CODOR DataScience BootCamp 参加者向け特典②

『Mini Project 拡張テンプレート集』

「ポートフォリオを今日から構築しよう」

PARTICIPANT EXCLUSIVE　PORTFOLIO READY

はじめに（Introduction）

本ドキュメントは、CODOR DataScience BootCamp参加者が学習した内容を即座に実践に移し、就職・転職活動に直結するポートフォリオを効率的に構築するためのテンプレート集です。各プロジェクトは「課題設定」「データ処理」「モデル構築」「ビジネス適用」の一連の流れを網羅しています。

目的と使い方

- **目的：** 理論の実践への転換、および採用担当者にアピールできる高品質なポートフォリオ作成の支援。
- **使い方：** 下記のコードをベースに、独自のデータセットや仮説を組み込んで拡張してください。そのままコピーするのではなく、コード内のコメントを参考に自分の言葉で分析結果を語れるようになることが重要です。

収録プロジェクト一覧

プロジェクト名	難易度	推定完成時間	主要技術スタック
Project 1: 顧客購買行動分析	★☆☆	3-5時間	Python (Pandas, K-Means), SQL (Window関数)
Project 2: 売上予測モデリング	★★☆	5-8時間	Python (RandomForest, GradientBoosting), 特徴量エンジニアリング
Project 3: 不正検知・異常検知	★★★	8-12時間	Python (Isolation Forest), 不均衡データ処理

ポートフォリオ構築フロー

Step 1: テーマ選択 - 自身の興味や志望業界に合わせてプロジェクトを1つ選択します。

Step 2: コード実行＆カスタマイズ - サンプルコードを動かし、パラメータや特徴量を調整して精度向上を試みます。

Step 3: レポート作成 - 分析結果をテンプレートに従ってまとめ、ビジネス視点での考察を加えます。

Step 4: GitHub公開 - コードとレポート（README.md）をリポジトリにアップロードし、URLを履歴書に記載します。

Project 1：顧客購買行動分析（E-Commerce Customer Analysis）

1-1. プロジェクト概要テンプレート

- **背景・課題設定：** ECサイトにおけるマーケティング予算の最適化。一律の販促ではなく、顧客価値（LTV）に基づいたセグメンテーションを行いたい。
- **分析目的・仮説：** RFM分析とクラスタリングを用いて顧客をグループ化し、「優良顧客」「離反懸念顧客」などを特定する。優良顧客は頻度が高く直近も購入しているはずであ

る。

- **使用データセット：** UCI Machine Learning Repository (Online Retail Data) または KaggleのE-Commerce Data。
- **期待されるアウトプット：** 顧客セグメント別の特徴サマリー、マーケティング施策の提言。

1-2. Python サンプルコード

```
# ========================================
# Project 1: 顧客購買行動分析
# CODOR DataScience BootCamp - Template
# ========================================

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

# --- Step 1: データ読み込み ---
# df = pd.read_csv('your_ecommerce_data.csv')
# サンプルデータ生成
np.random.seed(42)
n = 500
df = pd.DataFrame({
    'customer_id': range(1, n+1),
    'recency': np.random.randint(1, 365, n),
    'frequency': np.random.randint(1, 50, n),
    'monetary': np.random.uniform(100, 50000, n).round(2)
})

# --- Step 2: RFM分析 ---
def rfm_score(df):
    df['R_score'] = pd.qcut(df['recency'], q=5, labels=[5,4,3,2,1]).astype(int)
    df['F_score'] = pd.qcut(df['frequency'].rank(method='first'), q=5, labels=
[1,2,3,4,5]).astype(int)
    df['M_score'] = pd.qcut(df['monetary'], q=5, labels=[1,2,3,4,5]).astype(int)
    df['RFM_total'] = df['R_score'] + df['F_score'] + df['M_score']
    return df

df = rfm_score(df)

# --- Step 3: クラスタリング（K-Means）---
features = ['recency', 'frequency', 'monetary']
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df[features])
```

```
kmeans = KMeans(n_clusters=4, random_state=42, n_init=10)
df['cluster'] = kmeans.fit_predict(X_scaled)

# --- Step 4: 可視化 ---
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# RFMスコア分布
axes[0].hist(df['RFM_total'], bins=20, color='steelblue', edgecolor='white')
axes[0].set_title('RFM Score Distribution')
axes[0].set_xlabel('RFM Total Score')
axes[0].set_ylabel('Count')

# クラスター散布図
scatter = axes[1].scatter(df['recency'], df['monetary'],
                          c=df['cluster'], cmap='viridis', alpha=0.6)
axes[1].set_title('Customer Clusters (Recency vs Monetary)')
axes[1].set_xlabel('Recency (days)')
axes[1].set_ylabel('Monetary (¥)')
plt.colorbar(scatter, ax=axes[1], label='Cluster')

plt.tight_layout()
plt.savefig('rfm_analysis.png', dpi=150, bbox_inches='tight')
plt.show()

# --- Step 5: クラスター特性サマリー ---
cluster_summary = df.groupby('cluster')[features].mean().round(2)
print("\n=== Cluster Summary ===")
print(cluster_summary)
```

