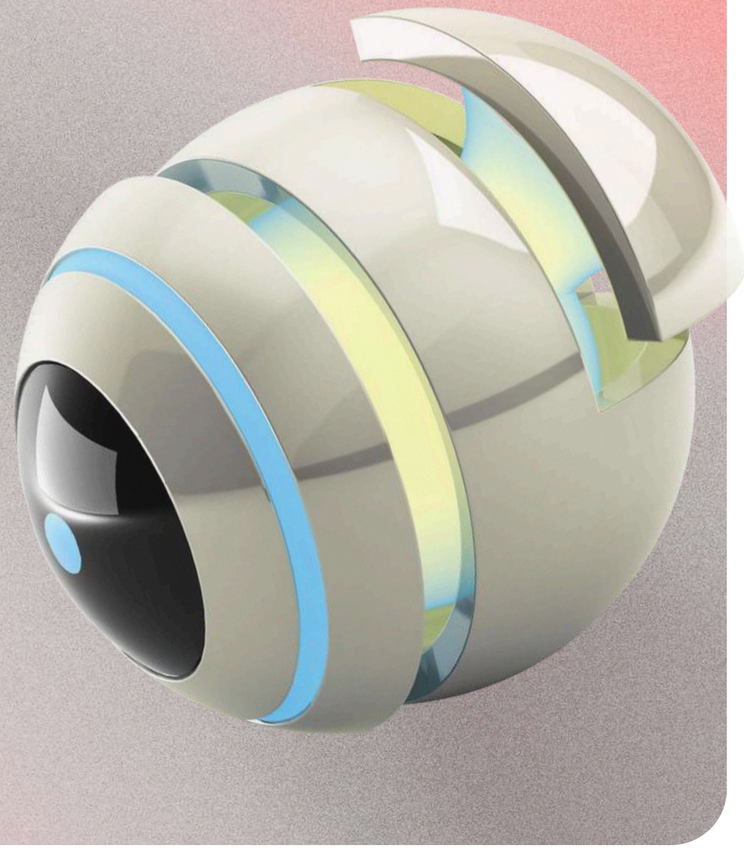
Climate and Floods in Jakarta



Dataset Link:

Climates and Floods in Jakarta, 2016 - 2020.

Check it! →

Our Members

Benediktus Arthur S.
27022555911

Hendy Cahyadi T.
2702290135

Stefanus Marcellino
2702215284

Joseph B.
2702261712

Jordi A. Iskandar
2702324631

Geoffrey D. Julianto
2702255773

Latar Belakang

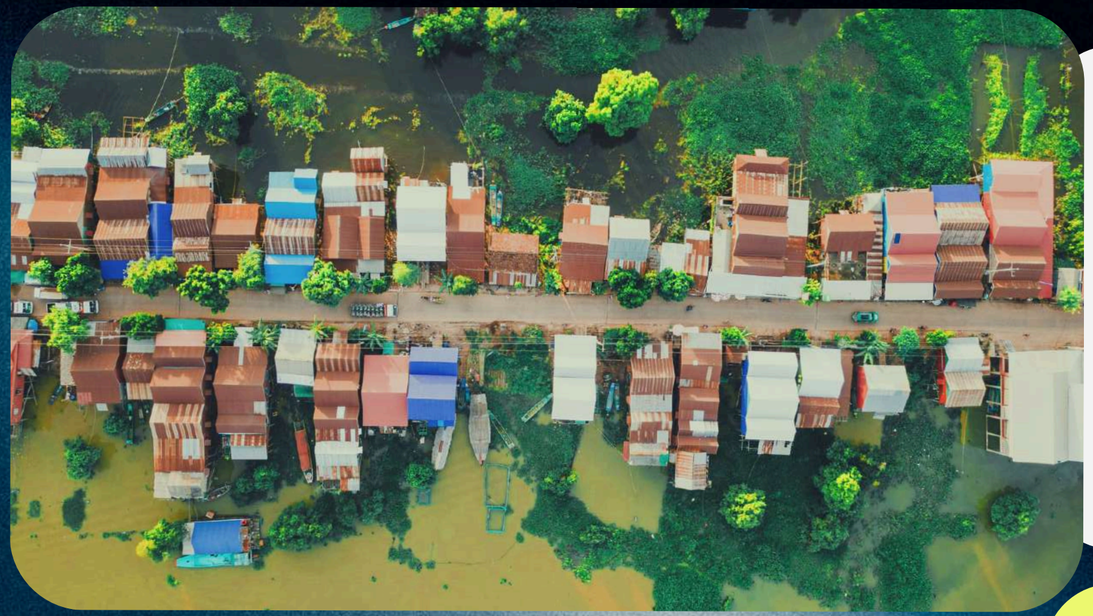
Jakarta merupakan salah satu kota di Indonesia yang sering mengalami bencana banjir, terutama saat musim hujan. Banjir menyebabkan kerugian besar, baik dari segi ekonomi, infrastruktur, maupun keselamatan masyarakat. Tingginya intensitas hujan, buruknya sistem drainase, serta urbanisasi yang cepat membuat banjir menjadi masalah yang kompleks dan berulang setiap tahun. Oleh karena itu, diperlukan sistem prediksi banjir yang dapat membantu pemerintah dan masyarakat dalam mengambil tindakan preventif secara lebih cepat dan tepat. Dalam proyek ini, kami membangun model prediksi banjir menggunakan data cuaca dan data stasiun pemantau untuk memperkirakan kemungkinan terjadinya banjir di wilayah Jakarta.

