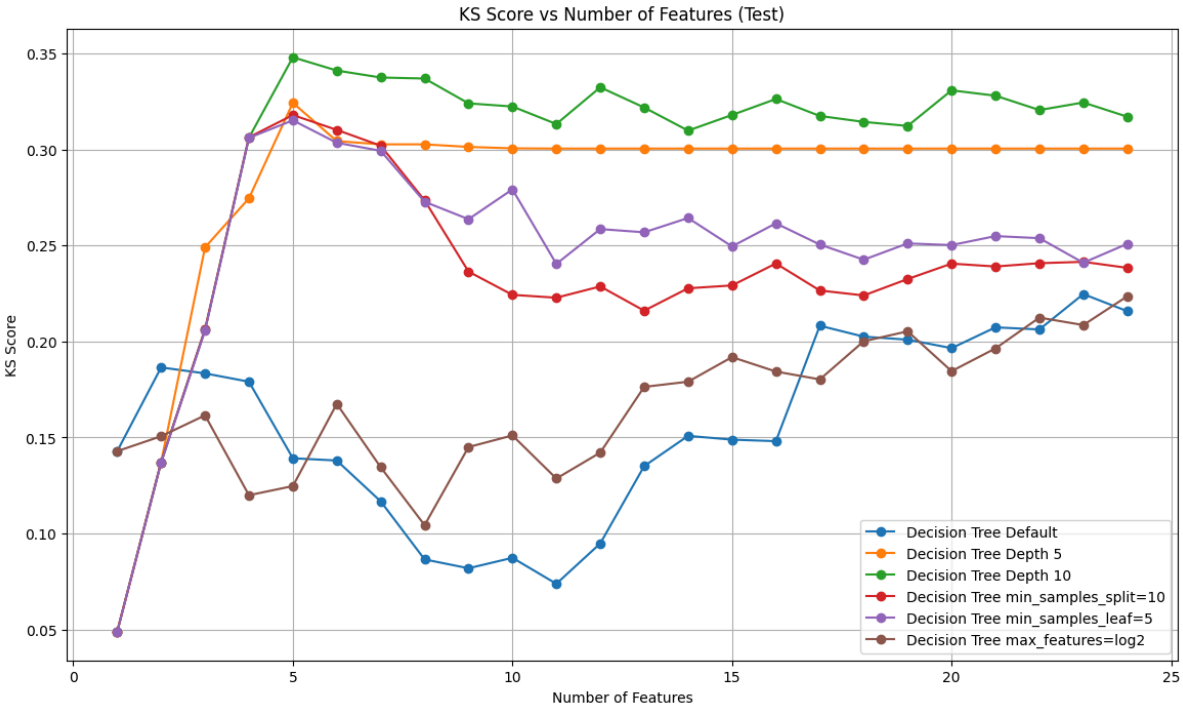
In [69]:
```
from sklearn.tree import DecisionTreeClassifier

models = [
    ('Decision Tree Default', DecisionTreeClassifier()),
    ('Decision Tree Depth 5', DecisionTreeClassifier(max_depth=5)),
    ('Decision Tree Depth 10', DecisionTreeClassifier(max_depth=10)),
    ('Decision Tree min_samples_split=10', DecisionTreeClassifier(min_sample
    ('Decision Tree min_samples_leaf=5', DecisionTreeClassifier(min_samples_
    ('Decision Tree max_features=log2', DecisionTreeClassifier(max_features=
]

# Organizowanie wyników dla wszystkich modeli
labels = [
    'Decision Tree Default', 'Decision Tree Depth 5', 'Decision Tree Depth 1
    'Decision Tree min_samples_split=10', 'Decision Tree min_samples_leaf=5
    'Decision Tree max_features=log2'
]

all_results = []

for name, model in models:
    results = calculate_scores(model, X_train_balanced.values, y_train_balar
    all_results.append(results)

organized_results = organize_results(*all_results, labels=labels)

# Tworzenie wykresów dla każdej metryki dla zbioru testowego
plot_comparison('gini', organized_results, labels, 'GINI Score', dataset='te
plot_comparison('type_i_error', organized_results, labels, 'Type I Error Rat
plot_comparison('type_ii_error', organized_results, labels, 'Type II Error R
```

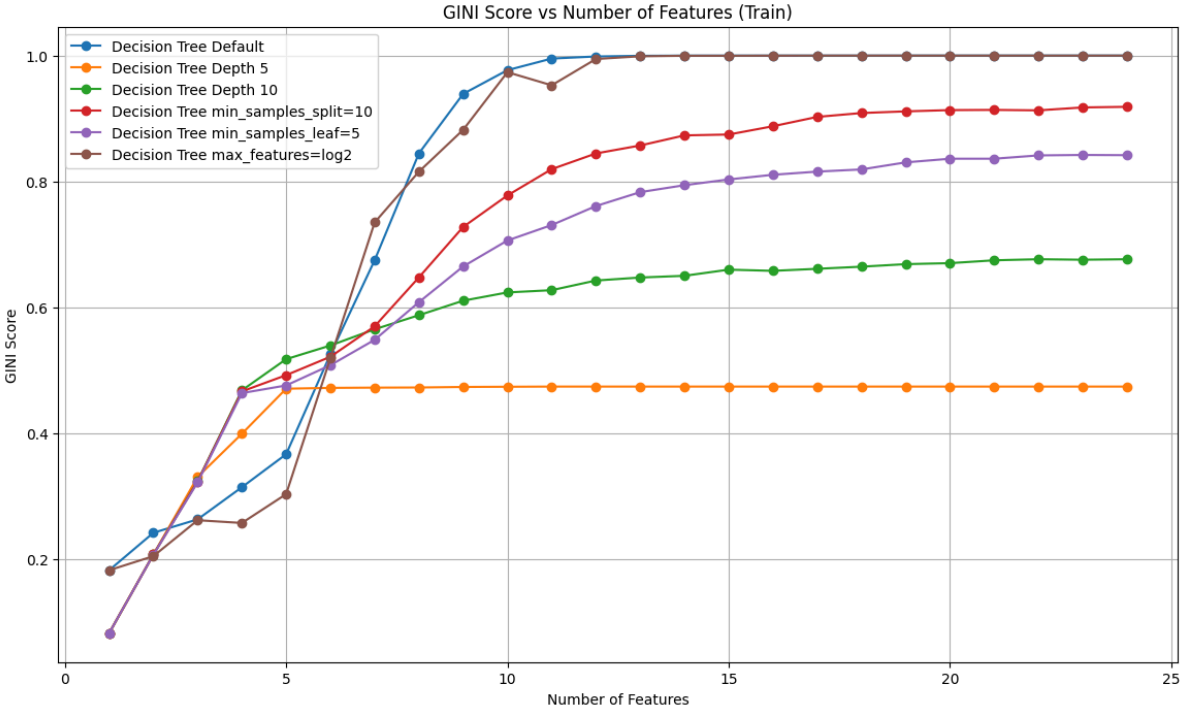
```
plot_comparison('ks', organized_results, labels, 'KS Score', dataset='test')

# Tworzenie wykresów dla każdej metryki dla zbioru treningowego
plot_comparison('gini', organized_results, labels, 'GINI Score', dataset='tr
plot_comparison('type_i_error', organized_results, labels, 'Type I Error Rat
plot_comparison('type_ii_error', organized_results, labels, 'Type II Error R
plot_comparison('ks', organized_results, labels, 'KS Score', dataset='train

# Generowanie tabeli wyników dla 5 i 10 cech
results_table_5_features = generate_results_table(organized_results, labels,
results_table_10_features = generate_results_table(organized_results, labels

# Wyświetlanie wyników w formie tabeli
print("Wyniki dla 5 cech:")
display(results_table_5_features)

print("Wyniki dla 10 cech:")
display(results_table_10_features)
```

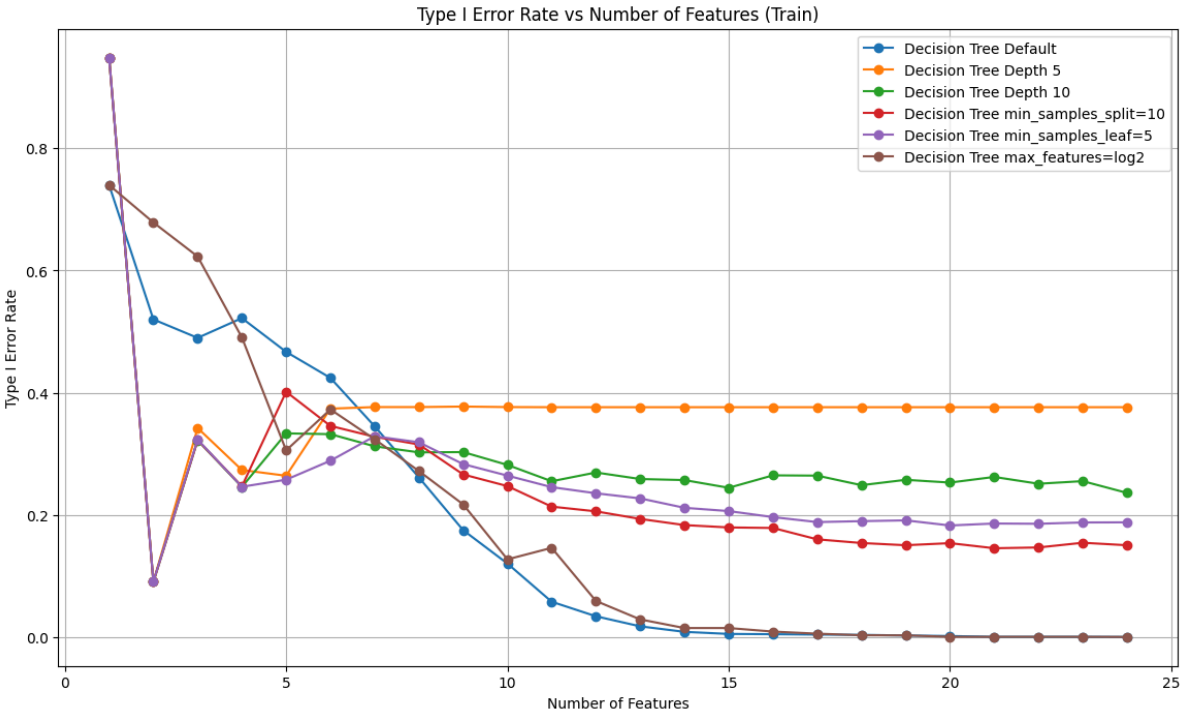


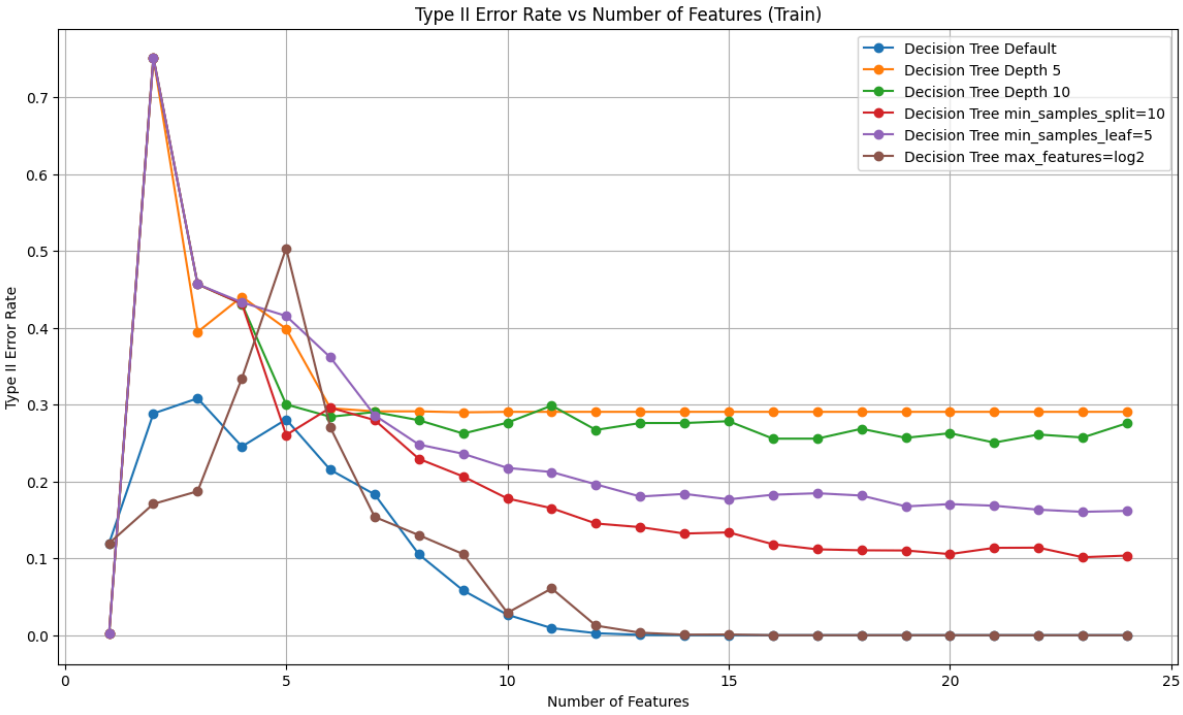


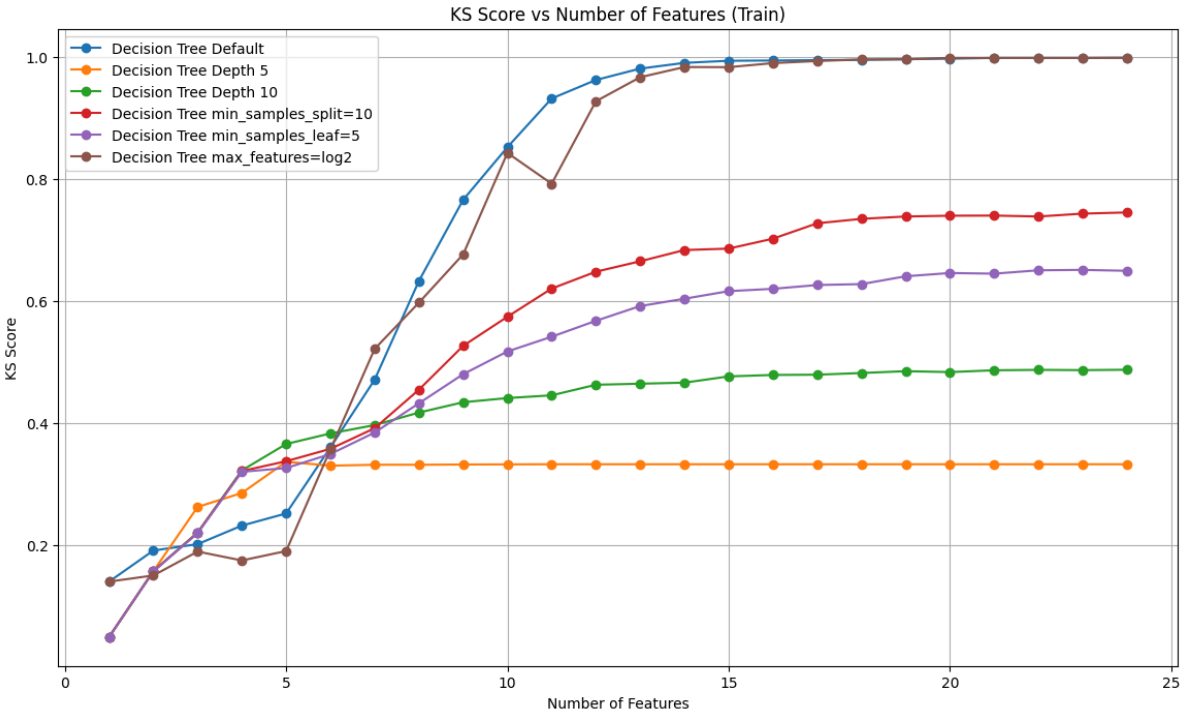

Type II Error Rate vs Number of Features (Test)
Type II Error Rate
Number of Features
Decision Tree Default
Decision Tree Depth 5
Decision Tree Depth 10
Decision Tree min_samples_split=10
Decision Tree min_samples_leaf=5
Decision Tree max_features=log2


KS Score vs Number of Features (Test)
KS Score
Number of Features
Decision Tree Default
Decision Tree Depth 5
Decision Tree Depth 10
Decision Tree min_samples_split=10
Decision Tree min_samples_leaf=5
Decision Tree max_features=log2


GINI Score vs Number of Features (Train)
Decision Tree Default
Decision Tree Depth 5
Decision Tree Depth 10
Decision Tree min_samples_split=10
Decision Tree min_samples_leaf=5
Decision Tree max_features=log2
GINI Score
Number of Features


Type I Error Rate vs Number of Features (Train)
Decision Tree Default
Decision Tree Depth 5
Decision Tree Depth 10
Decision Tree min_samples_split=10
Decision Tree min_samples_leaf=5
Decision Tree max_features=log2
Type I Error Rate
Number of Features





Wyniki dla 5 cech:

