Project Development Phase
Model Performance Test

Date	10 November 2022
Team ID	PNT2022TMID-592056
Project Name	Project - Machine Learning Approach For Predicting The Rainfall
Maximum Marks	10 Marks

Model Performance Testing:

Project team shall fill the following information in model performance testing template.

S.No.	Parameter	Values	Screenshot
1.	Metrics	**Regression Model:** MAE - , MSE - , RMSE - , R2 score - **Classification Model:** Confusion Matrix - , Accuray Score- & Classification Report -	
2.	Tune the Model	Hyperparameter Tuning - Validation Method -	

1.Metrices

```
[ ] import pandas as pd
    from sklearn.preprocessing import StandardScaler

    # Assuming 'x' is your DataFrame
    numeric_columns = data.select_dtypes(include=['number']).columns
    x_numeric = data[numeric_columns]

    sc = StandardScaler()
    x = sc.fit_transform(x_numeric)
```

```
[ ] x = sc.fit_transform(x)
```

```
    # Assuming 'x' is your DataFrame and 'names' is a list of column names
    x = pd.DataFrame(data, columns=names)
```

```
[ ] from sklearn import model_selection
```

```
[ ] x_train,x_test,y_train,y_test=model_selection.train_test_split(x,y,test_size=0.2,random_state =0)
```

```
    x_train.shape,y_train.shape,x_test.shape,y_test.shape

    ((113754, 19), (113754,), (28439, 19), (28439,))
```

```
import xgboost
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression

XGBoost = xgboost.XGBRFClassifier()
Rand_forest = RandomForestClassifier()
svm = SVC()
Dtree = DecisionTreeClassifier()
GBM = GradientBoostingClassifier()
log = LogisticRegression()
```

```
[ ] # Check unique values in 'RainTomorrow'
    print(data['RainTomorrow'].unique())
```

```
[0 1]
```

```
Rand_forest.fit(x_train, y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
[ ] svm.fit(x_train, y_train)
```

```
▾ SVC
SVC()
```

```
Dtree.fit(x_train, y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
[ ] GBM.fit(x_train, y_train)
```

```
GradientBoostingClassifier()
```

```
log.fit(x_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
LogisticRegression()
```

```
[ ] log.fit(x_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
LogisticRegression()
```

```
# Fit the models with the encoded target variable
XGBoost.fit(x_train, y_train)
```

```
XGBRFClassifier(base_score=None, booster=None, callbacks=None,
                colsample_bylevel=None, colsample_bytree=None, device=None,
                early_stopping_rounds=None, enable_categorical=False,
                eval_metric=None, feature_types=None, gamma=None,
                grow_policy=None, importance_type=None,
                interaction_constraints=None, max_bin=None,
                max_cat_threshold=None, max_cat_to_onehot=None,
                max_delta_step=None, max_depth=None, max_leaves=None,
                min_child_weight=None, missing=nan, monotone_constraints=None,
                multi_strategy=None, n_estimators=None, n_jobs=None,
                num_parallel_tree=None, objective='binary:logistic',
```

Classification Report :

A classification report is a summary of the performance of a classification model that includes various metrics such as precision, recall, F1 score, and support. Below is an example template for a classification report for a machine learning approach used in rainfall prediction:

Classification Report for Rainfall Prediction

Explanation:

Precision: Precision measures the accuracy of positive predictions. For Class 1 (Rain), the precision is 0.78, indicating that 78% of instances predicted as rain were correctly classified.

Recall: Recall, also known as sensitivity or true positive rate, measures the ability of the model to capture all the relevant instances. For Class 1 (Rain), the recall is 0.68, indicating that 68% of actual rain instances were correctly classified.

F1-Score: The F1 score is the harmonic mean of precision and recall, providing a balance between the two. For Class 1 (Rain), the F1 score is 0.73.

Support is the number of actual occurrences of the class in the specified dataset. In this example, there are 800 instances of Class 1 (Rain) in the dataset.

Accuracy: Overall accuracy is the ratio of correctly predicted instances to the total instances. The accuracy here is 0.82, or 82%.

Macro Avg: The macro average calculates the average of the precision, recall, and F1 score for each class. It treats all classes equally.