Dataset Features

Tn	min temperature (°C)
Tx	max temperature (°C)
Tavg	avg temperature (°C)
RH_avg	avg humidity(%)
RR	rainfall (mm)
ss	duration of sunshine(hour)
ff_x	max wind speed (m/s)
ddd_x	wind direction at maximum speed (°)

Dataset Features

`ff_avg`	max wind speed (m/s)
`ddd_car`	most wind direction (°)
`station_id`	station id which record the data
`station_name`	station name which record the data
`region_name`	location of the station
`flood`	1 means True and 0 means false



Methodology

Pre-Processing

Dataset yang didapat dari kaggle akan ditelusuri untuk menghapus data kosong, dan juga mengurangi outlier melalui metode IQR.

Exploratory Analysis

Dataset yang sudah bersih kemudian akan di-explore untuk mendapatkan knowledge baru yang berguna.

Methodology Cont.

Logistic Regression

Setelah itu, dataset bersih akan digunakan untuk training model prediktif. Data akan ditambahkan melalui SMOTE untuk mengatasi imbalance.

Random Forest

Selain Logistic Regression, algoritma Random Forest juga akan diterapkan untuk membandingkan akurasi model.

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer  # Still needed for
ColumnTransformer
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

warnings.filterwarnings('ignore')

def load_and_examine_data(file_path):
    df = pd.read_csv(file_path)
    print("Dataset shape:", df.shape)
    print("\nFirst 5 rows:")
    print(df.head())
    print("\nData types:")
    print(df.dtypes)
    print("\nMissing values per column:")
    print(df.isnull().sum()[df.isnull().sum() > 0])
    print("\nFlood event distribution:")
    print(df['flood'].value_counts(normalize=True) * 100)
    return df

def remove_missing_rows(df):
    print("\nRemoving rows with missing values...")
    before = df.shape[0]
    df = df.dropna()
    after = df.shape[0]
    print(f"Removed {before - after} rows. New shape: {df.shape}")
    return df

def detect_and_handle_outliers(df, columns_to_check=None):
    if columns_to_check is None:
        columns_to_check = [col for col in ['Tn', 'Tx', 'Tavg',
'RH_avg', 'RR', 'ss', 'ff_x', 'ff_avg']
                            if col in df.columns and
pd.api.types.is_numeric_dtype(df[col])]
    print("\nOutlier detection and handling:")
    for col in columns_to_check:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)]
        print(f"{col}: {len(outliers)} outliers capped")
        df[col] = np.where(df[col] < lower_bound, lower_bound, df[col])
        df[col] = np.where(df[col] > upper_bound, upper_bound, df[col])
    return df
```

