Gagana Uday Kumar - wov796

Niharika Bandlamudi - hbz194

Nupoor Karnik- jhr497

Direct- Mail Fundraising Modelling

BACKGROUND

In this study, we wanted to develop a predictive model to improve the cost-effectiveness of a national veteran's organization for direct marketing campaigns. The organization, with its in-house database of over 13 million donors, is one of the largest direct-mail fundraisers in the United States. According to their recent mailing records, the overall response rate is 5.1%. Out of those who responded (donated), the average donation is $13.00. Each mailing, which includes a gift of personalized address labels and assortments of cards and envelopes, costs $0.68 to produce and send. Using these facts, we wanted to develop a classification model that can effectively capture donors so that the expected net profit is maximized. Weighted sampling was used so that the sample has equal numbers of donors and non-donors.

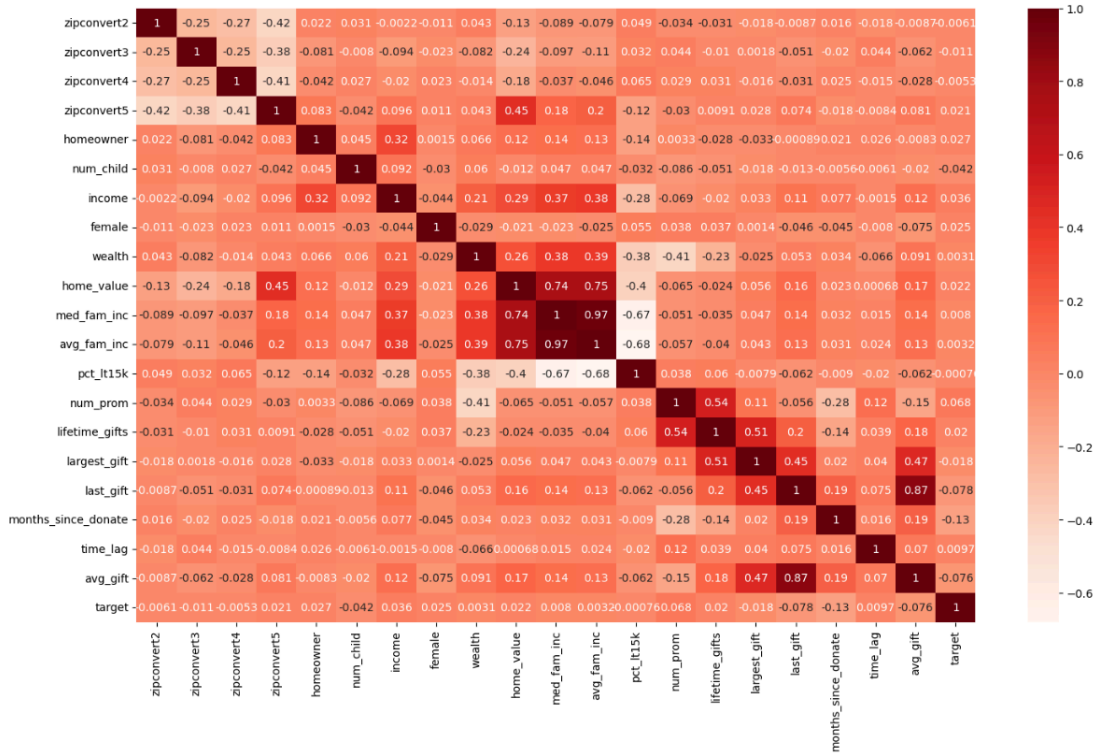
DATA EXPLORATION

The sample dataset was sourced from the national veteran's organization's recent mailing list. The 'fundraising' dataset used for this study was in a csv format. It had 3000 rows and 21 variables which we as follows zip, homeowner, num_child, income, female, wealth, home_value, med_fam_inc, avg_fam_inc, pct_lt15k, num_prom, lifetime_gifts, largest_gift, last_gift, months_since_donate, time_lag, avg_gift, and target. The target variable was an outcome variable with binary indicator for response Yes = donor, No = non-donor which we converted to 1 and 0 respectively. The 'future_fundraising.csv' file was the prediction dataset. It had 120 rows and 20 columns, same as 'fundraising.csv' excluding the 'target' variable.

After performing data transformation, we encoded object data types into integer variables for better analysis in both datasets. Following were the variables which were converted in 'fundraising.csv'- zipconvert2', zipconvert3', 'zipconvert4', 'zipconvert5', 'homeowner', 'female' and 'target'. Same variables were converted in the future_fundraising.csv as well.

