Test EDA / Data Wrangling - Test 2

Sujet: Exploratory Data Analysis et Data Wrangling
Niveau: Intermédiaire
Nombre de questions: 25

Questions et Réponses

Q1. Quelle est la différence entre df.loc[] et df.iloc[]?

R1.

```
# loc: sélection par LABEL (nom)
df.loc[0:5, 'montant']  # Lignes 0 à 5 inclus, colonne 'montant'
df.loc[df['age'] > 30, ['nom', 'age']]  # Filtrage + colonnes

# iloc: sélection par POSITION (index numérique)
df.iloc[0:5, 0]  # Lignes 0 à 4 (5 exclu), colonne index 0
df.iloc[:, 2:5]  # Toutes lignes, colonnes 2, 3, 4
```

Règle: loc = labels, iloc = integers

Q2. Comment identifier et visualiser les patterns de valeurs manquantes?

R2.

```
import missingno as msno

# Matrice de nullité
msno.matrix(df)

# Heatmap de corrélation des manquants
msno.heatmap(df)

# Bar chart
msno.bar(df)

# Pattern analysis
df.isnull().sum().sort_values(ascending=False)
```

Q3. Comment calculer le skewness et kurtosis et les interpréter?

R3.

```
from scipy.stats import skew, kurtosis

skewness = df['montant'].skew()
kurt = df['montant'].kurtosis()

print(f"Skewness: {skewness:.2f}")
print(f"Kurtosis: {kurt:.2f}")
```

Interprétation: - **Skewness:** - = 0: Symétrique - > 0: Asymétrie droite (queue à droite) - < 0: Asymétrie gauche

- **Kurtosis (excès):**
  - = 0: Normal
  - 0: Leptokurtique (queues lourdes)
  - < 0: Platykurtique (queues légères)

Q4. Comment appliquer une transformation log et quand l'utiliser?

R4.

```
import numpy as np

# Transformation log (pour valeurs > 0)
df['log_montant'] = np.log(df['montant'])

# log1p pour gérer les zéros
df['log_montant'] = np.log1p(df['montant'])

# Inverse
df['montant_original'] = np.expm1(df['log_montant'])
```

Quand utiliser: - Distribution très asymétrique (skew > 1) - Données couvrant plusieurs ordres de grandeur - Relation multiplicative (pas additive)

Q5. Comment utiliser apply() pour des transformations personnalisées?

R5.

```
# Sur une colonne
df['categorie_risque'] = df['score'].apply(lambda x: 'High' if x < 500 else 'Low')

# Sur plusieurs colonnes
df['DTI'] = df.apply(lambda row: row['dette'] / row['revenu'], axis=1)

# Fonction personnalisée
def categorize_age(age):
    if age < 30:
        return 'Jeune'
    elif age < 50:
        return 'Adulte'
    else:
        return 'Senior'

df['age_category'] = df['age'].apply(categorize_age)
```

Q6. Comment utiliser np.where() et np.select() pour créer des colonnes conditionnelles?

R6.

```
import numpy as np

# np.where: 2 conditions (if-else)
df['statut'] = np.where(df['solde'] > 0, 'Positif', 'Négatif')

# np.select: conditions multiples
```

```python
conditions = [
    df['score'] >= 750,
    df['score'] >= 650,
    df['score'] >= 500
]
choices = ['Excellent', 'Bon', 'Moyen']
df['rating'] = np.select(conditions, choices, default='Faible')
```

Q7. Comment créer des bins/catégories à partir de données continues?

R7.

```python
# Bins égaux (largeur)
df['montant_bin'] = pd.cut(df['montant'], bins=5, labels=['Très faible', 'Faible', 'Moyen', 'Élevé', 'T

# Bins personnalisés
bins = [0, 10000, 50000, 100000, float('inf')]
labels = ['Petit', 'Moyen', 'Grand', 'Très grand']
df['montant_cat'] = pd.cut(df['montant'], bins=bins, labels=labels)

# Quantiles (effectifs égaux)
df['montant_quantile'] = pd.qcut(df['montant'], q=4, labels=['Q1', 'Q2', 'Q3', 'Q4'])
```

Q8. Comment calculer des statistiques par groupe et les joindre au DataFrame original?

