Université de Carthage
Faculté des sciences Economiques et de Gestion de Nabeul

Analyses des données boursières du cours d'une action

Matière : logiciel python

Présenté par : khouloud kalboussi
Nourhène bel feki

Niveau et Filière : deuxième année mastère professionnel en ingénierie économique et financière

Année Universitaire 2023-2024

Plan

Introduction générale

Le projet dédié à l'exploration des marchés boursiers représente une initiative essentielle visant à décrypter, analyser et appréhender les données financières provenant du marché boursier, en utilisant des outils avancés de programmation et d'analyse de données à l'aide du langage Python. Cet effort significatif se révèle indispensable pour une variété d'acteurs, notamment les investisseurs, les analystes financiers et les professionnels du secteur financier, en fournissant des informations cruciales qui sous-tendent la prise de décisions éclairées en matière d'investissement.

Cette approche méthodique permettra de dévoiler les dynamiques complexes des marchés boursiers, offrant ainsi une vision approfondie des tendances, des modèles et des opportunités d’investissement.

Concepts de base

Python : est un langage de programmation interprété, polyvalent et facile à lire. Il se distingue par sa syntaxe concise, son orientation objet et sa large adoption dans divers domaines tels que le développement web, l'analyse de données et l'intelligence artificielle. Python possède une vaste bibliothèque standard qui facilite le développement d'applications.

Anaconda : est une distribution Python complète qui offre un ensemble d'outils et de ressources prêts à l'emploi pour la science des données, l'analyse de données et l'apprentissage automatique, ce qui en fait un choix populaire pour les professionnels et les débutants dans ces domaines.

Jupyter : fournit un environnement flexible et interactif pour écrire du code, créer des visualisations, documenter des analyses et partager des résultats, ce qui en fait un outil puissant et largement utilisé dans le domaine de la science des données, de l'apprentissage automatique et de la recherche scientifique.

Pandas : est une bibliothèque essentielle en Python pour le travail et l'analyse de données. Elle offre des fonctionnalités puissantes pour explorer, nettoyer, manipuler, analyser et visualiser des ensembles de données, ce qui en fait un outil précieux pour les scientifiques des données, les analystes et les développeurs travaillant sur des projets impliquant la manipulation de données.

Filtrage : permet de sélectionner des parties spécifiques des données en fonction de critères définis, ce qui est essentiel pour explorer et analyser des ensembles de données plus grands et pour extraire des informations pertinentes pour une analyse ou une visualisation ultérieure.

Tri : est le processus de réarrangement d'un ensemble d'éléments dans un ordre spécifique, souvent croissant ou décroissant, selon un critère défini. Il est largement utilisé en informatique pour organiser des données de manière efficace.

Groupage avec **groupby () :** en Pandas permet de découper des données en sous-ensembles en fonction de critères spécifiques, offrant ainsi une manière puissante d'analyser et de manipuler des données par groupes pour des analyses plus approfondies.

Numpty : est une bibliothèque puissante et essentielle en Python pour le calcul scientifique et numérique, fournissant des structures de données et des fonctionnalités qui facilitent la manipulation, l'analyse et le traitement des données.

Matplotlib : est une bibliothèque de visualisation puissante et polyvalente qui fournit des outils essentiels pour créer des graphiques de données personnalisés et informatifs, ce qui en fait un choix populaire parmi les scientifiques des données, les chercheurs et les professionnels de divers domaines.

Seaborn : est une bibliothèque complémentaire à Matplotlib, offrant des fonctionnalités avancées de visualisation statistique, une facilité d'utilisation et une personnalisation esthétique pour créer des graphiques informatifs et attrayants dans le domaine de la science des données et de l'analyse de données.

Matplotlib offre une plus grande flexibilité pour la création de graphiques personnalisés, tandis que **Seaborn** facilite la création rapide de graphiques statistiques attrayants en utilisant des fonctions de haut niveau, en se basant sur Matplotlib pour sa puissance sous-jacente.

Le choix entre les deux dépend souvent du niveau de personnalisation nécessaire et de la complexité des graphiques à créer. Parfois, ces deux bibliothèques sont utilisées ensemble pour tirer parti de leurs forces respectives.

Objectif de projet

Les scripts fournis visent à effectuer diverses opérations sur des données tabulaires, principalement financières, en utilisant le langage de programmation Python et des bibliothèques telles que Pandas, NumPty, Matplotlib et Seaborn. L'objectif global de ces scripts est de faciliter le processus d'importation, de manipulation, d'analyse et de visualisation des données, offrant ainsi aux utilisateurs la possibilité d'explorer les informations contenues dans les fichiers CSV. Les différentes étapes comprennent le traitement des valeurs manquantes, la vérification rapide de la présence de données nulles dans des colonnes spécifiques, le filtrage des données en fonction de certaines conditions, le tri des données par date, le regroupement des données par certaines caractéristiques, et l'obtention de statistiques descriptives comme la moyenne, la médiane et l'écart-type. De plus, les codes génèrent des visualisations, telles que des histogrammes et des diagrammes en secteurs, pour aider à la compréhension visuelle des distributions et des relations entre différentes variables. Enfin, un code spécifique recommande des périodes d'investissement basées sur des conditions de volume spécifiques. En combinant ces techniques, les scripts fournissent un ensemble complet d'outils pour explorer, analyser et tirer des insights à partir de données tabulaires. Voici un résumé des objectifs généraux de chaque script :

Script 3 : Importation des Données

Objectif : Charger les données financières à partir d'un fichier CSV dans un format tabulaire, prêt pour la manipulation, l'analyse et le traitement ultérieur avec Python.

Script 4 : Résumé des Valeurs Manquantes

Objectif : Obtenir un résumé rapide du nombre de valeurs manquantes dans chaque colonne du Data Frame, permettant d'identifier les colonnes nécessitant une gestion des valeurs nulles.

Script 5 : Vérification des Valeurs Nulles dans 'Open'

Objectif : Vérifier la présence de valeurs nulles dans les 50 premières lignes de la colonne 'Open' du Data Frame 'Titanic Data' pour la gestion précoce des données manquantes.

Script 6 : Filtrage basé sur la Condition de Volume

Objectif : Filtrer et afficher les données répondant à la condition spécifique ('Volume' > 20 000), permettant l'analyse d'un sous-ensemble où les volumes échangés sont significativement élevés.

Script 7 : Tri Croissant par Date et Extraction des 100 Premières Lignes

Objectif : Trier le Data Frame 'Titanic Data' par ordre croissant en utilisant la colonne 'Date' comme critère principal. Extraire ensuite les 100 premières lignes pour fournir une vue chronologique détaillée des données.

Script 8 : Tri Décroissant par Date et Extraction des 200 Premières Lignes

Objectif : Trier le Data Frame 'Titanic Data' par ordre décroissant en utilisant la colonne 'Date'. Extraire les 200 premières lignes pour fournir une vue détaillée des enregistrements les plus récents.

Scripts 1 et 2 : Tri Décroissant par Date et Groupage par 'Close'

Objectif : Trier les données en fonction de la colonne 'Date' de manière décroissante. Groupage des données basé sur la colonne 'Close' pour fournir des perspectives détaillées sur la relation entre les dates et les valeurs de clôture.

Script 9 : Groupage des Données par 'Close' et Affichage des Premières Lignes de Chaque Groupe

Objectif : Effectuer un regroupement des données financières basé sur les valeurs de clôture ('Close') pour obtenir des informations détaillées sur chaque groupe créé par le groupby.

Scripts 10 et 11 : Calcul des Statistiques Descriptives sur 'Close'

Objectif : Obtenir des statistiques descriptives telles que la moyenne, la médiane et l'écart-type de la colonne 'Close' pour fournir des informations sur la tendance centrale et la dispersion des données.

Script 12 : Génération d'un Histogramme pour la Colonne 'Low'

Objectif : Utiliser Matplotlib pour générer un histogramme représentant la distribution des valeurs de la colonne 'Low', offrant une visualisation de la répartition des prix bas dans les données financières.

Script 13 : Génération d'un Histogramme pour la Colonne 'Low' avec Matplotlib

Objectif : Générer un histogramme représentant la distribution des valeurs de la colonne 'Low' dans les données financières à l'aide de Matplotlib.

