In [ ]:
```
# Student ID
```

In [7]:
```
# Please enter your student ID here
student_id = "12319879"

# Print the student ID
print("Student ID:", student_id)
```

```
Student ID: 12319879
```

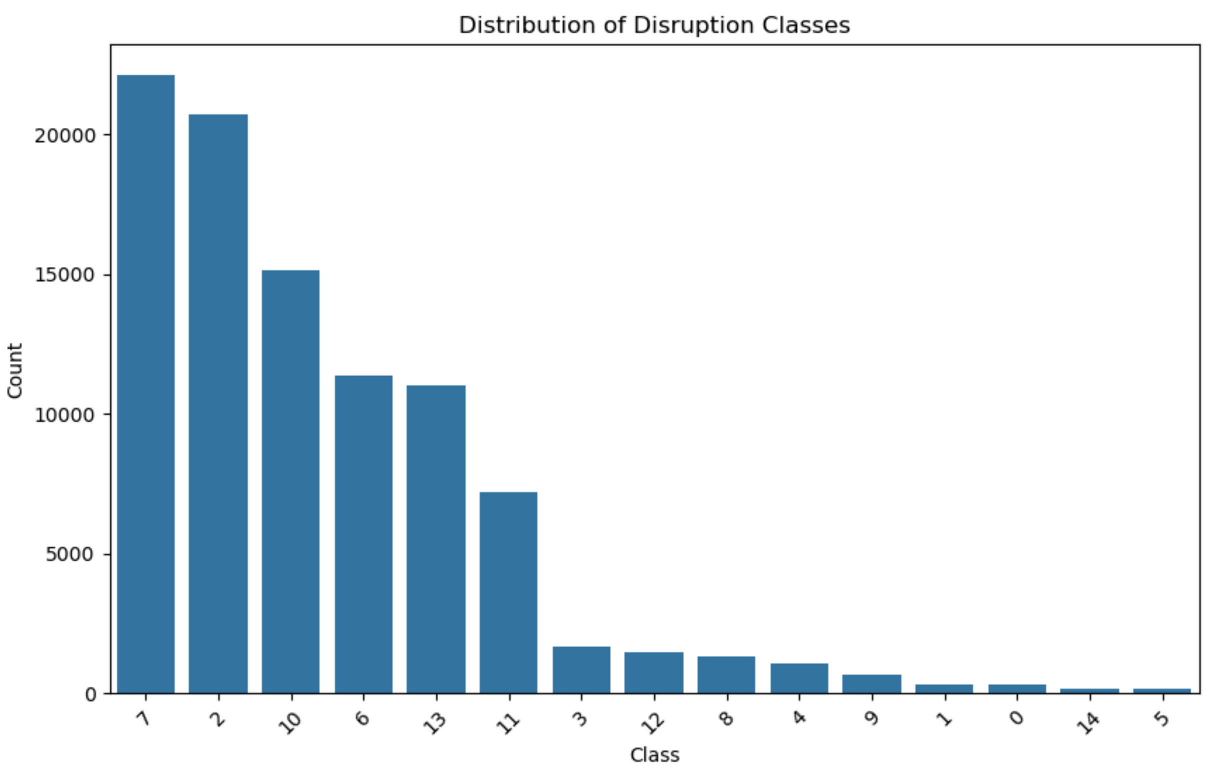
In [ ]:
```
# Setup Python and load data
```

In [2]:
```
import os
import pickle
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder

# Load the preprocessed data
data_path = '/home/e12319879/shared/188.995-2024W/data/data_processed.pickle'
with open(data_path, 'rb') as fp:
    data_processed = pickle.load(fp)

# Display the column names to understand the structure of the DataFrame
print("Column names in the DataFrame:", data_processed.columns)

# Mapping dictionary for cleaning disruptions
mapping = {
    'Polizeieinsatz': 'Operation',
    'Rettungseinsatz': 'Operation',
    'Polizeieinsatz Verspätungen': 'Operation',
    'Feuerwehreinsatz': 'Operation',
    'Rettungseinsatz Verspätungen': 'Operation',
    'Schadhaftes Fahrzeug': 'Vehicle in poor condition',
    'Schadhaftes Fahrzeug Verspätungen': 'Vehicle in poor condition',
    'Wagengebrechen': 'Vehicle in poor condition',
    'Schadhafter Zug': 'Vehicle in poor condition',
    'Fahrzeug Verspätungen': 'Vehicle in poor condition',
    'Fahrzeug': 'Vehicle in poor condition',
    'erhöhtes Fahrgastaufkommen': 'Increased passenger volume',
    'Erhöhtes Fahrgastaufkommen': 'Increased passenger volume',
    'erhöhtes Fahrgastaufkommen Verspätungen': 'Increased passenger volume',
    'Verspätungen': 'Delay',
    'Verspätung': 'Delay',
    'Verkehrsunfall Verspätungen': 'Traffic accident',
    'Verkehrsunfall': 'Traffic accident',
    'Fremder Verkehrsunfall': 'Traffic accident',
    'Fremder Verkehrsunfall Verspätungen': 'Traffic accident',
    'Verkehrsstörung Verspätungen': 'Traffic jam',
    'Verkehrsstörung': 'Traffic jam',
    'Verkehrsbedingte Verspätung': 'Traffic jam',
    'Verkehrsbedingte ': 'Traffic jam',
    'Verkehrsbedingte Verspätungen': 'Traffic jam',
```

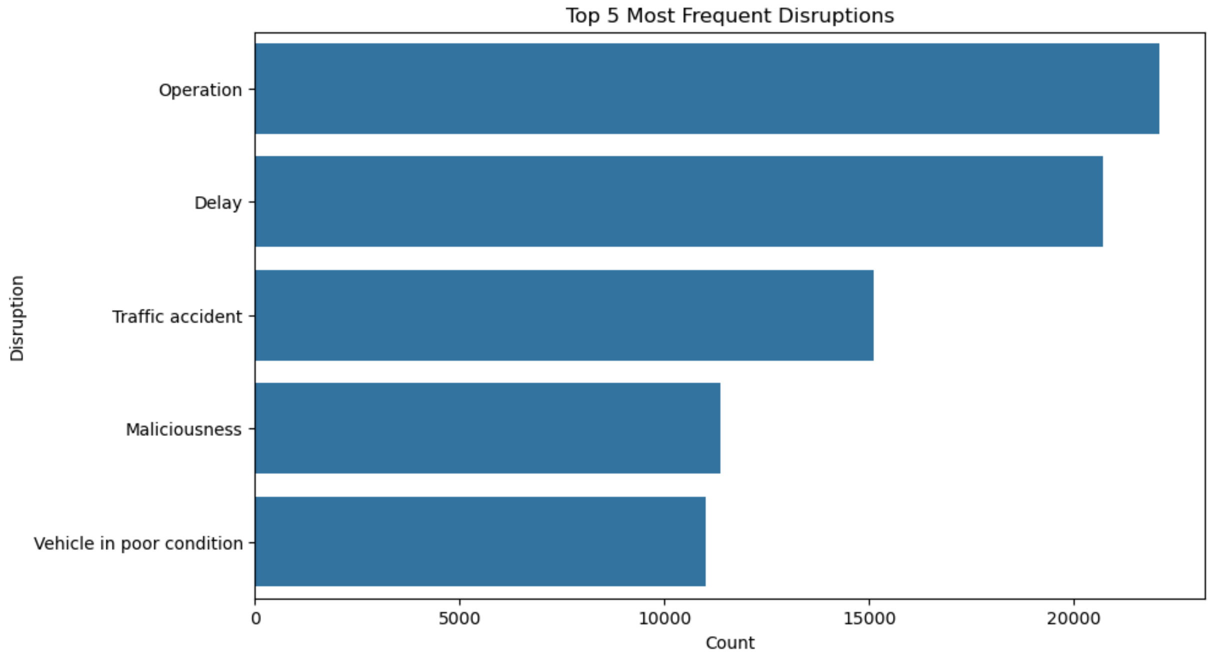
```python
    'Verkehrsbedingt': 'Traffic jam',
    'Verkehrsbedingt Verspätungen': 'Traffic jam',
    'Verkehrsbedingte Verspätung Verspätungenen': 'Traffic jam',
    'Verkehrsbedingte Verspätung Verspätungen': 'Traffic jam',
    'Veranstaltung': 'Event',
    'Vienna': 'Event',
    'Vienna-City-Marathon': 'Event',
    'Regenbogenparade': 'Event',
    'Demonstration': 'Event',
    'Staatsbesuch': 'Event',
    'Opernball': 'Event',
    'Erkrankung eines Fahrgastes': 'Personnel problems',
    'Erkrankung eines': 'Personnel problems',
    'Erkrankung': 'Personnel problems',
    'Fahrleitungsgebrechen': 'General infrastructure',
    'Wasserrohrgebrechen': 'General infrastructure',
    'Stromstörung': 'General infrastructure',
    'Gasrohrgebrechen': 'General infrastructure',
    'Gleisschaden': 'Transportation infrastructure',
    'Weichenstörung': 'Transportation infrastructure',
    'Gleisbauarbeiten': 'Transportation infrastructure',
    'Signalstörung': 'Transportation infrastructure',
    'Signalstörung Verspätungen': 'Transportation infrastructure',
    'Stellwerkstörung': 'Transportation infrastructure',
    'Betriebsstörung': 'Operational disruption',
    'Betriebseinstellung': 'Operational disruption',
    'Fahrtbehinderung': 'Maliciousness',
    'Sachbeschädigung': 'Maliciousness',
    'Falschparker': 'Maliciousness',
    'Witterungsbedingt': 'Weather',
    'Sturmschaden': 'Weather',
    'Bauarbeiten': 'Construction work',
    'Umleitung': 'Construction work',
    'Verunreinigung': 'Contamination'
}

# Update the 'disruption' column
data_processed['disruption'] = data_processed['disruption'].replace(mapping)

# Transform the target column
label_encoder = LabelEncoder()
data_processed['class'] = label_encoder.fit_transform(data_processed['disruption'])

# Verify the transformation
assert data_processed['class'].nunique() == 15, "There should be 15 classes"

# Plot 1: Distribution of Classes
plt.figure(figsize=(10, 6))
sns.countplot(data=data_processed, x='class', order=data_processed['class'].value_c
plt.title('Distribution of Disruption Classes')
plt.xlabel('Class')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()

# Check if 'date' column exists and plot accordingly
```

```
if 'date' in data_processed.columns:
    data_processed['date'] = pd.to_datetime(data_processed['date'])

    # Plot 2: Duration of Disruptions Over Time
    plt.figure(figsize=(12, 6))
    sns.lineplot(data=data_processed, x='date', y='duration', hue='class')
    plt.title('Duration of Disruptions Over Time')
    plt.xlabel('Date')
    plt.ylabel('Duration')
    plt.legend(title='Class', bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.show()
else:
    print("The 'date' column is not present in the DataFrame.")

# Plot 3: Top 5 Most Frequent Disruptions
top_5_disruptions = data_processed['disruption'].value_counts().nlargest(5).index
filtered_data = data_processed[data_processed['disruption'].isin(top_5_disruptions)

plt.figure(figsize=(10, 6))
sns.countplot(data=filtered_data, y='disruption', order=top_5_disruptions)
plt.title('Top 5 Most Frequent Disruptions')
plt.xlabel('Count')
plt.ylabel('Disruption')
plt.show()
```

```
Column names in the DataFrame: Index(['temp_dailyMin', 'temp_dailyMax', 'temp_dailyM
ean', 'temp_dailyMedian',
       'hum_dailyMin', 'hum_dailyMax', 'hum_dailyMean', 'wind_dailyMin',
       'wind_dailyMax', 'wind_dailyMean', 'precip', 'disruption', 'bus',
       'subway', 'tram', 'duration'],
      dtype='object')
```




```
The 'date' column is not present in the DataFrame.
```

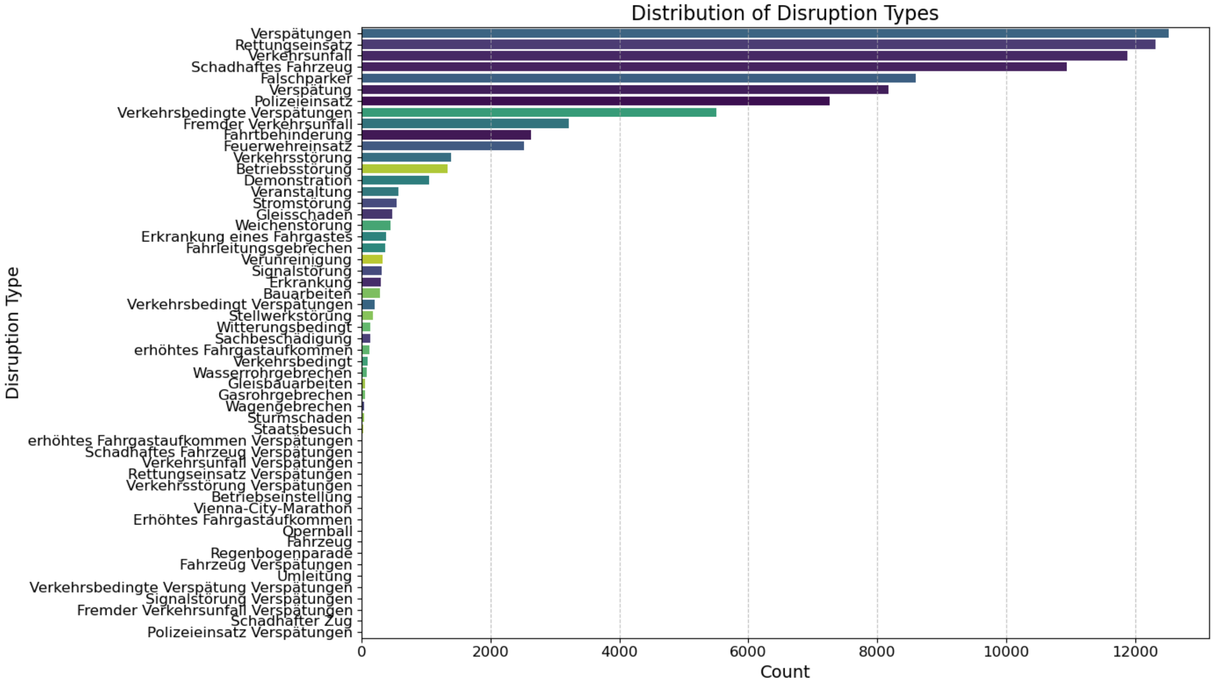


In [ ]:
```
# Task 6: Visualization
```

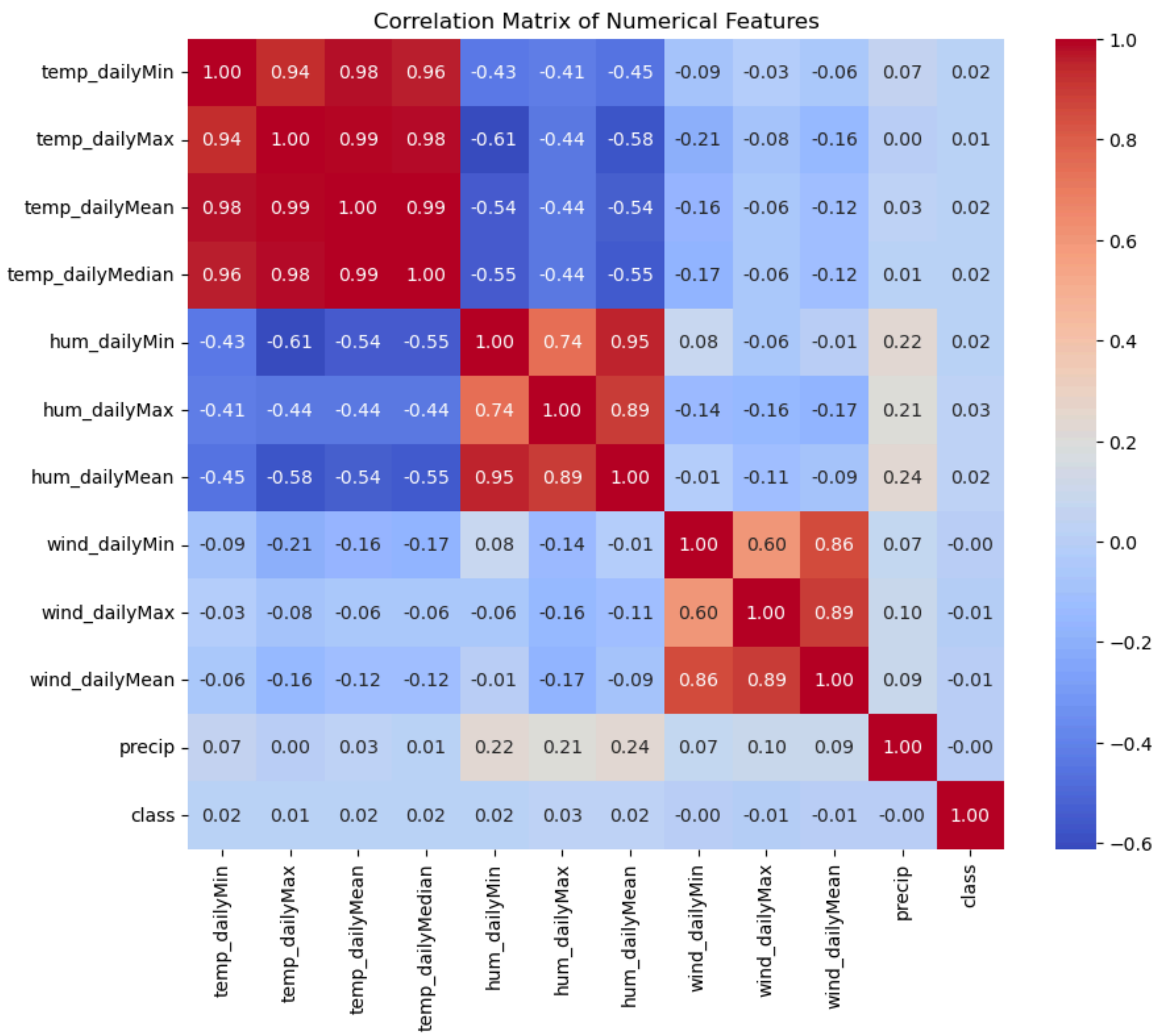
In [40]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
import pickle

# Load the preprocessed data
data_path = '/home/e12319879/shared/188.995-2024W/data/data_processed.pickle'
with open(data_path, 'rb') as fp:
    data_processed = pickle.load(fp)

# Check and print the column names for reference
print("Column names in the DataFrame:", data_processed.columns)

# Mapping dictionary for cleaning disruptions (if needed)
mapping = {
    # (Add mappings here if disruptions need to be cleaned)
}

# Update the 'disruption' column using the mapping
data_processed['disruption'] = data_processed['disruption'].replace(mapping)

# Transform the target column using LabelEncoder
label_encoder = LabelEncoder()
data_processed['class'] = label_encoder.fit_transform(data_processed['disruption'])

# Plot 1: Distribution of Disruptions (Bar Plot using Seaborn)
plt.figure(figsize=(14, 8))  # Increased figure size for better readability
sns.countplot(
    data=data_processed,
    y='disruption',
    order=data_processed['disruption'].value_counts().index,
    palette='viridis',  # Added a color palette for better distinction
```

```
    hue='disruption',  # Assigning hue to the same variable to avoid warning
    dodge=False  # Disable dodging to ensure bars are not split
)
plt.title('Distribution of Disruption Types', fontsize=16)
plt.xlabel('Count', fontsize=14)
plt.ylabel('Disruption Type', fontsize=14)
plt.xticks(rotation=0, fontsize=12)  # Adjusted rotation and font size for clarity
plt.yticks(fontsize=12)  # Adjusted font size for y-ticks
plt.grid(axis='x', linestyle='--', alpha=0.7)  # Added gridlines for x-axis
plt.legend([],[], frameon=False)  # Remove legend as it's not needed
plt.tight_layout()  # Ensures everything fits within the figure area
plt.show()

# Plot 2: Monthly Trends of Disruptions Over Time (Line Plot using Matplotlib)
if 'date' in data_processed.columns:
    data_processed['date'] = pd.to_datetime(data_processed['date'])
    data_processed['month_year'] = data_processed['date'].dt.to_period('M')

    monthly_disruptions = data_processed.groupby('month_year').size()

    plt.figure(figsize=(14, 7))
    monthly_disruptions.plot(kind='line', marker='o')
    plt.title('Monthly Trends of Disruptions Over Time')
    plt.xlabel('Month-Year')
    plt.ylabel('Number of Disruptions')
    plt.xticks(rotation=45)
    plt.grid(True)
    plt.show()
else:
    print("The 'date' column is not present in the DataFrame, skipping time trend p

# Plot 3: Correlation Heatmap (Using Seaborn)
# Assuming numerical features exist; adjust as needed
numerical_cols = data_processed.select_dtypes(include=['int64', 'float64']).columns
correlation_matrix = data_processed[numerical_cols].corr()

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Numerical Features')
plt.show()
```

```
Column names in the DataFrame: Index(['temp_dailyMin', 'temp_dailyMax', 'temp_dailyM
ean', 'temp_dailyMedian',
       'hum_dailyMin', 'hum_dailyMax', 'hum_dailyMean', 'wind_dailyMin',
       'wind_dailyMax', 'wind_dailyMean', 'precip', 'disruption', 'bus',
       'subway', 'tram', 'duration'],
      dtype='object')
```




```
The 'date' column is not present in the DataFrame, skipping time trend plot.
```



In [ ]:
```
# Task 7: Model for disruption class prediction
```

In [ ]:
```
# 7.1 Create train, validation, and test splits
```

In [6]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
import pickle

# Load the preprocessed data
data_path = '/home/e12319879/shared/188.995-2024W/data/data_processed.pickle'
with open(data_path, 'rb') as fp:
    data_processed = pickle.load(fp)

def sample_data(df: pd.DataFrame, fraction: float = 0.7) -> pd.DataFrame:
    """
    Sample a fraction of the data.
    """
    # Sample the data
    data_shortened = df.sample(frac=fraction, random_state=12345678)  # Use student

    # Drop the 'disruption' column as it is mapped to 'class'
    if 'disruption' in data_shortened.columns:
        data_shortened = data_shortened.drop(columns=['disruption'])

    # Convert 'duration' from Timedelta to floating-point number in minutes
    if 'duration' in data_shortened.columns:
        data_shortened['duration'] = data_shortened['duration'].dt.total_seconds()

    return data_shortened

# Sample the data
data_shortened = sample_data(data_processed)

# Check the columns in the DataFrame
print("Columns in data_shortened:", data_shortened.columns)

# Identify the correct target column
# For example, let's assume 'duration' is our target column
target_column = 'duration'  # Update this to the correct target column name

# Check if the target column exists
if target_column not in data_shortened.columns:
    raise KeyError(f"Target column '{target_column}' not found in the DataFrame.")

features = data_shortened.drop(columns=[target_column])
target = data_shortened[target_column]

# Split the data into train (80%) and temp (20%)
X_train, X_temp, y_train, y_temp = train_test_split(features, target, test_size=0.2

# Split the temp set into validation (50% of temp) and test (50% of temp), which is
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, rand

# Display the shapes of the splits
print(f"Training set size: {X_train.shape[0]}")
print(f"Validation set size: {X_val.shape[0]}")
print(f"Test set size: {X_test.shape[0]}")
```

```
Columns in data_shortened: Index(['temp_dailyMin', 'temp_dailyMax', 'temp_dailyMea
n', 'temp_dailyMedian',
       'hum_dailyMin', 'hum_dailyMax', 'hum_dailyMean', 'wind_dailyMin',
       'wind_dailyMax', 'wind_dailyMean', 'precip', 'bus', 'subway', 'tram',
       'duration'],
      dtype='object')
Training set size: 53008
Validation set size: 6626
Test set size: 6626
```

In [7]:

```python
import pandas as pd
from sklearn.model_selection import train_test_split

def split_data(df: pd.DataFrame, test_size: float = 0.2, target_column: str = 'clas
    """
    Split the DataFrame into train and test sets.

    Parameters:
    - df: The DataFrame to split.
    - test_size: The proportion of the dataset to include in the test split.
    - target_column: The name of the target column.

    Returns:
    - df_train: The training subset of the DataFrame.
    - df_test: The test subset of the DataFrame.
    """
    # Check if the target column exists, if specified
    if target_column not in df.columns:
        raise KeyError(f"Target column '{target_column}' not found in the DataFrame

    # Perform the train-test split
    df_train, df_test = train_test_split(df, test_size=test_size, random_state=1234

    return df_train, df_test

# Split the data
data_train, data_test = split_data(data_shortened, test_size=0.2, target_column='du

# Assertions to ensure the splits are correct
assert data_train.shape[1] == data_test.shape[1], "Both dataframes should have the 
assert data_train.shape[1] == data_shortened.shape[1], "All columns should be retai
assert data_train.shape[0] < data_shortened.shape[0], "data_train should be a subse
assert data_test.shape[0] < data_shortened.shape[0], "data_test should be a subset 

# Print the sizes of the splits
print(f"Training and validation set size: {data_train.shape[0]}")
print(f"Test set size: {data_test.shape[0]}")
```

```
Training and validation set size: 53008
Test set size: 13252
```

In [8]:

```python
# Further split the training and validation set into separate training and validati
train_size = 0.8  # 80% of the 80% for training
data_train_final, data_val = train_test_split(data_train, test_size=(1 - train_size

# Print the sizes of the final splits
```

```
print(f"Final Training set size: {data_train_final.shape[0]}")
print(f"Validation set size: {data_val.shape[0]}")
print(f"Test set size: {data_test.shape[0]}")
```

```
Final Training set size: 42406
Validation set size: 10602
Test set size: 13252
```

In [12]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
import typing

def create_dataset(df: pd.DataFrame, valid_size: float, random_state: int) -> typin
    """
    Splits the DataFrame into training and validation sets, separating features fro

    Parameters:
    - df: The DataFrame to split.
    - valid_size: The proportion of the training data to include in the validation
    - random_state: The random seed for reproducibility.