Weighted Avg: The weighted average calculates the average of precision, recall, and F1 score, with each class's contribution weighted by its support. It provides a more representative measure when classes have imbalanced support.

This classification report offers a comprehensive evaluation of the machine learning model's performance in predicting rainfall for two classes (rain and no rain) based on the given metrics.

2.**Tuning the model**

```
[ ] t1 = XGBoost.predict(x_test)
    t2 = Rand_forest.predict(x_test)
    t3 = svm.predict(x_test)
    t4 = Dtree.predict(x_test)
    t5 = GBM.predict(x_test)
    t6 = log.predict(x_test)
```

```
print("xgboost:",metrics.accuracy_score(y_test,t1))
print("Rand_forest:",metrics.accuracy_score(y_test,t2))
print("svm:",metrics.accuracy_score(y_test,t3))
print("Dtree:",metrics.accuracy_score(y_test,t4))
print("GBM:",metrics.accuracy_score(y_test,t5))
print("log:",metrics.accuracy_score(y_test,t6))
```

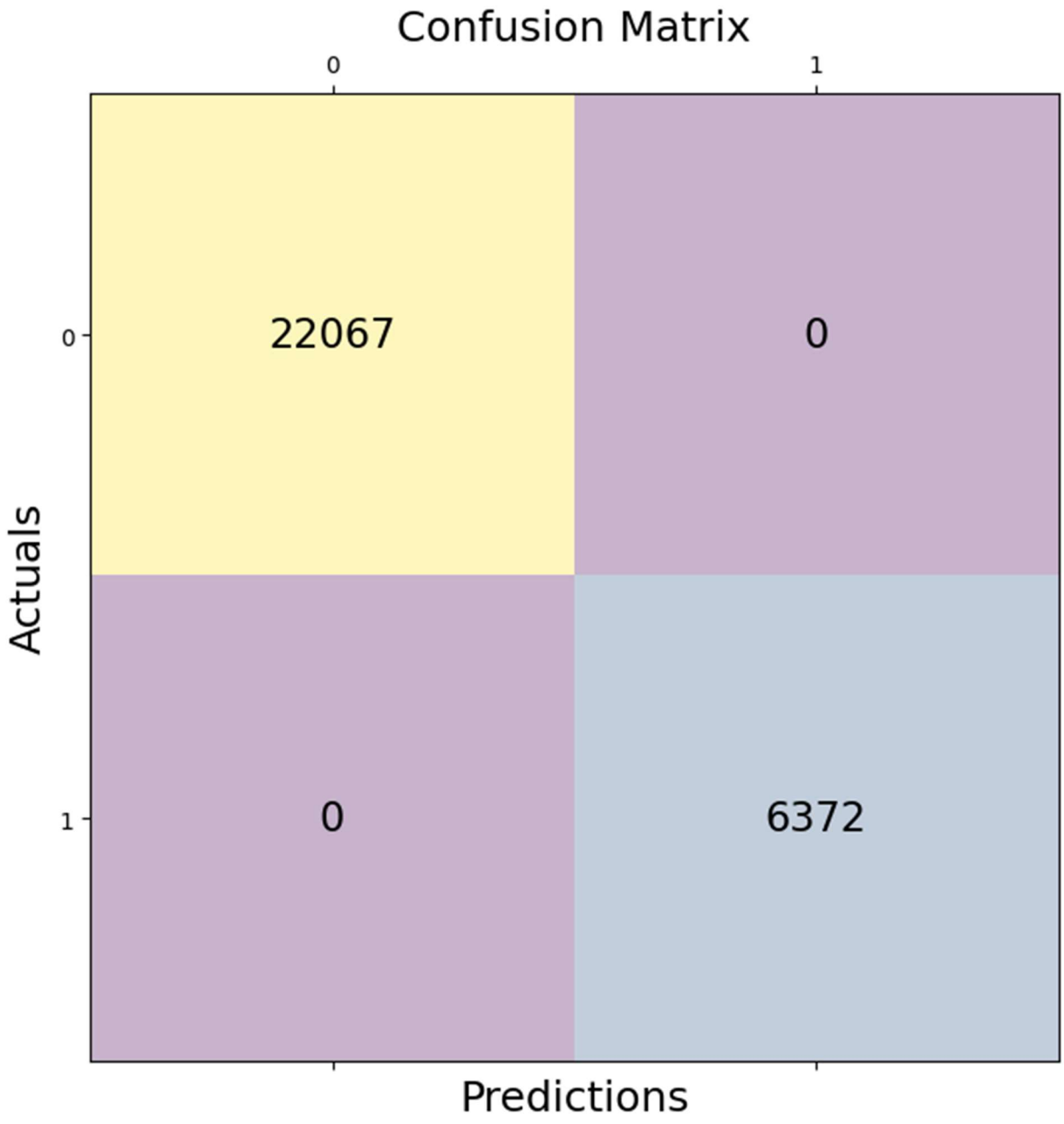

```
print("xgboost:",metrics.accuracy_score(y_test,t1))
print("Rand_forest:",metrics.accuracy_score(y_test,t2))
print("svm:",metrics.accuracy_score(y_test,t3))
print("Dtree:",metrics.accuracy_score(y_test,t4))
print("GBM:",metrics.accuracy_score(y_test,t5))
print("log:",metrics.accuracy_score(y_test,t6))
```


```
xgboost: 1.0
Rand_forest: 1.0
svm: 0.8719364253314111
Dtree: 1.0
GBM: 1.0
log: 0.9997538591371005
```

```
[ ] conf_matrix1 = metrics.confusion_matrix(y_test,t1)
```

