Project

April 25, 2025

0.1 Marriage Trends in India

0.2 Import Python Libraries and Data

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from imblearn.over_sampling import SMOTE
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA
from sklearn.metrics import classification_report, accuracy_score, f1_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV

data = pd.read_csv('marriage_data_india.csv')
display(data.head())
```

```
   ID Marriage_Type  Age_at_Marriage  Gender Education_Level Caste_Match  \
0   1          Love               23    Male        Graduate   Different
1   2          Love               28  Female          School        Same
2   3      Arranged               39    Male    Postgraduate        Same
3   4      Arranged               26  Female          School   Different
4   5          Love               32  Female        Graduate        Same

  Religion Parental_Approval Urban_Rural Dowry_Exchanged Marital_Satisfaction  \
```

```
0    Hindu              No      Urban          No          Medium
1    Hindu             Yes      Rural         Yes             Low
2   Muslim             Yes      Rural          No          Medium
3    Hindu             Yes      Urban         Yes             Low
4    Hindu         Partial      Rural         Yes          Medium

  Divorce_Status  Children_Count Income_Level  Years_Since_Marriage  \
0            Yes               5       Middle                    34
1             No               3       Middle                    42
2             No               0         High                    25
3             No               0         High                    12
4             No               1       Middle                    41

  Spouse_Working Inter-Caste Inter-Religion
0             No          No             No
1             No          No            Yes
2             No          No             No
3             No         Yes             No
4             No          No            Yes
```

0.3 Check for Null Values Within the Entire Dataset

[2]:
```python
display(data.isna().sum())
```

```
ID                      0
Marriage_Type           0
Age_at_Marriage         0
Gender                  0
Education_Level         0
Caste_Match             0
Religion                0
Parental_Approval       0
Urban_Rural             0
Dowry_Exchanged         0
Marital_Satisfaction    0
Divorce_Status          0
Children_Count          0
Income_Level            0
Years_Since_Marriage    0
Spouse_Working          0
Inter-Caste             0
Inter-Religion          0
dtype: int64
```

0.4 Do Some Initial Data Cleaning

[3]:
```
data.rename(columns={'Inter-Caste':'Inter_Caste', 'Inter-Religion':
 ↪'Inter_Religion'}, inplace=True)
data.drop(columns=['ID'], inplace=True)
```

0.5 Separate the Column Labels by Numeric and Categorical Data

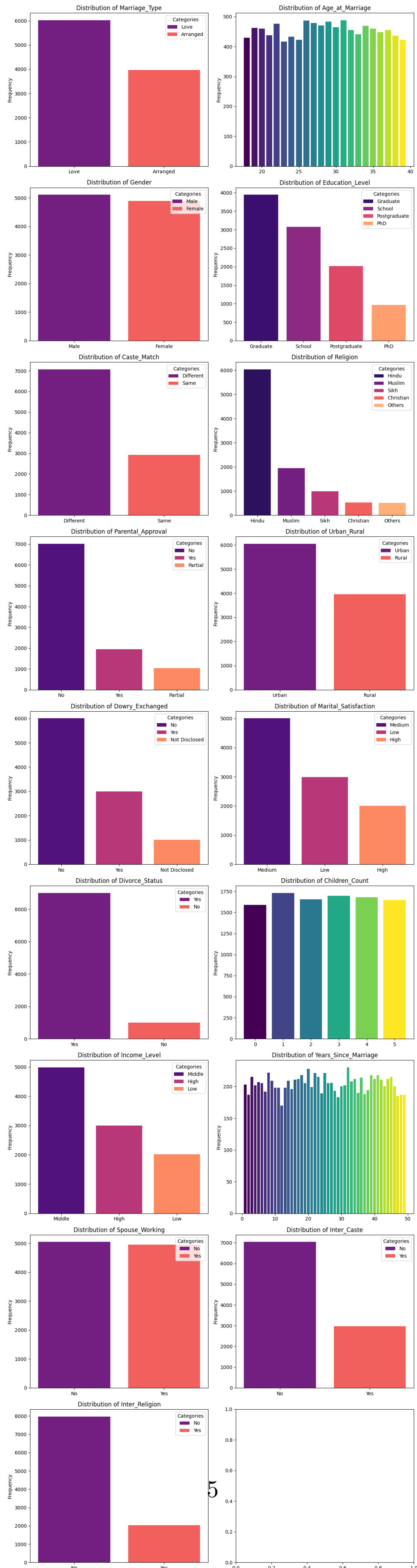
[4]:
```
cols_categorical = data.select_dtypes(include=['object']).columns.tolist()
print(f"Categorical Columns:\n{cols_categorical}")

cols_numeric = data.select_dtypes(include=[np.number]).columns.tolist()
print(f"Numeric Columns:\n{cols_numeric}")
```

```
Categorical Columns:
['Marriage_Type', 'Gender', 'Education_Level', 'Caste_Match', 'Religion',
'Parental_Approval', 'Urban_Rural', 'Dowry_Exchanged', 'Marital_Satisfaction',
'Divorce_Status', 'Income_Level', 'Spouse_Working', 'Inter_Caste',
'Inter_Religion']
Numeric Columns:
['Age_at_Marriage', 'Children_Count', 'Years_Since_Marriage']
```

0.6 Graph the Basic Counts of the Data

[5]:

```
fig, axes = plt.subplots(nrows=(len(data.columns) + 1) // 2, ncols=2,␣
↪figsize=(12, 5 * ((len(data.columns) + 1) // 2)))
axes = axes.flatten()
for i, col in enumerate(data.columns):
    ax = axes[i]
    x_cats = data[col].unique()
    if col in cols_numeric:
        x_cats = np.sort(x_cats)
        y_counts = data[col].value_counts().sort_index()
        colors = plt.cm.viridis(np.linspace(0, 1, len(y_counts)))
    else:
        y_counts = data[col].value_counts()
        colors = sns.color_palette('magma', len(y_counts))
    ax.bar(x_cats, y_counts, color=colors, label=x_cats)
    ax.set_ylabel('Frequency')
    ax.set_title(f'Distribution of {col}')
    if not col in cols_numeric:
        ax.legend(x_cats, title='Categories')
    extent = ax.get_window_extent().transformed(fig.dpi_scale_trans.inverted()).
↪expanded(1.3, 1.25)
    fig.savefig(f'{col}_Distribution.png', bbox_inches=extent)
plt.tight_layout()
plt.savefig('All_Distributions.png')
plt.show()
```


Distribution of Marriage_Type
Categories
Love
Arranged
Frequency
Distribution of Age_at_Marriage
Frequency
Distribution of Gender
Categories
Male
Female
Frequency
Distribution of Education_Level
Categories
Graduate
School
Postgraduate
PhD
Frequency
Distribution of Caste_Match
Categories
Different
Same
Frequency
Distribution of Religion
Categories
Hindu
Muslim
Sikh
Christian
Others
Frequency
Distribution of Parental_Approval
Categories
No
Yes
Partial
Frequency
Distribution of Urban_Rural
Categories
Urban
Rural
Frequency
Distribution of Dowry_Exchanged
Categories
No
Yes
Not Disclosed
Frequency
Distribution of Marital_Satisfaction
Categories
Medium
Low
High
Frequency
Distribution of Divorce_Status
Categories
Yes
No
Frequency
Distribution of Children_Count
Frequency
Distribution of Income_Level
Categories
Middle
High
Low
Frequency
Distribution of Years_Since_Marriage
Frequency
Distribution of Spouse_Working
Categories
No
Yes
Frequency
Distribution of Inter_Caste
Categories
No
Yes
Frequency
Distribution of Inter_Religion
Categories
No
Yes
Frequency

0.7 Separate the Data Out to X and y Variables

[6]:
```
X = data.drop(columns=['Divorce_Status'])
y = data['Divorce_Status']
```

0.8 One Hot Encode the Categorical Columns

[7]:
```
cols_categorical.remove('Divorce_Status')
X = pd.get_dummies(X, columns=cols_categorical, drop_first=True)
display(X.head())
```

