ML_MP2

Clarice Tee

2025-02-13

Table of contents

1 Part 2

1.Beyond the factors listed in Table 1, I think that whether a firm commits tax evasion or not will depend on many other factors, some difficult to quantify, others hard to uncover. The listed variables will already provide us with a valuable information that are reasons to beliee that tax evasion has occured. These include, the company's internal ethos, accounting transparency mechanisms, the types of accounts/assets the company owns (e.g. offshore/shell companies, crypto assets), types of "business expenses", bribery by the company/corruption by some government officials.

2. In this case, KNN is better for several reasons. Firstly, it is a non-parametric method, thus does not need/assume a functional form for the relationship between X and Y variables. So, even without explicity interaction terms, KNN is flexible enough to identify and incorporate the interactions. Secondly, this means that KNN is more adaptable to complex patterns (non-linear). Finally, KNN uses data to predict using data on observatins of the nearest neighbors which are similar. So KNN can effectively capture the interction between predictors if it is important to the true relationship because it is non-parametric, unlike LPM. (see: 112-115 and PhoenixAI)

2 Part 3

```
import pandas as pd
import numpy as np
import statsmodels.formula.api as smf
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import pyplot
import os
import statsmodels.formula.api as smf
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LinearRegression as lm
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, r2_score, mean_squared_error
from sklearn.metrics import accuracy_score, precision_score, confusion_matrix,
 ↪  roc_curve, roc_auc_score, RocCurveDisplay
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV, KFold, train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import precision_score
from sklearn.metrics import accuracy_score, precision_score, confusion_matrix,
 ↪  roc_curve, roc_auc_score, RocCurveDisplay
```

```
# Load the dataset
directory = r"C:\Users\clari\OneDrive\Documents\Machine Learning\mp2"
audit_path = os.path.join(directory, "Data-Audit.csv")
audit_df = pd.read_csv(audit_path)
print(audit_df.dtypes)
print(audit_df.shape)
audit_df.head()
```

```
Sector_score     float64
PARA_A           float64
Risk_A           float64
PARA_B           float64
Risk_B           float64
Money_Value      float64
Risk_D           float64
Score            float64
Inherent_Risk    float64
Audit_Risk       float64
Risk               int64
dtype: object
(776, 11)
```

	Sector_score	PARA_A	Risk_A	PARA_B	Risk_B	Money_Value	Risk_D	Score	Inherer
0	3.89	4.18	2.508	2.50	0.500	3.38	0.676	2.4	8.574
1	3.89	0.00	0.000	4.83	0.966	0.94	0.188	2.0	2.554
2	3.89	0.51	0.102	0.23	0.046	0.00	0.000	2.0	1.548
3	3.89	0.00	0.000	10.80	6.480	11.75	7.050	4.4	17.530
4	3.89	0.00	0.000	0.08	0.016	0.00	0.000	2.0	1.416

```
na_count = pd.DataFrame(np.sum(audit_df.isna(), axis = 0), columns = ["Count
 ↪  NAs"])
print(na_count)
audit_df = audit_df.dropna()
```

```
                Count NAs
Sector_score            0
PARA_A                  0
Risk_A                  0
PARA_B                  0
Risk_B                  0
Money_Value             1
Risk_D                  0
Score                   0
Inherent_Risk           0
Audit_Risk              0
Risk                    0
```

```
# Sort our target and features into different dataframes
X = audit_df.drop(['Risk'], axis = 1)
target = audit_df.loc[:,'Risk']
display(X.head())
display(target.head())

target.value_counts()

# Split training/validation set
X_train, X_valid, target_train, target_valid = train_test_split(
X, target, train_size=0.50, random_state=13
)
```

	Sector_score	PARA_A	Risk_A	PARA_B	Risk_B	Money_Value	Risk_D	Score	Inherer
0	3.89	4.18	2.508	2.50	0.500	3.38	0.676	2.4	8.574
1	3.89	0.00	0.000	4.83	0.966	0.94	0.188	2.0	2.554
2	3.89	0.51	0.102	0.23	0.046	0.00	0.000	2.0	1.548

	Sector_score	PARA_A	Risk_A	PARA_B	Risk_B	Money_Value	Risk_D	Score	Inherer
3	3.89	0.00	0.000	10.80	6.480	11.75	7.050	4.4	17.530
4	3.89	0.00	0.000	0.08	0.016	0.00	0.000	2.0	1.416

```
0    1
1    0
2    0
3    1
4    0
Name: Risk, dtype: int64
```