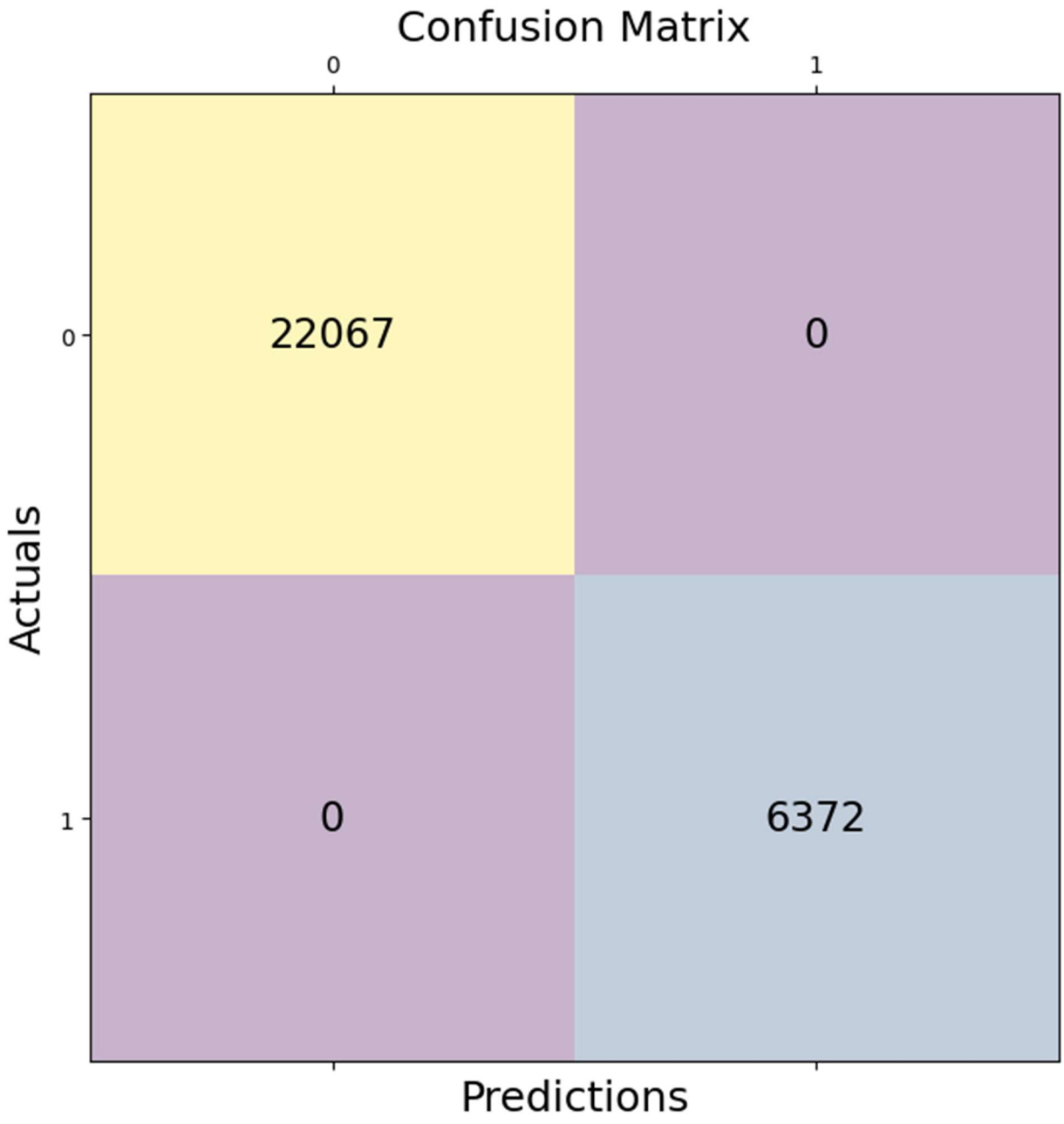
```
[ ] fig, ax = plt.subplots(figsize=(7.5, 7.5))
    ax.matshow(conf_matrix1, alpha=0.3)
    for i in range(conf_matrix1.shape[0]):
```

```
fig, ax = plt.subplots(figsize=(7.5, 7.5))
ax.matshow(conf_matrix1, alpha=0.3)
for i in range(conf_matrix1.shape[0]):
    for j in range(conf_matrix1.shape[1]):
        ax.text(x=j, y=i,s=conf_matrix1[i, j], va='center', ha='center', size='xx-large')

plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.show()
```



