```
!pip install xgboost
pip install scikit-learn

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

pd.set_option('display.max_columns', None)

all = pd.read_csv("ALL.csv")
```

For this analysis, we only want to look at customer churn for 2023 Quarter 1

```
one_two = all[all['quarter'].isin([1, 2])]

# here we are getting all of the unique customer IDS from quarter 1
and 2 to
# compare who churned vs whos still active

quarter_1_df = all[all['quarter'] == 1]

# Get all unique ids from the filtered DataFrame
unique_1=
quarter_1_df[['extid']].drop_duplicates().reset_index(drop=True)

quarter_2_df = all[all['quarter'] == 2]

# Get all unique ids from the filtered DataFrame
unique_2=
quarter_2_df[['extid']].drop_duplicates().reset_index(drop=True)

quarter_1_df = pd.merge(quarter_1_df, unique_1, on=['extid'], how=
'left')

unique_1['churn'] = unique_1['extid'].apply(lambda x: 0 if x in
unique_2['extid'].values else 1)

quarter_1_df.to_csv("quarter1_2.csv", index=False)

one_two = pd.read_csv("quarter1_2.csv")

# checking to see how many "0" we have which indicate no activity not
missing activity
zero_counts = (one_two == 0).sum()
percentage_zero_counts = zero_counts / 55829 * 100
percentage_zero_counts

# dropping extid and quarter for correlation matrix
quarter_1_df.drop(columns=['extid','quarter'], inplace=True)
```

```
# install packages
# Models
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB

# XGBoost
from xgboost import XGBClassifier

# Evaluation Metrics
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,
classification_report, roc_curve, auc, accuracy_score
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score

# checking for nan
nan_rows_all = one_two[one_two.isna().any(axis=1)]
nan_rows_all

# Calculate the correlation matrix
corr = one_two.corr()

# Create a heatmap
plt.figure(figsize=(13,8))
sns.heatmap(corr, annot=True, fmt=".2f", cmap='viridis',
linewidths=.5)
plt.title('Heatmap of Attribute Correlation')
plt.show()

# print out any high correlations
high_corr_pairs = []
for i in range(len(corr.columns)):
    for j in range(i+1, len(corr.columns)):
        if corr.iloc[i, j] > 0.8:
            high_corr_pairs.append((corr.columns[i], corr.columns[j],
corr.iloc[i, j]))

print("Pairs of variables with correlation greater than 0.8:")
for pair in high_corr_pairs:
    print(f"{pair[0]} and {pair[1]} with correlation {pair[2]:.2f}")

# check distribution of our y variable, 0 and 1 counts
churn_count = one_two['churn'].value_counts()
churn_count

# here we have an imbalance, not churned (0) - 39529 while churned (1)
- 16300
```

```
# now we need to balance our dataset in terms of our y variable, here
we want a 50/50 split
# so we're going to match the number of not churn (0) with churn (1)
# churn has 16300 so we're going to randomly sample 16300 rows of not
churned

selected_rows_churn_0 = one_two[one_two['churn'] == 0].sample(n=16300,
random_state=42)

# Filter rows where churn is 1
rows_churn_1 = one_two[one_two['churn'] == 1]

# Concatenate selected_rows_churn_0 with rows_churn_1
new_quarter = pd.concat([selected_rows_churn_0, rows_churn_1],
ignore_index=True)

churn_count = new_quarter['churn'].value_counts()
churn_count

# now we are going to prpare our datasets for training/testing

X = new_quarter.drop('churn', axis=1)
y = new_quarter['churn']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Checking Accuarcy for some Baby models

classifiers = [

['DecisionTree :', DecisionTreeClassifier(max_depth=5)],
['Naive Bayes :', GaussianNB()],
['KNeighbours :', KNeighborsClassifier()]]

# creating a classifier using each of the algorithms and prediciting
their accuracies
print('Accuracies:')
for name, classifier in classifiers:
    classifier.fit(X_train, y_train)
    predictions = classifier.predict(X_test)
    print(name, accuracy_score(y_test, predictions))

# HERE WE ARE GOING TO RUN AN ALL-IN LOGISTIC REGRESSION

# Initialize a Logistic Regression classifier
logreg_classifier = LogisticRegression()

# Perform cross-validation
cv_scores = cross_val_score(logreg_classifier, X_train, y_train,
cv=10)

# Train the model on the training set
logreg_classifier.fit(X_train, y_train)

# Predict on the test set
y_pred_lr = logreg_classifier.predict(X_test)

# Calculate performance metrics
accuracy = accuracy_score(y_test, y_pred_lr)
classification_report_result = classification_report(y_test,
y_pred_lr)
confusion_matrix_result = confusion_matrix(y_test, y_pred_lr)

print("Logistic Regression Cross-Validation Scores:", cv_scores)
print("Logistic Regression Cross-Validation Scores Average:",
np.mean(cv_scores))
print("Logistic Regression Accuracy:", accuracy)
print("\nLogistic Regression Classification Report:\n",
classification_report_result)
print("\nLogistic Regression Confusion Matrix:\n",
confusion_matrix_result)

# THIS ALL IN LOGISITIC REGRESSION HAS AN ACCURACY OF 73%

# to check for variance
from statsmodels.stats.outliers_influence import
variance_inflation_factor

# Assuming X_train is a DataFrame excluding the target variable
X = sm.add_constant(X_train)  # Adding a constant for intercept
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in
range(len(X.columns))]

print(vif_data)

# to display variables and their significance
logit_model = sm.Logit(y_train, X_train)

# Fit the model
result = logit_model.fit()

# Print out the summary of the model
print(result.summary())

# because this is a logisitic regression, we are now calculating the
# probabilities so we can better interpret the results

# Extract coefficients
```

```
coefficients = result.params
odds_ratios = np.exp(coefficients)

# Create a DataFrame for easy interpretation
interpretation_df = pd.DataFrame({
    'Feature': coefficients.index,
    'Coefficient (Log Odds)': coefficients.values,
    'Odds Ratio': odds_ratios
})

print(interpretation_df)
```