```python
def prepare_features_and_target(df):
    numerical_features = [col for col in ['Tn', 'Tx', 'Tavg', 'RH_avg',
'RR', 'ss', 'ff_x', 'ff_avg']
                          if col in df.columns and
pd.api.types.is_numeric_dtype(df[col])]

    potential_categorical = ['station_id', 'station_name',
'region_name', 'ddd_x', 'ddd_car']
    categorical_features = [col for col in potential_categorical if col
in df.columns]

    target = 'flood'
    X = df[numerical_features + categorical_features]
    y = df[target]

    numeric_transformer = Pipeline(steps=[
        ('scaler', StandardScaler())
    ])

    if categorical_features:
        categorical_transformer = Pipeline(steps=[
            ('encoder', OneHotEncoder(handle_unknown='ignore',
sparse_output=False))
        ])

        preprocessor = ColumnTransformer(transformers=[
            ('num', numeric_transformer, numerical_features),
            ('cat', categorical_transformer, categorical_features)
        ])
    else:
        preprocessor = ColumnTransformer(transformers=[
            ('num', numeric_transformer, numerical_features)
        ])

    X_processed = preprocessor.fit_transform(X)

    if categorical_features:
        try:
            categorical_feature_names = preprocessor.transformers_[1]
[1].named_steps['encoder'].get_feature_names_out(categorical_features).t
olist()
            feature_names = numerical_features +
categorical_feature_names
        except:
            feature_names = numerical_features + [f'cat_{i}' for i in
range(X_processed.shape[1] - len(numerical_features))]
    else:
        feature_names = numerical_features

    print(f"\nFinal preprocessed shape: {X_processed.shape}")
    print(f"Number of features: {len(feature_names)}")
    return X_processed, y, preprocessor, feature_names
```

```python
def preprocess_data(file_path):
    print("Starting data preprocessing pipeline...")
    df = load_and_examine_data(file_path)
    df = remove_missing_rows(df)

    numeric_cols = [col for col in ['Tn', 'Tx', 'Tavg', 'RH_avg', 'RR',
'ss', 'ff_x', 'ff_avg']
                    if col in df.columns and
pd.api.types.is_numeric_dtype(df[col])]

    df = detect_and_handle_outliers(df, numeric_cols)

    try:
        X_processed, y, preprocessor, feature_names =
prepare_features_and_target(df)
        print("Preprocessing complete!")
        return X_processed, y, df, preprocessor, feature_names
    except Exception as e:
        print(f"Error during preprocessing: {str(e)}")
        return None, None, df, None, None

if __name__ == "__main__":
    file_path = "data_finish.csv"  # Replace with your actual file path
    X_processed, y, df_processed, preprocessor, feature_names =
preprocess_data(file_path)