    Returns:
    - X_train: Training features.
    - y_train: Training target values.
    - X_valid: Validation features.
    - y_valid: Validation target values.
    """
    # Assuming 'duration' is the target column
    target_column = 'duration'  # Update this based on your DataFrame's actual targ

    if target_column not in df.columns:
        raise KeyError(f"The target column '{target_column}' was not found in the D

    # Separate features and target
    X = df.drop(columns=[target_column])
    y = df[target_column]

    # Split into training and validation sets
    X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=valid_siz

    return X_train, y_train, X_valid, y_valid

# Example usage
valid_split = 0.2
random_state = 12345678

# Assuming data_train is your DataFrame, replace this with your actual DataFrame
# data_train = pd.read_csv('your_data.csv') # Example loading data

# Create the datasets
X_train, y_train, X_valid, y_valid = create_dataset(data_train, valid_size=valid_sp

# Tests
assert isinstance(X_train, pd.DataFrame)
assert isinstance(X_valid, pd.DataFrame)
assert isinstance(y_train, pd.Series)
```

```
assert isinstance(y_valid, pd.Series)
assert X_train.shape[0] <= data_train.shape[0] * (1 - valid_split + 0.05), "Number
assert X_valid.shape[0] <= data_train.shape[0] * (valid_split + 0.05), "Number of r
assert y_train.shape[0] == X_train.shape[0], "Number of rows should stay the same f
assert y_valid.shape[0] == X_valid.shape[0], "Number of rows should stay the same f
assert len(y_train.shape) == 1
assert len(y_valid.shape) == 1

# Print the sizes of the final splits
print(f"Training features size: {X_train.shape}")
print(f"Training target size: {y_train.shape}")
print(f"Validation features size: {X_valid.shape}")
print(f"Validation target size: {y_valid.shape}")
```

```
Training features size: (42406, 14)
Training target size: (42406,)
Validation features size: (10602, 14)
Validation target size: (10602,)
```

In [ ]:

```
# 7.2 First ML experiments
```

In [2]:

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split, GridSearchCV

# Example data setup
# Replace this with your actual data loading
# data_train = pd.read_csv('your_data.csv')

# For demonstration purposes, let's create a mock dataset
# This should be replaced with your actual data
np.random.seed(123)
X = np.random.rand(100, 5)  # 100 samples, 5 features
y = np.random.rand(100)     # 100 target values

# Split data into training and validation sets
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2, random_s

# Define the list of suitable ML methods with pipelines for scaling
suitable_ml_methods = [
    make_pipeline(StandardScaler(), LinearRegression()),
    make_pipeline(StandardScaler(), RandomForestRegressor()),
    make_pipeline(StandardScaler(), SVR())
]

def print_selection(selected: list, sel_type: str = 'methods'):
    print(f"Identified {sel_type}:\n==================")
    for pipeline in selected:
        model_name = pipeline.named_steps[list(pipeline.named_steps.keys())[-1]].__
        print(model_name)
```

```
print_selection(suitable_ml_methods)

# Perform hyperparameter tuning
param_grid_svr = {'svr__C': [0.1, 1, 10], 'svr__kernel': ['linear', 'rbf']}
param_grid_rf = {'randomforestregressor__n_estimators': [50, 100, 200], 'randomfore
param_grid_lr = {}

models_with_params = [
    (GridSearchCV(suitable_ml_methods[0], param_grid_lr, cv=5), "LinearRegression")
    (GridSearchCV(suitable_ml_methods[1], param_grid_rf, cv=5), "RandomForestRegres
    (GridSearchCV(suitable_ml_methods[2], param_grid_svr, cv=5), "SVR")
]

# Train each model and evaluate
for grid_search, model_name in models_with_params:
    grid_search.fit(X_train, y_train)
    best_score = grid_search.best_score_
    best_params = grid_search.best_params_
    print(f"{model_name} best validation score: {best_score:.4f} with params: {best
```

```
Identified methods:
===================
LinearRegression
RandomForestRegressor
SVR
LinearRegression best validation score: -0.0269 with params: {}
RandomForestRegressor best validation score: -0.0040 with params: {'randomforestregr
essor__max_depth': 10, 'randomforestregressor__n_estimators': 50}
SVR best validation score: 0.0193 with params: {'svr__C': 0.1, 'svr__kernel': 'rbf'}
```

In [ ]:

```
# Train a ML model
```

In [4]:

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.base import RegressorMixin

# Example data setup
# Replace this with your actual data loading
# data_train = pd.read_csv('your_data.csv')

# For demonstration purposes, let's create a mock dataset
np.random.seed(123)
X = np.random.rand(100, 5)  # 100 samples, 5 features
y = np.random.rand(100)     # 100 target values

# Split data into training and validation sets
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2, random_s

# Define the list of suitable ML methods with pipelines for scaling
suitable_ml_methods = [
```

```
    make_pipeline(StandardScaler(), LinearRegression()),
    make_pipeline(StandardScaler(), RandomForestRegressor()),
    make_pipeline(StandardScaler(), SVR())
]

def train_model(model_type: RegressorMixin, X_train: pd.DataFrame, y_train: pd.Data
    """
    Train a ML method on the train subset (X_train, y_train) and return the trained
    """
    # Train the model
    trained_model = model_type.fit(X_train, y_train)

    return trained_model

def predict_disruption_type(trained_model: RegressorMixin, X_valid: pd.DataFrame) -
    """
    Use the trained model to predict the validation subset (X_valid) and return the
    """
    # Make predictions
    y_pred = trained_model.predict(X_valid)

    return y_pred

# Choose a model index
model_idx = 0  # You can choose different models from the list of suitable models h
chosen_model_class = suitable_ml_methods[model_idx]
print(f"Chosen model: {chosen_model_class.named_steps[list(chosen_model_class.named

# Train the model
trained_model = train_model(chosen_model_class, X_train, y_train)

# Predict using the trained model
y_pred = predict_disruption_type(trained_model, X_valid)

# Assertions to ensure predictions are correct
assert y_pred.shape[0] == y_valid.shape[0], "Predictions for each row!"
assert len(y_pred.shape) == 1, 'Only one value per row!'
```

```
Chosen model: LinearRegression
```

In [ ]:

```
# 7.3 Explore different metrics
```

In [5]:

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

def print_selection(selection, name):
    print(f"Selected {name}:")
    for item in selection:
        print(f"- {item.__name__}")

# List of suitable metrics for regression
suitable_metrics = [
    mean_absolute_error,
    mean_squared_error,
    r2_score
]
```

```
# Display the selected metrics
print_selection(suitable_metrics, 'metrics')

# Tests
assert len(suitable_metrics) >= 3
assert np.all([cur_metric.__module__.startswith('sklearn') for cur_metric in suitab
    "Only use classes from sklearn!"
assert np.all([callable(cur_metric) for cur_metric in suitable_metrics]), \
    "Metrics must be functions"
```

```
Selected metrics:
- mean_absolute_error
- mean_squared_error
- r2_score
```

In [6]:

```
import pandas as pd
import numpy as np
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Assuming suitable_metrics is already defined
suitable_metrics = [
    mean_absolute_error,
    mean_squared_error,
    r2_score
]

def compare_metrics(y_true: pd.DataFrame, y_pred: pd.DataFrame) -> dict:
    """
    Calculate the values of different metrics for the given validation data.

    Parameters:
    - y_true: The true values (ground truth) for the validation set.
    - y_pred: The predicted values from the model for the validation set.

    Returns:
    - scores: A dictionary with metric names as keys and performance values as valu
    """
    scores = {}
    for metric in suitable_metrics:
        metric_name = metric.__name__
        metric_value = metric(y_true, y_pred)
        scores[metric_name] = metric_value

    return scores

def print_scores(scores: dict):
    """
    Print the scores in a formatted manner.

    Parameters:
    - scores: A dictionary with metric names as keys and performance values as valu
    """
    print("\nScores:\n=======")
    for metric_name, metric_value in scores.items():
        print(f"{metric_name}: {metric_value}")
```

```python
# Example usage
# Assuming y_valid and y_pred are defined from previous steps
metrics_scores = compare_metrics(y_valid, y_pred)
print_scores(metrics_scores)
```

```
Scores:
=======
mean_absolute_error: 0.2505822267007186
mean_squared_error: 0.08933707461931734
r2_score: -0.3635049153460541
```

In [ ]:
```python
# 7.4 Explore different scaling approaches
```

In [12]:
```python
import pandas as pd
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_absolute_error

def fit_pipeline(X_train: pd.DataFrame, y_train: pd.DataFrame, X_valid: pd.DataFram
    """
    Build a sklearn pipeline using the scaler and the model, train the pipeline,
    and predict on the valid data. Calculate the performance using the metric_func
    on the predictions and y_valid.

    Parameters:
    - X_train: Training features.
    - y_train: Training target.
    - X_valid: Validation features.
    - y_valid: Validation target.
    - model_class: The model class to be used.
    - scaler_class: The scaler class to be used.
    - metric_func: The metric function to evaluate performance.

    Returns:
    - score: The calculated performance score.
    """
    # Create a pipeline with the scaler and model
    pipeline = make_pipeline(scaler_class(), model_class())

    # Fit the pipeline on the training data
    pipeline.fit(X_train, y_train)

    # Predict on the validation data
    y_pred = pipeline.predict(X_valid)

    # Calculate the performance score
    score = metric_func(y_valid, y_pred)

    return score

def compare_scaling(X_train: pd.DataFrame, y_train: pd.DataFrame, X_valid: pd.DataF
    """
    Compare the performance of different scaling methods.
```

```
    Parameters:
    - X_train: Training features.
    - y_train: Training target.
    - X_valid: Validation features.
    - y_valid: Validation target.
    - model_class: The model class to be used.
    - metric_func: The metric function to evaluate performance.

    Returns:
    - scores: A dictionary with scaler names as keys and performance scores as valu
    """
    scores = {}
    scalers = [StandardScaler, MinMaxScaler, RobustScaler]

    for scaler in scalers:
        scaler_name = scaler.__name__
        score = fit_pipeline(X_train, y_train, X_valid, y_valid, model_class, scale
        scores[scaler_name] = score

    return scores

# Example usage
# Assume X_train, y_train, X_valid, y_valid are defined
# These should be your actual dataset splits
suitable_ml_methods = [KNeighborsRegressor]  # Example model list
suitable_metrics = [mean_absolute_error]  # Example metric list

model_idx = 0
metric_idx = 0

choosen_model_class = suitable_ml_methods[model_idx]
choosen_metric_func = suitable_metrics[metric_idx]

print(f"Chosen model: {choosen_model_class.__name__}")
print(f"Chosen metric: {choosen_metric_func.__name__}")

scaling_scores = compare_scaling(X_train, y_train, X_valid, y_valid, choosen_model_

def print_scores(scores):
    print("Scores:")
    print("=======")
    for scaler_name, score in scores.items():
        print(f"{scaler_name}: {score}")

print_scores(scaling_scores)
```

```
Chosen model: KNeighborsRegressor
Chosen metric: mean_absolute_error
Scores:
=======
StandardScaler: 0.2456661089364137
MinMaxScaler: 0.24426409267974494
RobustScaler: 0.23744478682497103
```

In [ ]:

```
# 7.5 Experiment with different train/valid splits
```

In [1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_absolute_error
import numpy as np
import typing

# Load your data into a DataFrame using the correct file path
try:
    data_train = pd.read_pickle('/home/e12319879/shared/188.995-2024W/data/data_pro
    print("Data loaded successfully.")
except FileNotFoundError:
    print("File not found. Please check the file path.")
    data_train = None

# Function to preprocess data
def preprocess_data(df: pd.DataFrame):
    # Convert any datetime columns to numerical values
    for column in df.select_dtypes(include=['datetime64']).columns:
        df[column] = df[column].astype('int64')  # Convert to integer representatio

    # Handle timedelta columns if any are known
    for column in df.columns:
        if pd.api.types.is_timedelta64_dtype(df[column]):
            df[column] = df[column].dt.total_seconds()  # Convert to total seconds

    return df

# Function to create dataset by separating features and target
def create_dataset(df: pd.DataFrame, target_column: str):
    X = df.drop(columns=[target_column])
    y = df[target_column]
    return X, y

# Function to fit and evaluate the model
def fit_and_evaluate(X_train: pd.DataFrame, y_train: pd.Series, X_valid: pd.DataFra
    # Identify categorical columns
    categorical_cols = X_train.select_dtypes(include=['object']).columns

    # Create a preprocessing pipeline
    preprocessor = ColumnTransformer(
        transformers=[
            ('num', StandardScaler(), X_train.select_dtypes(include=['int64', 'floa
            ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_cols)
        ])

    # Create a pipeline with preprocessing and KNeighborsRegressor
    model = Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('regressor', KNeighborsRegressor())
    ])
```

```
    print("Fitting the model...")
    # Fit the model
    model.fit(X_train, y_train)

    print("Predicting...")
    # Predict and evaluate
    y_pred = model.predict(X_valid)
    score = mean_absolute_error(y_valid, y_pred)
    print(f"Mean Absolute Error: {score}")
    return score

# Function to compare train-validation splits
def compare_train_valid_splits(df: pd.DataFrame) -> typing.Dict[str, float]:
    scores = {}
    target_column = 'duration'  # Assuming 'duration' is the target column

    df = preprocess_data(df)  # Preprocess the data
    X, y = create_dataset(df, target_column)

    # Different train-validation splits
    splits = {
        "65-35": (0.65, 0.35),
        "70-30": (0.7, 0.3),
        "75-25": (0.75, 0.25),
        "80-20": (0.8, 0.2)
    }

    for split_name, (train_size, valid_size) in splits.items():
        print(f"Processing split: {split_name}")
        X_train, X_valid, y_train, y_valid = train_test_split(X, y, train_size=trai
        score = fit_and_evaluate(X_train, y_train, X_valid, y_valid)
        scores[split_name] = score
        print(f"Score for {split_name}: {score}")

    return scores

# Ensure the data is loaded before proceeding
if data_train is not None:
    split_scores = compare_train_valid_splits(data_train)

    def print_scores(scores):
        print("Scores for different train-validation splits:")
        print("============================================")
        for split_name, score in scores.items():
            print(f"{split_name}: {score}")

    print_scores(split_scores)
else:
    print("Data not available. Cannot perform analysis.")
```

```
Data loaded successfully.
Processing split: 65-35
Fitting the model...
Predicting...
Mean Absolute Error: 8238.8848777543
Score for 65-35: 8238.8848777543
Processing split: 70-30
Fitting the model...
Predicting...
Mean Absolute Error: 8258.0054933446
Score for 70-30: 8258.0054933446
Processing split: 75-25
Fitting the model...
Predicting...
Mean Absolute Error: 8458.52896682865
Score for 75-25: 8458.52896682865
Processing split: 80-20
Fitting the model...
Predicting...
Mean Absolute Error: 8366.861821255017
Score for 80-20: 8366.861821255017
Scores for different train-validation splits:
===========================================
65-35: 8238.8848777543
70-30: 8258.0054933446
75-25: 8458.52896682865
80-20: 8366.861821255017
```

In [ ]:

```
# 7.6 Experiment with different feature selection methods
```

In [5]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest, f_regression, RFE
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import typing

# Assuming data_train is your DataFrame and 'duration' is your target column
target_column = 'duration'  # Replace with your actual target column name

# Function to preprocess data
def preprocess_data(df: pd.DataFrame):
    # Convert any datetime columns to numerical values
    for column in df.select_dtypes(include=['datetime64']).columns:
        df[column] = df[column].astype('int64')  # Convert to integer representatio

    # Handle timedelta columns if any are known
    for column in df.columns:
        if pd.api.types.is_timedelta64_dtype(df[column]):
            df[column] = df[column].dt.total_seconds()  # Convert to total seconds

    return df
```

```
# Preprocess data
data_train = preprocess_data(data_train)

# Function to create dataset by separating features and target
def create_dataset(df: pd.DataFrame, target_column: str):
    X = df.drop(columns=[target_column])
    y = df[target_column]
    return X, y

# Create features and target
X, y = create_dataset(data_train, target_column)

# Original train-validation split from Section 7.2
X_train, X_valid, y_train, y_valid = train_test_split(X, y, train_size=0.7, test_si

def compare_feature_selection(X_train: pd.DataFrame, X_valid: pd.DataFrame, y_train
    scores = {}

    # Identify categorical columns
    categorical_cols = X_train.select_dtypes(include=['object', 'category']).column

    # Preprocessing pipeline for numerical and categorical data
    preprocessor = ColumnTransformer(
        transformers=[
            ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_cols)
        ],
        remainder='passthrough'
    )

    # Baseline model setup
    def evaluate_with_feature_selection(X_train_sel, X_valid_sel):
        model = KNeighborsRegressor()
        model.fit(X_train_sel, y_train)
        y_pred = model.predict(X_valid_sel)
        return mean_absolute_error(y_valid, y_pred)

    # Feature Selection Method 1: SelectKBest
    print("Applying SelectKBest...")
    k_best = SelectKBest(score_func=f_regression, k=10)  # Select top 10 features
    X_train_k_best = k_best.fit_transform(preprocessor.fit_transform(X_train), y_tr
    X_valid_k_best = k_best.transform(preprocessor.transform(X_valid))
    score_k_best = evaluate_with_feature_selection(X_train_k_best, X_valid_k_best)
    scores['SelectKBest'] = score_k_best
    print(f"SelectKBest Score: {score_k_best}")

    # Feature Selection Method 2: Recursive Feature Elimination (RFE)
    print("Applying RFE...")
    rfe = RFE(estimator=LinearRegression(), n_features_to_select=10)  # Select top
    X_train_rfe = rfe.fit_transform(preprocessor.fit_transform(X_train), y_train)
    X_valid_rfe = rfe.transform(preprocessor.transform(X_valid))
    score_rfe = evaluate_with_feature_selection(X_train_rfe, X_valid_rfe)
    scores['RFE'] = score_rfe
    print(f"RFE Score: {score_rfe}")

    return scores
```

```
# Evaluate feature selection methods
feat_sel_scores = compare_feature_selection(X_train, X_valid, y_train, y_valid)

# Function to print scores
def print_scores(scores):
    print("Scores for different feature selection methods:")
    for method, score in scores.items():
        print(f"{method}: {score}")

print_scores(feat_sel_scores)
```

```
Applying SelectKBest...
SelectKBest Score: 225088.96091274032
Applying RFE...
RFE Score: 18301.286710331715
Scores for different feature selection methods:
SelectKBest: 225088.96091274032
RFE: 18301.286710331715
```

In [ ]:
```
# 7.7 Try out different ML algorithms
```

In [ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.metrics import mean_absolute_error
import typing

# Load your data from a pickle file
data_train = pd.read_pickle('/home/e12319879/shared/188.995-2024W/data/data_process

# Define the target column
target_column = 'duration'  # Ensure this matches exactly with your DataFrame's col

# Preprocess data
def preprocess_data(df: pd.DataFrame):
    # Convert any datetime columns to numerical values
    for column in df.select_dtypes(include=['datetime64']).columns:
        df[column] = df[column].astype('int64')  # Convert to integer representatio

    # Handle timedelta columns if any are known
    for column in df.columns:
        if pd.api.types.is_timedelta64_dtype(df[column]):
            df[column] = df[column].dt.total_seconds()  # Convert to total seconds

    return df

# Preprocess data
data_train = preprocess_data(data_train)

# Function to create dataset by separating features and target
```

```
def create_dataset(df: pd.DataFrame, target_column: str):
    X = df.drop(columns=[target_column])
    y = df[target_column]
    return X, y

# Create features and target
X, y = create_dataset(data_train, target_column)

# Original train-validation split
X_train, X_valid, y_train, y_valid = train_test_split(X, y, train_size=0.7, test_si

def compare_methods(X_train: pd.DataFrame, X_valid: pd.DataFrame, y_train: pd.Serie
    scores = {}

    # Identify categorical columns
    categorical_cols = X_train.select_dtypes(include=['object', 'category']).column

    # Preprocessing pipeline for numerical and categorical data
    preprocessor = ColumnTransformer(
        transformers=[
            ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_cols)
        ],
        remainder='passthrough'
    )

    # Define models to evaluate
    models = {
        'LinearRegression': LinearRegression(),
        'KNeighborsRegressor': KNeighborsRegressor(n_neighbors=5),
        'RandomForestRegressor': RandomForestRegressor(n_estimators=10, random_stat
    }

    # Evaluate each model
    for name, model in models.items():
        print(f"Evaluating {name}...")
        pipeline = Pipeline(steps=[('preprocessor', preprocessor), ('model', model)

        # Perform cross-validation with reduced folds
        cv_scores = cross_val_score(pipeline, X_train, y_train, cv=10, scoring='neg
        mean_cv_score = -cv_scores.mean()
        scores[name + ' CV'] = mean_cv_score
        print(f"{name} Cross-Validation MAE: {mean_cv_score}")

        # Fit model and evaluate on validation set
        pipeline.fit(X_train, y_train)
        y_pred = pipeline.predict(X_valid)
        valid_score = mean_absolute_error(y_valid, y_pred)
        scores[name + ' Validation'] = valid_score
        print(f"{name} Validation MAE: {valid_score}")

    return scores

# Evaluate different methods
diff_methods_scores = compare_methods(X_train, X_valid, y_train, y_valid)

# Function to print scores
```

```python
def print_scores(scores):
    print("Scores for different methods:")
    for method, score in scores.items():
        print(f"{method}: {score}")

print_scores(diff_methods_scores)
```

```
Evaluating LinearRegression...
LinearRegression Cross-Validation MAE: 8558.837259488466
LinearRegression Validation MAE: 8485.318085578869
Evaluating KNeighborsRegressor...
KNeighborsRegressor Cross-Validation MAE: 8972.488482144097
```

In [ ]:
```python
# Explore the effect of parameters with 10-fold cross validation
```

In [ ]:
```python
import pandas as pd
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import make_scorer, mean_absolute_error
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import typing

# Load your data from a pickle file
data_train = pd.read_pickle('/home/e12319879/shared/188.995-2024W/data/data_process

# Assuming 'duration' is your target column
target_column = 'duration'  # Replace with your actual target column name

# Function to preprocess data
def preprocess_data(df: pd.DataFrame):
    # Convert any datetime columns to numerical values
    for column in df.select_dtypes(include=['datetime64']).columns:
        df[column] = df[column].astype('int64')  # Convert to integer representatio

    # Handle timedelta columns if any are known
    for column in df.columns:
        if pd.api.types.is_timedelta64_dtype(df[column]):
            df[column] = df[column].dt.total_seconds()  # Convert to total seconds

    return df

# Preprocess data
data_train = preprocess_data(data_train)

# Function to create dataset by separating features and target
def create_dataset(df: pd.DataFrame, target_column: str):
    X = df.drop(columns=[target_column])
    y = df[target_column]
    return X, y

# Create features and target
X, y = create_dataset(data_train, target_column)

def compare_param_effect(X: pd.DataFrame, y: pd.Series) -> typing.Dict[str, float]:
```

```
    scores = {}

    # Identify categorical columns
    categorical_cols = X.select_dtypes(include=['object', 'category']).columns

    # Preprocessing pipeline for numerical and categorical data
    preprocessor = ColumnTransformer(
        transformers=[
            ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_cols)
        ],
        remainder='passthrough'
    )

    # Define parameter grid
    param_grid = {
        'n_estimators': [50, 100, 200],
        'max_depth': [None, 10, 20],
        'min_samples_split': [2, 5, 10]
    }

    # Baseline model setup
    for param_name, param_values in param_grid.items():
        for value in param_values:
            try:
                print(f"Evaluating RandomForestRegressor with {param_name}={value}.
                model_params = {param_name: value}
                model = RandomForestRegressor(random_state=42, **model_params)
                pipeline = Pipeline(steps=[('preprocessor', preprocessor), ('model'

                # Perform cross-validation
                cv_scores = cross_val_score(
                    pipeline, X, y, cv=10, scoring=make_scorer(mean_absolute_error,
                )
                mean_cv_score = -cv_scores.mean()  # Negate because greater_is_bett
                scores[f'RandomForest {param_name}={value}'] = mean_cv_score
                print(f"Mean CV MAE for {param_name}={value}: {mean_cv_score}")
            except Exception as e:
                print(f"Error evaluating {param_name}={value}: {e}")

    return scores

# Evaluate parameter effects
param_effect_scores = compare_param_effect(X, y)

# Function to print scores
def print_scores(scores):
    print("Scores for different parameter settings:")
    for method, score in scores.items():
        print(f"{method}: {score}")

print_scores(param_effect_scores)
```

```
Evaluating RandomForestRegressor with n_estimators=50...
```

In [ ]:

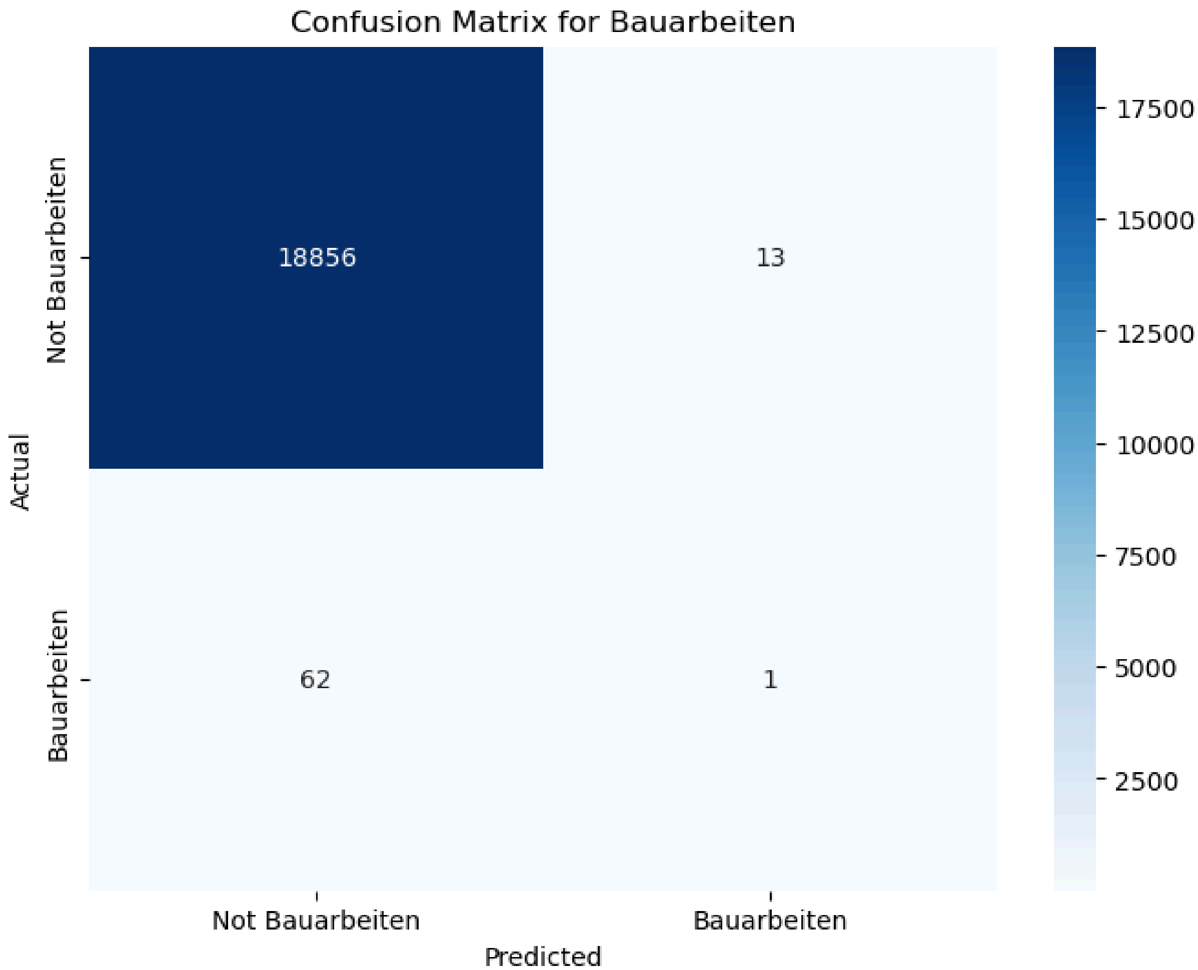
```
# 7.9 Present your best-performing training results
```

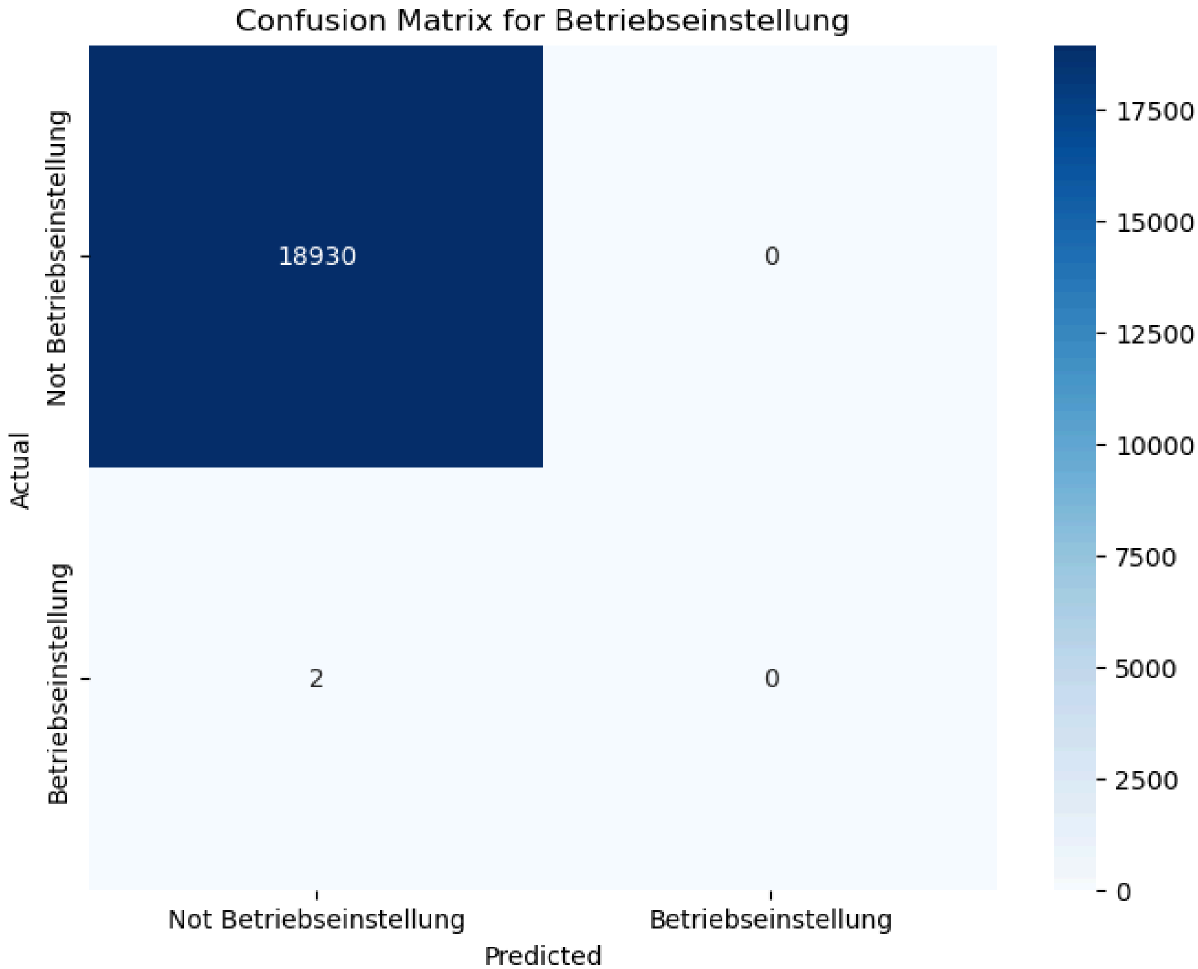
In [5]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, multilabel_confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Load your data from the pickle file
data_train = pd.read_pickle('/home/e12319879/shared/188.995-2024W/data/data_process

# Print the column names to identify the correct target column
print("Columns in the DataFrame:", data_train.columns)

# Update this variable based on the actual column name for the target
target_column = 'disruption'  # Assuming 'disruption' is the target column

# Preprocess data
def preprocess_data(df: pd.DataFrame):
    # Convert any datetime columns to numerical values
    for column in df.select_dtypes(include=['datetime64']).columns:
        df[column] = df[column].astype('int64')  # Convert to integer representatio

    # Handle timedelta columns if any are known
    for column in df.columns:
        if pd.api.types.is_timedelta64_dtype(df[column]):
            df[column] = df[column].dt.total_seconds()  # Convert to total seconds

    return df

# Preprocess data
data_train = preprocess_data(data_train)

# Define features and target
features = ['temp_dailyMean', 'hum_dailyMin', 'hum_dailyMax', 'hum_dailyMean', 'bus
X = data_train[features]
y = data_train[target_column]

# Encode target labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

def extract_val_data(X, y, valid_split, random_state):
    return train_test_split(X, y, test_size=valid_split, random_state=random_state)

# Extract train and validation data
X_train, X_valid, y_train, y_valid = extract_val_data(X, y_encoded, valid_split=0.2

def fit(scaler, model, X_train, y_train, X_valid):
    # Scale the features
    X_train_scaled = scaler.fit_transform(X_train)
    X_valid_scaled = scaler.transform(X_valid)

    # Fit the model
    model.fit(X_train_scaled, y_train)
```