```
   Age_at_Marriage  Children_Count  Years_Since_Marriage  Marriage_Type_Love  \
0               23               5                    34                True
1               28               3                    42                True
2               39               0                    25               False
3               26               0                    12               False
4               32               1                    41                True

   Gender_Male  Education_Level_PhD  Education_Level_Postgraduate  \
0         True                False                         False
1        False                False                         False
2         True                False                          True
3        False                False                         False
4        False                False                         False

   Education_Level_School  Caste_Match_Same  Religion_Hindu  …  \
0                   False             False            True  …
1                    True              True            True  …
2                   False              True           False  …
3                    True             False            True  …
4                   False              True            True  …

   Urban_Rural_Urban  Dowry_Exchanged_Not Disclosed  Dowry_Exchanged_Yes  \
0               True                          False                False
1              False                          False                 True
2              False                          False                False
3               True                          False                 True
4              False                          False                 True

   Marital_Satisfaction_Low  Marital_Satisfaction_Medium  Income_Level_Low  \
0                     False                         True             False
1                      True                        False             False
2                     False                         True             False
3                      True                        False             False
4                     False                         True             False
```

```
   Income_Level_Middle  Spouse_Working_Yes  Inter_Caste_Yes  \
0                 True               False            False
1                 True               False            False
2                False               False            False
3                False               False             True
4                 True               False            False

   Inter_Religion_Yes
0               False
1                True
2               False
3               False
4                True

[5 rows x 25 columns]
```

0.9 Label Encode the y Vector

[8]:
```python
le = LabelEncoder()
y = le.fit_transform(y)
```

0.10 Use SMOTE to Balance the Data

[9]:
```python
print(f"Number of '0' class instances in y: {sum(y==0)}")
print(f"Number of '1' class instances in y: {sum(y==1)}")

smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

print(f"Number of '0' class instances in y after resample:␣
  ↪{sum(y_resampled==0)}")
print(f"Number of '1' class instances in y after resample:␣
  ↪{sum(y_resampled==1)}")
```

```
Number of '0' class instances in y: 8999
Number of '1' class instances in y: 1001
Number of '0' class instances in y after resample: 8999
Number of '1' class instances in y after resample: 8999
```

0.11 Perform Basic Logistic Regression Before Backwards Elimination

[10]:
```python
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,␣
  ↪test_size=.2, random_state=42, stratify=y_resampled)
scaler = StandardScaler()
X_train[cols_numeric] = scaler.fit_transform(X_train[cols_numeric])
X_test[cols_numeric] = scaler.transform(X_test[cols_numeric])
```

```
clsfrName = "Pre-PCA Logistic Regression"
classifier = LogisticRegression(random_state=42)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
print(f"Classification Report for Logistic Regression for Testing Set:\n")
print(classification_report(y_test, y_pred))

basic_lr = (clsfrName, classifier)
class_scores_columns = ['ModelName', 'Classifier Score', 'Cross Validation␣
↪Mean', 'Cross Validation Standard Deviation', 'F1-Score']
crossVal = cross_val_score(estimator=classifier, X=X_train, y=y_train, cv=10)
f1 = f1_score(y_test, y_pred)
basic_lr_scores = pd.DataFrame([[clsfrName, classifier.score(X_test, y_test),␣
↪crossVal.mean(), crossVal.std(), f1]], columns=class_scores_columns)
display(basic_lr_scores)
```

```
Classification Report for Logistic Regression for Testing Set:

              precision    recall  f1-score   support

           0       0.84      0.85      0.84      1800
           1       0.84      0.83      0.84      1800

    accuracy                           0.84      3600
   macro avg       0.84      0.84      0.84      3600
weighted avg       0.84      0.84      0.84      3600


                     ModelName  Classifier Score  Cross Validation Mean  \
0  Pre-PCA Logistic Regression          0.840556               0.842686

   Cross Validation Standard Deviation  F1-Score
0                             0.010196  0.839665
```

0.12 Perform Backwards Elimination to Find the Most Important Variables in the Dataset

[11]:
```
def backwardElimination(x, y, sl):
    numVars = len(x[0])
    indices = list(range(numVars))
    for i in range(0, numVars):
        obj_OLS = sm.OLS(y, x).fit()
        maxVar = max(obj_OLS.pvalues).astype(float)
        if maxVar > sl:
            for j in range(0, len(indices)):
                if (obj_OLS.pvalues[j].astype(float) == maxVar):
                    x = np.delete(x, j, 1)
                    indices.pop(j)
```

```
                    break
    display(obj_OLS.summary())
    x = x[:, 1:]
    indices.pop(0)
    indices = [i - 1 for i in indices]
    return x, indices