1-3. SQL サンプルコード

```
-- ========================================
-- Project 1: 顧客購買行動分析 - SQL版
-- ========================================

-- RFM計算クエリ
WITH order_stats AS (
    SELECT
        customer_id,
        DATEDIFF(CURDATE(), MAX(order_date)) AS recency,
        COUNT(DISTINCT order_id)             AS frequency,
        SUM(order_amount)                    AS monetary
    FROM orders
    WHERE order_date >= DATE_SUB(CURDATE(), INTERVAL 1 YEAR)
    GROUP BY customer_id
),
rfm_scores AS (
```

```
    SELECT
        customer_id,
        recency,
        frequency,
        monetary,
        NTILE(5) OVER (ORDER BY recency DESC)   AS r_score,
        NTILE(5) OVER (ORDER BY frequency ASC)  AS f_score,
        NTILE(5) OVER (ORDER BY monetary ASC)   AS m_score
    FROM order_stats
),
rfm_segments AS (
    SELECT *,
        (r_score + f_score + m_score) AS rfm_total,
        CASE
            WHEN (r_score + f_score + m_score) >= 12 THEN 'Champions'
            WHEN (r_score + f_score + m_score) >= 9  THEN 'Loyal Customers'
            WHEN (r_score + f_score + m_score) >= 6  THEN 'Potential Loyalists'
            ELSE 'At Risk / Lost'
        END AS segment
    FROM rfm_scores
)
SELECT
    segment,
    COUNT(*)                    AS customer_count,
    ROUND(AVG(monetary), 0)     AS avg_monetary,
    ROUND(AVG(frequency), 1)    AS avg_frequency
FROM rfm_segments
GROUP BY segment
ORDER BY avg_monetary DESC;
```

1-4. KPI設定シート

KPI項目	定義	目標値	測定方法	更新頻度
顧客セグメント分布	各セグメント（優良、離反など）に属する顧客数の割合	優良顧客20%以上	SQL集計 / BIツール	月次
RFMスコア平均値	全顧客のRFM総合スコアの平均	前月比 +5%	Pythonスクリプト	月次
優良顧客割合	RFMスコアが上位20%に入る顧客の割合	20%維持	SQL集計	月次
クラスター内凝集度	クラスタリング結果のまとまり具合（Elbow method等で評価）	-	Python (Inertia)	モデル更新時

KPI項目	定義	目標値	測定方法	更新頻度
セグメント別購買単価	セグメントごとの平均購入金額	各セグメントで上昇	SQL / BIツール	週次

1-5. レポートフォーマット

レポート構成案

- **エグゼクティブサマリー：** 分析の結果、顧客は4つの主要グループに分類されました。「ロイヤル顧客」層は全体の20%ですが、売上の60%を占めています。
- **分析アプローチ説明：** 過去1年間の購買データを使用し、RFM分析とK-Means法によるクラスタリングを実施。
- **主要発見事項（Key Findings）：** 「直近購入なし・高頻度」の離反予備軍グループが存在することが判明。
- **ビジネス提言：** 離反予備軍に対し、カムバックキャンペーン（クーポン配布）を実施すべき。
- **次のアクション：** キャンペーン実施後のROI測定、および半年後のセグメント変化の追跡。