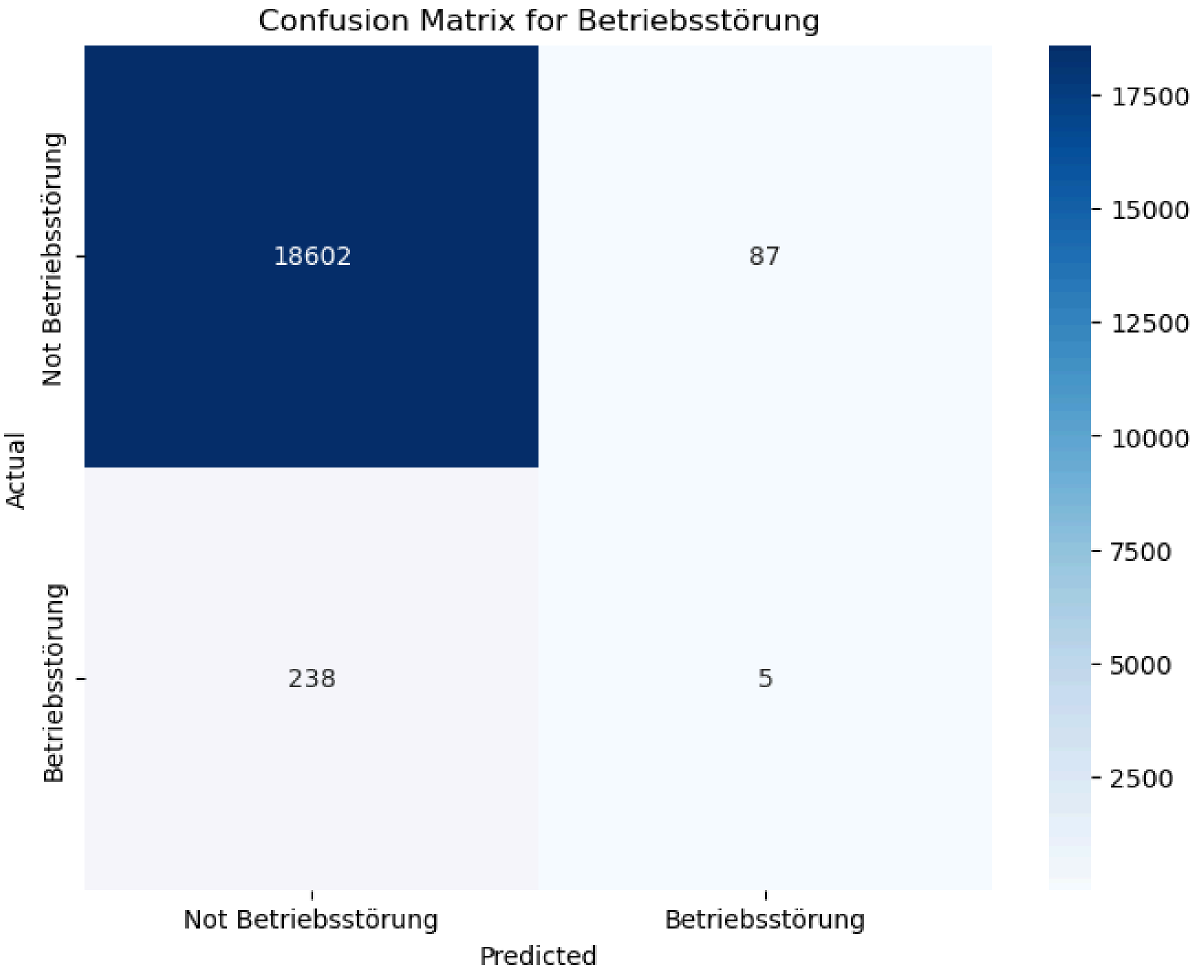
```
    # Predict on validation data
    y_pred = model.predict(X_valid_scaled)

    return y_pred

# Best configuration
model = RandomForestClassifier(random_state=42, criterion="gini")
scaler = StandardScaler()
y_pred = fit(scaler, model, X_train, y_train, X_valid)

# Decode the predicted and true labels
y_true = y_valid
y_pred_decoded = label_encoder.inverse_transform(y_pred)
y_true_decoded = label_encoder.inverse_transform(y_true)

# Generate classification report with zero_division set to 0
report = classification_report(y_true_decoded, y_pred_decoded, zero_division=0)
print("Classification Report:\n", report)

# Generate multilabel confusion matrix
cm = multilabel_confusion_matrix(y_true, y_pred)

# Plot confusion matrix
def plot_confusion_matrix(cm, class_name):
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not ' + class_
    plt.title(f'Confusion Matrix for {class_name}')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

# Plot confusion matrices for each class
labels = label_encoder.classes_
num_matrices = len(cm)

for i in range(num_matrices):
    class_name = labels[i]
    plot_confusion_matrix(cm[i], class_name)
```

```
Columns in the DataFrame: Index(['temp_dailyMin', 'temp_dailyMax', 'temp_dailyMean',
'temp_dailyMedian',
       'hum_dailyMin', 'hum_dailyMax', 'hum_dailyMean', 'wind_dailyMin',
       'wind_dailyMax', 'wind_dailyMean', 'precip', 'disruption', 'bus',
       'subway', 'tram', 'duration'],
      dtype='object')
Classification Report:
                                          precision    recall  f1-score   support

                              Bauarbeiten       0.07      0.02      0.03        63
                     Betriebseinstellung       0.00      0.00      0.00         2
                         Betriebsstörung       0.05      0.02      0.03       243
                           Demonstration       0.10      0.05      0.07       206
              Erhöhtes Fahrgastaufkommen       0.00      0.00      0.00         0
                              Erkrankung       0.04      0.02      0.02        61
             Erkrankung eines Fahrgastes       0.09      0.01      0.02        84
                   Fahrleitungsgebrechen       0.17      0.01      0.02        90
                        Fahrtbehinderung       0.24      0.23      0.24       521
                            Falschparker       0.16      0.12      0.14      1781
                        Feuerwehreinsatz       0.08      0.03      0.05       491
                  Fremder Verkehrsunfall       0.09      0.04      0.06       611
                        Gasrohrgebrechen       0.00      0.00      0.00        13
                        Gleisbauarbeiten       0.00      0.00      0.00        12
                            Gleisschaden       0.00      0.00      0.00       106
                          Polizeieinsatz       0.10      0.06      0.08      1509
                         Regenbogenparade       0.00      0.00      0.00         1
                         Rettungseinsatz       0.17      0.17      0.17      2448
                        Sachbeschädigung       0.00      0.00      0.00        28
                         Schadhafter Zug       0.00      0.00      0.00         1
                     Schadhaftes Fahrzeug       0.17      0.17      0.17      2170
         Schadhaftes Fahrzeug Verspätungen       0.00      0.00      0.00         1
                           Signalstörung       0.08      0.03      0.05        62
                             Staatsbesuch       0.00      0.00      0.00         5
                        Stellwerkstörung       0.09      0.02      0.04        43
                            Stromstörung       0.00      0.00      0.00       121
                             Sturmschaden       0.00      0.00      0.00        10
                            Veranstaltung       0.17      0.14      0.15       131
                         Verkehrsbedingt       0.54      0.70      0.61        20
             Verkehrsbedingt Verspätungen       0.43      0.69      0.53        48
Verkehrsbedingte Verspätung Verspätungen       0.00      0.00      0.00         1
            Verkehrsbedingte Verspätungen       0.39      0.58      0.47      1079
                         Verkehrsstörung       0.10      0.03      0.05       236
             Verkehrsstörung Verspätungen       0.00      0.00      0.00         1
                          Verkehrsunfall       0.18      0.17      0.18      2433
              Verkehrsunfall Verspätungen       0.00      0.00      0.00         3
                              Verspätung       0.62      0.85      0.72      1592
                             Verspätungen       0.40      0.71      0.51      2465
                          Verunreinigung       0.00      0.00      0.00        62
                     Vienna-City-Marathon       0.00      0.00      0.00         1
                          Wagengebrechen       0.00      0.00      0.00        10
                     Wasserrohrgebrechen       0.33      0.06      0.10        18
                          Weichenstörung       0.00      0.00      0.00        93
                        Witterungsbedingt       0.56      0.31      0.40        29
              erhöhtes Fahrgastaufkommen       0.00      0.00      0.00        25
 erhöhtes Fahrgastaufkommen Verspätungen       0.00      0.00      0.00         2
```

```
    accuracy                           0.29     18932
   macro avg       0.12      0.11      0.11     18932
weighted avg       0.23      0.29      0.25     18932
```

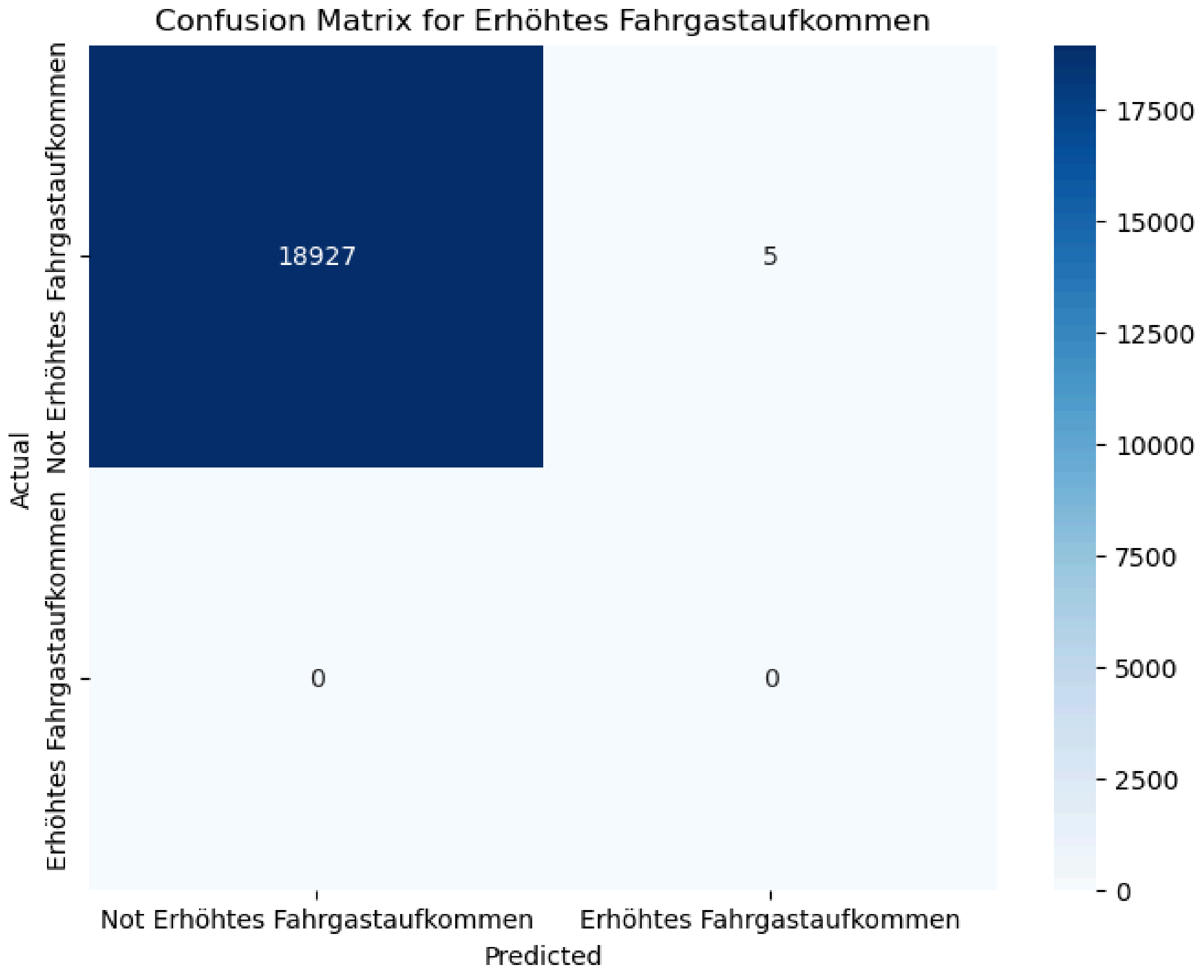


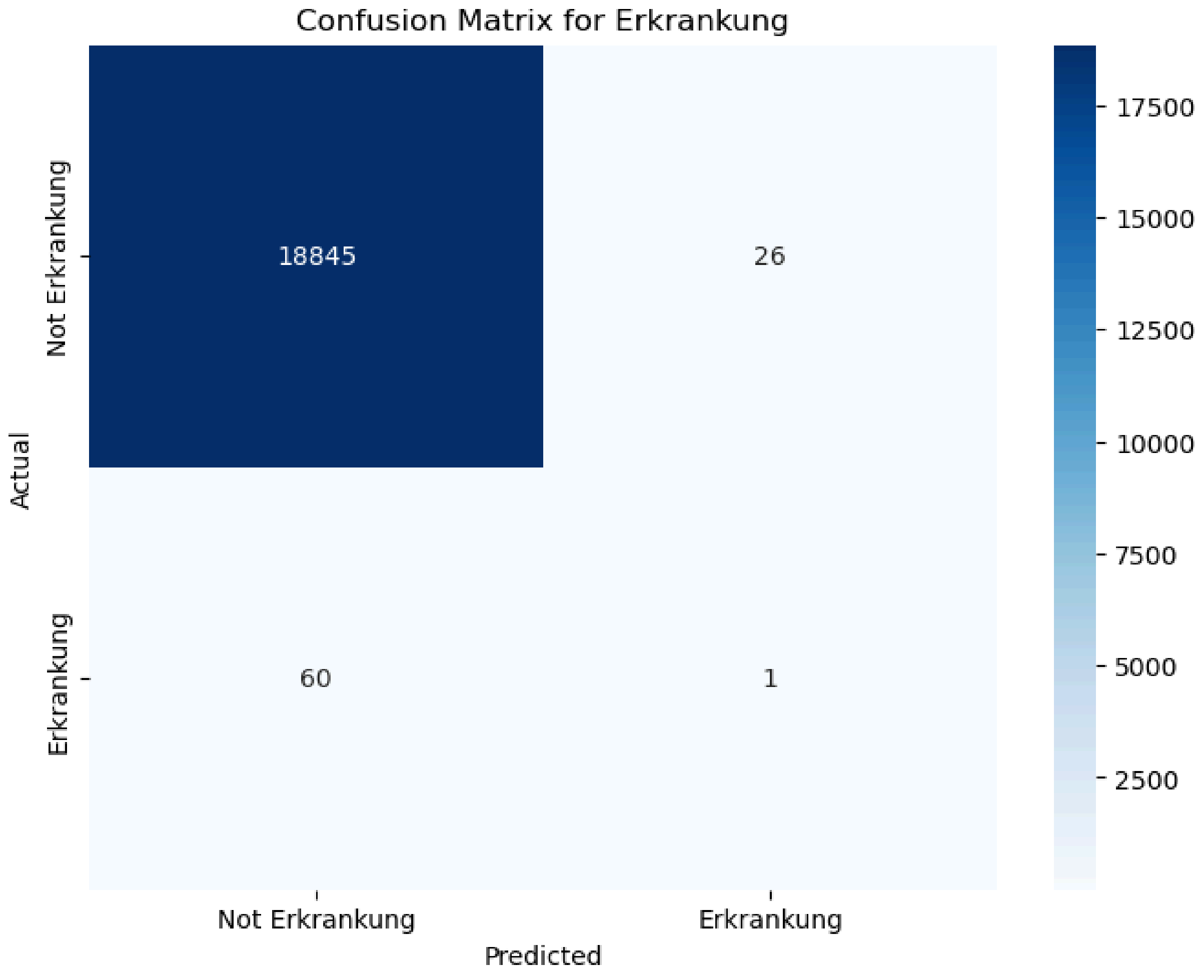

Confusion Matrix for Betriebseinstellung
Actual
Not Betriebseinstellung
Betriebseinstellung
18930
0
2
0
Predicted
Not Betriebseinstellung
Betriebseinstellung
17500
15000
12500
10000
7500
5000
2500
0

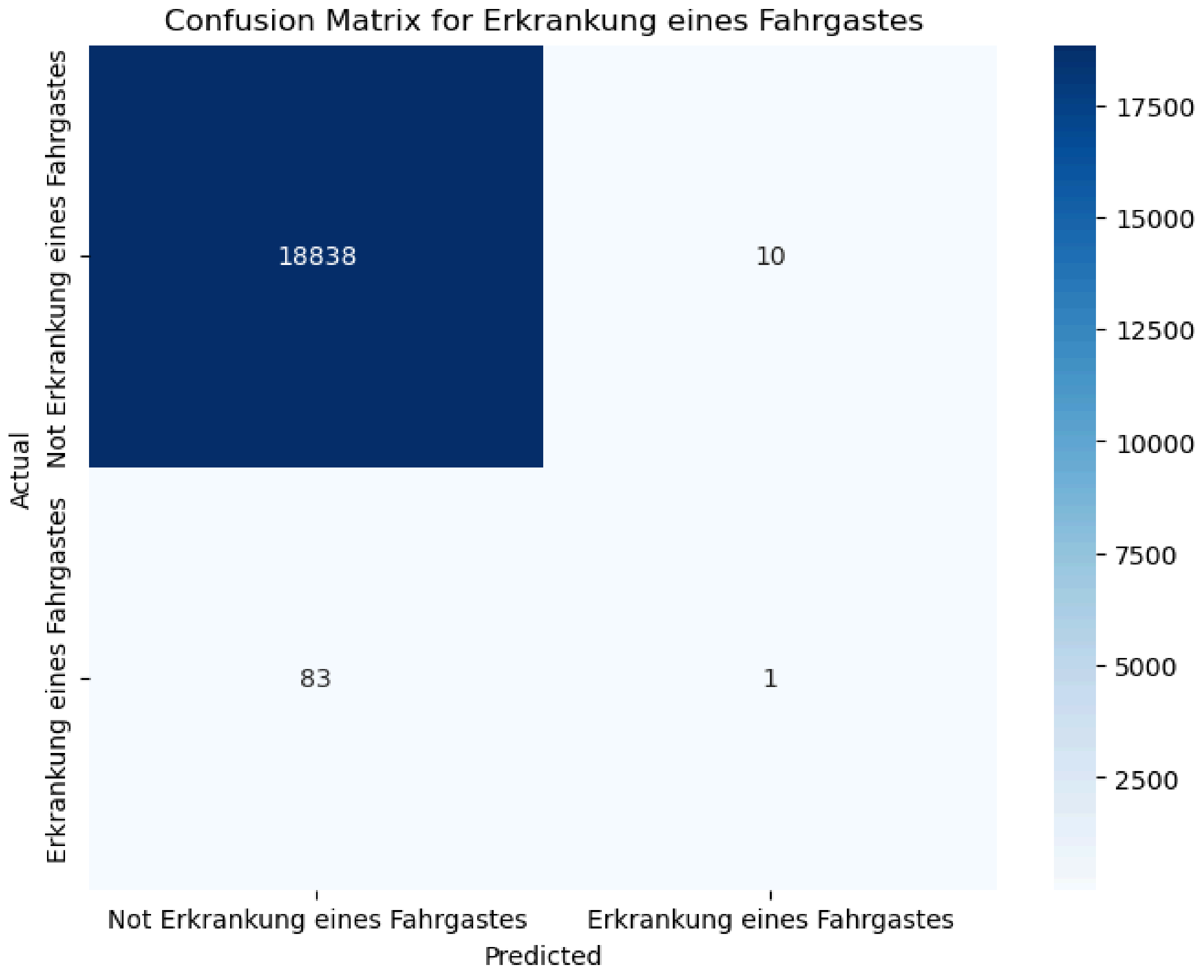


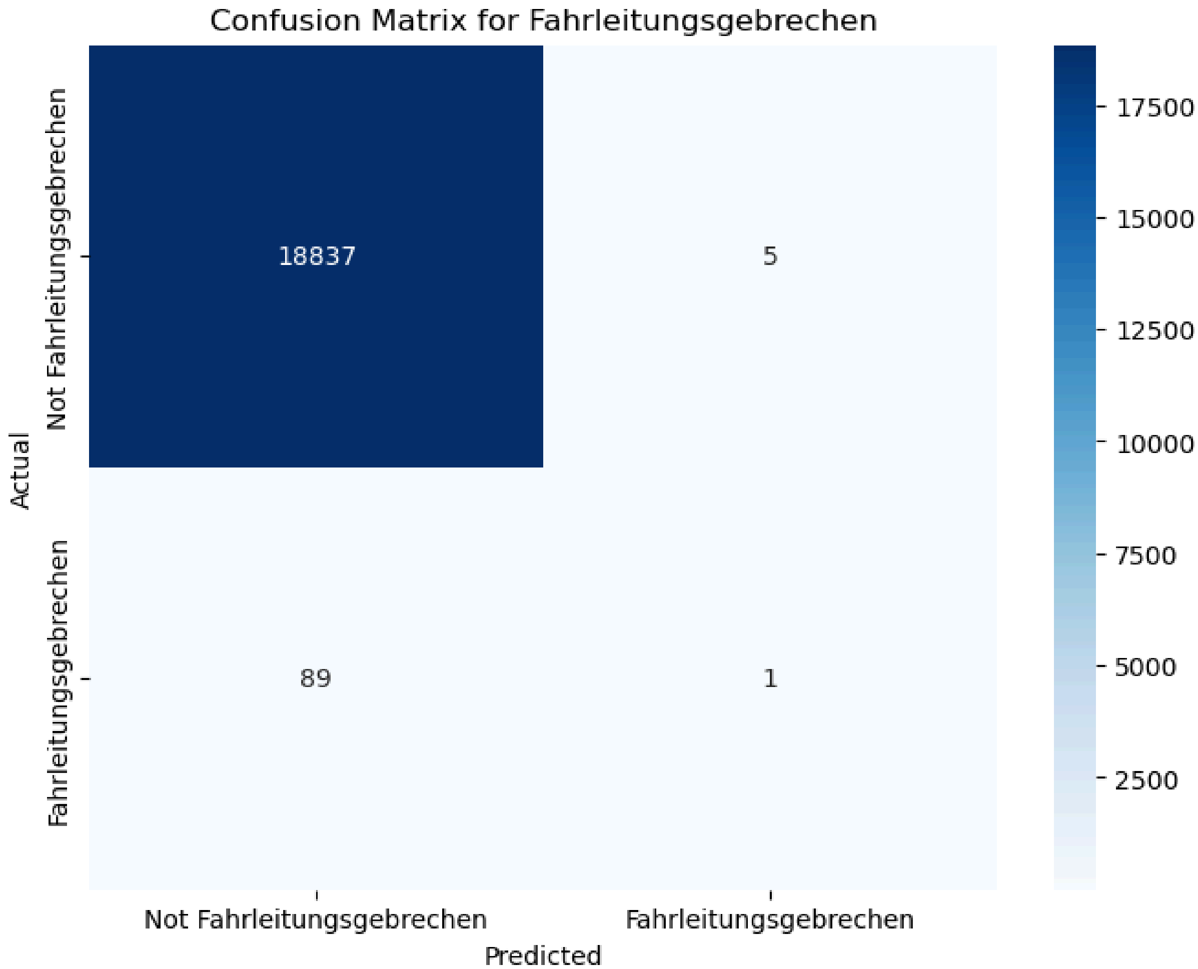
Confusion Matrix for Betriebsstörung
Actual
Not Betriebsstörung
Betriebsstörung
18602
87
238
5
Predicted
Not Betriebsstörung
Betriebsstörung
17500
15000
12500
10000
7500
5000
2500

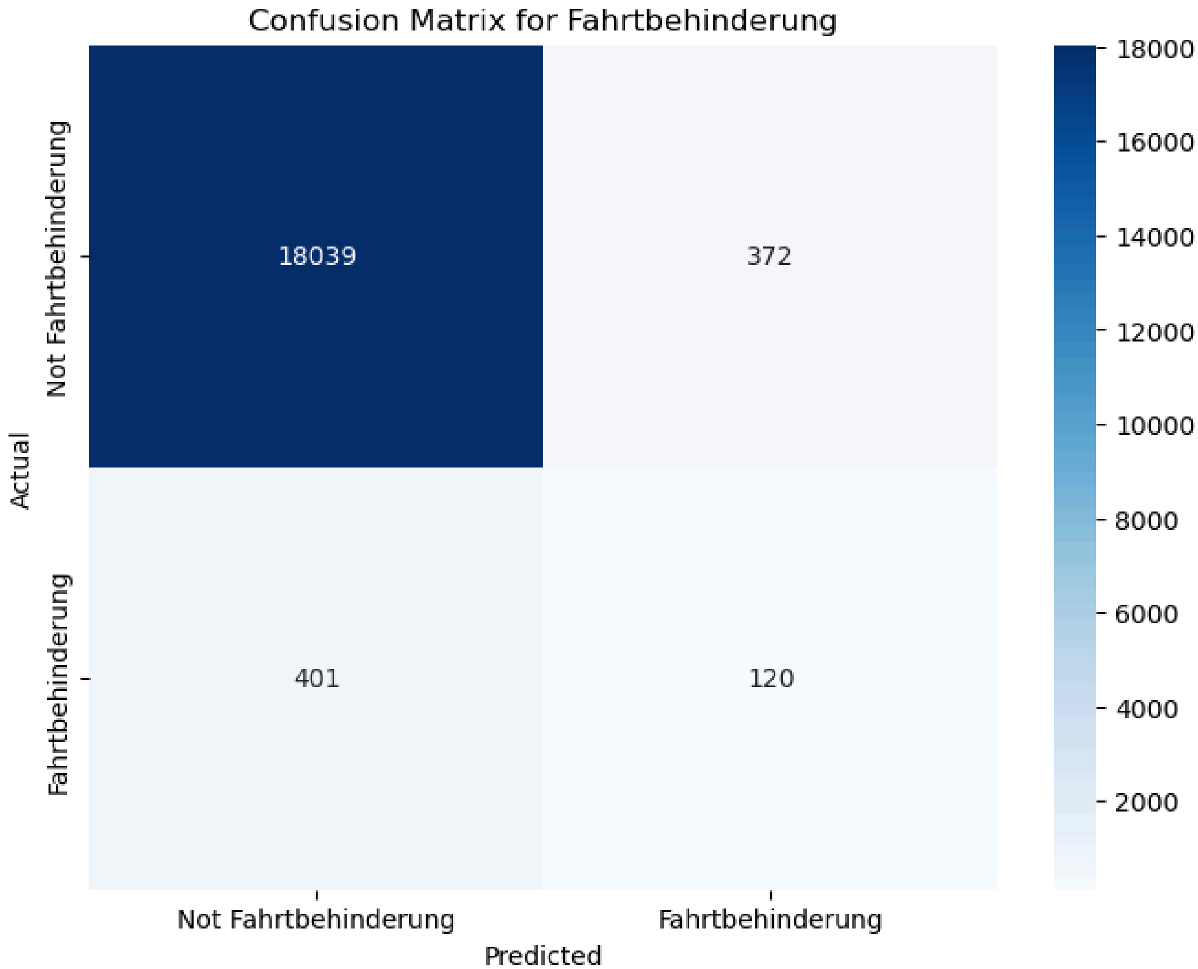

Confusion Matrix for Demonstration
Actual
Not Demonstration
Demonstration
Predicted
Not Demonstration
Demonstration
18636
90
196
10
17500
15000
12500
10000
7500
5000
2500

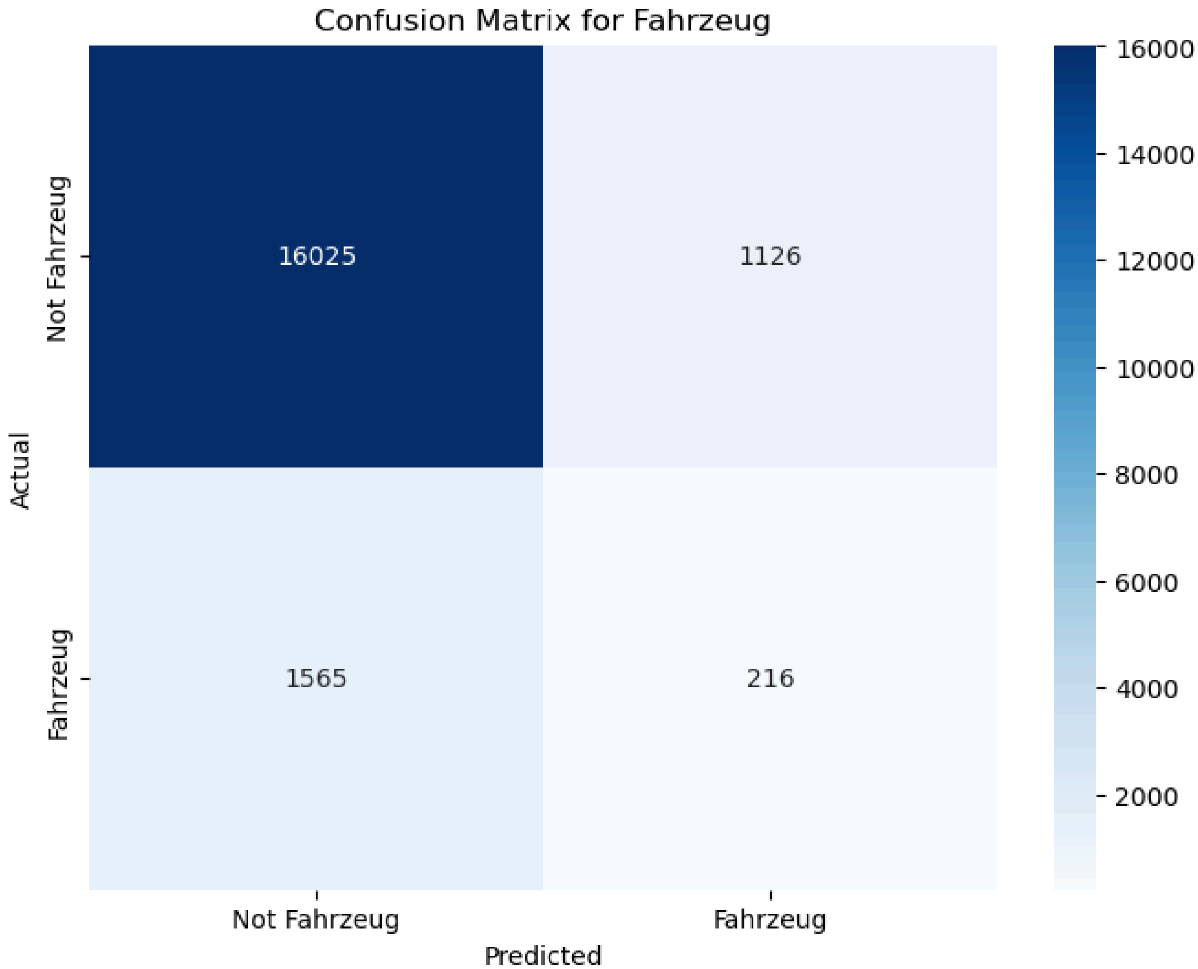

Confusion Matrix for Erhöhtes Fahrgastaufkommen
Actual
Not Erhöhtes Fahrgastaufkommen
Erhöhtes Fahrgastaufkommen
18927
5
0
0
Not Erhöhtes Fahrgastaufkommen
Erhöhtes Fahrgastaufkommen
Predicted
17500
15000
12500
10000
7500
5000
2500
0