SL = 0.05
X_backe = np.append(arr=np.ones((len(X_resampled),1)), values=X_resampled,␣
 ↪axis=1)
X_backe = X_backe.astype('float64')
X_sig = X_backe
X_Modeled, indices = backwardElimination(X_sig, y_resampled, SL)
print(f'Selected Features Indices: {indices}')
```

Dep. Variable:	y	**R-squared:**	0.478
Model:	OLS	**Adj. R-squared:**	0.477
Method:	Least Squares	**F-statistic:**	686.1
Date:	Fri, 25 Apr 2025	**Prob (F-statistic):**	0.00
Time:	16:18:50	**Log-Likelihood:**	-7210.3
No. Observations:	17998	**AIC:**	1.447e+04
Df Residuals:	17973	**BIC:**	1.467e+04
Df Model:	24		
Covariance Type:	nonrobust		

	coef	std err	t	P> |t|	[0.025	0.975]
const	1.3211	0.015	85.457	0.000	1.291	1.351
x1	0.0011	0.000	2.605	0.009	0.000	0.002
x2	-0.0140	0.002	-8.622	0.000	-0.017	-0.011
x3	-0.0913	0.006	-15.385	0.000	-0.103	-0.080
x4	-0.1074	0.006	-19.173	0.000	-0.118	-0.096
x5	-0.2422	0.012	-20.371	0.000	-0.265	-0.219
x6	-0.2255	0.008	-26.614	0.000	-0.242	-0.209
x7	-0.1842	0.007	-26.631	0.000	-0.198	-0.171
x8	-0.0623	0.006	-10.997	0.000	-0.073	-0.051
x9	-0.2002	0.007	-30.285	0.000	-0.213	-0.187
x10	-0.2801	0.009	-29.722	0.000	-0.299	-0.262
x11	-0.3473	0.017	-20.978	0.000	-0.380	-0.315
x12	-0.3624	0.017	-21.642	0.000	-0.395	-0.330
x13	-0.2765	0.010	-28.645	0.000	-0.295	-0.258
x14	-0.1602	0.007	-23.794	0.000	-0.173	-0.147
x15	-0.0616	0.005	-11.290	0.000	-0.072	-0.051
x16	-0.1826	0.012	-15.851	0.000	-0.205	-0.160
x17	-0.1187	0.007	-17.803	0.000	-0.132	-0.106
x18	-0.2076	0.009	-23.911	0.000	-0.225	-0.191
x19	-0.1369	0.006	-23.063	0.000	-0.149	-0.125
x20	-0.1666	0.007	-22.230	0.000	-0.181	-0.152
x21	-0.1191	0.006	-18.822	0.000	-0.132	-0.107
x22	-0.0899	0.006	-16.026	0.000	-0.101	-0.079
x23	-0.1308	0.007	-19.226	0.000	-0.144	-0.118
x24	-0.1241	0.008	-15.492	0.000	-0.140	-0.108

Omnibus:	270.514	**Durbin-Watson:**	1.494
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	282.542
Skew:	0.306	**Prob(JB):**	4.43e-62
Kurtosis:	3.041	**Cond. No.**	201.

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

```
Selected Features Indices: [0, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
16, 17, 18, 19, 20, 21, 22, 23, 24]
```

0.13 Display the head of the X variable with only the 'important' indices selected

[12]:
```
X_resampled = pd.DataFrame(X_resampled, columns=X.columns[indices].to_list())
X_resampled.head()
```

[12]:
```
   Age_at_Marriage  Children_Count  Marriage_Type_Love  Gender_Male  \
0               23               5                True         True
1               28               3                True        False
2               39               0               False         True
3               26               0               False        False
```

```
4             32                1                True        False

   Education_Level_PhD  Education_Level_Postgraduate  Education_Level_School  \
0                False                         False                   False
1                False                         False                    True
2                False                          True                   False
3                False                         False                    True
4                False                         False                   False

   Caste_Match_Same  Religion_Hindu  Religion_Muslim  …  Urban_Rural_Urban  \
0             False            True            False  …               True
1              True            True            False  …              False
2              True           False             True  …              False
3             False            True            False  …               True
4              True            True            False  …              False

   Dowry_Exchanged_Not Disclosed  Dowry_Exchanged_Yes  \
0                          False                False
1                          False                 True
2                          False                False
3                          False                 True
4                          False                 True

   Marital_Satisfaction_Low  Marital_Satisfaction_Medium  Income_Level_Low  \
0                     False                         True             False
1                      True                        False             False
2                     False                         True             False
3                      True                        False             False
4                     False                         True             False

   Income_Level_Middle  Spouse_Working_Yes  Inter_Caste_Yes  \
0                 True               False            False
1                 True               False            False
2                False               False            False
3                False               False             True
4                 True               False            False

   Inter_Religion_Yes
0               False
1                True
2               False
3               False
4                True

[5 rows x 24 columns]
```

0.14 Get the numeric columns in the adjusted dataset

[13]:
```
cols_numeric = X_resampled.select_dtypes(include=[np.number]).columns.tolist()
print(f"Numeric Columns:\n{cols_numeric}")
```

```
Numeric Columns:
['Age_at_Marriage', 'Children_Count']
```

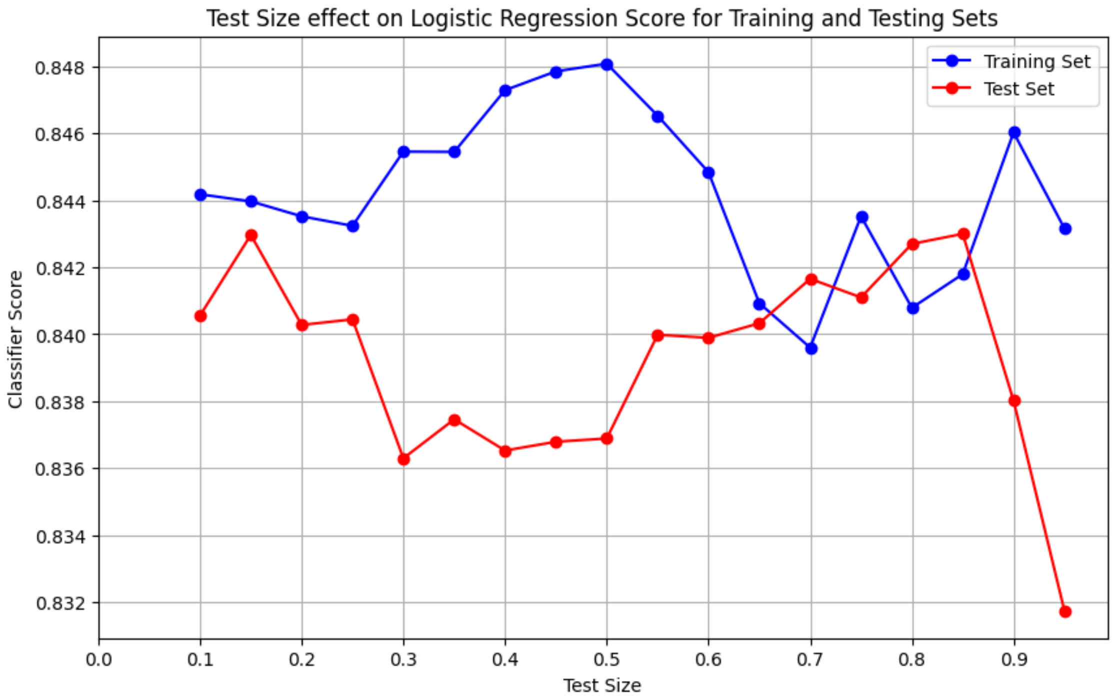
0.15 Perform a Test/Train Split Along the Data

[14]:
```
testScores = []
trainScores = []
testSampleSizes = np.arange(0.1, 1.0, 0.05)
tableVals = []
for i in testSampleSizes:
    X_train, X_test, y_train, y_test = train_test_split(X_resampled,␣
 ↪y_resampled, test_size=i, random_state=42, stratify=y_resampled)