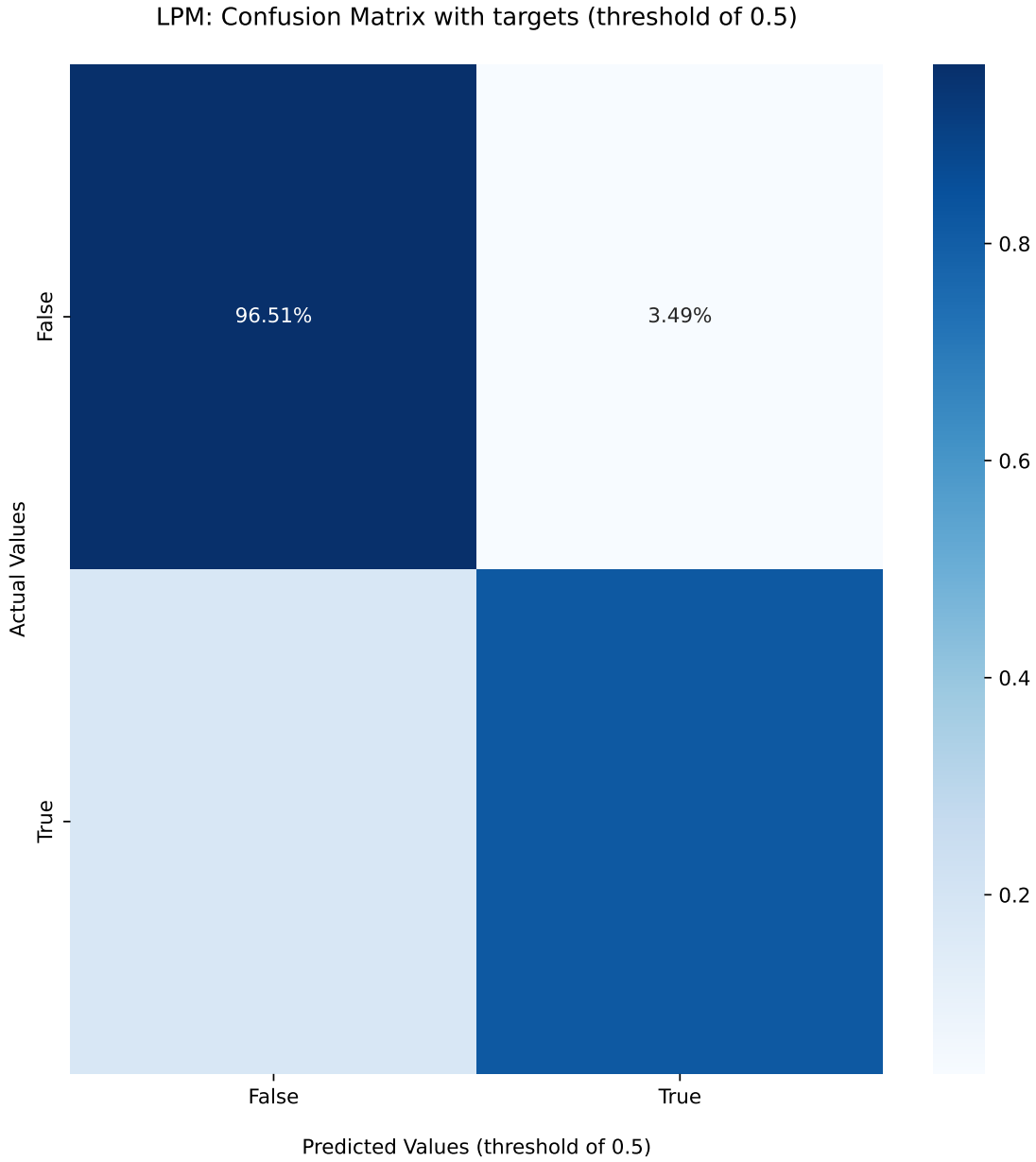
LPM

```
# Fit the model
lpm=lm().fit(X_train, target_train)
# Apply the model to the validation set
lpm_pred = lpm.predict(X_valid)
```

3.a. For firms with a predicted probability of tax evasion greater than 0:5, construct the confusion matrix.

```
lpm_pred_5 = np.where(lpm_pred > 0.5, 1, 0)

# Confusion matrix
lmp_cm_5 = confusion_matrix(target_valid, lpm_pred_5, normalize="true")
# Display the visualization of the confusion matrix
fig, ax = plt.subplots(figsize=(9, 9))
ax = sns.heatmap(
lmp_cm_5, annot=True, fmt=".2%", cmap="Blues"
)
ax.set_title("LPM: Confusion Matrix with targets (threshold of 0.5)\n")
ax.set_xlabel("\nPredicted Values (threshold of 0.5)")
ax.set_ylabel("Actual Values")
ax.xaxis.set_ticklabels(["False", "True"])
ax.yaxis.set_ticklabels(["False", "True"])
plt.show()
```