Project 2：売上予測モデリング（Sales Forecasting）

2-1. プロジェクト概要テンプレート

- **背景・課題設定：** 在庫切れによる機会損失と、過剰在庫による保管コストのトレードオフを解消したい。
- **予測対象・期間：** 特定商品カテゴリーの向こう30日間の日次売上予測。
- **使用データと特徴量エンジニアリング方針：** 過去の売上実績、曜日、祝日フラグ、気温、プロモーション有無を使用。ラグ特徴量（7日前、30日前）を作成して時系列情報をモデルに組み込む。

- **モデル選定理由：** 非線形な関係や交互作用（プロモーション×休日など）を捉えやすいRandom ForestおよびGradient Boostingを採用。

2-2. Python サンプルコード

```
# ========================================
# Project 2: 売上予測モデリング
# CODOR DataScience BootCamp - Template
# ========================================

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.preprocessing import LabelEncoder
import warnings
warnings.filterwarnings('ignore')

# --- Step 1: サンプルデータ生成 ---
np.random.seed(42)
dates = pd.date_range('2022-01-01', '2024-12-31', freq='D')
n = len(dates)

df = pd.DataFrame({
    'date': dates,
    'sales': (
        3000 +
        np.sin(np.arange(n) * 2 * np.pi / 365) * 500 +   # 季節性
        np.arange(n) * 0.5 +                               # トレンド
        np.random.normal(0, 200, n)                        # ノイズ
    ).round(0),
    'temperature': np.random.normal(20, 10, n).round(1),
    'is_holiday': np.random.choice([0, 1], n, p=[0.95, 0.05]),
    'promotion': np.random.choice([0, 1], n, p=[0.85, 0.15])
})

# --- Step 2: 特徴量エンジニアリング ---
df['year']        = df['date'].dt.year
df['month']       = df['date'].dt.month
df['day_of_week'] = df['date'].dt.dayofweek
df['quarter']     = df['date'].dt.quarter
df['is_weekend']  = (df['day_of_week'] >= 5).astype(int)
df['sales_lag7']  = df['sales'].shift(7)   # 7日前の売上
df['sales_lag30'] = df['sales'].shift(30)  # 30日前の売上
df['rolling_mean_7'] = df['sales'].rolling(7).mean()
df = df.dropna()
```

```
# --- Step 3: モデル学習 ---
features = ['year','month','day_of_week','quarter','is_weekend',
            'temperature','is_holiday','promotion',
            'sales_lag7','sales_lag30','rolling_mean_7']
X = df[features]
y = df['sales']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
shuffle=False)

models = {
    'RandomForest': RandomForestRegressor(n_estimators=100, random_state=42),
    'GradientBoosting': GradientBoostingRegressor(n_estimators=100,
random_state=42)
}

results = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    results[name] = {
        'MAE':  round(mean_absolute_error(y_test, y_pred), 2),
        'RMSE': round(np.sqrt(mean_squared_error(y_test, y_pred)), 2),
        'R2':   round(r2_score(y_test, y_pred), 4)
    }

# --- Step 4: 結果表示 ---
print("=== Model Comparison ===")
for name, metrics in results.items():
    print(f"{name}: MAE={metrics['MAE']}, RMSE={metrics['RMSE']}, R²=
{metrics['R2']}")

# --- Step 5: 特徴量重要度 ---
best_model = models['RandomForest']
importance_df = pd.DataFrame({
    'feature': features,
    'importance': best_model.feature_importances_
}).sort_values('importance', ascending=False)

plt.figure(figsize=(10, 6))
plt.barh(importance_df['feature'], importance_df['importance'],
color='steelblue')
plt.title('Feature Importance - Random Forest')
plt.xlabel('Importance')
plt.tight_layout()
plt.savefig('feature_importance.png', dpi=150, bbox_inches='tight')
plt.show()
```