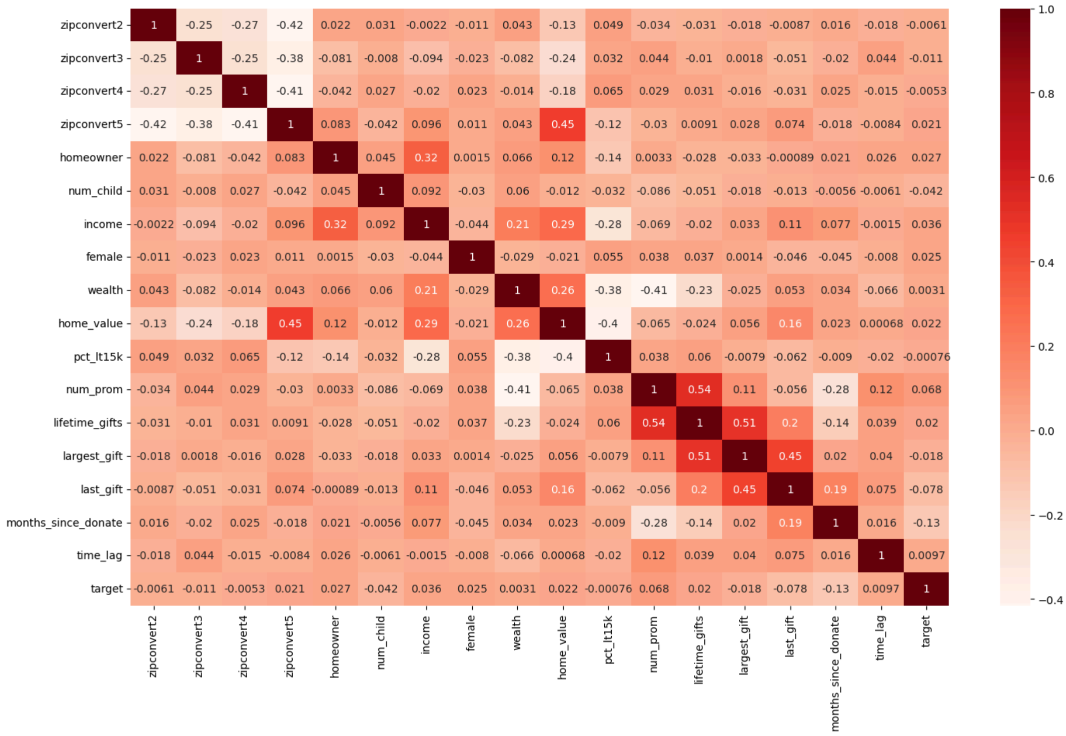
After that we performed correlation matrix on 'fundraising.csv' to find highly correlated variables if any. It was found that 'avg_fam_inc' and 'med_fam_inc' were highly correlated and 'avg_gift' was highly correlated with 'last_gift' and hence these three variables were highly correlated with other variables 'avg_fam_inc', 'med_fam_inc' and 'avg_gift' were removed as well. The correlation matrix was run again and it was found that there were no highly correlated variables. Same variables were removed from 'future_fundraising.csv' ensuring consistency in preprocessing between datasets.

Before removing the variables the correlation matrix was like:



Plot.1: Correlation Matrix

After removing the highly correlated variables the correlation matrix was as follows:



Plot.2: Revised Correlation Matrix

WEIGHTED SAMPLING

The fundraising data was divided into train and test sets using an 80-20 split. To ensure that both 'donor' and 'no donor' categories are equally represented in both training and test sets, we are employing weighted sampling. By doing so, we aim to mitigate bias and ensure better generalization of the model to unseen data. Weighted sampling allows us to train with an equal number of donors and no donors, thereby preventing potential bias towards one category during testing. Random sampling poses a risk of training on predominantly one type of variable (e.g., only donors), leaving the other type predominantly in the test set, which could lead to poor model performance.

METHODOLOGY

After cleaning the data, we perform various machine learning models like Logistic Regression, Random Forest Classifier, Gradient Boosting Classifier and Naive Bayes comparing their performance metrics such as accuracy, precision and recall to determine the most suitable model for the task.The ROC AUC plot was used as well as it shows the trade-off between true positive rate and false positive rate for different threshold settings in a binary classification model. A higher AUC indicates better overall performance in discriminating between positive and negative instances.

Model performance and Validation Results

The dataset was loaded using "pd.read_csv" and preprocessed to convert categorical variables into numerical format wherever required. Missing values were handled, and certain columns were dropped for analysis. Features as X variable and “target” as y were defined, and the dataset was split into 80% training and 20% validation sets using train_test_split. Several machine learning models, including Logistic Regression, Random Forest Classifier, Gradient Boosting Classifier, and Naive Bayes, were trained and evaluated on the validation set. Evaluation metrics such as accuracy, ROC AUC, and confusion matrix were calculated for each model.

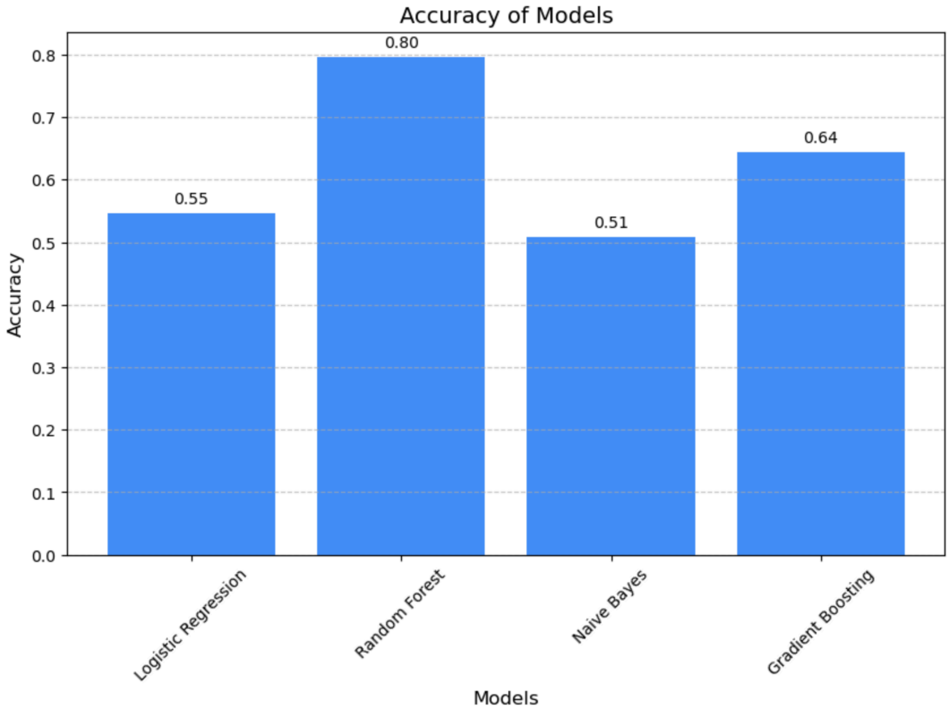
The Random Forest model performed the best, with an accuracy of 79.7% and an ROC AUC of 0.911, indicating high predictive accuracy of the classes and robustness. It also had the lowest mean squared error amongst all, suggesting that it provides more accurate predictions compared to the other models.The confusion matrix for Random Forest showed balanced performance, with few false positives and false negatives, suggesting effective classification of both positive and negative instances as compared to others. Another model that performed well was gradient boosting classifier with accuracy of 64.5% and mse of 0.355 indicating good predictive ability but has shown higher true negatives and false positives identified.