    if X_processed is not None:
        print("Successfully preprocessed the data!")
        pd.DataFrame(X_processed,
columns=feature_names).to_csv("processed_features.csv", index=False)
        df_processed.to_csv("processed_full_data.csv", index=False)
    else:
        print("Preprocessing encountered errors.")
```

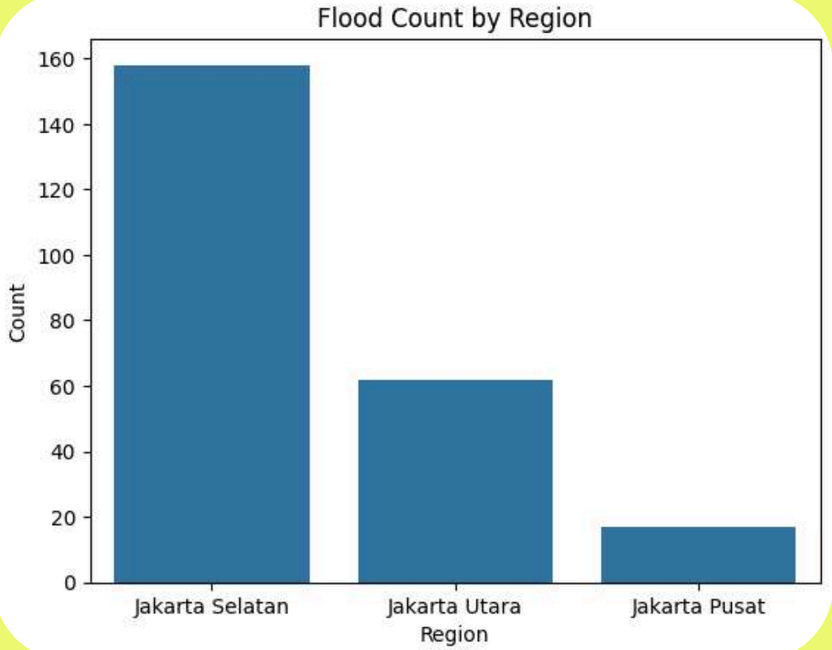
Data Pre-Processing

Null dan duplicate di hapus, outlier di cap.

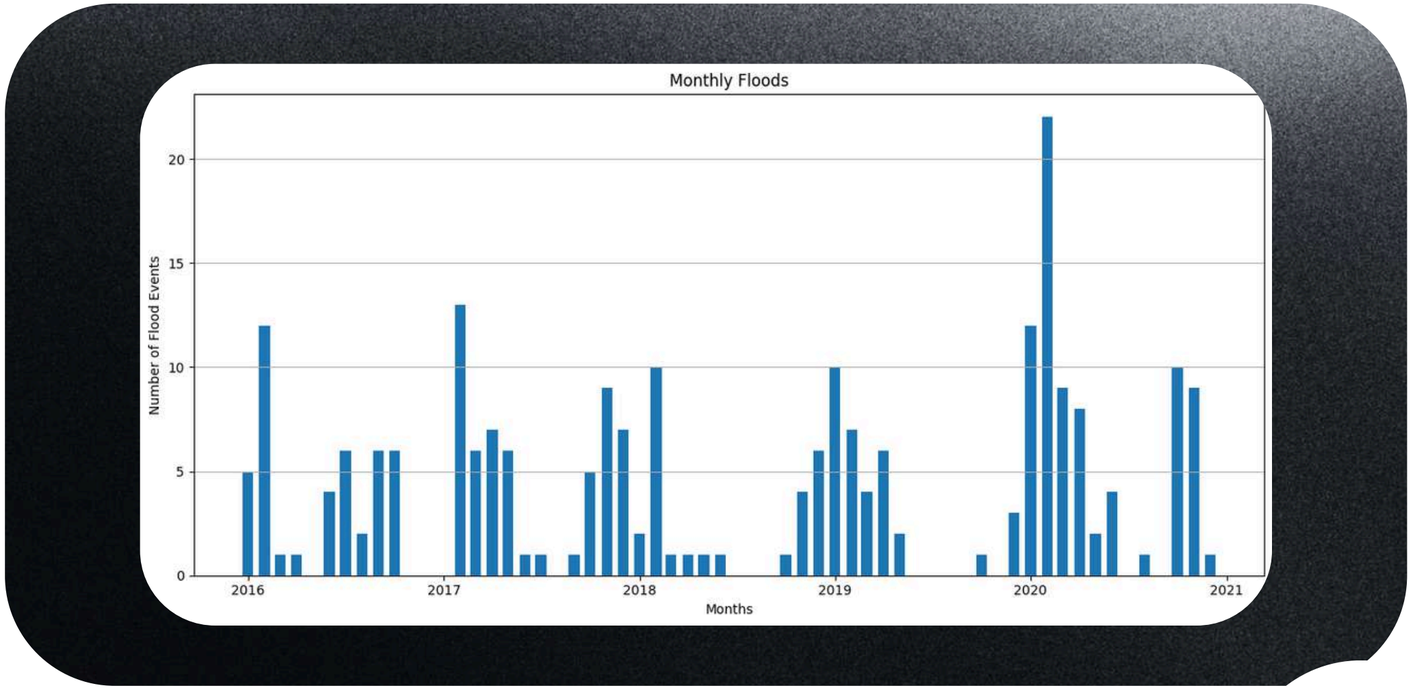
```
sns.countplot(x='region_name', data=df_processed[df_processed['flood']
== 1])
plt.title("Flood Count by Region")
plt.xlabel("Region")
plt.ylabel("Count")
plt.xticks(rotation=0)
plt.show()
```

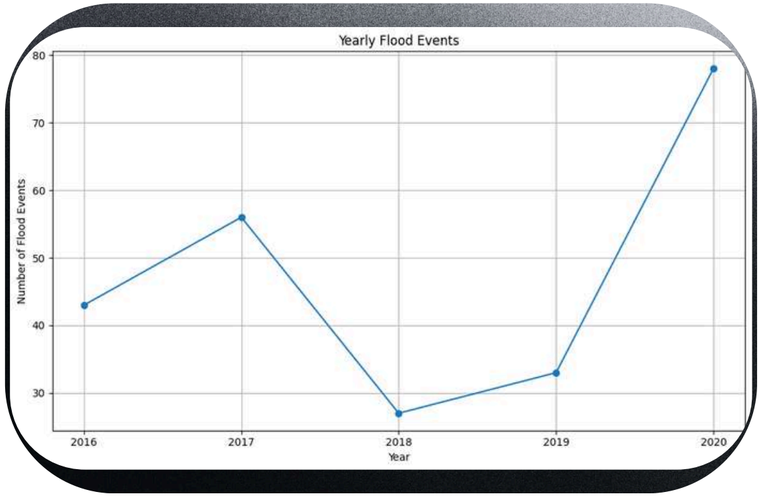
Exploratory Analysis 1

Visualisasi mengenai jumlah kebanjiran jika dipisahkan sesuai dengan daerah terjadinya.