2-3. SQL サンプルコード

```sql
-- ========================================
-- Project 2: 売上予測 - データ前処理SQL
-- ========================================

-- 日次売上集計 + 特徴量生成
WITH daily_sales AS (
    SELECT
        DATE(order_date)        AS sale_date,
        SUM(order_amount)       AS daily_sales,
        COUNT(DISTINCT order_id) AS order_count,
        COUNT(DISTINCT customer_id) AS unique_customers
    FROM orders
    GROUP BY DATE(order_date)
),
sales_with_features AS (
    SELECT
        sale_date,
        daily_sales,
        order_count,
        unique_customers,
        YEAR(sale_date)                         AS year,
        MONTH(sale_date)                        AS month,
        DAYOFWEEK(sale_date)                    AS day_of_week,
        QUARTER(sale_date)                      AS quarter,
        -- ラグ特徴量
        LAG(daily_sales, 7)  OVER (ORDER BY sale_date) AS sales_lag_7d,
        LAG(daily_sales, 30) OVER (ORDER BY sale_date) AS sales_lag_30d,
        -- 移動平均
        AVG(daily_sales) OVER (
            ORDER BY sale_date
            ROWS BETWEEN 6 PRECEDING AND CURRENT ROW
        ) AS rolling_avg_7d,
        -- 前年同日比
        LAG(daily_sales, 365) OVER (ORDER BY sale_date) AS same_day_last_year,
        ROUND(
            daily_sales / NULLIF(LAG(daily_sales, 365) OVER (ORDER BY sale_date),
0) - 1,
            4
        ) AS yoy_growth_rate
    FROM daily_sales
)
SELECT * FROM sales_with_features
WHERE sale_date >= '2023-01-01'
ORDER BY sale_date;
```

2-4. KPI設定シート

KPI項目	定義	目標値	測定方法	更新頻度
予測精度 (MAE)	実測値と予測値の平均絶対誤差	5000円以下	モデル評価スクリプト	モデル更新時
予測精度 (RMSE)	二乗平均平方根誤差（大きな外しを罰する）	7000円以下	モデル評価スクリプト	モデル更新時
決定係数 (R^2)	モデルの当てはまりの良さ	0.80以上	モデル評価スクリプト	モデル更新時
在庫削減率	予測導入前後の平均在庫量の変化	-10%	在庫管理システム	月次
欠品率	注文に対して在庫がなく出荷できなかった割合	1%未満	受注管理システム	週次

2-5. レポートフォーマット

レポート構成案

- **予測精度サマリー：** Random Forestモデルを採用し、テストデータにおいて R^2=0.85、MAE=4,200円を達成しました。
- **モデル比較結果：** 線形回帰と比較して、決定木ベースのモデルの方が季節性やプロモーション効果をよく捉えています。
- **主要特徴量と解釈：** 「7日前の売上（直近トレンド）」と「プロモーション有無」が最も予測に寄与していることが判明しました。
- **予測結果の事業活用提案：** 向こう1週間の予測値を基に、発注量を自動調整するシステムへの組み込みを推奨します。

Project 3：不正検知・異常検知システム（Fraud/Anomaly Detection）

3-1. プロジェクト概要テンプレート

- **背景・課題設定：** クレジットカード決済における不正利用が増加しており、チャージバック（払戻し）による損失を削減したい。
- **検知対象・アプローチ：** 正常な取引パターンから逸脱するデータを検出する「異常検知（Anomaly Detection）」アプローチを採用。Isolation Forestを使用。
- **使用データとラベリング方針：** 取引金額、時間帯、地理的距離などの特徴量を使用。過去の確定不正データを検証用に使用するが、学習は教師なし学習（正常データが圧倒的多数という前提）で行う。
- **モデル選定と評価指標：** Isolation Forest。評価にはPrecision（適合率）、Recall（再現率）、F1-Scoreを用いる。特に見逃しを防ぐためRecallを重視する。