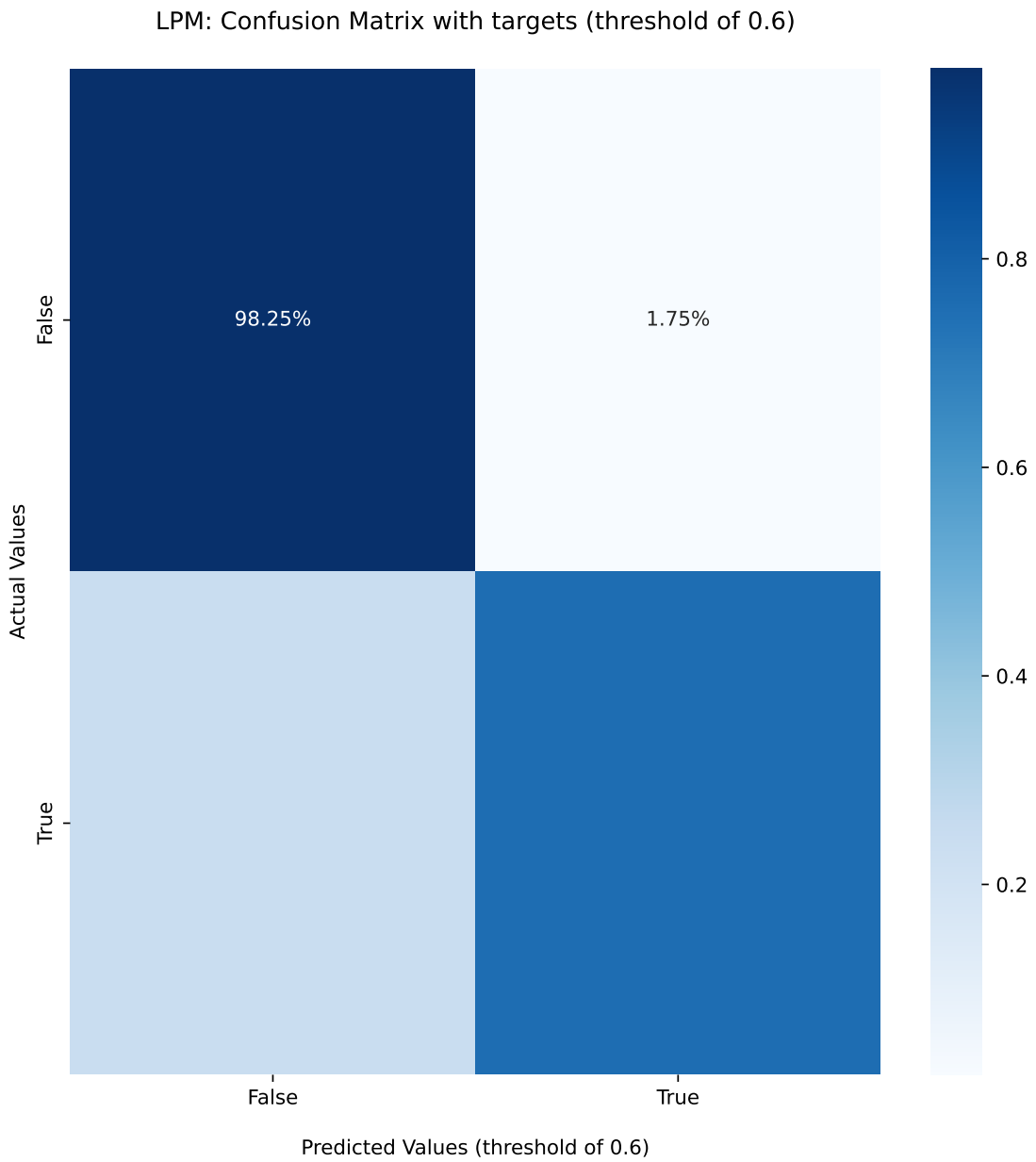


3.b.

```
lpm_pred_6 = np.where(lpm_pred > 0.6, 1, 0)

# Confusion matrix
lmp_cm_6 = confusion_matrix(target_valid, lpm_pred_6, normalize="true")
# Display the visualization of the confusion matrix
fig, ax = plt.subplots(figsize=(9, 9))
ax = sns.heatmap(
```

```
lmp_cm_6, annot=True, fmt=".2%", cmap="Blues"
)
ax.set_title("LPM: Confusion Matrix with targets (threshold of 0.6)\n")
ax.set_xlabel("\nPredicted Values (threshold of 0.6)")
ax.set_ylabel("Actual Values")
ax.xaxis.set_ticklabels(["False", "True"])
ax.yaxis.set_ticklabels(["False", "True"])
plt.show()
```



3.c. Error rate

```
error_lpm_5 = 1 - accuracy_score(target_valid, lpm_pred_5)
error_lpm_6 = 1 - accuracy_score(target_valid, lpm_pred_6)

print(f"The error rate at the 0.5 threshold is approximately:
 ↪  {round(error_lpm_5, 4)*100}%")
print(f"The error rate at the 0.6 threshold is approximately{round(error_lpm_6,
 ↪  4)*100}%.")
```

```
The error rate at the 0.5 threshold is approximately: 9.54%
The error rate at the 0.6 threshold is approximately11.08%.
```

The 0.5 threshold results in a more accurate prediction.

