Installation et Imports

In [1]:
```python
# Requirements d'installation
# pip install pandas pyarrow fastparquet

import pandas as pd
import pyarrow as pa
import pyarrow.parquet as pq
import numpy as np
import os
import time

print("Démonstration Parquet Simplifiée - NYC Taxi Data")
```

```
Démonstration Parquet Simplifiée - NYC Taxi Data
```

Fonctions de Téléchargement et Simulation

In [2]:
```python
def download_taxi_data():
    """Télécharge les données NYC Taxi 2025 ou crée des données simulées"""

    # URLs des données NYC Taxi 2025
    urls_2025 = [
        "https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_2025-01.parquet",
        "https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_2024-12.parquet",
        "https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_2024-11.parquet"
    ]

    for url in urls_2025:
        try:
            print(f"📥 Tentative de téléchargement: {url.split('/')[-1]}")
            df = pd.read_parquet(url)
            print(f"Données téléchargées: {len(df):,} lignes")
            return df.head(50000)  # Limiter pour la démo
        except Exception as e:
            print(f" Erreur: {e}")
            continue

    print("Tous les téléchargements ont échoué, création de données simulées...")
    return create_simulated_taxi_data()

def create_simulated_taxi_data(n_rows=50000):
    """Crée des données de taxi simulées NYC style"""

    np.random.seed(42)

    data = {
        'tpep_pickup_datetime': pd.date_range('2025-01-01', periods=n_rows, freq='1min'),
        'tpep_dropoff_datetime': pd.date_range('2025-01-01 00:10:00', periods=n_rows, freq='1min'),
        'passenger_count': np.random.choice([1, 2, 3, 4, 5], n_rows, p=[0.5, 0.3, 0.1, 0.05, 0.05]),
        'trip_distance': np.random.exponential(2.5, n_rows),
        'fare_amount': np.random.gamma(2, 5, n_rows),
        'extra': np.random.choice([0, 0.5, 1], n_rows, p=[0.7, 0.2, 0.1]),
        'tip_amount': np.random.gamma(1, 2, n_rows),
        'total_amount': np.random.gamma(3, 7, n_rows),
        'pickup_location_id': np.random.randint(1, 265, n_rows),
        'dropoff_location_id': np.random.randint(1, 265, n_rows),
        'payment_type': np.random.choice([1, 2, 3, 4], n_rows, p=[0.6, 0.3, 0.05, 0.05])
    }

    df = pd.DataFrame(data)
    df['total_amount'] = df['fare_amount'] + df['tip_amount'] + df['extra']

    return df
```

