

پروژه پایان ترم درس برنامه نویسی پیشرفته استاد شیرازی

ترم پاییز 1404

اسامی افراد گروه 18 :

شیما سلطانی : 14044110

پویا جزء سلیمانی : 14044114

مهری اوصالی : 14044126

لينک گيت هاب :

<https://github.com/Shimsol/AP-Final-Project-14044126-14044110-14044114.git>

1. Data Acquisition

این بخش از کتابخانه های مورد نیاز پروژه را برای پردازش داده، محاسبات عددی و مصورسازی بارگذاری می کند. همچنین استایل پیش فرض نمودارها تنظیم شده و با تعیین seed ثابت، قابلیت بازتولید نتایج به صورت کامل تضمین می شود.

1. Data Acquisition

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Set plot style
sns.set(style="whitegrid")

# Set random seed for full reproducibility
np.random.seed(42)

print("Libraries imported and setup complete.")
```

[85]

... Libraries imported and setup complete.

در این بخش، دیتاست واقعی **UCI Air Quality** بارگذاری شده و تنها پیش پردازش های حداقلی روی آن انجام می شود تا اصالت داده ها حفظ شود. این مراحل شامل ترکیب ستون های تاریخ و زمان، مدیریت مقادیر گمشده رسمی دیتاست و جلوگیری از افزودن نویز یا داده مصنوعی است.

این کد، دیتاست کیفیت هوا را بارگذاری کرده و ستون های Date و Time را به یک ستون زمانی واحد (DateTime) تبدیل می کند. سپس ستون های اولیه حذف شده و مقدار 200- که نشان دهنده داده های گمشده است، با NaN جایگزین می شود. در نهایت، ابعاد دیتاست گزارش شده و فرآیند پاک سازی داده های واقعی تکمیل می گردد.

```
print("==> Loading and Cleaning Real UCI Air Quality Data ==\n")

# Load the dataset
df_real = pd.read_csv("AirQualityUCI.csv", sep=";", decimal=",")

# Combine Date and Time
df_real['DateTime'] = pd.to_datetime(df_real['Date'] + ' ' + df_real['Time'],
                                     format='%d/%m/%Y %H.%M.%S')

# Drop original columns
df_real.drop(['Date', 'Time'], axis=1, inplace=True)

# Replace -200 (missing value marker) with NaN
df_real.replace(-200, np.nan, inplace=True)

print("Replaced all -200 values with NaN.")
print(f"Dataset shape: {df_real.shape}")
print("Real data cleaning complete.\n")

[86]
...
    ==> Loading and Cleaning Real UCI Air Quality Data ==

Replaced all -200 values with NaN.
Dataset shape: (9471, 16)
Real data cleaning complete.
```

در این بخش، توزیع ویژگی های عددی سنسورها پس از پاک سازی داده ها بررسی می شود. هدف، درک رفتار آماری داده ها و شناسایی الگوهای پراکندگی و ناهنجاری های احتمالی در ویژگی ها است. این کد تمام ستون های عددی دیتاست واقعی را استخراج کرده و برای هر ویژگی، هیستوگرام رسم می کند. مقادیر گمراه حذف شده و نمودارها به صورت شبکه ای نمایش داده می شوند تا مقایسه آسان تر باشد. در نهایت، خروجی مصورسازی ذخیره شده و برای تحلیل توزیع ویژگی ها مورد استفاده قرار می گیرد.

1-2. Feature Distributions (After Cleaning)

Visualizing the distribution of all numerical sensor features after replacing -200 with NaN.

```
numerical_cols = df_real.select_dtypes(include=np.number).columns

n_cols_plot = 4
n_rows_plot = (len(numerical_cols) + n_cols_plot - 1) // n_cols_plot

plt.figure(figsize=(16, 4 * n_rows_plot))
for i, col in enumerate(numerical_cols):
    plt.subplot(n_rows_plot, n_cols_plot, i + 1)
    df_real[col].dropna().hist(bins=50, color='salmon', edgecolor='black')
    plt.title(f'{col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')

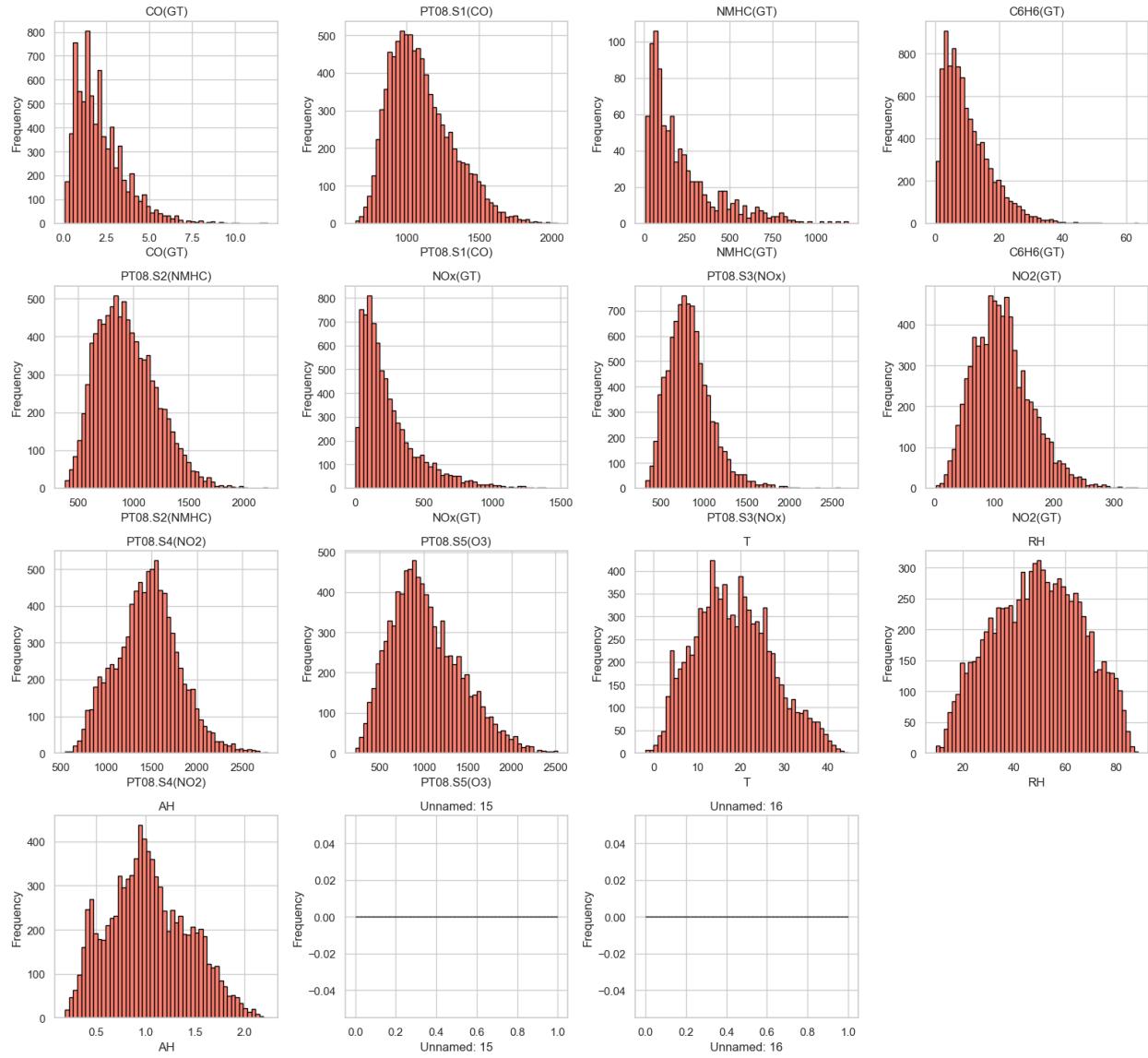
plt.suptitle('Real Data Feature Distributions\n(After replacing -200 with NaN)', fontsize=16)
plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.savefig("real_data_distributions.png", dpi=300)
plt.show()

print("Real data distributions plotted and saved as 'real_data_distributions.png'")
```

[87]

خروجی این قسمت از کد، در صفحه بعد:

Real Data Feature Distributions
(After replacing -200 with NaN)



در این بخش، خلاصه‌ای آماری از ویژگی‌های عددی دیتاست پس از پاک‌سازی ارائه می‌شود. علاوه بر شاخص‌های پایه آماری، میزان داده‌های پرت و درصد مقادیر گمشده برای هر ویژگی محاسبه می‌شود تا کیفیت داده‌ها ارزیابی گردد.

این کد برای هر ویژگی عددی، آماره‌های شامل میانگین، واریانس، حداقل و حداکثر را محاسبه می‌کند. داده‌های پرت با استفاده از روش **IQR** شناسایی شده و درصد آن‌ها نسبت به داده‌های معتبر گزارش می‌شود. همچنین درصد مقادیر گمشده در هر ویژگی محاسبه شده و نتایج به صورت جدولی و خواناً چاپ می‌گردد.

1-3. Statistical Summary & Outlier Analysis

Summary statistics including:

- Mean, Variance, Min, Max
- Outlier percentage (IQR method)
- Missing and Valid percentages

```
# --- Statistics & Outliers (on cleaned real data) ---
print("== Statistical Summary (Cleaned Real Data) ==\n")
# Header: Range is now left-aligned with extra space separation
print(f'{ "Feature":<15} {"Mean":>10} {"Variance":>12} {"Min":>10} {"Max":>10} {"Outlier %":>12} {"Missing %":>12} {"Range"}')
print("-" * 120)

for col in numerical_cols:
    data = df_real[col].dropna()
    total_rows = len(df_real)
    missing_pct = (df_real[col].isna().sum() / total_rows) * 100

    if len(data) == 0:
        print(f'{col:<15} {-:>10} {-:>12} {-:>10} {-:>10} {0.00%:>12} {missing_pct:11.2f}% {[ - , - ]}')
        continue

    mean_val = data.mean()
    var_val = data.var()
    min_val = data.min()

    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR

    outliers = data[(data < lower) | (data > upper)]
    outlier_pct = (len(outliers) / len(data)) * 100 if len(data) > 0 else 0.0