3.d. Precision = True Positives / (True Positives + False Positives) or Out of all the positive predictions made, how many are actually positive? (proportion of positive identifications that were actually correct)

```
# For threshold = 0.5
precision_5 = precision_score(target_valid, lpm_pred_5, zero_division=0)
print(f"Positive predictions that were actually correct at the 0.5 threshold:
 ↪  {precision_5:.2%}")

# For threshold = 0.6
precision_6 = precision_score(target_valid, lpm_pred_6, zero_division=0)
print(f"Positive predictions that were actually correct at the 0.6 threshold:
 ↪  {precision_6:.2%}")
```

```
Positive predictions that were actually correct at the 0.5 threshold:
94.20%
Positive predictions that were actually correct at the 0.6 threshold:
96.77%
```

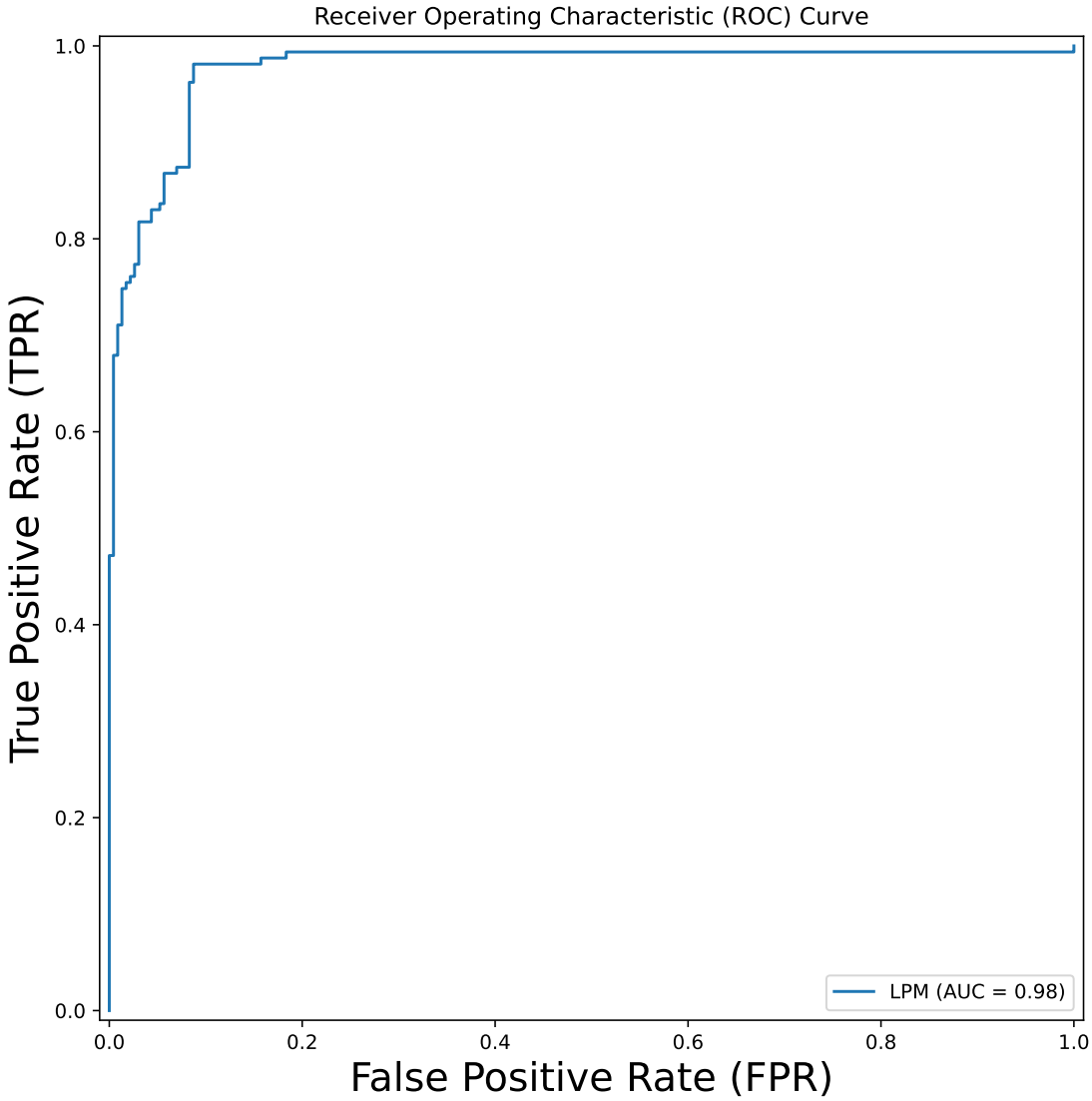
3.e.

```
# ROC and AUC
fpr_lpm, tpr_lpm, thresholds_lpm = roc_curve(target_valid, lpm_pred)
auc_lpm = roc_auc_score(target_valid, lpm_pred)

roc_curve_lpm = RocCurveDisplay(
fpr=fpr_lpm, tpr=tpr_lpm,
roc_auc=auc_lpm,
estimator_name="LPM"
)

fig, ax = plt.subplots(figsize=(9, 9))
```

```
roc_curve_lpm.plot(ax=ax)
ax.set_ylabel("True Positive Rate (TPR)", fontsize=20)
ax.set_xlabel("False Positive Rate (FPR)", fontsize=20)
ax.set_title("Receiver Operating Characteristic (ROC) Curve")
plt.show()
```