Chargement des Données

In [3]:
```python
# Chargement des données
taxi_df = download_taxi_data()
print(f"📊 Dataset: {taxi_df.shape[0]:,} lignes, {taxi_df.shape[1]} colonnes")

# Aperçu des données
print("\n📋 Aperçu des données:")
print(taxi_df.head())
print(f"\n📊 Info sur le dataset:")
print(taxi_df.info())
```

```
📥 Tentative de téléchargement: yellow_tripdata_2025-01.parquet
Données téléchargées: 3,475,226 lignes
📊 Dataset: 50,000 lignes, 20 colonnes

📋 Aperçu des données:
   VendorID tpep_pickup_datetime tpep_dropoff_datetime  passenger_count  \
0         1  2025-01-01 00:18:38   2025-01-01 00:26:59              1.0   
1         1  2025-01-01 00:32:40   2025-01-01 00:35:13              1.0   
2         1  2025-01-01 00:44:04   2025-01-01 00:46:01              1.0   
3         2  2025-01-01 00:14:27   2025-01-01 00:20:01              3.0   
4         2  2025-01-01 00:21:34   2025-01-01 00:25:06              3.0   

   trip_distance  RatecodeID store_and_fwd_flag  PULocationID  DOLocationID  \
0           1.60         1.0                  N           229           237   
1           0.50         1.0                  N           236           237   
2           0.60         1.0                  N           141           141   
3           0.52         1.0                  N           244           244   
4           0.66         1.0                  N           244           116   

   payment_type  fare_amount  extra  mta_tax  tip_amount  tolls_amount  \
0             1         10.0    3.5      0.5        3.00           0.0   
1             1          5.1    3.5      0.5        2.02           0.0   
2             1          5.1    3.5      0.5        2.00           0.0   
3             2          7.2    1.0      0.5        0.00           0.0   
4             2          5.8    1.0      0.5        0.00           0.0   

   improvement_surcharge  total_amount  congestion_surcharge  Airport_fee  \
0                    1.0         18.00                   2.5          0.0   
1                    1.0         12.12                   2.5          0.0   
2                    1.0         12.10                   2.5          0.0   
3                    1.0          9.70                   0.0          0.0   
4                    1.0          8.30                   0.0          0.0   

   cbd_congestion_fee  
0                 0.0  
1                 0.0  
2                 0.0  
3                 0.0  
4                 0.0  

📊 Info sur le dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 20 columns):
 #   Column                 Non-Null Count  Dtype         
---  ------                 --------------  -----         
 0   VendorID               50000 non-null  int32         
 1   tpep_pickup_datetime   50000 non-null  datetime64[us]
 2   tpep_dropoff_datetime  50000 non-null  datetime64[us]
 3   passenger_count        50000 non-null  float64       
 4   trip_distance          50000 non-null  float64       
 5   RatecodeID             50000 non-null  float64       
 6   store_and_fwd_flag     50000 non-null  object        
 7   PULocationID           50000 non-null  int32         
 8   DOLocationID           50000 non-null  int32         
 9   payment_type           50000 non-null  int64         
 10  fare_amount            50000 non-null  float64       
 11  extra                  50000 non-null  float64       
 12  mta_tax                50000 non-null  float64       
 13  tip_amount             50000 non-null  float64       
 14  tolls_amount           50000 non-null  float64       
 15  improvement_surcharge  50000 non-null  float64       
 16  total_amount           50000 non-null  float64       
 17  congestion_surcharge   50000 non-null  float64       
 18  Airport_fee            50000 non-null  float64       
 19  cbd_congestion_fee     50000 non-null  float64       
dtypes: datetime64[us](2), float64(13), int32(3), int64(1), object(1)
memory usage: 7.1+ MB
None
```

Sauvegarde en Différents Formats

In [4]:
```python
print("\n" + "=" * 60)
print("💾 SAUVEGARDE EN DIFFÉRENTS FORMATS")
print("=" * 60)

# CSV
print("📄 Sauvegarde CSV...")
start_time = time.time()
taxi_df.to_csv("taxi_data.csv", index=False)
csv_time = time.time() - start_time
csv_size = os.path.getsize("taxi_data.csv") / 1024**2

# Parquet avec différentes compressions
print("📦 Sauvegarde Parquet (sans compression)...")
start_time = time.time()
taxi_df.to_parquet("taxi_data_none.parquet", compression=None)
parquet_none_time = time.time() - start_time
parquet_none_size = os.path.getsize("taxi_data_none.parquet") / 1024**2

print("📦 Sauvegarde Parquet (Snappy)...")
start_time = time.time()
taxi_df.to_parquet("taxi_data_snappy.parquet", compression='snappy')
parquet_snappy_time = time.time() - start_time
parquet_snappy_size = os.path.getsize("taxi_data_snappy.parquet") / 1024**2

print("📦 Sauvegarde Parquet (Gzip)...")
start_time = time.time()
taxi_df.to_parquet("taxi_data_gzip.parquet", compression='gzip')
parquet_gzip_time = time.time() - start_time
parquet_gzip_size = os.path.getsize("taxi_data_gzip.parquet") / 1024**2

print("✅ Toutes les sauvegardes terminées!")
```

```
============================================================
💾 SAUVEGARDE EN DIFFÉRENTS FORMATS
============================================================
📄 Sauvegarde CSV...
📦 Sauvegarde Parquet (sans compression)...
📦 Sauvegarde Parquet (Snappy)...
📦 Sauvegarde Parquet (Gzip)...
✅ Toutes les sauvegardes terminées!
```

