By hand Complex

In [49]:
```
import pandas as pd
pd.set_option("max_colwidth", None)

import pycaret
import numpy as np
import matplotlib.pyplot as plt
from pycaret.classification import *
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

from functions.homebrew import *
import numpy as np
import pandas as pd

from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA, QuadraticDiscriminantAnalysis as QDA
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score
from tqdm import tqdm
from itertools import combinations
import pickle
import os

# If you're using statsmodels or ISLP for specific tasks, keep these imports
import statsmodels.api as sm
# Assuming ISLP and homebrew are custom modules specific to your project
from ISLP import load_data, confusion_table
from ISLP.models import ModelSpec as MS, summarize, contrast
import statsmodels.api as sm
from scipy import stats
```

Helper Functions

In [50]:
```
def add_transformations(data, cont_cols):
    for var in cont_cols:
        data[f'log_{var}'] = np.log(data[var] + 1)
        data[f'sq_{var}'] = data[var]**2
        data[f'sqrt_{var}'] = np.sqrt(data[var])
        data[f'inv_{var}'] = 1 / (data[var] + 1)
        data[f'boxcox_{var}'], _ = stats.boxcox(data[var] + 1)
        data[f'sigmoid_{var}'] = 1 / (1 + np.exp(-data[var]))
        data[f'sin_{var}'] = np.sin(data[var])
        data[f'cos_{var}'] = np.cos(data[var])
```

In [51]:
```python
def convert_confusion_matrix(df, name):
    """
    Converts a confusion matrix dataframe into a format with columns for model name, TP, TN, FP, FN.

    Args:
    df (pd.DataFrame): Confusion matrix dataframe with multi-index (Truth, Predicted) and columns [0, 1].

    Returns:
    pd.DataFrame: Reformatted dataframe with model evaluation metrics.
    """
    # Extracting the values from the confusion matrix
    tn, fp, fn, tp = df.iloc[0, 0], df.iloc[0, 1], df.iloc[1, 0], df.iloc[1, 1]
    acc = (tp + tn) / (tp + tn + fp + fn)
    prec = tp / (tp +fp)
    recall = tp / (tp + fn)
    f1 = 2 * ((prec * recall)/(prec + recall))
    # Creating a new dataframe with the desired format
    metrics_df = pd.DataFrame({
        "name": name,
        "tp": [tp],
        "tn": [tn],
        "fp": [fp],
        "fn": [fn],
        'acc': acc,
        'prec': prec,
        'recall': recall,
        'f1': f1
    })

    return metrics_df
```

In [52]:
```python
def format_results(df):
    df = np.where(df == 1, 'Donor','No Donor')
    return df
```

LOAD DATA

In [53]:
```python
df = pd.read_csv('./data/df.csv').drop('Unnamed: 0', axis=1)
```

In [54]:
```python
train = df[df['type'] == 'train'].drop('type',axis =1)
dev = df[df['type'] == 'dev'].drop('type',axis =1)
test = df[df['type'] == 'test'].drop('type',axis =1)
```