The outputs of each models:

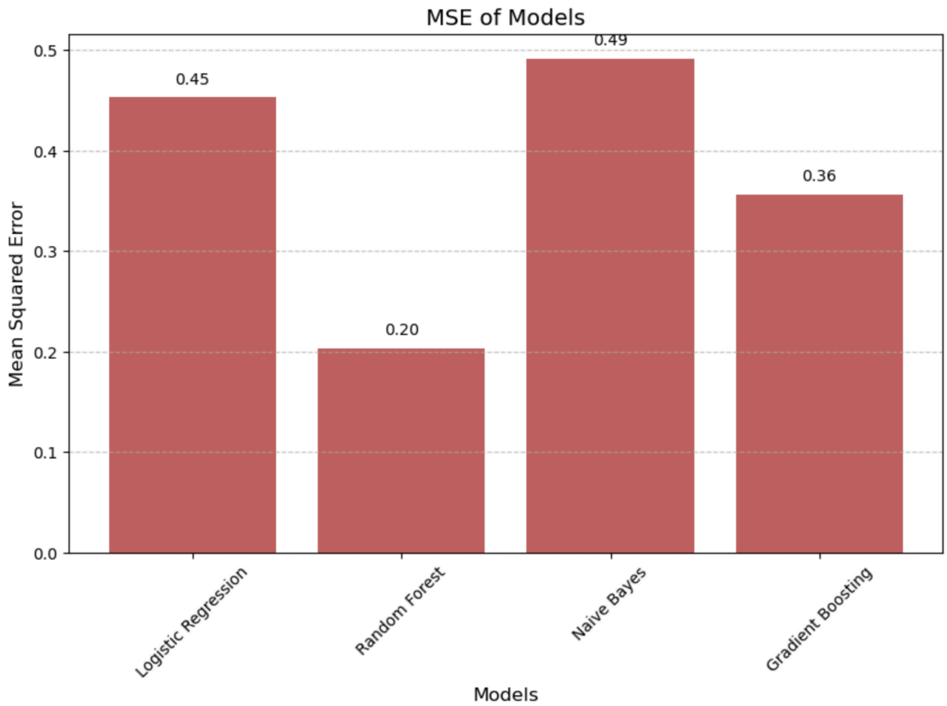
Models	Accuracy	ROC AUC	MSE	Confusion Matrix
Random Forest	0.797	0.911	0.203	[[230 68] [55 247]]
Logistic Regression	0.555	0.562	0.445	[[156 142] [138 164]]
Naive Bayes	0.5083	0.519	0.4917	[[258 33] [262 47]]
Gradient Boosting Classifier	0.645	0.70	0.355	[[192 106] [84 218]]

After finding out the best model for classifying we retrained the 'fundraising.csv' file using Random Forest Classifier. Firstly standardized the features using 'StandardScaler()' for both train and test datasets so that it can make predictions on which clients will be a 'Donor' or 'No Donor'.

Following are the graphs representing the accuracy of the models and the mean squared error showcasing the best performing model as Random Forest with least MSE.



Plot.3: Barplot of accuracies of models



Plot.4: Barplot of mean squared errors of models

CONCLUSION

In conclusion, the comprehensive analysis and evaluation of multiple machine learning models revealed that the Random Forest Classifier emerged as the most effective model for predicting donor behavior in direct-mail fundraising campaigns for the national veteran's organization. By leveraging its high accuracy, robustness, and efficient classification of both positive and negative instances, the organization can optimize its resource allocation and enhance the cost-effectiveness of its fundraising initiatives. Implementing the Random Forest model in practice holds the potential to significantly improve the organization's ability to target and engage potential donors, thereby maximizing fundraising outcomes and supporting its mission more effectively.

RECOMMENDATIONS

- ➢ Future research endeavors could explore alternative train and test split ratios, such as the commonly used 70/30 split. This approach can provide additional insights into the robustness and generalization capabilities of the models trained on varying proportions of the dataset.

- ➢ Researchers may consider employing a diverse range of modeling techniques beyond Random Forest, such as Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA), clustering algorithms, and other such methods. By evaluating multiple models, researchers can gain a deeper understanding of the underlying patterns in the data and identify which approach yields the most effective predictive performance for optimizing mailing campaign outcomes.

- ➢ Experimenting with different random seed values during model training can help assess the stability and reproducibility of results. By systematically changing the random seed, researchers can evaluate whether variations in model performance are influenced by random chance or inherent characteristics of the data.

- ➢ Researchers can explore novel approaches to feature engineering and variable selection by combining different sets of predictor variables. By incorporating a diverse range of features, including demographic, behavioral, and transactional data, researchers can uncover hidden patterns and relationships that may not be captured by individual variables alone. This comprehensive approach to feature engineering can enhance model performance and provide valuable insights for optimizing mailing campaign strategies.

APPENDIX

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
import seaborn as sns
import statsmodels.api as sm
from statsmodels.stats.anova import anova_lm
from ISLP import load_data
from ISLP.models import ModelSpec as MS, summarize
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from math import sqrt
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, roc_auc_score, confusion_matrix
from sklearn.utils.class_weight import compute_class_weight
from sklearn.utils import shuffle
from torch.utils.data import WeightedRandomSampler
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