    # Format range string
    range_str = f"[{min_val:.2f}, {max_val:.2f}]"
    # Left-align Range with extra spacing before it
    print(f'{col:<15} {mean_val:10.3f} {var_val:12.3f} {min_val:10.3f} {max_val:10.3f} "
          f"{outlier_pct:11.2f}% {missing_pct:11.2f}% {range_str}")"

print("\n(Statistics and outliers calculated only on valid values)")
print("Range shows the observed min-max values of each feature after cleaning.")
```

خروجی این قسمت به صورت زیر است:

--- Statistical Summary (Cleaned Real Data) ---							
Feature	Mean	Variance	Min	Max	Outlier %	Missing %	Range
CO(GT)	2.153	2.112	0.100	11.900	2.80%	18.97%	[0.10, 11.90]
PT08.S1(CO)	1099.833	47123.743	647.000	2040.000	1.31%	5.07%	[647.00, 2040.00]
NMHC(GT)	218.812	41803.859	7.000	1189.000	6.02%	90.35%	[7.00, 1189.00]
C6H6(GT)	10.083	55.500	0.100	63.700	2.54%	5.07%	[0.10, 63.70]
PT08.S2(NMHC)	939.153	71199.011	383.000	2214.000	0.72%	5.07%	[383.00, 2214.00]
NOx(GT)	246.897	45360.126	2.000	1479.000	5.64%	18.51%	[2.00, 1479.00]
PT08.S3(NOx)	835.494	65955.136	322.000	2683.000	2.68%	5.07%	[322.00, 2683.00]
N02(GT)	113.091	2339.667	2.000	340.000	1.39%	18.54%	[2.00, 340.00]
PT08.S4(N02)	1456.265	119859.144	551.000	2775.000	1.08%	5.07%	[551.00, 2775.00]
PT08.S5(O3)	1022.906	158789.728	221.000	2523.000	1.03%	5.07%	[221.00, 2523.00]
T	18.318	78.006	-1.900	44.600	0.03%	5.07%	[-1.90, 44.60]
RH	49.234	299.875	9.200	88.700	0.00%	5.07%	[9.20, 88.70]
AH	1.026	0.163	0.185	2.231	0.02%	5.07%	[0.18, 2.23]
Unnamed: 15	-	-	-	-	0.00%	100.00%	[-, -]
Unnamed: 16	-	-	-	-	0.00%	100.00%	[-, -]

این بخش دیتاست واقعی پاک سازی شده را بدون اندیس در قالب فایل CSV ذخیره می کند. هدف از این مرحله، آماده سازی داده ها برای ادامه ای فرآیند پیش پردازش در گام بعدی پروژه و تحويل آن به مرحله بعدی پایپ لاین است.

1-4. saving the raw data

```
df_real.to_csv("real_air_quality_data_raw.csv", index=False)
print("Cleaned real data saved as 'real_air_quality_data_raw.csv'")
print("→ Ready for preprocessing by Person A (next step)\n")
[89]
...
Cleaned real data saved as 'real_air_quality_data_raw.csv'
→ Ready for preprocessing by Person A (next step)
```

در این بخش، داده های مصنوعی سنسور های IoT با رفتار واقع گرایانه تولید می شوند. داده ها شامل الگوهای فصلی، نویز سنسوری و مقادیر گمشده تصادفی هستند تا شبیه واقعی سیستم ها IoT شوی سازی شود.

این کد یک دیتاست زمانی ساعتی ایجاد کرده و برای هر ویژگی عددی، داده مصنوعی مبتنی بر آمار دیتاست واقعی تولید می کند. سیگنال ها حول میانگین واقعی ویژگی ها ساخته شده و با الگوی سینوسی فصلی و نویز گاوی ترکیب می شوند. برای شبیه سازی افت داده در سنسورها، حدود ۵٪ مقادیر گمشده به صورت کاملاً قابل بازتولید اضافه شده و در نهایت دیتاست ذخیره می گردد.

2. Synthetic IoT Sensor Data

Generating realistic synthetic sensor readings:

- Temperature and Relative Humidity
- Seasonal patterns (sinusoidal)
- Gaussian sensor noise
- ~5% random missing values to simulate real IoT dropouts

2-1 Generating synthetic data

```
import numpy as np
import pandas as pd
import hashlib

print("==> Generating Synthetic Sensor Data (Seasonal + Noise, Centered on Real Means) ==>\n")

# Global seed (kept for consistency, not relied on for feature RNG)
np.random.seed(42)

# Hourly timestamps
time = pd.date_range("2025-01-01", periods=5000, freq="h")
```

ادامه تصاویر در صفحه بعد:

```

import numpy as np
import pandas as pd
import hashlib

print("== Generating Synthetic Sensor Data (Seasonal + Noise, Centered on Real Means) ==\n")

# Global seed (kept for consistency, not relied on for feature RNG)
np.random.seed(42)

# Hourly timestamps
time = pd.date_range("2025-01-01", periods=5000, freq="h")

# Start DataFrame
df_synthetic = pd.DataFrame({"DateTime": time})

numerical_cols = df_real.select_dtypes(include=np.number).columns

def stable_feature_seed(column_name, base_seed=42):
    """
    Generate a fully deterministic seed from a column name.
    Reproducible across runs, machines, and Python versions.
    """
    return base_seed + int(
        hashlib.md5(column_name.encode("utf-8")).hexdigest(),
        16
    ) % 10000

```

```

for col in numerical_cols:
    data_real = df_real[col].dropna()
    if len(data_real) == 0:
        df_synthetic[col] = np.nan
        continue

    mean_val = data_real.mean()
    std_val = data_real.std() # realistic spread

    # Fully reproducible RNG per feature
    feature_seed = stable_feature_seed(col)
    rng = np.random.default_rng(feature_seed)

    # Seasonal pattern: multi-cycle sine wave
    seasonal = np.sin(np.linspace(0, 12 * np.pi, len(time))) # ~6 cycles

    # Base signal centered on real mean
    values = mean_val + seasonal * std_val * 0.5

    # Gaussian noise (realistic sensor noise)
    noise = rng.normal(0, std_val * 0.3, len(time))
    values += noise

    # Add ~5% missing values (fully reproducible)
    missing_mask = rng.choice([True, False], size=len(time), p=[0.05, 0.95])
    values[missing_mask] = np.nan

```

خروجی :

```
# Save
df_synthetic.to_csv("synthetic_sensor_data_raw.csv", index=False)

print(f"Synthetic data generated: {len(df_synthetic)} rows, matching all {len(numerical_cols)} real features")
print("→ Centered on real mean")
print("→ Seasonal sinusoidal pattern + realistic Gaussian noise (scaled from real std)")
print("→ No clipping → smooth, natural distributions (no end spikes)")
print("→ ~5% missing values per feature")
print("→ 100% reproducible across runs & machines")
print("Saved as 'synthetic_sensor_data_raw.csv'\n")



== Generating Synthetic Sensor Data (Seasonal + Noise, Centered on Real Means) ==

Synthetic data generated: 5000 rows, matching all 15 real features
→ Centered on real mean
→ Seasonal sinusoidal pattern + realistic Gaussian noise (scaled from real std)
→ No clipping → smooth, natural distributions (no end spikes)
→ ~5% missing values per feature
→ 100% reproducible across runs & machines
Saved as 'synthetic_sensor_data_raw.csv'
```

در این بخش، توزیع ویژگی های عددی داده های مصنوعی پس از افزودن الگوهای فصلی، نویز و مقادیر گمشده بررسی می شود. این تحلیل به ارزیابی میزان شباهت داده های مصنوعی به رفتار واقعی سنسورهای IoT کمک می کند. این کد برای هر ویژگی عددی در دیتاست مصنوعی، هیستوگرام توزیع مقادیر را رسم می کند. مقادیر گمشده حذف شده و نمودارها به صورت شبکه ای برای مقایسه بصری نمایش داده می شوند. در پایان، خروجی مصورسازی ذخیره شده و برای تحلیل کیفیت داده های مصنوعی مورد استفاده قرار می گیرد.

2-2. Feature Distributions (After adding noise and outliers)

```

print("--- Generating Distributions (Synthetic Data) ---\n")

n_cols_plot = 4
n_rows_plot = (len(numerical_cols) + n_cols_plot - 1) // n_cols_plot

plt.figure(figsize=(16, 4 * n_rows_plot))
for i, col in enumerate(numerical_cols):
    plt.subplot(n_rows_plot, n_cols_plot, i + 1)
    df_synthetic[col].dropna().hist(bins=50, color='skyblue', edgecolor='black')
    plt.title(f'{col} (Synthetic)')
    plt.xlabel(col)
    plt.ylabel('Frequency')

plt.suptitle('Synthetic Data Feature Distributions\n(Values constrained to real observed ranges + noise + ~5% missing)', fontsize=16)
plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.savefig("synthetic_data_distributions.png", dpi=300)
plt.show()

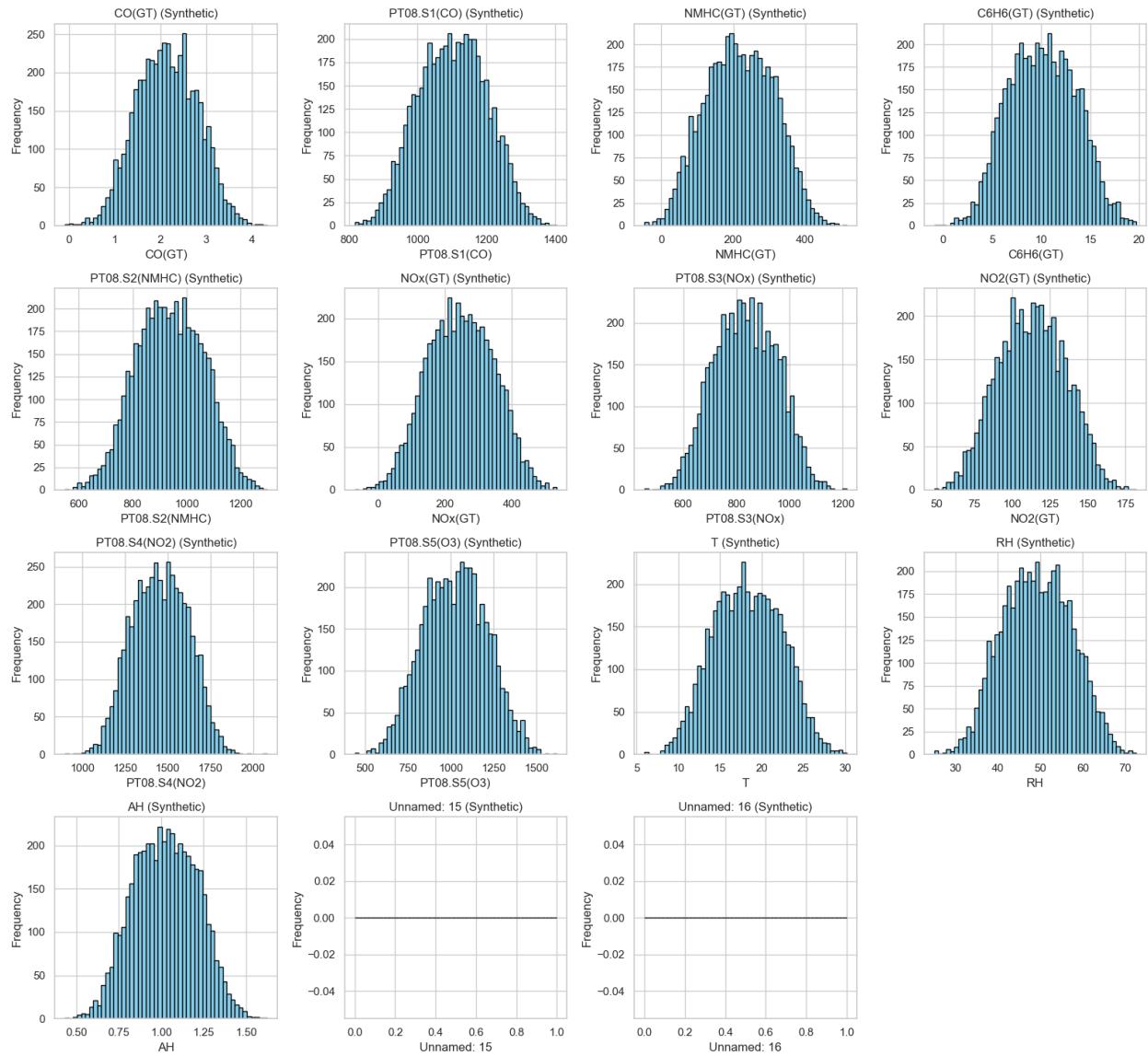
print("Synthetic distributions saved as 'synthetic_data_distributions.png'\n")