This curve is closer to the top left corner, which is ideal because we see a high TPR and a low FPR. This means that the classifier can correctly identify most positives and has few false positives. This represents a high level of discrimination capability.

4. In measuring performance, policymkaers should care more about minimizing false negatives if their priority is ensuring that they are able to catch tax evaders. This

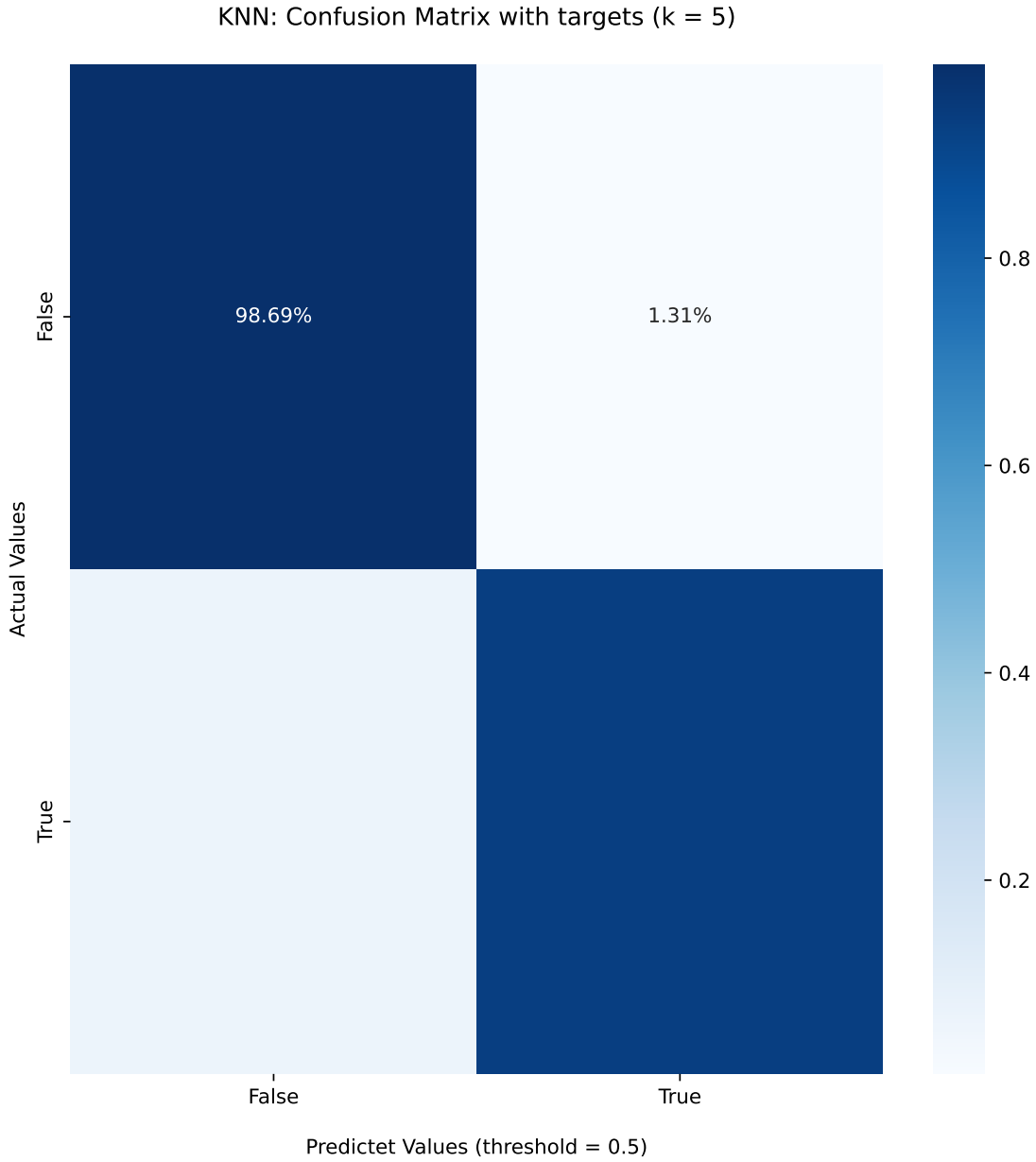
would mean they should have a lower threashold (less FN, higher FP). But if the priority is to not waste resources while auditing, then a higher threshold would perform better (Min FP, higher FN). We see that there is an inherent trade-off here, so government must decide if they want to prioritize curbing tax evasion or minimizing their costs.