fr = pd.read_csv("fundraising.csv")
fr
```

```
        zipconvert2     zipconvert3     zipconvert4     zipconvert5     homeowner       num_child
        income  female  wealth  home_value      ...     avg_fam_inc     pct_lt15k       num_prom
        lifetime_gifts  largest_gift    last_gift  months_since_donate  time_lag avg_gift  target
0       Yes     No      No      No      Yes     1       1       No      7       698     ...     463
        4       46      94.0    12.0    12.0    34      6       9.400000        Donor
1       No      No      No      Yes     No      2       5       Yes     8       828     ...     376
        13      32      30.0    10.0    5.0     29      7       4.285714        Donor
2       No      No      No      Yes     Yes     1       3       No      4       1471    ...     546
        4       94      177.0   10.0    8.0     30      3       7.080000        No Donor
3       No      Yes     No      No      Yes     1       4       No      8       547     ...     432
        7       20      23.0    11.0    11.0    30      6       7.666667        No Donor
4       No      Yes     No      No      Yes     1       4       Yes     8       482     ...     275
        28      38      73.0    10.0    10.0    31      3       7.300000        Donor
...     ...     ...     ...     ...     ...     ...     ...     ...     ...     ...     ...     ...
        ...     ...     ...     ...     ...     ...     ...     ...     ...
2995    No      No      Yes     No      Yes     1       5       Yes     4       882     ...     374
        21      46      111.0   22.0    9.0     36      5       9.250000        No Donor
2996    No      No      No      Yes     Yes     1       5       No      7       2085    ...     494
        9       55      93.0    15.0    15.0    29      3       9.300000        No Donor
2997    No      No      No      Yes     Yes     3       7       Yes     2       897     ...     334
        16      57      162.0   20.0    20.0    29      15      14.727273       No Donor
2998    No      No      No      Yes     No      1       2       Yes     5       429     ...     360
        15      54      80.0    10.0    10.0    32      1       8.000000        No Donor
2999    No      No      No      Yes     Yes     1       4       No      8       731     ...     341
        17      25      36.0    11.0    10.0    32      6       9.000000        Donor
```

```
# Count the number of donors and non-donors in the dataset
```

```
donor_counts = fr['target'].value_counts()

# Display the counts
print(donor_counts)
```

```
target
No Donor    1501
Donor       1499
Name: count, dtype: int64
```

```
frf = pd.read_csv("future_fundraising.csv")
frf
```

```
        zipconvert2     zipconvert3     zipconvert4     zipconvert5     homeowner       num_child
        income  female  wealth  home_value      med_fam_inc     avg_fam_inc     pct_lt15k
num_prom        lifetime_gifts  largest_gift    last_gift  months_since_donate  time_lag avg_gift
0       No      Yes     No      No      Yes     1       5       Yes     9       1399    637     703
        1       74      102.0   6.0     5.0     29      3       4.857143
1       Yes     No      No      No      Yes     1       1       No      7       1355    411     497
        9       77      249.0   15.0    7.0     35      3       9.576923
2       No      No      No      Yes     Yes     1       4       Yes     1       835     310     364
        22      70      126.0   6.0     6.0     34      8       4.344828
3       No      No      Yes     No      Yes     1       4       No      8       1019    389     473
        15      21      26.0    16.0    16.0    37      5       13.000000
4       No      Yes     No      No      Yes     1       2       Yes     7       992     524     563
        6       63      100.0   20.0    3.0     21      6       7.692308
...     ...     ...     ...     ...     ...     ...     ...     ...     ...     ...     ...     ...
        ...     ...     ...     ...     ...     ...     ...     ...
115     No      Yes     No      No      Yes     1       6       Yes     8       1126    609     657
        3       18      25.0    25.0    25.0    37      5       25.000000
116     No      Yes     No      No      Yes     1       4       No      2       604     259     295
        23      56      80.0    5.0     5.0     33      2       2.962963
117     No      No      No      Yes     No      1       2       No      8       412     240     299
        25      61      238.0   25.0    25.0    19      0       21.636364
118     No      No      Yes     No      Yes     1       5       No      8       1207    601     594
        0       46      105.0   15.0    10.0    29      7       10.500000
119     No      No      Yes     No      No      1       3       Yes     8       951     264     340
        28      70      306.0   25.0    20.0    18      2       15.300000
```

```
fr.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 21 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   zipconvert2          3000 non-null   object
 1   zipconvert3          3000 non-null   object
 2   zipconvert4          3000 non-null   object
 3   zipconvert5          3000 non-null   object
 4   homeowner            3000 non-null   object
 5   num_child            3000 non-null   int64
 6   income               3000 non-null   int64
 7   female               3000 non-null   object
 8   wealth               3000 non-null   int64
 9   home_value           3000 non-null   int64
 10  med_fam_inc          3000 non-null   int64
 11  avg_fam_inc          3000 non-null   int64
```

```
 12  pct_lt15k           3000 non-null   int64
 13  num_prom            3000 non-null   int64
 14  lifetime_gifts      3000 non-null   float64
 15  largest_gift        3000 non-null   float64
 16  last_gift           3000 non-null   float64
 17  months_since_donate  3000 non-null   int64
 18  time_lag            3000 non-null   int64
 19  avg_gift            3000 non-null   float64
 20  target              3000 non-null   object
dtypes: float64(4), int64(10), object(7)


frf.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 120 entries, 0 to 119
Data columns (total 20 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   zipconvert2          120 non-null    object
 1   zipconvert3          120 non-null    object
 2   zipconvert4          120 non-null    object
 3   zipconvert5          120 non-null    object
 4   homeowner            120 non-null    object
 5   num_child            120 non-null    int64
 6   income               120 non-null    int64
 7   female               120 non-null    object
 8   wealth               120 non-null    int64
 9   home_value           120 non-null    int64
 10  med_fam_inc          120 non-null    int64
 11  avg_fam_inc          120 non-null    int64
 12  pct_lt15k            120 non-null    int64
 13  num_prom             120 non-null    int64
 14  lifetime_gifts       120 non-null    float64
 15  largest_gift         120 non-null    float64
 16  last_gift            120 non-null    float64
 17  months_since_donate  120 non-null    int64
 18  time_lag             120 non-null    int64
 19  avg_gift             120 non-null    float64
dtypes: float64(4), int64(10), object(6)
memory usage: 18.9+ KB

fr['zipconvert2'] = fr['zipconvert2'].map({'Yes': 1, 'No': 0})
fr['zipconvert3'] = fr['zipconvert3'].map({'Yes': 1, 'No': 0})
fr['zipconvert4'] = fr['zipconvert4'].map({'Yes': 1, 'No': 0})
fr['zipconvert5'] = fr['zipconvert5'].map({'Yes': 1, 'No': 0})
fr['homeowner'] = fr['homeowner'].map({'Yes': 1, 'No': 0})
fr['female'] = fr['female'].map({'Yes': 1, 'No': 0})
fr['target'] = fr['target'].map({'Donor': 1, 'No Donor': 0})
#carseats = pd.get_dummies(carseats, prefix=['ShelveLoc'])
fr.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 21 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   zipconvert2          3000 non-null   int64
 1   zipconvert3          3000 non-null   int64
```

