with Python

NumPy Pandas

Table of contents

1

1.1

-
- **Python**
-
- **NumPy** -
- **Pandas** -
- **Matplotlib/Seaborn** -
- **Scikit-learn** -

```
#
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# 1.
# 2.
# 3.      EDA
# 4.
# 5.
```

1.2

RevealJS

:

- : Page Up/Down
- / :

- : ↑↓ ←→
- :

– ESC -

— F -
— S -

1.3 NumPy -

```
import numpy as np

#
arr1 = np.array([1, 2, 3, 4, 5])
arr2 = np.array([[1, 2, 3], [4, 5, 6]])

print(f"1  : {arr1}")           # [1 2 3 4 5]
print(f"2  :\n{arr2}")          # [[1 2 3], [4 5 6]]
print(f" : {arr2.shape}")          # (2, 3)
print(f"  : {arr1.dtype}")      # int64

#
zeros = np.zeros((3, 4))              #
ones = np.ones((2, 3))                # 1
range_arr = np.arange(0, 10, 2)       # [0 2 4 6 8]
linspace = np.linspace(0, 1, 5)       # [0.   0.25 0.5  0.75 1.  ]
random_arr = np.random.rand(3, 3)     #

#
arr = np.array([1, 2, 3, 4, 5])
print(arr * 2)                        # [2 4 6 8 10]
print(arr ** 2)                       # [1 4 9 16 25]
print(np.sqrt(arr))                   # [1. 1.41 1.73 2. 2.24]
```

1.4 NumPy

```
#
data = np.random.randn(1000)  #     1000

#
print(f" : {np.mean(data):.3f}")
print(f"  : {np.std(data):.3f}")
```

```
print(f"  : {np.min(data):.3f}")
print(f"  : {np.max(data):.3f}")
print(f"  : {np.median(data):.3f}")

#
matrix = np.arange(12).reshape(3, 4)
print(f"   : {matrix.shape}")
print(f" : {matrix.T.shape}")

#
print(f"   : {matrix[0]}")
print(f"   : {matrix[:, -1]}")
print(f"    : {matrix[matrix > 5]}")

#
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
combined = np.concatenate([arr1, arr2])  # [1 2 3 4 5 6]
stacked = np.stack([arr1, arr2])         # [[1 2 3], [4 5 6]]

#
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
print(f"  : \n{np.dot(A, B)}")
print(f"  : \n{np.linalg.inv(A)}")
print(f"  : {np.linalg.eigvals(A)}")
```

1.5 Pandas -

```
import pandas as pd

# Series 1
series = pd.Series([1, 3, 5, 7, 9], index=['a', 'b', 'c', 'd', 'e'])
print(series)

# DataFrame 2
data = {
    ' ': [' ', ' ', ' ', ' '],
    ' ': [25, 30, 35, 28],
```

```
    ' ': ['   ', '   ', '    ', '      '],
    ' ': [5000000, 4500000, 7000000, 6000000]
}

df = pd.DataFrame(data)
print(df)
```

```
0       25            5000000
1       30            4500000
2       35            7000000
3       28            6000000
```

1.6 Pandas

```
#
print(df.info())                    #
print(df.describe())                #
print(df.head())                    #   5
print(df.tail(3))                   #   3

#
print(df[' '])                  #
print(df[[' ', ' ']])       #
print(df.iloc[0])                   #
print(df.loc[df[' '] > 30])     #

#
   = df[df[' '] > 5000000]
    = df[df[' '] < 30]
    = df[df[' '].str.contains('   ')]

#
   = df.sort_values(' ')
    = df.sort_values(' ', ascending=False)

#
    = df.groupby(' ').agg({
    ' ': 'mean',
```

```
    ' ': ['mean', 'max', 'min']
})

#
df['    '] = df[' '] / 10000
df['    '] = pd.cut(df[' '], bins=[0, 30, 40, 100],
                        labels=[' ', ' ', '   '])
```