Script 14 : Tracé de l'Évolution des Valeurs de 'Low' par Rapport à la Date

Objectif : Utiliser Matplotlib pour tracer l'évolution des valeurs de la colonne 'Low' par rapport à la date, permettant l'observation des tendances temporelles.

Script 15 : Création d'un Graphique à Barres Comparant 'Close' et 'Low'

Objectif : Utiliser Seaborn pour créer un graphique à barres comparant les valeurs de la colonne 'Close' en fonction des différentes valeurs de la colonne 'Low', explorant ainsi la relation entre ces deux variables.

Script 16 : Création d'un Diagramme en Secteurs pour la Répartition de 'Close' par 'Low'

Objectif : Utiliser Matplotlib pour créer un diagramme en secteurs représentant la répartition des valeurs de la colonne 'Close' en fonction des catégories de la colonne 'Low', offrant une visualisation claire de cette répartition.

Script 17 : Diagramme en Secteurs avec Pourcentages et Titre

Objectif : Améliorer le diagramme en secteurs en ajoutant des pourcentages pour chaque tranche, un angle de départ spécifié, et un titre, afin d'enrichir la présentation de la répartition de 'Close' par 'Low'.

Script 18 : Recommandation d'Investissement basée sur le Volume

Objectif : Vérifier si les valeurs de la colonne 'Volume' sont inférieures à 100 000 et recommander d'investir dans les périodes correspondantes. Afficher les données des périodes où le volume était en dessous du seuil spécifié.

En résumé, ces scripts couvrent un large éventail de tâches, de l'importation et du nettoyage des données à la visualisation avancée et aux analyses statistiques, fournissant ainsi un ensemble complet d'outils pour explorer et comprendre les données financières.

Partie pratique

1. Manipulation des données : a)

In [3]:
```
#importer des donnees
titanicData = pd.read_csv("Yahoo-Finance.csv", encoding="UTF-8")
titanicData
```

Out[3]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2022-12-12	24.480000	24.480000	24.030001	24.209999	23.819849	77900
1	2022-12-13	25.110001	26.790001	25.040001	26.350000	25.925364	275400
2	2022-12-14	26.290001	26.900000	25.629999	25.870001	25.453098	184800
3	2022-12-15	25.639999	26.049999	25.049999	25.820000	25.403904	153600
4	2022-12-16	25.780001	27.840000	25.760000	27.610001	27.165060	340700
...	...	...	...	...	...	...	...
245	2023-12-04	23.150000	23.660000	23.150000	23.510000	23.510000	85400
246	2023-12-05	23.309999	23.520000	22.889999	23.430000	23.430000	138800
247	2023-12-06	23.770000	24.100000	23.530001	23.570000	23.570000	83600
248	2023-12-07	23.570000	24.580000	23.320000	24.559999	24.559999	134500
249	2023-12-08	24.500000	25.309999	24.500000	24.910000	24.910000	83900

250 rows × 7 columns

Commentaire de code :

1. Import pandas as pd

Cette ligne importe la bibliothèque pandas en utilisant l'alias 'pd' pour un accès facile à ses fonctions.

2. Titanic Data = pd.read_csv("Yahoo-Finance.csv", encoding="UTF-8")

Ici, on utilise la fonction 'read_csv' de pandas pour lire le fichier CSV "Yahoo-Finance.csv" et stocker les données dans la variable 'titanicData'. Le paramètre 'encoding="UTF-8"' est utilisé pour spécifier l'encodage des caractères du fichier.

3. TitanicData

Cette ligne affiche le contenu de la variable 'titanicData', ce qui permet de visualiser l'ensemble de données importé.

Commentaire de résultat :

Les données financières comprennent sept colonnes, fournissant des informations telles que la date d'enregistrement, le prix d'ouverture, le prix le plus élevé, le prix le plus bas, le prix de clôture, le prix de clôture ajusté et le volume d'actions échangées pour chaque journée de trading. Par exemple, la première ligne indique que le 12 décembre 2022, le prix d'ouverture était de 24,48, le prix le plus élevé était de 24,48, le prix le plus bas était de 24,03, le prix de clôture était de 24,21, le prix de clôture ajusté était de 23,82 et le volume était de 77 900.

```
In [4]: print(titanicData.isnull().sum())
```

```
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

Commentaire de code :

Le code **print(titanicData.isnull().sum())** en Python utilise la bibliothèque Pandas pour afficher le nombre de valeurs manquantes dans chaque colonne du jeu de données **titanicData**. Voici une explication détaillée de chaque partie du code :

1. **titanicData.isnull()**: Cette partie du code utilise la méthode **isnull ()** pour vérifier chaque élément du jeu de données et renvoyer une valeur booléenne indiquant si l'élément est manquant (True) ou non (False).

2. **sum()**: Ensuite, la méthode **sum()** est appliquée à la sortie de **isnull()** pour compter le nombre de valeurs manquantes dans chaque colonne. Cela se fait en traitant les valeurs manquantes comme 1 et les valeurs non manquantes comme 0, puis en additionnant ces valeurs pour chaque colonne.
3. **print()**: Enfin, la fonction **print()** est utilisée pour afficher le résultat du décompte des valeurs manquantes dans chaque colonne.

    ⇨ Ce code est utile pour évaluer rapidement l'intégrité des données et identifier les colonnes qui ont des valeurs manquantes, ce qui est important dans le processus d'analyse de données et de prétraitement des données.

Commentaire de résultat :

Le résultat affiché indique qu'il n'y a aucune valeur manquante dans le jeu de données. Chaque colonne est suivie de zéro, ce qui signifie qu'aucune donnée n'est manquante. Ainsi, le jeu de données est complet et ne nécessite pas de traitement supplémentaire pour les valeurs manquantes. Cela est rassurant car l'absence de valeurs manquantes facilite l'analyse et l'utilisation des données pour des tâches telles que la modélisation et la visualisation.