--- Generating Distributions (Synthetic Data) ---

```

خروجی در صفحه بعد:

Synthetic Data Feature Distributions
(Values constrained to real observed ranges + noise + ~5% missing)



این بخش مشابه تحلیل آماری داده های واقعی است، اما روی داده های مصنوعی اعمال می شود. برای هر ویژگی عددی، میانگین، واریانس، حداقل و حداکثر، درصد مقادیر گمشده و درصد داده های پرت با روش IQR محاسبه و نمایش داده می شود. هدف بررسی کیفیت داده مصنوعی، تشخیص الگوهای پرت و اطمینان از شباهت آماری به داده واقعی است.

```

# Same header as real data
print(f'{Feature}<:15} {"Mean":>10} {"Variance":>12} {"Min":>10} {"Max":>10} {"Outlier %":>12} {"Missing %":>12} {"Range"}')
print("-" * 120)

# Use same numerical columns (they match exactly)
for col in numerical_cols:
    data = df_synthetic[col].dropna()
    total_rows = len(df_synthetic)
    missing_pct = (df_synthetic[col].isna().sum() / total_rows) * 100

    if len(data) == 0:
        print(f'{col}<:15} {"-":>10} {"-":>12} {"-":>10} {"-":>10} {"0.00%":>12} {missing_pct:11.2f}%      {'[- , -]'}')
        continue

    mean_val = data.mean()
    var_val = data.var()
    min_val = data.min()
    max_val = data.max()

    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR

    outliers = data[(data < lower) | (data > upper)]
    outlier_pct = (len(outliers) / len(data)) * 100 if len(data) > 0 else 0.0

    range_str = f"[{min_val:.2f}, {max_val:.2f}]"

    print(f'{col}<:15} {mean_val:.3f} {var_val:.3f} {min_val:.3f} {max_val:.3f} "'
    |   f"outlier_pct:11.2f% {missing_pct:11.2f}%      {range_str}"')

print("\n(Statistics and outliers calculated only on valid values in synthetic data)")
print("Note: Missing % is approximately 5% due to simulated IoT dropouts.")

```

خروجی:

Feature	Mean	Variance	Min	Max	Outlier %	Missing %	Range
CO(GT)	2.143	0.443	-0.088	4.339	0.25%	4.86%	[-0.09, 4.34]
PT08.S1(CO)	1101.351	9845.322	817.043	1404.637	0.02%	5.04%	[817.04, 1404.64]
NMHC(GT)	218.851	8916.423	-46.591	514.856	0.02%	4.94%	[-46.59, 514.86]
C6H6(GT)	10.125	11.792	-0.901	19.711	0.04%	5.34%	[-0.90, 19.71]
PT08.S2(NMHC)	937.672	15360.070	549.440	1298.223	0.02%	5.14%	[549.44, 1298.22]
NOx(GT)	247.698	9661.046	-68.921	534.944	0.02%	5.24%	[-68.92, 534.94]
PT08.S3(NOx)	834.324	14024.911	454.120	1216.092	0.08%	5.76%	[454.12, 1216.09]
NO2(GT)	113.076	497.212	48.516	182.349	0.06%	5.52%	[48.52, 182.35]
PT08.S4(NO2)	1455.405	26291.951	900.095	2082.085	0.08%	5.10%	[900.10, 2082.08]
PT08.S5(O3)	1022.887	34204.917	437.633	1618.596	0.06%	4.98%	[437.63, 1618.60]
T	18.318	16.504	5.957	30.184	0.04%	5.02%	[5.96, 30.18]
RH	49.247	63.946	25.022	72.590	0.04%	5.00%	[25.02, 72.59]
AH	1.025	0.035	0.434	1.624	0.06%	4.78%	[0.43, 1.62]
Unnamed: 15	-	-	-	-	0.00%	100.00%	[-, -]
Unnamed: 16	-	-	-	-	0.00%	100.00%	[-, -]

این بخش، دیتاست واقعی پاک سازی شده را بازگذاری می کند و ستون DateTime را به نوع datetime تبدیل و بر اساس زمان مرتب می کند. هدف، آماده سازی داده ها برای مرحله ی پیش پردازش و تقسیم بندی به مجموعه های آموزش و تست است.

3. Cleaning & Preprocessing

3-1. Load Real Data

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

# Load real raw data
df_real = pd.read_csv("real_air_quality_data_raw.csv")

# Ensure DateTime is datetime and sorted
df_real['DateTime'] = pd.to_datetime(df_real['DateTime'])
df_real = df_real.sort_values('DateTime').reset_index(drop=True)

print("Real data loaded.")
print(f"Shape: {df_real.shape}")
df_real.head()
```

```
Real data loaded.
Shape: (9471, 16)
```

این بخش تمام ستون های عددی دیتاست واقعی را شناسایی می کند و ستون هایی که کاملاً خالی هستند را حذف می کند. هدف، اطمینان از اینکه فقط ویژگی های معتبر برای مدل سازی استفاده شوند و ابعاد دیتاست به روزرسانی شود.

3-2. Identify Numerical Columns and drop fully empty columns

```
numerical_cols = df_real.select_dtypes(include=np.number).columns
print("Numerical columns:", list(numerical_cols))

empty_cols = df_real.columns[df_real.isna().all()]
if len(empty_cols) > 0:
    print(f"Dropping fully empty columns: {list(empty_cols)}")
    df_real.drop(empty_cols, axis=1, inplace=True)
numerical_cols = numerical_cols.drop(empty_cols, errors='ignore')

print(f"Shape after drop: {df_real.shape}")

Numerical columns: ['CO(GT)', 'PT08.S1(CO)', 'NMHC(GT)', 'C6H6(GT)', 'PT08.S2(NMHC)', 'NOx(GT)', 'PT08.S3(NOx)', 'NO2(GT)', 'PT08.S4(NO2)', 'PT08.S5(O3'
Dropping fully empty columns: ['Unnamed: 15', 'Unnamed: 16']
Shape after drop: (9471, 14)
```

Python

این بخش ستون هایی که بیش از 80% مقادیر آن ها گمشده هستند شناسایی و حذف می شوند. هدف، کاهش تأثیر ویژگی های با اطلاعات ناکافی و حفظ کیفیت داده ها برای مرحله بعدی مدل سازی است.

3-3. Drop High-Missing Columns

```
missing_rates = df_real[numerical_cols].isna().mean()
high_missing = missing_rates[missing_rates > 0.8].index
if len(high_missing) > 0:
    print(f"Dropping columns with >80% missing: {list(high_missing)}")
    df_real.drop(high_missing, axis=1, inplace=True)
    numerical_cols = numerical_cols.drop(high_missing)

print(f"Shape after drop: {df_real.shape}")
```

```
Dropping columns with >80% missing: ['NMHC(GT)']
Shape after drop: (9471, 13)
```

این بخش ستون هدف مدل (C6H6(GT)) را شناسایی کرده و آن را به انتهای دیتابس منقل می کند. هدف، جداسازی واضح ویژگی ها از ستون پیشینی شده برای آماده سازی داده ها جهت مدل سازی است.

3-4. Move Target to End

```
target_col = "C6H6(GT)" # Change if needed
if target_col in df_real.columns:
    target = df_real.pop(target_col)
    df_real[target_col] = target
    print(f"Target '{target_col}' moved to end.")
else:
    print(f"Warning: Target '{target_col}' not found.")
df_real.head()
```

```
Target 'C6H6(GT)' moved to end.
```

این بخش دیتاست واقعی را بر اساس ترتیب زمانی به دو مجموعه آموزش (80% اول) و تست (20% آخر) تقسیم می کند. هدف، حفظ ترتیب زمانی داده ها برای مدل های پیشینی سری های زمانی و آماده سازی برای آموزش و ارزیابی مدل است.

3-5. Split Into Train/Test

```
split_idx = int(len(df_real) * 0.8)
train_real = df_real.iloc[:split_idx].copy()
test_real = df_real.iloc[split_idx: ].copy()

print("Train shape:", train_real.shape)
print("Test shape:", test_real.shape)
```

```
Train shape: (7576, 13)
Test shape: (1895, 13)
```

این بخش مقادیر گمشده ویژگی ها را در مجموعه های آموزش و تست به صورت جداگانه با پر می کند. ستون هدف در مجموعه تست دست نخورده باقی می ماند backward-fill و forward-fill تا برای ارزیابی واقعی پیشینی استفاده شود و ردیف های با هدف گمشده در مجموعه آموزش حذف می شوند. هدف، آماده سازی داده ها برای مدل بدون ایجاد داده مصنوعی در ستون هدف است.

3-6. Impute Missing Values (forward-fill + backward-fill for time-series)

```
feature_cols = [col for col in numerical_cols if col != target_col]

# Impute only features in train and test separately
for df, name in [(train_real, "Train"), (test_real, "Test")]:
    print(f"Imputing {name} features with forward-fill + backward-fill.")
    df[feature_cols] = df[feature_cols].ffill().bfill()

    # Check remaining missing in features (should be 0)
    print(f"{name} remaining missing % in features:")
    print(df[feature_cols].isna().mean() * 100)

# Do NOT impute target in test - leave as-is for real prediction
# For train: Drop rows with missing target
train_real = train_real.dropna(subset=[target_col])
print("\nTrain rows after dropping missing target:", len(train_real))
```

```
Train remaining missing % in features:
CO(GT)          0.0
PT08.S1(CO)     0.0
PT08.S2(NMHC)   0.0
NOx(GT)         0.0
PT08.S3(NOx)    0.0
NO2(GT)         0.0
PT08.S4(NO2)    0.0
PT08.S5(O3)    0.0
T               0.0
RH              0.0
AH              0.0
dtype: float64
Imputing Test features with forward-fill + backward-fill.
Test remaining missing % in features:
CO(GT)          0.0
PT08.S1(CO)     0.0
PT08.S2(NMHC)   0.0
NOx(GT)         0.0
PT08.S3(NOx)    0.0
NO2(GT)         0.0
PT08.S4(NO2)    0.0
PT08.S5(O3)    0.0
T               0.0
RH              0.0
AH              0.0
dtype: float64

Train rows after dropping missing target: 7296
```

این بخش، ویژگی های مجموعه آموزش و تست را با میانگین متحرک (rolling mean) با پنجره ۳ صاف می کند تا نوسانات کوتاه مدت و نویز سنسوری کاهش یابد.
هدف، افزایش هموار بودن سیگنال ها و بهبود کیفیت ورودی برای مدل پیش‌بینی است.