1.7

```
# CSV
# df = pd.read_csv('data.csv', encoding='utf-8')

#
# df = pd.read_excel('data.xlsx')
# df = pd.read_json('data.json')
# df = pd.read_sql('SELECT * FROM table', connection)

#
import pandas as pd
import numpy as np

#
np.random.seed(42)
dates = pd.date_range('2023-01-01', '2023-12-31', freq='D')
products = [' A', ' B', ' C', ' D']

sales_data = []
for date in dates:
    for product in products:
        sales = np.random.poisson(lam=50) + np.random.randint(0, 100)
        price = np.random.uniform(1000, 5000)
        sales_data.append({
            ' ': date,
            ' ': product,
            '  ': sales,
            ' ': round(price),
            ' ': sales * round(price)
        })
```

```
sales_df = pd.DataFrame(sales_data)

#
sales_df.to_csv('sales_data.csv', index=False, encoding='utf-8')
print("       ")
print(sales_df.head())
```

1.8 - Matplotlib

```
import matplotlib.pyplot as plt
plt.style.use('default')  #
plt.rcParams['font.family'] = 'DejaVu Sans'  #

#
fig, axes = plt.subplots(2, 2, figsize=(12, 8))

#
x = np.linspace(0, 10, 100)
y = np.sin(x)
axes[0, 0].plot(x, y, 'b-', linewidth=2)
axes[0, 0].set_title('Sine Wave')
axes[0, 0].grid(True)

#
x_scatter = np.random.randn(100)
y_scatter = 2 * x_scatter + np.random.randn(100)
axes[0, 1].scatter(x_scatter, y_scatter, alpha=0.6)
axes[0, 1].set_title('Scatter Plot')

#
data_hist = np.random.normal(0, 1, 1000)
axes[1, 0].hist(data_hist, bins=30, alpha=0.7, color='green')
axes[1, 0].set_title('Histogram')

#
categories = ['A', 'B', 'C', 'D']
values = [23, 17, 35, 29]
axes[1, 1].bar(categories, values, color='orange')
axes[1, 1].set_title('Bar Chart')
```

```
plt.tight_layout()
plt.show()

#
plt.figure(figsize=(10, 6))

#
months = ['1 ', '2 ', '3 ', '4 ', '5 ', '6 ']
product_a = [120, 135, 158, 142, 167, 183]
product_b = [98, 112, 126, 139, 145, 156]

plt.plot(months, product_a, 'o-', label=' A', linewidth=2, markersize=8)
plt.plot(months, product_b, 's-', label=' B', linewidth=2, markersize=8)
plt.xlabel(' ')
plt.ylabel('   ')
plt.title('   ')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```

1.9

```
#
def analyze_sales_data():
    #
    df = sales_df.copy()

    #   datetime
    df[' '] = pd.to_datetime(df[' '])
    df[' '] = df[' '].dt.month
    df[' '] = df[' '].dt.day_name()

    #
    print("===    ===")
    print(f"  : {df[' '].sum():,} ")
    print(f"   : {df.groupby(' ')[' '].sum().mean():,.0f} ")
    print(f"   : {df.groupby(' ')[' '].sum().max():,} ")

    #
```

```
print("\n===     ===")
product_analysis = df.groupby(' ').agg({
    ' ': ['sum', 'mean'],
    '  ': 'sum',
    ' ': 'mean'
}).round(0)
print(product_analysis)

#
monthly_sales = df.groupby(' ')[' '].sum()

#
weekday_sales = df.groupby(' ')[' '].sum()

#
correlation = df[['  ', ' ', ' ']].corr()
print("\n===    ===")
print(correlation)

#
fig, axes = plt.subplots(2, 2, figsize=(15, 10))

#
product_total = df.groupby(' ')[' '].sum()
axes[0, 0].pie(product_total.values, labels=product_total.index, autopct='%1.1f%%')
axes[0, 0].set_title('    ')

#
axes[0, 1].plot(monthly_sales.index, monthly_sales.values, 'o-', linewidth=2)
axes[0, 1].set_title('    ')
axes[0, 1].set_xlabel(' ')
axes[0, 1].set_ylabel('   ')
axes[0, 1].grid(True)