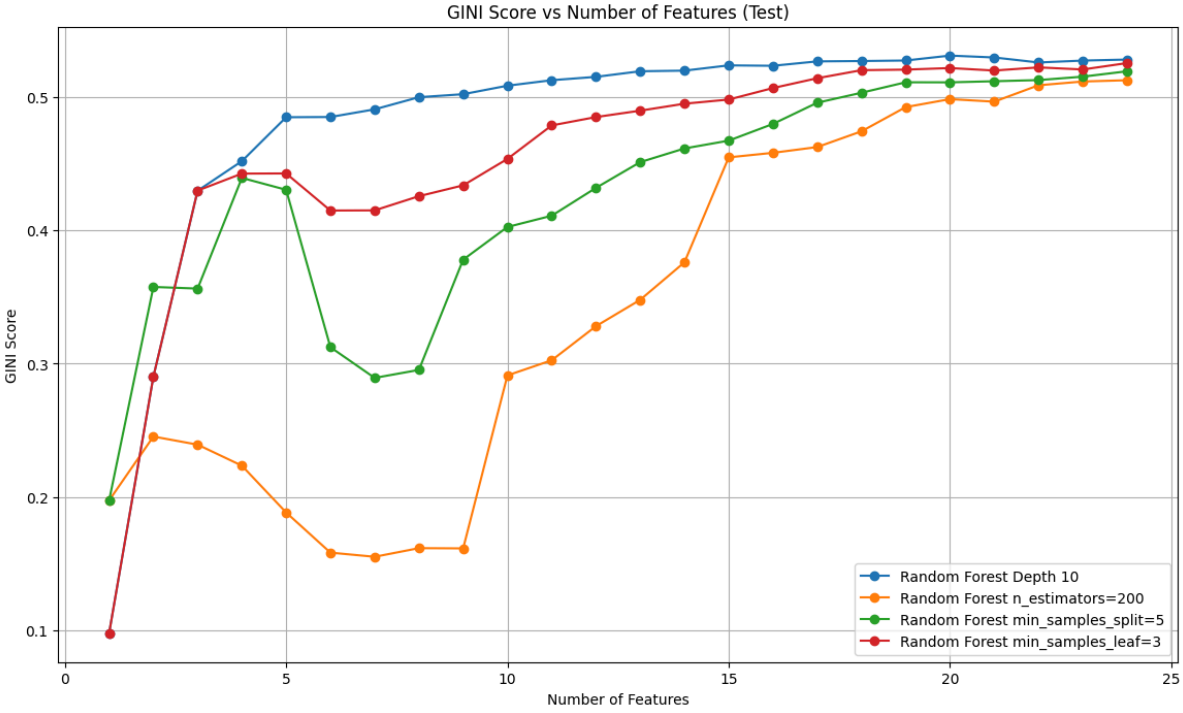
	Model	GINI Score Train	GINI Score Test	Type I Error Train	Type I Error Test	Type II Error Train	Type II Error Test	KS S
0	Decision Tree Default	0.366974	0.177819	0.467052	0.517794	0.280702	0.348944	0.25
1	Decision Tree Depth 5	0.471031	0.452973	0.264442	0.267273	0.398588	0.408451	0.33
2	Decision Tree Depth 10	0.517930	0.482318	0.333761	0.345786	0.300171	0.319366	0.36
3	Decision Tree min_samples_split=10	0.492513	0.449874	0.401797	0.410060	0.260377	0.272887	0.33
4	Decision Tree min_samples_leaf=5	0.476186	0.456656	0.258023	0.265438	0.415276	0.422535	0.32
5	Decision Tree max_features=log2	0.303595	0.185012	0.305948	0.334243	0.503423	0.556338	0.19

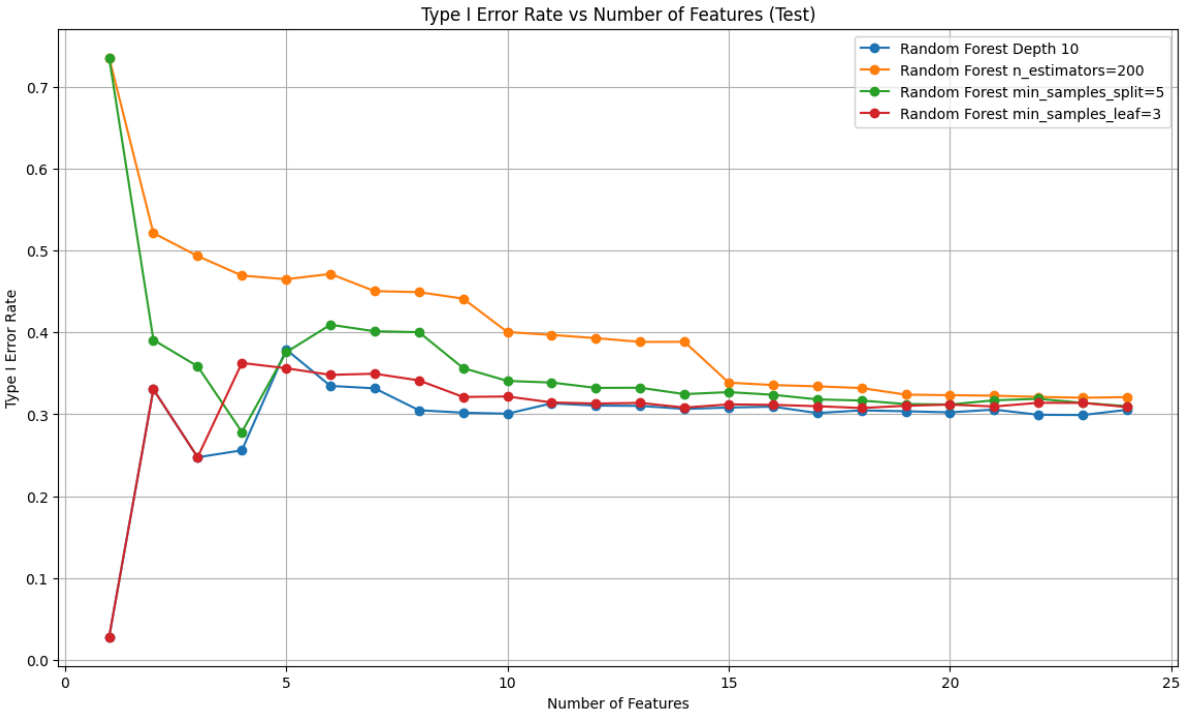
Wyniki dla 10 cech:

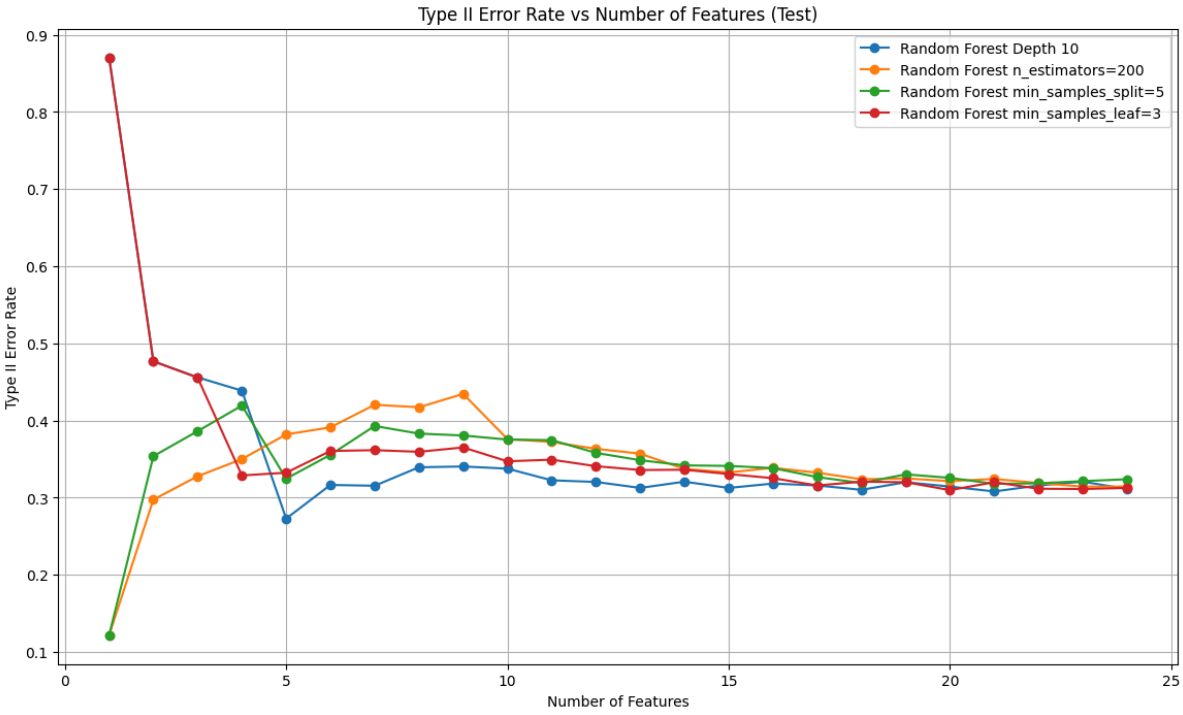
	Model	GINI Score Train	GINI Score Test	Type I Error Train	Type I Error Test	Type II Error Train	Type II Error Test	KS S
0	Decision Tree Default	0.976881	0.085337	0.120881	0.493345	0.026530	0.421831	0.85
1	Decision Tree Depth 5	0.474096	0.435942	0.376765	0.390316	0.290543	0.309155	0.33
2	Decision Tree Depth 10	0.623916	0.434482	0.282199	0.353059	0.276423	0.338028	0.44
3	Decision Tree min_samples_split=10	0.777913	0.308801	0.247540	0.423968	0.177792	0.352465	0.57
4	Decision Tree min_samples_leaf=5	0.706388	0.381807	0.264656	0.382766	0.217587	0.338028	0.51
5	Decision Tree max_features=log2	0.973665	0.158728	0.127728	0.472162	0.029097	0.376761	0.84

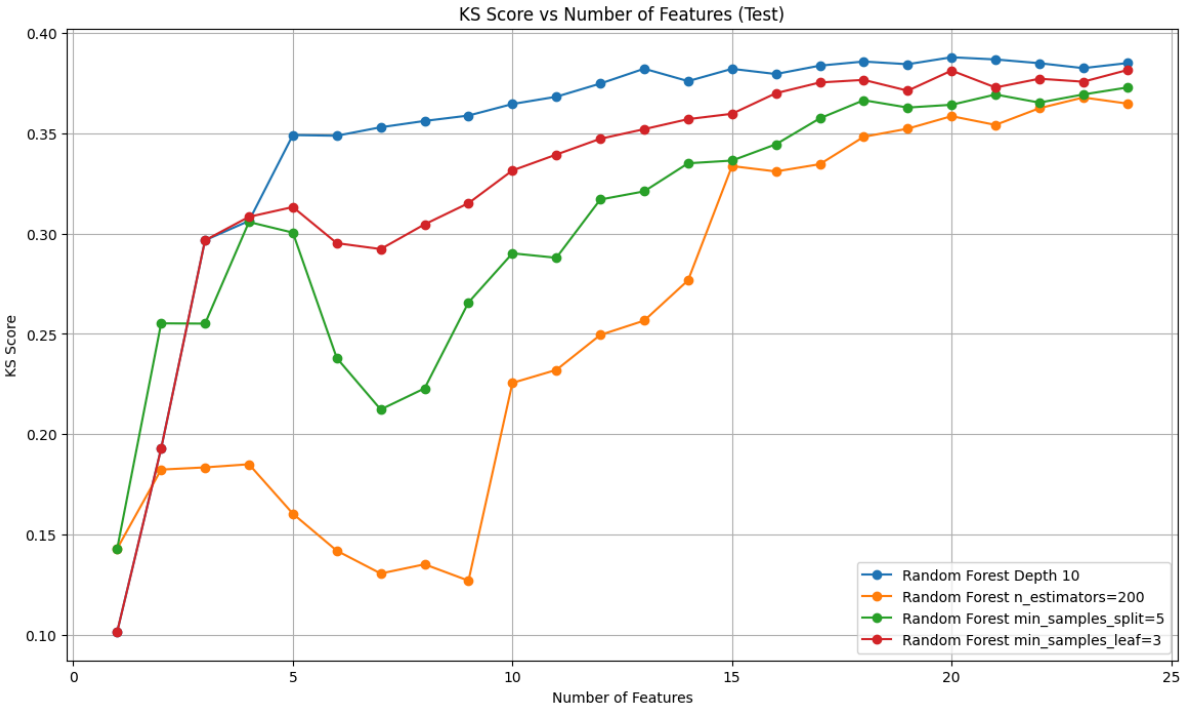
Las

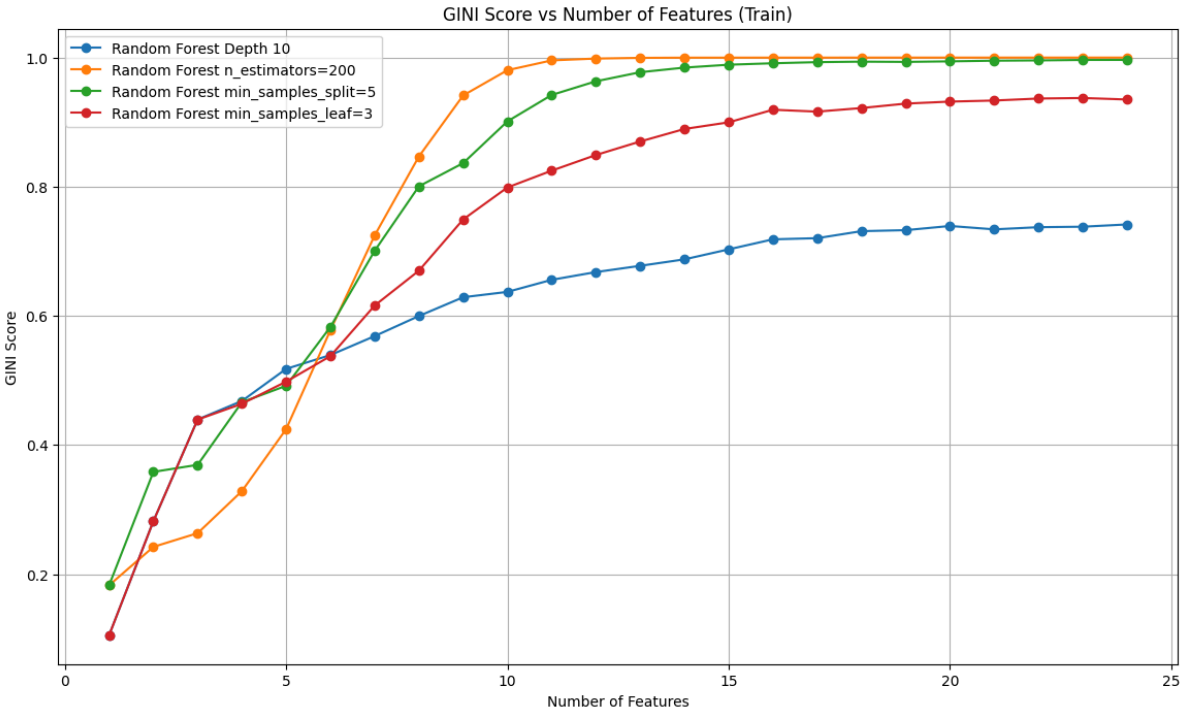
In [71]:
```
# Organizowanie wyników dla wszystkich modeli
labels = [
    'Random Forest Depth 10', 'Random Forest n_estimators=200',
    'Random Forest min_samples_split=5', 'Random Forest min_samples_leaf=3',
]

all_results = [
    results_rf_depth10, results_rf_estimators200,
    results_rf_min_samples_split5, results_rf_min_samples_leaf3,
]

organized_results = organize_results(*all_results, labels=labels)

# Tworzenie wykresów dla każdej metryki dla zbioru testowego
plot_comparison('gini', organized_results, labels, 'GINI Score', dataset='te
plot_comparison('type_i_error', organized_results, labels, 'Type I Error Rat
plot_comparison('type_ii_error', organized_results, labels, 'Type II Error R
plot_comparison('ks', organized_results, labels, 'KS Score', dataset='test')

# Tworzenie wykresów dla każdej metryki dla zbioru treningowego
plot_comparison('gini', organized_results, labels, 'GINI Score', dataset='tr
plot_comparison('type_i_error', organized_results, labels, 'Type I Error Rat
plot_comparison('type_ii_error', organized_results, labels, 'Type II Error R
plot_comparison('ks', organized_results, labels, 'KS Score', dataset='train

# Generowanie tabeli wyników dla 5 i 10 cech
results_table_5_features = generate_results_table(organized_results, labels,
results_table_10_features = generate_results_table(organized_results, labels

# Wyświetlanie wyników w formie tabeli
print("Wyniki dla 5 cech:")
display(results_table_5_features)

print("Wyniki dla 10 cech:")
display(results_table_10_features)
```

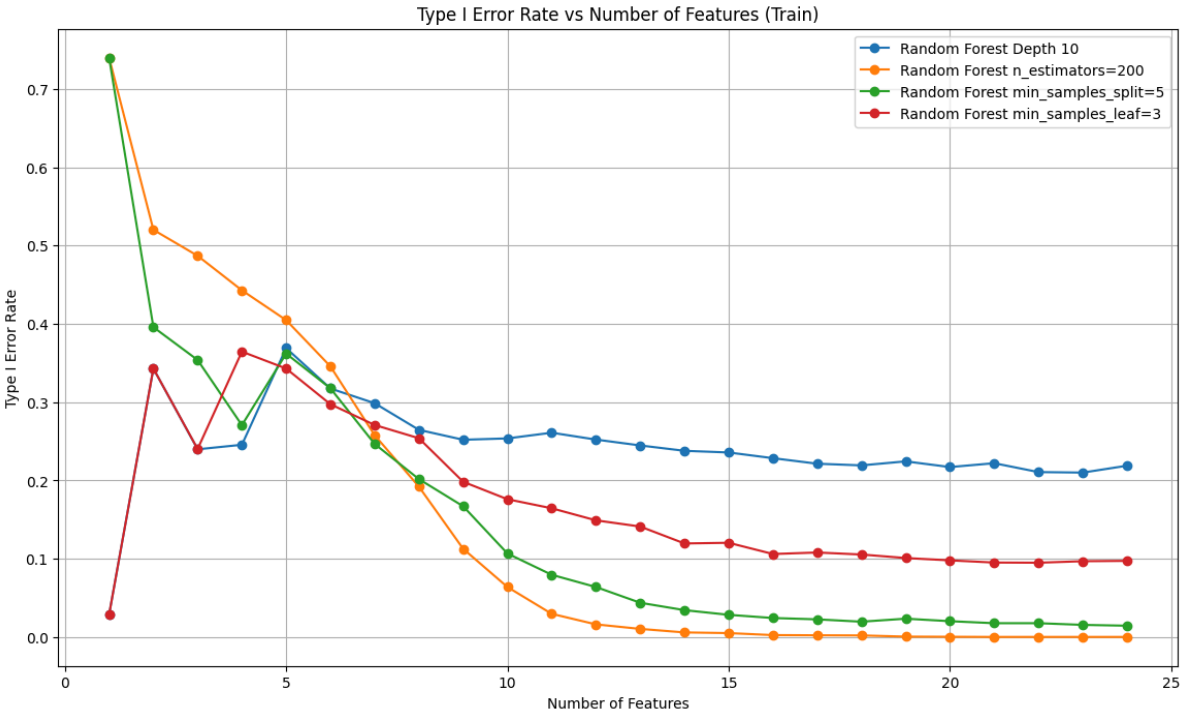

GINI Score vs Number of Features (Test)
GINI Score
Number of Features
Random Forest Depth 10
Random Forest n_estimators=200
Random Forest min_samples_split=5
Random Forest min_samples_leaf=3