3-7. Smooth Time-Series

```
# Smooth only features in train and test separately
for df, name in [(train_real, "Train"), (test_real, "Test")]:
    print(f"Smoothing {name} features with rolling mean (window=3).")
    for col in feature_cols:
        df[col] = df[col].rolling(window=3, min_periods=1, center=True).mean()

    print("Smoothing complete.")
train_real.head()
[99]
...
Smoothing Train features with rolling mean (window=3).
Smoothing Test features with rolling mean (window=3).
Smoothing complete.
```

این بخش ویژگی های مجموعه آموزش را با Min-Max Scaling نرمال سازی می کند و سپس همان مقیاس را روی مجموعه تست اعمال می کند. هدف، همسان سازی مقیاس ویژگی ها برای مدل و جلوگیری از نشت اطلاعات از مجموعه تست است.

3-8. Normalize Features

```
# Fit scaler on train features only
scaler = MinMaxScaler()
scaler.fit(train_real[feature_cols])

# Transform both train and test features
train_real[feature_cols] = scaler.transform(train_real[feature_cols])
test_real[feature_cols] = scaler.transform(test_real[feature_cols])

print("Normalization complete (fit on train, applied to both).")
train_real.describe()
```

```
Normalization complete (fit on train, applied to both).
```

این بخش ویژگی هایی که پس از نرمال سازی واریانس بسیار کمی دارند شناسایی و از مجموعه های آموزش و تست حذف می شوند. هدف، حذف ویژگی های کم اطلاعات برای کاهش پیچیدگی مدل و بهبود عملکرد پیشینی است.

3-9. Drop Low-Variance Features

```
# Calculate variance on train features after scaling
variances = train_real[feature_cols].var()
low_var = variances[variances < 0.01].index
if len(low_var) > 0:
    print(f"Dropping low-variance features: {list(low_var)}")
    train_real.drop(low_var, axis=1, inplace=True)
    test_real.drop(low_var, axis=1, inplace=True)
    feature_cols = [col for col in feature_cols if col not in low_var]

print(f"Train shape after drop: {train_real.shape}")
print(f"Test shape after drop: {test_real.shape}")
```

```
Train shape after drop: (7296, 13)
Test shape after drop: (1895, 13)
```

این بخش مجموعه های داده واقعی پیش پردازش شده، آموزش و تست را در قالب فایل CSV ذخیره می کند. هدف، آماده سازی داده ها برای مراحل بعدی مدل سازی و اطمینان از دسترسی آسان به نسخه های نهایی داده ها برای تحلیل و آموزش مدل است.

3-10. Save Preprocessed Real Files

```
df_real.to_csv("preprocessed_real.csv", index=False)
train_real.to_csv("train_real.csv", index=False)
test_real.to_csv("test_real.csv", index=False)
print("Real data files saved:")
print("- preprocessed_real.csv")
print("- train_real.csv")
print("- test_real.csv")
```

```
Real data files saved:
- preprocessed_real.csv
- train_real.csv
- test_real.csv
```

این بخش دیتاست مصنوعی سنسور های IoT را بازگذاری کرده و ستون `DateTime` را به نوع `datetime` تبدیل و بر اساس زمان مرتب می کند. هدف، آماده سازی داده های مصنوعی برای پیش پردازش مشابه داده واقعی و استفاده در مدل سازی است.

4. Cleaning & Preprocessing for Synthetic Data

4-1. Load Synthetic Data

```
# Load synthetic raw data
df_synthetic = pd.read_csv("synthetic_sensor_data_raw.csv")

# Ensure DateTime is datetime and sorted
df_synthetic['DateTime'] = pd.to_datetime(df_synthetic['DateTime'])
df_synthetic = df_synthetic.sort_values('DateTime').reset_index(drop=True)

print("Synthetic data loaded.")
print(f"Shape: {df_synthetic.shape}")
df_synthetic.head()

[103]

...
Synthetic data loaded.
Shape: (5000, 16)
```

این بخش تمام ستون های عددی دیتاست مصنوعی را شناسایی می کند و ستون هایی که کاملاً خالی هستند حذف می شوند. هدف، اطمینان از استفاده از ویژگی های معتبر و کاهش ابعاد داده برای مراحل پیش پردازش و مدل سازی است.

4-2. Identify Numerical Columns and Drop Fully Empty Columns

```
numerical_cols_synth = df.synthetic.select_dtypes(include=np.number).columns
print("Numerical columns (synthetic):", list(numerical_cols_synth))

empty_cols_synth = df.synthetic.columns[df.synthetic.isna().all()]
if len(empty_cols_synth) > 0:
    print(f"Dropping fully empty columns: {list(empty_cols_synth)}")
    df.synthetic.drop(empty_cols_synth, axis=1, inplace=True)
    numerical_cols_synth = numerical_cols_synth.drop(empty_cols_synth, errors='ignore')

print(f"Shape after drop: {df.synthetic.shape}")

Numerical columns (synthetic): ['CO(GT)', 'PT08.S1(CO)', 'NMHC(GT)', 'C6H6(GT)', 'PT08.S2(NMHC)', 'NOx(GT)', 'PT08.S3(NOx)', 'NO2(GT)', 'PT08.S4(NO2)', 'Dropping fully empty columns: ['Unnamed: 15', 'Unnamed: 16']'
Shape after drop: (5000, 14)
```

این بخش ستون هایی که بیش از 80% مقادیر آنها گمشده هستند در دیتاست مصنوعی شناسایی و حذف می شوند. هدف، حذف ویژگی های کم اطلاعات و حفظ کیفیت داده ها برای پیش پردازش و مدل سازی است.

4-3. Drop High-Missing Columns (Synthetic)

```
missing_rates_synth = df_synthetic[numerical_cols_synth].isna().mean()
high_missing_synth = missing_rates_synth[missing_rates_synth > 0.8].index
if len(high_missing_synth) > 0:
    print(f"Dropping columns with >80% missing: {list(high_missing_synth)}")
    df_synthetic.drop(high_missing_synth, axis=1, inplace=True)
    numerical_cols_synth = numerical_cols_synth.drop(high_missing_synth)

print(f"Shape after drop: {df_synthetic.shape}")
```

```
Shape after drop: (5000, 14)
```

این بخش ستون هدف در دیتاست مصنوعی (C6H6(GT)) را شناسایی کرده و به انتهای دیتاست منتقل می کند. هدف، جدا کردن ویژگی ها از ستون پیشビینی شده و آماده سازی داده ها برای مراحل پیش پردازش و مدل سازی است.

4-4. Move Target to End (Synthetic)

```
target_col_synth = "C6H6(GT)" # Change if needed
if target_col_synth in df_synthetic.columns:
    target_synth = df_synthetic.pop(target_col_synth)
    df_synthetic[target_col_synth] = target_synth
    print(f"Target '{target_col_synth}' moved to end.")
else:
    print(f"Warning: Target '{target_col_synth}' not found.")
df_synthetic.head()
```

```
Target 'C6H6(GT)' moved to end.
```

این بخش دیتاست مصنوعی را به ترتیب زمانی به دو مجموعه آموزش (80% اول) و تست (20% آخر) تقسیم می کند. هدف، حفظ ترتیب زمانی برای پیش‌بینی سری های زمانی و آماده سازی داده‌ها برای آموزش و ارزیابی مدل است.

4-5. Split Into Train/Test (Synthetic)

```
split_idx_synth = int(len(df_synthetic) * 0.8)
train_synthetic = df_synthetic.iloc[:split_idx_synth].copy()
test_synthetic = df_synthetic.iloc[split_idx_synth: ].copy()

print("Train shape:", train_synthetic.shape)
print("Test shape:", test_synthetic.shape)
```

```
Train shape: (4000, 14)
Test shape: (1000, 14)
```

این بخش مقادیر گمشده ویژگی ها در مجموعه های آموزش و تست دیتاست مصنوعی با forward-fill و backward-fill پر می شود. ستون هدف در مجموعه تست دست نخورده باقی می ماند و ردیف های با هدف گمشده در مجموعه آموزش حذف می شوند. هدف، آماده سازی داده ها برای مدل بدون ایجاد داده مصنوعی در ستون هدف است.

4-6. Impute Missing Values (Synthetic)

```
feature_cols_synth = [col for col in numerical_cols_synth if col != target_col_synth]

# Impute only features in train and test separately
for df, name in [(train_synthetic, "Train"), (test_synthetic, "Test")]:
    print(f"Imputing {name} features with forward-fill + backward-fill.")
    df[feature_cols_synth] = df[feature_cols_synth].ffill().bfill()

    print(f"{name} remaining missing % in features:")
    print(df[feature_cols_synth].isna().mean() * 100)

# Do NOT impute target in test
# For train: Drop rows with missing target
train_synthetic = train_synthetic.dropna(subset=[target_col_synth])
print("\nTrain rows after dropping missing target:", len(train_synthetic))
```

```
Imputing Train features with forward-fill + backward-fill.
Train remaining missing % in features:
```

این بخش ویژگی های مجموعه آموزش و تست دیتاست مصنوعی را با میانگین متحرک با پنجره ۳ Smooth می کند. هدف، کاهش نویز و نوسانات کوتاه مدت در سیگنال های سنسوری و بهبود کیفیت داده ها برای مدل پیشینی است.

4-7. Smooth Time-Series (Synthetic)

```
# Smooth only features in train and test separately
for df, name in [(train_synthetic, "Train"), (test_synthetic, "Test")]:
    print(f"Smoothing {name} features with rolling mean (window=3).")
    for col in feature_cols_synth:
        df[col] = df[col].rolling(window=3, min_periods=1, center=True).mean()

print("Smoothing complete.")
train_synthetic.head()
```

```
Smoothing Train features with rolling mean (window=3).
Smoothing Test features with rolling mean (window=3).
Smoothing complete.
```

این بخش ویژگی های مجموعه آموزش دیتاست مصنوعی را با Min-Max Scaling نرمال سازی می کند و همان مقیاس را روی مجموعه تست اعمال می کند. هدف، همسان سازی مقیاس ویژگی ها و آمده سازی داده ها برای مدل پیش بینی است.