    tableVals.append([f'{1-i:.2f}, {i:.2f}', len(X_train), len(X_test),␣
 ↪len(y_train), len(y_test)])

    scaler = StandardScaler()
    X_train[cols_numeric] = scaler.fit_transform(X_train[cols_numeric])
    X_test[cols_numeric] = scaler.transform(X_test[cols_numeric])

    lr = LogisticRegression(random_state=42)
    lr.fit(X_train, y_train)
    trainScores.append(lr.score(X_train, y_train))
    testScores.append(lr.score(X_test, y_test))

tableCols = ['Train, Test', 'X Train', 'X Test', 'y Train', 'y Test']
fig, ax = plt.subplots(figsize=(14,7))
ax.set_axis_off()
table = ax.table(cellText=tableVals, colLabels=tableCols, loc='center')
table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1, 1.5)
ax.title.set_text("Sample Size Distributions")
plt.savefig('Sample_Size_Distributions.png')
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(testSampleSizes, trainScores, marker='o', color='blue', linestyle='-',␣
 ↪label='Training Set')
plt.plot(testSampleSizes, testScores, marker='o', color='red', linestyle='-',␣
 ↪label='Test Set')
```

```python
plt.title('Test Size effect on Logistic Regression Score for Training and Testing Sets')
plt.xlabel('Test Size')
plt.ylabel('Classifier Score')
plt.xticks(np.arange(0, 1, .1))
plt.savefig('Dataset_Split_Effect_On_LR_Performance.png')
plt.legend()
plt.grid()
plt.show()
```

Sample Size Distributions

Train, Test	X Train	X Test	y Train	y Test
0.90, 0.10	16198	1800	16198	1800
0.85, 0.15	15298	2700	15298	2700
0.80, 0.20	14398	3600	14398	3600
0.75, 0.25	13498	4500	13498	4500
0.70, 0.30	12598	5400	12598	5400
0.65, 0.35	11698	6300	11698	6300
0.60, 0.40	10798	7200	10798	7200
0.55, 0.45	9898	8100	9898	8100
0.50, 0.50	8998	9000	8998	9000
0.45, 0.55	8099	9899	8099	9899
0.40, 0.60	7199	10799	7199	10799
0.35, 0.65	6299	11699	6299	11699
0.30, 0.70	5399	12599	5399	12599
0.25, 0.75	4499	13499	4499	13499
0.20, 0.80	3599	14399	3599	14399
0.15, 0.85	2699	15299	2699	15299
0.10, 0.90	1799	16199	1799	16199
0.05, 0.95	899	17099	899	17099



0.16 0.15 Seems to be the Best

[15]:
```
#Train/validation
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,␣
↪test_size=0.15, random_state=42, stratify=y_resampled)
```

0.17 Use a Standard Scaler to Scale the Numeric Data

[16]:
```
scaler = StandardScaler()
X_train[cols_numeric] = scaler.fit_transform(X_train[cols_numeric])
X_test[cols_numeric] = scaler.transform(X_test[cols_numeric])

print("X Training Set Head:")
display(X_train.head())

print("X Testing Set Head:")
display(X_test.head())
```

X Training Set Head:

	Age_at_Marriage	Children_Count	Marriage_Type_Love	Gender_Male	\
11999	1.358809	-1.399603	False	True	
10624	0.874834	-1.399603	True	False	
759	-1.706368	1.591446	False	False	
15295	0.068208	-0.203183	False	True	

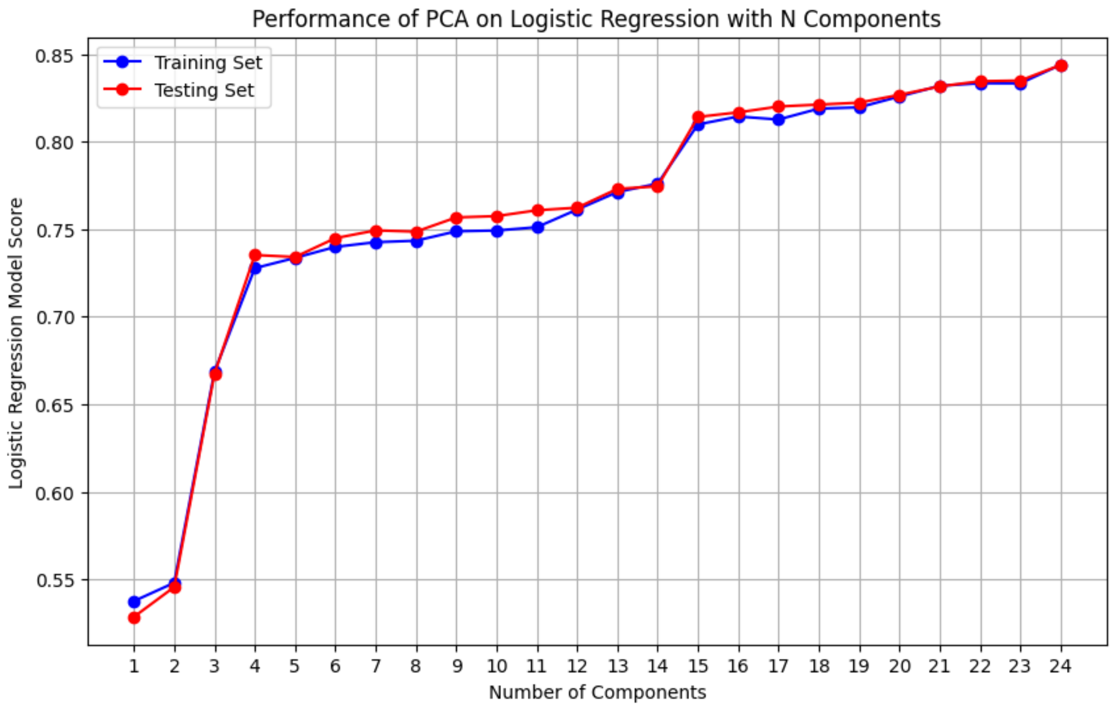
```
427          0.713509      -0.203183              False        True

       Education_Level_PhD  Education_Level_Postgraduate  \
11999                False                         False
10624                False                         False
759                  False                         False
15295                False                          True
427                  False                         False

       Education_Level_School  Caste_Match_Same  Religion_Hindu  \
11999                   False              True           False
10624                   False             False            True
759                      True              True            True
15295                   False              True            True
427                     False              True           False

