```python
import pandas as pd
import numpy as np
from scipy.io import arff
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from xgboost import XGBClassifier
from scipy.spatial.distance import jensenshannon
from scipy.stats import wasserstein_distance
from tabulate import tabulate
import sys
import os

# Set environment variable to allow CPU for large dataset
os.environ["TABPFN_ALLOW_CPU_LARGE_DATASET"] = "1"

# Add local TabPFGen path
sys.path.insert(0, './src/tabpfngen_backup')
from tabpfgen import TabPFGen

# =======================
# Step 1: Load the Dataset
# =======================
file_path = r'C:\\Users\\Manthan Goyal\\Desktop\\Team-Project\\TabPFGen\\datasets\\chess.arff'
print("рџ"№ Loading Chess dataset...")
data, meta = arff.loadarff(file_path)
df = pd.DataFrame(data)

# Decode byte strings
df = df.applymap(lambda x: x.decode('utf-8') if isinstance(x, bytes) else x)

# Display dataset information
print(f"рџ"Џ Dataset shape: {df.shape}")

# Encode features and target
X = df.drop(columns=['class']).applymap(lambda x: float(x) if x.isdigit() else x)
y = pd.factorize(df['class'])[0]

# =======================
# Step 2: Cross-Validation and Evaluation
# =======================
kf = KFold(n_splits=2, shuffle=True, random_state=42)
classifiers = {
    'LR': LogisticRegression(max_iter=1000),
    'RF': RandomForestClassifier(),
    'XG BOOST': XGBClassifier(eval_metric='logloss'),
    'MLP': MLPClassifier(max_iter=500)
}

# Storage for results
results = []

# Perform 3 Repeats of 2-Fold Cross-Validation
for repeat in range(1, 4):
    for fold, (train_index, test_index) in enumerate(kf.split(X), 1):
        # Split the data
        train_X, test_X = X.iloc[train_index], X.iloc[test_index]
        train_y, test_y = y[train_index], y[test_index]

        # Generate Synthetic Data
        print(f"рџ"Ѓ Generating synthetic data for Repeat {repeat}, Fold {fold}...")
        generator = TabPFGen(n_sgld_steps=100)
        X_synth, y_synth = generator.generate_classification(
            train_X.to_numpy(),
            train_y,
            int(0.5 * len(train_X)) #50% of data
        )

        # Inject missing classes if needed
        missing_classes = set(np.unique(train_y)) - set(np.unique(y_synth))
        for cls in missing_classes:
            print(f"вљ пёЏ Injecting missing class {cls} directly from real data.")
            samples_to_inject = train_X[train_y == cls]
            labels_to_inject = train_y[train_y == cls]
            X_synth = np.vstack((X_synth, samples_to_inject.to_numpy()))
            y_synth = np.hstack((y_synth, labels_to_inject))

        # Evaluate each classifier
        for name, clf in classifiers.items():
            print(f"вњ… Evaluating {name} for R{repeat}-F{fold}...")
            clf.fit(train_X, train_y)
            real_acc = accuracy_score(test_y, clf.predict(test_X))
            results.append([repeat, fold, name, real_acc])

        # JSD and Wasserstein calculations
        real_dist = np.bincount(train_y, minlength=len(np.unique(train_y))) / len(train_y)
        synth_dist = np.bincount(y_synth, minlength=len(np.unique(train_y))) / len(y_synth)
        jsd_value = jensenshannon(real_dist, synth_dist) if len(real_dist) == len(synth_dist) else np.nan
        wd_value = wasserstein_distance(np.sort(train_y), np.sort(y_synth))

        # Store distance metrics
        results.append([repeat, fold, "JSD", jsd_value])
        results.append([repeat, fold, "WD", wd_value])

# =======================
# Step 3: Format the Output
# =======================
# Create a new DataFrame to match the required structure
models = ['LR', 'RF', 'MLP', 'XG BOOST', 'JSD', 'WD']
columns = ['R1-F1', 'R1-F2', 'R2-F1', 'R2-F2', 'R3-F1', 'R3-F2', 'AVERAGE']
output_df = pd.DataFrame(index=models, columns=columns)

# Fill the DataFrame with values
for repeat in range(1, 4):
    for fold in range(1, 3):
        col_name = f'R{repeat}-F{fold}'

        for model in models:
            value = [x[3] for x in results if x[0] == repeat and x[1] == fold and x[2] == model]
            if value:
                output_df.at[model, col_name] = value[0]

# Calculate the average for each row
output_df['AVERAGE'] = output_df.iloc[:, :-1].apply(pd.to_numeric, errors='coerce').mean(axis=1)

# =======================
# Step 4: Display in Terminal
# =======================
print("\nрџ"Љ Final Cross-Validation Summary for Chess Dataset:\n")
print(tabulate(output_df, headers='keys', tablefmt='grid'))
```