```
 2   zipconvert4         3000 non-null   int64
 3   zipconvert5         3000 non-null   int64
 4   homeowner           3000 non-null   int64
 5   num_child           3000 non-null   int64
 6   income              3000 non-null   int64
 7   female              3000 non-null   int64
 8   wealth              3000 non-null   int64
 9   home_value          3000 non-null   int64
 10  med_fam_inc         3000 non-null   int64
 11  avg_fam_inc         3000 non-null   int64
 12  pct_lt15k           3000 non-null   int64
 13  num_prom            3000 non-null   int64
 14  lifetime_gifts      3000 non-null   float64
 15  largest_gift        3000 non-null   float64
 16  last_gift           3000 non-null   float64
 17  months_since_donate  3000 non-null   int64
 18  time_lag            3000 non-null   int64
 19  avg_gift            3000 non-null   float64
 20  target              3000 non-null   int64
dtypes: float64(4), int64(17)
memory usage: 492.3 KB

frf['zipconvert2'] = frf['zipconvert2'].map({'Yes': 1, 'No': 0})
frf['zipconvert3'] = frf['zipconvert3'].map({'Yes': 1, 'No': 0})
frf['zipconvert4'] = frf['zipconvert4'].map({'Yes': 1, 'No': 0})
frf['zipconvert5'] = frf['zipconvert5'].map({'Yes': 1, 'No': 0})
frf['homeowner'] = frf['homeowner'].map({'Yes': 1, 'No': 0})
frf['female'] = frf['female'].map({'Yes': 1, 'No': 0})

frf.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 120 entries, 0 to 119
Data columns (total 20 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   zipconvert2         120 non-null    int64
 1   zipconvert3         120 non-null    int64
 2   zipconvert4         120 non-null    int64
 3   zipconvert5         120 non-null    int64
 4   homeowner           120 non-null    int64
 5   num_child           120 non-null    int64
 6   income              120 non-null    int64
 7   female              120 non-null    int64
 8   wealth              120 non-null    int64
 9   home_value          120 non-null    int64
 10  med_fam_inc         120 non-null    int64
 11  avg_fam_inc         120 non-null    int64
 12  pct_lt15k           120 non-null    int64
 13  num_prom            120 non-null    int64
 14  lifetime_gifts      120 non-null    float64
 15  largest_gift        120 non-null    float64
 16  last_gift           120 non-null    float64
 17  months_since_donate  120 non-null    int64
 18  time_lag            120 non-null    int64
 19  avg_gift            120 non-null    float64
dtypes: float64(4), int64(16)
memory usage: 18.9 KB
```

```
corr_matrix = fr.corr()

# create a heatmap to visualize the correlation matrix
plt.figure(figsize=(17,10))
sns.heatmap(corr_matrix, annot=True, cmap=plt.cm.Reds)
plt.show()

fr.drop(columns=['avg_fam_inc'], inplace=True)
frf.drop(columns=['avg_fam_inc'], inplace=True)
fr.drop(columns=['med_fam_inc'], inplace=True)
frf.drop(columns=['med_fam_inc'], inplace=True)
fr.drop(columns=['avg_gift'], inplace=True)
frf.drop(columns=['avg_gift'], inplace=True)

X = fr.drop(columns=["target"])
y = fr["target"]
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=1)


# Separate features and target
X = fr.drop(columns=["target"])
y = fr["target"]

# Calculate class weights
class_weights = compute_class_weight(class_weight='balanced', classes=np.unique(y), y=y)

# Convert class weights to a list
weights = [class_weights[int(cls == "Donor")] for cls in y]

# Create a WeightedRandomSampler
sampler = WeightedRandomSampler(weights, num_samples=len(y))

# Sample indices from the sampler
sample_indices = list(sampler)

# Use the sampled indices to create a balanced training set
X_sampled = X.iloc[sample_indices]
y_sampled = y.iloc[sample_indices]

# Split the sampled dataset into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_sampled, y_sampled, test_size=0.2, random_state=1)

from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, roc_auc_score, confusion_matrix

# Set random seed for reproducibility
random_state_seed = 12345

# Scale the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)

# Model 1: Logistic Regression with regularization and increased max_iter
logreg_model = LogisticRegression(class_weight='balanced',random_state=random_state_seed)
```

```
logreg_model.fit(X_train_scaled, y_train)

# Model 2: Random Forest Classifier
rf_model = RandomForestClassifier(random_state=random_state_seed)
rf_model.fit(X_train, y_train)

# Make predictions on the validation set
y_val_pred_logreg = logreg_model.predict(X_val_scaled)
y_val_pred_rf = rf_model.predict(X_val)

# Evaluate the models
accuracy_logreg = accuracy_score(y_val, y_val_pred_logreg)
accuracy_rf = accuracy_score(y_val, y_val_pred_rf)

roc_auc_logreg = roc_auc_score(y_val, logreg_model.predict_proba(X_val_scaled)[:, 1])
roc_auc_rf = roc_auc_score(y_val, rf_model.predict_proba(X_val)[:, 1])

conf_matrix_logreg = confusion_matrix(y_val, y_val_pred_logreg)
conf_matrix_rf = confusion_matrix(y_val, y_val_pred_rf)

mse_logreg = mean_squared_error(y_val, y_val_pred_logreg)
mse_rf = mean_squared_error(y_val, y_val_pred_rf)

# Print results
print("Logistic Regression - Accuracy:", accuracy_logreg)
print("Logistic Regression - ROC AUC:", roc_auc_logreg)
print(f"Logistic Regression -  Mean Squared Error (MSE):",mse_logreg)
print("Logistic Regression - Confusion Matrix:")
print(conf_matrix_logreg)

print("Random Forest - Accuracy:", accuracy_rf)
print("Random Forest - ROC AUC:", roc_auc_rf)
print(f"Random Forest -  Mean Squared Error (MSE):",mse_rf)
print("Random Forest - Confusion Matrix:")
print(conf_matrix_rf)
```