VIF

In [7]:
```python
dummies = pd.get_dummies(df, drop_first=True)

cat_cols = [
    'zipconvert2_Yes', 'zipconvert3_Yes', 'zipconvert4_Yes', 'boxcox_zipconvert5_Yes',
    'homeowner_Yes', 'female_Yes', 'type_train', 'type_dev', 'type_test'
]

cont_cols = [col for col in dummies.columns if col not in cat_cols + ['target']]
add_transformations(dummies, cont_cols)

kept, removed = remove_high_vif_features(X=dummies.drop('target_No Donor', axis=1), y=dummies['target_No Donor'], vif_threshol
print('REMOVED:', removed)
```

```
REMOVED: ['num_child', 'income', 'cos_target_No Donor', 'sin_target_No Donor', 'sigmoid_target_No Donor', 'boxcox_target_No Dono
r', 'inv_target_No Donor', 'sqrt_target_No Donor', 'sq_target_No Donor', 'cos_zipconvert5_Yes', 'sin_zipconvert5_Yes', 'sigmoid_z
ipconvert5_Yes', 'boxcox_zipconvert5_Yes', 'inv_zipconvert5_Yes', 'sqrt_zipconvert5_Yes', 'sq_zipconvert5_Yes', 'log_zipconvert5_
Yes', 'log_income', 'sq_income', 'log_num_child', 'sq_num_child', 'sigmoid_avg_fam_inc', 'inv_num_child', 'sqrt_months_since_dona
te', 'sqrt_med_fam_inc', 'sq_wealth', 'boxcox_avg_fam_inc', 'boxcox_num_prom', 'log_last_gift', 'months_since_donate', 'log_avg_g
ift', 'inv_home_value', 'inv_avg_fam_inc', 'boxcox_time_lag', 'log_wealth', 'inv_med_fam_inc', 'log_largest_gift', 'boxcox_home_v
alue', 'sqrt_num_prom', 'log_lifetime_gifts', 'sqrt_income', 'boxcox_pct_lt15k', 'sqrt_wealth', 'sq_months_since_donate', 'sqrt_a
vg_fam_inc', 'sqrt_avg_gift', 'boxcox_med_fam_inc', 'sqrt_time_lag', 'boxcox_largest_gift', 'sqrt_pct_lt15k', 'inv_income', 'sqrt
_home_value', 'sigmoid_num_child', 'log_num_prom', 'sqrt_last_gift', 'log_months_since_donate', 'boxcox_avg_gift', 'sqrt_lifetime
_gifts', 'avg_fam_inc', 'wealth', 'largest_gift', 'med_fam_inc', 'boxcox_last_gift', 'inv_pct_lt15k', 'zipconvert5_Yes', 'num_pro
m', 'pct_lt15k', 'home_value', 'log_avg_fam_inc', 'inv_wealth', 'log_time_lag', 'boxcox_lifetime_gifts', 'sqrt_num_child', 'avg_g
ift', 'inv_time_lag', 'log_med_fam_inc', 'boxcox_income', 'inv_largest_gift', 'inv_last_gift', 'sq_avg_fam_inc', 'inv_num_prom',
'last_gift']
```

In [55]:
```python
final_vars = list(kept.corr().drop('target')[np.abs(kept.corr()['target'].drop('target')) > .05].index)
```

In [56]:
```python
regress = kept[final_vars]
```

In [57]:
```
regress['target'] = (df['target'] == 'Donor').astype(int)
```

In [72]:
```
kept = kept.drop('log_target_No Donor', axis =1)
```

In [59]:
```
train = kept[regress['type_train'] ==1]
dev = kept[(regress['type_test'] == 0) & (regress['type_train'] == 0)]
test = kept[regress['type_test'] ==1]
```

In [60]:
```
for data in [train, dev, test]:
    data.drop('type_train', inplace = True, axis = 1)
    data.drop('type_test', inplace = True, axis = 1)
test = test.drop('target', axis =1)
```

Logistic Regression

In [61]:
```
results_df = pd.DataFrame()
```

In [62]:
```
# Selecting features and target variable for training data
X_train = train.drop(['target'], axis =1 )
y_train = train['target']
X_test = dev.drop(['target'], axis = 1)
y_test = dev['target']

# Fitting logistic regression model
glm = sm.GLM(y_train, X_train, family=sm.families.Binomial())
glm = glm.fit()

# Summarizing results
# print(results.summary())
```

In [63]:
```
log_preds = (glm.predict(X_test) >= 0.5).astype(int)
log_acc = accuracy_score(log_preds, y_test)
print(log_acc)

d = confusion_table(log_preds,y_test)
results_df = pd.concat([results_df,convert_confusion_matrix(d, 'Logistic Regression')])

log_test_preds = (glm.predict(test) >= 0.5).astype(int)
log_test_preds = format_results(log_test_preds)

save_df = pd.DataFrame(log_test_preds, columns=['values'])
save_df.to_csv('./preds/log.csv', index=False)
```

```
0.5116666666666667
```

LDA

In [64]:
```
lda = LDA(store_covariance=True)
lda.fit(X_train, y_train)

lda_preds = lda.predict(X_test)

lda_acc = accuracy_score(lda_preds,y_test)
print(lda_acc)

d = confusion_table(lda_preds,y_test)
results_df = pd.concat([results_df,convert_confusion_matrix(d, 'LDA')])


lda_test_preds = (lda.predict(test) >= 0.5).astype(int)
lda_test_preds = format_results(lda_test_preds)

save_df = pd.DataFrame(lda_test_preds, columns=['values'])
save_df.to_csv('./preds/lda.csv', index=False)
```

```
0.5066666666666667
```

QDA.

