Pivot Tables in Pandas

Basic Syntax Example

`df.pivot_table(index='industry', columns='year', values='unemployed', aggfunc='mean')`

Common aggfunc

Single function: `'mean', 'sum', 'count', 'median', 'max', 'min'`, etc.

Multiple functions: `aggfunc=['mean', 'median']` (creates hierarchical column headers)

Different functions for different columns: `aggfunc={'User_Score': 'median', 'User_Count': 'sum'}`

Multiple Columns in values

`df.pivot_table(index='industry', values=['rate', 'unemployed'], aggfunc=['mean', 'median'])`

Hierarchical (Multi-Level) Indexing

You can pass multiple fields to index or columns, such as `index=['year', 'month']`.

After pivoting, slice rows or columns using `.loc` (e.g., `summary.loc[2000, 'mean']`).

Remember:

`index` = rows

`columns` = columns

`values` = what to aggregate

`aggfunc` = how to aggregate

Random Numbers in Python

Uniform Distribution

Use numpy to generate floats from 0 to 1:

`np.random.rand(n)` # 1D array of size n

`np.random.rand(m,n)` # 2D array of shape (m,n)

To get uniform in [a,b), do: `(b - a) * np.random.rand(n) + a`

Normal (Gaussian) Distribution

`np.random.randn(n)` # mean 0, std 1

Scale and shift for mean `m`, std `s`: `s * np.random.randn(n) + m`

Random Integers

`np.random.randint(low, high, size=(m,n))` # integers in [low, high)

Example: `np.random.randint(1, 6, size=5)`, output: `array([2, 5, 2, 5, 2])`

np.random.seed

`np.random.seed(123)` # sets seed for NumPy's random

Shuffling

`np.random.shuffle(my_array)` # in-place shuffle for NumPy arrays

Remember: setting a seed makes results reproducible, while shuffling rearranges elements in-place.

Data Cleaning in Pandas

Checking Data Types

Use `df.dtypes` to check column types. `object` dtype indicates mixed data types, which can be inefficient.

Standardizing Missing Values

Pandas only detects `NaN` (`np.nan`) and `None` as missing values. Use `df.replace(['N/A', '?', ''], np.nan)` to replace other representations with `NaN`.

Finding Missing Values

Use `df.isnull()` to check for missing values and `df.isnull().sum()` to count them per column.

Dropping Missing Values

Use `df.dropna()` to remove all rows with `NaN`.

To drop `NaN` only in specific columns, use `df.dropna(subset=['column_name'])`.

Filling Missing Values

Use `df.fillna(value)` to replace `NaN`.

To fill a column with its mean:

`df.fillna({"age": df.age.mean()})`

Standardizing Categorical Variables

Use `df.replace(["yes", "y"], "Y")` and `df.replace(["no", "NO"], "N")` to unify categorical values.

Removing Duplicate Rows

Use `df.drop_duplicates()` to remove duplicate rows.

For specific columns: `df.drop_duplicates(subset=['ID'])`.

Renaming Columns

Rename all columns: `df.columns = ['new_name1', 'new_name2']`.

Rename specific columns: `df.rename(columns={'old_name': 'new_name'})`.

Dropping Extra Columns or Rows

Drop a column: `df.drop(columns='column_name')`.

Drop rows by index: `df.drop(index=[1, 2])`.

Removing Outliers

Filter out values outside ±3 standard deviations:

`df[(df.age > mean - 3 * std) & (df.age < mean + 3 * std)]`

Ensure values make sense, e.g., `df[df.age >= 0]`.

K-Means in Python

Basic Syntax

```
from sklearn.cluster import KMeans
km = KMeans(n_clusters=4, random_state=42)
km.fit(df)
y_hat = km.predict(df)
```

Key Methods

```
km.fit(X)  # Train model
km.predict(X_new)  # Assign clusters
km.fit_predict(X)  # Fit + predict
km.cluster_centers_  # Cluster centroids
km.inertia_  # Sum of squared distances
km.n_iter_  # Iterations until convergence
```

Dimensionality of Prediction

```
X_new = np.array([[1, 2], [3, 4], [5, 6]])
# Shape: (3,2)
y = km.predict(X_new)  # Shape: (3,)
print(y)  # Example output: [1, 0, 1]
```

Limitations & Failure Cases

- **Assumes spherical clusters** → Fails on non-convex shapes.
- **Sensitive to outliers** → Can distort centroids.
- **Fixed k value** → Must predefine k, which may not be optimal.
- **Different initializations** → Can lead to different results.
- **Unequal cluster sizes/densities** → Struggles with imbalanced data.

PCA in Python

Basic Syntax

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(X)
pca.transform(X)
pca.explained_variance_ratio_
pca.explained_variance_
pca.n_components_
```

Common Methods

fit(X) – learns principal components from X
transform(X) – applies learned transformation to X
fit_transform(X) – fits + transforms in one step
inverse_transform(X) – reconstructs approximate original features

StandardScaler

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Standardizing data (mean=0, std=1) ensures all features have equal weight in PCA.

Example

```
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

df = pd.DataFrame({
    'A': [1, 2, 3],
    'B': [4, 5, 6],
    'C': [7, 8, 9]
})

scaler = StandardScaler()
df_std = scaler.fit_transform(df)

pca = PCA(n_components=x)
df_pca = pca.fit_transform(df_std)

print(pca.explained_variance_ratio_)
print(df_pca)
```

Limitations

Loses interpretability (principal components are linear mixes of features)
Sensitive to scaling (always standardize first)
Captures only linear relationships
Not designed for categorical features without proper encoding
May overlook important lower-variance dimensions