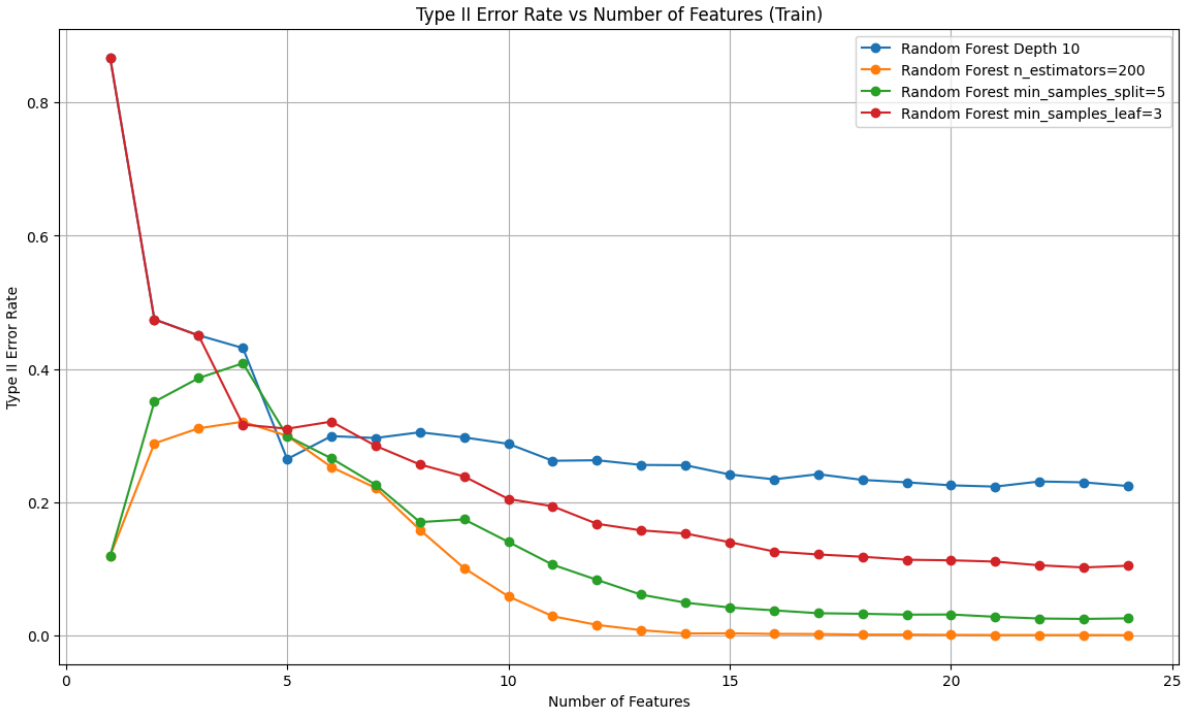

Type I Error Rate vs Number of Features (Test)
Type I Error Rate
Number of Features
Random Forest Depth 10
Random Forest n_estimators=200
Random Forest min_samples_split=5
Random Forest min_samples_leaf=3

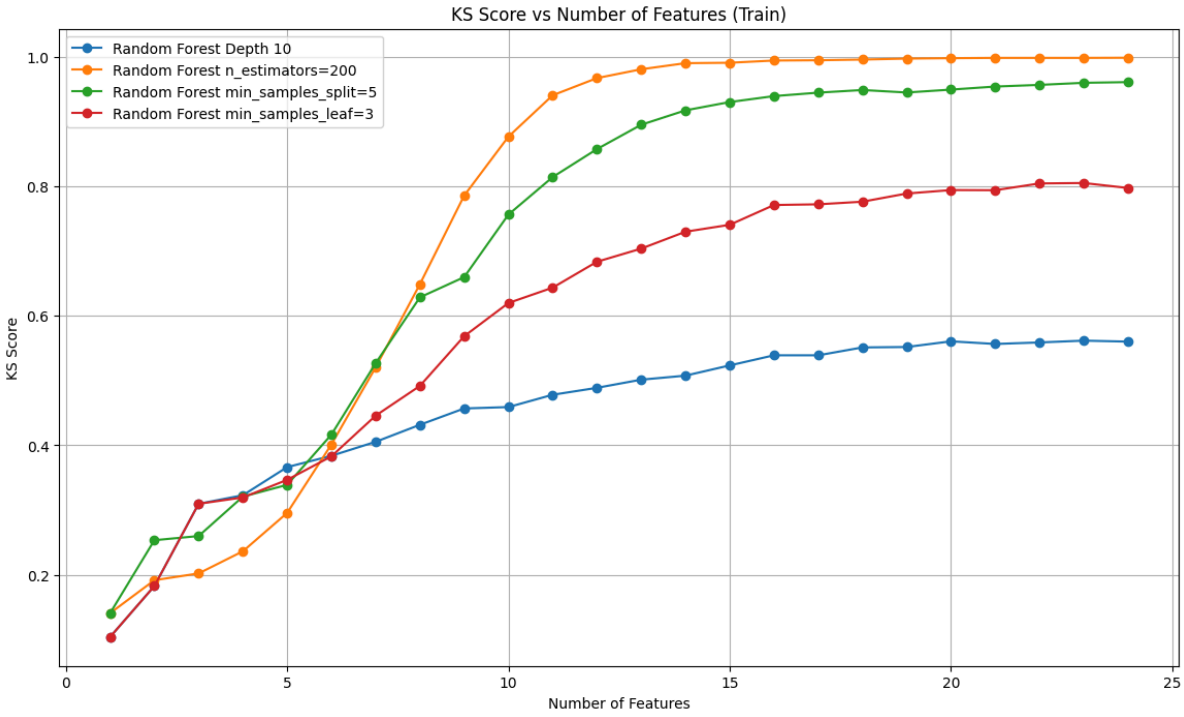

Type II Error Rate vs Number of Features (Test)
Type II Error Rate
Number of Features
Random Forest Depth 10
Random Forest n_estimators=200
Random Forest min_samples_split=5
Random Forest min_samples_leaf=3


KS Score vs Number of Features (Test)
KS Score
Number of Features
Random Forest Depth 10
Random Forest n_estimators=200
Random Forest min_samples_split=5
Random Forest min_samples_leaf=3


GINI Score vs Number of Features (Train)
Random Forest Depth 10
Random Forest n_estimators=200
Random Forest min_samples_split=5
Random Forest min_samples_leaf=3
GINI Score
Number of Features


Type I Error Rate vs Number of Features (Train)
Random Forest Depth 10
Random Forest n_estimators=200
Random Forest min_samples_split=5
Random Forest min_samples_leaf=3
Type I Error Rate
Number of Features





Wyniki dla 5 cech:

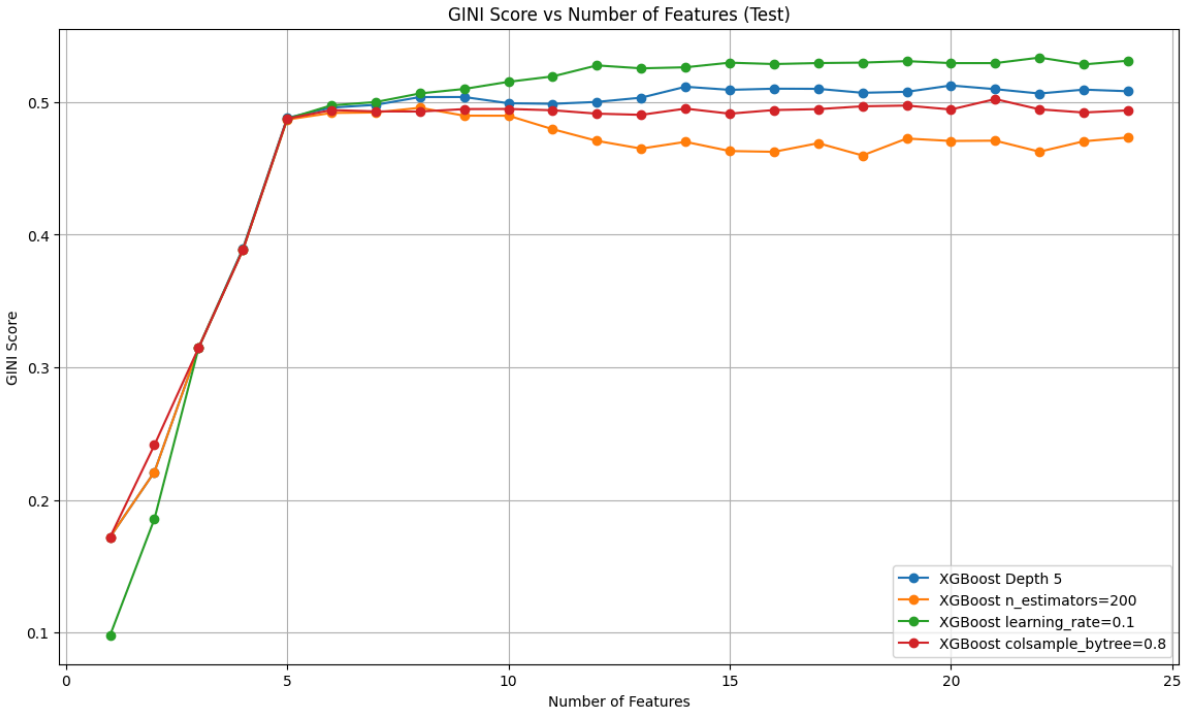
	Model	GINI Score Train	GINI Score Test	Type I Error Train	Type I Error Test	Type II Error Train	Type II Error Test	KS Sc T
0	Random Forest Depth 10	0.517877	0.484602	0.369063	0.379163	0.264869	0.272887	0.366
1	Random Forest n_estimators=200	0.424568	0.188553	0.404792	0.464976	0.299743	0.382042	0.295
2	Random Forest min_samples_split=5	0.491674	0.430251	0.362003	0.375761	0.299529	0.324648	0.338
3	Random Forest min_samples_leaf=3	0.498129	0.442502	0.342961	0.356246	0.310441	0.332394	0.346

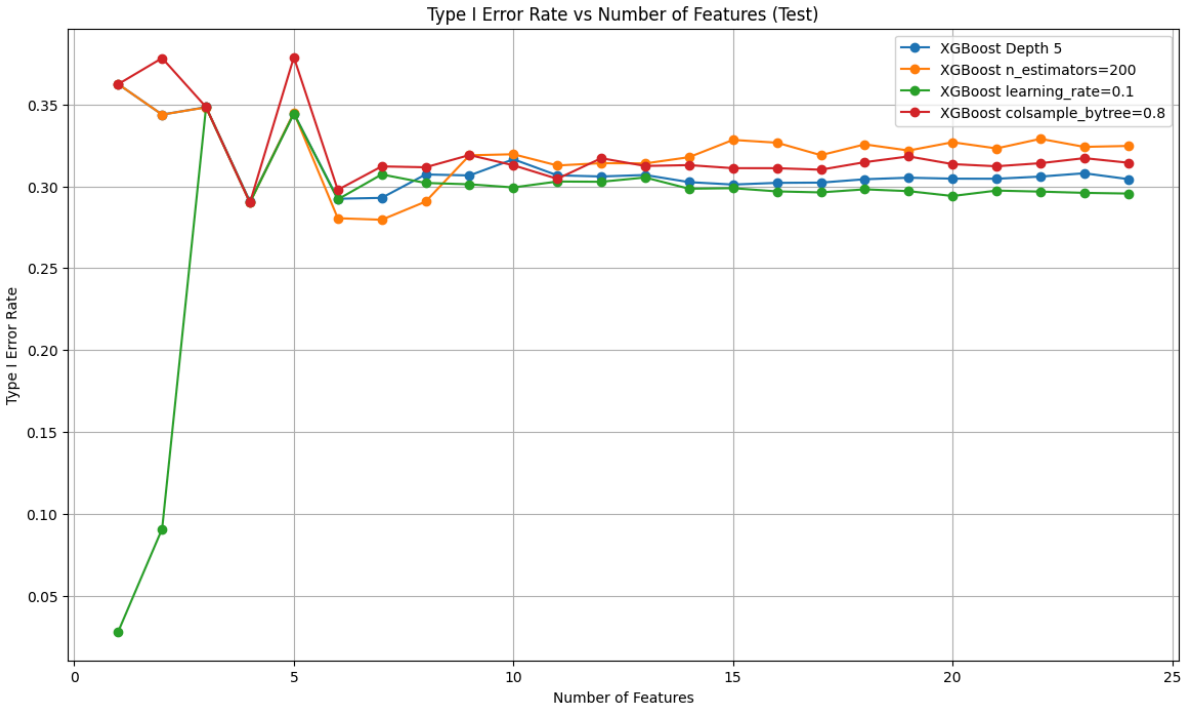
Wyniki dla 10 cech:

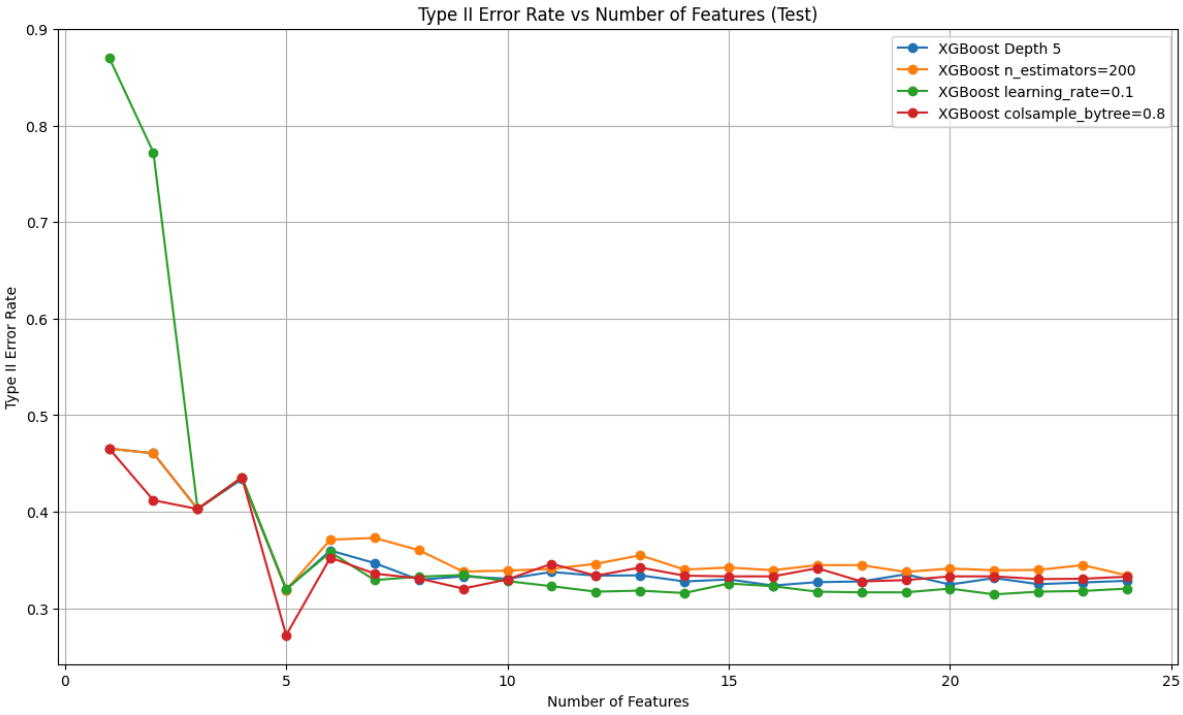
	Model	GINI Score Train	GINI Score Test	Type I Error Train	Type I Error Test	Type II Error Train	Type II Error Test	KS Sc Tr
0	Random Forest Depth 10	0.637125	0.508216	0.253744	0.300765	0.287762	0.337676	0.459
1	Random Forest n_estimators=200	0.980510	0.291243	0.063971	0.400473	0.058836	0.376056	0.877
2	Random Forest min_samples_split=5	0.901226	0.402405	0.106761	0.340751	0.140565	0.375352	0.756
3	Random Forest min_samples_leaf=3	0.798866	0.453384	0.176080	0.321766	0.204964	0.347183	0.620