Now we are going to run random forest to help with feature selection since our main dataset has 52 variables

Additionally we want to see if we can improve our accuracy

```
# here we are tuning our hyperparameters by searching for the bext n

# Define a range of n_estimators to test
n_estimators_range = [50, 100, 150, 200, 250, 300, 350, 400, 450, 500]

# Initialize an empty list to store the mean cross-validation scores
cv_scores = []

# Perform cross-validation for each value of n_estimators
for n in n_estimators_range:
    rf_model = RandomForestClassifier(n_estimators=n, random_state=42)
    scores = cross_val_score(rf_model, X_train, y_train, cv=5,
scoring='accuracy')
    cv_scores.append(np.mean(scores))

# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(n_estimators_range, cv_scores, marker='o')
plt.xlabel('Number of Estimators (n_estimators)')
plt.ylabel('Cross-Validation Accuracy')
plt.title('Cross-Validation Accuracy vs. Number of Estimators')
plt.grid(True)
plt.show()

# Find the n_estimators with the best cross-validation score
best_n_estimators = n_estimators_range[np.argmax(cv_scores)]
print(f"Best Number of Estimators: {best_n_estimators}")

# Train the final model with the best number of estimators
rf_model = RandomForestClassifier(n_estimators=best_n_estimators,
random_state=42)
rf_model.fit(X_train, y_train)
```

```
# Evaluate the model on the test set
y_pred = rf_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Test Set Accuracy: {accuracy}")

# the best n was 350 so plug into code below

from sklearn.ensemble import RandomForestClassifier

rf_model= RandomForestClassifier(n_estimators = 350)
rf_model.fit(X_train, y_train)

from sklearn.metrics import accuracy_score

y_pred = rf_model.predict(X_test)
print("Accuracy Score: {}".format(accuracy_score(y_test, y_pred)))

feature_df = new_quarter.drop(['churn'], axis = 1)
# storing the feature names in a list
feature_names_from_dataset =
new_quarter.columns.drop('churn').tolist()
feature_names_from_dataset

# To display feature importances
feature_importances = rf_model.feature_importances_
feature_importance_df = pd.DataFrame({'Feature': X.columns,
'Importance': feature_importances})
print(feature_importance_df.sort_values(by='Importance',
ascending=False))

importances = rf_model.feature_importances_
std = np.std([tree.feature_importances_ for tree in
rf_model.estimators_], axis=0)

forest_importances = pd.Series(importances, index =
feature_names_from_dataset)
# print(forest_importances)

fig, ax = plt.subplots()
forest_importances.plot.bar(yerr=std, ax=ax)
ax.set_title("Feature importances using MDI")
ax.set_ylabel("Mean decrease in impurity")
fig.tight_layout()
```