```
In [5]: print(titanicData['Open'].head(50).isnull())
```

```
0     False
1     False
2     False
3     False
4     False
5     False
6     False
7     False
8     False
9     False
10    False
11    False
12    False
13    False
14    False
15    False
16    False
17    False
18    False
19    False
20    False
21    False
22    False
23    False
24    False
25    False
26    False
27    False
28    False
29    False
30    False
31    False
32    False
33    False
34    False
35    False
36    False
37    False
38    False
39    False
40    False
41    False
42    False
43    False
44    False
45    False
46    False
47    False
48    False
49    False
Name: Open, dtype: bool
```


Commentaire code

- **TitanicData**: Un Data Frame supposé contenir des données liées au Titanic.
- **['Open']**: Sélectionne la colonne 'Open' du Data Frame.
- **.Head(50)**: Retourne les 50 premières lignes de la colonne sélectionnée.
- **.isnull()**: Indique True pour les valeurs nulles et False pour les valeurs non nulles dans la colonne.
- **print()**: Affiche le résultat dans la console.

Ainsi, cette ligne de code affiche un ensemble de valeurs booléennes pour les 50 premières lignes de la colonne 'Open', indiquant si chaque valeur est nulle (True) ou non nulle (False). Cela est utile pour repérer les données manquantes dans cette colonne spécifique.

Commentaire de résultat

Le résultat affiche une série de valeurs booléennes (True/False) pour les 50 premières lignes de la colonne 'Open' du Data Frame. Chaque ligne indique si la valeur correspondante dans la colonne 'Open' est nulle (False) ou non nulle (True). Dans ce cas, toutes les valeurs semblent être non nulles (False), ce qui suggère qu'il n'y a pas de données manquantes dans les 50 premières lignes de la colonne 'Open'. Cela peut être vérifié visuellement en parcourant les lignes et en observant les résultats "False" pour chaque ligne, indiquant l'absence de valeurs nulles dans cette portion spécifique des données.

1.b. filtrage

In [6]:
```
titanicData = titanicData[titanicData['Volume'] > 20000]
titanicData.head(50)
```

Out[6]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2022-12-12	24.480000	24.480000	24.030001	24.209999	23.819849	77900
1	2022-12-13	25.110001	26.790001	25.040001	26.350000	25.925364	275400
2	2022-12-14	26.290001	26.900000	25.629999	25.870001	25.453098	184800
3	2022-12-15	25.639999	26.049999	25.049999	25.820000	25.403904	153600
4	2022-12-16	25.780001	27.840000	25.760000	27.610001	27.165060	340700
5	2022-12-19	27.510000	27.510000	26.480000	26.650000	26.220528	157900
6	2022-12-20	26.490000	27.290001	26.150000	26.959999	26.525534	176700
7	2022-12-21	26.950001	27.680000	26.950001	27.170000	26.732149	134100
8	2022-12-22	26.950001	27.299999	26.430000	27.299999	26.860052	89400
9	2022-12-23	27.000000	27.709999	27.000000	27.469999	27.027313	87700
10	2022-12-27	27.510000	27.510000	26.850000	27.139999	26.702633	52900
11	2022-12-28	27.190001	27.719999	27.100000	27.200001	26.761665	135800
12	2022-12-29	27.180000	27.700001	27.010000	27.629999	27.184734	75700
13	2022-12-30	27.540001	27.889999	27.350000	27.540001	27.096186	85900
14	2023-01-03	27.600000	28.620001	27.600000	28.559999	28.099749	161600
15	2023-01-04	28.719999	29.330000	28.559999	29.129999	28.660561	82700
16	2023-01-05	29.070000	29.150000	28.350000	28.940001	28.473625	139000

17	2023-01-06	29.260000	30.299999	29.080000	30.030001	29.546059	67400
18	2023-01-09	30.090000	30.320000	29.250000	29.270000	28.798309	82500
19	2023-01-10	29.000000	30.290001	29.000000	30.230000	29.742836	62800
20	2023-01-11	30.330000	30.750000	30.059999	30.360001	29.870741	58100
21	2023-01-12	30.719999	30.719999	30.059999	30.209999	29.723158	68600
22	2023-01-13	30.080000	30.440001	29.799999	30.270000	29.782192	74700
23	2023-01-17	30.330000	31.240000	30.330000	30.559999	30.067518	77600
24	2023-01-18	30.760000	30.900000	30.059999	30.110001	29.624769	54000
25	2023-01-19	29.760000	30.480000	29.510000	30.370001	29.880579	52400
26	2023-01-20	30.610001	30.770000	30.110001	30.570000	30.077356	63500
27	2023-01-23	30.719999	31.230000	30.370001	30.650000	30.156067	68100
28	2023-01-24	31.730000	31.730000	30.709999	30.850000	30.352846	98700
29	2023-01-25	30.700001	30.850000	30.340000	30.850000	30.352846	47700
30	2023-01-26	31.000000	31.530001	30.700001	31.490000	30.982533	48000
31	2023-01-27	31.410000	31.670000	31.190001	31.290001	30.785753	39800
32	2023-01-30	31.139999	31.350000	30.680000	31.100000	30.598816	63400
33	2023-01-31	31.219999	32.459999	31.219999	32.419998	31.897539	89300
34	2023-02-01	32.490002	32.869999	31.850000	32.410000	31.887707	76300
35	2023-02-02	32.779999	33.090000	32.369999	32.700001	32.173031	63600
36	2023-02-03	32.340000	33.110001	32.099998	32.660000	32.245106	106400
37	2023-02-06	32.669998	32.980000	32.169998	32.490002	32.077263	124000
38	2023-02-07	32.389999	32.389999	31.709999	32.200001	31.790947	69800
39	2023-02-08	32.130001	32.130001	31.270000	31.400000	31.001110	67800
40	2023-02-09	31.680000	31.809999	30.650000	30.680000	30.290257	73800
41	2023-02-10	30.760000	31.709999	30.620001	31.620001	31.218315	74800
42	2023-02-13	31.660000	32.529999	31.549999	32.500000	32.087139	82800
43	2023-02-14	32.380001	32.980000	32.310001	32.730000	32.314213	78000
44	2023-02-15	32.770000	34.320000	32.770000	34.290001	33.854401	120900
45	2023-02-16	33.849998	34.180000	33.459999	33.849998	33.419987	92200
46	2023-02-17	33.980000	34.230000	33.389999	33.480000	33.054688	90600
47	2023-02-21	33.060001	33.709999	32.509998	32.689999	32.274719	131300
48	2023-02-22	32.939999	33.230000	32.029999	33.009998	32.590656	85600
49	2023-02-23	33.009998	35.990002	32.799999	35.459999	35.009537	258800

Commentaire de code :

1. **TitanicData**: Il s'agit du Data Frame qui semble être lié aux données du Titanic, bien que le nom puisse être inapproprié pour les données financières (voir l'exemple précédent).
2. **['Volume']**: Cela sélectionne la colonne 'Volume' du Data Frame.
3. **TitanicData['Volume'] > 20000**: C'est une condition de filtrage qui sélectionne uniquement les lignes où la valeur dans la colonne 'Volume' est supérieure à 20000.
4. **TitanicData[titanicData['Volume'] > 20000]**: Cela applique le filtre au DataFrame, ne conservant que les lignes qui satisfont à la condition spécifiée.
5. **.head(50)**: Cette méthode affiche les 50 premières lignes du DataFrame résultant, montrant ainsi les données filtrées.

Ainsi, le code filtre le DataFrame 'titanicData' pour inclure uniquement les lignes où la valeur dans la colonne 'Volume' est supérieure à 20000, puis affiche les 50 premières lignes du DataFrame filtré. Cela permet de visualiser les données après l'application du filtre de condition.

Commentaire de résultat :

Le résultat affiche les 50 premières lignes du DataFrame 'titanicData' après avoir appliqué un filtre. Ce filtre sélectionne uniquement les lignes où la valeur dans la colonne 'Volume' est supérieure à 20 000. Voici une brève interprétation du résultat :

Les données montrent les informations pour chaque jour, notamment la date, les valeurs d'ouverture, de clôture, de plus haut, de plus bas, d'ajustement, et le volume des transactions. Après l'application du filtre (> 20 000 dans la colonne 'Volume'), les lignes affichées représentent les jours où le volume des transactions était significativement élevé, dépassant le seuil de 20 000. Cela peut être utile pour se concentrer sur les jours de forte activité commerciale.

Tri

In [7]:
```
titanicData = pd.DataFrame(titanicData).sort_values(by='Date', ascending=True)
titanicData.head(100)
```

Out[7]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2022-12-12	24.480000	24.480000	24.030001	24.209999	23.819849	77900
1	2022-12-13	25.110001	26.790001	25.040001	26.350000	25.925364	275400
2	2022-12-14	26.290001	26.900000	25.629999	25.870001	25.453098	184800
3	2022-12-15	25.639999	26.049999	25.049999	25.820000	25.403904	153600
4	2022-12-16	25.780001	27.840000	25.760000	27.610001	27.165060	340700
...	...	...	...	...	...	...	...
95	2023-05-01	28.799999	29.740000	28.799999	29.690001	29.312832	59000
96	2023-05-02	29.389999	29.389999	28.469999	28.959999	28.592108	132900
97	2023-05-03	29.000000	29.360001	27.830000	28.090000	27.733158	138700
98	2023-05-04	29.170000	29.170000	26.120001	26.740000	26.400307	187900
99	2023-05-05	26.930000	27.709999	26.590000	27.590000	27.355108	146900

100 rows × 7 columns

Commentaire de code :

1. **pd.DataFrame(titanicData)**: Cela convertit les données existantes (supposées être dans un format compatible avec un DataFrame) en un nouveau DataFrame nommé 'titanicData'. Cependant, le nom de la variable 'titanicData' peut être trompeur ici, car il suggère des données liées au Titanic, alors que les exemples précédents semblent être des données financières.
2. **.sort_values(by='Date', ascending=True)**: Cette méthode trie le DataFrame par la colonne 'Date' par ordre croissant. Cela permet de réorganiser les données chronologiquement en fonction de la date.

3. **.head(100)**: Cette méthode affiche les 100 premières lignes du DataFrame trié.

Ainsi, le code crée un nouveau DataFrame à partir des données existantes, le trie par la colonne 'Date' dans l'ordre croissant, puis affiche les 100 premières lignes du DataFrame trié. Cela permet de visualiser les données dans un ordre chronologique croissant en fonction de la date.

Commentaire de résultat :

Le résultat affiche les 100 premières lignes du DataFrame trié par date, avec les colonnes 'Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', et 'Volume'. Voici une brève interprétation du résultat :

Les données semblent représenter des informations financières sur une période de temps allant du 12 décembre 2022 au 5 mai 2023. Chaque ligne correspond à un jour donné et fournit des détails tels que les prix d'ouverture, de clôture, les valeurs maximales (High) et minimales (Low), le volume des transactions, et la valeur ajustée (Adj Close). Le DataFrame est trié par ordre croissant en fonction de la date, permettant une visualisation chronologique des données.

Le code peut être utile pour analyser l'évolution temporelle des variables financières présentées dans le DataFrame.

In [8]:
```
titanicData = pd.DataFrame(titanicData).sort_values(by='Date', ascending=False)
titanicData.head(200)
```

Out[8]:

	Date	Open	High	Low	Close	Adj Close	Volume
249	2023-12-08	24.500000	25.309999	24.500000	24.910000	24.910000	83900
248	2023-12-07	23.570000	24.580000	23.320000	24.559999	24.559999	134500
247	2023-12-06	23.770000	24.100000	23.530001	23.570000	23.570000	83600
246	2023-12-05	23.309999	23.520000	22.889999	23.430000	23.430000	138800
245	2023-12-04	23.150000	23.660000	23.150000	23.510000	23.510000	85400
...	...	...	...	...	...	...	...
54	2023-03-02	33.720001	34.130001	33.470001	33.880001	33.449608	91700
53	2023-03-01	33.919998	34.509998	32.910000	33.959999	33.528587	124000
52	2023-02-28	33.980000	34.340000	33.740002	33.919998	33.489094	114700
51	2023-02-27	34.400002	34.759998	33.799999	34.020000	33.587826	101300
50	2023-02-24	35.330002	35.330002	33.439999	34.020000	33.587826	166900

200 rows × 7 columns

Commentaire de code :

1. **pd.DataFrame(titanicData)**: Cela convertit les données existantes (supposées être dans un format compatible avec un DataFrame) en un nouveau DataFrame nommé 'titanicData'. Cependant, le nom de la variable 'titanicData' peut être trompeur ici, car il suggère des données liées au Titanic, alors que les exemples précédents semblent être des données financières.
2. **.sort_values(by='Date', ascending=False)**: Cette méthode trie le DataFrame par la colonne 'Date' par ordre décroissant. Cela permet de réorganiser les données chronologiquement en fonction de la date, mais cette fois-ci dans l'ordre décroissant.
3. **.head(200)**: Cette méthode affiche les 200 premières lignes du DataFrame trié.

Ainsi, le code crée un nouveau DataFrame à partir des données existantes, le trie par la colonne 'Date' dans l'ordre décroissant, puis affiche les 200 premières lignes du DataFrame trié. Cela permet de visualiser les données dans un ordre chronologique décroissant en fonction de la date.

Commentaire de résultat

Le résultat affiche les 200 premières lignes du DataFrame trié par date, avec les colonnes 'Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', et 'Volume'. Voici une brève interprétation du résultat :

Les données semblent représenter des informations financières sur une période allant du 24 février 2023 au 8 décembre 2023. Chaque ligne correspond à un jour donné et fournit des détails tels que les prix d'ouverture, de clôture, les valeurs maximales (High) et minimales (Low), le volume des transactions, et la valeur ajustée (Adj Close). Le DataFrame est trié par ordre décroissant en fonction de la date, permettant une visualisation chronologique inversée des données.

Le code peut être utile pour analyser l'évolution temporelle des variables financières présentées dans le DataFrame, en mettant l'accent sur les jours les plus récents.

Groupby () :

In [9]:
```
grouped_data = titanicData.groupby('Close')
# Afficher les premières lignes de chaque groupe après le groupby
for close_value, group in grouped_data:
    print(f"Valeur 'Close': {close_value}")
    print(group.head())  # Afficher les premières lignes de chaque groupe
    print("-----------------------------------")
```