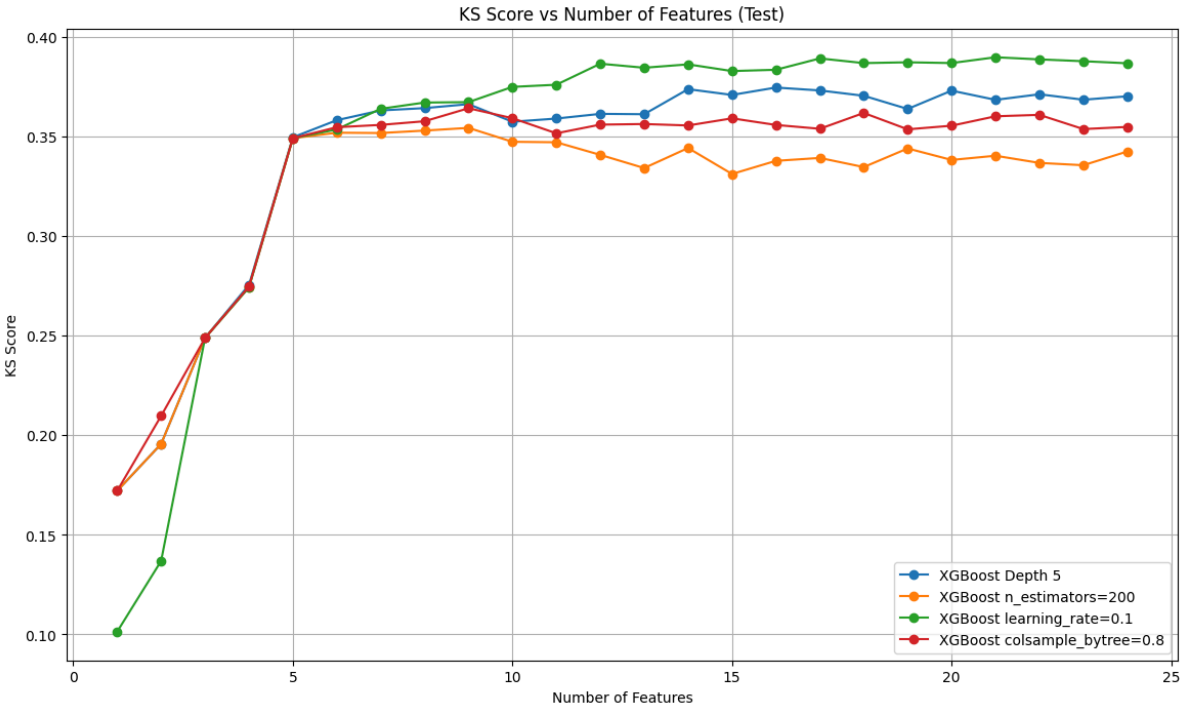
XGBoost

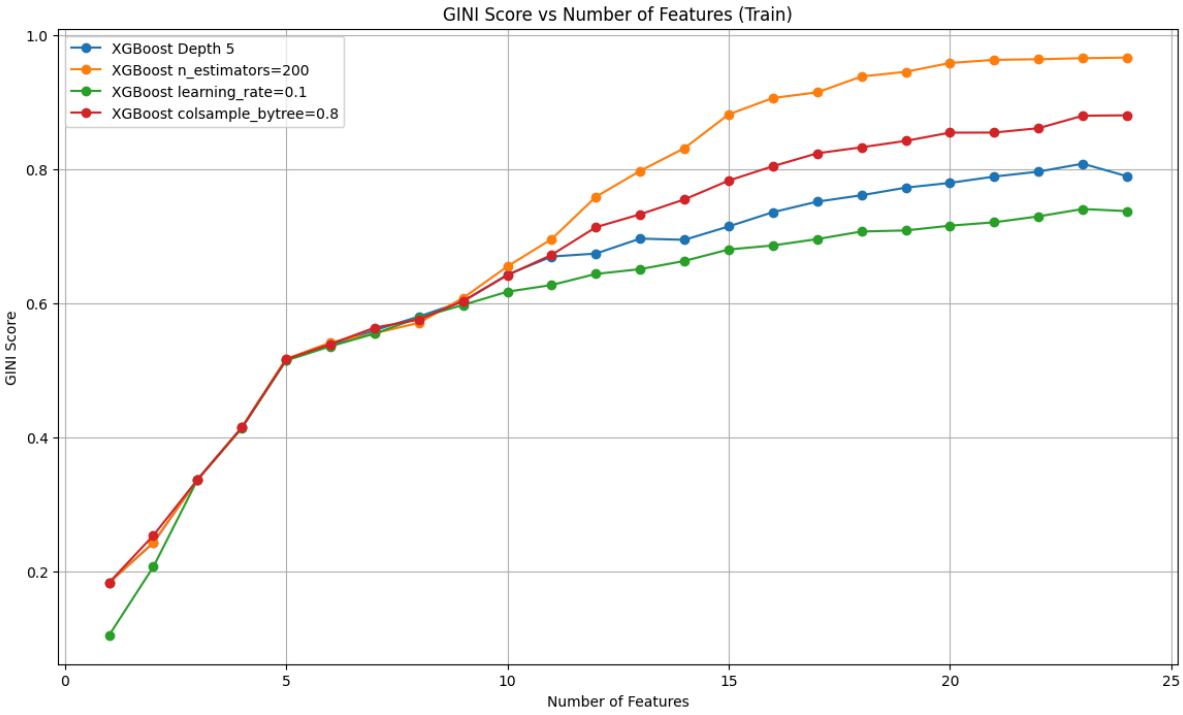
In [74]:
```
# Modele XGBoost z różnymi hiperparametrami
models_xgb = [
    ('XGBoost Depth 5', XGBClassifier(max_depth=5, use_label_encoder=False,
    ('XGBoost n_estimators=200', XGBClassifier(n_estimators=200, use_label_e
    ('XGBoost learning_rate=0.1', XGBClassifier(learning_rate=0.1, use_labe
    ('XGBoost colsample_bytree=0.8', XGBClassifier(colsample_bytree=0.8, use
]

# Zbieranie wyników dla wszystkich modeli
all_results = []
labels = []

for name, model in models_xgb:
    results = calculate_scores(model, X_train_balanced.values, y_train_balan
    all_results.append(results)
    labels.append(name)

# Organizowanie wyników
organized_results = organize_results(*all_results, labels=labels)

# Tworzenie wykresów dla każdej metryki dla zbioru testowego
plot_comparison('gini', organized_results, labels, 'GINI Score', dataset='te
plot_comparison('type_i_error', organized_results, labels, 'Type I Error Rat
plot_comparison('type_ii_error', organized_results, labels, 'Type II Error R
plot_comparison('ks', organized_results, labels, 'KS Score', dataset='test')

# Tworzenie wykresów dla każdej metryki dla zbioru treningowego
plot_comparison('gini', organized_results, labels, 'GINI Score', dataset='tr
plot_comparison('type_i_error', organized_results, labels, 'Type I Error Rat
plot_comparison('type_ii_error', organized_results, labels, 'Type II Error R
plot_comparison('ks', organized_results, labels, 'KS Score', dataset='train

# Generowanie tabeli wyników dla 5 i 10 cech
results_table_5_features = generate_results_table(organized_results, labels,
results_table_10_features = generate_results_table(organized_results, labels

# Wyświetlanie wyników w formie tabeli
print("Wyniki dla 5 cech:")
display(results_table_5_features)

print("Wyniki dla 10 cech:")
display(results_table_10_features)
```

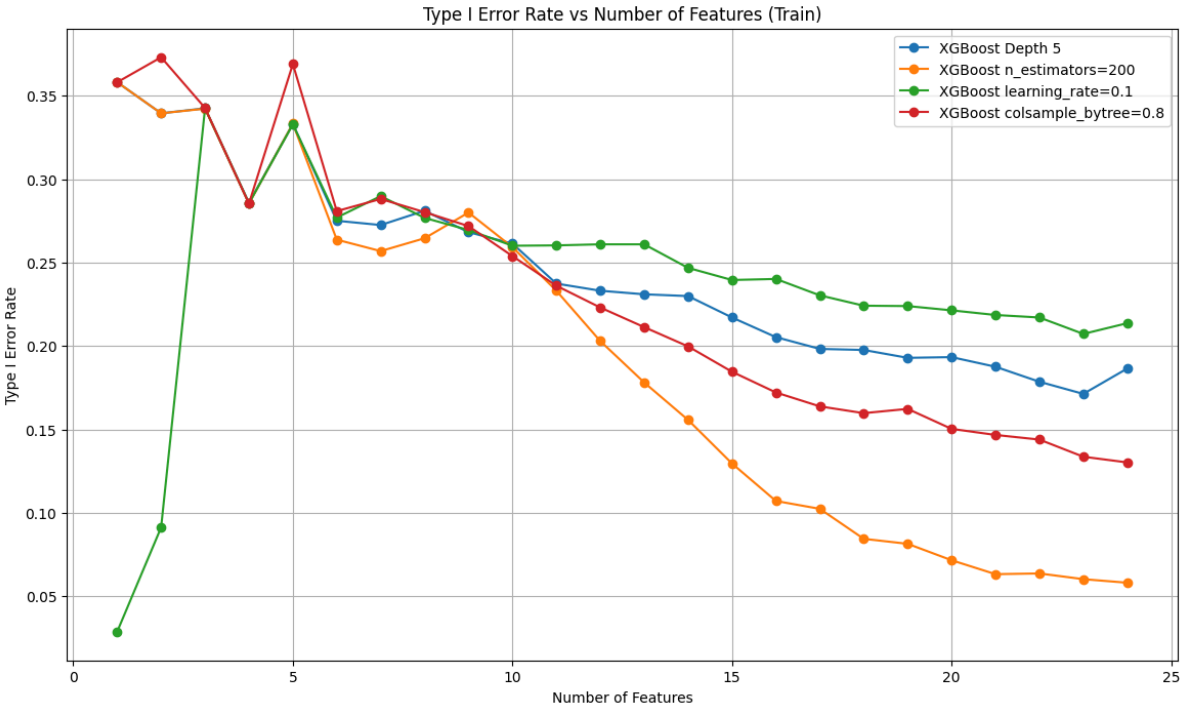

GINI Score vs Number of Features (Test)
GINI Score
Number of Features
XGBoost Depth 5
XGBoost n_estimators=200
XGBoost learning_rate=0.1
XGBoost colsample_bytree=0.8

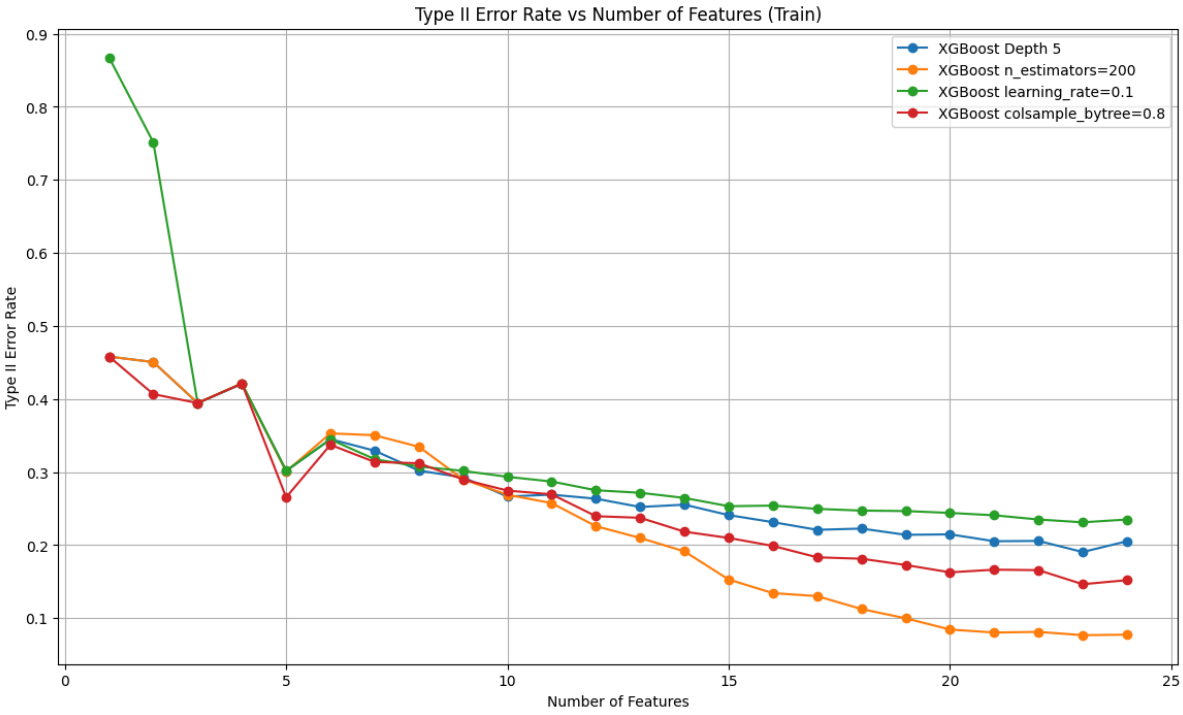

Type I Error Rate vs Number of Features (Test)
Type I Error Rate
Number of Features
XGBoost Depth 5
XGBoost n_estimators=200
XGBoost learning_rate=0.1
XGBoost colsample_bytree=0.8

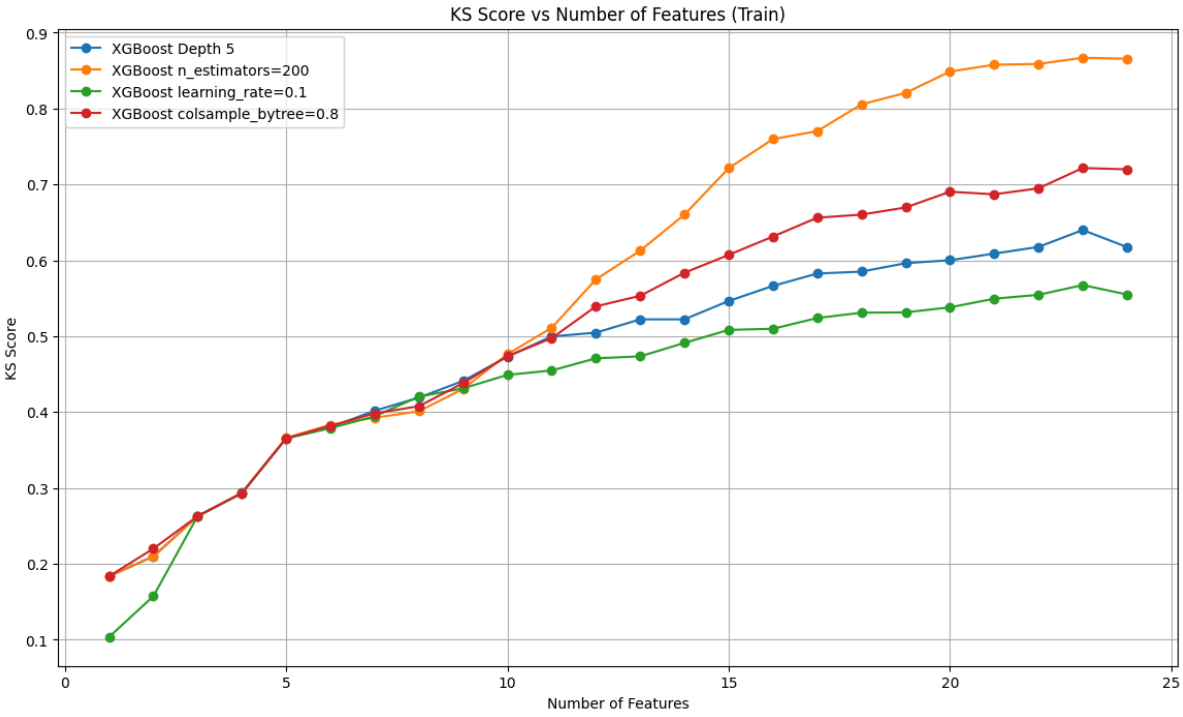

Type II Error Rate vs Number of Features (Test)
Type II Error Rate
Number of Features
XGBoost Depth 5
XGBoost n_estimators=200
XGBoost learning_rate=0.1
XGBoost colsample_bytree=0.8


KS Score vs Number of Features (Test)
KS Score
Number of Features
XGBoost Depth 5
XGBoost n_estimators=200
XGBoost learning_rate=0.1
XGBoost colsample_bytree=0.8


GINI Score vs Number of Features (Train)
XGBoost Depth 5
XGBoost n_estimators=200
XGBoost learning_rate=0.1
XGBoost colsample_bytree=0.8
GINI Score
Number of Features
1.0
0.8
0.6
0.4
0.2
0
5
10
15
20
25


Type I Error Rate vs Number of Features (Train)
XGBoost Depth 5
XGBoost n_estimators=200
XGBoost learning_rate=0.1
XGBoost colsample_bytree=0.8
Type I Error Rate
Number of Features
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0
5
10
15
20
25





Wyniki dla 5 cech:

	Model	GINI Score Train	GINI Score Test	Type I Error Train	Type I Error Test	Type II Error Train	Type II Error Test	KS Sc T
0	XGBoost Depth 5	0.515731	0.487963	0.333119	0.344448	0.301455	0.319366	0.365
1	XGBoost n_estimators=200	0.517137	0.486726	0.333333	0.344966	0.300813	0.319014	0.365
2	XGBoost learning_rate=0.1	0.514805	0.487311	0.332905	0.344408	0.302097	0.320070	0.365
3	XGBoost colsample_bytree=0.8	0.516698	0.487369	0.369063	0.378766	0.265511	0.272183	0.365

Wyniki dla 10 cech:

	Model	GINI Score Train	GINI Score Test	Type I Error Train	Type I Error Test	Type II Error Train	Type II Error Test	KS S T
0	XGBoost Depth 5	0.642791	0.499041	0.261446	0.316496	0.266581	0.330634	0.473
1	XGBoost n_estimators=200	0.655073	0.489637	0.259307	0.319561	0.268935	0.339085	0.476
2	XGBoost learning_rate=0.1	0.617394	0.515177	0.260163	0.299300	0.293539	0.328169	0.449
3	XGBoost colsample_bytree=0.8	0.641985	0.494729	0.253958	0.312825	0.274711	0.329930	0.473

Wybrane Modele

In [108…

```
def organize_results_with_lift(*results, labels):
    organized_results = {
        'gini': {'train': [], 'test': []},
        'type_i_error': {'train': [], 'test': []},
        'type_ii_error': {'train': [], 'test': []},
        'ks': {'train': [], 'test': []},
        'lift': {'train': [], 'test': []}
    }

    for result in results:
        (gini_scores_train, gini_scores_test, type_i_errors_train, type_i_er
         type_ii_errors_train, type_ii_errors_test, ks_scores_train, ks_scor
         lift_scores_train, lift_scores_test) = result

        organized_results['gini']['train'].append(gini_scores_train)
        organized_results['gini']['test'].append(gini_scores_test)
        organized_results['type_i_error']['train'].append(type_i_errors_tra
        organized_results['type_i_error']['test'].append(type_i_errors_test
        organized_results['type_ii_error']['train'].append(type_ii_errors_t
        organized_results['type_ii_error']['test'].append(type_ii_errors_te
        organized_results['ks']['train'].append(ks_scores_train)
        organized_results['ks']['test'].append(ks_scores_test)
        organized_results['lift']['train'].append(lift_scores_train)
        organized_results['lift']['test'].append(lift_scores_test)

    return organized_results
```