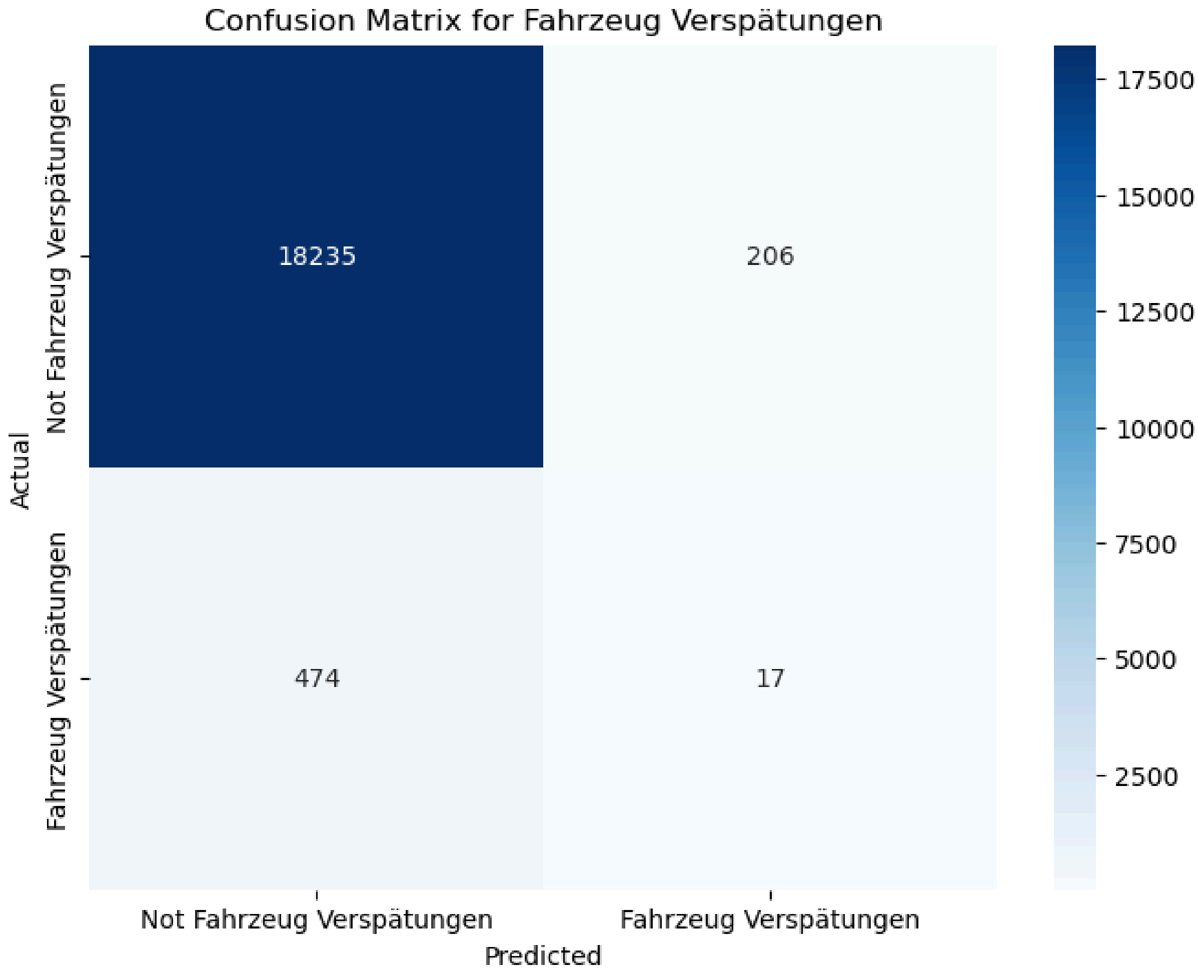

Confusion Matrix for Erkrankung
Actual
Not Erkrankung
Erkrankung
18845
26
60
1
Predicted
Not Erkrankung
Erkrankung
17500
15000
12500
10000
7500
5000
2500

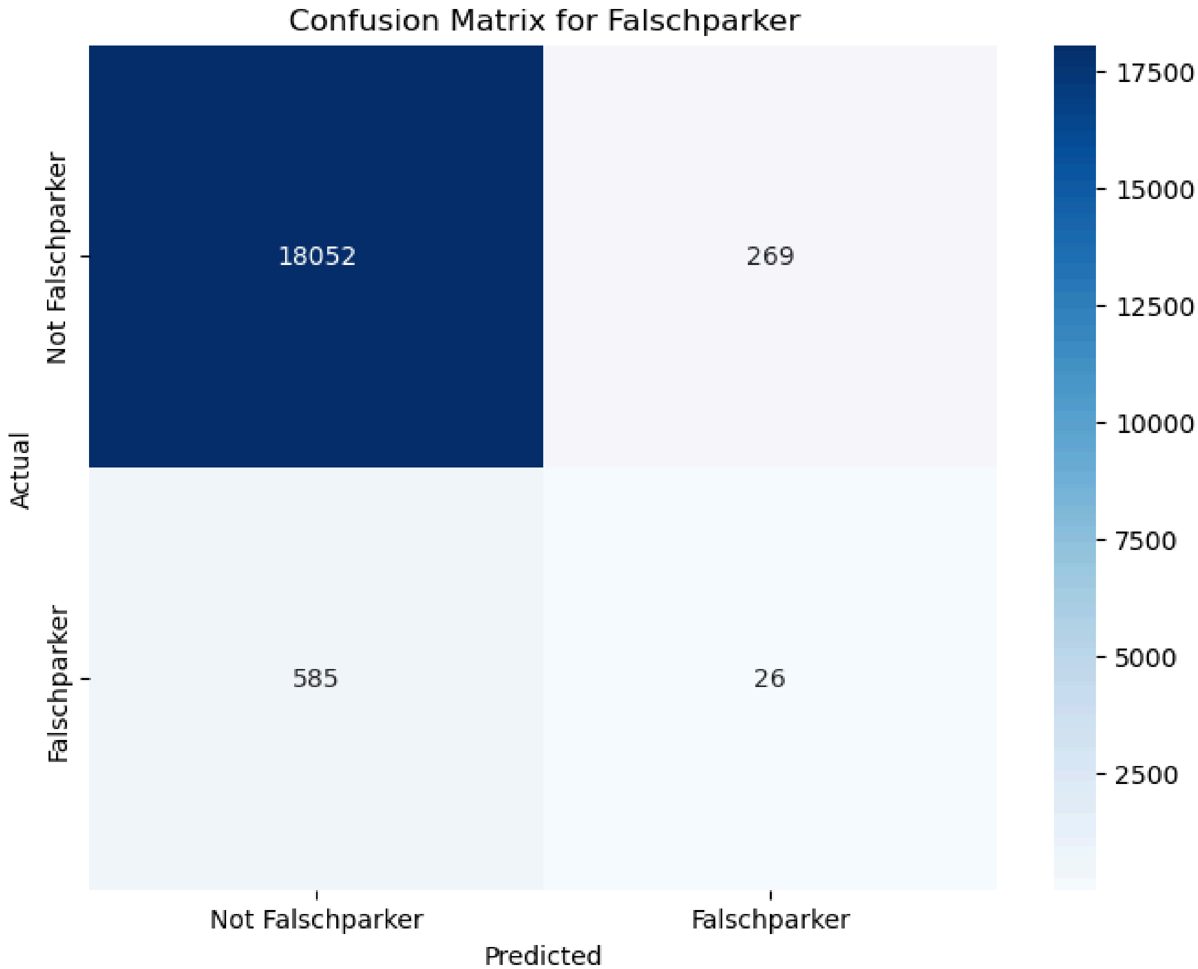

Confusion Matrix for Erkrankung eines Fahrgastes
Actual
Not Erkrankung eines Fahrgastes
Erkrankung eines Fahrgastes
18838
10
83
1
Predicted
Not Erkrankung eines Fahrgastes
Erkrankung eines Fahrgastes
17500
15000
12500
10000
7500
5000
2500

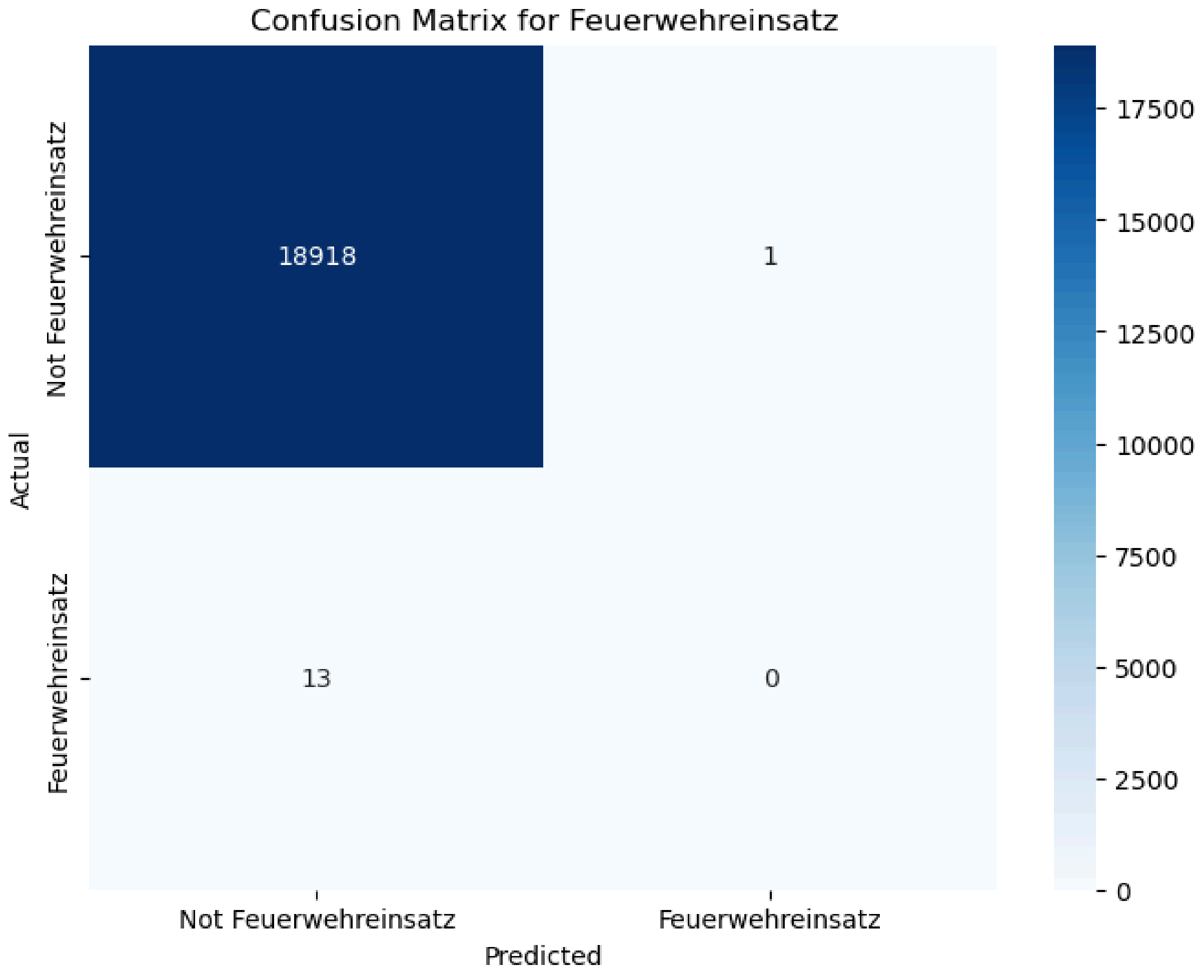

Confusion Matrix for Fahrleitungsgebrechen
Actual
Not Fahrleitungsgebrechen
Fahrleitungsgebrechen
18837
5
89
1
Not Fahrleitungsgebrechen
Fahrleitungsgebrechen
Predicted
17500
15000
12500
10000
7500
5000
2500


Confusion Matrix for Fahrtbehinderung
Actual
Not Fahrtbehinderung
Fahrtbehinderung
18039
372
401
120
Predicted
Not Fahrtbehinderung
Fahrtbehinderung
18000
16000
14000
12000
10000
8000
6000
4000
2000

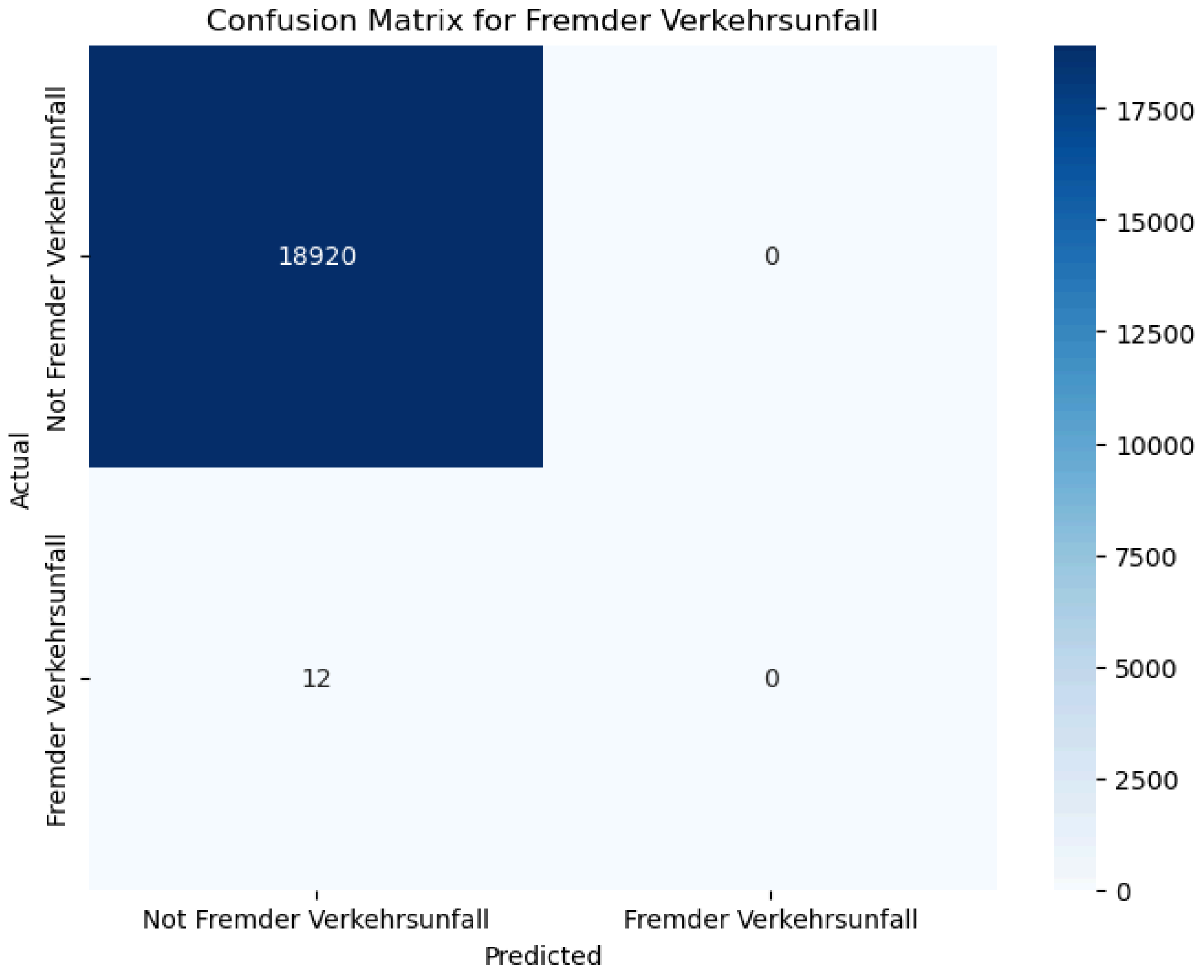

Confusion Matrix for Fahrzeug
Actual
Not Fahrzeug
Fahrzeug
16025
1126
1565
216
Not Fahrzeug
Fahrzeug
Predicted
16000
14000
12000
10000
8000
6000
4000
2000

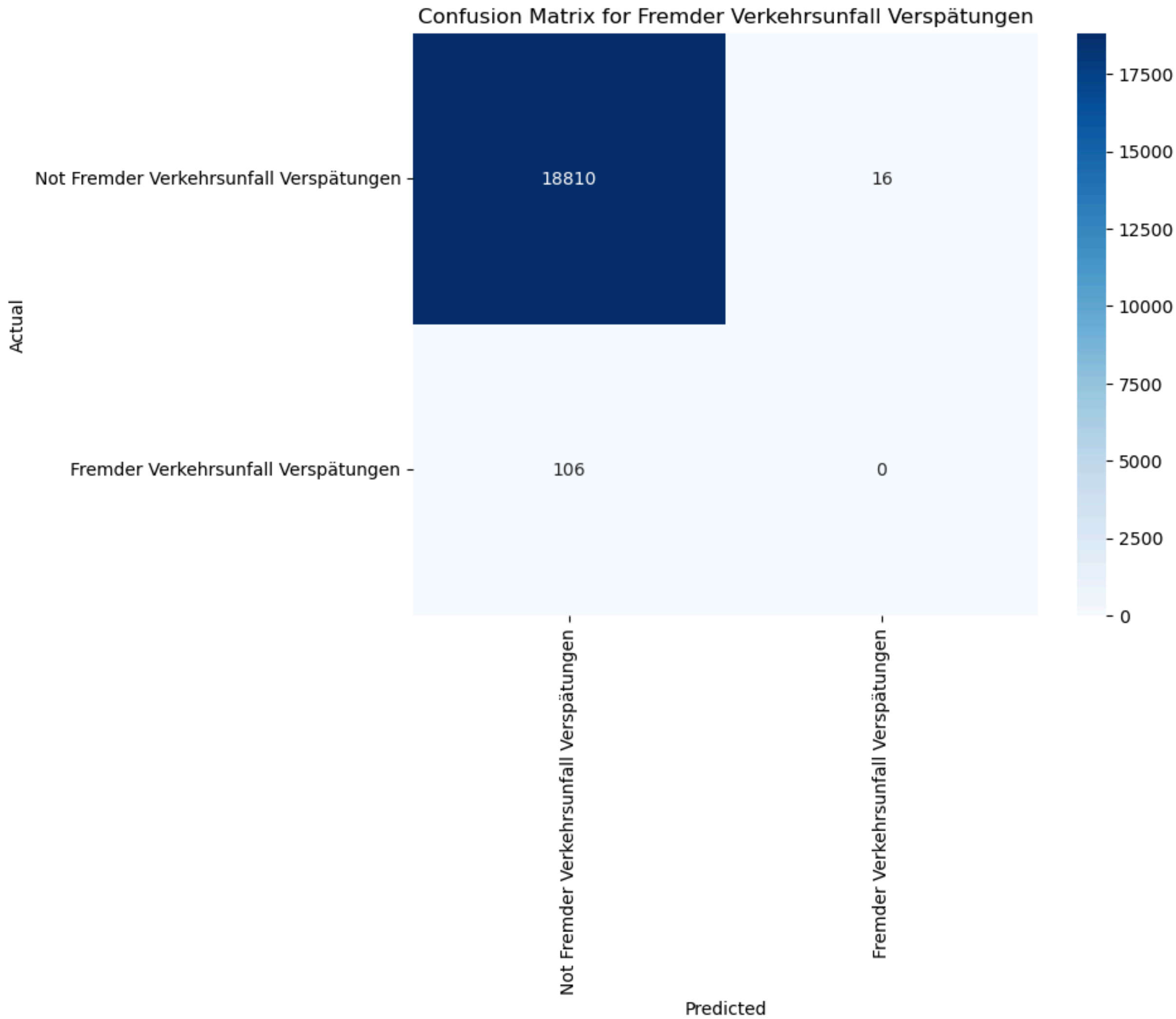

Confusion Matrix for Fahrzeug Verspätungen
Actual
Not Fahrzeug Verspätungen
Fahrzeug Verspätungen
18235
206
474
17
Predicted
Not Fahrzeug Verspätungen
Fahrzeug Verspätungen
17500
15000
12500
10000
7500
5000
2500

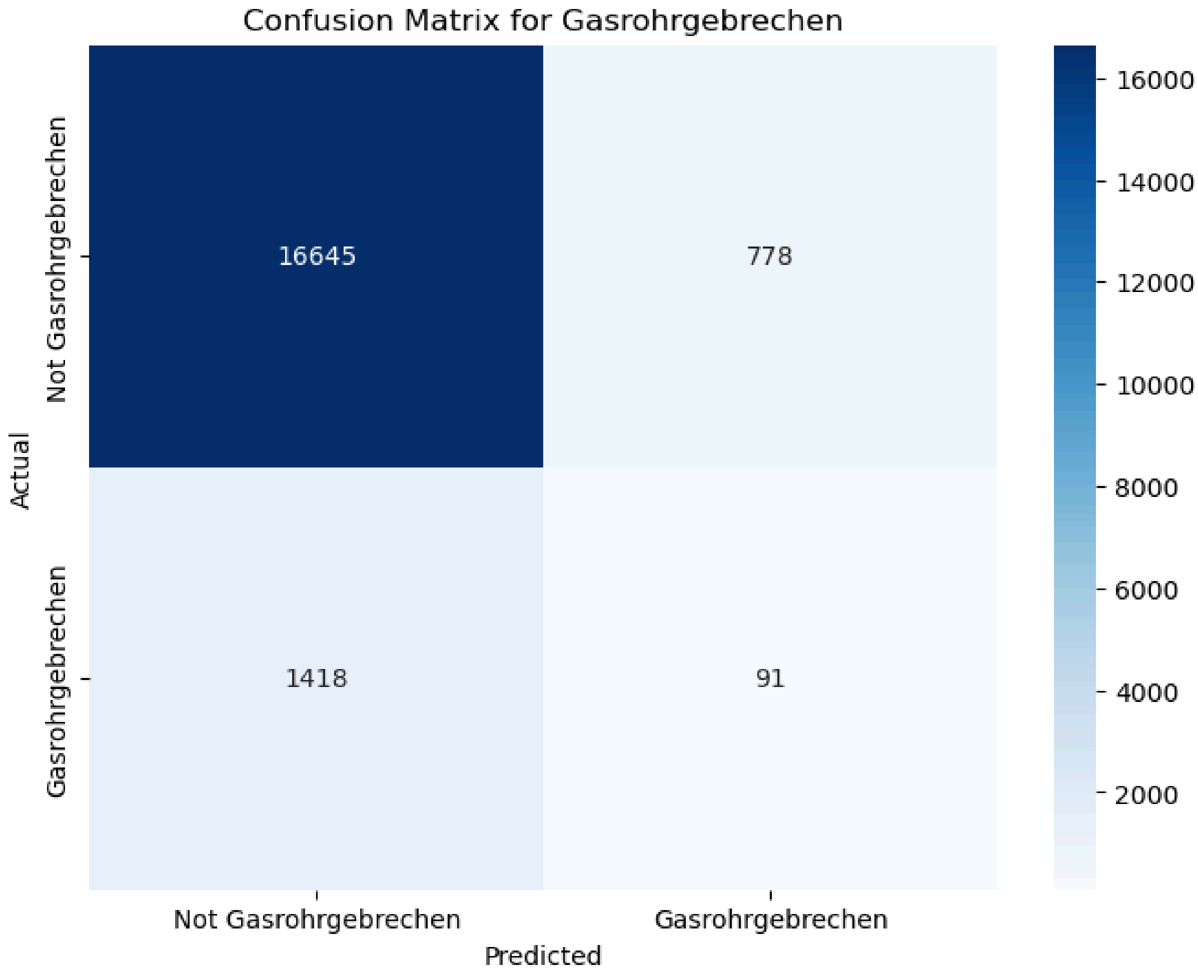

Confusion Matrix for Falschparker
Actual
Not Falschparker
Falschparker
18052
269
585
26
Predicted
Not Falschparker
Falschparker
17500
15000
12500
10000
7500
5000
2500



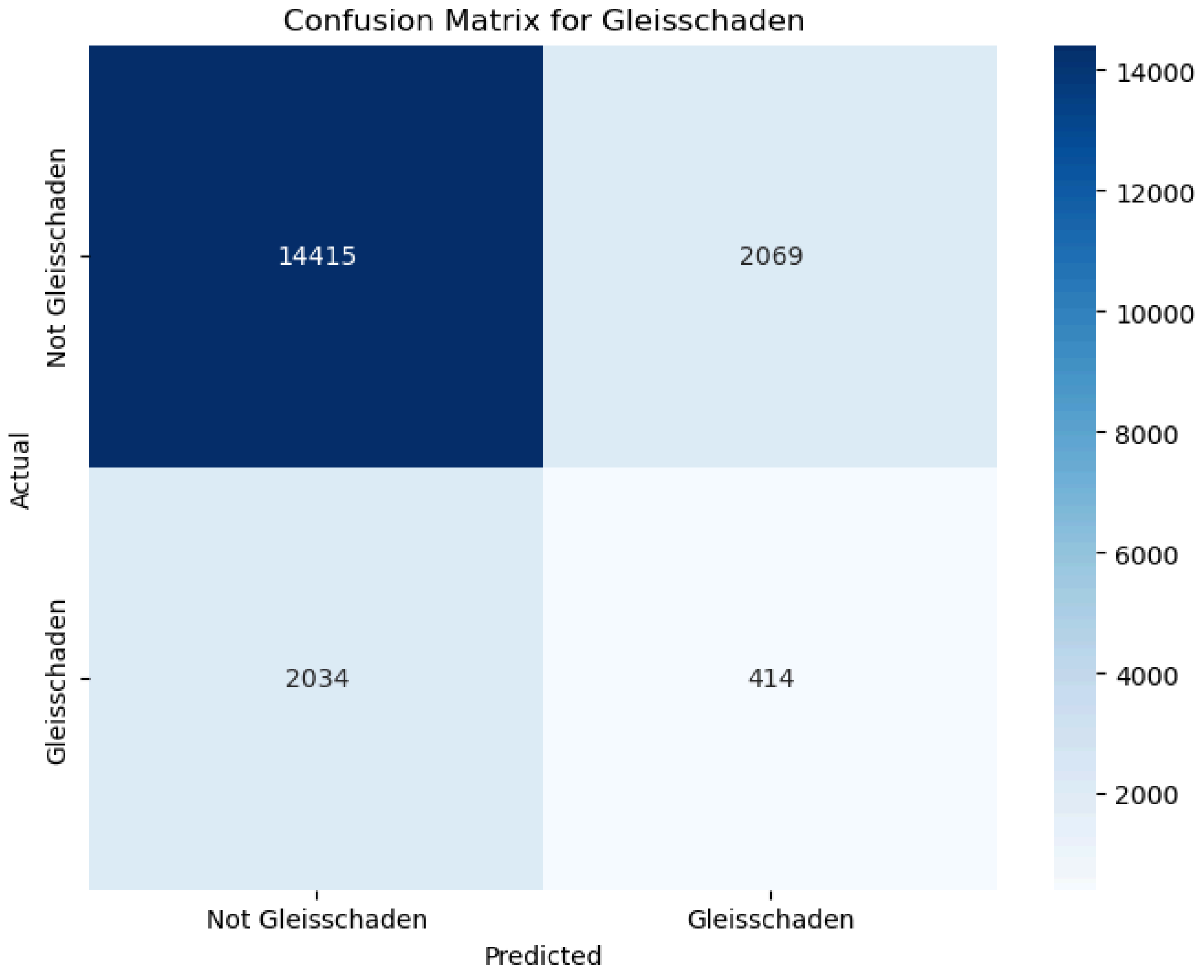
Confusion Matrix for Feuerwehreinsatz
Actual
Not Feuerwehreinsatz
Feuerwehreinsatz
Predicted
Not Feuerwehreinsatz
Feuerwehreinsatz
18918
1
13
0
17500
15000
12500
10000
7500
5000
2500
0