```
Valeur 'Close': 20.15
           Date   Open   High        Low  Close  Adj Close  Volume
231  2023-11-13  20.91  21.52  20.110001  20.15      20.15  221100
-----------------------------------
Valeur 'Close': 20.969998999999998
           Date       Open       High        Low      Close  Adj Close  Volume
230  2023-11-10  21.280001  21.280001  20.280001  20.969999  20.969999  235200
-----------------------------------
Valeur 'Close': 21.17
           Date   Open   High        Low  Close  Adj Close  Volume
229  2023-11-09  20.66  22.07  18.059999  21.17      21.17  514700
-----------------------------------
Valeur 'Close': 21.18
           Date       Open       High    Low  Close  Adj Close  Volume
223  2023-11-01  21.459999  21.459999  20.91  21.18  21.075624  106600
220  2023-10-27  21.379999  21.469999  21.00  21.18  21.075624  238300
-----------------------------------
Valeur 'Close': 21.4
           Date       Open       High   Low  Close  Adj Close  Volume
234  2023-11-16  21.610001  21.870001  21.1   21.4       21.4  170400
-----------------------------------
Valeur 'Close': 21.450001
           Date   Open   High        Low      Close  Adj Close  Volume
221  2023-10-30  21.48  21.92  21.280001  21.450001  21.344294  115800
-----------------------------------
Valeur 'Close': 21.52
           Date  Open  High    Low  Close  Adj Close  Volume
219  2023-10-26  22.0  22.0  21.34  21.52  21.413948   80700
-----------------------------------
```

Commentaire de code :

1. **titanicData.groupby('Close')**: Cette étape crée des groupes en regroupant les lignes du DataFrame 'titanicData' en fonction des valeurs uniques dans la colonne 'Close'. Cela signifie que toutes les lignes ayant la même valeur dans la colonne 'Close' seront regroupées ensemble.
2. **For close_value, group in grouped_data:** Cela itère à travers chaque groupe résultant du groupement. **close_value** est la valeur unique de 'Close' pour le groupe actuel, et **group** est le DataFrame correspondant à ce groupe.
3. **print(f"Valeur 'Close': {close_value}")**: Cela affiche la valeur de 'Close' pour le groupe actuel.
4. **print(group.head())**: Cela affiche les premières lignes du DataFrame associé à ce groupe, permettant d'examiner une partie des données pour chaque valeur unique de 'Close'.
5. **print("----------")**: Cette ligne ajoute une séparation visuelle entre les différents groupes.

En résumé, le code permet d'inspecter les premières lignes de chaque groupe créé par le groupement des données en fonction des valeurs uniques de la colonne 'Close'. Cela peut être utile pour comprendre la distribution des données en fonction des différentes valeurs de clôture ('Close').

Commentaire de résultat :

Le code effectue un regroupement des données du DataFrame **titanicData** en fonction de la colonne 'Close'. En analysant les premières lignes de chaque groupe obtenu, on peut tirer quelques observations :

1. **Valeur 'Close': 20.15** : Pour la valeur 'Close' de 20.15, la date associée est le 13 novembre 2023. Le prix d'ouverture (Open) était de 20.91, et le prix de clôture (Close) était de 20.15.

2. **Valeur 'Close': 20.969998999999998** : Pour la valeur 'Close' de 20.97, la date associée est le 10 novembre 2023. Le prix d'ouverture était de 21.28, et le prix de clôture était de 20.97.

3. **Valeur 'Close': 21.17** : Pour la valeur 'Close' de 21.17, la date associée est le 9 novembre 2023. Le prix d'ouverture était de 20.66, et le prix de clôture était de 21.17. Le volume de transactions était élevé à 514,700.

4. **Valeur 'Close': 21.18** : Pour la valeur 'Close' de 21.18, les deux premières lignes montrent des transactions le 1er novembre et le 27 octobre 2023, avec des prix de clôture identiques de 21.18.

5. **Valeur 'Close': 21.4** : Pour la valeur 'Close' de 21.4, la date associée est le 16 novembre 2023. Le prix d'ouverture était de 21.61, et le prix de clôture était de 21.4.

6. **Valeur 'Close': 21.450001** : Pour la valeur 'Close' de 21.45, la date associée est le 30 octobre 2023. Le prix d'ouverture était de 21.48, et le prix de clôture était de 21.45.

7. **Valeur 'Close': 21.52** : Pour la valeur 'Close' de 21.52, la date associée est le 26 octobre 2023. Le prix d'ouverture était de 22.0, et le prix de clôture était de 21.52.

En résumé, ces informations fournissent un aperçu des variations des prix de clôture pour différentes valeurs de 'Close', avec des détails sur les dates, les prix d'ouverture et de clôture, ainsi que les volumes de transactions associés. Cela peut être utile pour une analyse plus approfondie des tendances du marché ou des comportements spécifiques de certains actifs.

2) Analyse et visualisation :

a.1) calcul des statistiques descriptives avec pandas : (moyennes, médianes, écarts types,)

In [10]:
```
# Calcul des statistiques descriptives avec Pandas
close_column = titanicData['Close']