#   vs
axes[1, 0].scatter(df['  '], df[' '], alpha=0.6)
axes[1, 0].set_title('   vs  ')
axes[1, 0].set_xlabel('  ')
axes[1, 0].set_ylabel('   ')

#
weekday_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Su
```

```
    weekday_sales_ordered = weekday_sales.reindex(weekday_order)
    axes[1, 1].bar(range(7), weekday_sales_ordered.values)
    axes[1, 1].set_title('   ')
    axes[1, 1].set_xlabel(' ')
    axes[1, 1].set_ylabel('   ')
    axes[1, 1].set_xticks(range(7))
    axes[1, 1].set_xticklabels([' ', ' ', ' ', ' ', ' ', ' ', ' '])

    plt.tight_layout()
    plt.show()

    return df

#
analyzed_df = analyze_sales_data()
```

1.10 - Scikit-learn

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

#
np.random.seed(42)
n_samples = 1000

#   :
area = np.random.uniform(30, 150, n_samples)
age = np.random.uniform(0, 30, n_samples)
station_distance = np.random.uniform(1, 20, n_samples)
rooms = np.random.randint(1, 6, n_samples)

#     :
price = (area * 30 + rooms * 500 - age * 20 - station_distance * 50 +
         np.random.normal(0, 300, n_samples))
price = np.maximum(price, 1000)  #
```

```
# DataFrame
housing_data = pd.DataFrame({
    ' ': area,
    '  ': age,
    '  ': station_distance,
    '  ': rooms,
    ' ': price
})

print("     :")
print(housing_data.head())
print(f"\n   : {housing_data.shape}")
print("\n  :")
print(housing_data.describe())
```

1.11

```
#
X = housing_data[[' ', '  ', '  ', '  ']]
y = housing_data[' ']

#
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

print(f"   : {X_train.shape[0]} ")
print(f"    : {X_test.shape[0]} ")

#   1:
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)

#   2:
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

#
lr_pred = lr_model.predict(X_test)
rf_pred = rf_model.predict(X_test)
```

```
#
def evaluate_model(y_true, y_pred, model_name):
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_true, y_pred)

    print(f"\n=== {model_name} ===")
    print(f"RMSE: {rmse:.2f}")
    print(f"R²  : {r2:.4f}")
    return rmse, r2

lr_rmse, lr_r2 = evaluate_model(y_test, lr_pred, "  ")
rf_rmse, rf_r2 = evaluate_model(y_test, rf_pred, "    ")

#
feature_importance = pd.DataFrame({
    '  ': X.columns,
    '  ': rf_model.feature_importances_
}).sort_values('  ', ascending=False)

print("\n   :")
print(feature_importance)

#
fig, axes = plt.subplots(1, 3, figsize=(18, 5))

#   vs
axes[0].scatter(y_test, lr_pred, alpha=0.6)
axes[0].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
axes[0].set_xlabel('   ')
axes[0].set_ylabel('  ')
axes[0].set_title(f'   (R² = {lr_r2:.3f})')

#   vs
axes[1].scatter(y_test, rf_pred, alpha=0.6)
axes[1].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
axes[1].set_xlabel('   ')
axes[1].set_ylabel('  ')
axes[1].set_title(f'      (R² = {rf_r2:.3f})')

#
axes[2].barh(feature_importance['  '], feature_importance['  '])
```

```
axes[2].set_xlabel('  ')
axes[2].set_title('    ')

plt.tight_layout()
plt.show()

#
new_house = pd.DataFrame({
    ' ': [80],
    '  ': [5],
    '  ': [8],
    '  ': [3]
})

lr_prediction = lr_model.predict(new_house)[0]
rf_prediction = rf_model.predict(new_house)[0]

print(f"\n      :")
print(f" : {new_house[' '].iloc[0]} ,   : {new_house['  '].iloc[0]} ,   : {new_house['  '].iloc
print(f"    : {lr_prediction:,.0f} ")
print(f"       : {rf_prediction:,.0f} ")
```