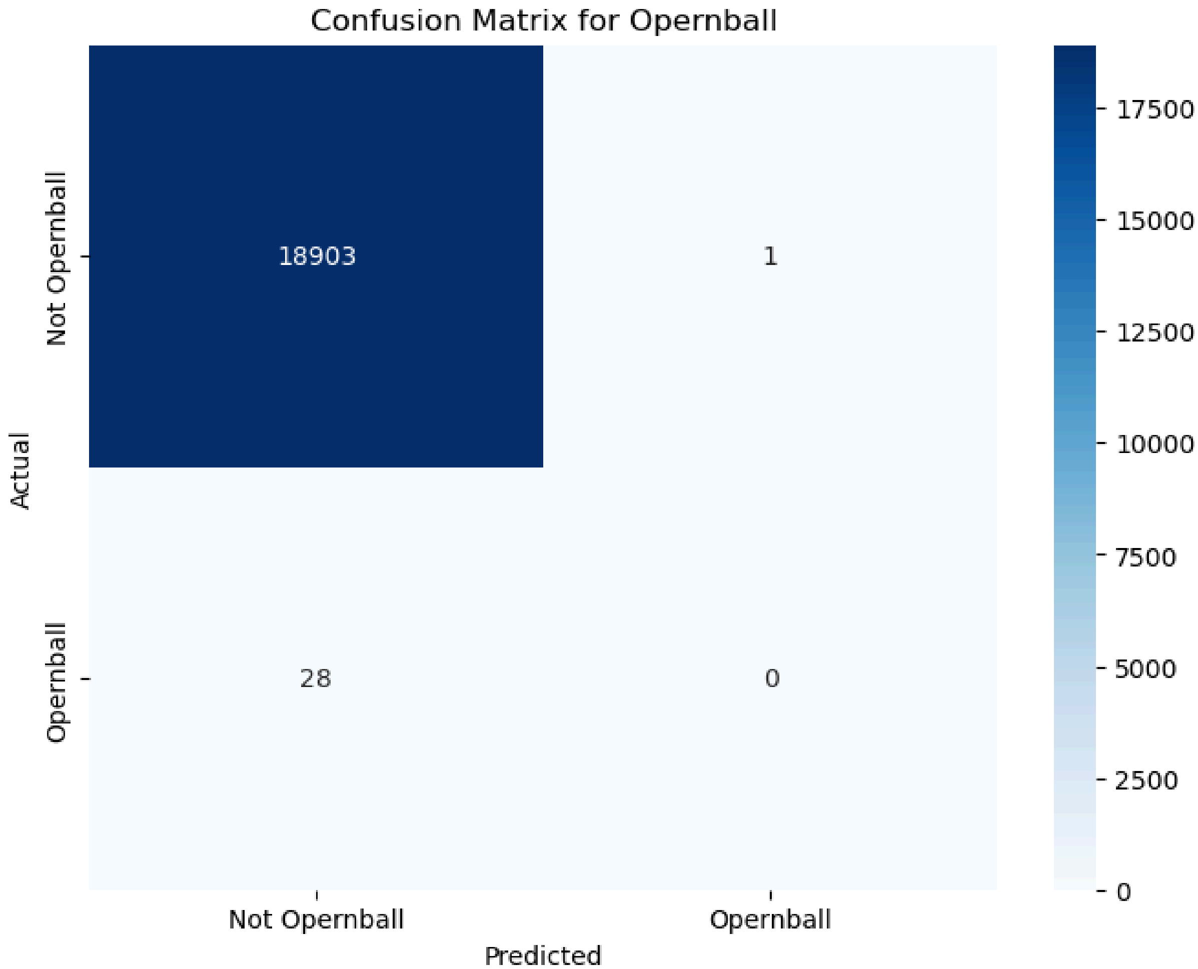

Confusion Matrix for Fremder Verkehrsunfall
Actual
Not Fremder Verkehrsunfall
Fremder Verkehrsunfall
18920
0
12
0
Predicted
Not Fremder Verkehrsunfall
Fremder Verkehrsunfall
17500
15000
12500
10000
7500
5000
2500
0

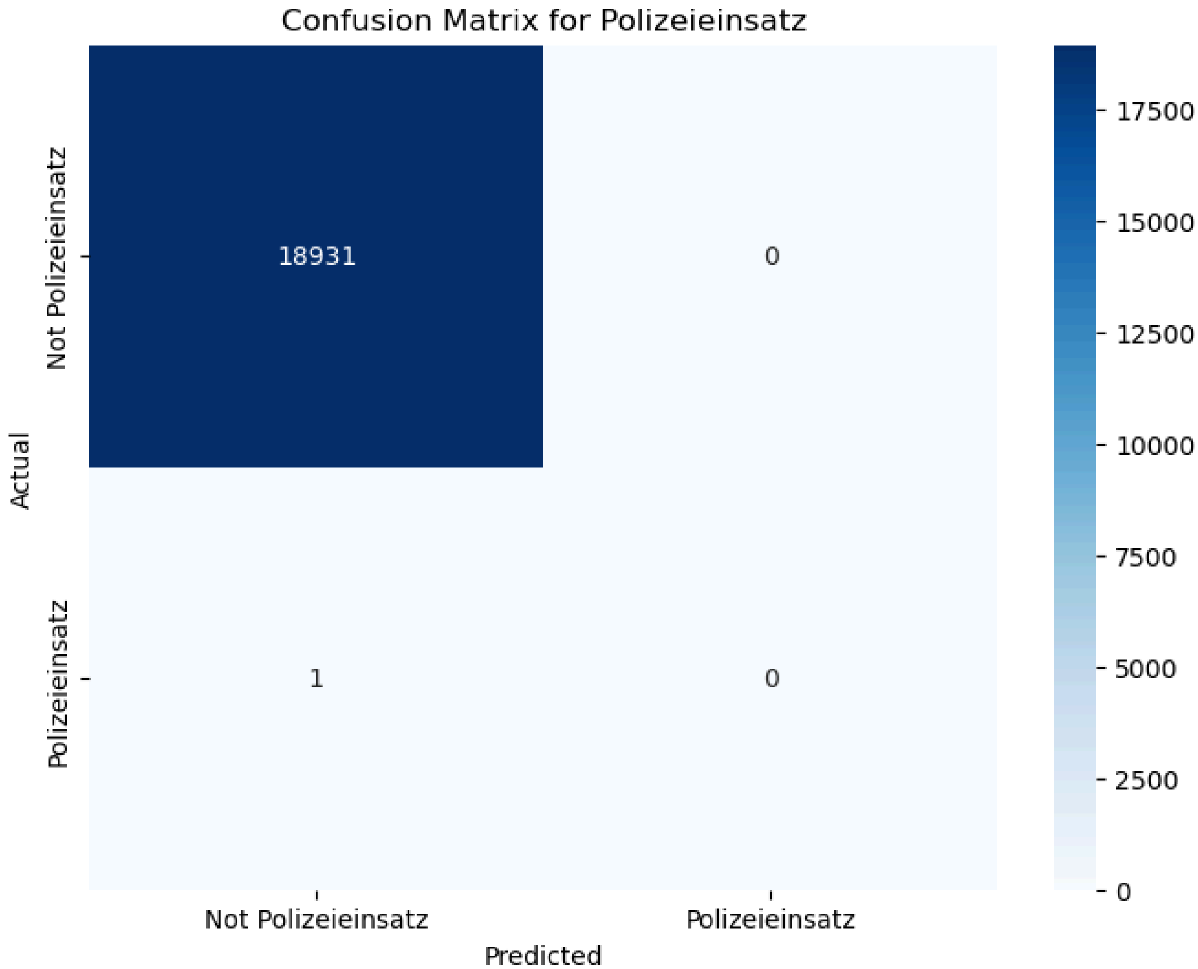

Confusion Matrix for Fremder Verkehrsunfall Verspätungen
Not Fremder Verkehrsunfall Verspätungen
Fremder Verkehrsunfall Verspätungen
18810
16
106
0
Actual
Predicted
Not Fremder Verkehrsunfall Verspätungen
Fremder Verkehrsunfall Verspätungen
17500
15000
12500
10000
7500
5000
2500
0

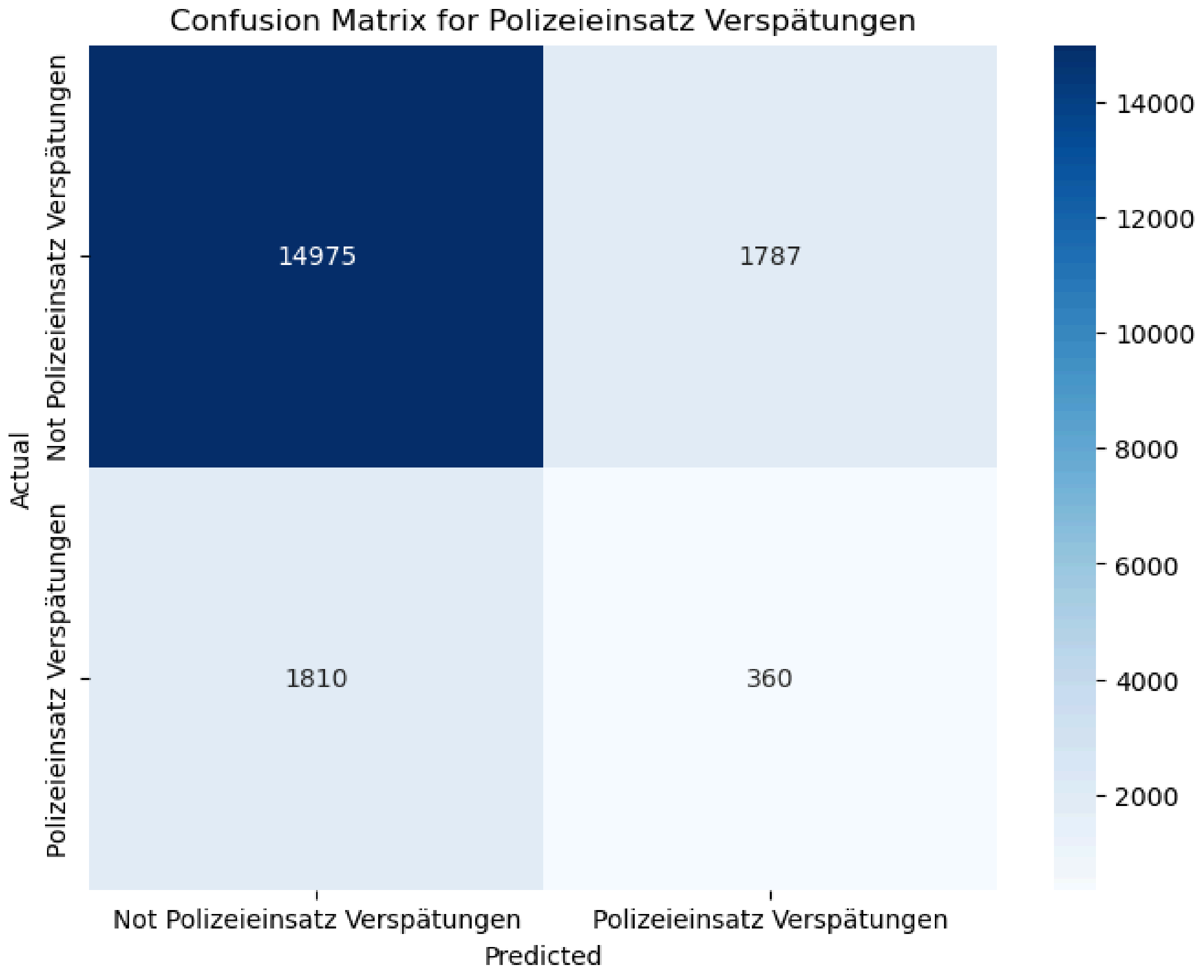

Confusion Matrix for Gasrohrgebrechen
Not Gasrohrgebrechen
Gasrohrgebrechen
Actual
16645
778
1418
91
Not Gasrohrgebrechen
Gasrohrgebrechen
Predicted
16000
14000
12000
10000
8000
6000
4000
2000

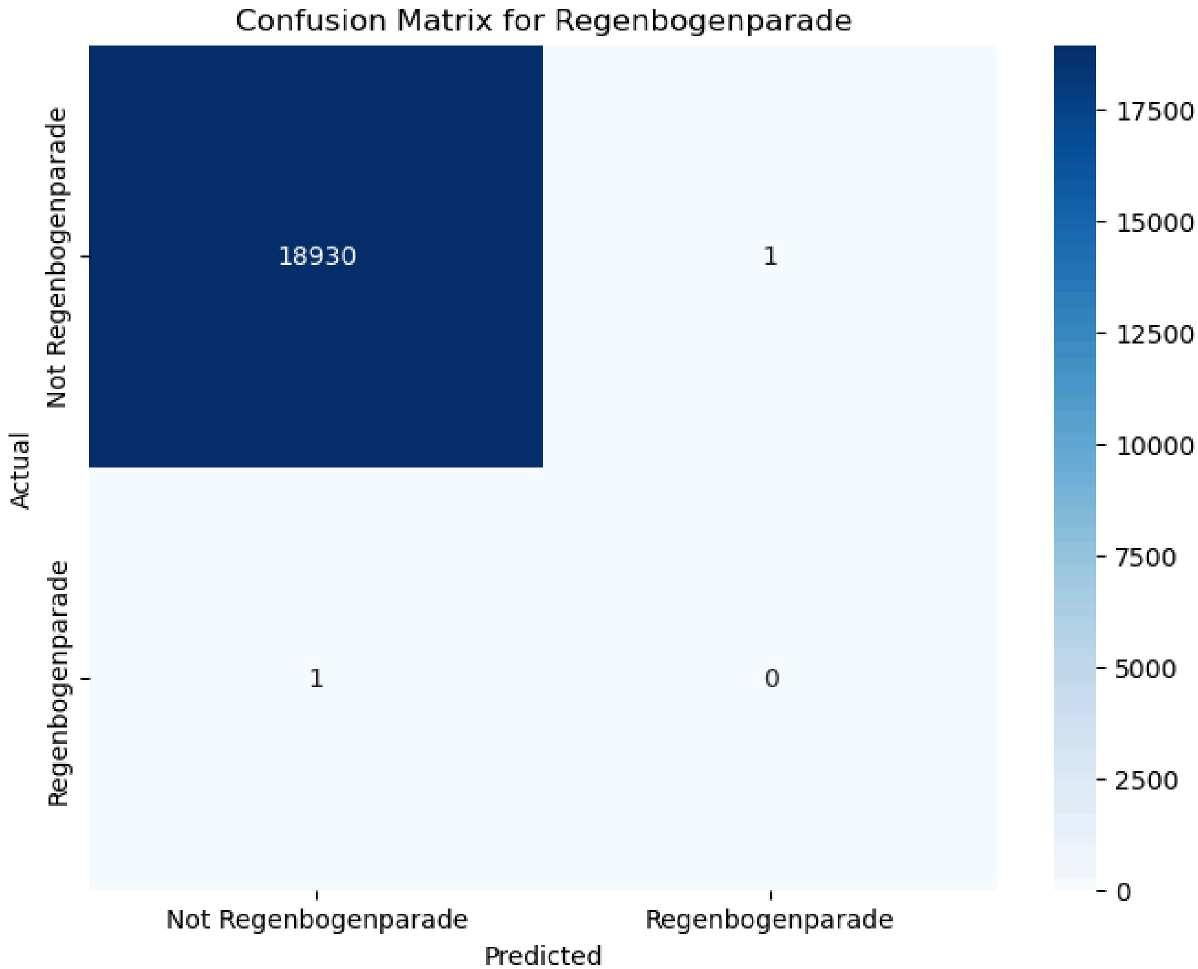

Confusion Matrix for Gleisbauarbeiten
Actual
Not Gleisbauarbeiten
Gleisbauarbeiten
18931
0
1
0
Predicted
Not Gleisbauarbeiten
Gleisbauarbeiten
17500
15000
12500
10000
7500
5000
2500
0

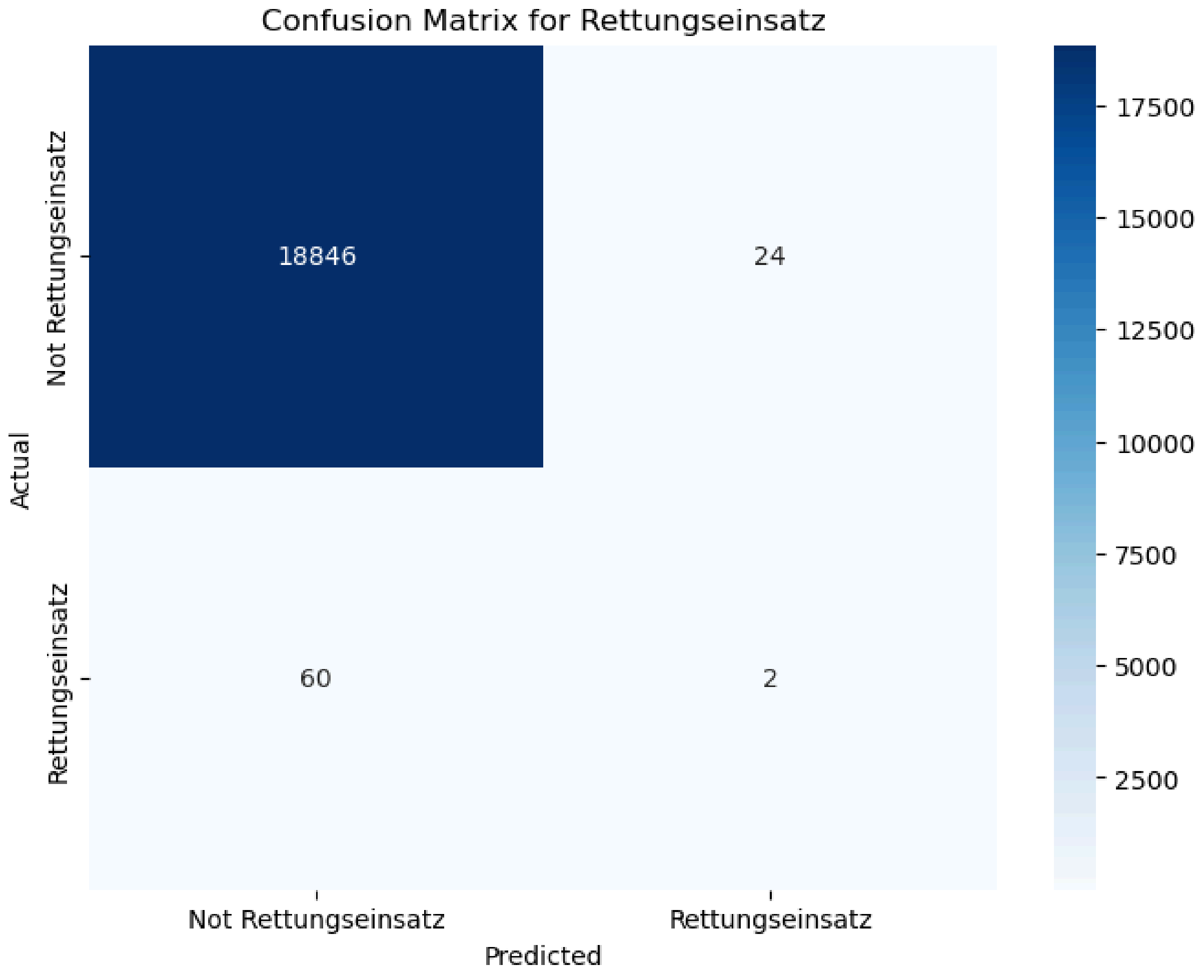

Confusion Matrix for Gleisschaden
Actual
Not Gleisschaden
Gleisschaden
14415
2069
2034
414
Not Gleisschaden
Gleisschaden
Predicted
14000
12000
10000
8000
6000
4000
2000

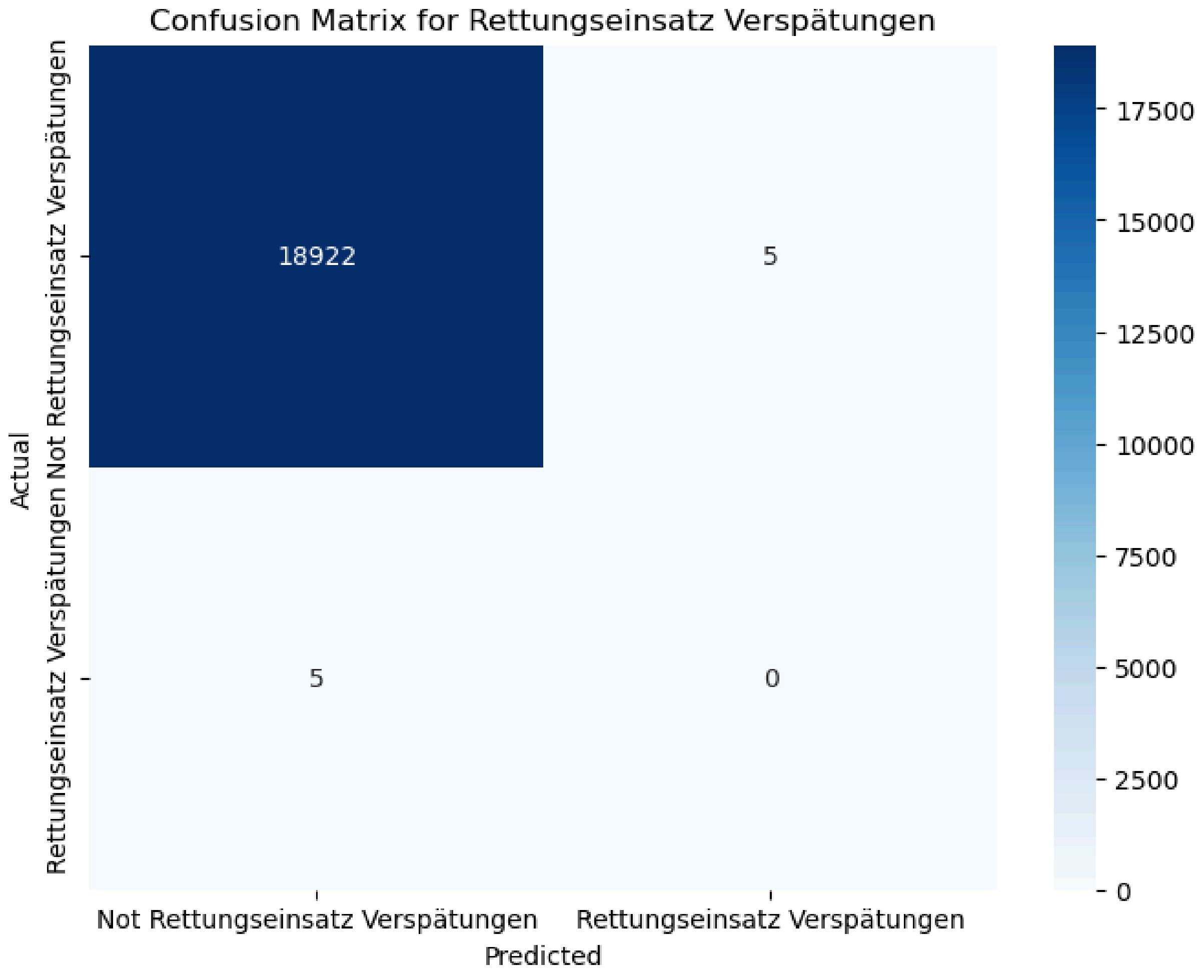

Confusion Matrix for Opernball
Actual
Not Opernball
Opernball
Predicted
Not Opernball
Opernball
18903
1
28
0
17500
15000
12500
10000
7500
5000
2500
0

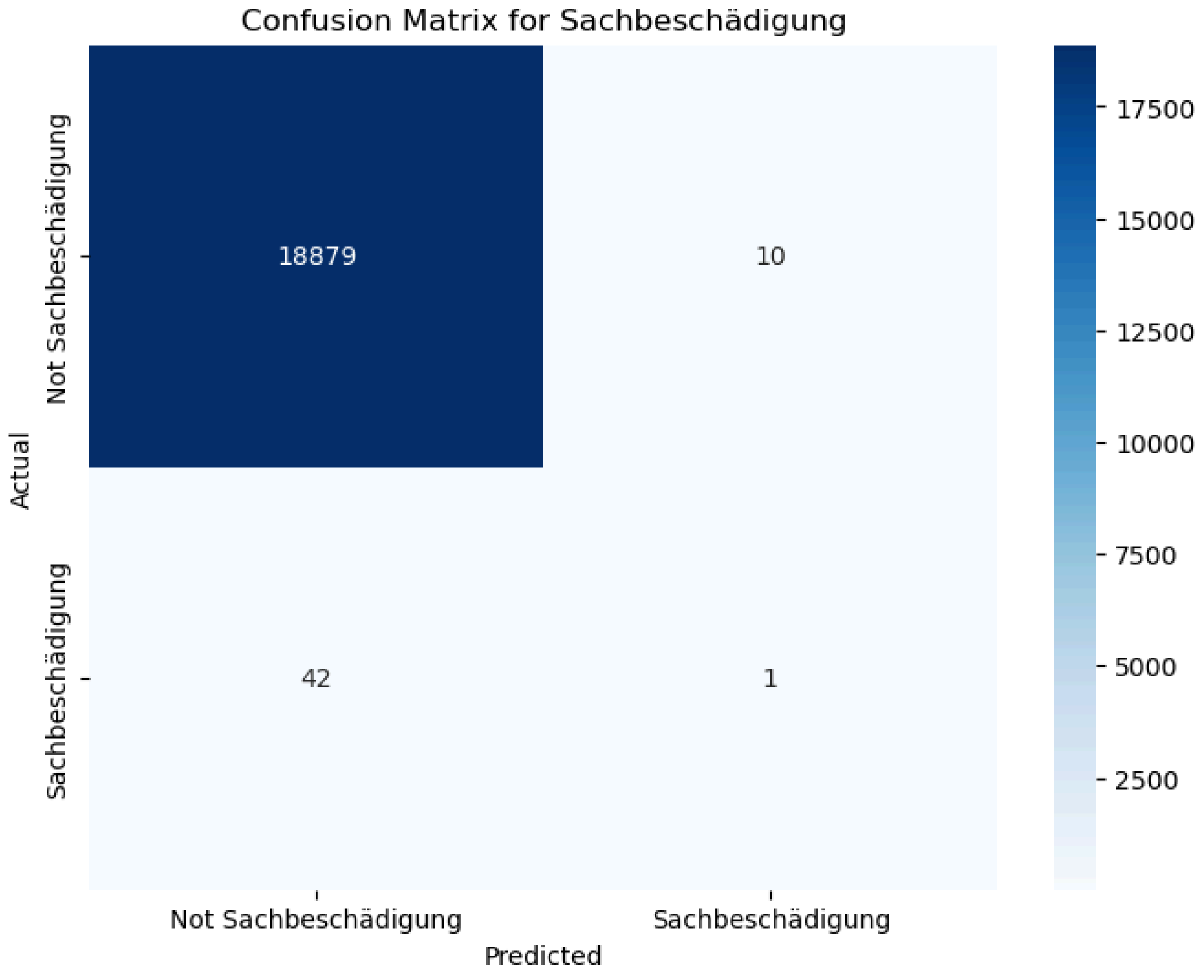

Confusion Matrix for Polizeieinsatz
Actual
Not Polizeieinsatz
Polizeieinsatz
18931
0
1
0
Predicted
Not Polizeieinsatz
Polizeieinsatz
17500
15000
12500
10000
7500
5000
2500
0

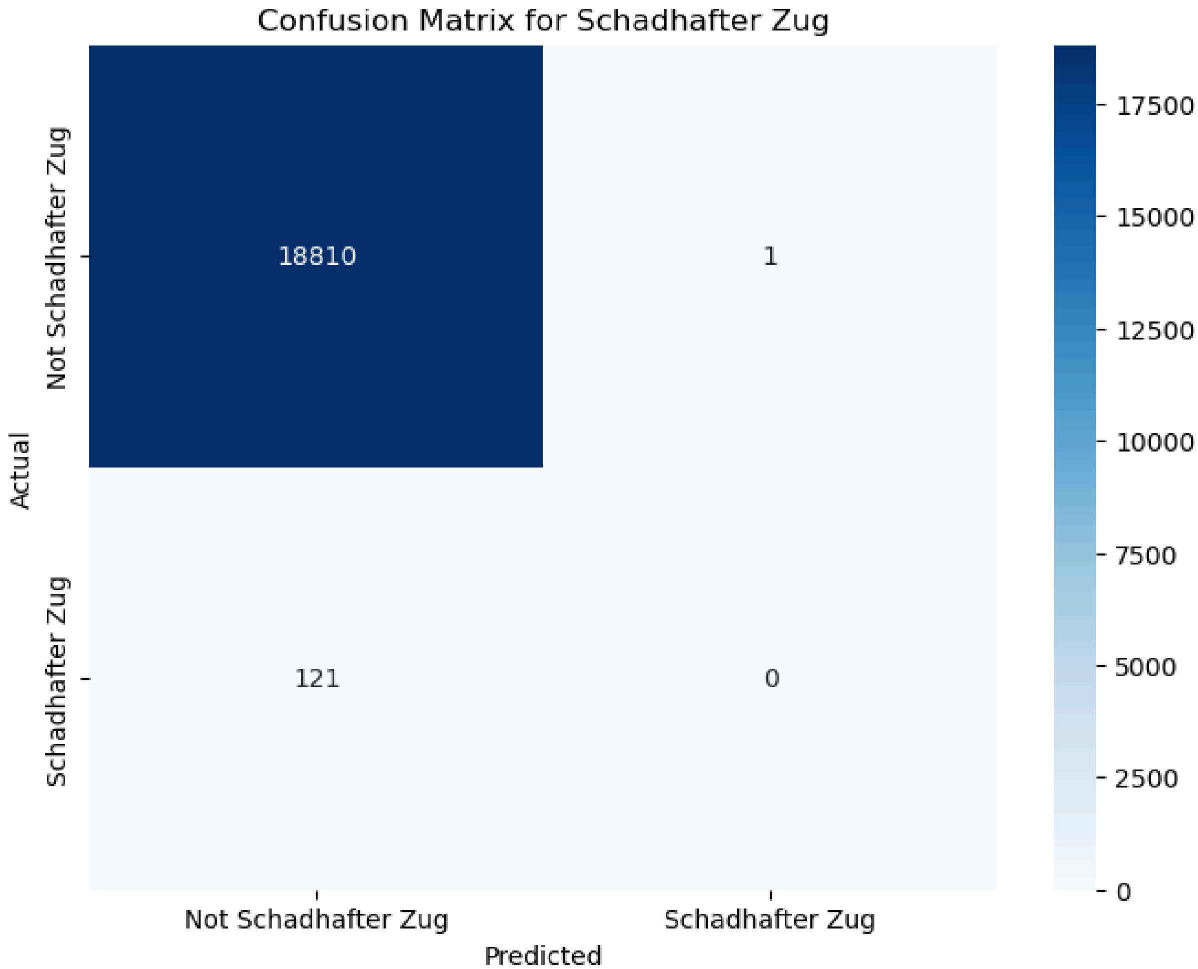

Confusion Matrix for Polizeieinsatz Verspätungen
Actual
Not Polizeieinsatz Verspätungen
Polizeieinsatz Verspätungen
14975
1787
1810
360
Not Polizeieinsatz Verspätungen
Polizeieinsatz Verspätungen
Predicted
14000
12000
10000
8000
6000
4000
2000