# Moyenne
mean_close = close_column.mean()

# Médiane
median_close = close_column.median()

# Écart-type
std_dev_close = close_column.std()

# Affichage des résultats
print(f"Moyenne : {mean_close}")
print(f"Médiane : {median_close}")
print(f"Écart-type : {std_dev_close}")
```

```
Moyenne : 28.78099995199999
Médiane : 29.1099995
Écart-type : 3.5776269941812817
```

Commentaire de code :

Ce code utilise la bibliothèque Pandas pour calculer certaines statistiques descriptives à partir de la colonne 'Close' du DataFrame **titanicData**. Voici une brève explication des éléments entre parenthèses :

1. **close_column = titanicData['Close']**: Il extrait la colonne 'Close' du DataFrame **titanicData** et la stocke dans la variable **close_column**. Cette colonne contient probablement les prix de clôture d'un actif financier, comme indiqué dans le précédent exemple.
2. **mean_close = close_column.mean()**: Il calcule la moyenne des valeurs de la colonne 'Close' à l'aide de la méthode **mean()** de Pandas et stocke le résultat dans la variable **mean_close**.
3. **median_close = close_column.median()**: Il calcule la médiane des valeurs de la colonne 'Close' à l'aide de la méthode **median()** de Pandas et stocke le résultat dans la variable **median_close**.

4. **std_dev_close = close_column.std()**: Il calcule l'écart-type des valeurs de la colonne 'Close' à l'aide de la méthode **std()** de Pandas et stocke le résultat dans la variable **std_dev_close**.
5. **Affichage des résultats** : Enfin, il imprime les résultats, affichant la moyenne, la médiane et l'écart-type des valeurs de la colonne 'Close'.

En résumé, ces calculs fournissent des mesures statistiques importantes pour comprendre la distribution des prix de clôture de l'actif financier, notamment la moyenne qui donne une indication de la tendance centrale, la médiane qui représente la valeur au milieu de la distribution, et l'écart-type qui mesure la dispersion des valeurs par rapport à la moyenne. Ces statistiques sont utiles pour caractériser la variabilité et la tendance générale des données financières.

Commentaire de résutat :

Les résultats des statistiques descriptives pour la colonne 'Close' sont les suivants :

- **Moyenne : 28.781** : La moyenne représente la valeur centrale des prix de clôture. Dans ce cas, elle est d'environ 28.78, ce qui suggère que, en moyenne, les prix de clôture tendent à tourner autour de cette valeur.
- **Médiane : 29.11** : La médiane est la valeur qui divise la distribution en deux parties égales. Avec une médiane de 29.11, cela indique que la moitié des prix de clôture sont inférieurs à cette valeur et l'autre moitié sont supérieurs. Comparée à la moyenne, la médiane donne une indication de la symétrie de la distribution.
- **Écart-type : 3.578** : L'écart-type mesure la dispersion des données par rapport à la moyenne. Un écart-type plus élevé suggère une plus grande variabilité des prix de clôture par rapport à la moyenne. Dans ce cas, un écart-type d'environ 3.578 indique une certaine variabilité dans les prix de clôture autour de la moyenne.

En résumé, ces statistiques suggèrent que les prix de clôture ont une certaine variabilité, comme indiqué par l'écart-type. La moyenne et la médiane sont relativement proches, ce qui peut indiquer une distribution approximativement symétrique. Ces mesures sont

utiles pour avoir une compréhension globale de la distribution des prix de clôture de l'actif financier.

a.2) Calculer des statistiques descriptives avec Numpty (moyennes, médianes, écarts types.)

```
In [11]: import numpy as np
         close_column = titanicData['Close']
         np.mean(close_column)

Out[11]: 28.78099995199999

In [12]: close_column = titanicData['Close']
         np.median(close_column)

Out[12]: 29.1099995

In [13]: close_column = titanicData['Close']
         np.std(close_column)

Out[13]: 3.5704645705925464
```

Commentaire de code et de résultat : (moyenne)

Ce code utilise la bibliothèque NumPty pour calculer la moyenne des valeurs dans la colonne 'Close' du DataFrame **titanicData**. Voici une brève explication des éléments dans la parenthèse :

1. **import numpty as np**: Cette ligne importe la bibliothèque NumPty sous l'alias 'np'. NumPty est une bibliothèque très utilisée en Python pour les calculs numériques, y compris les opérations sur des tableaux (arrays) et des matrices.
2. **close_column = titanicData['Close']**: Cette ligne extrait la colonne 'Close' du DataFrame **titanicData** et l'assigne à la variable **close_column**. La colonne 'Close' contient probablement les prix de clôture d'un actif financier.
3. **np.mean(close_column)**: Cette ligne utilise la fonction **mean** de NumPty pour calculer la moyenne des valeurs de la colonne 'Close'. En d'autres termes, elle donne la valeur moyenne des prix de clôture.
4. **Résultat 28.78099995199999**: C'est le résultat de la moyenne calculée. Environ 28.78 est la moyenne des prix de clôture dans la colonne 'Close'.

En résumé, ce code avec NumPty réalise le même calcul que celui avec Pandas pour la moyenne des prix de clôture.

Commentaire de code et de résultat : (médiane)

Le code utilise la bibliothèque NumPty pour calculer la médiane des valeurs dans la colonne 'Close' du DataFrame **titanicData**. Voici une brève explication des éléments entre les parenthèses :

1. **close_column = titanicData['Close']**: Cette ligne extrait la colonne 'Close' du DataFrame **titanicData** et l'assigne à la variable **close_column**. La colonne 'Close' contient probablement les prix de clôture d'un actif financier.
2. **np.median(close_column)**: Cette ligne utilise la fonction **median** de NumPty pour calculer la médiane des valeurs de la colonne 'Close'. En d'autres termes, elle donne la valeur médiane des prix de clôture.
3. **Résultat 29.1099995**: C'est le résultat de la médiane calculée. Environ 29.11 est la médiane des prix de clôture dans la colonne 'Close'.

En résumé, ce code utilise NumPty pour calculer la médiane des prix de clôture dans la colonne 'Close' du DataFrame **titanicData**. La médiane représente la valeur centrale de la distribution, indiquant que la moitié des prix de clôture sont inférieurs à 29.11 et l'autre moitié sont supérieurs.

Commentaire de code et de résultat : (écart type)

Le code utilise la bibliothèque NumPty pour calculer l'écart-type des valeurs dans la colonne 'Close' du DataFrame **titanicData**. Voici une brève explication des éléments entre les parenthèses :

1. **close_column = titanicData['Close']**: Cette ligne extrait la colonne 'Close' du DataFrame **titanicData** et l'assigne à la variable **close_column**. La colonne 'Close' contient probablement les prix de clôture d'un actif financier.

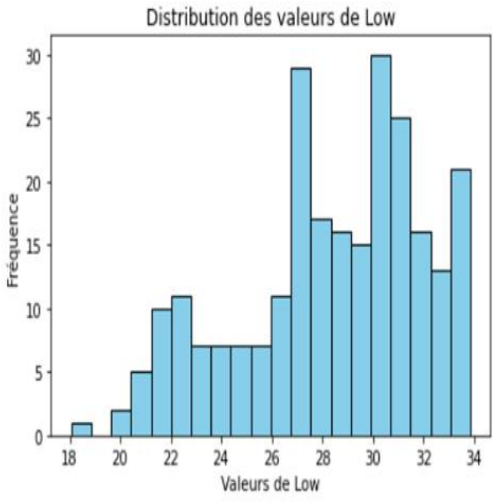
2. **np.std(close_column)**: Cette ligne utilise la fonction **std** de NumPty pour calculer l'écart-type des valeurs de la colonne 'Close'. L'écart-type mesure la dispersion des données par rapport à la moyenne.
3. **Résultat 3.5704645705925464**: C'est le résultat de l'écart-type calculé. Environ 3.57 est l'écart-type des prix de clôture dans la colonne 'Close'.

En résumé, cet écart-type indique la variabilité ou la dispersion des prix de clôture par rapport à la moyenne. Un écart-type plus élevé suggère une dispersion plus importante des valeurs autour de la moyenne, tandis qu'un écart-type plus faible indique une dispersion moindre. Dans ce contexte, 3.57 indique une certaine variabilité des prix de clôture par rapport à la moyenne de la colonne 'Close' de **titanicData**.

2. b) Création des graphiques à l'aide de Matplotlib et Seaborn

In [14]:
```
import matplotlib.pyplot as plt