       Religion_Muslim  …  Urban_Rural_Urban  Dowry_Exchanged_Not Disclosed  \
11999            False  …               True                          False
10624            False  …               True                          False
759              False  …               True                          False
15295            False  …              False                          False
427              False  …              False                          False

       Dowry_Exchanged_Yes  Marital_Satisfaction_Low  \
11999                False                     False
10624                False                     False
759                  False                     False
15295                False                     False
427                  False                     False

       Marital_Satisfaction_Medium  Income_Level_Low  Income_Level_Middle  \
11999                         True             False                False
10624                         True             False                False
759                           True             False                 True
15295                        False             False                 True
427                           True             False                 True

       Spouse_Working_Yes  Inter_Caste_Yes  Inter_Religion_Yes
11999               False            False               False
10624                True            False               False
759                  True            False               False
15295               False            False               False
427                 False            False               False

[5 rows x 24 columns]

X Testing Set Head:

       Age_at_Marriage  Children_Count  Marriage_Type_Love  Gender_Male  \
```

```
7222      -1.222393   -0.801393              True        True
11027      0.874834    0.395026             False       False
78         0.552184    1.591446             False        True
17457     -1.222393   -1.399603             False       False
15819     -0.577092   -0.203183             False       False

       Education_Level_PhD  Education_Level_Postgraduate  \
7222                 False                         False
11027                False                         False
78                   False                         False
17457                False                         False
15819                False                          True

       Education_Level_School  Caste_Match_Same  Religion_Hindu  \
7222                    False              True            True
11027                    True              True           False
78                      False              True            True
17457                   False              True           False
15819                   False             False            True

       Religion_Muslim  …  Urban_Rural_Urban  Dowry_Exchanged_Not Disclosed  \
7222             False  …               True                          False
11027            False  …              False                          False
78               False  …               True                          False
17457            False  …              False                          False
15819            False  …              False                          False

       Dowry_Exchanged_Yes  Marital_Satisfaction_Low  \
7222                 False                     False
11027                False                     False
78                   False                     False
17457                False                     False
15819                False                     False

       Marital_Satisfaction_Medium  Income_Level_Low  Income_Level_Middle  \
7222                          True              True                False
11027                         True             False                 True
78                           False             False                 True
17457                        False             False                 True
15819                        False             False                 True

       Spouse_Working_Yes  Inter_Caste_Yes  Inter_Religion_Yes
7222                False            False                True
11027               False            False               False
78                  False            False               False
17457               False            False               False
15819                True             True               False
```

[5 rows x 24 columns]

0.18 Perform PCA on the Prepped Data

[17]:
```
train_scores = []
test_scores = []
upperLimit = len(X_train.columns) + 1
for i in range(1, upperLimit):
    pca = PCA(n_components=i)
    X_train_pca = pca.fit_transform(X_train)
    X_test_pca = pca.transform(X_test)

    lr = LogisticRegression()
    lr.fit(X_train_pca, y_train)
    train_scores.append(lr.score(X_train_pca, y_train))
    test_scores.append(lr.score(X_test_pca, y_test))


plt.figure(figsize=(10, 6))
plt.plot(range(1, upperLimit), train_scores, marker='o', color='blue',␣
 ↪linestyle='-', label='Training Set')
plt.plot(range(1, upperLimit), test_scores, marker='o', color='red',␣
 ↪linestyle='-', label='Testing Set')
plt.title('Performance of PCA on Logistic Regression with N Components')
plt.xlabel('Number of Components')
plt.ylabel('Logistic Regression Model Score')
plt.xticks(range(1, upperLimit))
plt.grid()
plt.savefig('PCA_Performance_By_N_Components.png')
plt.legend()
plt.show()
```



0.19 Use 15 as the Number of Principal Components

[18]:
```
pca = PCA(n_components=15)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
```

0.20 Build a Reusable Method for Training and Testing Models

[19]:
```
class_scores = basic_lr_scores.copy()
def ModelTrainFitAndReport(classifier, classifierName, scoresDataFrame):
    classifier.fit(X_train_pca, y_train)
    y_pred = classifier.predict(X_test_pca)
    print(f"Classification Report for {classifierName} for Testing Set:\n")
    print(classification_report(y_test, y_pred))
    crossVal = cross_val_score(estimator=classifier, X=X_train_pca, y=y_train,␣
 ↪cv=10)
    f1 = f1_score(y_test, y_pred)
    scoresDataFrame.loc[-1] = [classifierName, classifier.score(X_test_pca,␣
 ↪y_test), crossVal.mean(), crossVal.std(), f1]
    scoresDataFrame.index = scoresDataFrame.index + 1
    scoresDataFrame = scoresDataFrame.sort_index()
    display(scoresDataFrame)
```

```
    return (classifierName, classifier)
```

0.21 Model Selection

0.21.1 Logistic Regression

[20]:
```
logistic_regression = ModelTrainFitAndReport(LogisticRegression(), "Logistic␣
↪Regression", class_scores)
```

```
Classification Report for Logistic Regression for Testing Set:

              precision    recall  f1-score   support

           0       0.81      0.82      0.82      1350
           1       0.82      0.81      0.81      1350

    accuracy                           0.81      2700
   macro avg       0.81      0.81      0.81      2700
weighted avg       0.81      0.81      0.81      2700


                     ModelName  Classifier Score  Cross Validation Mean  \
0          Logistic Regression          0.814074               0.810173
1  Pre-PCA Logistic Regression          0.840556               0.842686

   Cross Validation Standard Deviation  F1-Score
0                             0.013402  0.812687
1                             0.010196  0.839665
```

0.21.2 Random Forest

[21]:
```
random_forest = ModelTrainFitAndReport(RandomForestClassifier(), "Random␣
↪Forest", class_scores)
```

```
Classification Report for Random Forest for Testing Set:

              precision    recall  f1-score   support

           0       0.89      0.92      0.90      1350
           1       0.92      0.89      0.90      1350

    accuracy                           0.90      2700
   macro avg       0.90      0.90      0.90      2700
weighted avg       0.90      0.90      0.90      2700


                     ModelName  Classifier Score  Cross Validation Mean  \
0                Random Forest          0.902222               0.893125
1          Logistic Regression          0.814074               0.810173
2  Pre-PCA Logistic Regression          0.840556               0.842686
```

```
   Cross Validation Standard Deviation  F1-Score
0                             0.010368  0.900528
1                             0.013402  0.812687
2                             0.010196  0.839665
```

0.21.3 Support Vector Classifier

RBF Kernel

[22]:
```python
rbf_svm = ModelTrainFitAndReport(SVC(kernel='rbf'), "RBF SVM", class_scores)
```

```
Classification Report for RBF SVM for Testing Set:

              precision    recall  f1-score   support

           0       0.83      0.89      0.86      1350
           1       0.88      0.82      0.85      1350

    accuracy                           0.86      2700
   macro avg       0.86      0.86      0.86      2700
weighted avg       0.86      0.86      0.86      2700