5. KNN for validation set

```
knn_5 = KNeighborsClassifier(n_neighbors=5)
knn_5.fit(X_train, target_train)
knn_5_pred = knn_5.predict(X_valid)
```

5.a.

```
knn_cm_5 = confusion_matrix(target_valid, knn_5_pred, normalize="true")
# Display the visualization of the confusion matrix
fig, ax = plt.subplots(figsize=(9, 9))
ax = sns.heatmap(
knn_cm_5, annot=True, fmt=".2%", cmap="Blues"
)
ax.set_title("KNN: Confusion Matrix with targets (k = 5)\n")
ax.set_xlabel("\nPredictet Values (threshold = 0.5)")
ax.set_ylabel("Actual Values")
ax.xaxis.set_ticklabels(["False", "True"])
ax.yaxis.set_ticklabels(["False", "True"])
plt.show()
```



5.b.Error rate

```
error_knn_5 = 1 - accuracy_score(target_valid, knn_5_pred)
print(f"The error rate at the 0.5 threshold for KNN is approximately:
 ↪  {round(error_knn_5, 4)*100}%")
```

```
The error rate at the 0.5 threshold for KNN is approximately: 3.61%
```

HOW ACCURATE

5.c. Precision

```
precision_knn_5 = precision_score(target_valid, knn_5_pred, zero_division=0)
print(f"Positive predictions that were actually correct at the 0.5 threshold:
↪  {precision_knn_5:.2%}")
```

```
Positive predictions that were actually correct at the 0.5 threshold:
98.01%
```

6. Scaling

```
X.dtypes
# we can scale since they are all floats
```

```
Sector_score     float64
PARA_A           float64
Risk_A           float64
PARA_B           float64
Risk_B           float64
Money_Value      float64
Risk_D           float64
Score            float64
Inherent_Risk    float64
Audit_Risk       float64
dtype: object
```

```
scaler = StandardScaler()  # Initialize the scalar
X_scaled = scaler.fit_transform(X)
print(type(X_scaled))
cols = X.columns
X_final = pd.DataFrame(X_scaled, columns=cols)
X_final.head()
```

```
<class 'numpy.ndarray'>
```