Comparaison des Tailles

In [5]:
```python
print("\n" + "=" * 60)
print("📏 COMPARAISON DES TAILLES")
print("=" * 60)

print(f"Format                | Taille (MB) | Temps (s) | Ratio vs CSV")
print("-" * 65)
print(f"CSV                   | {csv_size:8.1f}   | {csv_time:6.2f}   | 1.0x")
print(f"Parquet (aucune)      | {parquet_none_size:8.1f}   | {parquet_none_time:6.2f}   | {csv_size/parquet_none_size:.1f}x")
print(f"Parquet (Snappy)      | {parquet_snappy_size:8.1f}   | {parquet_snappy_time:6.2f}   | {csv_size/parquet_snappy_size:.1f}x")
print(f"Parquet (Gzip)        | {parquet_gzip_size:8.1f}   | {parquet_gzip_time:6.2f}   | {csv_size/parquet_gzip_size:.1f}x")

print(f"\n🏆 Meilleure compression: {csv_size/min(parquet_none_size, parquet_snappy_size, parquet_gzip_size):.1f}x plus compact")
```

```
============================================================
📏 COMPARAISON DES TAILLES
============================================================
Format                | Taille (MB) | Temps (s) | Ratio vs CSV
-----------------------------------------------------------------
CSV                   |      5.3   |   1.33   | 1.0x
Parquet (aucune)      |      1.2   |   0.09   | 4.3x
Parquet (Snappy)      |      1.1   |   0.09   | 5.0x
Parquet (Gzip)        |      0.8   |   0.27   | 6.2x

🏆 Meilleure compression: 6.2x plus compact
```

Performance de Lecture Complète

In [6]:
```python
print("\n" + "=" * 60)
print("⚡ PERFORMANCE DE LECTURE")
print("=" * 60)

# Lecture complète CSV
print("📖 Test lecture CSV...")
start_time = time.time()
df_csv = pd.read_csv("taxi_data.csv")
csv_read_time = time.time() - start_time

# Lecture complète Parquet
print("📖 Test lecture Parquet...")
start_time = time.time()
df_parquet = pd.read_parquet("taxi_data_snappy.parquet")
parquet_read_time = time.time() - start_time

print(f"\nFormat    | Temps lecture | Lignes")
print("-" * 40)
print(f"CSV       | {csv_read_time:8.2f}s     | {len(df_csv):,}")
print(f"Parquet   | {parquet_read_time:8.2f}s     | {len(df_parquet):,}")
print(f"\n🚀 Parquet est {csv_read_time/parquet_read_time:.1f}x plus rapide pour la lecture")
```

```
============================================================
⚡ PERFORMANCE DE LECTURE
============================================================
📖 Test lecture CSV...
📖 Test lecture Parquet...

Format    | Temps lecture | Lignes
----------------------------------------
CSV       |     0.52s     | 50,000
Parquet   |     0.06s     | 50,000

🚀 Parquet est 9.1x plus rapide pour la lecture
```

Performance de Filtrage

In [7]:
```python
print("\n" + "=" * 60)
print("🔍 PERFORMANCE DE FILTRAGE")
print("=" * 60)

# Test: Courses avec montant > 20$
print("🎯 Test: Courses avec total_amount > 20$")

# Filtrage CSV (lecture complète puis filtrage)
start_time = time.time()
df_csv_full = pd.read_csv("taxi_data.csv")
df_csv_filtered = df_csv_full[df_csv_full['total_amount'] > 20]
csv_filter_time = time.time() - start_time

# Filtrage Parquet (avec predicate pushdown)
start_time = time.time()
df_parquet_filtered = pd.read_parquet("taxi_data_snappy.parquet",
                                     filters=[('total_amount', '>', 20)])
parquet_filter_time = time.time() - start_time

print(f"\nMéthode           | Temps    | Lignes résultat")
print("-" * 45)
print(f"CSV (full+filter) | {csv_filter_time:6.2f}s | {len(df_csv_filtered):,}")
print(f"Parquet (filter)  | {parquet_filter_time:6.2f}s | {len(df_parquet_filtered):,}")
print(f"\n🚀 Parquet filtrage est {csv_filter_time/parquet_filter_time:.1f}x plus rapide")
```

```
============================================================
🔍 PERFORMANCE DE FILTRAGE
============================================================
🎯 Test: Courses avec total_amount > 20$

Méthode           | Temps    | Lignes résultat
---------------------------------------------
CSV (full+filter) |   0.39s | 25,095
Parquet (filter)  |   0.03s | 25,095

🚀 Parquet filtrage est 12.2x plus rapide
```