                    ModelName  Classifier Score  Cross Validation Mean  \
0                     RBF SVM          0.856296               0.848740
1               Random Forest          0.902222               0.893125
2         Logistic Regression          0.814074               0.810173
3  Pre-PCA Logistic Regression          0.840556               0.842686

   Cross Validation Standard Deviation  F1-Score
0                             0.010638  0.851113
1                             0.010368  0.900528
2                             0.013402  0.812687
3                             0.010196  0.839665
```

Poly Kernel

[23]:
```python
poly_svm = ModelTrainFitAndReport(SVC(kernel='poly'), "Polynomial SVM",␣
↪class_scores)
```

```
Classification Report for Polynomial SVM for Testing Set:

              precision    recall  f1-score   support

           0       0.83      0.82      0.82      1350
           1       0.82      0.83      0.83      1350

    accuracy                           0.82      2700
   macro avg       0.82      0.82      0.82      2700
weighted avg       0.82      0.82      0.82      2700
```

```
                        ModelName  Classifier Score  Cross Validation Mean  \
0                  Polynomial SVM          0.824444               0.821546
1                         RBF SVM          0.856296               0.848740
2                   Random Forest          0.902222               0.893125
3             Logistic Regression          0.814074               0.810173
4     Pre-PCA Logistic Regression          0.840556               0.842686

   Cross Validation Standard Deviation  F1-Score
0                             0.010018  0.825863
1                             0.010638  0.851113
2                             0.010368  0.900528
3                             0.013402  0.812687
4                             0.010196  0.839665
```

0.21.4 Decision Tree Classifier

[24]:
```
decision_tree =␣
↪ModelTrainFitAndReport(DecisionTreeClassifier(criterion='entropy'),␣
↪"Decision Tree", class_scores)
```

```
Classification Report for Decision Tree for Testing Set:

              precision    recall  f1-score   support

           0       0.87      0.80      0.83      1350
           1       0.81      0.88      0.85      1350

    accuracy                           0.84      2700
   macro avg       0.84      0.84      0.84      2700
weighted avg       0.84      0.84      0.84      2700

                        ModelName  Classifier Score  Cross Validation Mean  \
0                   Decision Tree          0.838889               0.836581
1                  Polynomial SVM          0.824444               0.821546
2                         RBF SVM          0.856296               0.848740
3                   Random Forest          0.902222               0.893125
4             Logistic Regression          0.814074               0.810173
5     Pre-PCA Logistic Regression          0.840556               0.842686

   Cross Validation Standard Deviation  F1-Score
0                             0.012701  0.845030
1                             0.010018  0.825863
2                             0.010638  0.851113
3                             0.010368  0.900528
4                             0.013402  0.812687
5                             0.010196  0.839665
```

0.21.5 Naive Bayes Classifier

[25]:
```
naive_bayes = ModelTrainFitAndReport(GaussianNB(), "Gaussian Naive Bayes",␣
↪class_scores)
```

```
Classification Report for Gaussian Naive Bayes for Testing Set:

              precision    recall  f1-score   support

           0       0.78      0.81      0.80      1350
           1       0.80      0.77      0.79      1350

    accuracy                           0.79      2700
   macro avg       0.79      0.79      0.79      2700
weighted avg       0.79      0.79      0.79      2700


                       ModelName  Classifier Score  Cross Validation Mean  \
0           Gaussian Naive Bayes          0.791111               0.787294
1                  Decision Tree          0.838889               0.836581
2                 Polynomial SVM          0.824444               0.821546
3                        RBF SVM          0.856296               0.848740
4                  Random Forest          0.902222               0.893125
5            Logistic Regression          0.814074               0.810173
6  Pre-PCA Logistic Regression          0.840556               0.842686

   Cross Validation Standard Deviation  F1-Score
0                             0.012071  0.786687
1                             0.012701  0.845030
2                             0.010018  0.825863
3                             0.010638  0.851113
4                             0.010368  0.900528
5                             0.013402  0.812687
6                             0.010196  0.839665
```

0.21.6 Adaptive Boost Classifier

[26]:
```
ada_boost = ModelTrainFitAndReport(AdaBoostClassifier(), "AdaBoost",␣
↪class_scores)
```

```
Classification Report for AdaBoost for Testing Set:

              precision    recall  f1-score   support

           0       0.80      0.81      0.81      1350
           1       0.81      0.79      0.80      1350

    accuracy                           0.80      2700
   macro avg       0.80      0.80      0.80      2700
```

```
weighted avg       0.80      0.80      0.80      2700

                    ModelName  Classifier Score  Cross Validation Mean  \
0                    AdaBoost          0.803333               0.801022
1        Gaussian Naive Bayes          0.791111               0.787294
2               Decision Tree          0.838889               0.836581
3              Polynomial SVM          0.824444               0.821546
4                     RBF SVM          0.856296               0.848740
5               Random Forest          0.902222               0.893125
6         Logistic Regression          0.814074               0.810173
7  Pre-PCA Logistic Regression          0.840556               0.842686

   Cross Validation Standard Deviation  F1-Score
0                             0.014308  0.801198
1                             0.012071  0.786687
2                             0.012701  0.845030
3                             0.010018  0.825863
4                             0.010638  0.851113
5                             0.010368  0.900528
6                             0.013402  0.812687
7                             0.010196  0.839665
```

0.21.7 Gradient Boost Classifier

[27]:
```
gradient_boost = ModelTrainFitAndReport(GradientBoostingClassifier(), "Gradient Boost", class_scores)
```

```
Classification Report for Gradient Boost for Testing Set:

              precision    recall  f1-score   support

           0       0.82      0.87      0.85      1350
           1       0.86      0.81      0.84      1350

    accuracy                           0.84      2700
   macro avg       0.84      0.84      0.84      2700
weighted avg       0.84      0.84      0.84      2700


                    ModelName  Classifier Score  Cross Validation Mean  \
0              Gradient Boost          0.840741               0.832530
1                    AdaBoost          0.803333               0.801022
2        Gaussian Naive Bayes          0.791111               0.787294
3               Decision Tree          0.838889               0.836581
4              Polynomial SVM          0.824444               0.821546
5                     RBF SVM          0.856296               0.848740
6               Random Forest          0.902222               0.893125
7         Logistic Regression          0.814074               0.810173
```