3-2. Python サンプルコード

```
# ========================================
# Project 3: 異常検知システム
# CODOR DataScience BootCamp - Template
# ========================================

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

# --- Step 1: サンプルデータ生成 ---
np.random.seed(42)
n_normal = 950
n_fraud  = 50

normal_data = pd.DataFrame({
    'transaction_amount': np.random.normal(5000, 2000, n_normal).clip(100),
    'hour_of_day':        np.random.randint(8, 22, n_normal),
    'transactions_today': np.random.poisson(3, n_normal),
    'distance_from_home': np.random.exponential(10, n_normal),
    'is_fraud': 0
})

fraud_data = pd.DataFrame({
    'transaction_amount': np.random.normal(80000, 30000, n_fraud).clip(10000),
    'hour_of_day':        np.random.choice([1, 2, 3, 23, 0], n_fraud),
    'transactions_today': np.random.poisson(15, n_fraud),
    'distance_from_home': np.random.exponential(200, n_fraud),
    'is_fraud': 1
})
```

```
df = pd.concat([normal_data, fraud_data], ignore_index=True)

# --- Step 2: 前処理 ---
features = ['transaction_amount', 'hour_of_day',
            'transactions_today', 'distance_from_home']
scaler = StandardScaler()
X = scaler.fit_transform(df[features])
y_true = df['is_fraud']

# --- Step 3: Isolation Forest ---
clf = IsolationForest(contamination=0.05, random_state=42, n_estimators=200)
df['anomaly_score'] = clf.fit_predict(X)
df['is_predicted_fraud'] = (df['anomaly_score'] == -1).astype(int)

# --- Step 4: 評価 ---
print("=== Classification Report ===")
print(classification_report(y_true, df['is_predicted_fraud'],
                            target_names=['Normal', 'Fraud']))

# --- Step 5: 混同行列の可視化 ---
cm = confusion_matrix(y_true, df['is_predicted_fraud'])
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Normal','Fraud'],
            yticklabels=['Normal','Fraud'])
plt.title('Confusion Matrix - Anomaly Detection')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.tight_layout()
plt.savefig('confusion_matrix.png', dpi=150, bbox_inches='tight')
plt.show()

# --- Step 6: 異常スコア分布 ---
fig, ax = plt.subplots(figsize=(10, 5))
ax.scatter(df[df['is_fraud']==0]['transaction_amount'],
           df[df['is_fraud']==0]['distance_from_home'],
           alpha=0.4, label='Normal', color='steelblue', s=30)
ax.scatter(df[df['is_fraud']==1]['transaction_amount'],
           df[df['is_fraud']==1]['distance_from_home'],
           alpha=0.8, label='Fraud', color='crimson', s=60, marker='x')
ax.set_xlabel('Transaction Amount (¥)')
ax.set_ylabel('Distance from Home (km)')
ax.set_title('Transaction Pattern: Normal vs Fraud')
ax.legend()
plt.tight_layout()
plt.savefig('fraud_scatter.png', dpi=150, bbox_inches='tight')
plt.show()
```

3-3. SQL サンプルコード

```sql
-- ========================================
-- Project 3: 不正検知 - ルールベース検知SQL
-- ========================================

-- 疑わしいトランザクションの検出
WITH transaction_stats AS (
    SELECT
        customer_id,
        AVG(amount)                  AS avg_amount,
        STDDEV(amount)               AS std_amount,
        AVG(amount) + 3*STDDEV(amount) AS upper_threshold
    FROM transactions
    WHERE transaction_date >= DATE_SUB(CURDATE(), INTERVAL 90 DAY)
    GROUP BY customer_id
),
daily_tx_count AS (
    SELECT
        customer_id,
        DATE(transaction_date) AS tx_date,
        COUNT(*)               AS tx_count_today,
        SUM(amount)            AS total_amount_today
    FROM transactions
    GROUP BY customer_id, DATE(transaction_date)
),
fraud_flags AS (
    SELECT
        t.transaction_id,
        t.customer_id,
        t.amount,
        t.transaction_date,
        -- フラグ1: 平均+3σを超える高額取引
        CASE WHEN t.amount > ts.upper_threshold THEN 1 ELSE 0 END AS
flag_large_amount,
        -- フラグ2: 深夜取引（0-5時）
        CASE WHEN HOUR(t.transaction_date) BETWEEN 0 AND 5 THEN 1 ELSE 0 END AS
flag_late_night,
        -- フラグ3: 当日取引回数が異常に多い
        CASE WHEN dtx.tx_count_today > 10 THEN 1 ELSE 0 END AS
flag_high_frequency,
        (
            CASE WHEN t.amount > ts.upper_threshold THEN 1 ELSE 0 END +
            CASE WHEN HOUR(t.transaction_date) BETWEEN 0 AND 5 THEN 1 ELSE 0 END
+
            CASE WHEN dtx.tx_count_today > 10 THEN 1 ELSE 0 END
        ) AS total_flags
    FROM transactions t
    LEFT JOIN transaction_stats ts ON t.customer_id = ts.customer_id
    LEFT JOIN daily_tx_count dtx
           ON t.customer_id = dtx.customer_id
          AND DATE(t.transaction_date) = dtx.tx_date
    WHERE t.transaction_date >= CURDATE() - INTERVAL 7 DAY
```

```
)
SELECT *,
    CASE
        WHEN total_flags >= 2 THEN 'HIGH RISK'
        WHEN total_flags = 1  THEN 'MEDIUM RISK'
        ELSE 'LOW RISK'
    END AS risk_level
FROM fraud_flags
WHERE total_flags >= 1
ORDER BY total_flags DESC, amount DESC;
```