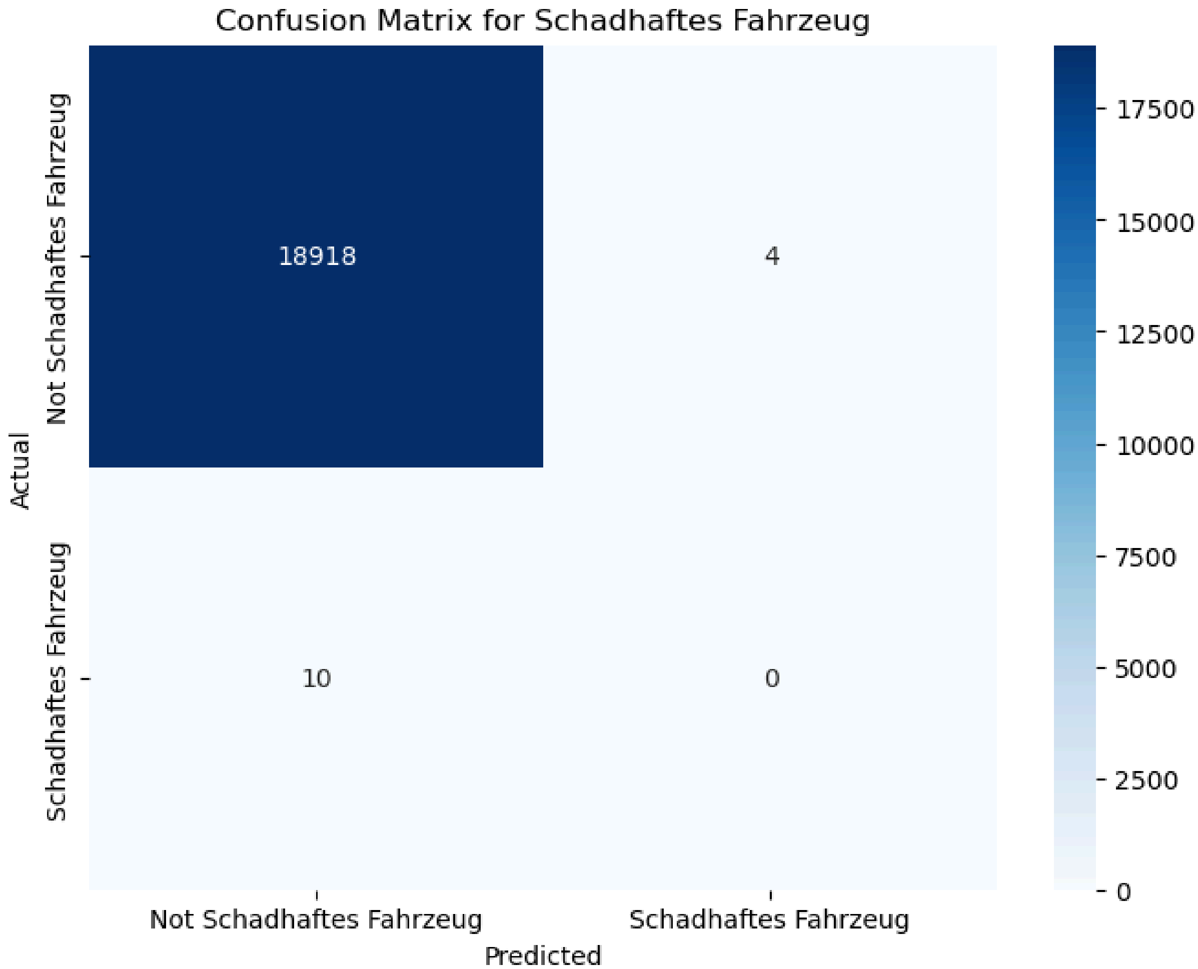

Confusion Matrix for Regenbogenparade
Actual
Not Regenbogenparade
Regenbogenparade
18930
1
1
0
Predicted
Not Regenbogenparade
Regenbogenparade
17500
15000
12500
10000
7500
5000
2500
0

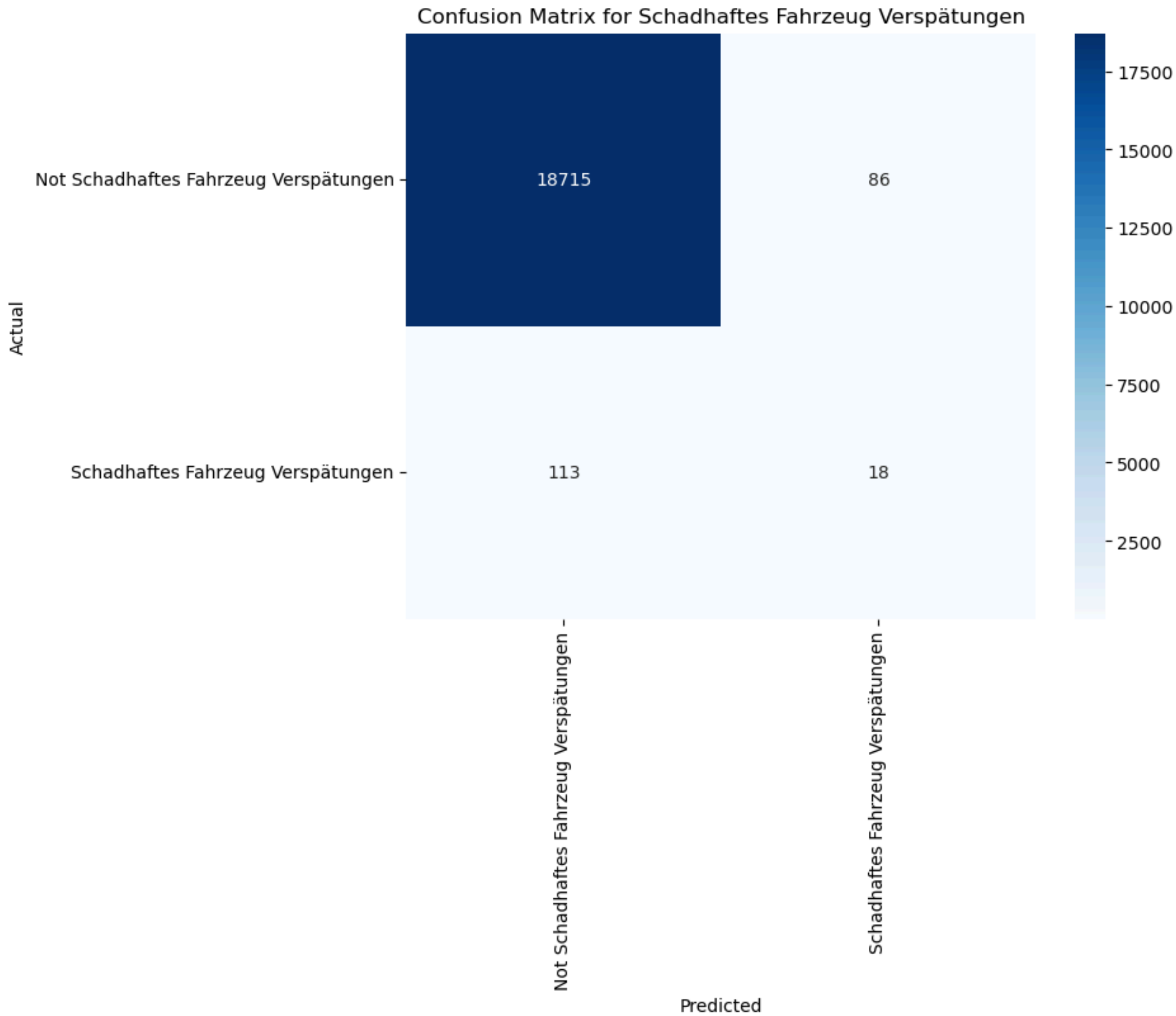

Confusion Matrix for Rettungseinsatz
Actual
Not Rettungseinsatz
Rettungseinsatz
18846
24
60
2
Predicted
17500
15000
12500
10000
7500
5000
2500

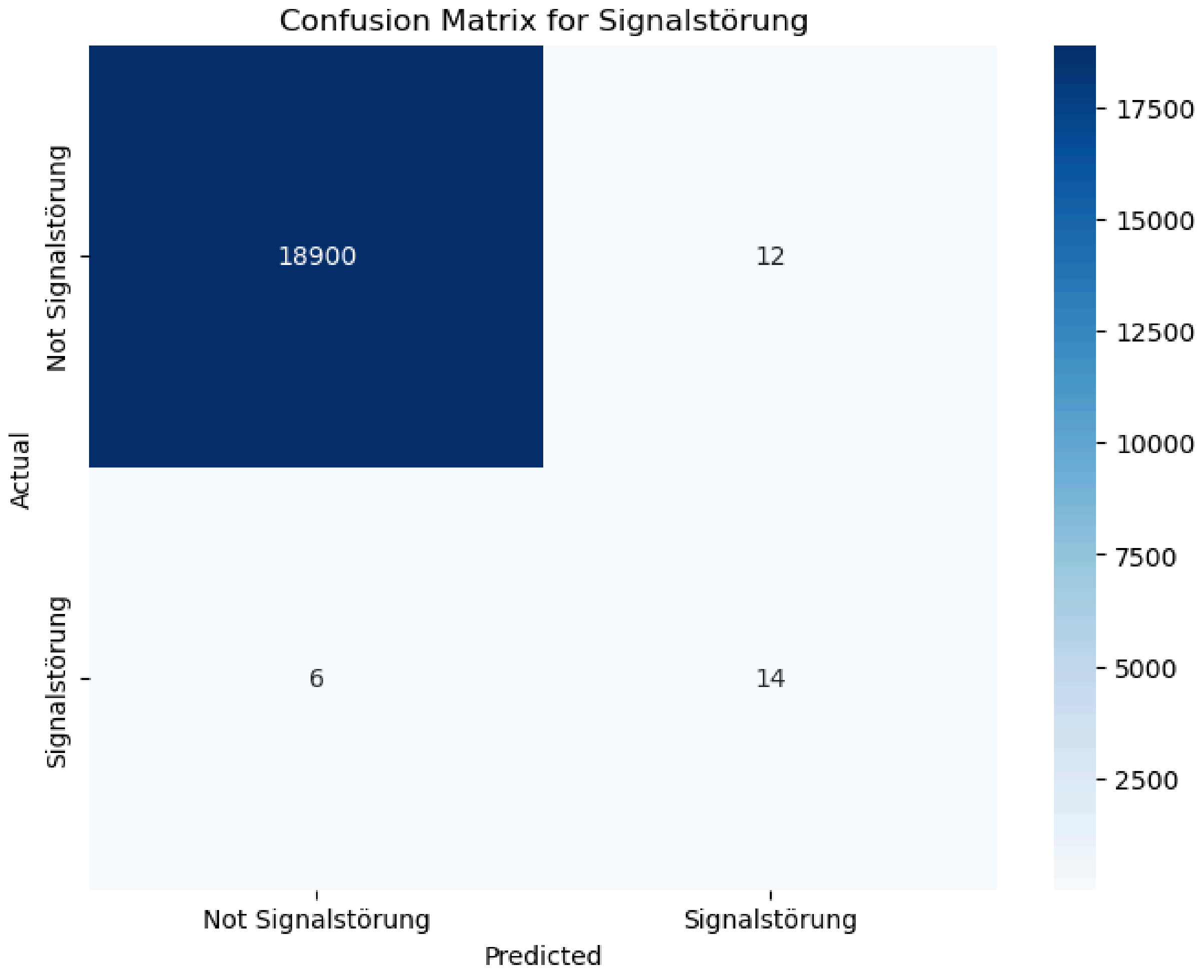

Confusion Matrix for Rettungseinsatz Verspätungen
Actual
Not Rettungseinsatz Verspätungen
Rettungseinsatz Verspätungen
Predicted
18922
5
5
0
17500
15000
12500
10000
7500
5000
2500
0

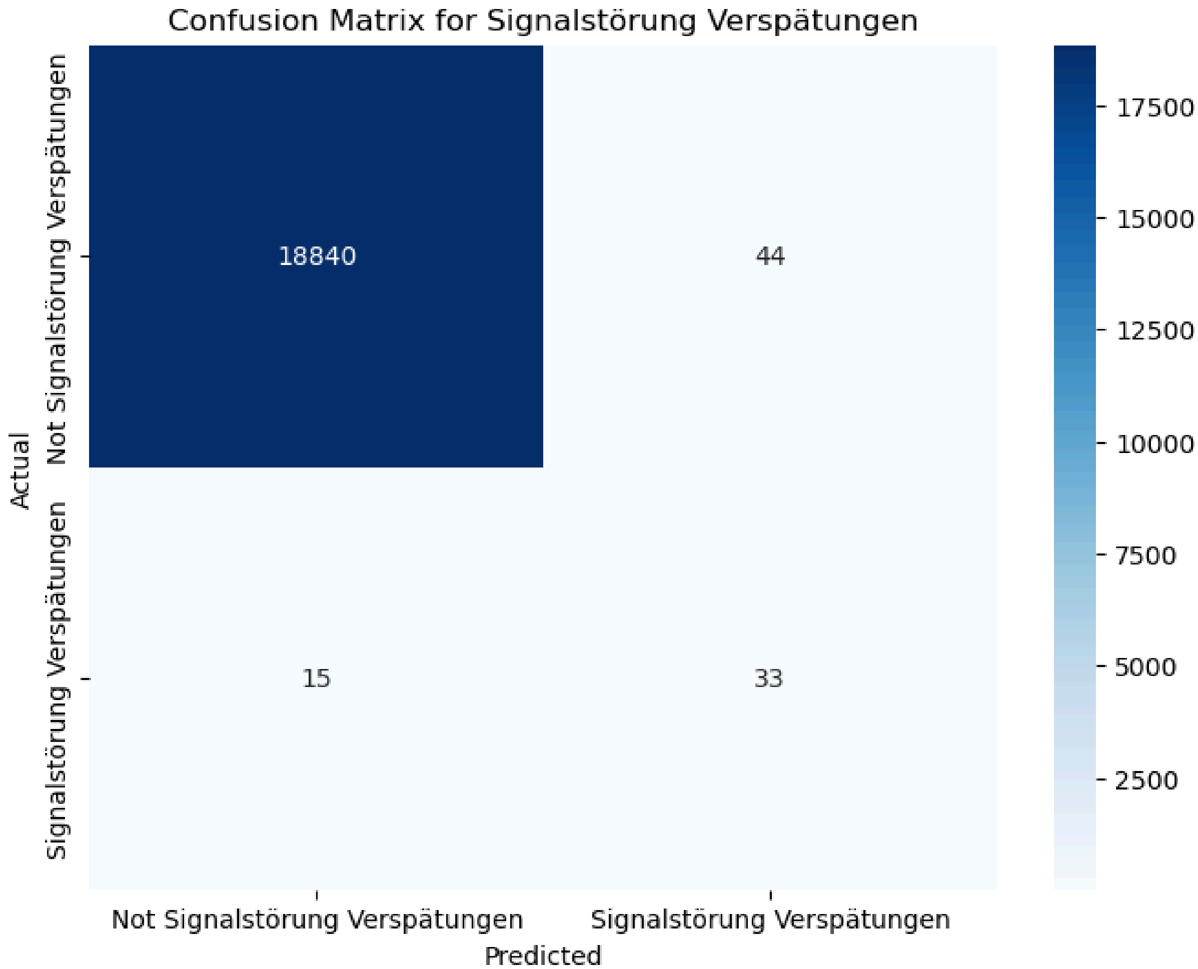

Confusion Matrix for Sachbeschädigung
Not Sachbeschädigung
Sachbeschädigung
Actual
18879
10
42
1
Not Sachbeschädigung
Sachbeschädigung
Predicted
17500
15000
12500
10000
7500
5000
2500

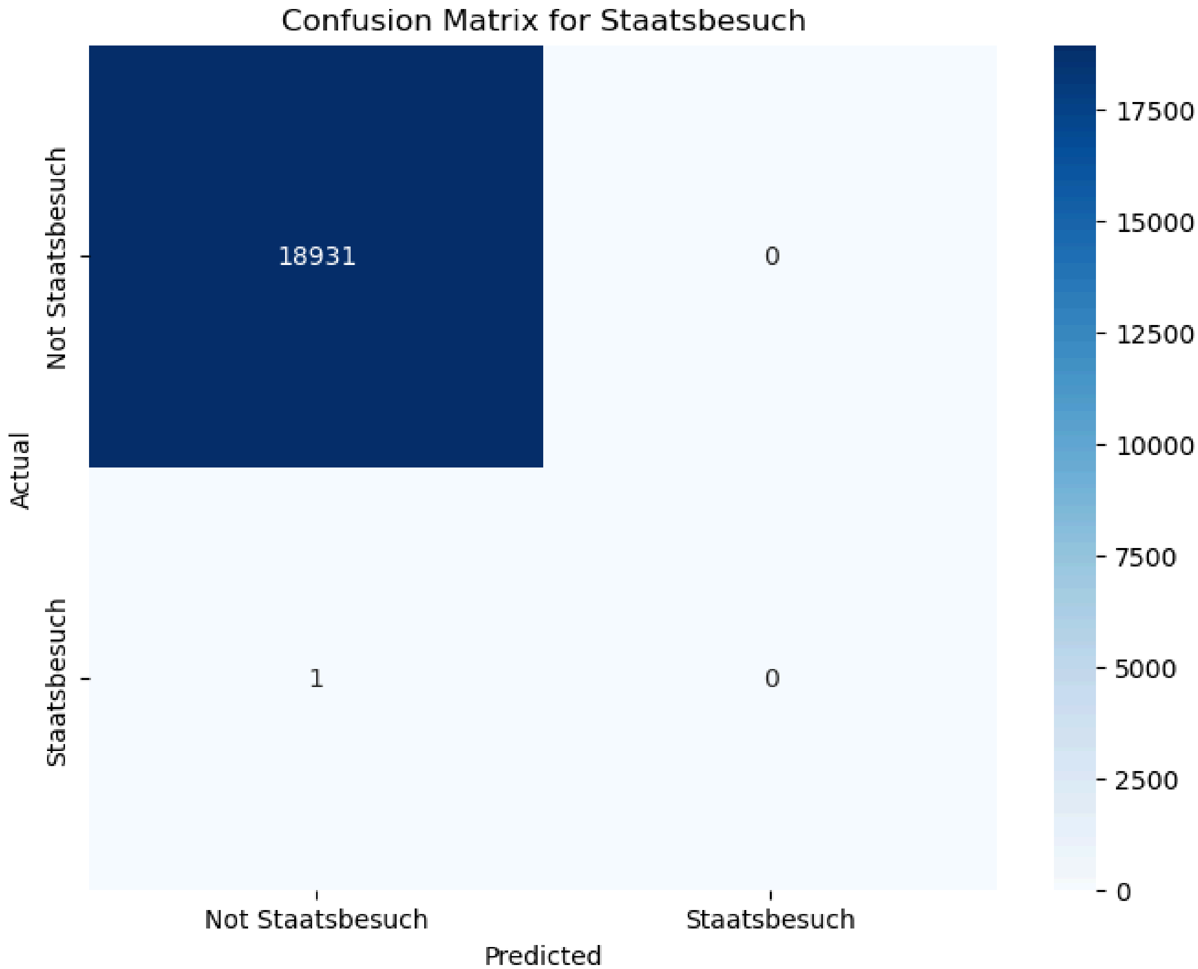

Confusion Matrix for Schadhafter Zug
Actual
Not Schadhafter Zug
Schadhafter Zug
18810
1
121
0
Predicted
Not Schadhafter Zug
Schadhafter Zug
17500
15000
12500
10000
7500
5000
2500
0

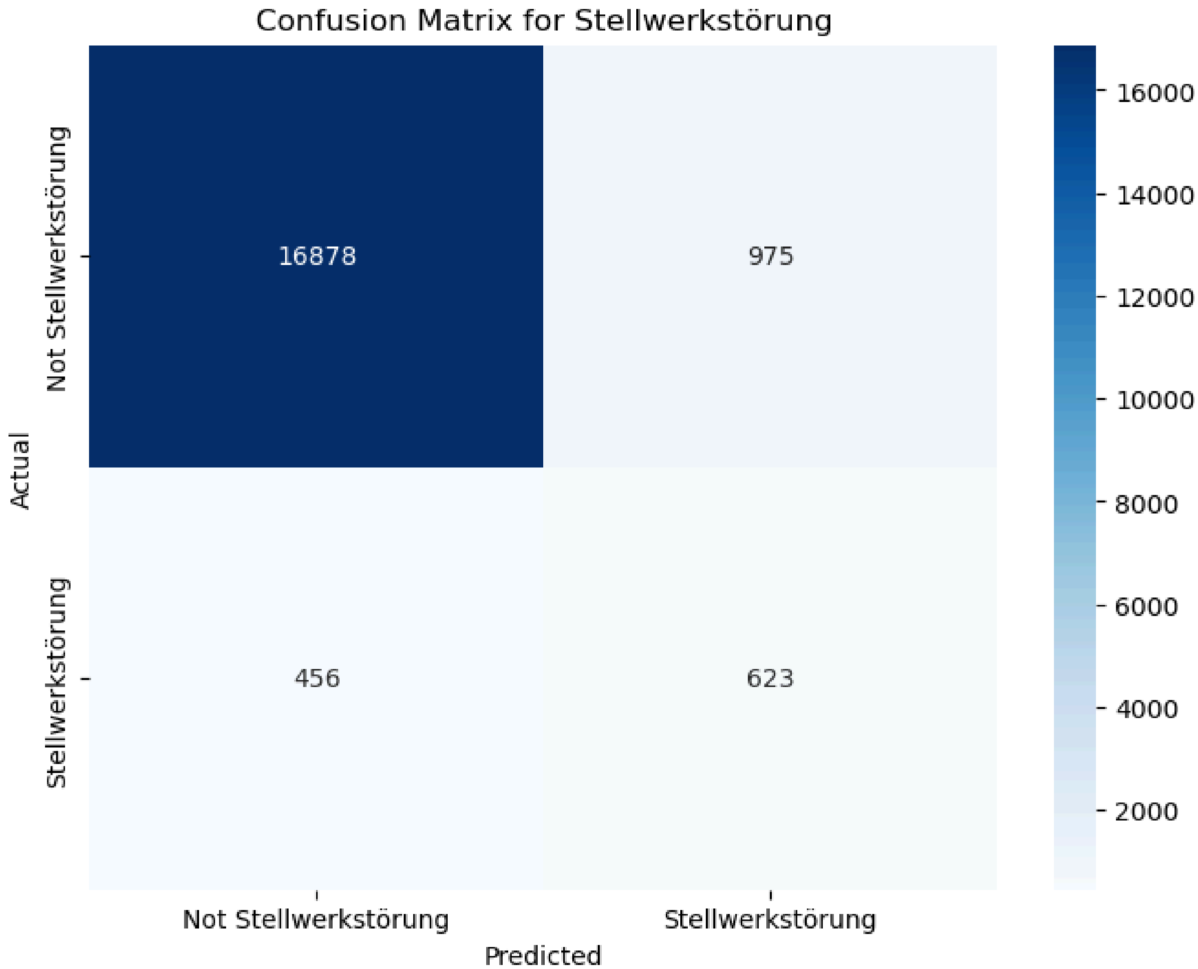

Confusion Matrix for Schadhaftes Fahrzeug
Actual
Not Schadhaftes Fahrzeug
Schadhaftes Fahrzeug
18918
4
10
0
Predicted
Not Schadhaftes Fahrzeug
Schadhaftes Fahrzeug
17500
15000
12500
10000
7500
5000
2500
0


Confusion Matrix for Schadhaftes Fahrzeug Verspätungen
Actual
Not Schadhaftes Fahrzeug Verspätungen
Schadhaftes Fahrzeug Verspätungen
18715
86
113
18
Predicted
Not Schadhaftes Fahrzeug Verspätungen
Schadhaftes Fahrzeug Verspätungen
17500
15000
12500
10000
7500
5000
2500

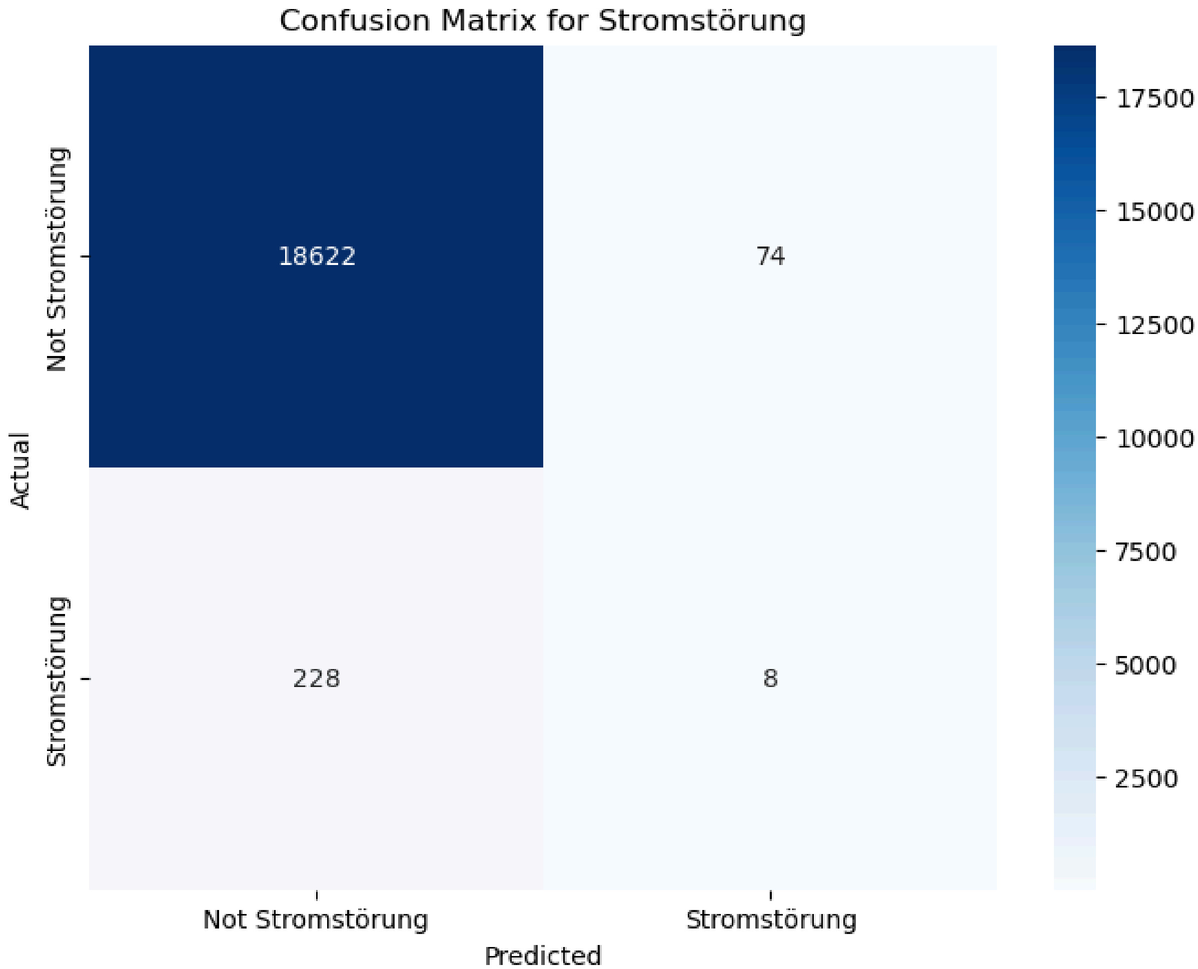

Confusion Matrix for Signalstörung
Actual
Not Signalstörung
Signalstörung
18900
12
6
14
Not Signalstörung
Signalstörung
Predicted
17500
15000
12500
10000
7500
5000
2500

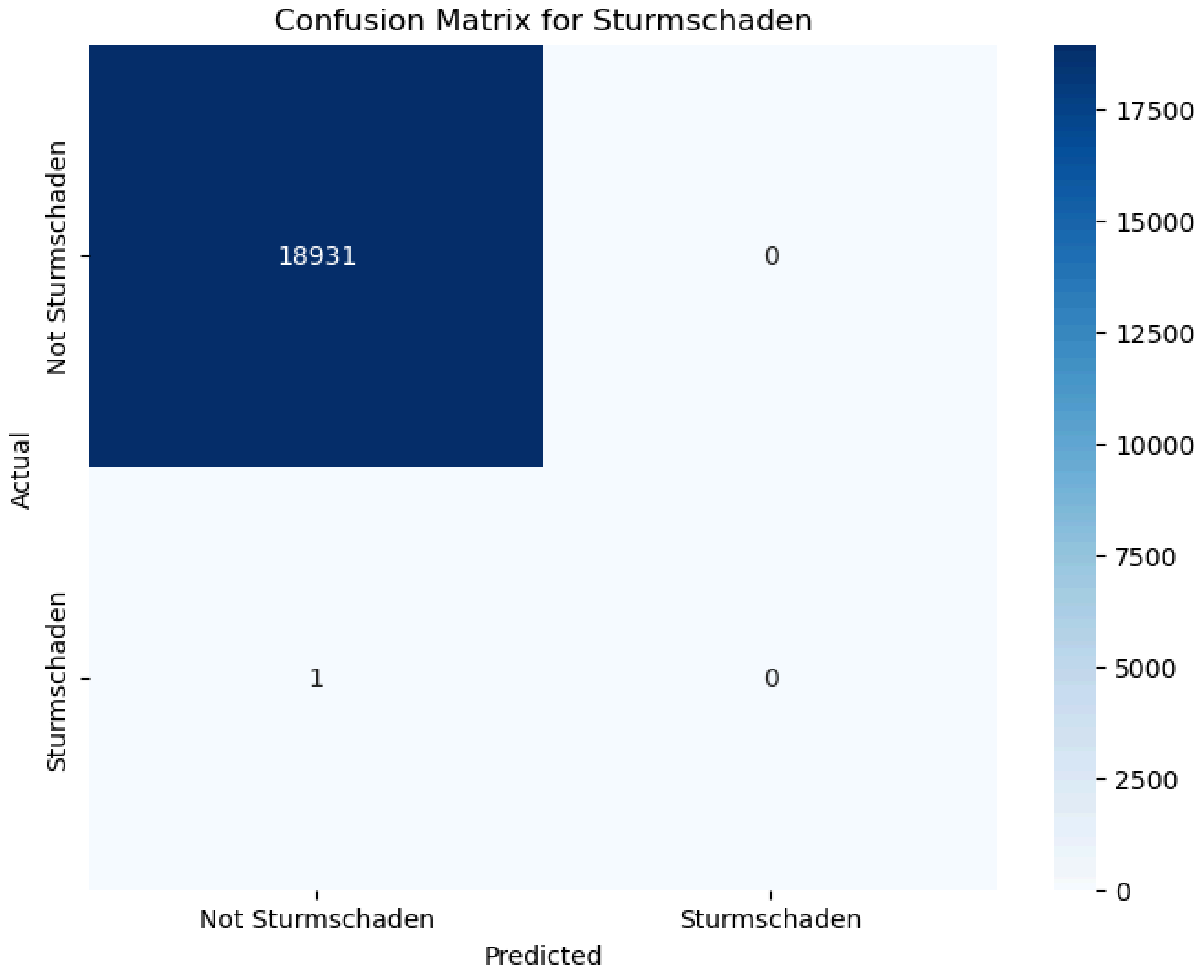

Confusion Matrix for Signalstörung Verspätungen
Actual
Not Signalstörung Verspätungen
Signalstörung Verspätungen
18840
44
15
33
Predicted
Not Signalstörung Verspätungen
Signalstörung Verspätungen
17500
15000
12500
10000
7500
5000
2500