# Création d'un histogramme pour la colonne 'Close'
titanicData['Low'].plot(kind='hist', bins=20, color='skyblue', edgecolor='black')
plt.xlabel('Valeurs de Low')
plt.ylabel('Fréquence')
plt.title('Distribution des valeurs de Low')
plt.show()
```



Matplotlib : Interprétation de graphique :

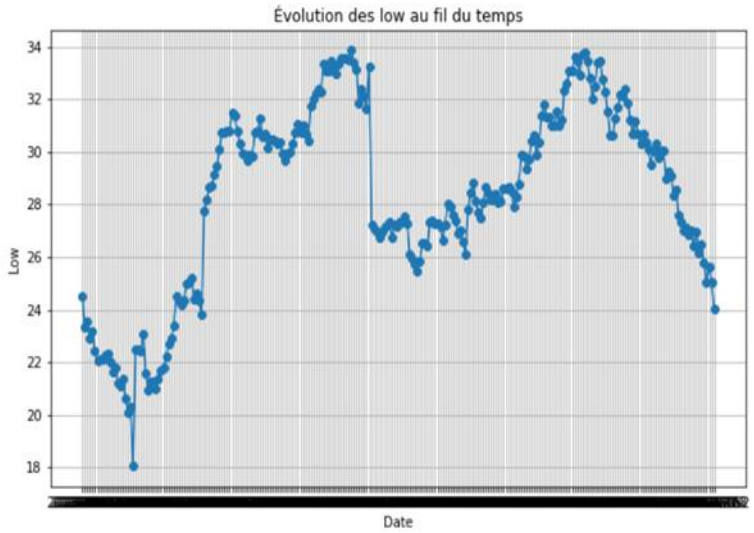
Ce code utilise la bibliothèque Matplotlib pour créer un histogramme des valeurs de la colonne 'Low' du DataFrame **titanicData**. Voici une brève explication de chaque élément entre les parenthèses :

1. **TitanicData['Low']**: Cette partie extrait la colonne 'Low' du DataFrame **titanicData**. La colonne 'Low' probablement représente les valeurs minimales des prix d'un actif financier.
2. **.plot(kind='hist', bins=20, color='skyblue', edgecolor='black')**: Cette partie utilise la méthode **plot** pour créer un histogramme. Les paramètres spécifiés sont :
    - **kind='hist'** : Indique que l'on souhaite créer un histogramme.
    - **bins=20** : Spécifie le nombre de bacs (intervalles) dans l'histogramme.
    - **color='skyblue'** : Définit la couleur des barres de l'histogramme.
    - **edgecolor='black'** : Définit la couleur de la bordure des barres.
3. **plt.xlabel('Valeurs de Low')**: Cette ligne ajoute une étiquette à l'axe des x, indiquant que l'histogramme représente les valeurs de la colonne 'Low'.
4. **plt.ylabel('Fréquence')**: Cette ligne ajoute une étiquette à l'axe des y, indiquant que l'axe vertical représente la fréquence des valeurs.
5. **plt.title('Distribution des valeurs de Low')**: Cette ligne ajoute un titre à l'histogramme, décrivant la distribution des valeurs de la colonne 'Low'.
6. **plt.show()**: Cette ligne affiche l'histogramme.

En résumé, le code crée un histogramme visuel qui représente la distribution des valeurs minimales (Low) dans le DataFrame **titanicData**. Cela permet d'avoir un aperçu de la répartition des valeurs basses des prix, mettant en évidence les tendances ou la concentration des données dans certaines plages de valeurs.

Autre graphique : (matplotlip)

```
In [15]: # Supposons que 'financial_data' est votre DataFrame avec une colonne 'Date' et une colonne 'low'
plt.figure(figsize=(10, 6))
plt.plot(titanicData['Date'], titanicData['Low'], marker='o', linestyle='-')
plt.xlabel('Date')
plt.ylabel('Low')
plt.title('Évolution des low au fil du temps')
plt.grid(True)
plt.show()
```



Interprétation de graphique :

Ce code utilise la bibliothèque Matplotlib pour créer un graphique de ligne représentant l'évolution des valeurs de la colonne 'Low' au fil du temps. Voici une brève explication de chaque élément entre les parenthèses :

1. **plt.figure(figsize=(10, 6))**: Cette ligne crée une nouvelle figure (graphique) avec une taille de 10 unités de largeur sur 6 unités de hauteur. Cela permet de spécifier les dimensions de la figure.
2. **plt.plot(titanicData['Date'], titanicData['Low'], marker='o', linestyle='-')**: Cette ligne trace le graphique de ligne. Les paramètres spécifiés sont :
    - **titanicData['Date']** : Les valeurs de l'axe des x, qui sont supposées représenter les dates.
    - **titanicData['Low']** : Les valeurs de l'axe des y, qui sont les valeurs minimales (Low) des prix.

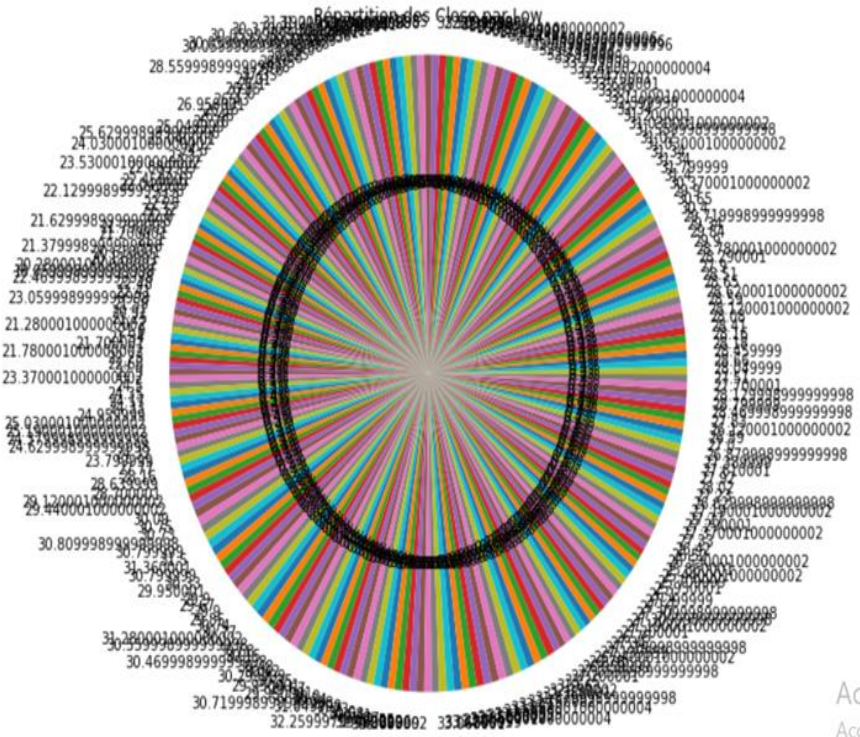
- **marker='o'** : Utilise des cercles comme marqueurs sur les points de données.
- **linestyle='-'** : Utilise une ligne solide pour connecter les points de données.

3. **plt.xlabel('Date')**: Cette ligne ajoute une étiquette à l'axe des x, indiquant que l'axe horizontal représente les dates.
4. **plt.ylabel('Low')**: Cette ligne ajoute une étiquette à l'axe des y, indiquant que l'axe vertical représente les valeurs minimales (Low) des prix.
5. **plt.title('Évolution des low au fil du temps')**: Cette ligne ajoute un titre au graphique, décrivant l'évolution des valeurs minimales des prix au fil du temps.
6. **plt.grid(True)**: Cette ligne ajoute une grille au graphique pour faciliter la lecture des valeurs.
7. **plt.show()**: Cette ligne affiche le graphique.

En résumé, le code crée un graphique de ligne qui illustre visuellement l'évolution des valeurs minimales (Low) des prix au fil du temps, permettant d'observer les tendances et les variations dans ces données.

Autre graphique : (matplotlip)

In [17]:
```
plt.figure(figsize=(8, 8))
plt.pie(titanicData['Close'], labels=titanicData['Low'], autopct='%1.1f%%', startangle=140)
plt.axis('equal')  # Pour que le diagramme soit un cercle plutôt qu'une ellipse
plt.title('Répartition des Close par Low')
plt.show()
```



Interprétation de graphique :

Ce code utilise la bibliothèque Matplotlib pour créer un diagramme circulaire (camembert) représentant la répartition des valeurs de la colonne 'Close' en fonction des valeurs de la colonne 'Low' dans le DataFrame **titanicData**. Voici une brève explication de chaque élément entre les parenthèses :

1. **plt.figure(figsize= (8, 8))**: Cette ligne crée une nouvelle figure (graphique) avec une taille de 8 unités de largeur sur 8 unités de hauteur. Cela permet de spécifier les dimensions de la figure.

2. **plt.pie(titanicData['Close'],labels=titanicData['Low'], autopct='%1.1f%%', startangle=140)**: Cette ligne crée le diagramme circulaire. Les paramètres spécifiés sont :

- **titanicData['Close']** : Les valeurs à représenter dans le camembert, provenant de la colonne 'Close'.
- **labels=titanicData['Low']** : Les étiquettes associées à chaque portion du camembert, provenant de la colonne 'Low'.
- **autopct='%1.1f%%'** : Le format pour afficher les pourcentages sur chaque portion du camembert.
- **startangle=140** : L'angle de départ pour le premier secteur du camembert.

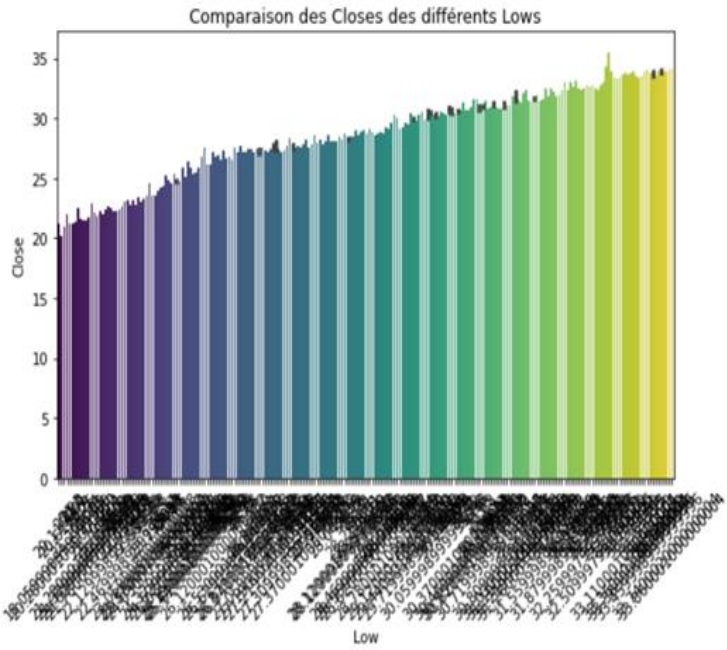
3. **plt.axis('equal')** : Cette ligne s'assure que le camembert est parfaitement circulaire plutôt qu'une ellipse, en maintenant les axes x et y égaux.
4. **plt.title('Répartition des Close par Low')** : Cette ligne ajoute un titre au graphique, décrivant la répartition des valeurs de 'Close' en fonction des valeurs de 'Low'.
5. **plt.show()** : Cette ligne affiche le graphique.

En résumé, le code crée un diagramme circulaire qui illustre la répartition des valeurs de 'Close' en fonction des valeurs de 'Low' dans le DataFrame **titanicData**. Chaque portion du camembert représente une catégorie de 'Low', avec les pourcentages affichés sur chaque portion pour indiquer la proportion de 'Close'.

Seaborn : interprétation de graphique :