```
Logistic Regression - Accuracy: 0.555
Logistic Regression - ROC AUC: 0.5617277772217218
Logistic Regression -  Mean Squared Error (MSE): 0.445
Logistic Regression - Confusion Matrix:
[[169 122]
 [145 164]]
Random Forest - Accuracy: 0.7966666666666666
Random Forest - ROC AUC: 0.9114592021708426
Random Forest -  Mean Squared Error (MSE): 0.20333333333333334
Random Forest - Confusion Matrix:
[[237  54]
 [ 68 241]]
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# Initialize the StandardScaler
scaler = StandardScaler()
```

```
# Fit the scaler on the training data and transform both training and validation data
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)

# Initialize and train the Naive Bayes model
nb_model = GaussianNB()
nb_model.fit(X_train_scaled, y_train)

# Make predictions on the validation set
y_val_pred = nb_model.predict(X_val_scaled)

# Calculate the accuracy of the model on the validation set
accuracy = accuracy_score(y_val, y_val_pred)
roc_auc_nb = roc_auc_score(y_val, y_val_pred)
conf_matrix_nb = confusion_matrix(y_val, y_val_pred)
mse_nb = mean_squared_error(y_val, y_val_pred)

# Print the accuracy
print("Naive Bayes Model - Accuracy:", accuracy)
print("Naive Bayes Model - ROC AUC:", roc_auc_nb)
print(f"Naive Bayes Model -  Mean Squared Error (MSE):",mse_nb)
print("Naive Bayes Model - Confusion Matrix:")
print(conf_matrix_nb)

Naive Bayes Model - Accuracy: 0.5083333333333333
Naive Bayes Model - ROC AUC: 0.51935074900744
Naive Bayes Model -  Mean Squared Error (MSE): 0.49166666666666664
Naive Bayes Model - Confusion Matrix:
[[258  33]
 [262  47]]

from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, roc_auc_score, confusion_matrix

# Set random seed for reproducibility
random_state = 12345

# Model 3: Gradient Boosting Classifier
gb_model = GradientBoostingClassifier(random_state=random_state)
gb_model.fit(X_train_scaled, y_train)

# Make predictions on the validation set
y_val_pred_gb = gb_model.predict(X_val_scaled)

# Evaluate the model
accuracy_gb = accuracy_score(y_val, y_val_pred_gb)
roc_auc_gb = roc_auc_score(y_val, gb_model.predict_proba(X_val_scaled)[:, 1])
conf_matrix_gb = confusion_matrix(y_val, y_val_pred_gb)
mse = mean_squared_error(y_val, y_val_pred_gb)

# Print evaluation results
print("Gradient Boosting - Accuracy:", accuracy_gb)
print("Gradient Boosting - ROC AUC:", roc_auc_gb)
print("Gradient Boosting -  Mean Squared Error (MSE):", mse)
print("Gradient Boosting - Confusion Matrix:")
print(conf_matrix_gb)

Gradient Boosting - Accuracy: 0.645
```

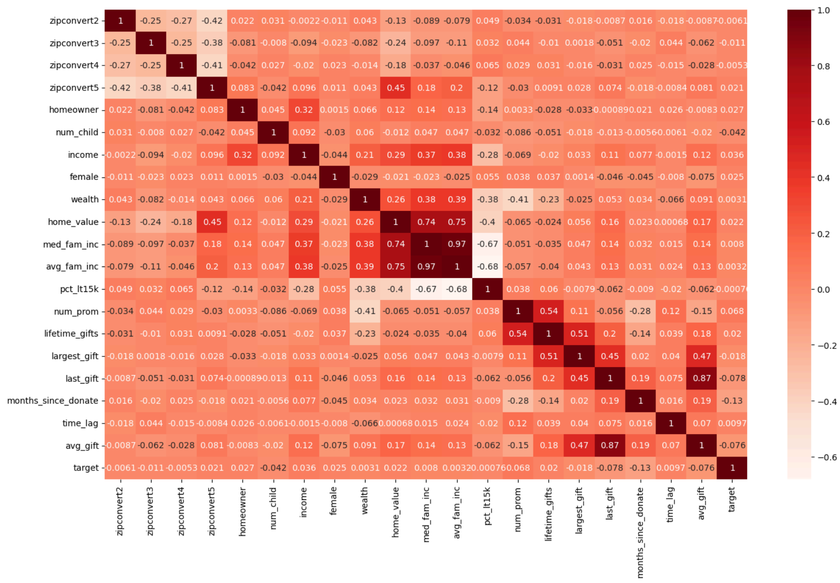
```
Gradient Boosting - ROC AUC: 0.7004859929492099
Gradient Boosting -  Mean Squared Error (MSE): 0.355
Gradient Boosting - Confusion Matrix:
[[191 100]
 [113 196]]
```

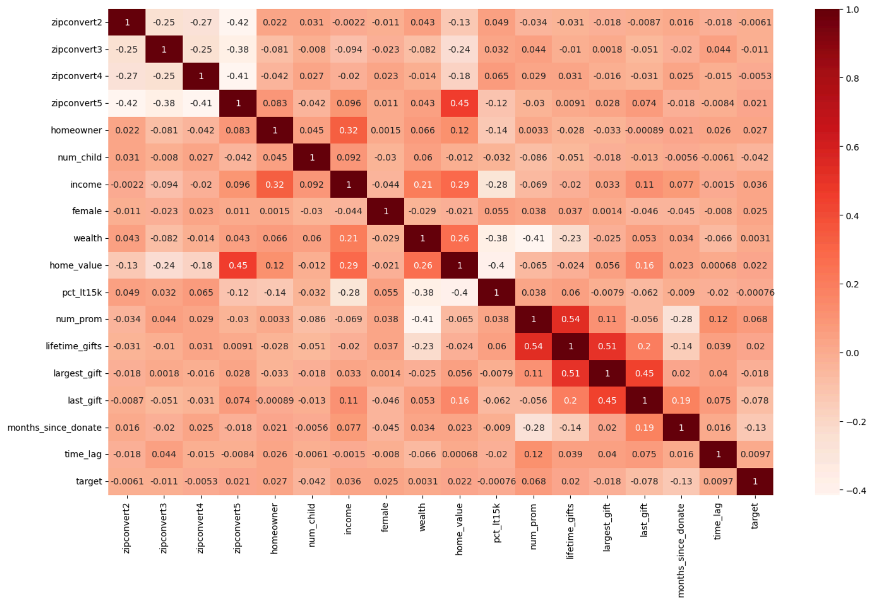
```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler

# Assuming `fr` (full training data) and `frf` (test data) are already loaded as pandas DataFrames

# Set the random state for reproducibility
random_state_seed = 12345

# Separate the features and target from the full training data
X_full_train = fr.drop(columns=['target'])
y_full_train = fr['target']

# Initialize the StandardScaler
scaler = StandardScaler()

# Fit the StandardScaler with the full training data (`X_full_train`) and transform the full training data
X_full_train_scaled = scaler.fit_transform(X_full_train)

# Transform the future fundraising test data (`frf`) using the fitted StandardScaler
X_test_scaled = scaler.transform(frf)

# Initialize the Random Forest Classifier
rf_model = RandomForestClassifier(random_state=random_state_seed)

# Train the Random Forest Classifier with the full training data
rf_model.fit(X_full_train, y_full_train)

# Use the trained Random Forest Classifier to make predictions on the scaled test data
y_test_pred = rf_model.predict(X_test_scaled)

# Create a DataFrame for the submission file
submission_df = pd.DataFrame({'Prediction': y_test_pred})

# Save the predictions to a CSV file for submission
submission_df.to_csv('C:/Users/gagss/OneDrive/Documents/spring 2024/Predictive
Modelling/Project/submission23.csv', index=False)

print("Predictions saved to submission23.csv")
```

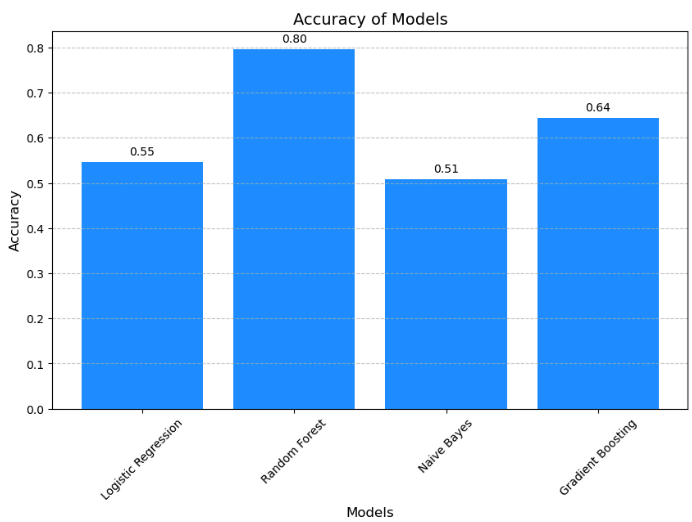
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, roc_auc_score, confusion_matrix, mean_squared_error
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Define the list of models
models = [
```

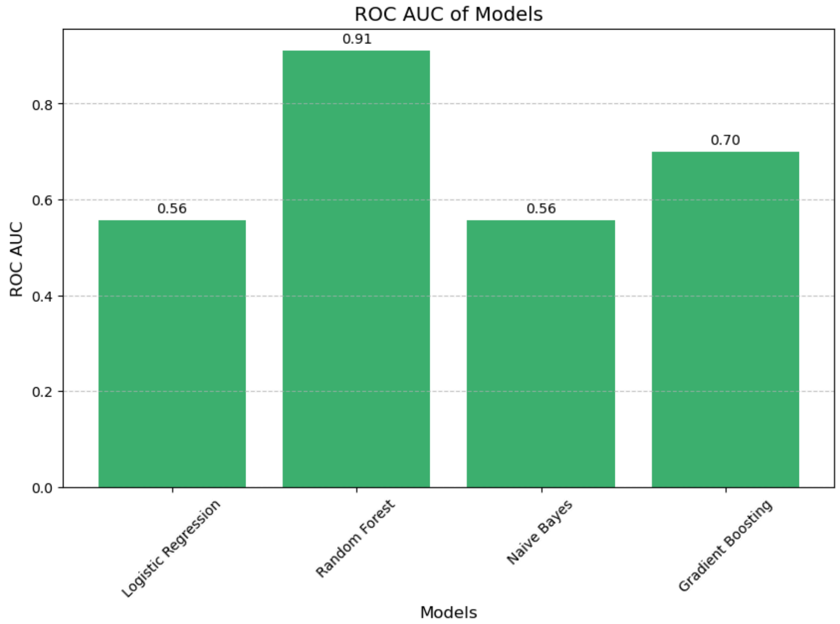
```
    {'name': 'Logistic Regression', 'model': LogisticRegression(class_weight='balanced',
random_state=random_state_seed)},
    {'name': 'Random Forest', 'model': RandomForestClassifier(random_state=random_state_seed)},
    {'name': 'Naive Bayes', 'model': GaussianNB()},
    {'name': 'Gradient Boosting', 'model': GradientBoostingClassifier(random_state=random_state_seed)}
]

# Scale the training and validation data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)

# Initialize lists to store the performance metrics
names = []
accuracies = []
roc_aucs = []
mses = []

# Loop through each model
for model_info in models:
    name = model_info['name']
    model = model_info['model']

    # Fit the model
    if name == 'Naive Bayes':
        # Use scaled data for Naive Bayes
        model.fit(X_train_scaled, y_train)
        y_val_pred = model.predict(X_val_scaled)
        y_val_pred_proba = model.predict_proba(X_val_scaled)[:, 1]
    else:
        # Use original data for other models
        model.fit(X_train, y_train)
        y_val_pred = model.predict(X_val)
        y_val_pred_proba = model.predict_proba(X_val)[:, 1]

    # Calculate metrics
    accuracy = accuracy_score(y_val, y_val_pred)
    roc_auc = roc_auc_score(y_val, y_val_pred_proba)
    mse = mean_squared_error(y_val, y_val_pred)
    conf_matrix = confusion_matrix(y_val, y_val_pred)

    # Print the metrics
    #print(f"{name} - Accuracy: {accuracy}")
    #print(f"{name} - ROC AUC: {roc_auc}")
    #print(f"{name} - Mean Squared Error (MSE): {mse}")
    #print(f"{name} - Confusion Matrix:")
    #print(conf_matrix)
    #print("\n")

    # Store the metrics for plotting
    names.append(name)
    accuracies.append(accuracy)
    roc_aucs.append(roc_auc)
    mses.append(mse)

# Import necessary libraries
import matplotlib.pyplot as plt
```