4-8. Normalize Features (Synthetic)

```
# Fit scaler on train features only
scaler_synth = MinMaxScaler()
scaler_synth.fit(train_synthetic[feature_cols_synth])

# Transform both train and test features
train_synthetic[feature_cols_synth] = scaler_synth.transform(train_synthetic[feature_cols_synth])
test_synthetic[feature_cols_synth] = scaler_synth.transform(test_synthetic[feature_cols_synth])

print("Normalization complete (fit on train, applied to both).")
train_synthetic.describe()

Normalization complete (fit on train, applied to both).
```

این بخش ویژگی های با واریانس بسیار پایین در دیتاست مصنوعی را شناسایی و از مجموعه های آموزش و تست حذف می کند. هدف، حذف ویژگی های کم اطلاعات برای کاهش پیچیدگی مدل و بهبود عملکرد پیش بینی است.

4-9. Drop Low-Variance Features (Synthetic)

```
# Calculate variance on train features after scaling
variances_synth = train_synthetic[feature_cols_synth].var()
low_var_synth = variances_synth[variances_synth < 0.01].index
if len(low_var_synth) > 0:
    print(f"Dropping low-variance features: {list(low_var_synth)}")
    train_synthetic.drop(low_var_synth, axis=1, inplace=True)
    test_synthetic.drop(low_var_synth, axis=1, inplace=True)
    feature_cols_synth = [col for col in feature_cols_synth if col not in low_var_synth]

print(f"Train shape after drop: {train_synthetic.shape}")
print(f"Test shape after drop: {test_synthetic.shape}")
```

```
Train shape after drop: (3800, 14)
Test shape after drop: (1000, 14)
```

این بخش مجموعه های داده مصنوعی پیش پردازش شده، آموزش و تست را در قالب فایل CSV نخیره می کند. هدف، آماده سازی داده ها برای مرحله بعدی مدل سازی و تضمین دسترسی آسان به نسخه های نهایی داده های مصنوعی است.

4-10. Save Preprocessed Synthetic Files

```
df_synthetic.to_csv("preprocessed_synthetic.csv", index=False)
train_synthetic.to_csv("train_synthetic.csv", index=False)
test_synthetic.to_csv("test_synthetic.csv", index=False)
print("Synthetic data files saved:")
print("- preprocessed_synthetic.csv")
print("- train_synthetic.csv")
print("- test_synthetic.csv")
```

```
Synthetic data files saved:
- preprocessed_synthetic.csv
- train_synthetic.csv
- test_synthetic.csv
```

Model Training

این بخش کتابخانه های مورد نیاز برای مدل سازی، ارزیابی، مصورسازی و استقرار مدل را بازگذاری می کند. مدل های مختلف رگرسیونی، معیار های ارزیابی، ابزار های اعتبار سنجی متقابل و قابلیت تبدیل مدل ها به فرمت ONNX برای اجرا در محیط های تولیدی آماده می شوند. در ادامه، ساختار پوشش های پروژه برای ذخیره مدل ها، نتایج، نمودارها و پیشیبینی ها ایجاد یا بررسی می شود. هدف، سازمان دهنده خروجی ها و آماده سازی پروژه برای مراحل آموزش، ارزیابی و استقرار مدل است.

5. Model Training

5-1. Directory structure setup

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.metrics import (mean_squared_error, mean_absolute_error, r2_score,
                             roc_curve, auc, precision_recall_curve, confusion_matrix,
                             average_precision_score)
from sklearn.model_selection import KFold, cross_val_score
from sklearn.base import clone
from sklearn.preprocessing import label_binarize
from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType
import onnxruntime as ort
import warnings
warnings.filterwarnings('ignore')
```

```
import warnings
warnings.filterwarnings('ignore')

# Set plot style
sns.set(style="whitegrid")
plt.rcParams['figure.figsize'] = (14, 6)

print("SECTION 5.1: CREATING DIRECTORY STRUCTURE")
directories = {
    'models': 'models/onnx',
    'visualizations': 'visualizations',
    'results': 'results',
    'predictions': 'predictions'
}
for name, path in directories.items():
    os.makedirs(path, exist_ok=True)
    print(f"Created/verified: {path}/")
print(f"\nDirectory structure ready!\n")
```

SECTION 5.1: CREATING DIRECTORY STRUCTURE

Created/verified: models/onnx/

Created/verified: visualizations/

Created/verified: results/

Created/verified: predictions/

Directory structure ready!

این بخش داده های پیش پردازش شده ی آموزش و تست برای هر دو دیتاست واقعی و مصنوعی را بارگذاری می کند. ستون هدف مدل مشخص شده و ابعاد هر مجموعه نمایش داده می شود تا صحت بارگذاری داده ها قبل از شروع آموزش و ارزیابی مدل ها بررسی گردد.

5-2. Data Loading

```
print("SECTION 5.2: LOADING PREPROCESSED DATA")
train_real = pd.read_csv("train_real.csv")
test_real = pd.read_csv("test_real.csv")
train_synthetic = pd.read_csv("train_synthetic.csv")
test_synthetic = pd.read_csv("test_synthetic.csv")
target_col = "C6H6(GT)"
print(f"Real Train: {train_real.shape}, Real Test: {test_real.shape}")
print(f"Synthetic Train: {train_synthetic.shape}, Synthetic Test: {test_synthetic.shape}\n")
```

[42] Python

```
... SECTION 5.2: LOADING PREPROCESSED DATA
Real Train: (7296, 13), Real Test: (1895, 13)
Synthetic Train: (3800, 14), Synthetic Test: (1000, 14)
```

این بخش داده ها را برای آموزش و ارزیابی مدل ها آماده می کند. ابتدا ویژگی ها و ستون هدف از دیتاست های واقعی و مصنوعی استخراج شده و مجموعه های آموزش و تست ساخته می شوند. سپس، مقادیر هدف در مجموعه تست به صورت دودویی باینری می شوند تا امکان استفاده از معیارهای ارزیابی شبیه طبقه بندی مانند ROC ، Precision-Recall و Confusion Matrix فراهم شود.

5-3. Data Preparation

```

print("SECTION 5.3: PREPARING FEATURES AND TARGET")
def prepare_data(train_df, test_df, target_col):
    feature_cols = [col for col in train_df.columns
                    if col not in ['DateTime', target_col]]

    X_train = train_df[feature_cols].values
    y_train = train_df[target_col].values

    test_valid = test_df.dropna(subset=[target_col])
    X_test = test_valid[feature_cols].values
    y_test = test_valid[target_col].values

    return X_train, y_train, X_test, y_test, feature_cols

X_train_real, y_train_real, X_test_real, y_test_real, features_real = \
    prepare_data(train_real, test_real, target_col)
X_train_synth, y_train_synth, X_test_synth, y_test_synth, features_synth = \
    prepare_data(train_synthetic, test_synthetic, target_col)

print(f"Real Dataset: Train {X_train_real.shape}, Test {X_test_real.shape}")
print(f"Synthetic Dataset: Train {X_train_synth.shape}, Test {X_test_synth.shape}\n")

# Binarization for classification-style plots (ROC, Confusion Matrix, Precision-Recall)
threshold_real = np.median(y_train_real)
threshold_synth = np.median(y_train_synth)
y_test_bin_real = (y_test_real >= threshold_real).astype(int)
y_test_bin_synth = (y_test_synth >= threshold_synth).astype(int)

```

```

...
... SECTION 5.3: PREPARING FEATURES AND TARGET
Real Dataset: Train (7296, 11), Test (1695, 11)
Synthetic Dataset: Train (3800, 12), Test (933, 12)

Binarized test targets created for classification-style evaluation

```

این بخش مجموعه‌ای از مدل‌های رگرسیونی متنوع را برای مقایسه عملکرد تعریف و پیکربندی می‌کند. مدل‌ها شامل روش‌های خطی، منظم سازی شده، درختی، ensemble SVM و ensemble هستند تا رفتارهای متفاوت مدل‌ها روی داده‌های واقعی و مصنوعی ارزیابی شود.

5-4. Model Definition



```
print("SECTION 5.4: DEFINING REGRESSION MODELS")
models = {
    "Linear Regression": LinearRegression(),
    "Ridge Regression": Ridge(alpha=1.0),
    "Lasso Regression": Lasso(alpha=0.1),
    "Random Forest": RandomForestRegressor(n_estimators=100, max_depth=10, random_state=42),
    "Gradient Boosting": GradientBoostingRegressor(n_estimators=100, max_depth=5, random_state=42),
    "Support Vector": SVR(kernel='rbf', C=100, gamma='scale')
}
print(f"Models configured: {list(models.keys())}\n")
```

[44] ... SECTION 5.4: DEFINING REGRESSION MODELS
Models configured: ['Linear Regression', 'Ridge Regression', 'Lasso Regression', 'Random Forest', 'Gradient Boosting', 'Support Vector']

Python

این بخش مسئول آموزش، اعتبارسنجی و ذخیره سازی مدل ها برای هر دو دیتاست واقعی و مصنوعی است. در ابتدا، مدل ها به صورت جدآگانه روی داده های واقعی و مصنوعی آموزش داده می شوند تا امکان مقایسه عملکرد آن ها فراهم شود. برای هر مدل، از Cross-Validation پنج تایی (K-Fold) استفاده می شود و معیار های RMSE و R^2 محاسبه می گرددند تا پایداری و قابلیت تعمیم مدل ارزیابی شود.

پس از آموزش، هر مدل به فرمت ONNX تبدیل و ذخیره می شود تا امکان استقرار و اجرای مستقل از پلتفرم مثلًا در سیستم های IoT یا Edge فراهم گردد. در نهایت، مسیر فایل های ذخیره شده و نتایج اعتبارسنجی برای استفاده در مراحل ارزیابی و استقرار ثبت می شوند.

5-5. Model Training & Saving To ONNX

```
print("SECTION 5.5: MODEL TRAINING & SAVING TO ONNX")

onnx_files = {}
cv_results = {}

for dataset_name, X_train, y_train, features in [
    ("real", X_train_real, y_train_real, features_real),
    ("synthetic", X_train_synth, y_train_synth, features_synth)
]:
    print(f"\n--- Training models on {dataset_name.upper()} data ---")

    for model_name, model_template in models.items():
        print(f"\nTraining {model_name}...")
        model = clone(model_template)
        model.fit(X_train, y_train)