```
8  Pre-PCA Logistic Regression          0.840556              0.842686

   Cross Validation Standard Deviation  F1-Score
0                             0.012898  0.836128
1                             0.014308  0.801198
2                             0.012071  0.786687
3                             0.012701  0.845030
4                             0.010018  0.825863
5                             0.010638  0.851113
6                             0.010368  0.900528
7                             0.013402  0.812687
8                             0.010196  0.839665
```

0.21.8 XGBoost Classifier

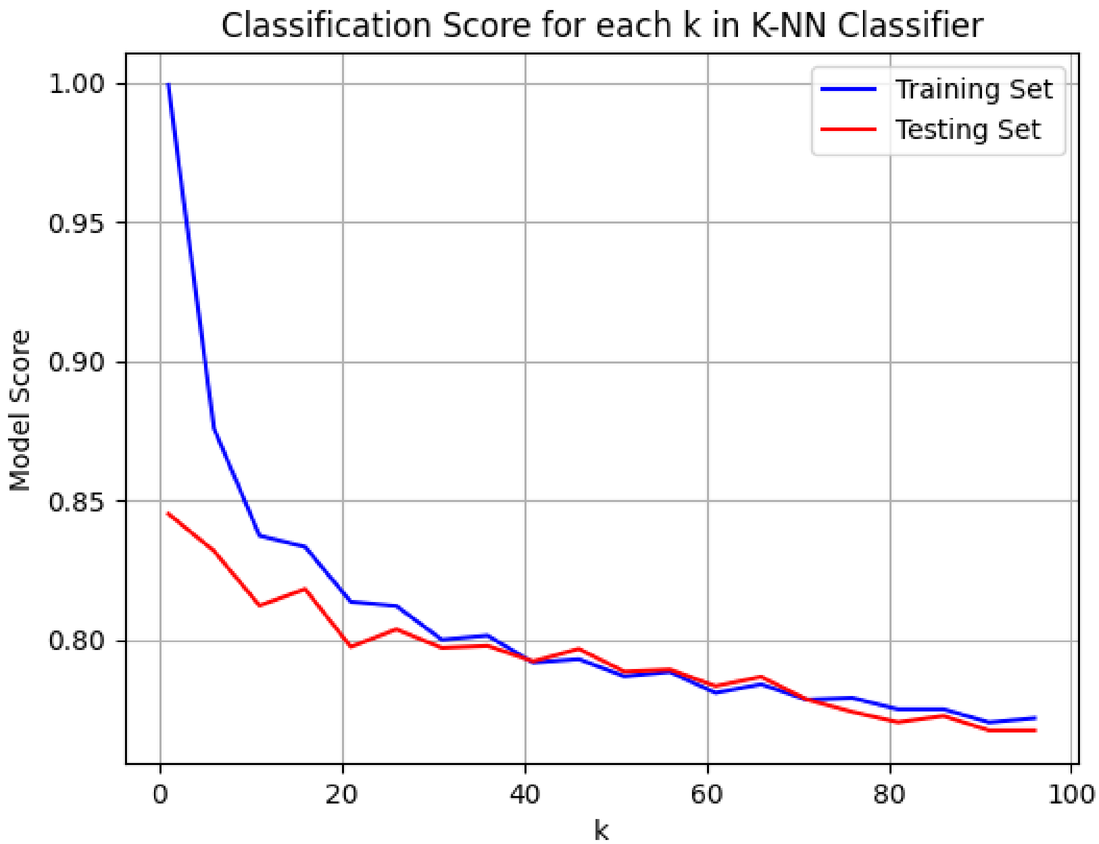
[28]:
```python
xgboost = ModelTrainFitAndReport(XGBClassifier(), "XGBoost", class_scores)
```

```
Classification Report for XGBoost for Testing Set:

              precision    recall  f1-score   support

           0       0.88      0.89      0.88      1350
           1       0.89      0.88      0.88      1350

    accuracy                           0.88      2700
   macro avg       0.88      0.88      0.88      2700
weighted avg       0.88      0.88      0.88      2700


                     ModelName  Classifier Score  Cross Validation Mean  \
0                      XGBoost          0.884444              0.879855
1               Gradient Boost          0.840741              0.832530
2                     AdaBoost          0.803333              0.801022
3         Gaussian Naive Bayes          0.791111              0.787294
4                Decision Tree          0.838889              0.836581
5               Polynomial SVM          0.824444              0.821546
6                      RBF SVM          0.856296              0.848740
7                Random Forest          0.902222              0.893125
8          Logistic Regression          0.814074              0.810173
9  Pre-PCA Logistic Regression          0.840556              0.842686

   Cross Validation Standard Deviation  F1-Score
0                             0.010313  0.883929
1                             0.012898  0.836128
2                             0.014308  0.801198
3                             0.012071  0.786687
4                             0.012701  0.845030
5                             0.010018  0.825863
6                             0.010638  0.851113
```

7	0.010368	0.900528
8	0.013402	0.812687
9	0.010196	0.839665

0.21.9 k-Nearest Neighbors Classifier

[29]:
```
train_scores = []
test_scores = []
ks = np.arange(1, 100, 5)
for k in ks:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_pca, y_train)
    train_scores.append(knn.score(X_train_pca, y_train))
    test_scores.append(knn.score(X_test_pca, y_test))

plt.plot(ks, train_scores, color='blue', label='Training Set')
plt.plot(ks, test_scores, color='red', label='Testing Set')
plt.xlabel('k')
plt.ylabel('Model Score')
plt.title('Classification Score for each k in K-NN Classifier')
plt.grid(True)
plt.legend()
plt.savefig('KNN_Scores.png')
plt.show()
```



Use best k value for k-NN

[30]:
```
knn = ModelTrainFitAndReport(KNeighborsClassifier(n_neighbors=1), "k-NN",␣
  ↪class_scores)
```

```
Classification Report for k-NN for Testing Set:

              precision    recall  f1-score   support

           0       0.91      0.77      0.83      1350
           1       0.80      0.93      0.86      1350

    accuracy                           0.85      2700
   macro avg       0.85      0.85      0.84      2700
weighted avg       0.85      0.85      0.84      2700

                 ModelName  Classifier Score  Cross Validation Mean  \
0                     k-NN          0.845185               0.838803
1                  XGBoost          0.884444               0.879855
2           Gradient Boost          0.840741               0.832530
3                 AdaBoost          0.803333               0.801022
```

```
4          Gaussian Naive Bayes          0.791111               0.787294
5                 Decision Tree          0.838889               0.836581
6                Polynomial SVM          0.824444               0.821546
7                       RBF SVM          0.856296               0.848740
8                 Random Forest          0.902222               0.893125
9           Logistic Regression          0.814074               0.810173
10  Pre-PCA Logistic Regression          0.840556               0.842686