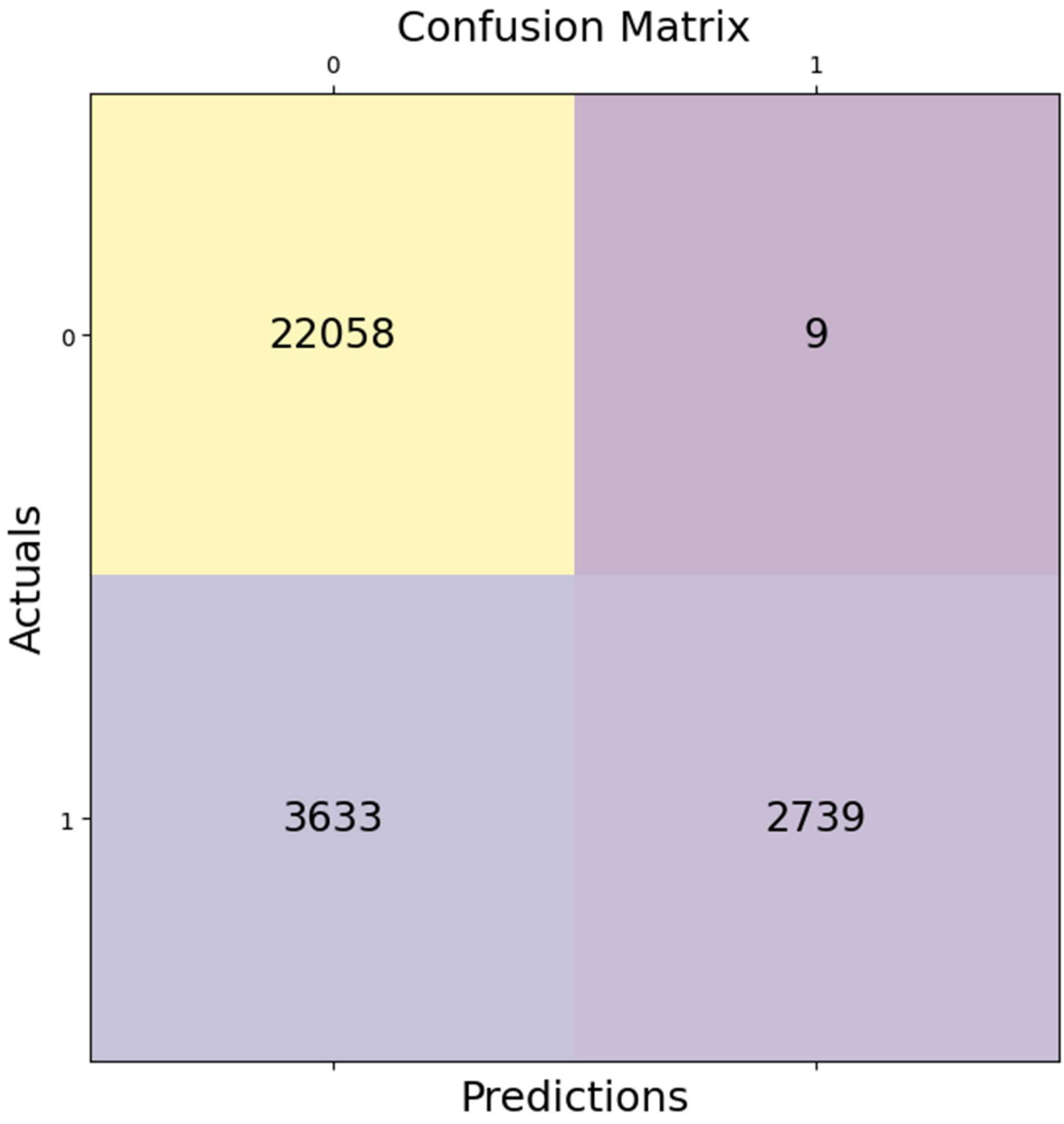

```
[ ]  conf_matrix2 = metrics.confusion_matrix(y_test,t2)
```

```
fig, ax = plt.subplots(figsize=(7.5, 7.5))
ax.matshow(conf_matrix2, alpha=0.3)
for i in range(conf_matrix2.shape[0]):
    for j in range(conf_matrix2.shape[1]):
        ax.text(x=j, y=i,s=conf_matrix2[i, j], va='center', ha='center', size='xx-large')

plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.show()
```