3-4. KPI設定シート

KPI項目	定義	目標値	測定方法	更新頻度
検知精度 (Precision)	検知した不正のうち、本当に不正だった割合	80%以上	実績突合	週次
再現率 (Recall)	実際の不正のうち、検知できた割合	90%以上	実績突合	週次
誤検知率 (FPR)	正常取引を誤って不正と判定した割合	1%未満	実績突合	週次
不正損失削減額	検知により阻止できた不正取引の総額	前月比増	会計システム	月次
アラート対応時間	検知アラート発生からオペレーター確認までの時間	15分以内	運用ログ	日次

3-5. レポートフォーマット

レポート構成案

- **検知システム概要：** 「深夜帯の高額決済」や「短時間の連続取引」といった異常パターンをIsolation Forestで自動検知するシステムを構築。
- **精度評価結果：** シミュレーションデータにおいて、Recall 92%、Precision 78%を達成。
- **検知されたパターンと特徴：** 検知された不正取引の多くは、自宅から長距離離れた場所での高額決済でした。

- **運用への適用方法：** リスクレベル「High」の取引は自動保留し、「Medium」はオペレーターによる事後確認を行うフローを提案します。

PPTスライド構成テンプレート（全プロジェクト共通）

1. **タイトルスライド：** プロジェクト名、実施者、日付、所属
2. **背景・課題：** ビジネス課題、データの状況、なぜこの分析が必要か（Why）
3. **データ概要：** データソース、サンプル数、変数説明、データ期間
4. **分析アプローチ：** 手法選定理由、前処理内容、モデル選択（How）
5. **主要結果：** グラフ・チャート、数値結果、統計的検定結果（What）
6. **インサイト・考察：** 発見事項、予想外の結果、ビジネスへの示唆（So What?）
7. **ビジネス提言：** 具体的なアクション、期待効果、実装優先度
8. **まとめ・次のアクション：** 要約、今後の展開、追加調査項目

総合KPI設定シート（マスターテンプレート）

モデル精度KPI

項目	説明
精度指標	Accuracy, Precision, Recall, F1, AUC-ROC, RMSE, MAE, R2
閾値	モデルが実運用に耐えうる最低ライン
評価方法	Cross-validation, Hold-out検証, A/Bテスト

ビジネスインパクトKPI

項目	説明
財務インパクト	売上増加額、コスト削減額、LTV向上率
顧客指標	CS（顧客満足度）、NPS、解約率（Churn Rate）

業務効率	工数削減時間、自動化率、処理速度

プロジェクト進捗KPI

項目	説明
進捗率	タスク完了数 / 全タスク数
期限遵守率	マイルストーン通りに進んでいるか

データ品質KPI

項目	説明
完全性	欠損値の割合（Missing Rate）
正確性	異常値（Outliers）の割合、フォーマット整合性
鮮度	データの更新頻度、ラグタイム

ポートフォリオ構築チェックリスト

- ☐ GitHubリポジトリを作成し、適切な名前をつけたか（例: `ec-customer-analysis` ）
- ☐ `README.md` に以下の内容を含めたか
  - ☐ プロジェクトの背景と目的
  - ☐ 使用した技術スタック（Python, Libraries, SQL）
  - ☐ 分析フローと主要な結果（可視化画像を含む）
  - ☐ セットアップと実行方法
- ☐ コードは整理され、適切なコメント（Docstring）が記述されているか
- ☐ `requirements.txt` または環境定義ファイルを含めたか
- ☐ 可視化画像（.png）は `images/` フォルダ等に保存され、READMEから参照されているか
- ☐ 機密情報（APIキーや個人情報）が含まれていないか確認したか
- ☐ ライセンス（MIT License等）と免責事項を記載したか

免責事項：本テンプレートはCODOR DataScience BootCamp参加者限定の特典です。

商用利用の際はデータの取り扱いに十分ご注意ください。

最終更新日：2026年3月版