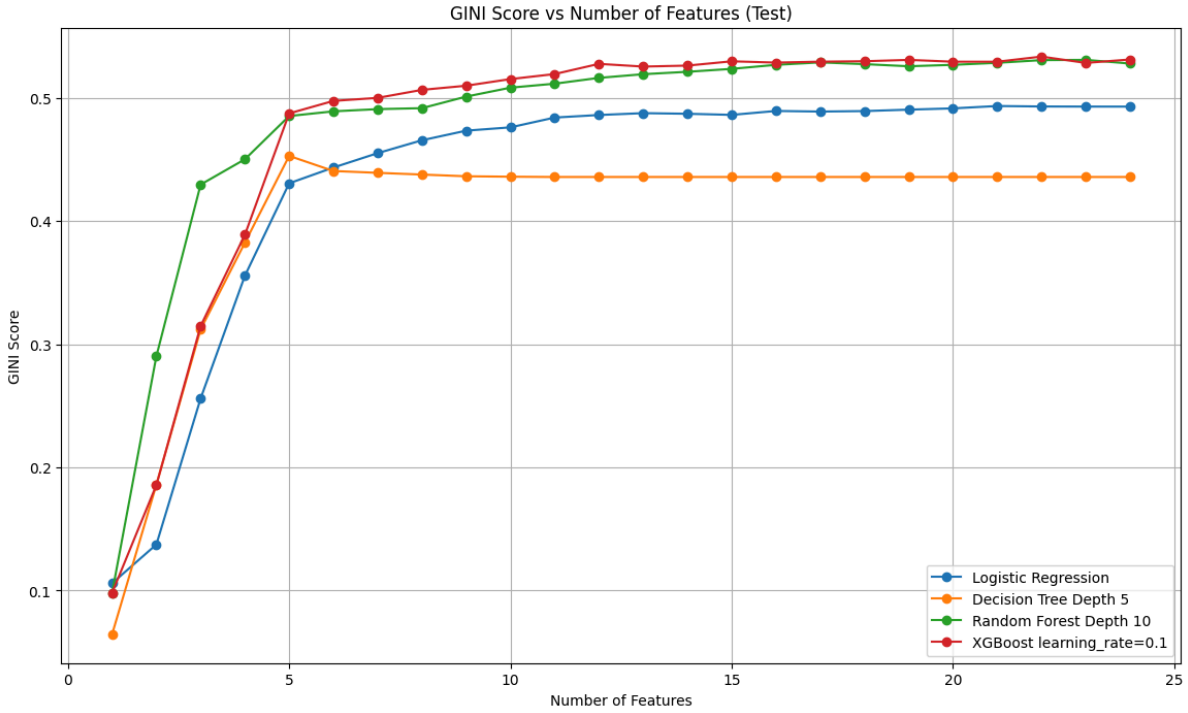
In [109…

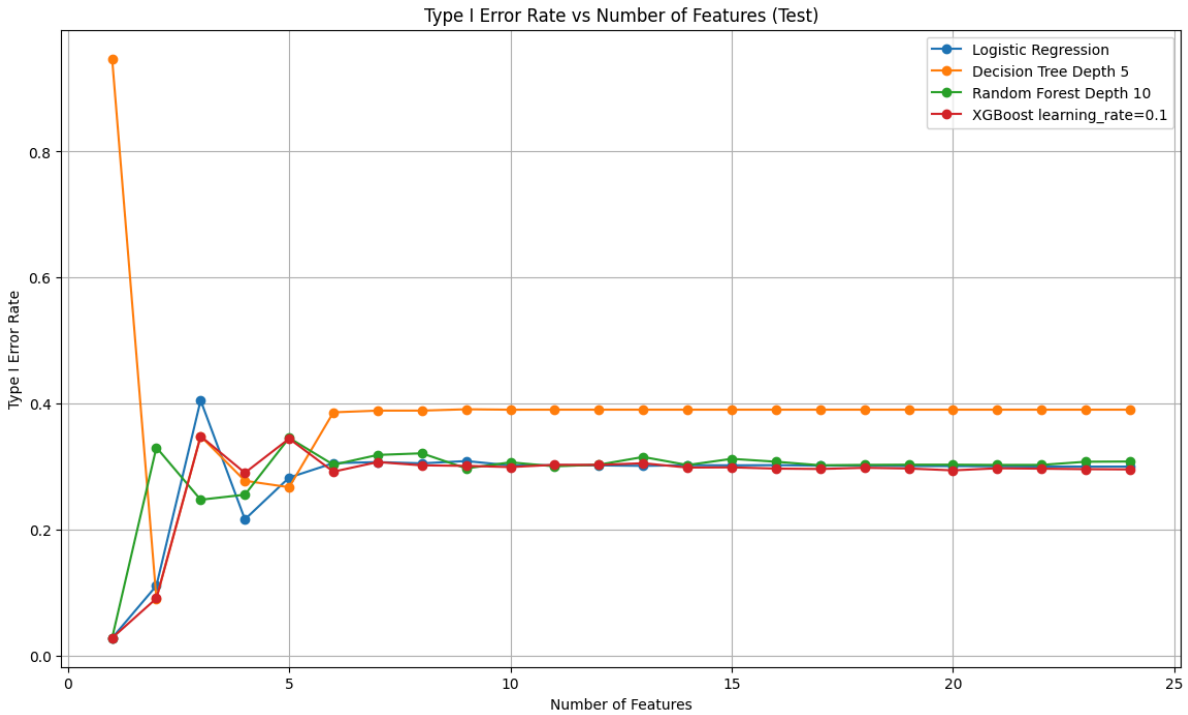
```
# Liczba wybranych cech
max_features = X_train_balanced.shape[1]

models = [
    ('Logistic Regression', LogisticRegression()),
    ('Decision Tree Depth 5', DecisionTreeClassifier(max_depth=5)),
    ('Random Forest Depth 10', RandomForestClassifier(max_depth=10)),
    ('XGBoost learning_rate=0.1', XGBClassifier(learning_rate=0.1, use_label
]

# Ewaluacja i wizualizacja wyników dla różnych modeli
results_logistic_regression = calculate_metrics(models[0][1], X_train_balanc
results_decision_tree = calculate_metrics(models[1][1], X_train_balanced.val
results_random_forest = calculate_metrics(models[2][1], X_train_balanced.val
results_xgboost = calculate_metrics(models[3][1], X_train_balanced.values, y

# Organizowanie wyników dla wszystkich modeli
labels = [
```

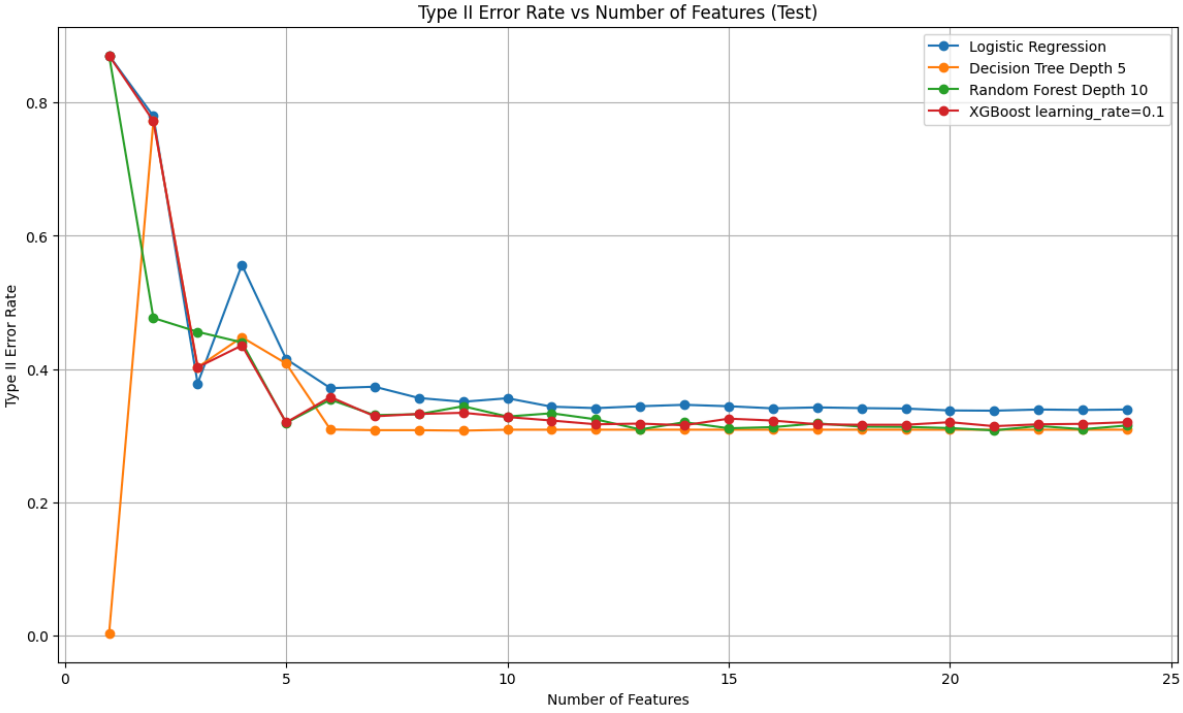
```
    'Logistic Regression', 'Decision Tree Depth 5', 'Random Forest Depth 10
]

all_results = [
    results_logistic_regression, results_decision_tree, results_random_fores
]

organized_results = organize_results_with_lift(*all_results, labels=labels)
```

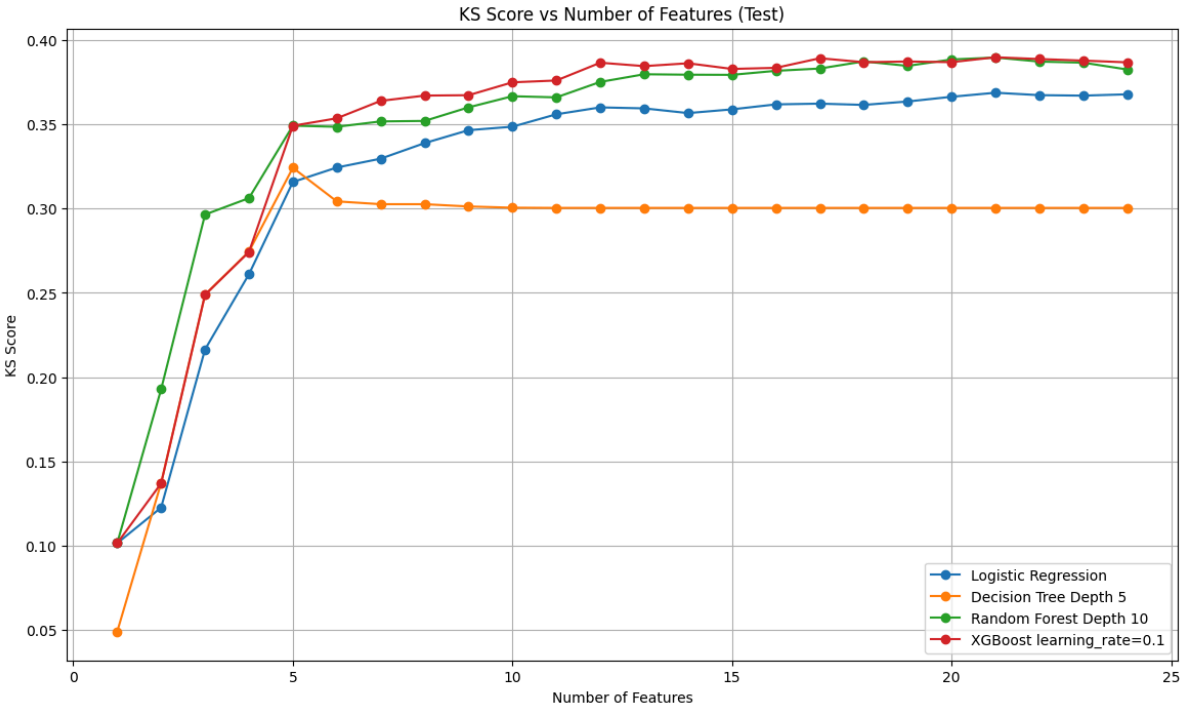
In [110…

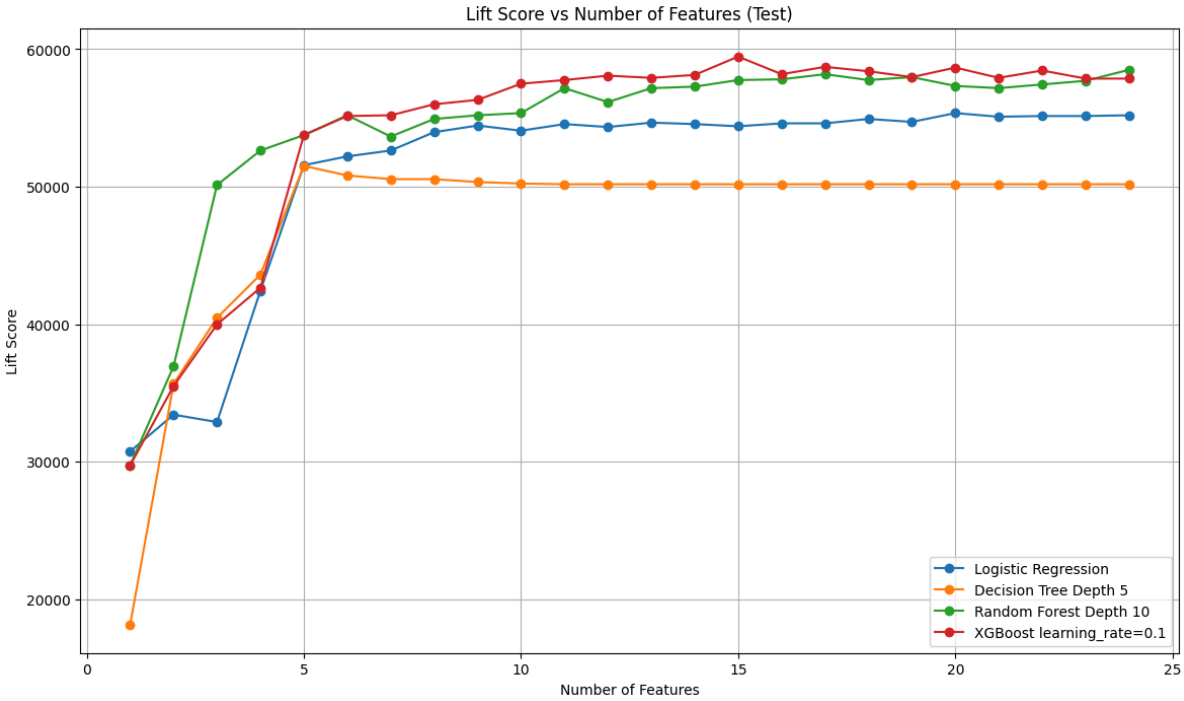
```
# Tworzenie wykresów dla każdej metryki
plot_comparison('gini', organized_results, labels, 'GINI Score')
plot_comparison('type_i_error', organized_results, labels, 'Type I Error Rat
plot_comparison('type_ii_error', organized_results, labels, 'Type II Error R
plot_comparison('ks', organized_results, labels, 'KS Score')
plot_comparison('lift', organized_results, labels, 'Lift Score')
```






Type II Error Rate vs Number of Features (Test)
Logistic Regression
Decision Tree Depth 5
Random Forest Depth 10
XGBoost learning_rate=0.1
Type II Error Rate
Number of Features
0.0
0.2
0.4
0.6
0.8
0
5
10
15
20
25


KS Score vs Number of Features (Test)
Logistic Regression
Decision Tree Depth 5
Random Forest Depth 10
XGBoost learning_rate=0.1
KS Score
Number of Features
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0
5
10
15
20
25



Zmienne

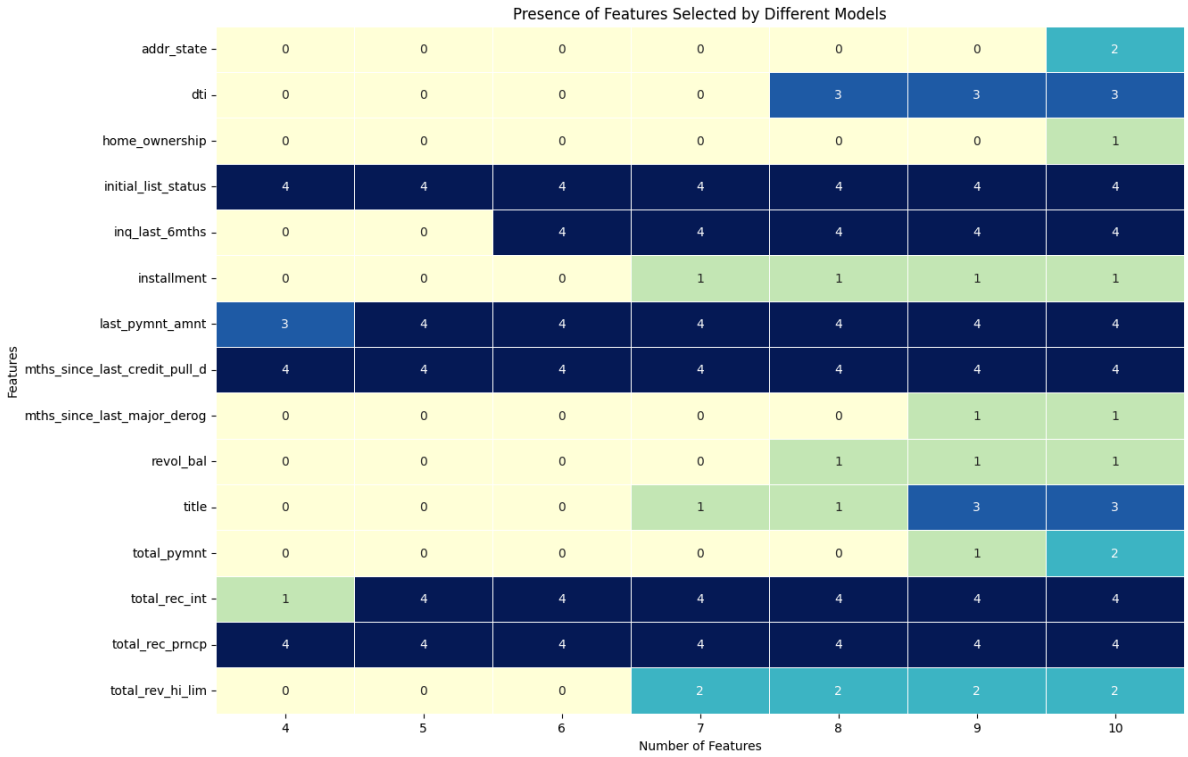
In [119…

```
def create_feature_table(estimator, X_train, y_train, feature_counts):
    feature_names = X_train.columns
    selected_features_list = []
    for n_features in feature_counts:
        selector = RFE(estimator, n_features_to_select=n_features)
        selector.fit(X_train, y_train)
        selected_features = feature_names[selector.support_].tolist()
        selected_features_list.append(selected_features)
    return selected_features_list

# Range of number of features to consider
feature_counts = range(4, 11)

# Create lists of selected features for different models
logistic_features = create_feature_table(LogisticRegression(), X_train_balar
decision_tree_features = create_feature_table(DecisionTreeClassifier(max_dep
random_forest_features = create_feature_table(RandomForestClassifier(max_dep
xgboost_features = create_feature_table(XGBClassifier(learning_rate=0.1, use

# Create DataFrame
features_df = pd.DataFrame({
    'Feature Count': feature_counts,
    'Logistic Regression': logistic_features,
    'Decision Tree Depth 5': decision_tree_features,
    'Random Forest Depth 10': random_forest_features,
    'XGBoost learning_rate=0.1': xgboost_features
})

display(features_df)
```

	Feature Count	Logistic Regression	Decision Tree Depth 5	Random Forest Depth 10	XGBoost learning_rate=0.1
0	4	[initial_list_status, total_rec_prncp, last_py...	[initial_list_status, total_rec_prncp, last_py...	[initial_list_status, total_rec_prncp, total_r...	[initial_list_status, total_rec_prncp, last_py...
1	5	[initial_list_status, total_rec_prncp, total_r...	[initial_list_status, total_rec_prncp, total_r...	[initial_list_status, total_rec_prncp, total_r...	[initial_list_status, total_rec_prncp, total_r...
2	6	[inq_last_6mths, initial_list_status, total_re...	[inq_last_6mths, initial_list_status, total_re...	[inq_last_6mths, initial_list_status, total_re...	[inq_last_6mths, initial_list_status, total_re...
3	7	[inq_last_6mths, initial_list_status, total_re...	[installment, inq_last_6mths, initial_list_sta...	[title, inq_last_6mths, initial_list_status, t...	[inq_last_6mths, initial_list_status, total_re...
4	8	[dti, inq_last_6mths, initial_list_status, tot...	[installment, inq_last_6mths, revol_bal, initi...	[title, dti, inq_last_6mths, initial_list_stat...	[dti, inq_last_6mths, initial_list_status, tot...
5	9	[title, dti, inq_last_6mths, initial_list_stat...	[installment, inq_last_6mths, revol_bal, initi...	[title, dti, inq_last_6mths, initial_list_stat...	[title, dti, inq_last_6mths, initial_list_stat...
6	10	[title, addr_state, dti, inq_last_6mths, initi...	[installment, inq_last_6mths, revol_bal, initi...	[title, addr_state, dti, inq_last_6mths, initi...	[home_ownership, title, dti, inq_last_6mths, i...