```
import seaborn as sns

# Supposons que 'titanicData' est votre DataFrame avec une colonne 'Low' et une colonne 'Close'
plt.figure(figsize=(8, 6))
sns.barplot(x='Low', y='Close', data=titanicData, palette='viridis')
plt.xlabel('Low')
plt.ylabel('Close')
plt.title('Comparaison des Closes des différents Lows')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Ce code utilise la bibliothèque Seaborn pour créer un graphique à barres comparant les valeurs de la colonne 'Close' en fonction des valeurs de la colonne 'Low' dans le DataFrame **titanicData**. Voici une brève explication de chaque élément entre les parenthèses :

1. **plt.figure(figsize=(8, 6))**: Cette ligne crée une nouvelle figure (graphique) avec une taille de 8 unités de largeur sur 6 unités de hauteur. Cela permet de spécifier les dimensions de la figure.
2. **sns.barplot(x='Low', y='Close', data=titanicData, palette='viridis')**: Cette ligne crée un graphique à barres à l'aide de Seaborn. Les paramètres spécifiés sont :
    - **x='Low'** : Les valeurs de l'axe des x sont extraites de la colonne 'Low'.

- **y='Close'** : Les hauteurs des barres sont déterminées par les valeurs de la colonne 'Close'.
- **data=titanicData** : Les données proviennent du DataFrame **titanicData**.
- **palette='viridis'** : Utilise la palette de couleurs 'viridis' pour les barres.

3. **plt.xlabel('Low')**: Cette ligne ajoute une étiquette à l'axe des x, indiquant que l'axe horizontal représente les valeurs de la colonne 'Low'.
4. **plt.ylabel('Close')**: Cette ligne ajoute une étiquette à l'axe des y, indiquant que l'axe vertical représente les valeurs de la colonne 'Close'.
5. **plt.title('Comparaison des Closes des différents Lows')**: Cette ligne ajoute un titre au graphique, décrivant la comparaison entre les valeurs de 'Close' pour différents niveaux de 'Low'.
6. **plt.xticks(rotation=45)**: Cette ligne fait pivoter les étiquettes de l'axe des x de 45 degrés pour améliorer la lisibilité.
7. **plt.tight_layout()**: Cette ligne ajuste automatiquement la mise en page du graphique pour éviter que les étiquettes ne se chevauchent.
8. **plt.show()**: Cette ligne affiche le graphique.

En résumé, le code crée un graphique à barres qui met en évidence la relation entre les valeurs de 'Low' et 'Close' dans le DataFrame **titanicData**, utilisant la palette de couleurs 'viridis' pour représenter visuellement ces relations.

3) Prise de décision: a +b

In [18]:
```
# Supposons que 'titanicData' est votre DataFrame avec une colonne 'Volume'

# Vérifier si le volume est inférieur à 100000 pour recommander l'investissement
vol_seuil = 100000
investir = titanicData['Volume'] < vol_seuil

# Afficher les recommandations d'investissement
recommandations = titanicData.loc[investir]
print(recommandations)
```

```
          Date       Open       High        Low      Close  Adj Close  Volume
249 2023-12-08  24.500000  25.309999  24.500000  24.910000  24.910000   83900
247 2023-12-06  23.770000  24.100000  23.530001  23.570000  23.570000   83600
245 2023-12-04  23.150000  23.660000  23.150000  23.510000  23.510000   85400
242 2023-11-29  22.799999  23.030001  22.129999  22.309999  22.309999   97800
241 2023-11-28  22.600000  22.770000  22.100000  22.570000  22.570000   67100
..         ...        ...        ...        ...        ...        ...     ...
12  2022-12-29  27.180000  27.700001  27.010000  27.629999  27.184734   75700
10  2022-12-27  27.510000  27.510000  26.850000  27.139999  26.702633   52900
9   2022-12-23  27.000000  27.709999  27.000000  27.469999  27.027313   87700
8   2022-12-22  26.950001  27.299999  26.430000  27.299999  26.860052   89400
0   2022-12-12  24.480000  24.480000  24.030001  24.209999  23.819849   77900

[161 rows x 7 columns]
```

Commentaire de code :

Ce code effectue une vérification sur la colonne 'Volume' du DataFrame **titanicData** pour recommander ou non un investissement en fonction d'un seuil défini. Voici une brève explication de chaque élément entre les parenthèses :

1. **Vol_seuil = 100000** : Cette ligne définit un seuil de volume (100 000 dans ce cas) en dessous duquel une recommandation d'investissement serait générée.
2. **investir = titanicData['Volume'] < vol_seuil** : Cette ligne crée une série booléenne (**investir**) en comparant chaque valeur de la colonne 'Volume' à **vol_seuil**. Chaque valeur de la série est **True** si le volume est inférieur à 100 000, sinon **False**.
3. **Recommandations = titanicData.loc[investir]** : Cette ligne utilise l'indexation conditionnelle pour sélectionner uniquement les lignes du DataFrame **titanicData** où la condition définie dans **investir** est vraie. Cela crée un nouveau DataFrame (**recommandations**) contenant les données qui satisfont la condition.
4. **print(recommandations)** : Cette ligne affiche le DataFrame **recommandations**, c'est-à-dire les lignes où le volume est inférieur à 100 000.