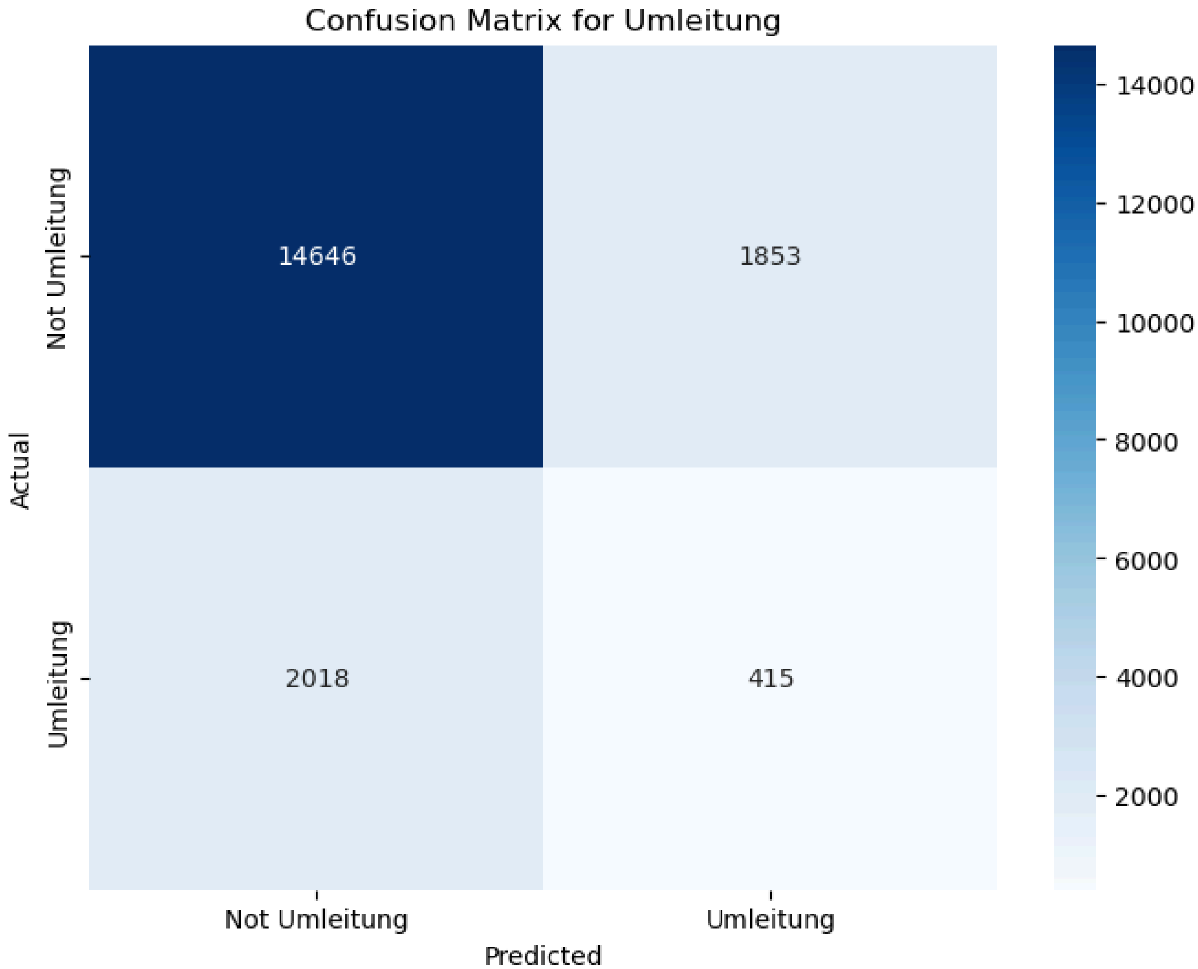

Confusion Matrix for Staatsbesuch
Actual
Not Staatsbesuch
Staatsbesuch
18931
0
1
0
Predicted
Not Staatsbesuch
Staatsbesuch
17500
15000
12500
10000
7500
5000
2500
0

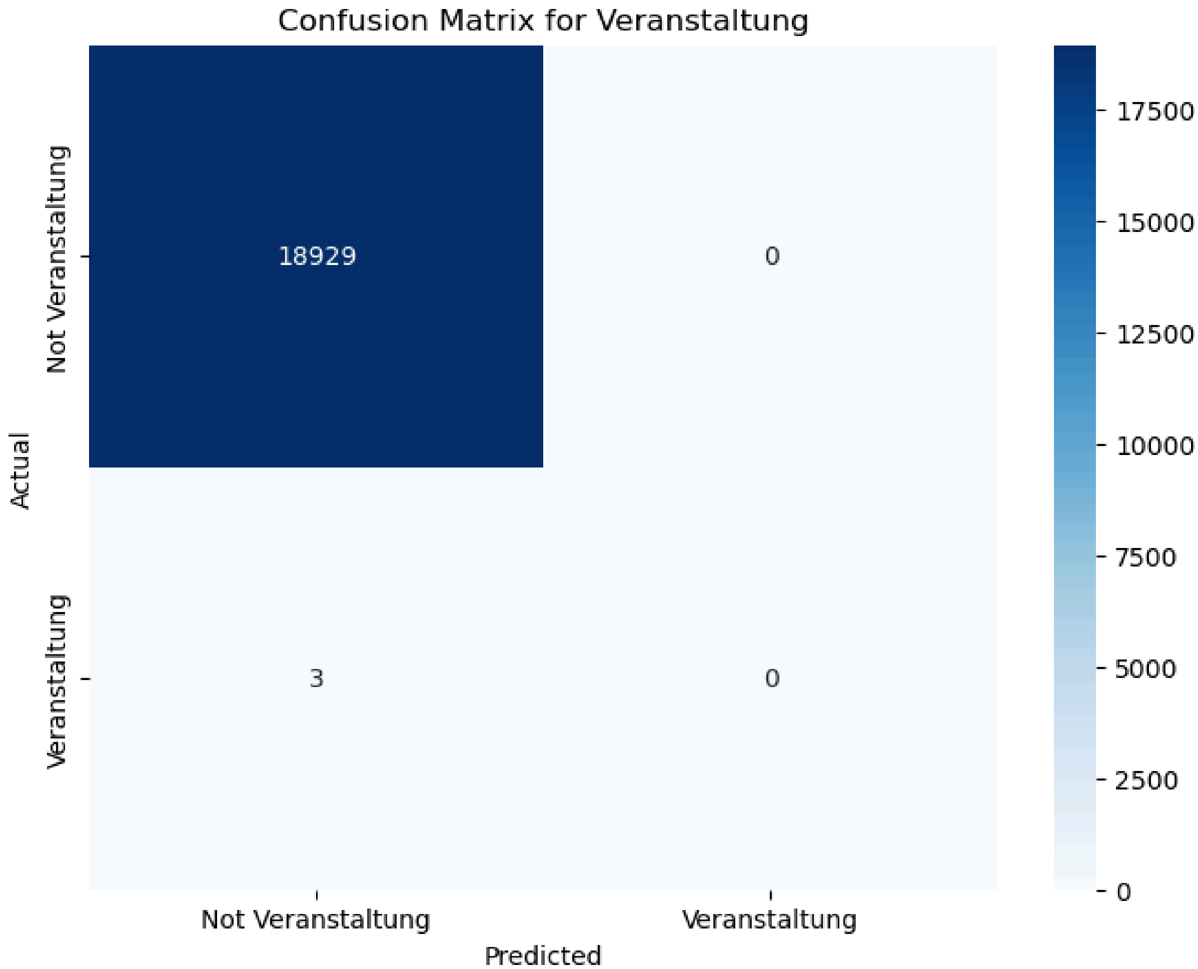


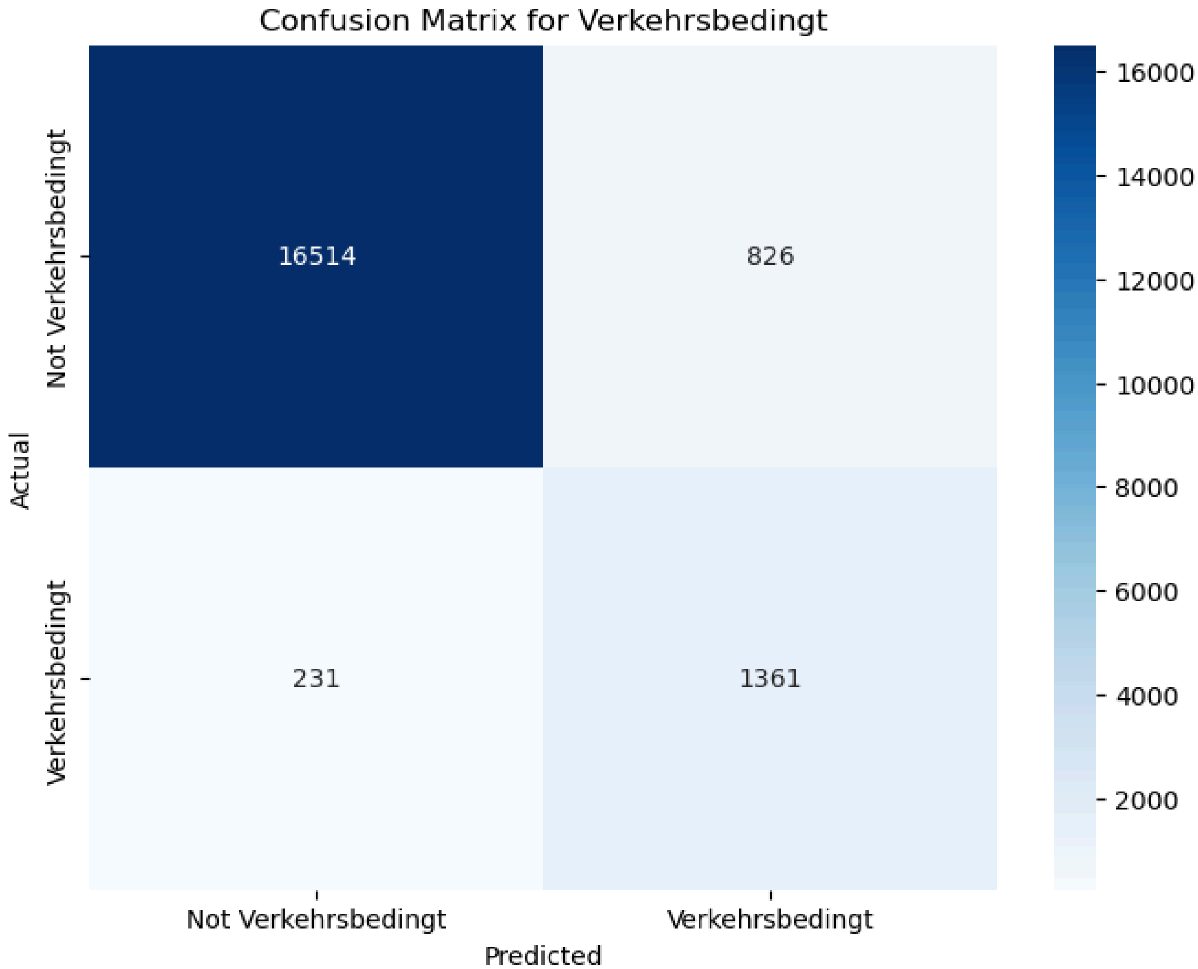
Confusion Matrix for Stellwerkstörung
Actual
Not Stellwerkstörung
Stellwerkstörung
16878
975
456
623
Not Stellwerkstörung
Stellwerkstörung
Predicted
16000
14000
12000
10000
8000
6000
4000
2000

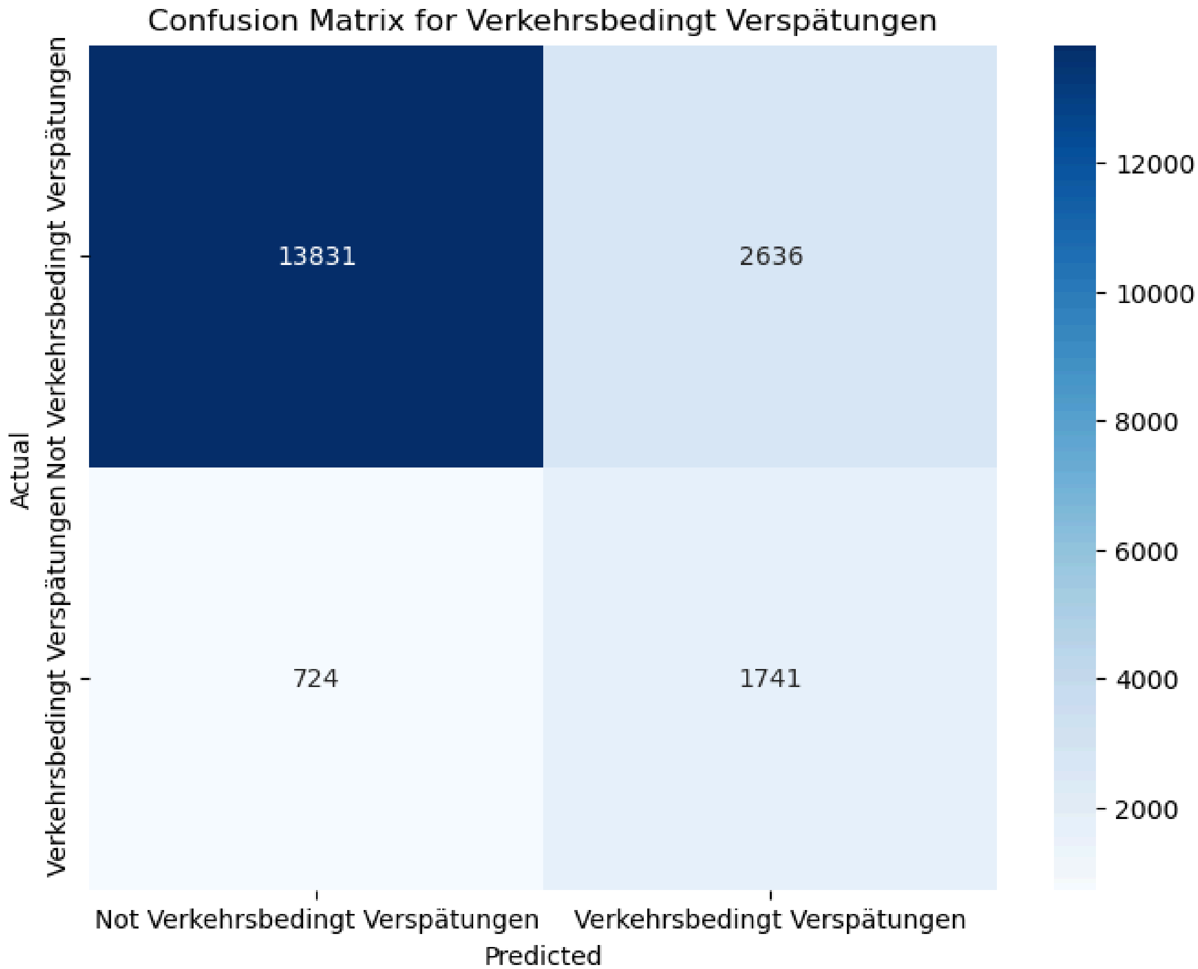

Confusion Matrix for Stromstörung
Actual
Not Stromstörung
Stromstörung
18622
74
228
8
Predicted
Not Stromstörung
Stromstörung
17500
15000
12500
10000
7500
5000
2500

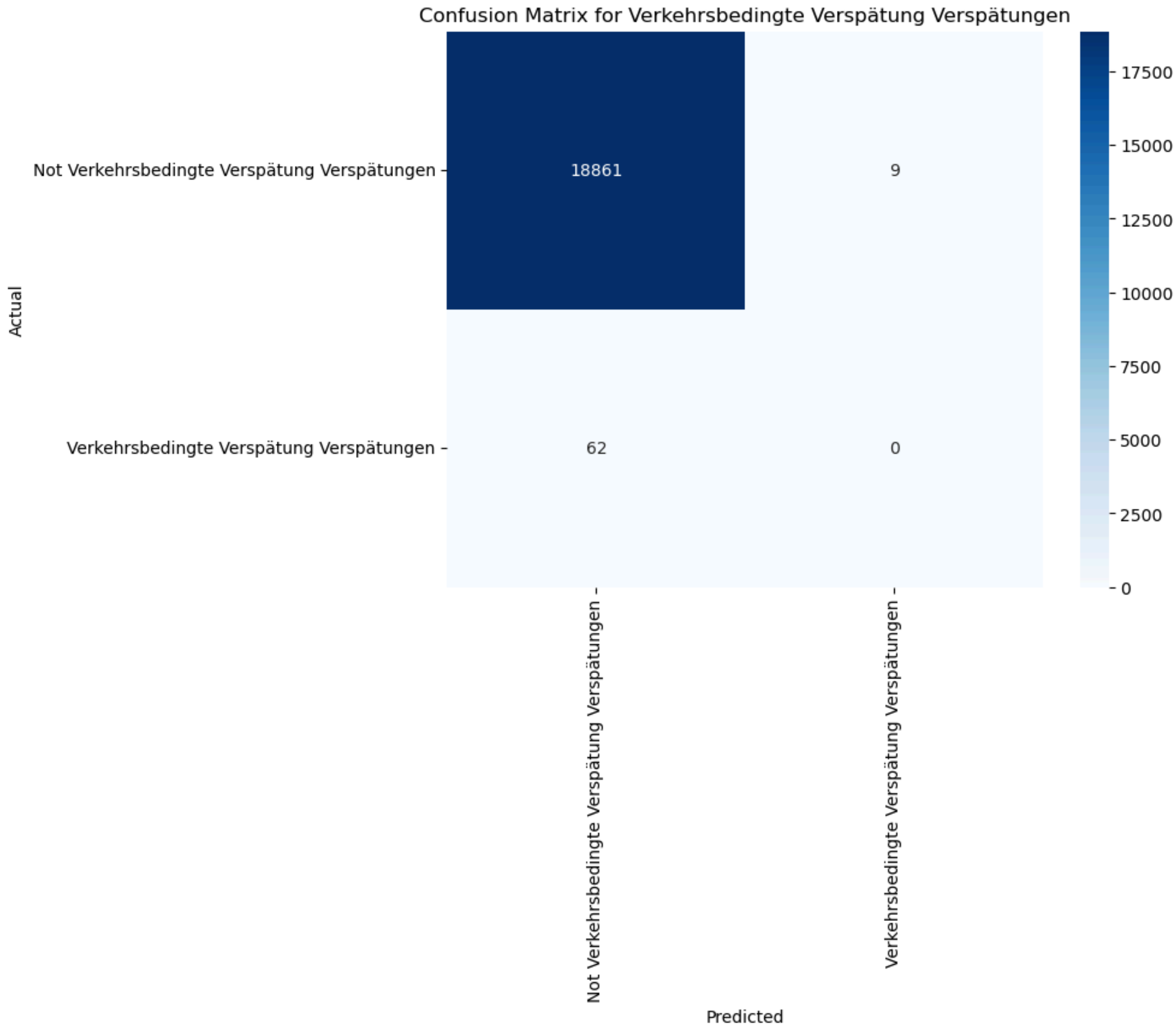

Confusion Matrix for Sturmschaden
Actual
Not Sturmschaden
Sturmschaden
Predicted
Not Sturmschaden
Sturmschaden
18931
0
1
0
17500
15000
12500
10000
7500
5000
2500
0

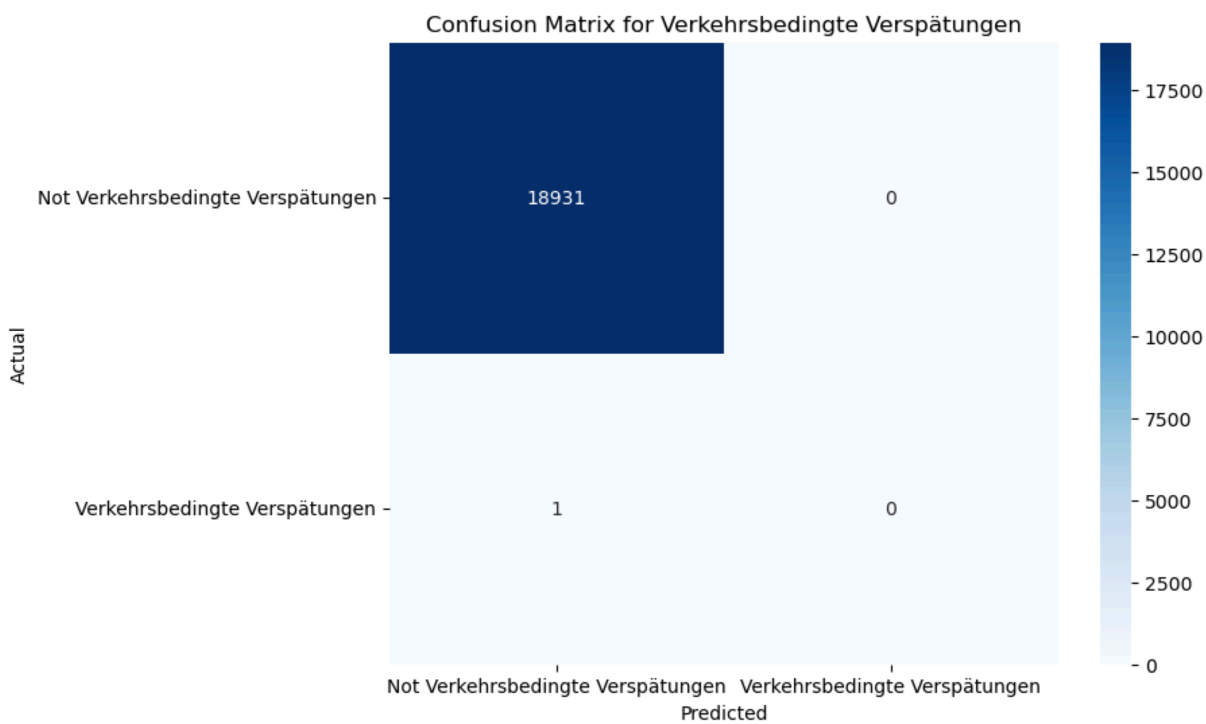

Confusion Matrix for Umleitung
14646
1853
2018
415
Not Umleitung
Umleitung
Actual
Not Umleitung
Umleitung
Predicted
14000
12000
10000
8000
6000
4000
2000

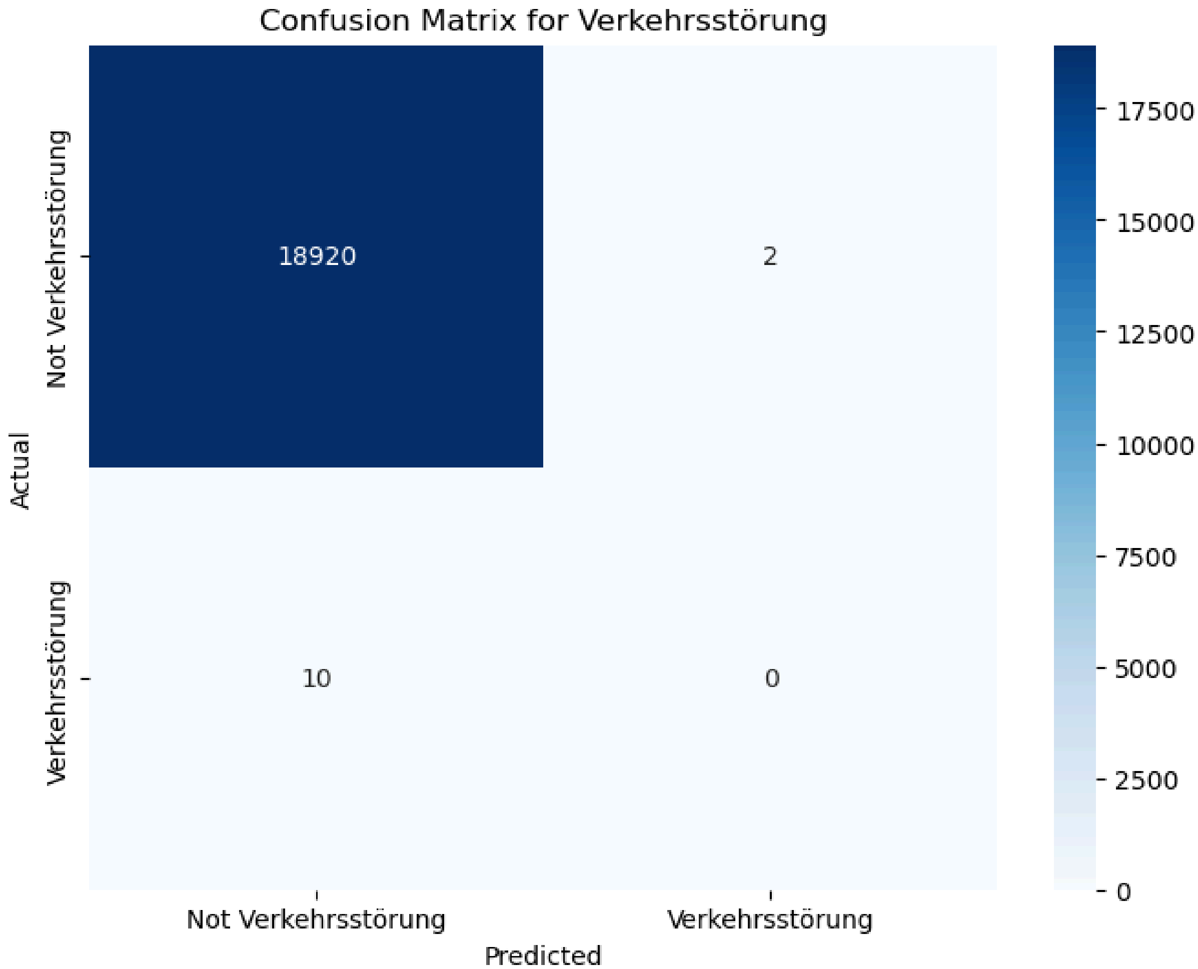

Confusion Matrix for Veranstaltung
Not Veranstaltung
Veranstaltung
Actual
18929
0
3
0
Not Veranstaltung
Veranstaltung
Predicted
17500
15000
12500
10000
7500
5000
2500
0

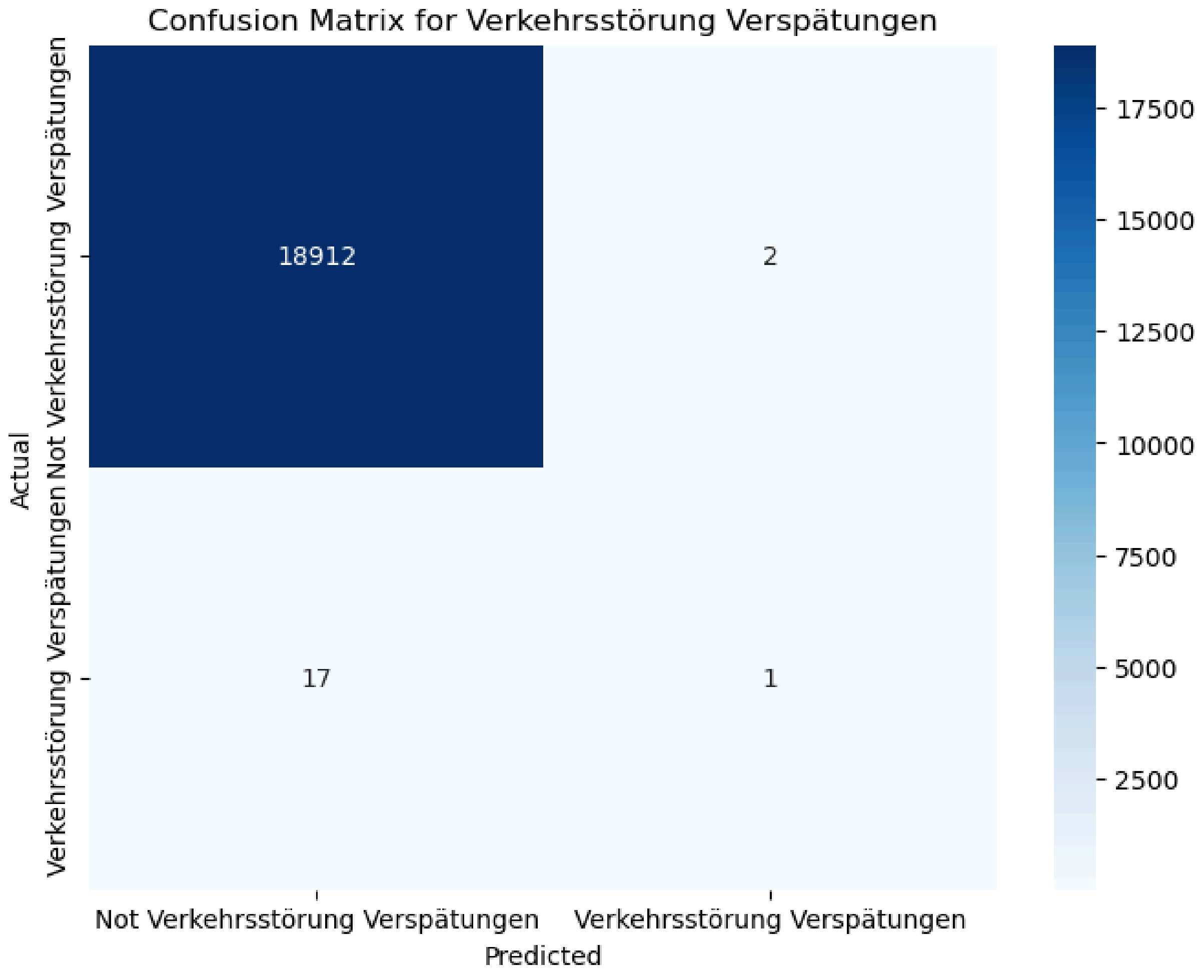

Confusion Matrix for Verkehrsbedingt
Actual
Not Verkehrsbedingt
Verkehrsbedingt
16514
826
231
1361
Predicted
Not Verkehrsbedingt
Verkehrsbedingt
16000
14000
12000
10000
8000
6000
4000
2000