In [120…

```
# Create a set of all features selected by any model
all_features = set()
for col in ['Logistic Regression', 'Decision Tree Depth 5', 'Random Forest D
    for feature_list in features_df[col]:
        all_features.update(feature_list)

all_features = sorted(all_features)

# Initialize presence matrix
presence_matrix = pd.DataFrame(0, index=all_features, columns=features_df['F

# Populate presence matrix
for feature in all_features:
    for count in features_df['Feature Count']:
        presence_matrix.loc[feature, count] = sum([feature in feature_list f

# Visualize heatmap
plt.figure(figsize=(14, 10))
sns.heatmap(presence_matrix.astype(float), annot=True, cmap='YlGnBu', cbar=F
plt.title('Presence of Features Selected by Different Models')
plt.xlabel('Number of Features')
plt.ylabel('Features')
plt.show()
```



In [122…
```
features_df.to_excel("selected_features_clean.xlsx", index=False)
```

SHAP global

In [125…
```
!pip install shap
```

```
Requirement already satisfied: shap in /Users/dominikamatusiak/anaconda3/li
b/python3.11/site-packages (0.45.1)
Requirement already satisfied: numpy in /Users/dominikamatusiak/anaconda3/l
ib/python3.11/site-packages (from shap) (1.24.3)
Requirement already satisfied: scipy in /Users/dominikamatusiak/anaconda3/l
ib/python3.11/site-packages (from shap) (1.11.1)
Requirement already satisfied: scikit-learn in /Users/dominikamatusiak/anac
onda3/lib/python3.11/site-packages (from shap) (1.5.0)
Requirement already satisfied: pandas in /Users/dominikamatusiak/anaconda3/
lib/python3.11/site-packages (from shap) (2.2.2)
Requirement already satisfied: tqdm>=4.27.0 in /Users/dominikamatusiak/anac
onda3/lib/python3.11/site-packages (from shap) (4.65.0)
Requirement already satisfied: packaging>20.9 in /Users/dominikamatusiak/an
aconda3/lib/python3.11/site-packages (from shap) (23.1)
Requirement already satisfied: slicer==0.0.8 in /Users/dominikamatusiak/ana
conda3/lib/python3.11/site-packages (from shap) (0.0.8)
Requirement already satisfied: numba in /Users/dominikamatusiak/anaconda3/l
ib/python3.11/site-packages (from shap) (0.57.1)
Requirement already satisfied: cloudpickle in /Users/dominikamatusiak/anaco
nda3/lib/python3.11/site-packages (from shap) (2.2.1)
Requirement already satisfied: llvmlite<0.41,>=0.40.0dev0 in /Users/dominik
amatusiak/anaconda3/lib/python3.11/site-packages (from numba->shap) (0.40.
0)
Requirement already satisfied: python-dateutil>=2.8.2 in /Users/dominikamat
usiak/anaconda3/lib/python3.11/site-packages (from pandas->shap) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /Users/dominikamatusiak/anac
onda3/lib/python3.11/site-packages (from pandas->shap) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.7 in /Users/dominikamatusiak/an
aconda3/lib/python3.11/site-packages (from pandas->shap) (2023.3)
Requirement already satisfied: joblib>=1.2.0 in /Users/dominikamatusiak/ana
conda3/lib/python3.11/site-packages (from scikit-learn->shap) (1.2.0)
Requirement already satisfied: threadpoolctl>=3.1.0 in /Users/dominikamatus
iak/anaconda3/lib/python3.11/site-packages (from scikit-learn->shap) (3.5.
0)
Requirement already satisfied: six>=1.5 in /Users/dominikamatusiak/anaconda
3/lib/python3.11/site-packages (from python-dateutil>=2.8.2->pandas->shap)
(1.16.0)
```

In [24]:
```python
import shap
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.feature_selection import RFE

# Wybrane modele
models = {
    'Logistic Regression': LogisticRegression(),
    'XGBoost learning_rate=0.1': XGBClassifier(learning_rate=0.1, use_label_
}
```

In [25]:
```python
print('hello')
```

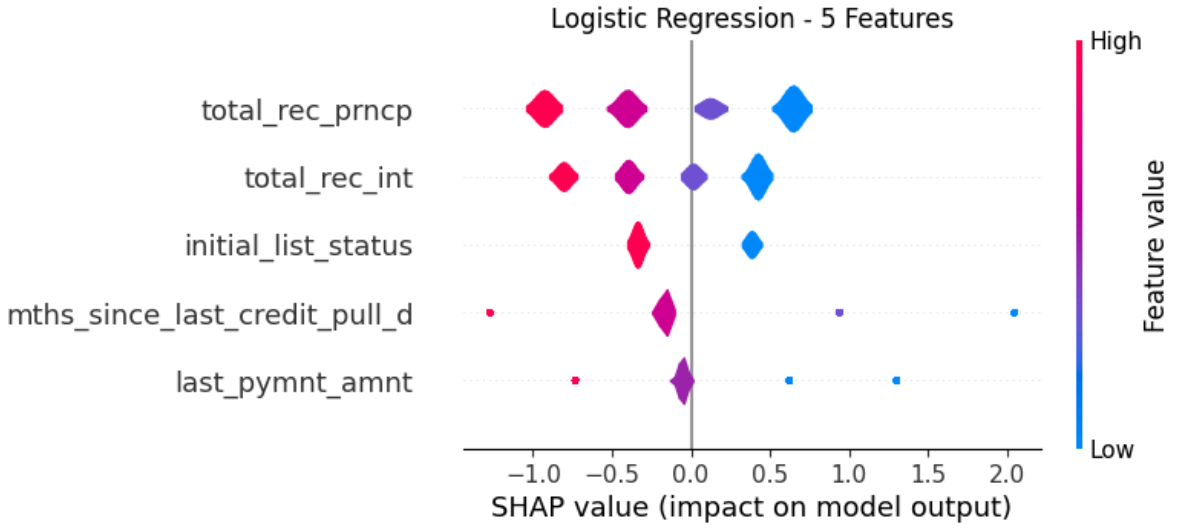
```
hello
```

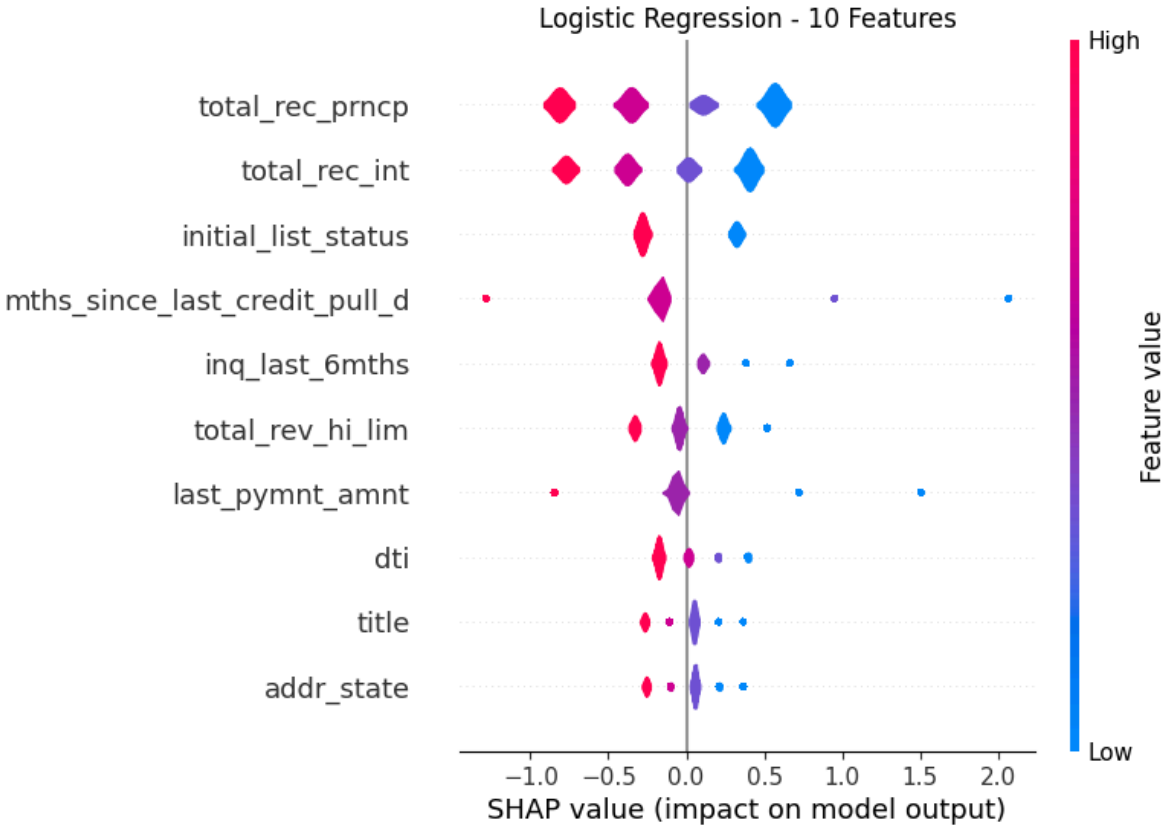
In [26]:
```python
def calculate_shap_values(model, X_train, y_train, X_test):
    model.fit(X_train, y_train)
    explainer = shap.Explainer(model, X_train)
    shap_values = explainer(X_test)
    return shap_values

# Obliczanie SHAP dla 5 i 10 zmiennych
feature_counts = [5, 10]
shap_values_dict = {}
```

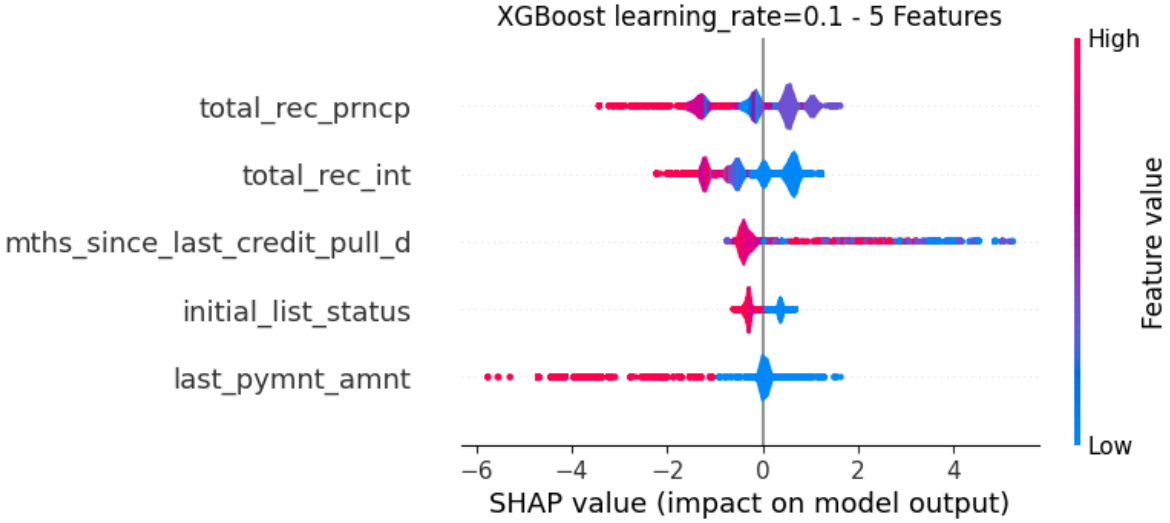
```
for model_name, model in models.items():
    shap_values_dict[model_name] = {}
    for count in feature_counts:
        selector = RFE(model, n_features_to_select=count)
        selector.fit(X_train_balanced, y_train_balanced)
        selected_features = X_train_balanced.columns[selector.support_]
        X_train_selected = X_train_balanced[selected_features]
        X_test_selected = X_test[selected_features]
        shap_values = calculate_shap_values(model, X_train_selected, y_trai
        shap_values_dict[model_name][count] = (shap_values, selected_feature
```

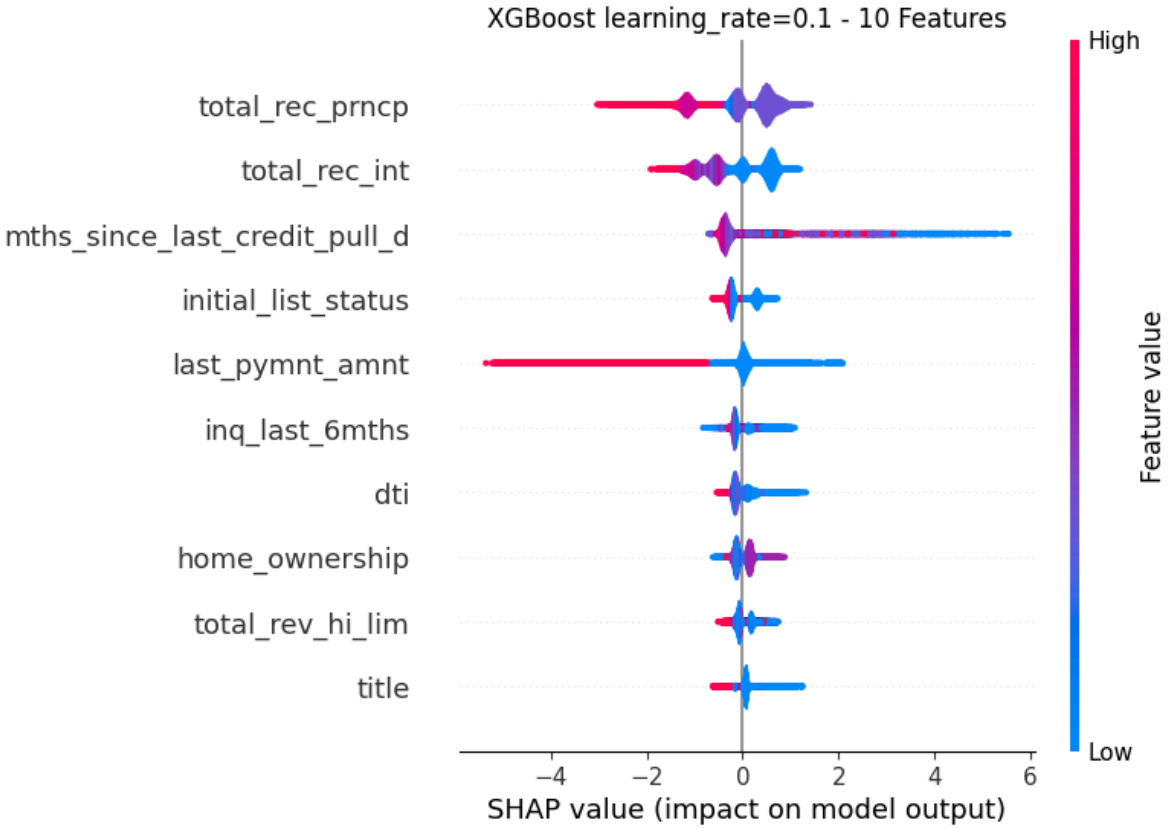
```
100%|====================| 151201/151594 [05:28<00:00]
```

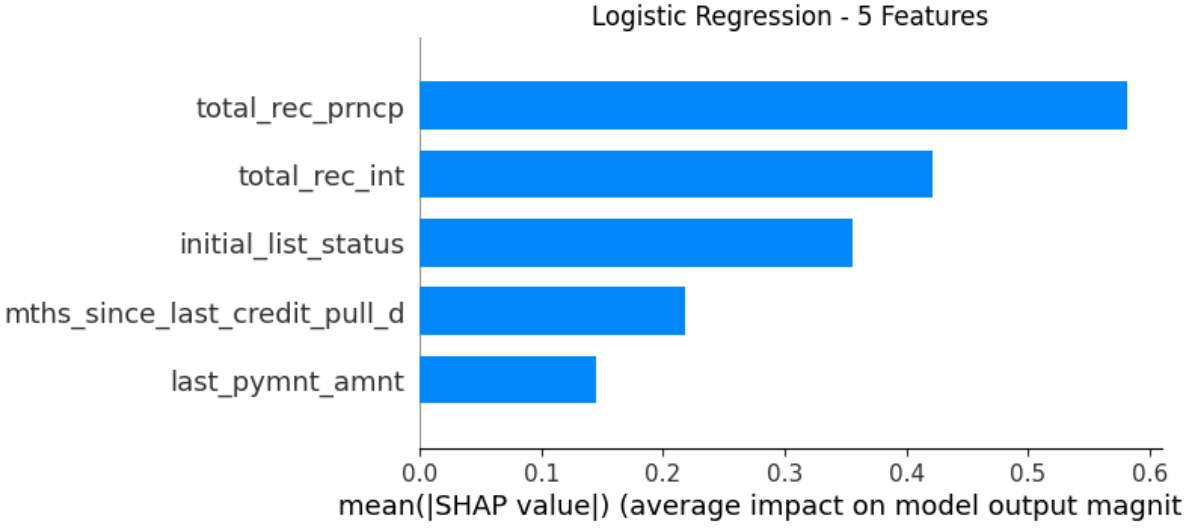
In [27]:
```
# Funkcja do tworzenia wykresu violin plot
def create_violin_plot(shap_values, X, title):
    plt.figure(figsize=(10, 6))
    shap.summary_plot(shap_values, X, plot_type="violin", show=False)
    plt.title(title)
    plt.show()
    plt.close()  # Zamknij figurę po wyświetleniu

# Tworzenie wykresów violin plot dla 5 i 10 zmiennych
for model_name, model_data in shap_values_dict.items():
    for count, (shap_values, selected_features, X_test_selected) in model_da
        title = f'{model_name} - {count} Features'
        create_violin_plot(shap_values, X_test_selected, title)

# Funkcja do tworzenia wykresu bar plot
def create_bar_plot(shap_values, X, title):
    plt.figure(figsize=(10, 6))
    shap.summary_plot(shap_values, X, plot_type="bar", show=False)
    plt.title(title)
    plt.show()
    plt.close()  # Zamknij figurę po wyświetleniu

# Tworzenie wykresów bar plot dla 5 i 10 zmiennych
for model_name, model_data in shap_values_dict.items():
    for count, (shap_values, selected_features, X_test_selected) in model_da
        title = f'{model_name} - {count} Features'
        create_bar_plot(shap_values, X_test_selected, title)
```