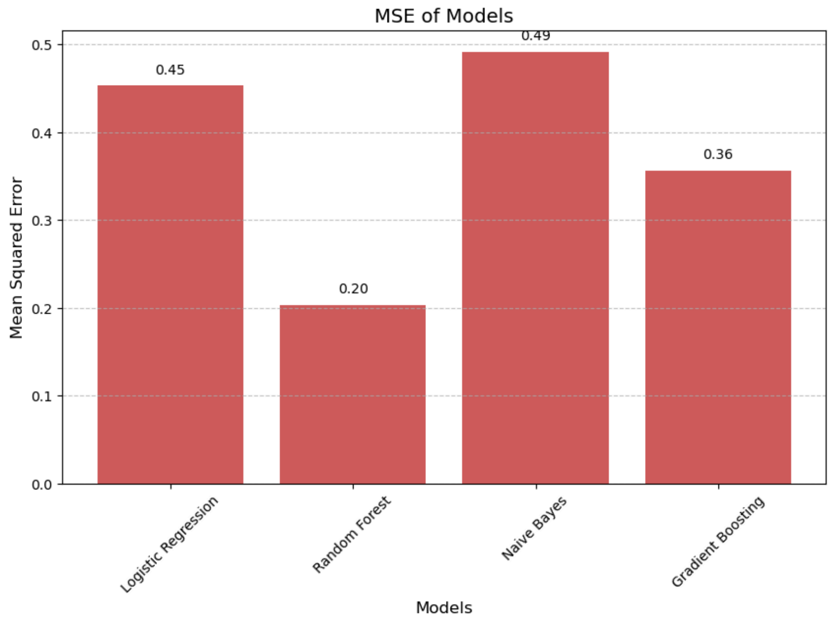
```
# Define function to plot accuracy of the models
def plot_accuracy(names, accuracies):
    plt.figure(figsize=(10, 6))
    # Plot bar chart
    plt.bar(names, accuracies, color='dodgerblue')
    plt.title('Accuracy of Models', fontsize=14)
    plt.ylabel('Accuracy', fontsize=12)
    plt.xlabel('Models', fontsize=12)
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.xticks(rotation=45, fontsize=10)
    plt.yticks(fontsize=10)
    # Annotate bars with their values
    for i, value in enumerate(accuracies):
        plt.text(i, value + 0.01, f'{value:.2f}', ha='center', va='bottom', fontsize=10)
    plt.show()

# Define function to plot ROC AUC of the models
def plot_roc_auc(names, roc_aucs):
    plt.figure(figsize=(10, 6))
    # Plot bar chart
    plt.bar(names, roc_aucs, color='mediumseagreen')
    plt.title('ROC AUC of Models', fontsize=14)
    plt.ylabel('ROC AUC', fontsize=12)
    plt.xlabel('Models', fontsize=12)
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.xticks(rotation=45, fontsize=10)
    plt.yticks(fontsize=10)
    # Annotate bars with their values
    for i, value in enumerate(roc_aucs):
        plt.text(i, value + 0.01, f'{value:.2f}', ha='center', va='bottom', fontsize=10)
    plt.show()

# Define function to plot mean squared error (MSE) of the models
def plot_mse(names, mses):
    plt.figure(figsize=(10, 6))
    # Plot bar chart
    plt.bar(names, mses, color='indianred')
    plt.title('MSE of Models', fontsize=14)
    plt.ylabel('Mean Squared Error', fontsize=12)
    plt.xlabel('Models', fontsize=12)
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.xticks(rotation=45, fontsize=10)
    plt.yticks(fontsize=10)
    # Annotate bars with their values
    for i, value in enumerate(mses):
        plt.text(i, value + 0.01, f'{value:.2f}', ha='center', va='bottom', fontsize=10)
    plt.show()

# Plot accuracy of models
plot_accuracy(names, accuracies)

# Plot ROC AUC of models
plot_roc_auc(names, roc_aucs)

# Plot mean squared error of models
plot_mse(names, mses)
```


zipconvert2
zipconvert3
zipconvert4
zipconvert5
homeowner
num_child
income
female
wealth
home_value
med_fam_inc
avg_fam_inc
pct_lt15k
num_prom
lifetime_gifts
largest_gift
last_gift
months_since_donate
time_lag
avg_gift
target
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6


zipconvert2
zipconvert3
zipconvert4
zipconvert5
homeowner
num_child
income
female
wealth
home_value
pct_lt15k
num_prom
lifetime_gifts
largest_gift
last_gift
months_since_donate
time_lag
target
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4


Accuracy of Models
Accuracy
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.55
0.80
0.51
0.64
Logistic Regression
Random Forest
Naive Bayes
Gradient Boosting
Models


ROC AUC of Models
0.91
0.8
0.70
0.6
0.56
0.56
ROC AUC
0.4
0.2
0.0
Logistic Regression
Random Forest
Naive Bayes
Gradient Boosting
Models


MSE of Models
0.5
0.49
0.45
0.4
0.36
Mean Squared Error
0.3
0.2
0.20
0.1
0.0
Logistic Regression
Random Forest
Naive Bayes
Gradient Boosting
Models