```python
df_processed['date'] = pd.to_datetime(df_processed['date'])
flood_df = df_processed[df_processed['flood'] == 1]


monthly_floods =
flood_df.groupby(flood_df['date'].dt.to_period('M')).size()
monthly_floods.index = monthly_floods.index.to_timestamp()  # Convert
Period to Timestamp


import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
plt.bar(monthly_floods.index, monthly_floods.values, width=20)  # width
controls bar thickness
plt.title('Monthly Floods')
plt.xlabel('Months')
plt.ylabel('Number of Flood Events')
plt.grid(axis='y')
plt.tight_layout()
plt.show()

yearly_floods = flood_df.groupby(flood_df['date'].dt.year).size()

# Plot line chart
plt.figure(figsize=(10, 6))
plt.plot(yearly_floods.index.values, yearly_floods.values, marker='o',
linestyle='-')
plt.title('Yearly Flood Events')
plt.xlabel('Year')
plt.ylabel('Number of Flood Events')
plt.xticks(yearly_floods.index.values)  # Make sure only year integers
appear
plt.grid(True)
plt.tight_layout()
plt.show()
```





Exploratory 2

Peristiwa banjir bulanan dan tahunan.

Why Logistic Regression?

Cocok untuk prediksi biner, klasifikasi banjir adalah 1 (banjir), dan 0 (tidak banjir).

Sederhana dan mudah dipahami.

Efisien dan cepat trainingnya.

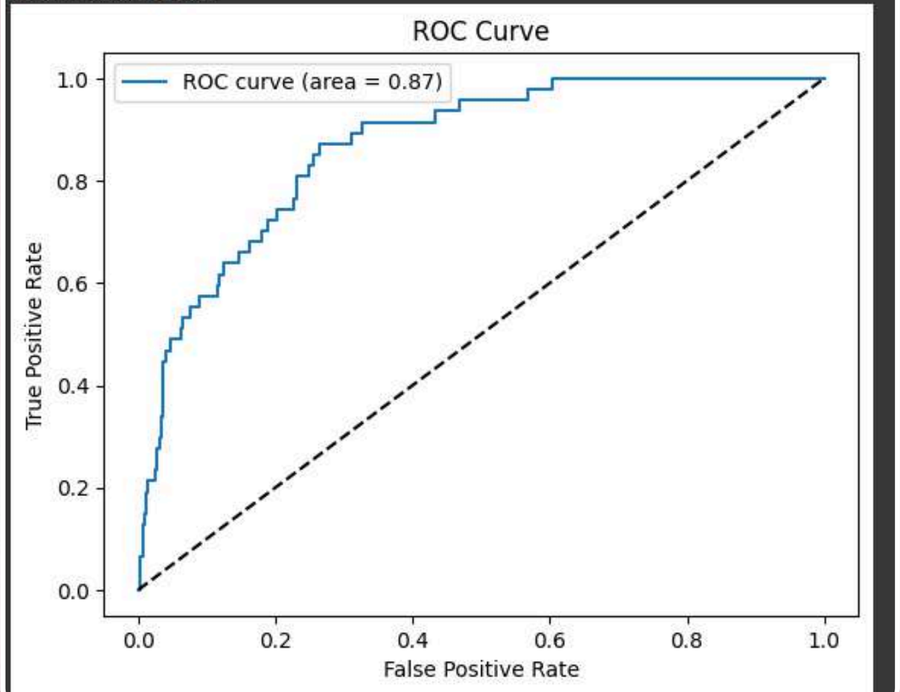
Performa baik pada data yang terstruktur.

Cocok dengan teknik SMOTE.

Logistic Regression