R8.

```python
# Méthode 1: transform
df['moyenne_agence'] = df.groupby('agence')['montant'].transform('mean')
df['ecart_moyenne'] = df['montant'] - df['moyenne_agence']

# Méthode 2: merge
stats_agence = df.groupby('agence').agg({
    'montant': ['mean', 'sum', 'count']
}).reset_index()
stats_agence.columns = ['agence', 'montant_mean', 'montant_sum', 'count']
df = df.merge(stats_agence, on='agence')
```

Q9. Comment détecter la multicolinéarité entre les features?

R9.

```python
# 1. Matrice de corrélation
corr_high = df.corr()[df.corr() > 0.8]

# 2. VIF (Variance Inflation Factor)
from statsmodels.stats.outliers_influence import variance_inflation_factor

X = df[['var1', 'var2', 'var3']]
vif = pd.DataFrame({
    'feature': X.columns,
    'VIF': [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
})
# VIF > 5 ou 10 indique multicolinéarité
```

Q10. Comment gérer les données textuelles dans un dataset?

R10.

```
# Nettoyage basique
df['texte'] = df['texte'].str.lower()
df['texte'] = df['texte'].str.strip()
df['texte'] = df['texte'].str.replace('[^\w\s]', '', regex=True)

# Extraction
df['longueur'] = df['texte'].str.len()
df['nb_mots'] = df['texte'].str.split().str.len()

# Contient un pattern
df['contient_urgent'] = df['texte'].str.contains('urgent', case=False)

# Extraction avec regex
df['code_postal'] = df['adresse'].str.extract(r'(\d{4})')
```

Q11. Comment faire une analyse de la qualité des données avec des assertions?

R11.

```
# Schema validation
def validate_data(df):
    errors = []

    # Types
    if not pd.api.types.is_numeric_dtype(df['montant']):
        errors.append("montant should be numeric")

    # Valeurs
    if (df['montant'] < 0).any():
        errors.append("montant has negative values")

    # Range
    if not df['taux'].between(0, 100).all():
        errors.append("taux out of range [0-100]")

    # Dates
    if df['date'].max() > pd.Timestamp.now():
        errors.append("Future dates detected")

    # Unique
    if df['client_id'].duplicated().any():
        errors.append("Duplicate client_id")

    return errors

# Execution
validation_errors = validate_data(df)
```

Q12. Comment utiliser pivot_table pour créer des tableaux croisés?

R12.

```
# Pivot simple
pivot = pd.pivot_table(
    df,
    values='montant',
    index='agence',
    columns='produit',
    aggfunc='sum',
    fill_value=0
)

# Pivot multi-agrégation
pivot = pd.pivot_table(
    df,
    values='montant',
    index=['region', 'agence'],
    columns='produit',
    aggfunc=['sum', 'mean', 'count'],
    margins=True  # Ajoute les totaux
)
```

Q13. Comment encoder les variables catégorielles avec des fréquences?

R13.

```
# Frequency encoding
freq = df['secteur'].value_counts(normalize=True)
df['secteur_freq'] = df['secteur'].map(freq)

# Count encoding
count = df['secteur'].value_counts()
df['secteur_count'] = df['secteur'].map(count)

# Target encoding (avec régularisation)
def target_encode(df, col, target, smoothing=10):
    global_mean = df[target].mean()
    agg = df.groupby(col)[target].agg(['mean', 'count'])
    smooth = (agg['mean'] * agg['count'] + global_mean * smoothing) / (agg['count'] + smoothing)
    return df[col].map(smooth)

df['secteur_target'] = target_encode(df, 'secteur', 'defaut')
```

Q14. Comment détecter et traiter les valeurs aberrantes dans les dates?

R14.

```
# Détecter dates invalides
df['date'] = pd.to_datetime(df['date'], errors='coerce')
invalid_dates = df[df['date'].isnull()]

# Dates futures
future_dates = df[df['date'] > pd.Timestamp.now()]
```

```python
# Dates trop anciennes
old_dates = df[df['date'] < '2000-01-01']

# Range valide
df = df[df['date'].between('2010-01-01', pd.Timestamp.now())]

# Imputation
df['date'] = df['date'].fillna(df['date'].median())
```

Q15. Comment créer des features basées sur le temps?