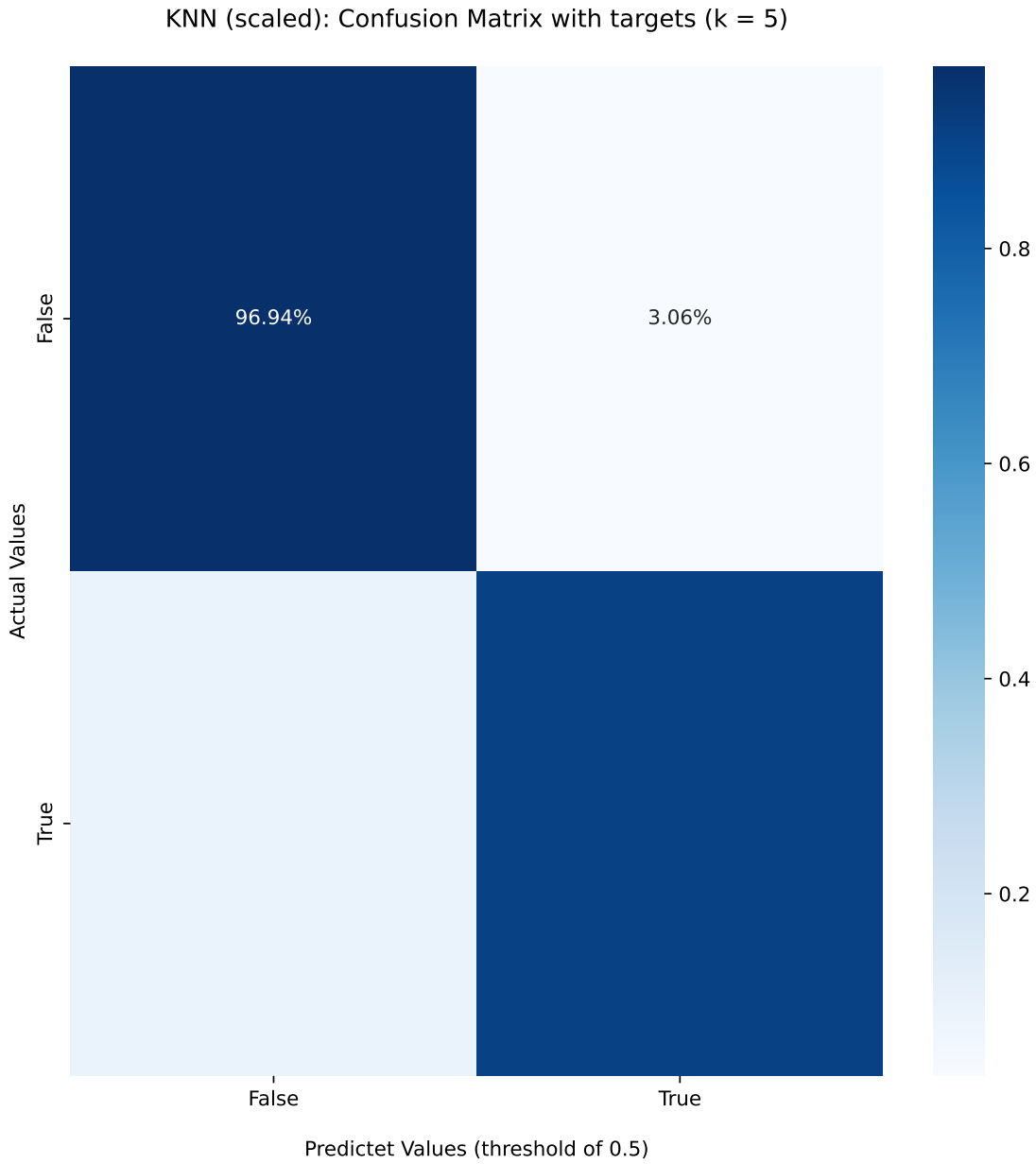
	Sector_score	PARA_A	Risk_A	PARA_B	Risk_B	Money_Value	Risk_D	Score
0	-0.669071	0.304129	0.335827	-0.166006	-0.194273	-0.161614	-0.190146	-0.353484
1	-0.669071	-0.432005	-0.393216	-0.119482	-0.178777	-0.198271	-0.202356	-0.819385
2	-0.669071	-0.342190	-0.363566	-0.211331	-0.209370	-0.212393	-0.207059	-0.819385
3	-0.669071	-0.432005	-0.393216	-0.000278	0.004583	-0.035870	-0.030676	1.976022
4	-0.669071	-0.432005	-0.393216	-0.214326	-0.210368	-0.212393	-0.207059	-0.819385

Now, splitting the data

```
X_train, X_valid, target_train, target_valid = train_test_split(
X_final, target, train_size=0.50, random_state=13
)
# Fit the model
knn_5_scaled = KNeighborsClassifier(n_neighbors=5).fit(X_train, target_train)
# Apply the model to the validation set
knn_5_scaled_pred = knn_5_scaled.predict(X_valid)
```

6.a.

```
# Construct confusion matrix
knn_scaled_cm_5 = confusion_matrix(target_valid, knn_5_scaled_pred,
 ↪  normalize="true")
# Display the visualization of the confusion matrix
fig, ax = plt.subplots(figsize=(9, 9))
ax = sns.heatmap(
knn_scaled_cm_5, annot=True, fmt=".2%", cmap="Blues"
)
ax.set_title("KNN (scaled): Confusion Matrix with targets (k = 5)\n")
ax.set_xlabel("\nPredictet Values (threshold of 0.5)")
ax.set_ylabel("Actual Values")
ax.xaxis.set_ticklabels(["False", "True"])
ax.yaxis.set_ticklabels(["False", "True"])
plt.show()
```



6.b.

```
error_knn_scaled_5 = 1 - accuracy_score(target_valid, knn_5_scaled_pred)
print(f"The error rate at the 0.5 threshold for KNN (scaled) is approximately:
↪  {round(error_knn_scaled_5, 4)*100}%")
```

```
The error rate at the 0.5 threshold for KNN (scaled) is approximately:
5.67%
```

How accurate

6.c.

```
precision_knn_scaled_5 = precision_score(target_valid, knn_5_scaled_pred,
↪  zero_division=0)
print(f"Positive predictions that were actually correct at the 0.5 threshold:
↪  {precision_knn_scaled_5:.2%}"
)
```

```
Positive predictions that were actually correct at the 0.5 threshold:
95.36%
```

7. The non-normalized KNN model appears to perform better in this case (Error rate is 3.61%, Precision is 98.01%) compared to the normalized one (Error rate is 5.67%, Precision is 95.36%). This is somewhat unexpected, as normalization usually improves KNN performance. Maybe the original scale of the features already captures important information about their relative importance.