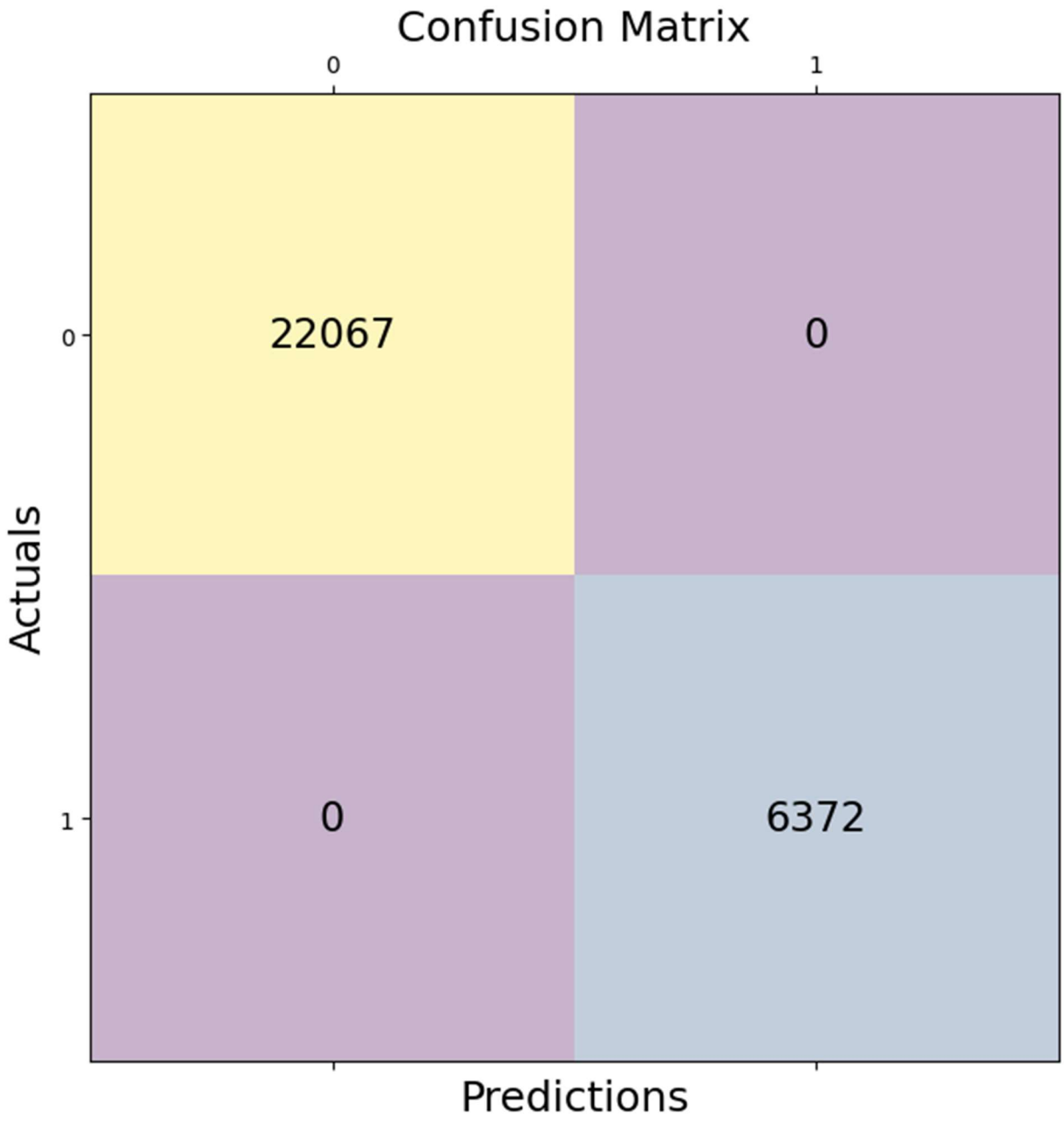

```
[ ]  conf_matrix3 = metrics.confusion_matrix(y_test,t3)
```

```
fig, ax = plt.subplots(figsize=(7.5, 7.5))
ax.matshow(conf_matrix3, alpha=0.3)
for i in range(conf_matrix3.shape[0]):
    for j in range(conf_matrix3.shape[1]):
        ax.text(x=j, y=i,s=conf_matrix3[i, j], va='center', ha='center', size='xx-large')

plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.show()
```