now we're going to take the top 10 most important features in random forest and test for their significance in a logisitic regression

```
# Sort features by importance
sorted_features = feature_importance_df.sort_values(by='Importance',
ascending=False)
```

```
# Select the top 10 features
top_10_features = sorted_features['Feature'].head(10).tolist()

# Add the 'churn' column to the list of features
top_10_features.append('churn')

# Make a copy of these columns from the original dataset one_two
one_two_copy = one_two[top_10_features].copy()

# Verify the results
print(one_two_copy.head())

# so resplit the data

selected_rows_churn_0 = one_two_copy[one_two_copy['churn'] ==
0].sample(n=16300, random_state=42)

# Filter rows where churn is 1
rows_churn_1 = one_two_copy[one_two_copy['churn'] == 1]

# Concatenate selected_rows_churn_0 with rows_churn_1
new_quarter = pd.concat([selected_rows_churn_0, rows_churn_1],
ignore_index=True)

churn_count = new_quarter['churn'].value_counts()
churn_count

X = new_quarter.drop('churn', axis=1)
y = new_quarter['churn']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Initialize a Logistic Regression classifier
logreg_classifier = LogisticRegression()

# Perform cross-validation
cv_scores = cross_val_score(logreg_classifier, X_train, y_train,
cv=10)

# Train the model on the training set
logreg_classifier.fit(X_train, y_train)

# Predict on the test set
y_pred_lr = logreg_classifier.predict(X_test)

# Calculate performance metrics
accuracy = accuracy_score(y_test, y_pred_lr)
classification_report_result = classification_report(y_test,
y_pred_lr)
confusion_matrix_result = confusion_matrix(y_test, y_pred_lr)

print("Logistic Regression Cross-Validation Scores:", cv_scores)
print("Logistic Regression Cross-Validation Scores Average:",
np.mean(cv_scores))
print("Logistic Regression Accuracy:", accuracy)
print("\nLogistic Regression Classification Report:\n",
classification_report_result)
print("\nLogistic Regression Confusion Matrix:\n",
confusion_matrix_result)

# accuracy of 73%

import statsmodels.api as sm
# Assuming X_train and y_train are pandas DataFrames
X_train = sm.add_constant(X_train)  # Adds an intercept term to the
model
logit_model = sm.Logit(y_train, X_train)

# Fit the model
result = logit_model.fit()

# Print out the summary of the model
print(result.summary())
```

from this second logistic regression, we see that 8/10 variables are statistically significant

so for our third analysis, we are going to combine some significant variables from our all-in regression, some important features from our random forest, and a few other variables that are important to look at from a business standpoint

```
# here we are creating a new dataset wil all variables that we think
are important

# Drop the 'mobile' column from one_two_copy
one_two_copy.drop(columns=['mobile'], inplace=True)

# Select the columns to be added from one_two
columns_to_add = ['top_ref_campaign', 'ptr', 'd2d', 'promo',
'other_ref_campaign']

# Ensure these columns exist in the one_two DataFrame
for column in columns_to_add:
    if column not in one_two.columns:
        raise ValueError(f"Column '{column}' does not exist in the
one_two DataFrame")

# Add these columns to one_two_copy
one_two_copy = one_two_copy.join(one_two[columns_to_add])
```

```
# Verify the results
print(one_two_copy.head())

# respliting the data

selected_rows_churn_0 = one_two_copy[one_two_copy['churn'] ==
0].sample(n=16300, random_state=42)

# Filter rows where churn is 1
rows_churn_1 = one_two_copy[one_two_copy['churn'] == 1]

# Concatenate selected_rows_churn_0 with rows_churn_1
new_quarter = pd.concat([selected_rows_churn_0, rows_churn_1],
ignore_index=True)

churn_count = new_quarter['churn'].value_counts()
churn_count

X = new_quarter.drop('churn', axis=1)
y = new_quarter['churn']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Initialize a Logistic Regression classifier
logreg_classifier = LogisticRegression()

# Perform cross-validation
cv_scores = cross_val_score(logreg_classifier, X_train, y_train,
cv=10)

# Train the model on the training set
logreg_classifier.fit(X_train, y_train)

# Predict on the test set
y_pred_lr = logreg_classifier.predict(X_test)

# Calculate performance metrics
accuracy = accuracy_score(y_test, y_pred_lr)
classification_report_result = classification_report(y_test,
y_pred_lr)
confusion_matrix_result = confusion_matrix(y_test, y_pred_lr)

print("Logistic Regression Cross-Validation Scores:", cv_scores)
print("Logistic Regression Cross-Validation Scores Average:",
np.mean(cv_scores))
print("Logistic Regression Accuracy:", accuracy)
print("\nLogistic Regression Classification Report:\n",
classification_report_result)
```

```
print("\nLogistic Regression Confusion Matrix:\n",
confusion_matrix_result)


# accuracy of 73%

import statsmodels.api as sm
# Assuming X_train and y_train are pandas DataFrames
X_train = sm.add_constant(X_train)  # Adds an intercept term to the
model
logit_model = sm.Logit(y_train, X_train)

# Fit the model
result = logit_model.fit()

# Print out the summary of the model
print(result.summary())

# because this is a logisitic regression, we are now calculating the
# probabilities so we can better interpret the results

# Extract coefficients
coefficients = result.params
odds_ratios = np.exp(coefficients)

# Create a DataFrame for easy interpretation
interpretation_df = pd.DataFrame({
    'Feature': coefficients.index,
    'Coefficient (Log Odds)': coefficients.values,
    'Odds Ratio': odds_ratios
})

print(interpretation_df)
```

Our final model achieves an accuracy of 73%, comparable to our initial two models. It stands out for its simplicity and ease of interpretation, making it more accessible for businesses. Additionally, the model incorporates relevant variables that can offer valuable insights into areas where the company can improve.