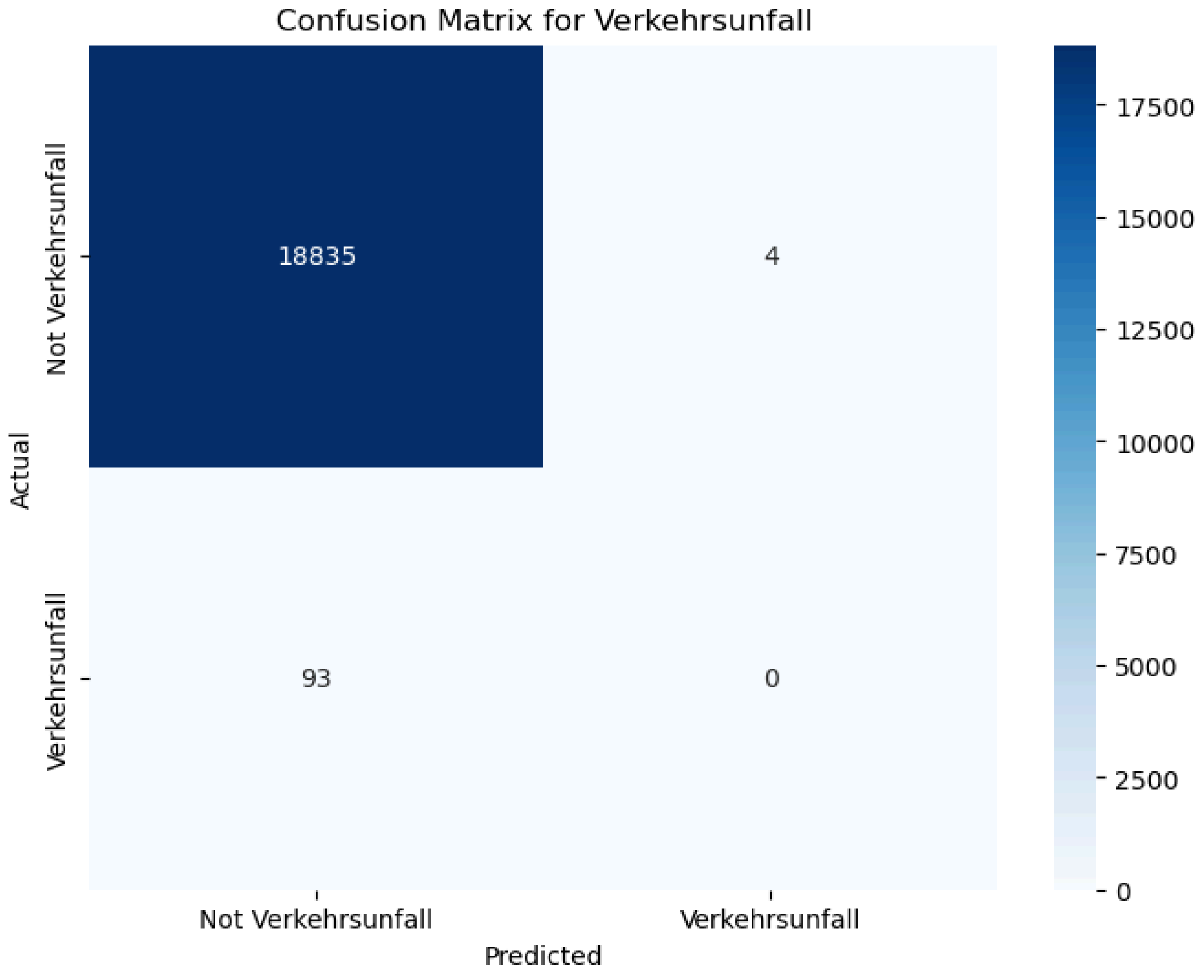

Confusion Matrix for Verkehrsbedingt Verspätungen
Actual
Not Verkehrsbedingt Verspätungen
Verkehrsbedingt Verspätungen
13831
2636
724
1741
Predicted
Not Verkehrsbedingt Verspätungen
Verkehrsbedingt Verspätungen
12000
10000
8000
6000
4000
2000

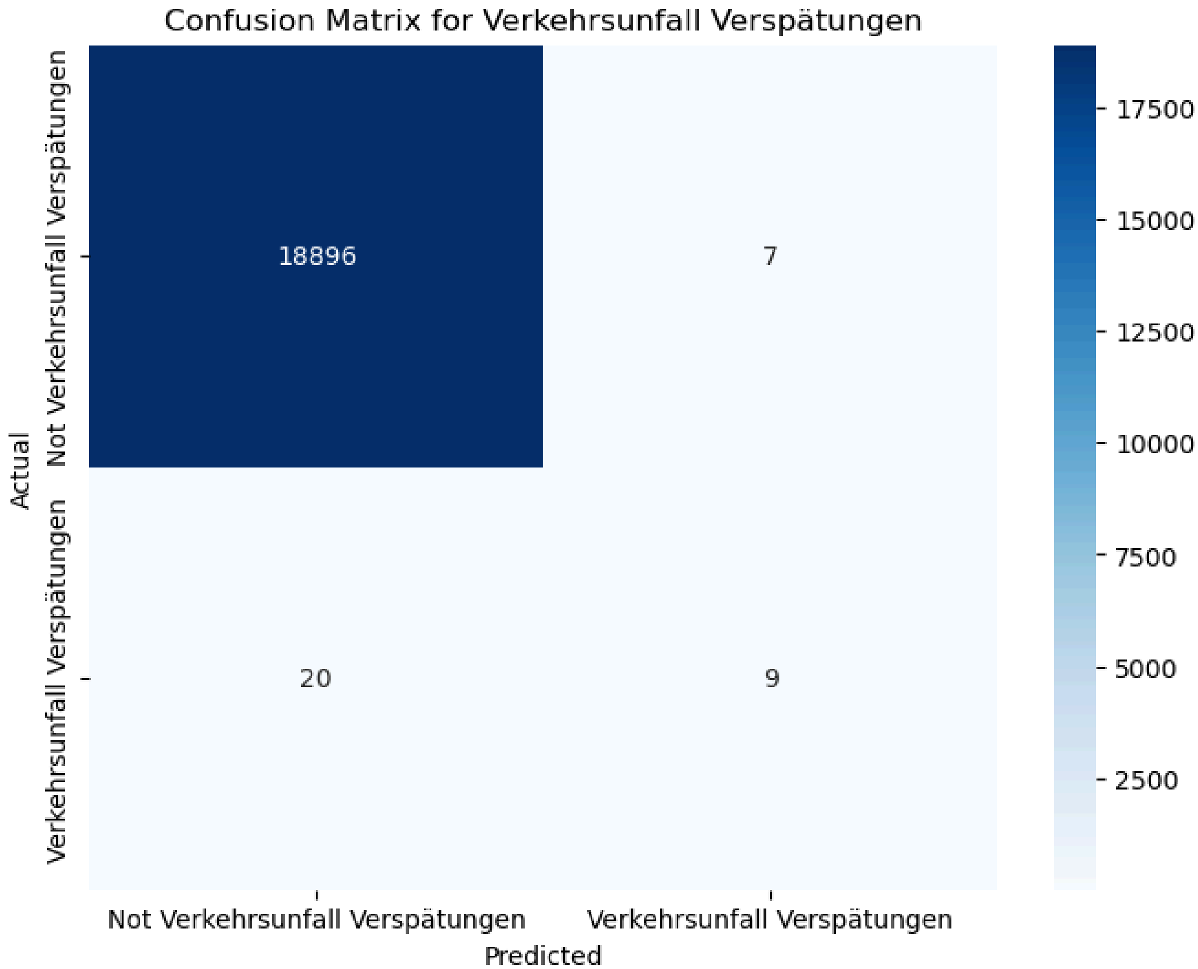

Confusion Matrix for Verkehrsbedingte Verspätung Verspätungen
Actual
Not Verkehrsbedingte Verspätung Verspätungen
Verkehrsbedingte Verspätung Verspätungen
18861
9
62
0
Predicted
17500
15000
12500
10000
7500
5000
2500
0

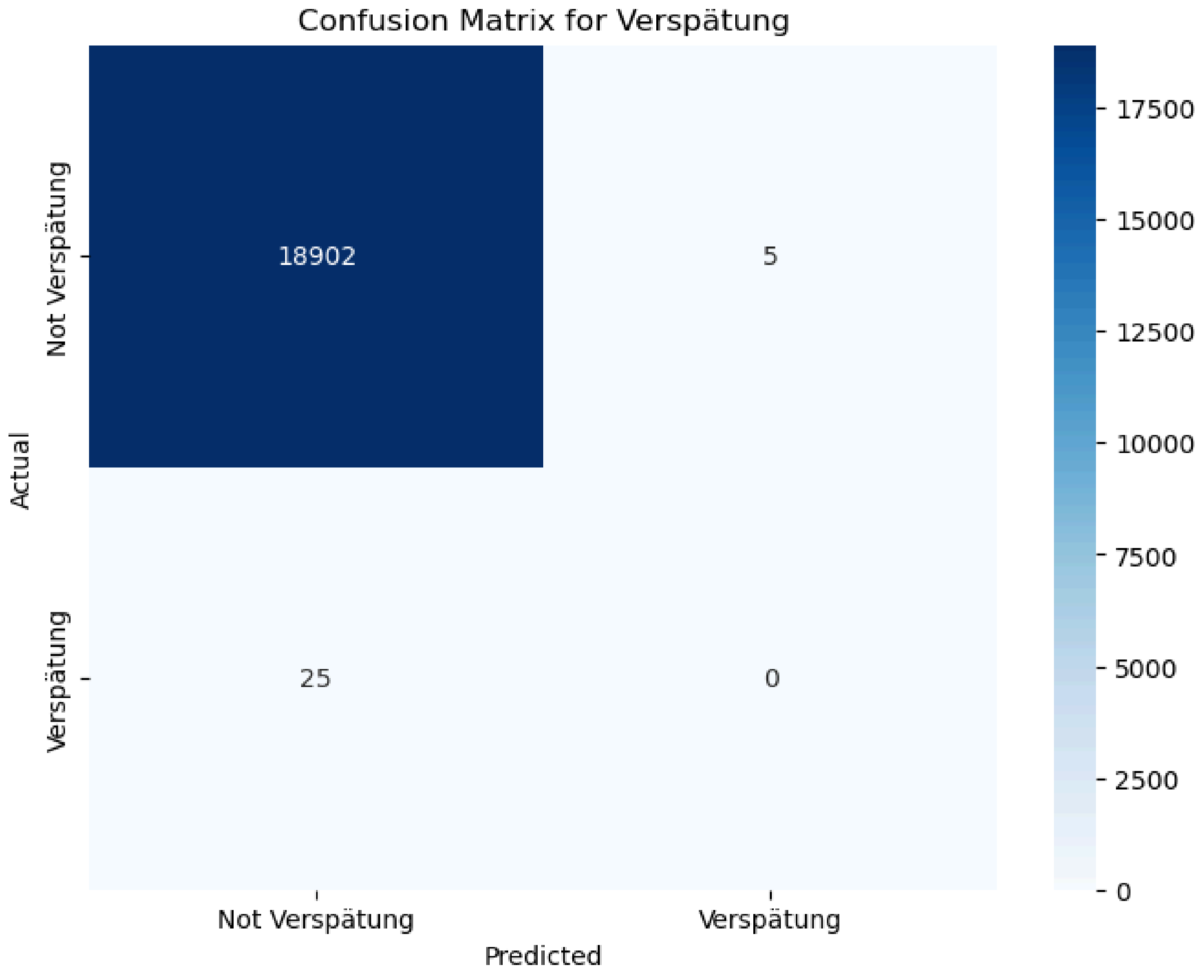

Confusion Matrix for Verkehrsbedingte Verspätungen
Actual
Not Verkehrsbedingte Verspätungen
Verkehrsbedingte Verspätungen
18931
0
1
0
Predicted
17500
15000
12500
10000
7500
5000
2500
0

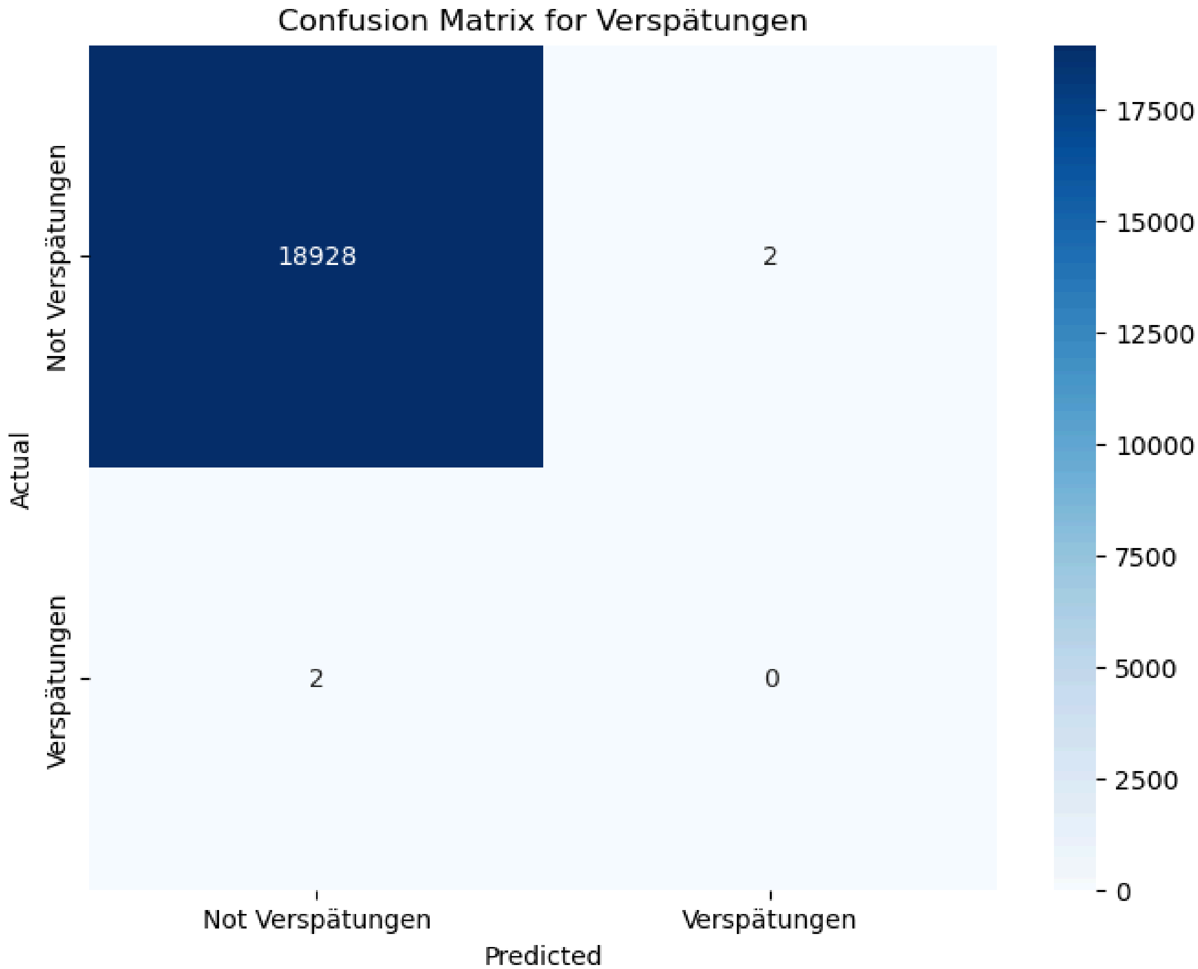

Confusion Matrix for Verkehrsstörung
Actual
Not Verkehrsstörung
Verkehrsstörung
Predicted
Not Verkehrsstörung
Verkehrsstörung
18920
2
10
0
17500
15000
12500
10000
7500
5000
2500
0


Confusion Matrix for Verkehrsstörung Verspätungen
Actual
Not Verkehrsstörung Verspätungen
Verkehrsstörung Verspätungen
18912
2
17
1
Predicted
Not Verkehrsstörung Verspätungen
Verkehrsstörung Verspätungen
17500
15000
12500
10000
7500
5000
2500


Confusion Matrix for Verkehrsunfall
Actual
Not Verkehrsunfall
Verkehrsunfall
18835
4
93
0
Predicted
Not Verkehrsunfall
Verkehrsunfall
17500
15000
12500
10000
7500
5000
2500
0


Confusion Matrix for Verkehrsunfall Verspätungen
Actual
Not Verkehrsunfall Verspätungen
Verkehrsunfall Verspätungen
18896
7
20
9
Predicted
Not Verkehrsunfall Verspätungen
Verkehrsunfall Verspätungen
17500
15000
12500
10000
7500
5000
2500


Confusion Matrix for Verspätung
Actual
Not Verspätung
Verspätung
18902
5
25
0
Predicted
Not Verspätung
Verspätung
17500
15000
12500
10000
7500
5000
2500
0



In [ ]:
```
# Task 8: Test model on unknown data
```

In [23]:
```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np

# Load your data
# Assuming data_train and data_test are your DataFrames
# data_train = pd.read_csv('path_to_train_data.csv')
# data_test = pd.read_csv('path_to_test_data.csv')

# Define features and target
features = ['temp_dailyMean', 'hum_dailyMin', 'hum_dailyMax', 'hum_dailyMean', 'bus
target_column = 'class'

# Set your best model with class weighting
best_model = make_pipeline(
    StandardScaler(),
    RandomForestClassifier(random_state=42, criterion="gini", class_weight='balance
)

def train_and_predict(best_model, train_data: pd.DataFrame, test_data: pd.DataFrame
```

```
    # Train the model
    best_model.fit(train_data[features], train_data[target_column])

    # Predict on the test data
    predictions = best_model.predict(test_data[features])

    return predictions

# Train with train data, predict on hidden test data
unknown_prediction = train_and_predict(best_model, data_train, data_test)

# Check the predictions
disruption_preds = np.unique(unknown_prediction)
print("Unique Disruption Predictions:", disruption_preds)

# Evaluate the model on the training data
train_predictions = best_model.predict(data_train[features])
print("Confusion Matrix on Training Data:\n", confusion_matrix(data_train[target_co
print("Classification Report on Training Data:\n", classification_report(data_train

# Ensure predictions are correct
assert len(unknown_prediction.shape) == 1, "Predictions should only have 1 column!"
assert unknown_prediction.shape[0] == data_test.shape[0], "Predictions should have
```

```
Unique Disruption Predictions: [0]
Confusion Matrix on Training Data:
 [[2 0]
 [0 1]]
Classification Report on Training Data:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00         2
           1       1.00      1.00      1.00         1

    accuracy                           1.00         3
   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3
```

In [ ]:

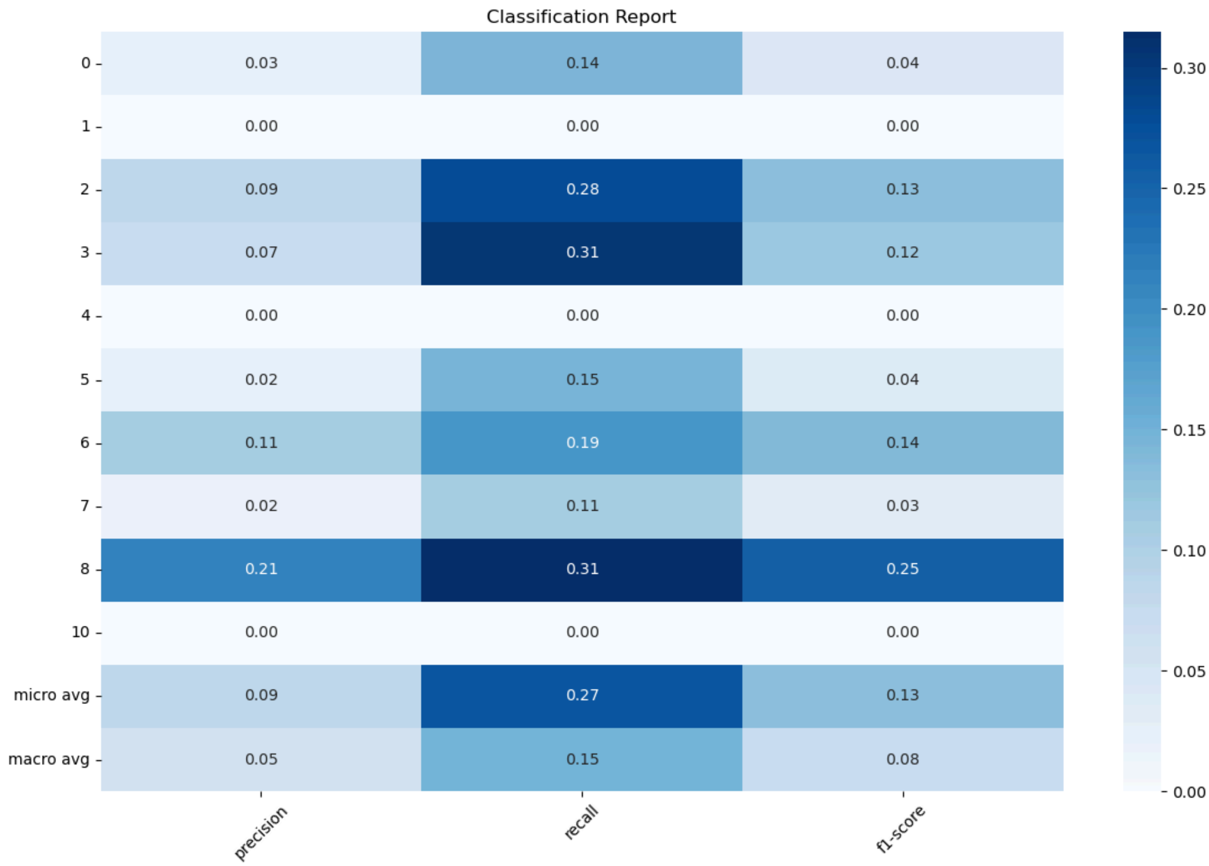
```
# 8.2 Visualize Results
```

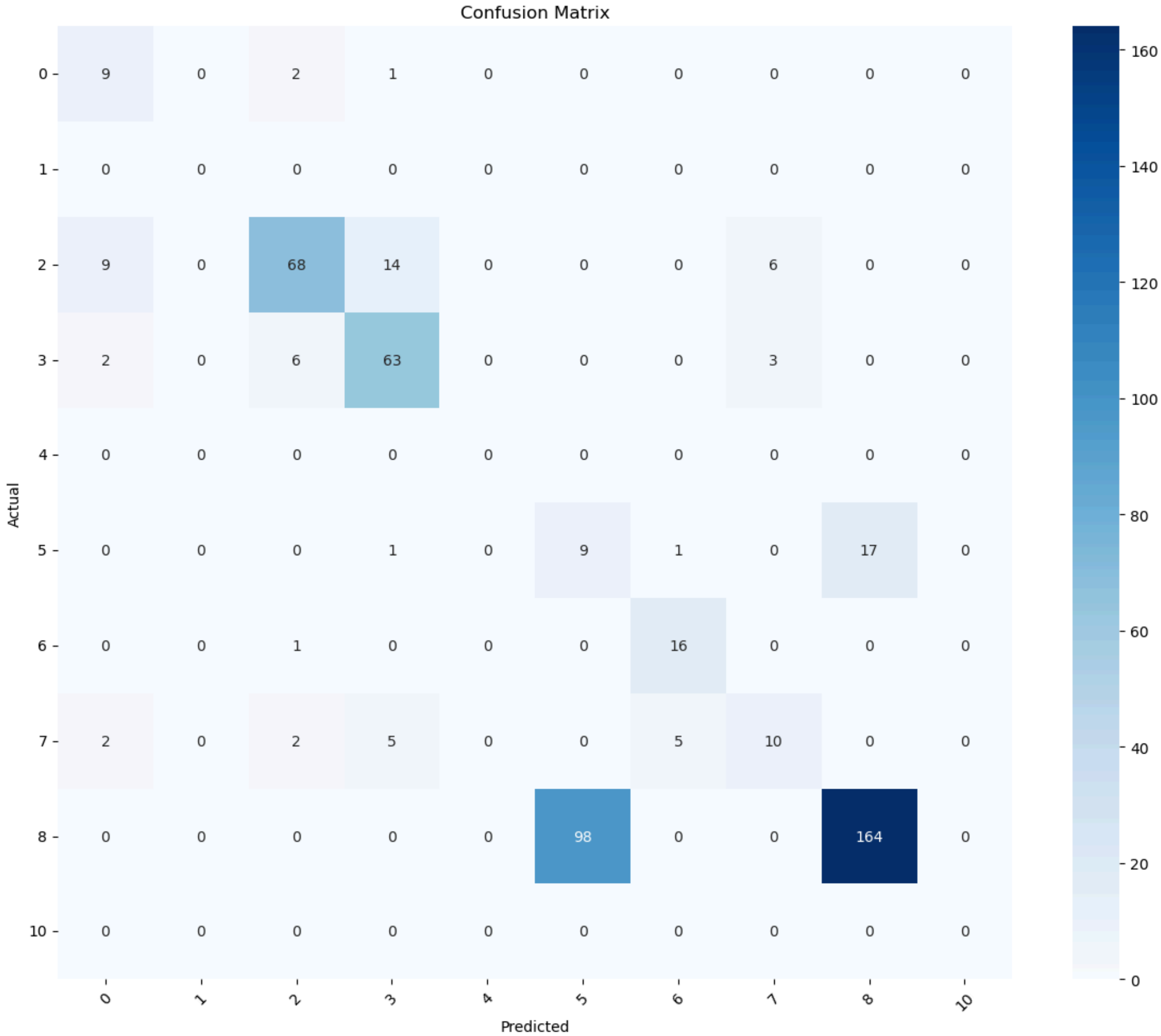
In [36]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Load the data
data_path = '/home/e12319879/shared/188.995-2024W/data/data_processed.pickle'
data = pd.read_pickle(data_path)

# Define features and target
```

```
features = ['temp_dailyMean', 'hum_dailyMin', 'hum_dailyMax', 'hum_dailyMean', 'bus
target_column = 'disruption'

# Encode the target labels
label_encoder = LabelEncoder()
data[target_column] = label_encoder.fit_transform(data[target_column])

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(data[features], data[target_col

# Set your model with class weighting
model = make_pipeline(
    StandardScaler(),
    RandomForestClassifier(random_state=42, criterion="gini", class_weight='balance
)

# Train the model
model.fit(X_train, y_train)

# Predict on the test data
predictions = model.predict(X_test)

# Get unique classes from the test data and predictions
unique_classes = np.unique(np.concatenate((y_test, predictions)))
target_names = label_encoder.inverse_transform(unique_classes)

# Select key classes for visualization
# You can modify this list based on your data insights
key_classes = unique_classes[:10]  # Adjust the number of classes as needed
key_class_names = label_encoder.inverse_transform(key_classes)

# First plot: Classification Report
def plot_classification_report(y_true, y_pred, labels):
    report = classification_report(y_true, y_pred, labels=labels, target_names=labe
    df_report = pd.DataFrame(report).transpose()

    plt.figure(figsize=(12, 8))
    sns.heatmap(df_report.iloc[:-1, :-1], annot=True, cmap='Blues', fmt='.2f')
    plt.title('Classification Report')
    plt.xticks(rotation=45)
    plt.yticks(rotation=0)
    plt.tight_layout()
    plt.show()

# Second plot: Confusion Matrix
def plot_confusion_matrix(y_true, y_pred, labels):
    cm = confusion_matrix(y_true, y_pred, labels=labels)

    plt.figure(figsize=(12, 10))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklab
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.xticks(rotation=45)
    plt.yticks(rotation=0)
    plt.tight_layout()
```

```
    plt.show()

# Implement the visualization for key classes
plot_classification_report(y_test, predictions, key_classes)
plot_confusion_matrix(y_test, predictions, key_classes)
```




Confusion Matrix
Actual
Predicted
0
1
2
3
4
5
6
7
8
10
9 0 2 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
9 0 68 14 0 0 0 6 0 0
2 0 6 63 0 0 0 3 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 9 1 0 17 0
0 0 1 0 0 0 16 0 0 0
2 0 2 5 0 0 5 10 0 0
0 0 0 0 0 98 0 0 164 0
0 0 0 0 0 0 0 0 0 0
160
140
120
100
80
60
40
20
0