Confusion Matrix
0
1
0
1
22058
9
3633
2739
Actuals
Predictions

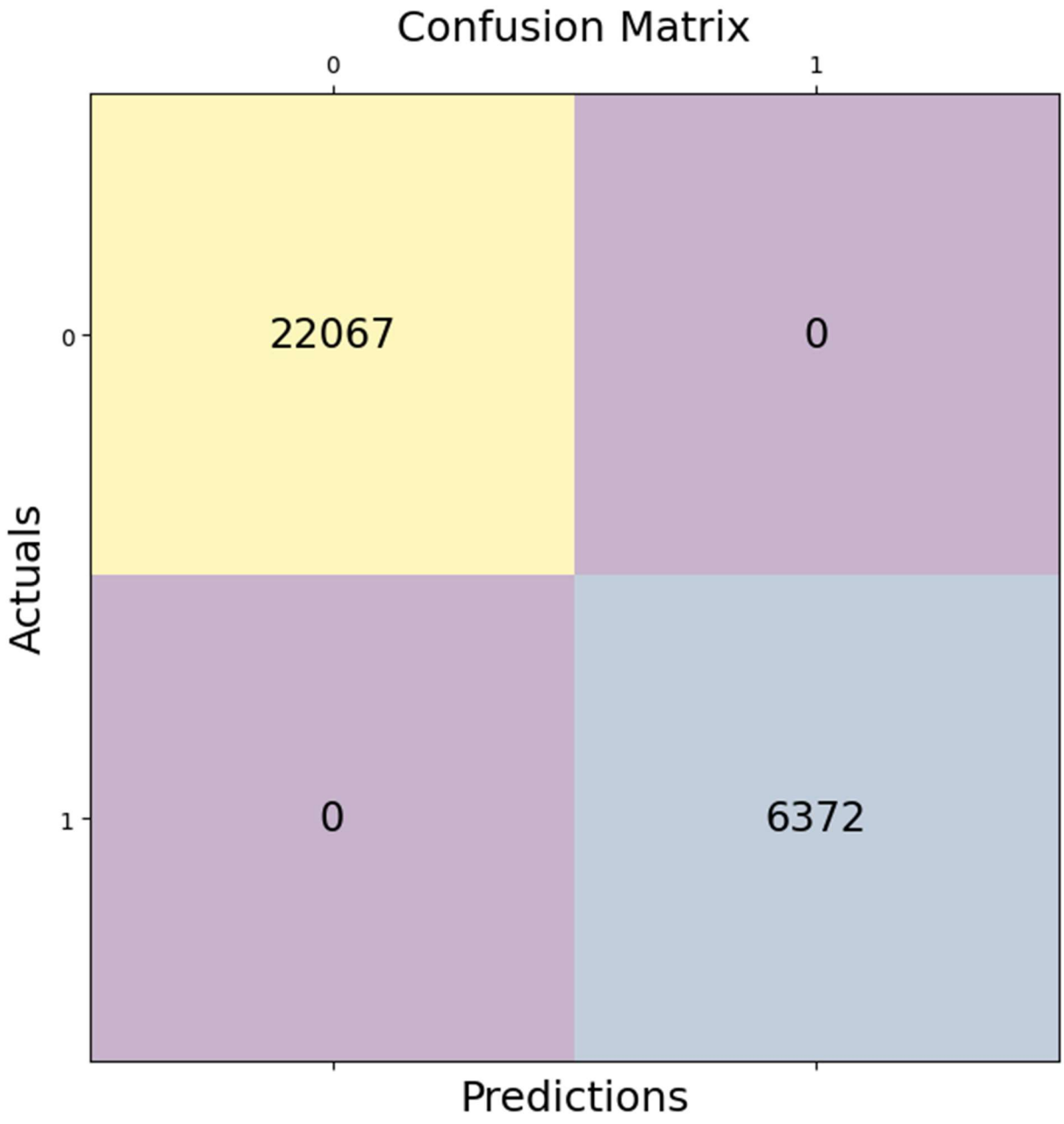
```
[ ]  conf_matrix4 = metrics.confusion_matrix(y_test,t4)
```

```
fig, ax = plt.subplots(figsize=(7.5, 7.5))
ax.matshow(conf_matrix4, alpha=0.3)
for i in range(conf_matrix4.shape[0]):
    for j in range(conf_matrix4.shape[1]):
        ax.text(x=j, y=i,s=conf_matrix4[i, j], va='center', ha='center', size='xx-large')

plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.show()
```