Performance de Lecture par Colonnes

In [8]:
```python
# Test: Lecture colonnes spécifiques
print(f"\n🎯 Test: Lecture colonnes spécifiques")
columns_to_read = ['tpep_pickup_datetime', 'passenger_count', 'total_amount']

# CSV - colonnes spécifiques
start_time = time.time()
df_csv_cols = pd.read_csv("taxi_data.csv", usecols=columns_to_read)
csv_cols_time = time.time() - start_time

# Parquet - colonnes spécifiques
start_time = time.time()
df_parquet_cols = pd.read_parquet("taxi_data_snappy.parquet", columns=columns_to_read)
parquet_cols_time = time.time() - start_time

print(f"\nMéthode           | Temps    | Colonnes")
print("-" * 40)
print(f"CSV colonnes      | {csv_cols_time:6.2f}s | {len(df_csv_cols.columns)}")
print(f"Parquet colonnes  | {parquet_cols_time:6.2f}s | {len(df_parquet_cols.columns)}")
print(f"\n🚀 Parquet colonnes est {csv_cols_time/parquet_cols_time:.1f}x plus rapide")

# Aperçu des données lues
print(f"\n📋 Aperçu des colonnes lues:")
print(df_parquet_cols.head())
```

```
🎯 Test: Lecture colonnes spécifiques

Méthode           | Temps    | Colonnes
----------------------------------------
CSV colonnes      |   0.24s | 3
Parquet colonnes  |   0.02s | 3

🚀 Parquet colonnes est 13.0x plus rapide

📋 Aperçu des colonnes lues:
  tpep_pickup_datetime  passenger_count  total_amount
0  2025-01-01 00:18:38              1.0         18.00
1  2025-01-01 00:32:40              1.0         12.12
2  2025-01-01 00:44:04              1.0         12.10
3  2025-01-01 00:14:27              3.0          9.70
4  2025-01-01 00:21:34              3.0          8.30
```

Métadonnées Parquet

In [9]:
```python
print("\n" + "=" * 60)
print("📋 MÉTADONNÉES PARQUET")
print("=" * 60)

# Ouvrir le fichier Parquet
parquet_file = pq.ParquetFile("taxi_data_snappy.parquet")

print("🔍 INFORMATIONS GÉNÉRALES:")
print(f"  Nombre de lignes: {parquet_file.metadata.num_rows:,}")
print(f"  Nombre de colonnes: {parquet_file.metadata.num_columns}")
print(f"  Nombre de row groups: {parquet_file.metadata.num_row_groups}")
print(f"  Taille fichier: {os.path.getsize('taxi_data_snappy.parquet') / 1024**2:.1f} MB")
print(f"  Compression: {parquet_file.metadata.row_group(0).column(0).compression}")

print(f"\n📊 SCHEMA DES COLONNES:")
schema = parquet_file.schema_arrow
for i, field in enumerate(schema):
    print(f"  {field.name:25} | {field.type}")
```

```
============================================================
📋 MÉTADONNÉES PARQUET
============================================================
🔍 INFORMATIONS GÉNÉRALES:
  Nombre de lignes: 50,000
  Nombre de colonnes: 20
  Nombre de row groups: 1
  Taille fichier: 1.1 MB
  Compression: SNAPPY

📊 SCHEMA DES COLONNES:
  VendorID                  | int32
  tpep_pickup_datetime      | timestamp[us]
  tpep_dropoff_datetime     | timestamp[us]
  passenger_count           | double
  trip_distance             | double
  RatecodeID                | double
  store_and_fwd_flag        | string
  PULocationID              | int32
  DOLocationID              | int32
  payment_type              | int64
  fare_amount               | double
  extra                     | double
  mta_tax                   | double
  tip_amount                | double
  tolls_amount              | double
  improvement_surcharge     | double
  total_amount              | double
  congestion_surcharge      | double
  Airport_fee               | double
  cbd_congestion_fee        | double
```