In [65]:
```
qda = QDA(store_covariance=True)
qda.fit(X_train, y_train)

qda_preds = qda.predict(X_test)

qda_acc = accuracy_score(qda_preds,y_test)

print(qda_acc)

d = confusion_table(qda_preds,y_test)
results_df = pd.concat([results_df,convert_confusion_matrix(d, 'QDA')])

qda_test_preds = (qda.predict(test) >= 0.5).astype(int)
qda_test_preds = format_results(qda_test_preds)

save_df = pd.DataFrame(qda_test_preds, columns=['values'])
save_df.to_csv('./preds/qda.csv', index=False)
```

```
0.49166666666666664
```

KNN

In [66]:
```
knn1 = KNeighborsClassifier(n_neighbors=1)
knn1.fit(X_train, y_train)
knn1_pred = knn1.predict(X_test)
knn1_acc = accuracy_score(knn1_pred,y_test)

print(knn1_acc)

d = confusion_table(knn1_pred, y_test)
results_df = pd.concat([results_df,convert_confusion_matrix(d, 'KNN')])

knn1_test_preds = (knn1.predict(test) >= 0.5).astype(int)
knn1_test_preds = format_results(knn1_test_preds)

save_df = pd.DataFrame(knn1_test_preds, columns=['values'])
save_df.to_csv('./preds/knn1.csv', index=False)
```

```
0.4816666666666667
```

NB

In [67]:
```
nb = GaussianNB()
nb.fit(X_train, y_train)
nb_preds = nb.predict(X_test)
nb_acc = accuracy_score(nb_preds,y_test)

print(nb_acc)
save_df = pd.DataFrame(nb_preds, columns=['values'])
save_df.to_csv('./preds/nb.csv', index=False)

d = confusion_table(nb_preds, y_test)
results_df = pd.concat([results_df,convert_confusion_matrix(d, 'Naïve Bayes')])

nb_test_preds = (nb.predict(test) >= 0.5).astype(int)
nb_test_preds = format_results(nb_test_preds)

save_df = pd.DataFrame(nb_test_preds, columns=['values'])
save_df.to_csv('./preds/nb.csv', index=False)
```

```
0.495
```

In [68]:
```
results_df
```

Out[68]:

	name	tp	tn	fp	fn	acc	prec	recall	f1
0	Logistic Regression	159	148	153	140	0.511667	0.509615	0.531773	0.520458
0	LDA	156	148	156	140	0.506667	0.500000	0.527027	0.513158
0	QDA	26	269	286	19	0.491667	0.083333	0.577778	0.145658
0	KNN	140	149	172	139	0.481667	0.448718	0.501792	0.473773
0	Naïve Bayes	18	279	294	9	0.495000	0.057692	0.666667	0.106195

In [69]:
```
# test_acc = {
#     'log': 0.5333333,
#     'lda': 0.5583333,
#     'qda': 0.525,
#     'knn':  0.475,
#     'nb': 0.5166667,
# }
```

In [70]:
```
# results_df['test_acc'] = test_acc.values()
```

In [71]:
```
results_df
```

Out[71]:

	name	tp	tn	fp	fn	acc	prec	recall	f1	test_acc
0	Logistic Regression	159	148	153	140	0.511667	0.509615	0.531773	0.520458	0.533333
0	LDA	156	148	156	140	0.506667	0.500000	0.527027	0.513158	0.558333
0	QDA	26	269	286	19	0.491667	0.083333	0.577778	0.145658	0.525000
0	KNN	140	149	172	139	0.481667	0.448718	0.501792	0.473773	0.475000
0	Naïve Bayes	18	279	294	9	0.495000	0.057692	0.666667	0.106195	0.516667