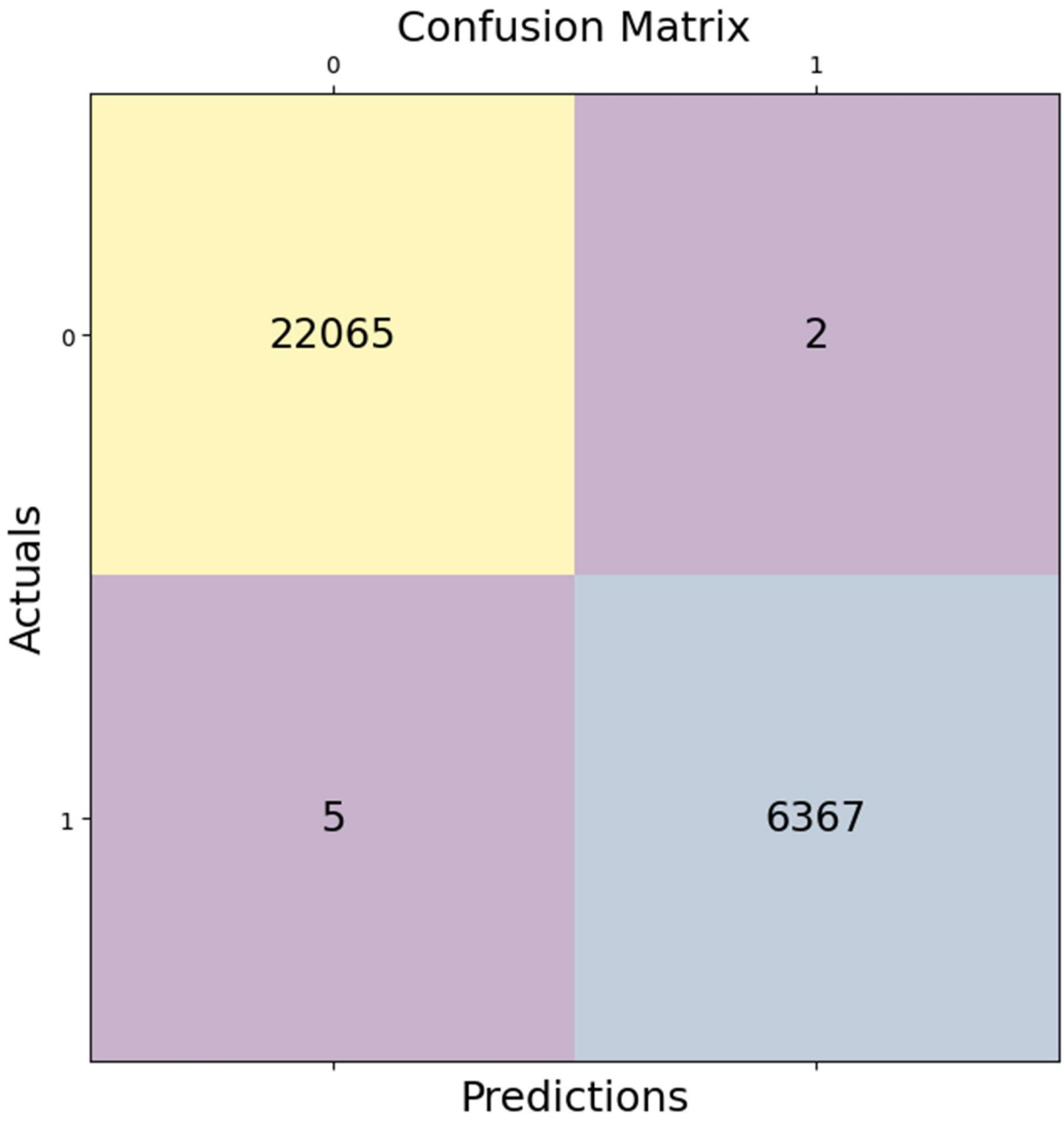

```
[ ]  conf_matrix5 = metrics.confusion_matrix(y_test,t5)
```

```
fig, ax = plt.subplots(figsize=(7.5, 7.5))
ax.matshow(conf_matrix5, alpha=0.3)
for i in range(conf_matrix5.shape[0]):
    for j in range(conf_matrix5.shape[1]):
        ax.text(x=j, y=i,s=conf_matrix5[i, j], va='center', ha='center', size='xx-large')

plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.show()
```