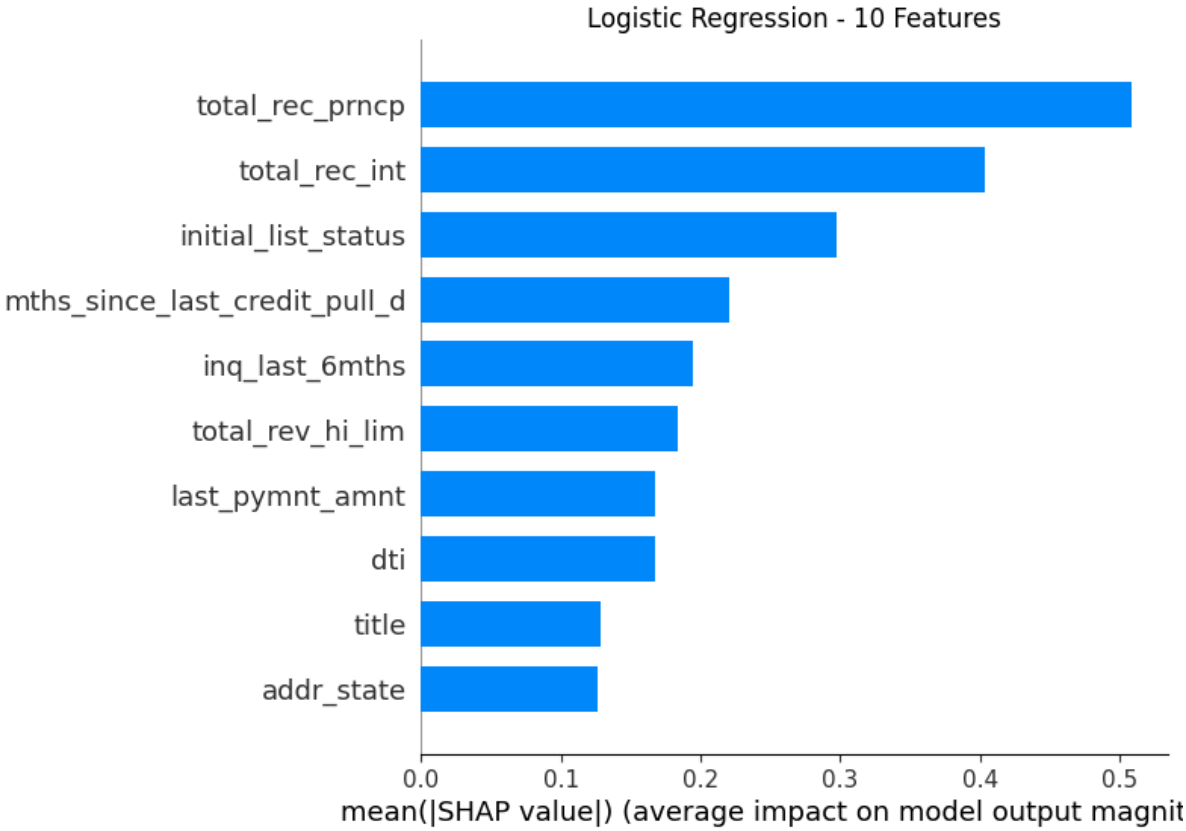


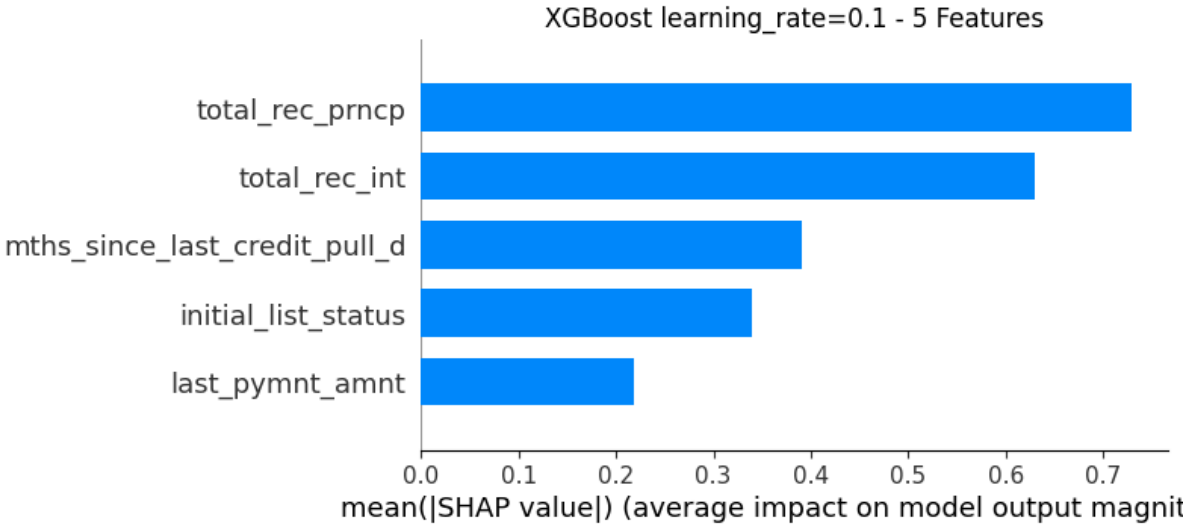

Logistic Regression - 10 Features
total_rec_prncp
total_rec_int
initial_list_status
mths_since_last_credit_pull_d
inq_last_6mths
total_rev_hi_lim
last_pymnt_amnt
dti
title
addr_state
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
SHAP value (impact on model output)
High
Feature value
Low

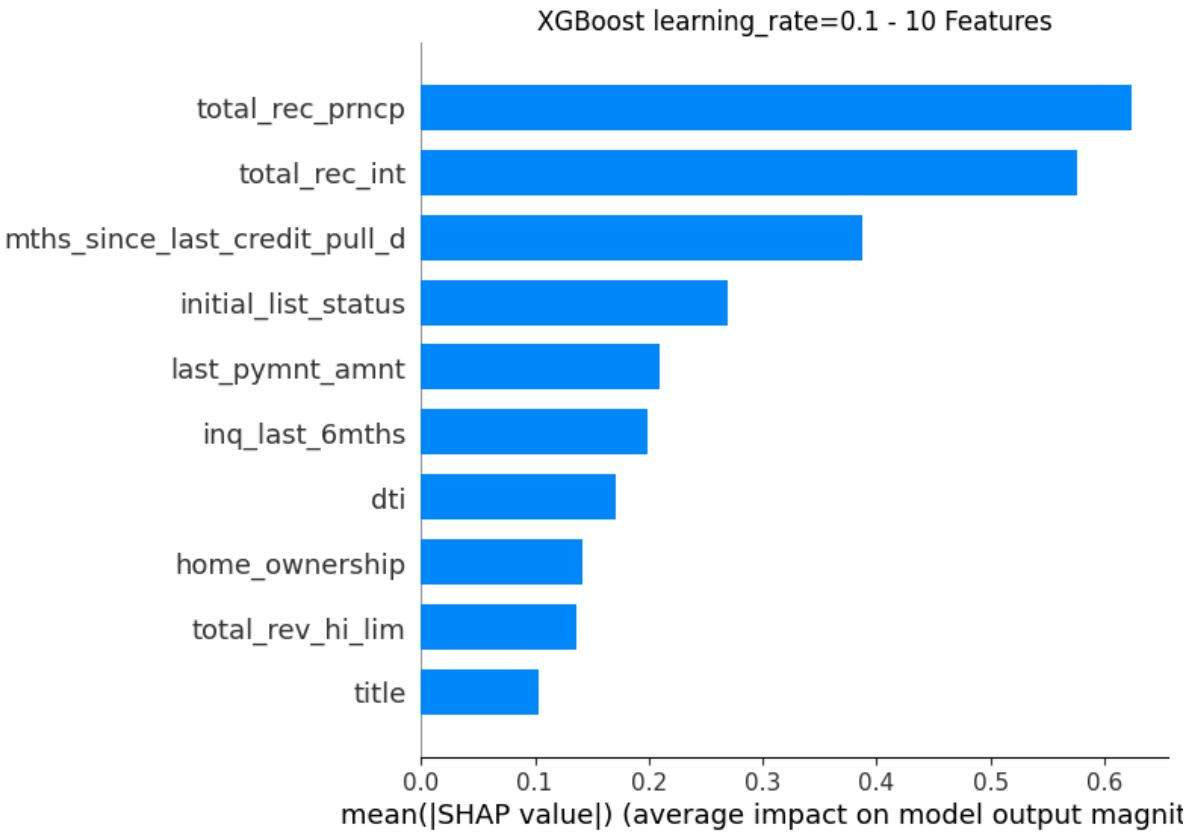

XGBoost learning_rate=0.1 - 5 Features
total_rec_prncp
total_rec_int
mths_since_last_credit_pull_d
initial_list_status
last_pymnt_amnt
−6
−4
−2
0
2
4
SHAP value (impact on model output)
High
Feature value
Low


XGBoost learning_rate=0.1 - 10 Features
total_rec_prncp
total_rec_int
mths_since_last_credit_pull_d
initial_list_status
last_pymnt_amnt
inq_last_6mths
dti
home_ownership
total_rev_hi_lim
title
−4
−2
0
2
4
6
SHAP value (impact on model output)
High
Feature value
Low


Logistic Regression - 5 Features
total_rec_prncp
total_rec_int
initial_list_status
mths_since_last_credit_pull_d
last_pymnt_amnt
0.0
0.1
0.2
0.3
0.4
0.5
0.6
mean(|SHAP value|) (average impact on model output magnit


Logistic Regression - 10 Features
total_rec_prncp
total_rec_int
initial_list_status
mths_since_last_credit_pull_d
inq_last_6mths
total_rev_hi_lim
last_pymnt_amnt
dti
title
addr_state
0.0
0.1
0.2
0.3
0.4
0.5
mean(|SHAP value|) (average impact on model output magnit

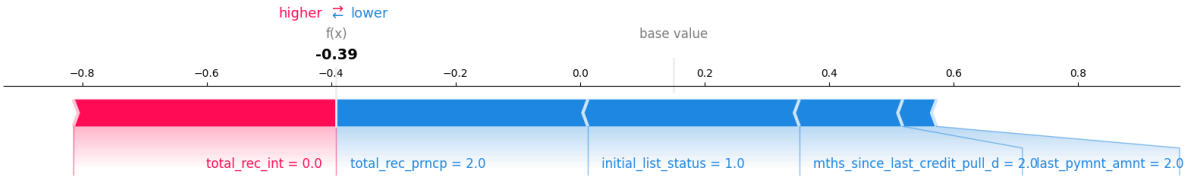

XGBoost learning_rate=0.1 - 5 Features
total_rec_prncp
total_rec_int
mths_since_last_credit_pull_d
initial_list_status
last_pymnt_amnt
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
mean(|SHAP value|) (average impact on model output magnit



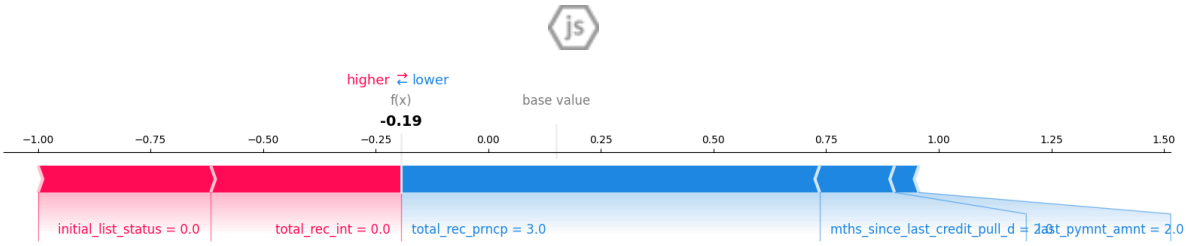
In [49]:
```
# Funkcja do tworzenia wykresu force plot
def create_force_plot(expected_value, shap_values, features, title):
    shap.initjs()
    shap_plot = shap.force_plot(expected_value, shap_values, features, matp
    print(title)
    display(shap_plot)
    plt.close()

# Wybór 5 zmiennych
n_features = 5
selected_indices = [0, 1, 2]  # Wybór indeksów obserwacji do analizy

# Obliczanie wartości SHAP dla wybranych modeli i zmiennych
shap_values_dict = {}

for model_name, model in models.items():
    selector = RFE(model, n_features_to_select=n_features)
    selector.fit(X_train_balanced, y_train_balanced)
    selected_features = X_train_balanced.columns[selector.support_]
    X_train_selected = X_train_balanced[selected_features]
    X_test_selected = X_test[selected_features]
    shap_values = calculate_shap_values(model, X_train_selected, y_train_ba
    shap_values_dict[model_name] = (shap_values, selected_features, X_test_s
```

```
100%|==================| 151226/151594 [03:25<00:00]
```

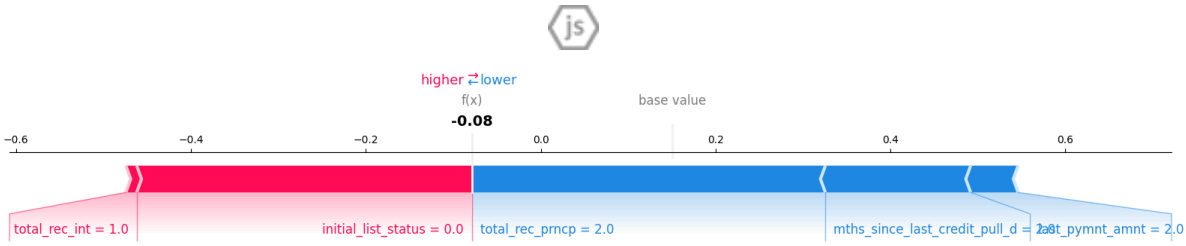
In [50]:
```
# Tworzenie wykresów force plot dla wybranych modeli i zmiennych
for model_name, (shap_values, selected_features, X_test_selected) in shap_va
    for idx in selected_indices:
        title = f'{model_name} - Obserwacja {idx+1}'
        create_force_plot(shap_values[idx].base_values, shap_values[idx].va
```



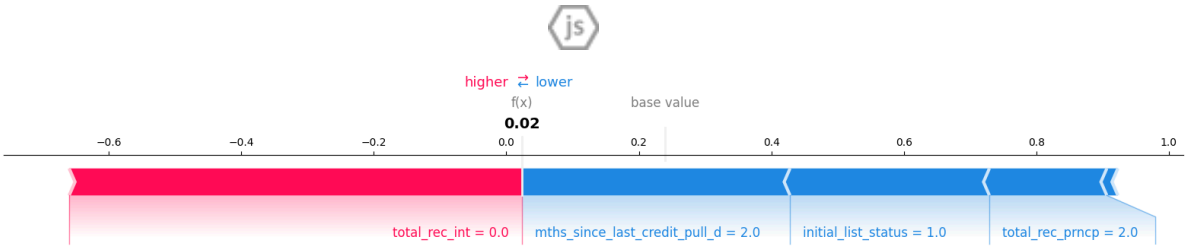
Logistic Regression – Obserwacja 1
None



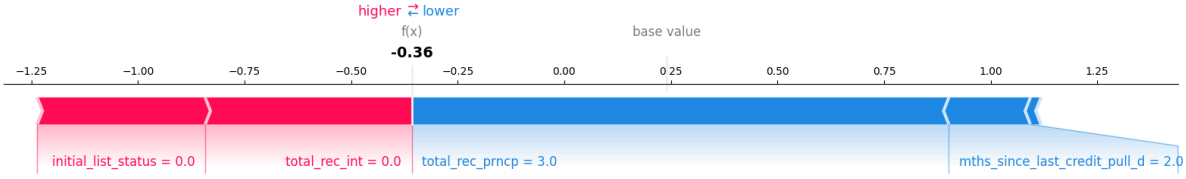
Logistic Regression – Obserwacja 2
None



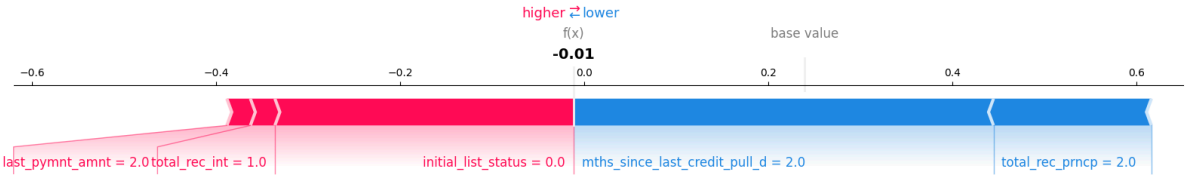
Logistic Regression – Obserwacja 3
None



XGBoost learning_rate=0.1 – Obserwacja 1
None



XGBoost learning_rate=0.1 – Obserwacja 2
None

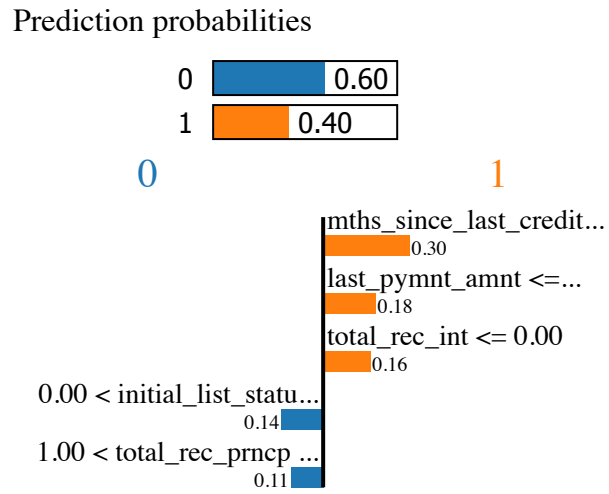


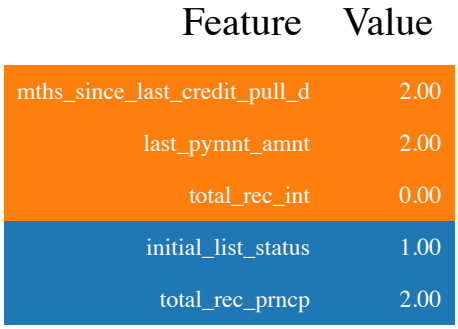

```
XGBoost learning_rate=0.1 - Obserwacja 3
None
```