```python
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score

# Assuming df_processed is your DataFrame and 'flood' is the target colum
X = df_processed.drop(columns=['flood', 'ddd_car', 'station_name', 'station_id', 'date', 'region_name'])
# drop non-numeric or unwanted columns
y = df_processed['flood']

# Split into train and test sets first
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Initialize SMOTE
smote = SMOTE(random_state=42)

# Apply SMOTE to training data only
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

# Train logistic regression on the balanced data
model = LogisticRegression(max_iter=1000, random_state=42)
model.fit(X_train_smote, y_train_smote)

# Predict on the original test set
y_pred = model.predict(X_test)
y_pred_proba = model.predict_proba(X_test)[:, 1]
roc_auc = roc_auc_score(y_test, y_pred_proba)

# Evaluate
print("Classification Report:")
print(classification_report(y_test, y_pred))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("ROC AUC Score:")
print(roc_auc_score(y_test, model.predict_proba(X_test)[:, 1]))

import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
plt.plot(fpr, tpr, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.77      0.86       581
           1       0.22      0.81      0.35        47

    accuracy                           0.77       628
   macro avg       0.60      0.79      0.60       628
weighted avg       0.92      0.77      0.82       628

Confusion Matrix:
[[447 134]
 [  9  38]]
ROC AUC Score:
0.8676163621049547
```



Hasil Logistic Regression

```
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.77      0.86       581
           1       0.22      0.81      0.35        47

    accuracy                           0.77       628
   macro avg       0.60      0.79      0.60       628
weighted avg       0.92      0.77      0.82       628

Confusion Matrix:
[[447 134]
 [  9  38]]
ROC AUC Score:
0.8676163621049547
```

Accuracy: 0.77

Memprediksi dengan benar sekitar 77% total data.

Precision (0): 0.98 - 0.77

Hampir semua prediksi “tidak banjir” benar, dan dari semua kejadian “tidak banjir”, 77% berhasil dikenali model.

Precision (1): 0.22 - 0.81

Dari semua yang diprediksi “banjir”, hanya 22% yang banjir sebenarnya. Model berhasil mengenali 81% dari semua kejadian banjir, ini hasil yang baik.

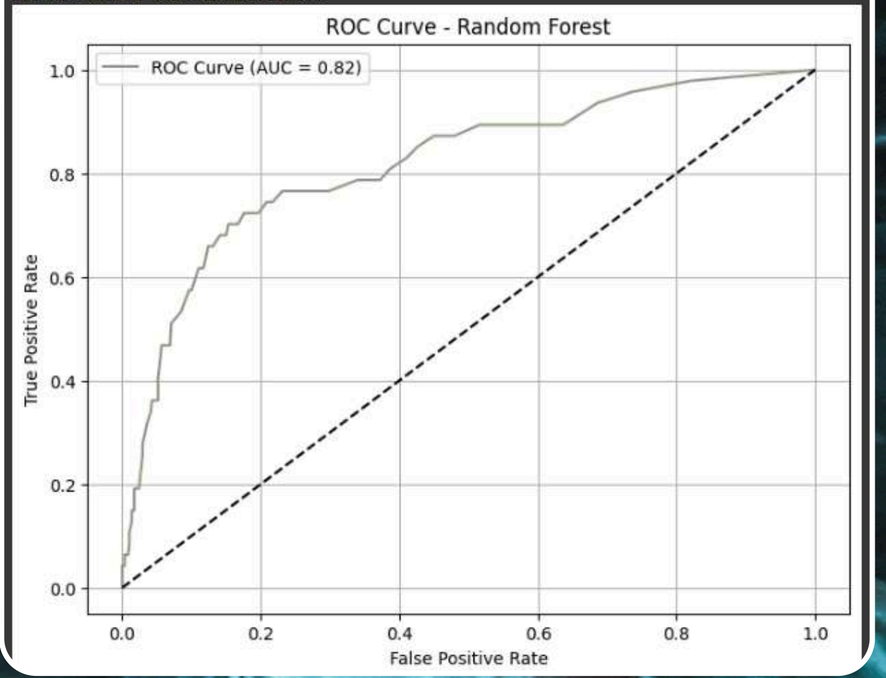
ROC AUC Score: 0.87

Model memiliki performa cukup baik dalam membedakan antara kelas banjir dan tidak banjir.

Random Forest