En résumé, le code identifie les jours où le volume est inférieur à 100 000 et recommande potentiellement un investissement sur ces jours. Les recommandations sont ensuite affichées dans le DataFrame **recommandations**.

Commentaire de résultat :

Le résultat affiché est le DataFrame **recommandations**, qui contient les lignes de **titanicData** où la condition "Volume inférieur à 100 000" est vraie. Voici quelques commentaires sur ce résultat :

1. **Nombre de lignes** : Le DataFrame **recommandations** contient 161 lignes, chacune représentant un jour où le volume était inférieur à 100 000.
2. **Informations sur chaque jour** : Chaque ligne donne des informations spécifiques pour un jour donné, telles que la date, les prix d'ouverture, de clôture, le plus bas, le plus haut, l'ajustement des clôtures, et le volume de transactions.
3. **Potentiel d'investissement** : Ces jours avec un volume plus bas peuvent susciter l'intérêt des investisseurs, car des volumes plus faibles peuvent parfois indiquer des périodes de calme ou de consolidation sur le marché. Cependant, toute décision d'investissement devrait être prise en tenant compte d'autres facteurs et de l'analyse approfondie du contexte du marché.

En résumé, le DataFrame fournit une liste de jours où le volume était inférieur à 100 000, ce qui peut être considéré comme un critère pour potentiellement recommander l'investissement.

Partie théorique

Présentation des données :

Importer un ensemble de données financières significatives depuis une source externe, notamment Yahoo Finance. Ces données financières comprennent 7 colonnes :

- Date : La date à laquelle les données ont été enregistrées.
- Open : Le prix d'ouverture des actions pour la journée.
- High : Le prix le plus élevé atteint par les actions pendant la journée.
- Low : Le prix le plus bas atteint par les actions pendant la journée.
- Close : Le prix de clôture des actions pour la journée.
- Adj Close : Le prix de clôture ajusté des actions pour la journée.
- Volume : Le volume d'actions échangées pendant la journée.

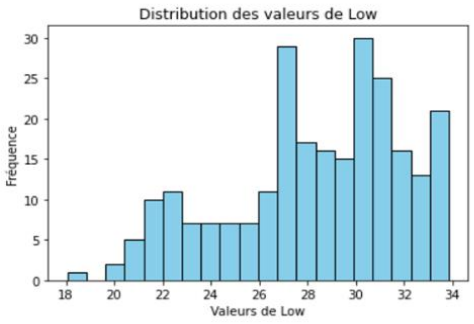
Chaque ligne représente les données pour une journée de trading spécifique.

Interprétation des visualisations graphiques :

Matplotlip

```
In [14]: import matplotlib.pyplot as plt

# Création d'un histogramme pour la colonne 'Close'
titanicData['Low'].plot(kind='hist', bins=20, color='skyblue', edgecolor='black')
plt.xlabel('Valeurs de Low')
plt.ylabel('Fréquence')
plt.title('Distribution des valeurs de Low')
plt.show()
```



L'histogramme de la colonne 'Low' du marché boursier représente la distribution des prix les plus bas au fil du temps. En observant les données fournies, voici une interprétation quantitative de l'évolution et des phases du marché :

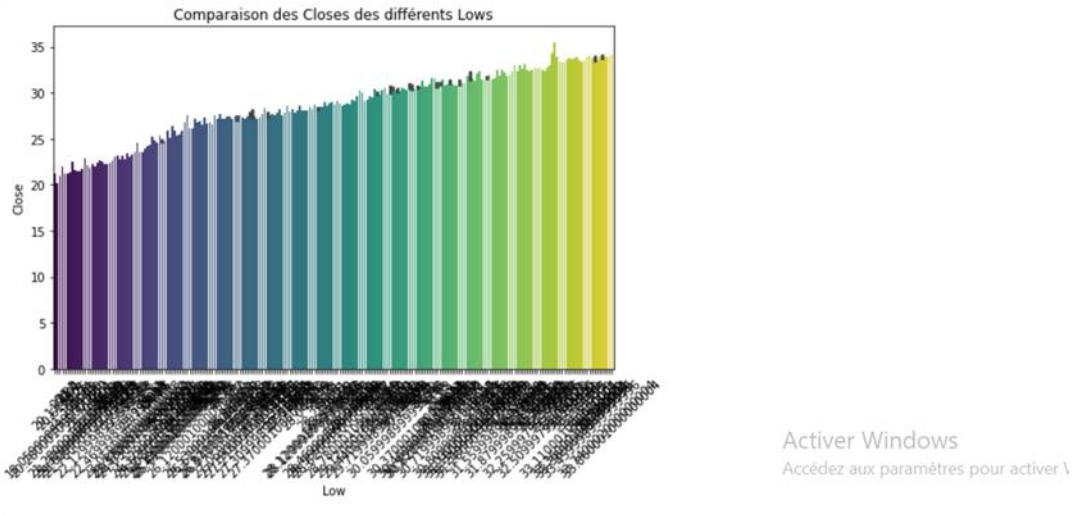
- Pic autour de 20-23: La majorité des observations se situent dans cette plage, indiquant des périodes où les prix les plus bas étaient relativement élevés, indiquant potentiellement des périodes de hausse ou d'optimisme sur le marché boursier.
- Plage de prix bas (23-26) : La majorité des observations se situent dans cette plage, indiquant des périodes où les prix les plus bas étaient relativement stables.
- Fluctuations entre 26 et 34 : Il y a des fluctuations régulières entre ces valeurs, représentant probablement des phases de volatilité ou d'ajustement du marché.

Cette interprétation suggère une évolution dynamique du marché avec des périodes de stabilité, de hausse, et de fluctuations. La densité des barres à différentes plages de valeurs permet de quantifier ces tendances et d'identifier les phases distinctes du marché boursier au fil du temps.

Seaborn

```
import seaborn as sns

# Supposons que 'titanicData' est votre DataFrame avec une colonne 'Low' et une colonne 'Close'
plt.figure(figsize=(8, 6))
sns.barplot(x='Low', y='Close', data=titanicData, palette='viridis')
plt.xlabel('Low')
plt.ylabel('Close')
plt.title('Comparaison des Closes des différents Lows')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



- La variation horizontale entre les barres indique des périodes où les valeurs de 'Low' restent relativement constantes.
- L'inclinaison ascendante des barres suggère une relation positive entre 'Low' et 'Close', montrant que, en général, lorsque le prix le plus bas ('Low') est élevé, le prix de clôture ('Close') a également tendance à être élevé.
- Les barres plus courtes peuvent indiquer des phases où malgré des variations importantes des prix les plus bas, les prix de clôture restent relativement stables.
- Les barres plus longues peuvent indiquer des périodes de volatilité où les variations des prix les plus bas ont un impact significatif sur les prix de clôture.

Conclusion générale

En conclusion, le projet de marchés boursiers adopte une approche complète dans l'analyse des données financières en utilisant le langage de programmation Python. En se concentrant sur des étapes clés telles que l'importation, la manipulation, l'exploration et la visualisation des données, le projet permet une compréhension approfondie des tendances et des comportements du marché boursier. Grâce à l'application de méthodes avancées d'analyse de données, il offre aux investisseurs et aux analystes financiers des informations significatives pour prendre des décisions éclairées et anticiper les évolutions du marché. Cette approche méthodologique garantit une exploration approfondie du paysage financier, contribuant ainsi à une meilleure compréhension des dynamiques du marché boursier et à l'identification d'opportunités stratégiques.