1.12

1.13 :

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

#
np.random.seed(42)
n_customers = 500

#
age = np.random.normal(40, 12, n_customers)
income = np.random.normal(500, 150, n_customers)  #
spending = np.random.normal(300, 100, n_customers)  #
frequency = np.random.poisson(12, n_customers)  #
```

```
#
age = np.clip(age, 18, 80)
income = np.clip(income, 200, 1000)
spending = np.clip(spending, 50, 800)

customer_data = pd.DataFrame({
    ' ': age,
    ' ': income,
    '  ': spending,
    '  ': frequency
})

print("     :")
print(customer_data.head())
print("\n  :")
print(customer_data.describe())

#
scaler = StandardScaler()
scaled_data = scaler.fit_transform(customer_data)

# K-means
kmeans = KMeans(n_clusters=4, random_state=42)
clusters = kmeans.fit_predict(scaled_data)

#
customer_data['  '] = clusters

#
cluster_analysis = customer_data.groupby('  ').agg({
    ' ': 'mean',
    ' ': 'mean',
    '  ': 'mean',
    '  ': 'mean'
}).round(1)

print("\n    :")
print(cluster_analysis)

#
cluster_labels = {
    0: "    ",
```

```
    1: "   ",
    2: "   ",
    3: "    "
}

customer_data['    '] = customer_data['   '].map(cluster_labels)

#
fig, axes = plt.subplots(2, 2, figsize=(15, 12))

#    vs
scatter = axes[0, 0].scatter(customer_data[' '], customer_data['   '],
                             c=customer_data['   '], cmap='viridis', alpha=0.7)
axes[0, 0].set_xlabel('    ')
axes[0, 0].set_ylabel('     ')
axes[0, 0].set_title('  vs    ')
plt.colorbar(scatter, ax=axes[0, 0])

#    vs
scatter2 = axes[0, 1].scatter(customer_data[' '], customer_data['   '],
                              c=customer_data['   '], cmap='viridis', alpha=0.7)
axes[0, 1].set_xlabel(' ')
axes[0, 1].set_ylabel('   ')
axes[0, 1].set_title('  vs    ')

#
cluster_counts = customer_data['    '].value_counts()
axes[1, 0].pie(cluster_counts.values, labels=cluster_counts.index, autopct='%1.1f%%')
axes[1, 0].set_title('     ')

#
cluster_means = customer_data.groupby('    ')[[' ', '   ']].mean()
cluster_means.plot(kind='bar', ax=axes[1, 1])
axes[1, 1].set_title('       ')
axes[1, 1].set_ylabel('    ')
axes[1, 1].tick_params(axis='x', rotation=45)

plt.tight_layout()
plt.show()

#
print("\n===         ===")
```

```
for cluster_id, label in cluster_labels.items():
    cluster_data = customer_data[customer_data['  '] == cluster_id]
    size = len(cluster_data)
    avg_income = cluster_data[' '].mean()
    avg_spending = cluster_data['  '].mean()
    avg_frequency = cluster_data['  '].mean()

    print(f"\n{label} ({size} , {size/len(customer_data)*100:.1f}%)")
    print(f"    : {avg_income:.0f} ")
    print(f"    : {avg_spending:.0f} ")
    print(f"     : {avg_frequency:.1f} ")

    #
    if cluster_id == 0:
        print("   :           ")
    elif cluster_id == 1:
        print("   :     VIP  ")
    elif cluster_id == 2:
        print("   :           ")
    else:
        print("   :         ")
```

1.14

1. **NumPy** -
2. **Pandas** -
3. **Matplotlib** -
4. **Scikit-learn** -
5. -

1.15

- **Seaborn** -
- **Plotly** -
- **TensorFlow/PyTorch** -
- **Jupyter Notebook** -
- - Kaggle

1.16

1.16.1

: - NumPy - Pandas - Scikit-learn - Kaggle -

1.17

:

:

|

:
IoT