        # Cross-validation
        kf = KFold(n_splits=5, shuffle=True, random_state=42)
        cv_rmse = np.sqrt(-cross_val_score(model, X_train, y_train, cv=kf, scoring='neg_mean_squared_error'))
        cv_r2 = cross_val_score(model, X_train, y_train, cv=kf, scoring='r2')
        print(f" CV RMSE: {cv_rmse.mean():.4f} ± {cv_rmse.std():.4f}")
        print(f" CV R²: {cv_r2.mean():.4f} ± {cv_r2.std():.4f}")

        cv_results[(model_name, dataset_name)] = (cv_rmse, cv_r2)

    onnx_files[dataset_name] = save_model(model, f"model_{dataset_name}.onnx")
```

```
# ONNX Export
safe_name = model_name.replace(" ", "_").replace("/", "_")
onnx_path = f"models/onnx/{safe_name}_{dataset_name}.onnx"
initial_type = [('float_input', FloatTensorType([None, X_train.shape[1]]))]

try:
    onnx_model = convert_sklearn(model, initial_types=initial_type)
    with open(onnx_path, "wb") as f:
        f.write(onnx_model.SerializeToString())
    print(f"Exported ONNX: {onnx_path}")
    onnx_files[(model_name, dataset_name)] = onnx_path
except Exception as e:
    print(f"Export failed: {e}")

print(f"TRAINING & ONNX EXPORT COMPLETE - {len(onnx_files)} models saved")
```

خروجی ها:

```
... SECTION 5.5: MODEL TRAINING & SAVING TO ONNX

--- Training models on REAL data ---

Training Linear Regression...
CV RMSE: 2.1157 ± 0.0977
CV R2: 0.9219 ± 0.0041
Exported ONNX: models/onnx/Linear_Regression_real.onnx

Training Ridge Regression...
CV RMSE: 2.1246 ± 0.1132
CV R2: 0.9213 ± 0.0050
Exported ONNX: models/onnx/Ridge_Regression_real.onnx

Training Lasso Regression...
CV RMSE: 2.3651 ± 0.1400
CV R2: 0.9026 ± 0.0049
Exported ONNX: models/onnx/Lasso_Regression_real.onnx

Training Random Forest...
CV RMSE: 1.7599 ± 0.0475
CV R2: 0.9459 ± 0.0023
Exported ONNX: models/onnx/Random_Forest_real.onnx

Training Gradient Boosting...
CV RMSE: 1.7853 ± 0.0621
CV R2: 0.9444 ± 0.0013
Exported ONNX: models/onnx/Gradient_Boosting_real.onnx
```

```
... Training Linear Regression...
  CV RMSE: 2.2737 ± 0.0516
  CV R²: 0.5640 ± 0.0274
Exported ONNX: models/onnx/Linear_Regression_synthetic.onnx

Training Ridge Regression...
  CV RMSE: 2.2732 ± 0.0514
  CV R²: 0.5641 ± 0.0274
Exported ONNX: models/onnx/Ridge_Regression_synthetic.onnx

Training Lasso Regression...
  CV RMSE: 2.3568 ± 0.0408
  CV R²: 0.5322 ± 0.0131
Exported ONNX: models/onnx/Lasso_Regression_synthetic.onnx

Training Random Forest...
  CV RMSE: 2.3088 ± 0.0449
  CV R²: 0.5507 ± 0.0229
Exported ONNX: models/onnx/Random_Forest_synthetic.onnx

Training Gradient Boosting...
  CV RMSE: 2.3414 ± 0.0467
  CV R²: 0.5380 ± 0.0215
Exported ONNX: models/onnx/Gradient_Boosting_synthetic.onnx

Training Support Vector...
  CV RMSE: 2.4925 ± 0.0338
  CV R²: 0.4764 ± 0.0220
```

این بخش مدل‌های ذخیره شده در قالب ONNX را بازگذاری کرده و با استفاده از آن‌ها روی داده‌های تست واقعی و مصنوعی پیش‌بینی انجام می‌دهد. برای هر مدل، پیش‌بینی‌ها اجرا شده و معیارهای ارزیابی شامل MAE و R^2 محاسبه می‌شوند تا عملکرد نهایی مدل‌ها بررسی گردد. همچنین نتایج اعتبارسنجی متقابل مرحله آموزش با نتایج تست مقایسه شده و تمام خروجی‌ها (پیش‌بینی‌ها و مقادیر واقعی) در فایل‌های CSV ذخیره می‌شوند. هدف، شبیه سازی اجرای مدل در محیط تولید و آماده سازی داده‌ها برای تحلیل و مصورسازی نهایی است.

5-6. Load ONNX Models & Predict

```
print("SECTION 5.6: LOAD ONNX MODELS & PRODUCE PREDICTIONS")

onnx_predictions_real = {}
onnx_predictions_synth = {}
results_real = []
results_synth = []

for (model_name, dataset_name), onnx_path in onnx_files.items():
    try:
        print(f"\nLoading: {onnx_path}")
        session = ort.InferenceSession(onnx_path)
        input_name = session.get_inputs()[0].name

        # Select correct test set
        X_test = X_test_real if dataset_name == "real" else X_test_synth
        y_test = y_test_real if dataset_name == "real" else y_test_synth
        y_test_bin = y_test_bin_real if dataset_name == "real" else y_test_bin_synth

        X_test_f32 = X_test.astype(np.float32)
        output = session.run(None, {input_name: X_test_f32})
        y_pred_onnx = np.array(output[0]).flatten()

        print(f"Predictions generated ({len(y_pred_onnx)} samples)")

    # Store predictions
    if dataset_name == "real":
        onnx_predictions_real[model_name] = y_pred_onnx
    else:
        onnx_predictions_synth[model_name] = y_pred_onnx

    # Compute metrics for later use
    test_rmse = np.sqrt(mean_squared_error(y_test, y_pred_onnx))
    test_mae = mean_absolute_error(y_test, y_pred_onnx)
    test_r2 = r2_score(y_test, y_pred_onnx)

    # Get CV from earlier
    cv_rmse_scores, cv_r2_scores = cv_results.get((model_name, dataset_name), (None, None))
    cv_rmse_mean = cv_rmse_scores.mean() if cv_rmse_scores is not None else None
    cv_r2_mean = cv_r2_scores.mean() if cv_r2_scores is not None else None

```

```
        result = {
            'model_name': model_name,
            'dataset': dataset_name,
            'y_test': y_test,
            'y_test_bin': y_test_bin,
            'y_pred_onnx': y_pred_onnx,
            'test_rmse': test_rmse,
            'test_mae': test_mae,
            'test_r2': test_r2,
            'cv_rmse_mean': cv_rmse_mean,
            'cv_rmse_scores': cv_rmse_scores,
            'cv_r2_scores': cv_r2_scores,
            'onnx_file': onnx_path
        }

        if dataset_name == "real":
            results_real.append(result)
        else:
            results_synth.append(result)

    except Exception as e:
        print(f"Failed to load/run {onnx_path}: {e}")
```

```
# Save predictions
os.makedirs("predictions", exist_ok=True)
if onnx_predictions_real:
    pd.DataFrame(onnx_predictions_real, index=range(len(y_test_real))) \
        .assign(Actual=y_test_real).to_csv("predictions/onnx_predictions_real.csv", index=False)
    print("Real ONNX predictions saved")
if onnx_predictions_synth:
    pd.DataFrame(onnx_predictions_synth, index=range(len(y_test_synth))) \
        .assign(Actual=y_test_synth).to_csv("predictions/onnx_predictions_synthetic.csv", index=False)
    print("Synthetic ONNX predictions saved")

print("ONNX LOADING & PREDICTION COMPLETE")
```

این بخش پیشینی های تولید شده توسط مدل های ONNX را به صورت نهایی ذخیره می کند. برای هر دیتاست واقعی و مصنوعی، خروجی پیشینی مدل ها در کنار مقدار واقعی هدف در یک فایل CSV ذخیره می شود. هدف، ایجاد خروجی استاندارد و قابل استفاده برای ارزیابی، مقایسه مدل ها و مصورسازی نتایج نهایی است.

5-7. Save ONNX Predictions To CSV

```
print("SAVING ONNX PREDICTIONS")
os.makedirs("predictions", exist_ok=True)

# Real data
if results_real:
    onnx_pred_real = {r['model_name']: r['y_pred_onnx'] for r in results_real}
    df_onnx_real = pd.DataFrame(onnx_pred_real)
    df_onnx_real['Actual'] = y_test_real
    df_onnx_real.to_csv("predictions/onnx_predictions_real.csv", index=False)
    print("Real ONNX predictions saved")
else:
    print("No successful ONNX predictions for real data")

# Synthetic data
if results_synth:
    onnx_pred_synth = {r['model_name']: r['y_pred_onnx'] for r in results_synth}
    df_onnx_synth = pd.DataFrame(onnx_pred_synth)
    df_onnx_synth['Actual'] = y_test_synth
    df_onnx_synth.to_csv("predictions/onnx_predictions_synthetic.csv", index=False)
    print("Synthetic ONNX predictions saved")
else:
    print("No successful ONNX predictions for synthetic data")
```

Visualizaion & Evaluation

در این بخش، برای هر دیتاست (واقعی و مصنوعی) مجموعه‌ای از ۳ نمودار میله‌ای ترسیم می‌شود که عملکرد مدل‌های مختلف را پس از اجرای ONNX مقایسه می‌کنند.

تعداد و نوع نمودار‌ها

برای هر دیتاست:

1. نمودار RMSE :

میزان خطای پیش‌بینی مدل‌ها را نشان می‌دهد؛ هرچه مقدار کمتر باشد، عملکرد مدل بهتر است.

2. نمودار MAE :

میانگین خطای مطلق پیش‌بینی‌ها را نمایش می‌دهد و دید شهودی تری از خطای واقعی مدل‌ها ارائه می‌کند.

3. نمودار R^2 :

توانایی مدل در توضیح تغییرات داده هدف را نشان می‌دهد؛ مقدار نزدیک‌تر به ۱ بیانگر عملکرد بهتر است.

تصاویر از کد‌ها در صفحات بعد:

6-1. Metrics Comparison

```
print("SECTION 6.1: VISUALIZATION - METRICS COMPARISON")

def plot_metrics_comparison(results, dataset_name):
    models_list = [r['model_name'] for r in results]
    rmse_vals = [r['test_rmse'] for r in results]
    mae_vals = [r['test_mae'] for r in results]
    r2_vals = [r['test_r2'] for r in results]

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

    axes[0].bar(models_list, rmse_vals, color='salmon', edgecolor='black')
    axes[0].set_title(f'RMSE Comparison (ONNX) - {dataset_name}', fontsize=14, fontweight='bold')
    axes[0].set_ylabel('RMSE', fontsize=12)
    axes[0].tick_params(axis='x', rotation=45)
    axes[0].grid(axis='y', alpha=0.3)