Statistiques des Colonnes

In [10]:
```python
print("\n" + "=" * 60)
print("📈 STATISTIQUES PAR COLONNE")
print("=" * 60)

# Statistiques du premier row group
rg_metadata = parquet_file.metadata.row_group(0)
schema_fields = parquet_file.schema_arrow

print("🔢 STATISTIQUES AUTOMATIQUES:")
for i in range(min(5, rg_metadata.num_columns)):  # Limiter à 5 colonnes
    col_metadata = rg_metadata.column(i)
    stats = col_metadata.statistics
    field_name = schema_fields[i].name
    field_type = schema_fields[i].type

    print(f"\n📊 {col_metadata.path_in_schema}:")
    print(f"  Type Arrow: {field_type}")
    print(f"  Type physique: {col_metadata.physical_type}")
    print(f"  Compression: {col_metadata.compression}")
    print(f"  Taille compressée: {col_metadata.total_compressed_size:,} bytes")
    print(f"  Taille non-compressée: {col_metadata.total_uncompressed_size:,} bytes")

    if stats:
        if stats.has_min_max:
            print(f"  Minimum: {stats.min}")
            print(f"  Maximum: {stats.max}")
        print(f"  Valeurs nulles: {stats.null_count:,}")
        if stats.distinct_count:
            print(f"  Valeurs distinctes: {stats.distinct_count:,}")
```

```
============================================================
📈 STATISTIQUES PAR COLONNE
============================================================
🔢 STATISTIQUES AUTOMATIQUES:

📊 VendorID:
  Type Arrow: int32
  Type physique: INT32
  Compression: SNAPPY
  Taille compressée: 9,871 bytes
  Taille non-compressée: 11,538 bytes
  Minimum: 1
  Maximum: 7
  Valeurs nulles: 0

📊 tpep_pickup_datetime:
  Type Arrow: timestamp[us]
  Type physique: INT64
  Compression: SNAPPY
  Taille compressée: 280,822 bytes
  Taille non-compressée: 337,738 bytes
  Minimum: 2024-12-31 20:47:55
  Maximum: 2025-01-01 17:02:00
  Valeurs nulles: 0

📊 tpep_dropoff_datetime:
  Type Arrow: timestamp[us]
  Type physique: INT64
  Compression: SNAPPY
  Taille compressée: 284,866 bytes
  Taille non-compressée: 339,850 bytes
  Minimum: 2024-12-31 20:54:00
  Maximum: 2025-01-02 16:37:05
  Valeurs nulles: 0

📊 passenger_count:
  Type Arrow: double
  Type physique: DOUBLE
  Compression: SNAPPY
  Taille compressée: 18,185 bytes
  Taille non-compressée: 24,065 bytes
  Minimum: -0.0
  Maximum: 9.0
  Valeurs nulles: 0

📊 trip_distance:
  Type Arrow: double
  Type physique: DOUBLE
  Compression: SNAPPY
  Taille compressée: 84,737 bytes
  Taille non-compressée: 93,499 bytes
  Minimum: -0.0
  Maximum: 133.3
  Valeurs nulles: 0
```

Résumé

In [11]:
```python
print("\n" + "=" * 60)
print("🎯 RÉSUMÉ")
print("=" * 60)

print("✅ AVANTAGES PARQUET DÉMONTRÉS:")
print(f"  📦 Compression: {csv_size/parquet_snappy_size:.1f}x plus compact que CSV")
print(f"  ⚡ Lecture: {csv_read_time/parquet_read_time:.1f}x plus rapide que CSV")
print(f"  🔍 Filtrage: {csv_filter_time/parquet_filter_time:.1f}x plus rapide avec prédicats")
print(f"  📊 Colonnes: {csv_cols_time/parquet_cols_time:.1f}x plus rapide pour lecture sélective")
print("  📋 Métadonnées: Schema automatique, statistiques, compression")

print("\n🎉 Démonstration terminée!")
print("=" * 60)
```

```
============================================================
🎯 RÉSUMÉ
============================================================
✅ AVANTAGES PARQUET DÉMONTRÉS:
  📦 Compression: 5.0x plus compact que CSV
  ⚡ Lecture: 9.1x plus rapide que CSV
  🔍 Filtrage: 12.2x plus rapide avec prédicats
  📊 Colonnes: 13.0x plus rapide pour lecture sélective
  📋 Métadonnées: Schema automatique, statistiques, compression
```

```
🎉 Démonstration terminée!
============================================================
```

```
In [ ]:
```