    Cross Validation Standard Deviation  F1-Score
0                              0.010944  0.856653
1                              0.010313  0.883929
2                              0.012898  0.836128
3                              0.014308  0.801198
4                              0.012071  0.786687
5                              0.012701  0.845030
6                              0.010018  0.825863
7                              0.010638  0.851113
8                              0.010368  0.900528
9                              0.013402  0.812687
10                             0.010196  0.839665
```

0.21.10 Voting Classifier

[31]:

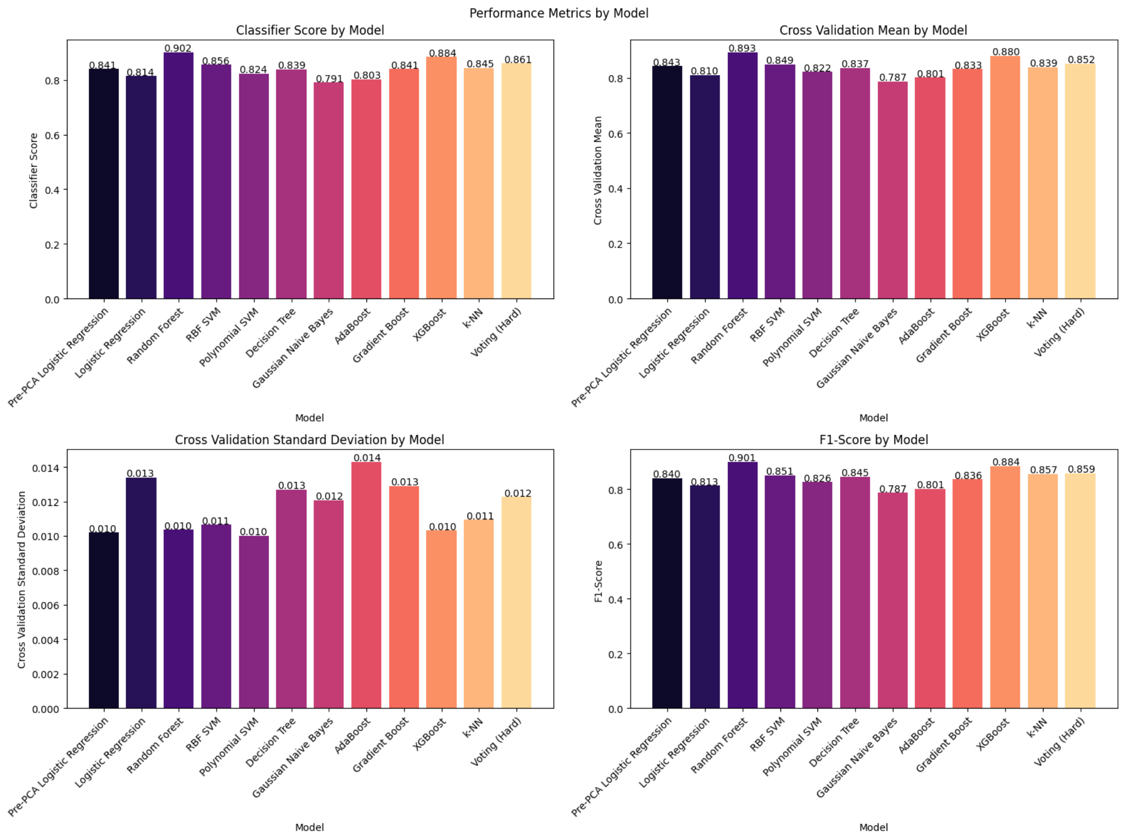
```python
classifiers = [basic_lr, logistic_regression, random_forest, rbf_svm, poly_svm,␣
 ↪decision_tree, naive_bayes, ada_boost, gradient_boost, xgboost, knn]
voting_classifier =␣
 ↪ModelTrainFitAndReport(VotingClassifier(estimators=classifiers,␣
 ↪voting='hard'), "Voting (Hard)", class_scores)
```

```
Classification Report for Voting (Hard) for Testing Set:

              precision    recall  f1-score   support

           0       0.85      0.88      0.86      1350
           1       0.87      0.85      0.86      1350

    accuracy                           0.86      2700
   macro avg       0.86      0.86      0.86      2700
weighted avg       0.86      0.86      0.86      2700


                    ModelName  Classifier Score  Cross Validation Mean  \
0               Voting (Hard)          0.861481               0.851878
1                        k-NN          0.845185               0.838803
2                     XGBoost          0.884444               0.879855
3              Gradient Boost          0.840741               0.832530
4                    AdaBoost          0.803333               0.801022
5        Gaussian Naive Bayes          0.791111               0.787294
```

```
6                Decision Tree         0.838889                  0.836581
7               Polynomial SVM         0.824444                  0.821546
8                      RBF SVM         0.856296                  0.848740
9                Random Forest         0.902222                  0.893125
10         Logistic Regression         0.814074                  0.810173
11  Pre-PCA Logistic Regression         0.840556                  0.842686

    Cross Validation Standard Deviation  F1-Score
0                              0.012249  0.859293
1                              0.010944  0.856653
2                              0.010313  0.883929
3                              0.012898  0.836128
4                              0.014308  0.801198
5                              0.012071  0.786687
6                              0.012701  0.845030
7                              0.010018  0.825863
8                              0.010638  0.851113
9                              0.010368  0.900528
10                             0.013402  0.812687
11                             0.010196  0.839665
```

0.22 Display Model Performance Metrics of Each Model

[35]:
```python
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(16, 12))
fig.suptitle('Performance Metrics by Model')
axes = axes.flatten()
for i in range(1, 5):
    ax = axes[i - 1]
    colors = sns.color_palette('magma', len(class_scores['ModelName']))
    ax.bar(class_scores['ModelName'], class_scores.iloc[:, i], color=colors)
    ax.set_ylabel(class_scores.columns[i])
    ax.set_xlabel('Model')
    ax.set_title(f'{class_scores.columns[i]} by Model')
    ax.set_xticks(np.arange(len(class_scores['ModelName'])))
    ax.set_xticklabels(class_scores['ModelName'], rotation=45, ha='right',␣
 ↪rotation_mode='anchor')
    #ax.tick_params(axis='x', labelrotation=45)
    for p in ax.patches:
      ax.annotate(f'{p.get_height():.3f}', (p.get_x() * 1.005, p.get_height() *␣
 ↪1.005))
    extent = ax.get_window_extent().transformed(fig.dpi_scale_trans.inverted()).
 ↪expanded(1.25, 1.25)
    fig.savefig(f'{col.replace(" ", "_")}_by_Model.png', bbox_inches=extent)
plt.tight_layout()
plt.savefig('All_Model_Metrics.png')
plt.show()
```



0.23 Test With New Data

[36]:
```
new_data = pd.DataFrame([[32, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0,
    0, 0, 0, 0, 1, 0, 0]], columns=X_resampled.columns)
new_data[cols_numeric] = scaler.transform(new_data[cols_numeric])
new_data = pca.transform(new_data)

# make a prediction
y_pred = voting_classifier[1].predict(new_data)

result = ""
if y_pred[0] == 1:
    result = "Will Divorce"
else:
    result = "Will Not Divorce"

print(f"Divorce Status Prediction: {result}")
```

```
Divorce Status Prediction: Will Not Divorce
```