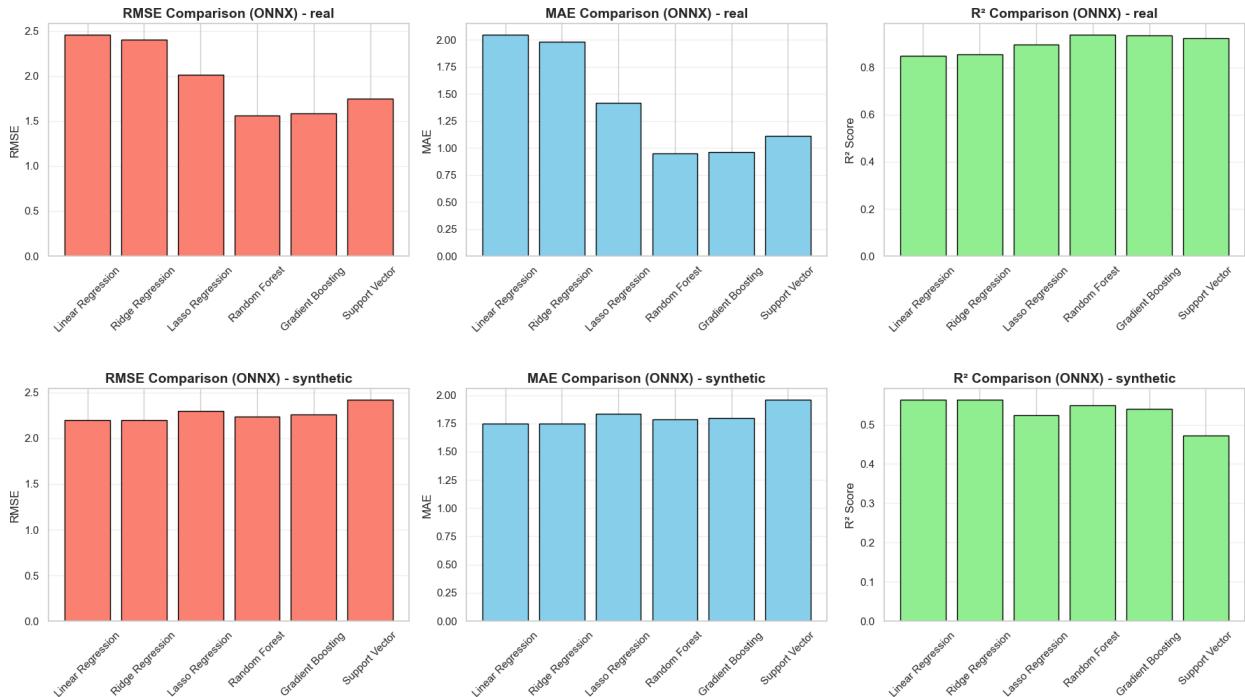
    axes[1].bar(models_list, mae_vals, color='skyblue', edgecolor='black')
    axes[1].set_title(f'MAE Comparison (ONNX) - {dataset_name}', fontsize=14, fontweight='bold')
    axes[1].set_ylabel('MAE', fontsize=12)
    axes[1].tick_params(axis='x', rotation=45)
    axes[1].grid(axis='y', alpha=0.3)

    axes[2].bar(models_list, r2_vals, color='lightgreen', edgecolor='black')
    axes[2].set_title(f'R2 Comparison (ONNX) - {dataset_name}', fontsize=14, fontweight='bold')
    axes[2].set_ylabel('R2 Score', fontsize=12)
    axes[2].tick_params(axis='x', rotation=45)
    axes[2].grid(axis='y', alpha=0.3)

    plt.tight_layout()
    plt.savefig(f"visualizations/metrics_comparison_onnx_{dataset_name}.png", dpi=300)
    plt.show()
    plt.close()

if results_real:
    plot_metrics_comparison(results_real, "real")
if results_synth:
    plot_metrics_comparison(results_synth, "synthetic")
print(" ONNX metrics comparison charts saved")
```

تصویر نمودار های رسم شده در صفحه بعد:



این بخش، پیشینی های مدل های ONNX را در مقابل مقادیر واقعی هدف به صورت بصری نمایش می دهد تا دقیقت و کیفیت پیشینی ها بررسی شود.

نحوه ترسیم نمودارها

- برای هر دیتاست (واقعی و مصنوعی)، یک شبکه از نمودارها با ۲ ردیف و ۳ ستون ایجاد می شود.
 - هر نمودار مربوط به یک مدل رگرسیونی است.
 - روی هر نمودار، نقاط آبی رنگ پیشینی ها در مقابل مقادیر واقعی ترسیم می شوند.
 - خط قرمز مورب ($y=x$) نشان دهنده پیشینی ایدهآل است؛ هرچه نقاط به این خط نزدیکتر باشند، عملکرد مدل بهتر است.

برای ۶ مدل تعریف شده، ۶ نمودار جداگانه در یک شبکه ترسیم می شود.

تصاویر کد ها و همچنین خروجی (نمودارها) در صفحات بعد :

```

print("SECTION 6.2: VISUALIZATION - PREDICTIONS VS ACTUAL")
def plot_predictions_vs_actual(results, dataset_name):
    fig, axes = plt.subplots(2, 3, figsize=(18, 10))
    axes = axes.flatten()

    for i, r in enumerate(results):
        ax = axes[i]
        # USE ONNX PREDICTIONS FOR PLOTTING
        ax.scatter(r['y_test'], r['y_pred_onnx'], alpha=0.6, s=20, color='dodgerblue')

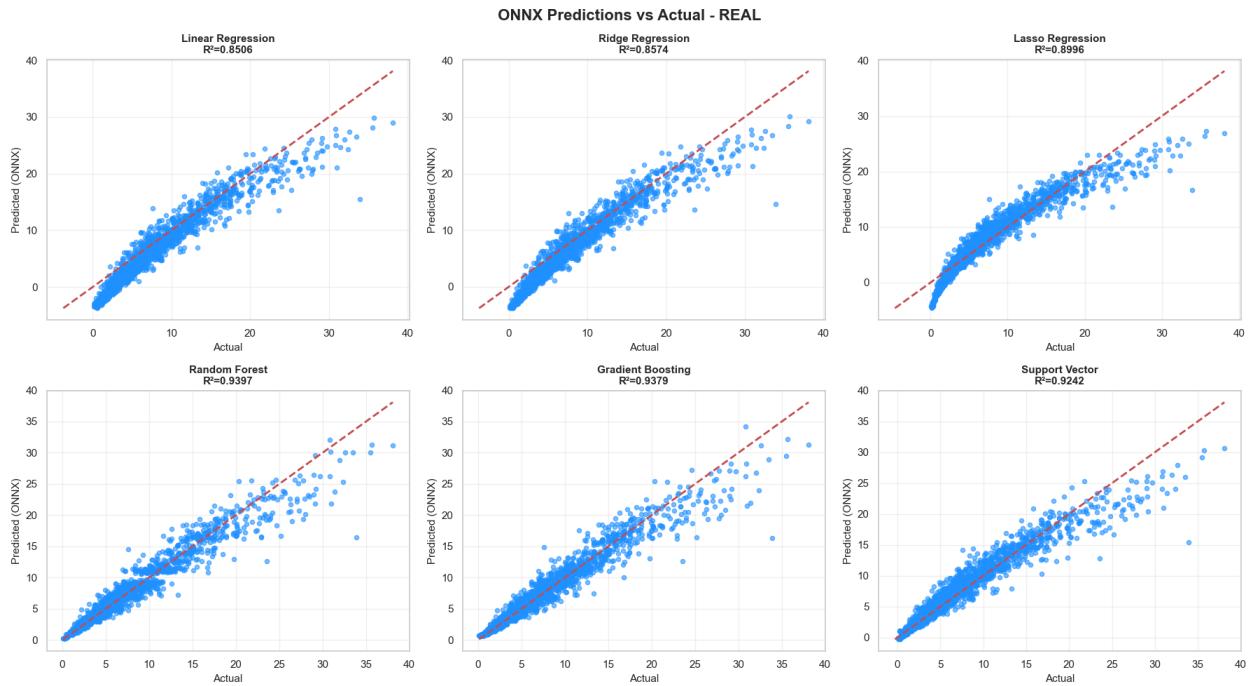
        min_val = min(r['y_test'].min(), r['y_pred_onnx'].min())
        max_val = max(r['y_test'].max(), r['y_pred_onnx'].max())
        ax.plot([min_val, max_val], [min_val, max_val], 'r--', linewidth=2)

        ax.set_xlabel('Actual', fontsize=11)
        ax.set_ylabel('Predicted (ONNX)', fontsize=11)
        ax.set_title(f'{r["model_name"]}\nR^2={r["test_r2"]:.4f}", fontsize=11, fontweight='bold')
        ax.grid(alpha=0.3)

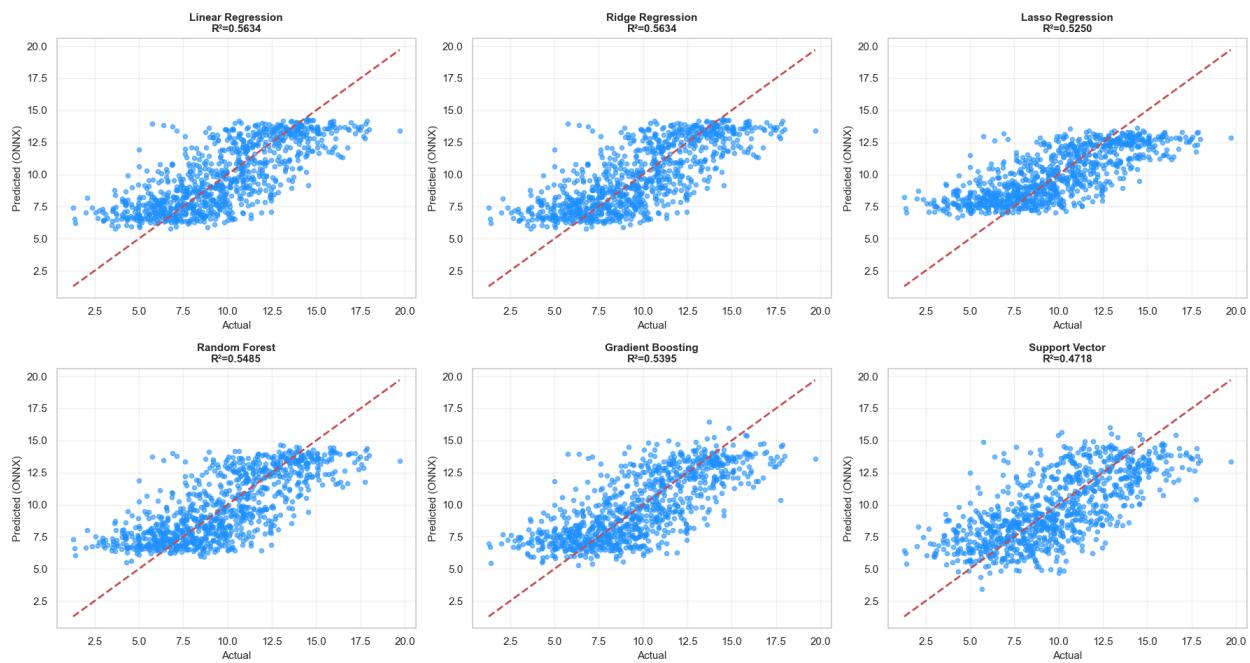
    plt.suptitle(f'ONNX Predictions vs Actual - {dataset_name.upper()}', fontsize=16, fontweight='bold')
    plt.tight_layout()
    plt.savefig(f"visualizations/onnx_predictions_vs_actual_{dataset_name}.png", dpi=300)
    plt.show()
    plt.close()