R15.

```python
# Extraction temporelle
df['annee'] = df['date'].dt.year
df['mois'] = df['date'].dt.month
df['jour'] = df['date'].dt.day
df['jour_semaine'] = df['date'].dt.dayofweek  # 0=Lundi
df['trimestre'] = df['date'].dt.quarter
df['semaine'] = df['date'].dt.isocalendar().week

# Features dérivées
df['est_weekend'] = df['jour_semaine'].isin([5, 6]).astype(int)
df['debut_mois'] = (df['jour'] <= 5).astype(int)
df['fin_mois'] = (df['jour'] >= 25).astype(int)

# Différence de dates
df['ancienneté_jours'] = (pd.Timestamp.now() - df['date_inscription']).dt.days
```

Q16. Comment faire un sampling stratifié?

R16.

```python
from sklearn.model_selection import train_test_split

# Stratifié sur la variable cible
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.3,
    stratify=y,  # Maintient les proportions
    random_state=42
)

# Sampling d'un DataFrame
df_sample = df.groupby('statut', group_keys=False).apply(
    lambda x: x.sample(min(len(x), 1000))
)
```

Q17. Comment utiliser method chaining pour un code plus lisible?

R17.

```python
# Mauvais: variables intermédiaires
df1 = df.dropna()
df2 = df1[df1['montant'] > 0]
df3 = df2.assign(ratio=df2['dette']/df2['revenu'])
df_final = df3.sort_values('ratio')

# Bon: method chaining
df_final = (df
    .dropna()
    .query('montant > 0')
    .assign(ratio=lambda x: x['dette'] / x['revenu'])
    .sort_values('ratio')
)
```

Q18. Comment identifier les relations non-linéaires entre variables?

R18.

```python
# 1. Scatter plots
sns.pairplot(df[['var1', 'var2', 'target']])

# 2. Corrélation de Spearman (non-linéaire monotone)
df.corr(method='spearman')

# 3. Mutual Information
from sklearn.feature_selection import mutual_info_regression
mi = mutual_info_regression(X, y)

# 4. Transformation polynomial
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly.fit_transform(X)
```

Q19. Comment utiliser query() pour filtrer les données?

R19.

```python
# Filtrage simple
df.query('age > 30')

# Conditions multiples
df.query('age > 30 and montant < 100000')

# Avec variables externes
seuil = 50000
df.query('montant > @seuil')

# Texte
df.query('secteur == "Agriculture"')

# In operator
secteurs = ['Agriculture', 'Commerce']
df.query('secteur in @secteurs')
```

Q20. Comment détecter les changements de distribution dans le temps (data drift)?

R20.

```python
from scipy.stats import ks_2samp

# Comparer deux périodes
df_old = df[df['date'] < '2024-01-01']['montant']
df_new = df[df['date'] >= '2024-01-01']['montant']

# Test KS (Kolmogorov-Smirnov)
stat, p_value = ks_2samp(df_old, df_new)
print(f"KS stat: {stat:.3f}, p-value: {p_value:.4f}")

if p_value < 0.05:
    print("Distribution change detected (drift)")

# PSI (Population Stability Index)
def calculate_psi(expected, actual, bins=10):
    expected_pct = np.histogram(expected, bins=bins)[0] / len(expected)
    actual_pct = np.histogram(actual, bins=bins)[0] / len(actual)
    psi = np.sum((actual_pct - expected_pct) * np.log(actual_pct / expected_pct))
    return psi
```

Q21. Comment créer un rapport EDA automatisé personnalisé?

R21.

```python
def eda_report(df):
    report = {}

    # Overview
    report['shape'] = df.shape
    report['memory_mb'] = df.memory_usage(deep=True).sum() / 1e6

    # Par colonne
    col_reports = []
    for col in df.columns:
        col_info = {
            'name': col,
            'dtype': str(df[col].dtype),
            'missing': df[col].isnull().sum(),
            'missing_pct': df[col].isnull().mean() * 100,
            'unique': df[col].nunique()
        }

        if pd.api.types.is_numeric_dtype(df[col]):
            col_info.update({
                'mean': df[col].mean(),
                'median': df[col].median(),
                'std': df[col].std(),
                'skew': df[col].skew()
            })
```

```
        col_reports.append(col_info)

    report['columns'] = pd.DataFrame(col_reports)
    return report
```

Q22. Comment optimiser la mémoire d'un DataFrame?

R22.

```
def optimize_memory(df):
    for col in df.columns:
        col_type = df[col].dtype

        if col_type == 'object':
            if df[col].nunique() / len(df) < 0.5:
                df[col] = df[col].astype('category')

        elif col_type == 'int64':
            if df[col].min() >= 0:
                if df[col].max() <= 255:
                    df[col] = df[col].astype('uint8')
                elif df[col].max() <= 65535:
                    df[col] = df[col].astype('uint16')
            else:
                if df[col].min() >= -128 and df[col].max() <= 127:
                    df[col] = df[col].astype('int8')
                elif df[col].min() >= -32768 and df[col].max() <= 32767:
                    df[col] = df[col].astype('int16')

        elif col_type == 'float64':
            df[col] = df[col].astype('float32')

    return df

# Peut réduire la mémoire de 50-80%
df_optimized = optimize_memory(df)
```

Q23. Comment utiliser groupby avec des fonctions d'agrégation personnalisées?

R23.

```
# Named aggregation
result = df.groupby('agence').agg(
    montant_total=('montant', 'sum'),
    montant_moyen=('montant', 'mean'),
    nb_clients=('client_id', 'nunique'),
    taux_defaut=('defaut', 'mean')
)

# Fonction personnalisée
def coefficient_variation(x):
    return x.std() / x.mean() * 100
```

```python
result = df.groupby('agence')['montant'].agg(['mean', 'std', coefficient_variation])

# Plusieurs colonnes avec lambda
result = df.groupby('agence').apply(
    lambda x: pd.Series({
        'weighted_avg': np.average(x['taux'], weights=x['montant']),
        'ratio': x['defaut'].sum() / x['montant'].sum()
    })
)
```

Q24. Comment gérer les outliers multivariés?

R24.

```python
# Isolation Forest (détection multivariée)
from sklearn.ensemble import IsolationForest

features = ['montant', 'nb_tx', 'anciennete']
X = df[features]

iso = IsolationForest(contamination=0.05, random_state=42)
df['outlier'] = iso.fit_predict(X)
outliers = df[df['outlier'] == -1]

# Mahalanobis distance
from scipy.spatial.distance import mahalanobis

mean = X.mean()
cov = X.cov()
cov_inv = np.linalg.inv(cov)

df['mahal_dist'] = X.apply(
    lambda row: mahalanobis(row, mean, cov_inv), axis=1
)
# Outliers si distance > seuil (chi2 avec df = nb features)
```

Q25. Décrivez un workflow complet de data wrangling pour préparer des données de scoring crédit.

R25.

```
1. CHARGEMENT
      Lire les données sources (CSV, SQL, Excel)
      Vérifier les encodages
      Définir les types appropriés

2. NETTOYAGE
      Valeurs manquantes
          Analyse patterns (MCAR/MAR/MNAR)
          Imputation adaptée par variable
          Créer indicateurs "is_missing"
      Outliers
          Détection IQR ou Isolation Forest
          Capping ou transformation log
```

```
      Doublons et erreurs
          Identifier par clé métier
          Documenter les suppressions

3. TRANSFORMATION
      Variables numériques
         Log pour asymétriques
         Scaling (StandardScaler)
         Binning si relation non-linéaire
      Variables catégorielles
         Target encoding pour haute cardinalité
         One-hot pour faible cardinalité
      Variables temporelles
          Extraction (mois, jour semaine)
          Ancienneté, recency

4. FEATURE ENGINEERING
      Ratios (DTI, LTV)
      Agrégations comportementales
      Interactions
      Features métier spécifiques

5. VALIDATION
      Vérifier distributions
      Corrélations et VIF
      Split stratifié
      Documenter transformations
```

Scoring

Score	Niveau
0-10	À améliorer
11-17	Intermédiaire
18-22	Avancé
23-25	Expert