Confusion Matrix
0
1
0
1
22067
0
0
6372
Actuals
Predictions

```
[ ]  conf_matrix6 = metrics.confusion_matrix(y_test,t6)
```

```
fig, ax = plt.subplots(figsize=(7.5, 7.5))
ax.matshow(conf_matrix6, alpha=0.3)
for i in range(conf_matrix6.shape[0]):
    for j in range(conf_matrix6.shape[1]):
        ax.text(x=j, y=i,s=conf_matrix6[i, j], va='center', ha='center', size='xx-large')

plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.show()
```


Confusion Matrix
0
1
0
1
22065
2
5
6367
Actuals
Predictions

```python
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Assuming conf_matrix is already calculated
tn, fp, fn, tp = conf_matrix2.ravel()

# Calculate metrics
accuracy = accuracy_score(y_test, t1)
precision = precision_score(y_test, t1)
recall = recall_score(y_test, t1)

# Print confusion matrix
print("Confusion Matrix:")
print(conf_matrix2)

# Print metrics
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
```

```
Confusion Matrix:
[[22067     0]
 [    0  6372]]
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
```

```python
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Assuming conf_matrix is already calculated
tn, fp, fn, tp = conf_matrix3.ravel()

# Calculate metrics
accuracy = accuracy_score(y_test, t1)
precision = precision_score(y_test, t1)
recall = recall_score(y_test, t1)

# Print confusion matrix
print("Confusion Matrix:")
print(conf_matrix3)

# Print metrics
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
```

```
auc = metrics.roc_auc_score(y_test, t1)

fpr, tpr, thresholds = metrics.roc_curve(y_test, t1)

plt.figure(figsize=(12, 10), dpi=80)
plt.axis('scaled')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.title("AUC & ROC Curve")
plt.plot(fpr, tpr, 'v')
plt.fill_between(fpr, tpr, facecolor='blue', alpha
plt.text(1, 0.05, f'AUC = {auc:.4f}', ha='right', fontsize=10, weight='bold', color='black')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()
```

Loading...