📦 `anomaly.py` – Anomaly Detection Module

This module provides functions to detect and remove outliers using two techniques:

🔍 Function: `remove_outliers(df, method="isolation_forest")`

def remove_outliers(df, method="isolation_forest"):

✅ Supported Methods:

1. **Isolation Forest** – A tree-based unsupervised learning method.
2. **Autoencoder** – A neural network that learns to reconstruct input data.

---

1. 🔧 Isolation Forest

```
if method == "isolation_forest":
    iso = IsolationForest()
    mask = iso.fit_predict(df) == 1
    return df[mask]
```

✅ What it does:

- Fits an `IsolationForest` model from `sklearn.ensemble` on the data.
- Each point is assigned a label: **1 (inlier)** or **-1 (outlier)**.
- Only rows with label **1** are retained.

📘 Why Isolation Forest?

- Efficient for high-dimensional data.
- Works well for time series anomalies in stock prices, volumes, etc.

---

2. 🔧 Autoencoder

elif method == "autoencoder":

    scaler = StandardScaler()

    df_scaled = scaler.fit_transform(df)

🔁 Step-by-step:

1. **Standardization**: Input features are normalized using `StandardScaler`.
2. **Model Construction**:

input_layer = Input(shape=(input_dim,))

encoded = Dense(64, activation='relu')(input_layer)

decoded = Dense(input_dim, activation='linear')(encoded)

autoencoder = Model(inputs=input_layer, outputs=decoded)

- A 1-layer encoder and decoder are used.
- Model learns to reconstruct the original data.

3. **Training**:

autoencoder.fit(df_scaled, df_scaled, epochs=10, batch_size=32, verbose=0)

4. **Anomaly Score**:

recon = autoencoder.predict(df_scaled)

loss = np.mean((df_scaled - recon)**2, axis=1)

- Mean squared error (MSE) is calculated for reconstruction.
- High MSE = potential anomaly.

5. **Filtering**:

return df[loss < np.percentile(loss, 95)]

- Top 5% highest MSE rows are considered anomalies and removed.

📦 `imputers.py` – Missing Value Imputation Module

This module offers intelligent ways to fill missing values.

🔍 Function: `smart_impute(df, method="knn")`

def smart_impute(df, method="knn"):

✅ Supported Methods:

1. **KNN (K-Nearest Neighbors)**
2. **XGBoost (Gradient Boosted Trees)**

---

1. 🔧 KNN Imputer

if method == "knn":

  imputer = KNNImputer()

  df[:] = imputer.fit_transform(df)

✅ What it does:

- Uses `sklearn.impute.KNNImputer`.
- For each missing value, finds similar rows (neighbors) and takes the average.

📘 Why KNN?

- Good for numeric datasets.
- Preserves multivariate relationships.

---

2. 🔧 XGBoost Imputation

```
elif method == "xgboost":
    for col in df.columns:
        if df[col].isnull().sum() > 0:
```

🔁 Step-by-step:

1. **Loop through each column with missing values**.
2. **Split** into:
    - **Train** = rows without missing values
    - **Test** = rows with missing values

```
train = df[df[col].notnull()]
test = df[df[col].isnull()]
```

3. **Train XGBoost model**:

```
model = xgb.XGBRegressor()
model.fit(train.drop(columns=[col]), train[col])
```

4. **Predict missing values**:

```
df.loc[df[col].isnull(), col] = model.predict(test.drop(columns=[col]))
```

📘 Why XGBoost?

- Learns patterns in data.
- Very powerful for financial datasets where trends and nonlinearities matter.

🔄 Summary: Module Interactions

Module	Used When...	Depends On
`anomaly.py`	Cleaning data from strange patterns	`scikit-learn`, `keras`
`imputers.py`	Fixing missing values smartly	`sklearn`, `xgboost`