8. For KNN, which k yields the lowest error rate? By 5-fold cross-validation (5FCV), find the k with the lowest classification error rate. Briefly explain. 6Use the entire dataset for 5FCV, shuffle the data randomly for splitting, and set random_state=13

```
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.neighbors import KNeighborsClassifier

## Re-split data so it is not normalized
X_train, X_valid, target_train, target_valid = train_test_split(
X, target, train_size=0.50, random_state=13
)


# Select a range of KNN neighbors (k) to evaluate
ks = list(range(1, 37, 2))
param_grid = {'n_neighbors': ks}

# Initialize the KNN classifier
knn = KNeighborsClassifier()

# Set up 5-fold cross-validation scheme
kfcv = KFold(5, random_state=13, shuffle=True)

# Perform GridSearchCV
knn_cv = GridSearchCV(knn, param_grid, cv=kfcv)
knn_cv.fit(X_train, target_train)
knn_cv_pred = knn_cv.predict(X_valid)
```

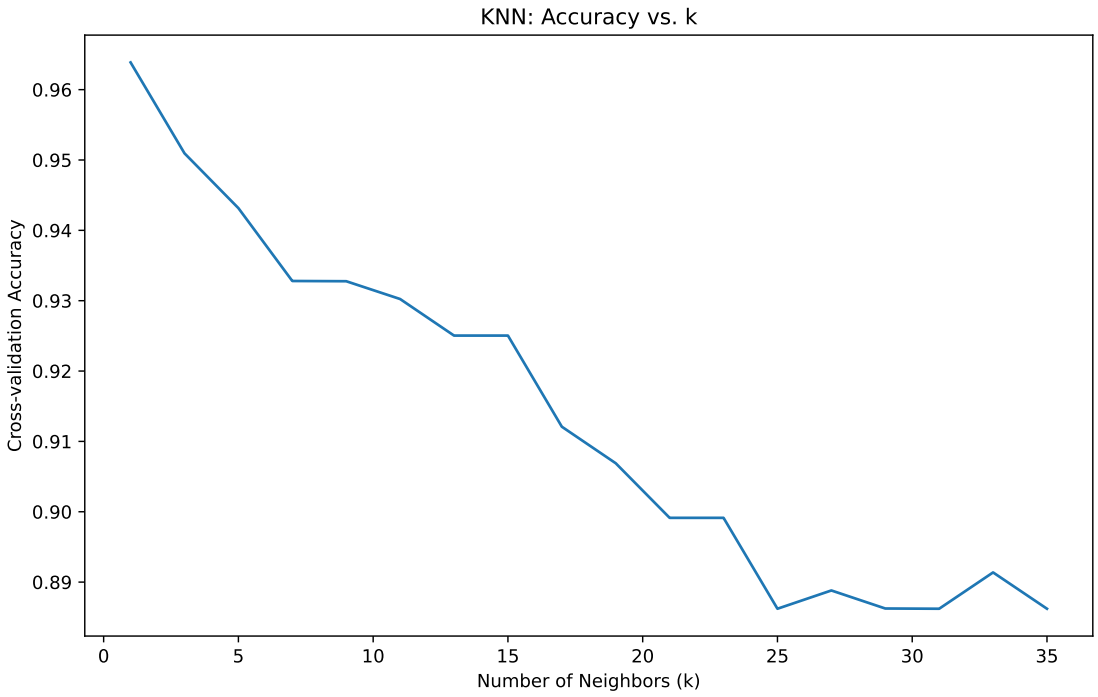
```
knn_cv_pred_acc = accuracy_score(target_valid, knn_cv_pred)
# Get the best k and its corresponding score
best_k = knn_cv.best_params_['n_neighbors']
best_score = knn_cv.best_score_

print("Best parameters:", knn_cv.best_params_)
print("Best cross-validation score:", knn_cv.best_score_)
print("Accuracy Score:", knn_cv_pred_acc)




# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(ks, knn_cv.cv_results_['mean_test_score'])
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Cross-validation Accuracy')
plt.title('KNN: Accuracy vs. k')
plt.show()
```

```
Best parameters: {'n_neighbors': 1}
Best cross-validation score: 0.9638694638694638
Accuracy Score: 0.9587628865979382
```



The optimal k for KNN using the 5FCV is k=1 because it gives us the best cross-validation score of approximately 96.39% and an accuracy score of 95.88% on the validation set. It is highly flexible since each observation is classified based solely on a single nearest neighbor. This can lead to low bias but potentially high variance. Since this flexible model is complex, it is likely that the relationship between the predictors and the dependent variable, RISK, is a complex one.

9. Since the dataset only contains firms that were audited, we mmay run into the problem of selection bias. We are looking at firms who we already suspected of foul play, so the model doesn't really capture the entire population of firms. In the long run, the model may not detect new patterns of tax evasion for unaudited firms and instead overfit on characteristics of firms that are in the data/audited. To deal with this, the government should regularly feed the model with new data and this time do a random sampling so that we can avoid the bias. Finally, we should make sure to regularly evaluate the model's performance.