```python
from sklearn.ensemble import RandomForestClassifier

X = df_processed.drop(columns=['flood', 'ddd_car', 'station_name', 'station_id', 'date', 'region_name'])  # drop non-numeric or unwanted columns
y = df_processed['flood']

features = X.columns

# Data splitting and balancing
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2, random_state=42)
sm = SMOTE(random_state=42)
X_train_res, y_train_res = sm.fit_resample(X_train, y_train)

# Random Forest Classifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train_res, y_train_res)

# Evaluation
y_pred = model.predict(X_test)
y_pred_proba = model.predict_proba(X_test)[:, 1]

print("\nClassification Report:")
print(classification_report(y_test, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("ROC AUC Score:", roc_auc_score(y_test, y_pred_proba))

# ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label='ROC Curve (AUC = %.2f)' % roc_auc_score(y_test, y_pred_proba))
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Random Forest')
plt.legend()
plt.grid(True)
plt.show()

# Feature Importance
feat_importances = pd.Series(model.feature_importances_, index=features).sort_values(ascending=False)
plt.figure(figsize=(10, 6))
sns.barplot(x=feat_importances, y=feat_importances.index)
plt.title('Feature Importances - Random Forest')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.tight_layout()
plt.show()
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.97      0.96       580
           1       0.42      0.30      0.35        47

    accuracy                           0.92       627
   macro avg       0.68      0.63      0.65       627
weighted avg       0.91      0.92      0.91       627

Confusion Matrix:
[[561  19]
 [ 33  14]]
ROC AUC Score: 0.8188371239911959
```



Hasil Random Forest

```
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.97      0.96       580
           1       0.42      0.30      0.35        47

    accuracy                           0.92       627
   macro avg       0.68      0.63      0.65       627
weighted avg       0.91      0.92      0.91       627

Confusion Matrix:
[[561  19]
 [ 33  14]]
ROC AUC Score: 0.8188371239911959
```

Accuracy: 0.92

Memprediksi dengan benar sekitar 92% total data.

Precision (0): 0.94 - 0.97

Hampir semua prediksi “tidak banjir” benar. Dari semua kejadian “tidak banjir”, 94% berhasil dikenali oleh model, dan model mengenali 97% dari semua data yang benar-benar tidak banjir.

Precision (1): 0.42 - 0.30

Dari semua yang diprediksi sebagai “banjir”, hanya 42% yang benar-benar banjir. Model berhasil mengenali 30% dari semua kejadian banjir yang sebenarnya, menunjukkan adanya kekurangan dalam mendeteksi kasus banjir.

ROC AUC Score: 0.82

Model memiliki kemampuan cukup baik dalam membedakan antara kelas banjir dan tidak banjir, meskipun lebih rendah dibandingkan Logistic Regression.

Conclusion

Jakarta semakin rentan terhadap banjir akibat perubahan iklim dan infrastruktur yang kurang memadai, sehingga diperlukan model prediktif yang akurat. Dalam penelitian ini, dua algoritma yang diuji adalah Logistic Regression dan Random Forest. Logistic Regression menunjukkan nilai ROC AUC sebesar 87% dan recall 81%, menjadikannya cocok untuk mendeteksi sebanyak mungkin kejadian banjir, meskipun menghasilkan banyak false positives. Di sisi lain, Random Forest memiliki akurasi keseluruhan 92% dan precision 42%, namun recall-nya hanya 30%, sehingga lebih cocok jika prioritasnya adalah mengurangi alarm palsu dan dapat mentoleransi terlewatnya beberapa kejadian banjir. Maka, pemilihan model tergantung pada prioritas: deteksi maksimal atau prediksi yang lebih spesifik.


Thank You.