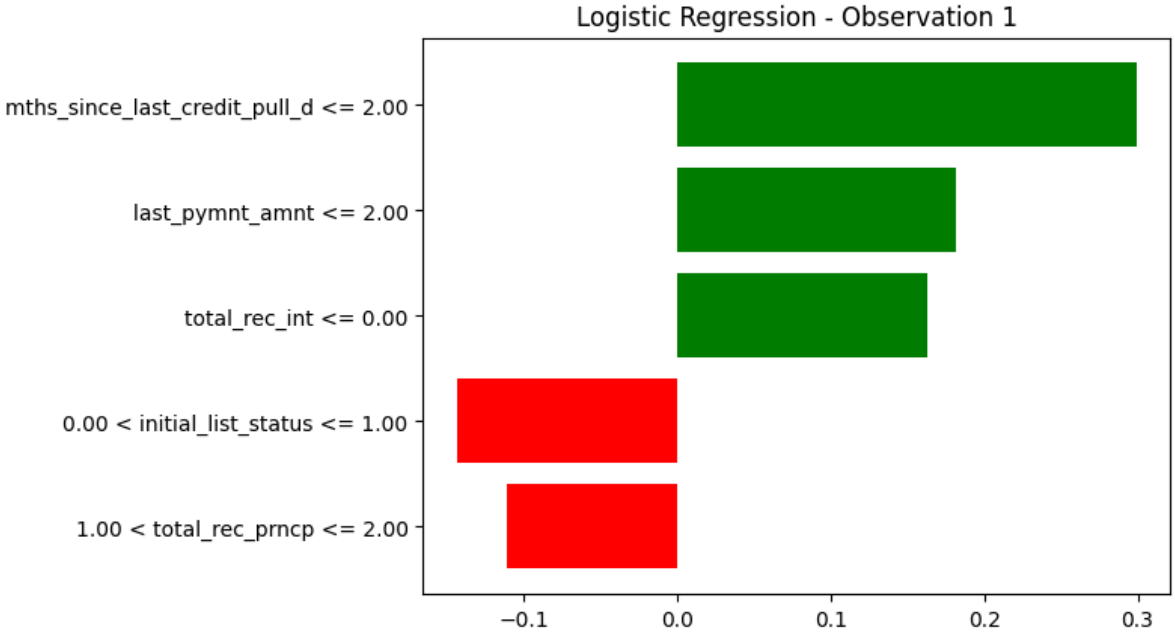
lime

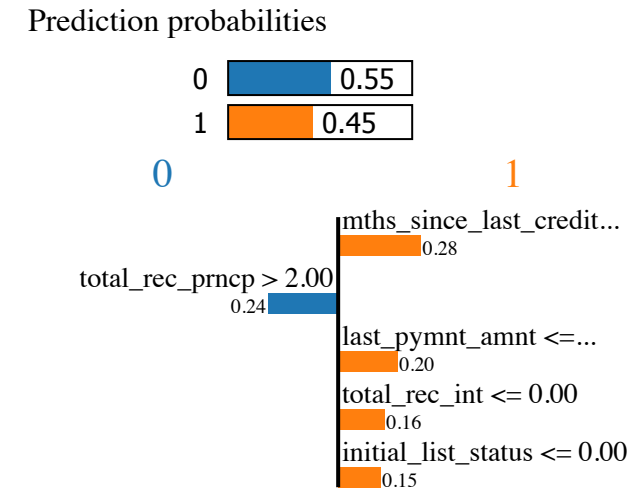
In [30]:
```python
import lime
from lime import lime_tabular

def explain_instance_with_lime(model, explainer, X_test, index):
    exp = explainer.explain_instance(
        data_row=X_test.iloc[index],
        predict_fn=model.predict_proba,
        num_features=len(X_test.columns)
    )
    return exp


# Obliczanie wartości SHAP dla wybranych modeli i zmiennych
lime_explainers = {}
explanations = {}

for model_name, model in models.items():
    selector = RFE(model, n_features_to_select=n_features)
    selector.fit(X_train_balanced, y_train_balanced)
    selected_features = X_train_balanced.columns[selector.support_]
    X_train_selected = X_train_balanced[selected_features]
    X_test_selected = X_test[selected_features]

    # Trenujemy model
    model.fit(X_train_selected, y_train_balanced)

    # Tworzymy obiekt explainer LIME
    explainer = lime_tabular.LimeTabularExplainer(
        training_data=X_train_selected.values,
        feature_names=selected_features,
        class_names=[0, 1],
        mode='classification'
    )

    lime_explainers[model_name] = explainer
    explanations[model_name] = {}

    for idx in selected_indices:
        exp = explain_instance_with_lime(model, explainer, X_test_selected,
        explanations[model_name][idx] = exp
```

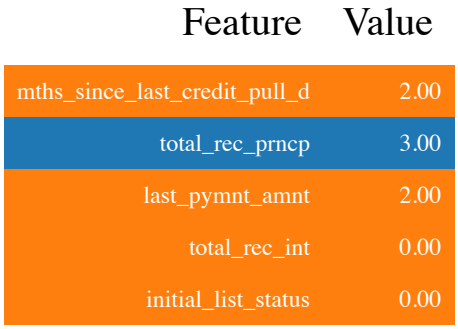
In [31]:
```python
for model_name, model_explanations in explanations.items():
    for idx, exp in model_explanations.items():
        title = f'{model_name} - Observation {idx+1}'
        exp.show_in_notebook(show_table=True)
        exp.as_pyplot_figure()
        plt.title(title)
        plt.show()
```

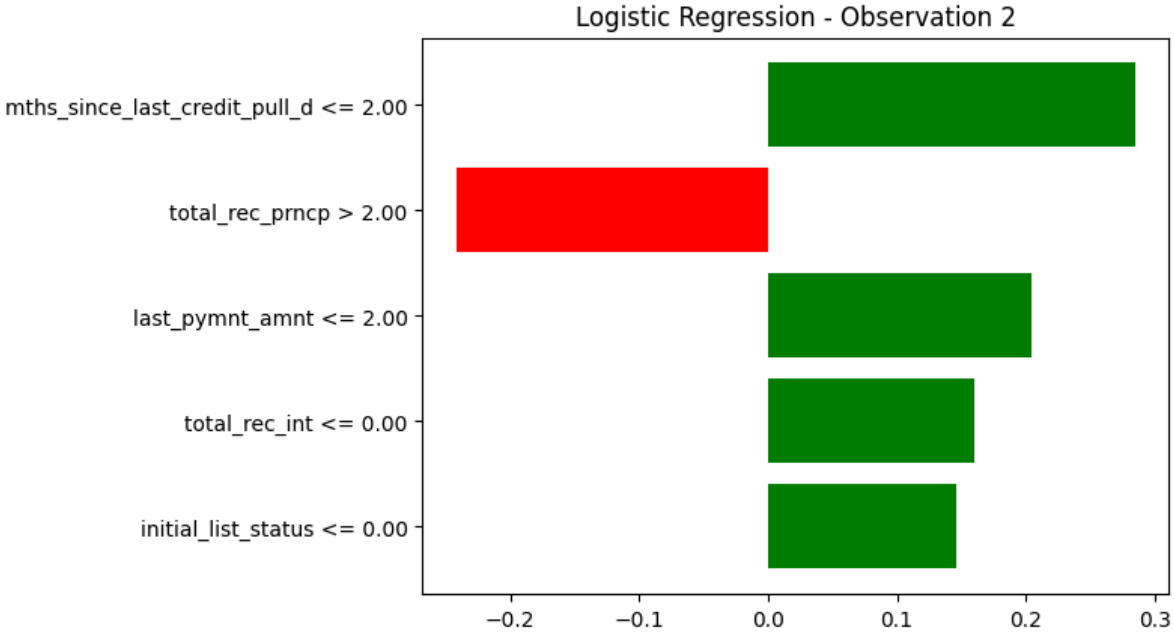

Prediction probabilities
0
0.60
1
0.40
0
1
mths_since_last_credit...
0.30
last_pymnt_amnt <=...
0.18
total_rec_int <= 0.00
0.16
0.00 < initial_list_statu...
0.14
1.00 < total_rec_prncp ...
0.11

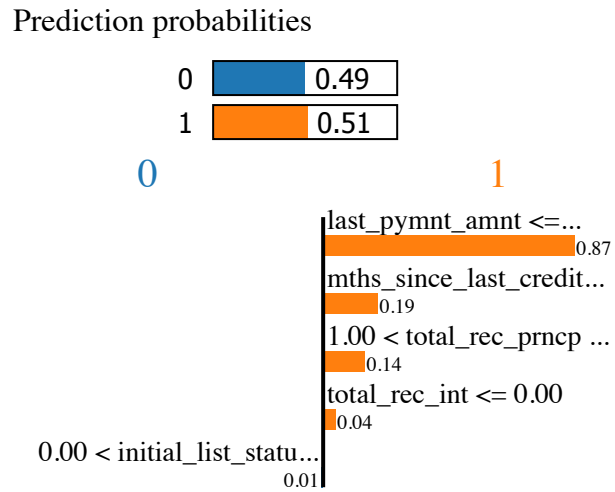

Feature
Value
mths_since_last_credit_pull_d
2.00
last_pymnt_amnt
2.00
total_rec_int
0.00
initial_list_status
1.00
total_rec_prncp
2.00

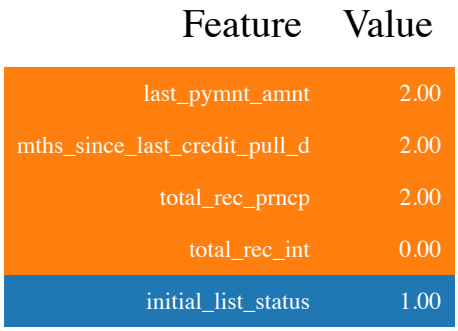

Logistic Regression - Observation 1
mths_since_last_credit_pull_d <= 2.00
last_pymnt_amnt <= 2.00
total_rec_int <= 0.00
0.00 < initial_list_status <= 1.00
1.00 < total_rec_prncp <= 2.00
−0.1
0.0
0.1
0.2
0.3

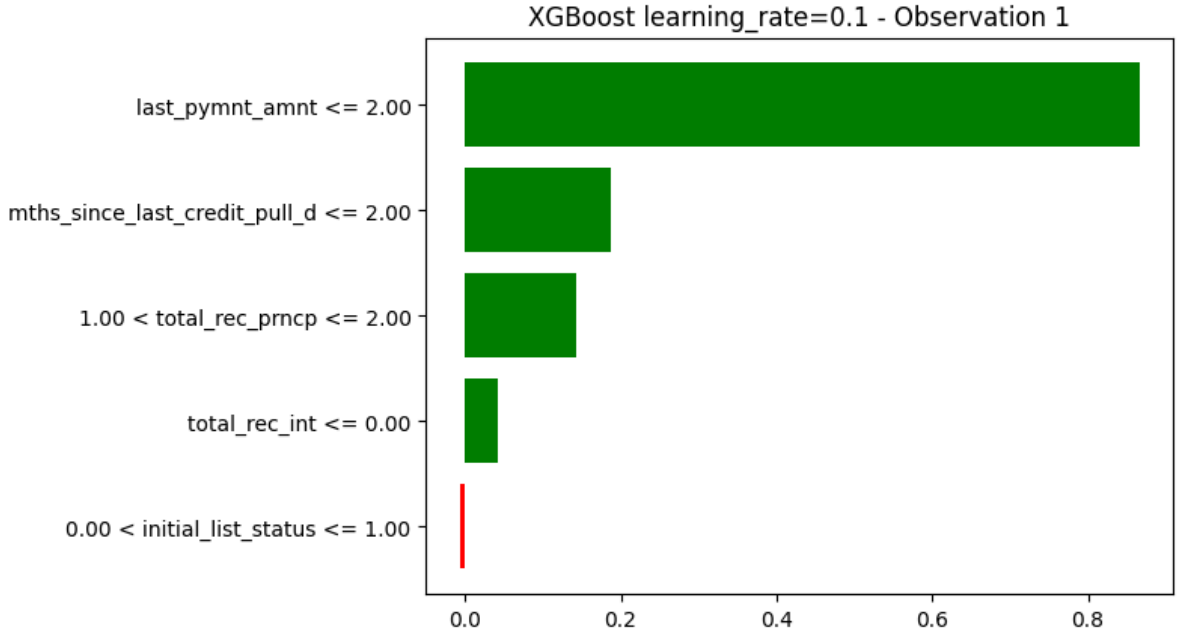

Prediction probabilities
0
0.55
1
0.45
0
1
mths_since_last_credit...
0.28
total_rec_prncp > 2.00
0.24
last_pymnt_amnt <=...
0.20
total_rec_int <= 0.00
0.16
initial_list_status <= 0.00
0.15

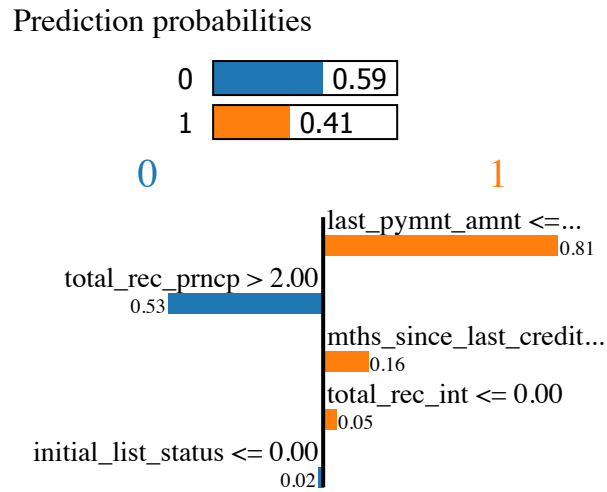

Feature
Value
mths_since_last_credit_pull_d
2.00
total_rec_prncp
3.00
last_pymnt_amnt
2.00
total_rec_int
0.00
initial_list_status
0.00

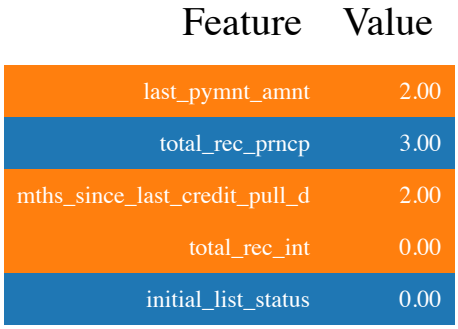

Logistic Regression - Observation 2
mths_since_last_credit_pull_d <= 2.00
total_rec_prncp > 2.00
last_pymnt_amnt <= 2.00
total_rec_int <= 0.00
initial_list_status <= 0.00
−0.2
−0.1
0.0
0.1
0.2
0.3

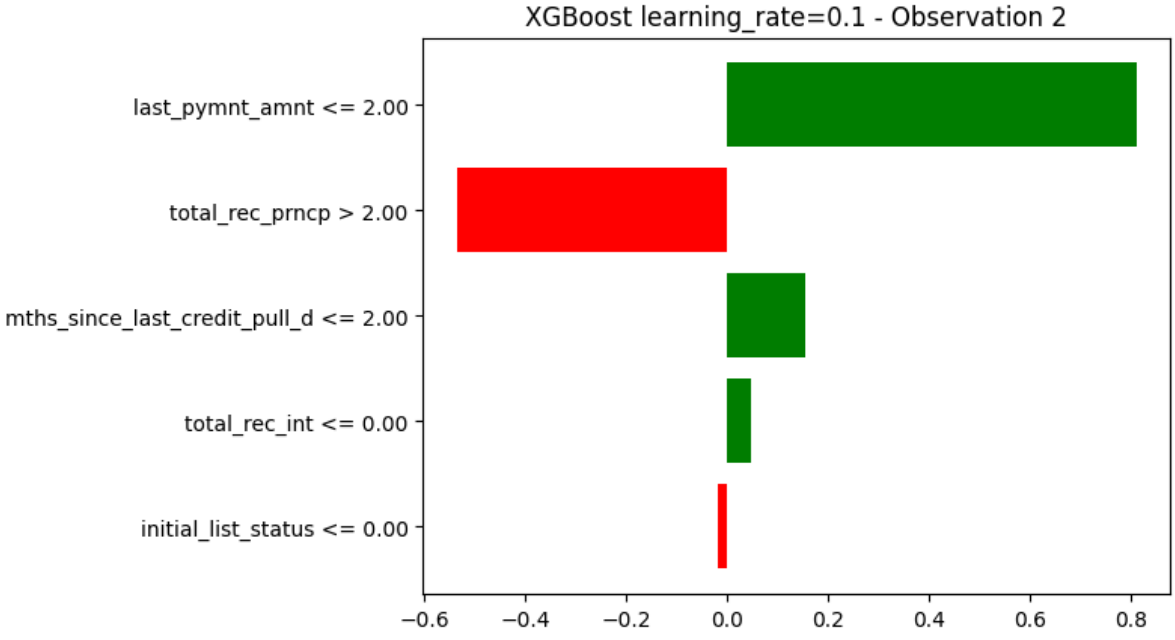

Prediction probabilities
0
0.49
1
0.51
0
1
last_pymnt_amnt <=...
0.87
mths_since_last_credit...
0.19
1.00 < total_rec_prncp ...
0.14
total_rec_int <= 0.00
0.04
0.00 < initial_list_statu...
0.01


Feature Value
last_pymnt_amnt 2.00
mths_since_last_credit_pull_d 2.00
total_rec_prncp 2.00
total_rec_int 0.00
initial_list_status 1.00


XGBoost learning_rate=0.1 - Observation 1
last_pymnt_amnt <= 2.00
mths_since_last_credit_pull_d <= 2.00
1.00 < total_rec_prncp <= 2.00
total_rec_int <= 0.00
0.00 < initial_list_status <= 1.00
0.0
0.2
0.4
0.6
0.8


Prediction probabilities
0
0.59
1
0.41
0
1
last_pymnt_amnt <=...
0.81
total_rec_prncp > 2.00
0.53
mths_since_last_credit...
0.16
total_rec_int <= 0.00
0.05
initial_list_status <= 0.00
0.02


Feature
Value
last_pymnt_amnt
2.00
total_rec_prncp
3.00
mths_since_last_credit_pull_d
2.00
total_rec_int
0.00
initial_list_status
0.00


XGBoost learning_rate=0.1 - Observation 2
last_pymnt_amnt <= 2.00
total_rec_prncp > 2.00
mths_since_last_credit_pull_d <= 2.00
total_rec_int <= 0.00
initial_list_status <= 0.00
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8

In [ ]: