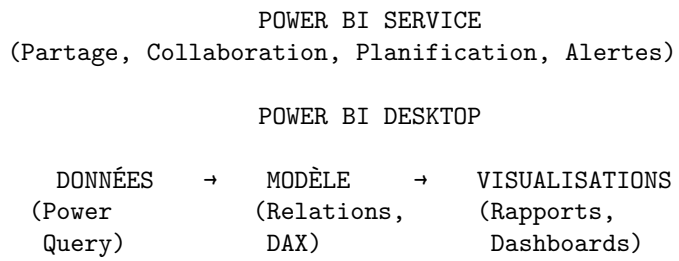
Manuel de Préparation: Power BI et Python

Introduction

Ce manuel couvre l'utilisation de Power BI et Python pour l'analyse de données dans le contexte bancaire. Power BI est l'outil de visualisation de Microsoft, tandis que Python avec Pandas offre des capacités d'analyse et de manipulation de données puissantes.

Partie 1: Power BI Fondamentaux

1.1 Architecture Power BI



1.2 Composants Clés

Composant	Fonction	Langage
Power Query	ETL, préparation des données	M
Modèle de données	Relations, calculs	DAX
Visualisations	Rapports, dashboards	-
Power BI Service	Publication, partage	-

1.3 Types de Connexion

- **Import:** Données copiées dans le modèle (recommandé pour analyses)
- **DirectQuery:** Requêtes en temps réel (données volumineuses)
- **Live Connection:** Connexion aux modèles existants (SSAS, datasets)
- **Composite:** Combinaison Import + DirectQuery

Partie 2: DAX (Data Analysis Expressions)

2.1 Concepts Fondamentaux

Colonnes Calculées vs Mesures

	Colonne Calculée	Mesure
Calcul	Au refresh des données	À la requête
Stockage	Consomme mémoire	Pas de stockage

	Colonne Calculée	Mesure
Contexte	Contexte de ligne	Contexte de filtre
Usage	Filtres, relations	Agrégations, KPIs

Contextes DAX Contexte de ligne (Row Context):

```
// Dans une colonne calculée - accède à chaque ligne
Profit = Sales[Revenue] - Sales[Cost]
```

Contexte de filtre (Filter Context):

```
// Dans une mesure - agrège selon les filtres actifs
Total Revenue = SUM(Sales[Revenue])
```

2.2 Fonctions DAX Essentielles

Agrégation

```
// Somme
Total Ventes = SUM(Ventes[Montant])

// Moyenne
Montant Moyen = AVERAGE(Transactions[Montant])

// Comptage
Nb Clients = DISTINCTCOUNT(Clients[ClientID])

// Min/Max
Plus Grand Pret = MAX(Prets[Montant])
```

CALCULATE - La fonction la plus importante

```
// Modifie le contexte de filtre
Ventes Premium = CALCULATE(
    SUM(Ventes[Montant]),
    Clients[Segment] = "Premium"
)

// Avec plusieurs filtres
Ventes Q1 Premium = CALCULATE(
    SUM(Ventes[Montant]),
    Clients[Segment] = "Premium",
    Calendrier[Trimestre] = "Q1"
)
```

FILTER

```
// Retourne une table filtrée
Gros Prets = CALCULATE(
    SUM(Prets[Montant]),
    FILTER(Prets, Prets[Montant] > 100000)
)
```

ALL, ALLEXCEPT, REMOVEFILTERS

```
// Ignore tous les filtres
Total Global = CALCULATE(
    SUM(Ventes[Montant]),
    ALL(Ventes)
)

// Pourcentage du total
% du Total = DIVIDE(
    SUM(Ventes[Montant]),
    CALCULATE(SUM(Ventes[Montant]), ALL(Ventes))
)

// Garde certains filtres
% par Agence = DIVIDE(
    SUM(Ventes[Montant]),
    CALCULATE(SUM(Ventes[Montant]), ALLEXCEPT(Ventes, Ventes[Agence]))
)
```

2.3 Time Intelligence

```
// Year to Date
YTD Ventes = TOTALYTD(SUM(Ventes[Montant]), Calendrier[Date])

// Quarter to Date
QTD Ventes = TOTALQTD(SUM(Ventes[Montant]), Calendrier[Date])

// Month to Date
MTD Ventes = TOTALMTD(SUM(Ventes[Montant]), Calendrier[Date])

// Année précédente
Ventes Annee Prec = CALCULATE(
    SUM(Ventes[Montant]),
    SAMEPERIODLASTYEAR(Calendrier[Date])
)

// Variation annuelle
Var YoY =
VAR VentesActuelles = SUM(Ventes[Montant])
VAR VentesPrecedentes = CALCULATE(
    SUM(Ventes[Montant]),
    SAMEPERIODLASTYEAR(Calendrier[Date])
)
RETURN
DIVIDE(VentesActuelles - VentesPrecedentes, VentesPrecedentes)

// Mois précédent
Ventes Mois Prec = CALCULATE(
    SUM(Ventes[Montant]),
    PREVIOUSMONTH(Calendrier[Date])
)

// Moyenne mobile 3 mois
```

```
MM3 Ventes = AVERAGEX(
    DATESINPERIOD(Calendrier[Date], MAX(Calendrier[Date]), -3, MONTH),
    CALCULATE(SUM(Ventes[Montant]))
)
```

2.4 Fonctions Logiques et Conditionnelles

```
// IF
Statut = IF(Prets[JoursRetard] > 90, "NPL", "Performing")

// SWITCH
Classification = SWITCH(
    TRUE(),
    Prets[JoursRetard] = 0, "A",
    Prets[JoursRetard] <= 30, "B",
    Prets[JoursRetard] <= 90, "C",
    Prets[JoursRetard] <= 180, "D",
    "E"
)

// COALESCE (première valeur non vide)
Valeur = COALESCE(Table[Col1], Table[Col2], 0)
```

2.5 Fonctions de Table

```
// SUMMARIZE - Grouper et agréger
Resume = SUMMARIZE(
    Ventes,
    Ventes[Agence],
    "Total", SUM(Ventes[Montant]),
    "Nb Transactions", COUNT(Ventes[ID])
)

// ADDCOLUMNS - Ajouter des colonnes
Enrichi = ADDCOLUMNS(
    Clients,
    "Total Achats", CALCULATE(SUM(Ventes[Montant]))
)

// TOPN
Top10Clients = TOPN(10, Clients, CALCULATE(SUM(Ventes[Montant])))

// RANKX
Rang Clients = RANKX(
    ALL(Clients),
    CALCULATE(SUM(Ventes[Montant])),,
    DESC,
    Dense
)
```

2.6 Variables (VAR/RETURN)

```
// Améliore la lisibilité et les performances
NPL Ratio =
```

```
VAR TotalPrets = SUM(Prets[Solde])
VAR NPL = CALCULATE(
    SUM(Prets[Solde]),
    Prets[JoursRetard] > 90
)
RETURN
DIVIDE(NPL, TotalPrets, 0)
```

Partie 3: Mesures Bancaires en DAX

3.1 Indicateurs de Performance

```
// ROE
ROE = DIVIDE(
    SUM(Resultats[ResultatNet]),
    AVERAGE(Bilan[CapitauxPropres])
) * 100

// ROA
ROA = DIVIDE(
    SUM(Resultats[ResultatNet]),
    AVERAGE(Bilan[TotalActifs])
) * 100

// Net Interest Margin
NIM = DIVIDE(
    SUM(Resultats[RevenuInteret]) - SUM(Resultats[ChargeInteret]),
    AVERAGE(Bilan[ActifsProductifs])
) * 100

// Cost to Income
CIR = DIVIDE(
    SUM(Resultats[ChargesExploitation]),
    SUM(Resultats[ProduitNetBancaire])
) * 100
```

3.2 Indicateurs de Risque

```
// NPL Ratio
NPL Ratio =
VAR NPL = CALCULATE(
    SUM(Prets[Solde]),
    Prets[JoursRetard] > 90
)
VAR Total = SUM(Prets[Solde])
RETURN DIVIDE(NPL, Total) * 100

// Coverage Ratio
Coverage Ratio = DIVIDE(
    SUM(Provisions[Montant]),
    CALCULATE(SUM(Prets[Solde]), Prets[JoursRetard] > 90)
) * 100
```

```
// Expected Loss
Expected Loss = SUMX(
    Prets,
    Prets[Solde] * Prets[PD] * Prets[LGD]
)
```

3.3 Indicateurs Clients

```
// Taux de churn
Taux Churn =
VAR ClientsDebut = CALCULATE(
    DISTINCTCOUNT(Clients[ClientID]),
    DATEADD(Calendrier[Date], -1, YEAR)
)
VAR ClientsPerdus = CALCULATE(
    DISTINCTCOUNT(Clients[ClientID]),
    Clients[Statut] = "Fermé",
    DATESINPERIOD(Calendrier[Date], MAX(Calendrier[Date]), -1, YEAR)
)
RETURN DIVIDE(ClientsPerdus, ClientsDebut) * 100

// Cross-sell Ratio
Cross Sell = DIVIDE(
    DISTINCTCOUNT(Produits[ProduitID]),
    DISTINCTCOUNT(Clients[ClientID])
)

// Customer Lifetime Value simplifié
CLV =
AVERAGEX(
    Clients,
    CALCULATE(SUM(Transactions[Marge])) * Clients[AncienneteMois] / 12
)
```

Partie 4: Python avec Pandas

4.1 Bases de Pandas

```python
import pandas as pd
import numpy as np

# Charger les données
df = pd.read_csv('transactions.csv')
df = pd.read_excel('clients.xlsx')
df = pd.read_sql('SELECT * FROM transactions', connection)

# Explorer
df.head()
df.info()
df.describe()
df.shape
```

```python
df.columns

# Sélection
df['colonne']           # Une colonne
df[['col1', 'col2']]    # Plusieurs colonnes
df.loc[0]               # Par label
df.iloc[0]              # Par position
df.loc[df['montant'] > 1000]  # Filtrage

# Types
df.dtypes
df['date'] = pd.to_datetime(df['date'])
df['montant'] = pd.to_numeric(df['montant'])
```

4.2 Manipulation de Données

```python
# Filtrage
df_high = df[df['montant'] > 10000]
df_premium = df[df['segment'] == 'Premium']
df_filtered = df[(df['montant'] > 1000) & (df['type'] == 'credit')]

# Tri
df.sort_values('montant', ascending=False)
df.sort_values(['agence', 'montant'])

# Agrégation
df.groupby('agence')['montant'].sum()
df.groupby('agence').agg({
    'montant': ['sum', 'mean', 'count'],
    'client_id': 'nunique'
})

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

# Merge/Join
df_merged = pd.merge(df_transactions, df_clients, on='client_id', how='left')

# Concatenation
df_combined = pd.concat([df1, df2], ignore_index=True)
```

4.3 Colonnes Calculées

```python
# Nouvelle colonne
df['profit'] = df['revenue'] - df['cost']
df['margin'] = df['profit'] / df['revenue'] * 100

# Conditionnelle
```

```python
df['segment'] = np.where(df['solde'] > 100000, 'High', 'Regular')

df['risk_category'] = np.select(
    [
        df['score'] >= 700,
        df['score'] >= 500,
        df['score'] >= 300
    ],
    ['Low', 'Medium', 'High'],
    default='Very High'
)

# Apply pour logique complexe
def categorize_client(row):
    if row['anciennete'] > 5 and row['produits'] >= 3:
        return 'Fidèle'
    elif row['anciennete'] < 1:
        return 'Nouveau'
    else:
        return 'Standard'

df['category'] = df.apply(categorize_client, axis=1)

# Extraction de date
df['year'] = df['date'].dt.year
df['month'] = df['date'].dt.month
df['day_of_week'] = df['date'].dt.dayofweek
df['is_weekend'] = df['day_of_week'].isin([5, 6])
```

4.4 Valeurs Manquantes et Nettoyage

```python
# Identifier
df.isnull().sum()
df.isna().mean() * 100

# Traiter
df.dropna()                           # Supprimer lignes
df.dropna(subset=['col_importante']) # Supprimer si col manquante
df.fillna(0)                          # Remplacer par 0
df['col'].fillna(df['col'].mean())    # Par la moyenne
df.fillna(method='ffill')             # Forward fill
df.fillna(method='bfill')             # Backward fill

# Doublons
df.duplicated().sum()
df.drop_duplicates()
df.drop_duplicates(subset=['client_id', 'date'], keep='last')
```

4.5 Analyse Temporelle

```python
# Resampling (données temporelles)
df.set_index('date', inplace=True)

# Agrégation mensuelle
```

```python
monthly = df.resample('M')['montant'].sum()

# Moyenne mobile
df['MA_7'] = df['montant'].rolling(window=7).mean()
df['MA_30'] = df['montant'].rolling(window=30).mean()

# Variation
df['pct_change'] = df['montant'].pct_change()
df['diff'] = df['montant'].diff()

# Décalage
df['montant_prev'] = df['montant'].shift(1)
df['montant_next'] = df['montant'].shift(-1)
```

Partie 5: Intégration Python dans Power BI

5.1 Python comme Source de Données

Dans Power Query:

```python
# Ce code s'exécute dans Power BI
import pandas as pd

# Traitement des données
df = dataset.copy()  # 'dataset' est fourni par Power BI
df['profit'] = df['revenue'] - df['cost']
df['category'] = df['amount'].apply(lambda x: 'High' if x > 10000 else 'Low')
```

5.2 Visuals Python dans Power BI

```python
# Le dataset est automatiquement disponible
import matplotlib.pyplot as plt
import seaborn as sns

# Exemple: Heatmap de corrélation
plt.figure(figsize=(10, 8))
correlation = dataset.corr()
sns.heatmap(correlation, annot=True, cmap='coolwarm', center=0)
plt.title('Matrice de Corrélation')
plt.tight_layout()
plt.show()
```

```python
# Distribution avec histogramme et KDE
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.histplot(data=dataset, x='montant', kde=True)
plt.title('Distribution des Montants')
plt.xlabel('Montant')
plt.ylabel('Fréquence')
plt.show()
```

5.3 Cas d'Usage Python + Power BI

Cas	Python pour...	Power BI pour...
Prédiction	ML models, scoring	Visualisation des résultats
Clustering	K-means, segmentation	Analyse par segment
Stats avancées	Tests, régression	Dashboard des métriques
Nettoyage	ETL complexe	Transformation simple

Partie 6: Questions Fréquentes Power BI

6.1 Modélisation

Q: Différence entre colonne calculée et mesure?

```
Colonne calculée:
- Calculée au refresh
- Stockée dans le modèle
- Utilisable pour filtres et slicers

Mesure:
- Calculée à la requête
- Pas de stockage
- Dynamique selon contexte
```

Q: Qu'est-ce qu'une table de dates et pourquoi est-elle importante?

```
Calendrier =
ADDCOLUMNS(
    CALENDAR(DATE(2020,1,1), DATE(2025,12,31)),
    "Année", YEAR([Date]),
    "Mois", MONTH([Date]),
    "NomMois", FORMAT([Date], "MMMM"),
    "Trimestre", "Q" & QUARTER([Date]),
    "Semaine", WEEKNUM([Date])
)
```

→ Nécessaire pour Time Intelligence

Q: Différence entre CALCULATE et FILTER?

```
CALCULATE: Modifie le contexte de filtre pour un calcul
FILTER: Retourne une table filtrée (souvent utilisé dans CALCULATE)
```

6.2 Performance

Q: Comment optimiser un rapport Power BI? - Réduire les colonnes inutiles - Utiliser des mesures plutôt que des colonnes calculées - Éviter les relations bidirectionnelles - Utiliser des types de données appropriés - Limiter les visuels par page

6.3 DAX Avancé

Q: Expliquez CALCULATE avec ALL

```
// % du total global
% Global = DIVIDE(
    SUM(Ventes[Montant]),
    CALCULATE(SUM(Ventes[Montant]), ALL(Ventes))
)

// ALL supprime tous les filtres de la table Ventes
// Donc le dénominateur est toujours le total global
```

Questions d'Entretien

1. **Différence entre CALCULATE et SUMX?** → CALCULATE modifie le contexte; SUMX itère sur une table
2. **Comment créer une mesure YTD?** → `TOTALYTD(SUM(Ventes[Montant]), Calendrier[Date])`
3. **Qu'est-ce que le contexte de filtre en DAX?** → L'ensemble des filtres actifs qui affectent un calcul
4. **Comment faire une moyenne mobile en DAX?** → Utiliser AVERAGEX avec DATESINPERIOD
5. **Comment intégrer Python dans Power BI?** → Via Power Query (source) ou visuels Python

---

Checklist Power BI

```
Comprendre les contextes DAX (ligne vs filtre)
Maîtriser CALCULATE et ses modificateurs
Connaître les fonctions Time Intelligence
Savoir créer une table de dates
Comprendre les relations et cardinalités
Optimiser les performances du modèle
Savoir utiliser les variables DAX
Intégrer Python quand nécessaire
```

Rappel: Power BI est puissant pour la visualisation, DAX pour les calculs analytiques, et Python pour les analyses avancées. La combinaison des trois offre une solution complète.