if results_real:
    plot_predictions_vs_actual(results_real, "real")
if results_synth:
    plot_predictions_vs_actual(results_synth, "synthetic")
print("ONNX predictions vs actual plots saved")

```



ONNX Predictions vs Actual - SYNTHETIC



این بخش به تحلیل Residuals (خطاهای پیش‌بینی) مدل های ONNX اختصاص دارد تا خطاهای مدل به صورت دقیق‌تر و بصری بررسی شوند.

نحوه ترسیم نمودارها

- برای هر دیتاست (واقعی و مصنوعی)، یک شبکه از نمودارها با ۲ ردیف و ۳ ستون ایجاد می‌شود.
- هر نمودار مربوط به یک مدل رگرسیونی است.
- محور X : مقادیر پیش‌بینی شده توسط مدل (Predicted ONNX)
- محور Y : $\text{Residuals} = \text{Actual} - \text{Predicted}$
- یک خط افقی در $y=0$ رسم می‌شود تا مرجع خطای صفر مشخص گردد.
- نقاط قرمز نشان دهنده پراکندگی خطاهای هستند؛ پراکندگی نزدیک به خط صفر، دقت بهتر مدل را نشان می‌دهد.

هدف نمودارها

- بررسی اینکه آیا خطاهایا به صورت تصادفی پخش شده اند یا الگوی سیستماتیک دارند.
- شناسایی مدل‌هایی با خطاهای بزرگ یا انحراف‌های خاص.

تصاویر کد و نمودارها در صفحات بعدی:

6-3. Residuals Analysis

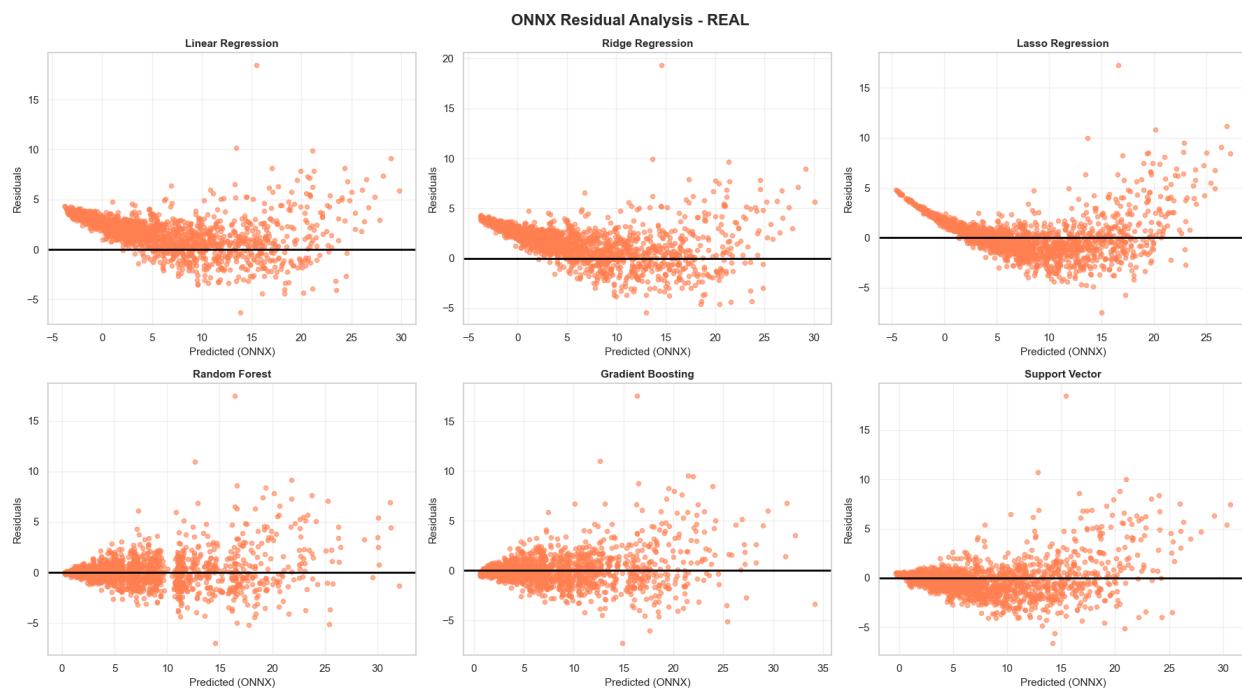
```
print("SECTION 6.3: VISUALIZATION - RESIDUALS ANALYSIS")
def plot_residuals(results, dataset_name):
    fig, axes = plt.subplots(2, 3, figsize=(18, 10))
    axes = axes.flatten()

    for i, r in enumerate(results):
        ax = axes[i]
        # USE ONNX PREDICTIONS FOR RESIDUALS
        residuals = r['y_test'] - r['y_pred_onnx']

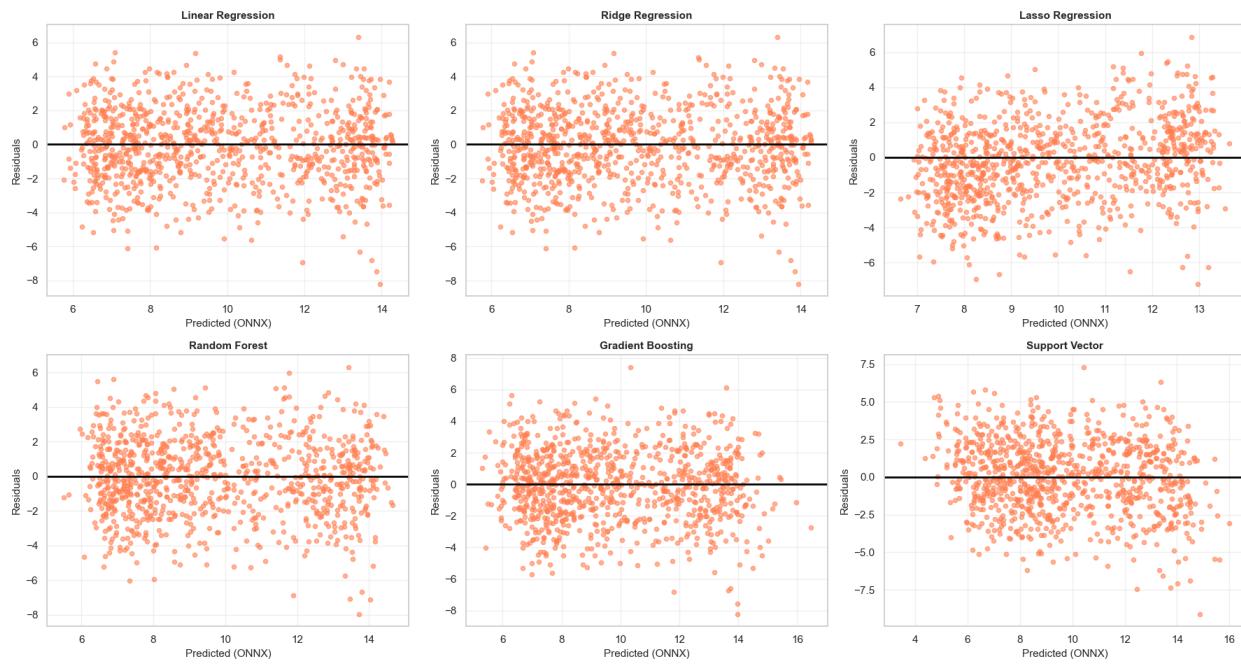
        ax.scatter(r['y_pred_onnx'], residuals, alpha=0.6, s=20, color='coral')
        ax.axhline(y=0, color='black', linestyle='-', linewidth=2)
        ax.set_xlabel('Predicted (ONNX)', fontsize=11)
        ax.set_ylabel('Residuals', fontsize=11)
        ax.set_title(f"{r['model_name']}", fontsize=11, fontweight='bold')
        ax.grid(alpha=0.3)

    plt.suptitle(f'ONNX Residual Analysis - {dataset_name.upper()}', fontsize=16, fontweight='bold')
    plt.tight_layout()
    plt.savefig(f"visualizations/onnx_residuals_analysis_{dataset_name}.png", dpi=300)
    plt.show()
    plt.close()

if results_real:
    plot_residuals(results_real, "real")
if results_synth:
    plot_residuals(results_synth, "synthetic")
print(" ONNX residual analysis plots saved")
```



ONNX Residual Analysis - SYNTHETIC



این بخش به تحلیل و مصورسازی اهمیت ویژگی ها (Feature Importance) مدل های آموزش دیده اختصاص دارد.

نحوه عملکرد:

1. برای هر مدل، یک نسخه تازه از مدل روی داده های آموزش بازآموزی می شود تا وزن ها یا شاخص اهمیت ویژگی ها به دست آید.
2. ایجاد DataFrame شامل نام ویژگی ها و میزان اهمیت و مرتب سازی به ترتیب نزولی.
3. ترسیم نمودار برای ۱۰ ویژگی برتر هر مدل در شبکه ای با ۲ ردیف و ۳ ستون.

6-4. Feature Importance

```
print("SECTION 6.4: VISUALIZATION - FEATURE IMPORTANCE")

def plot_feature_importance(results, dataset_name, feature_names):
    models_with_importance = []

    for r in results:
        model_name = r['model_name']

        # Get corresponding training data
        X_train = X_train_real if dataset_name == "real" else X_train_synth

        # Retrain a fresh model
        model_template = models[model_name]
        model = clone(model_template)
        model.fit(X_train, y_train_real if dataset_name == "real" else y_train_synth)

        # Extract importance
        if hasattr(model, 'feature_importances_'):
            importance = model.feature_importances_
        elif hasattr(model, 'coef_'):
            importance = np.abs(model.coef_)
        else:
            continue

        # Create DataFrame
        imp_df = pd.DataFrame({
            'feature': feature_names,
            'importance': importance
        }).sort_values('importance', ascending=False)
```

```

# Attach to result
enriched_r = r.copy()
enriched_r['feature_importance'] = imp_df
models_with_importance.append(enriched_r)

if len(models_with_importance) == 0:
    print(f"No models with feature importance available for {dataset_name}")
    return

# Plotting
fig, axes = plt.subplots(2, 3, figsize=(18, 10))
axes = axes.flatten()

for i, r in enumerate(models_with_importance):
    ax = axes[i]
    top_features = r['feature_importance'].head(10)

    ax.barh(range(len(top_features)), top_features['importance'], color='steelblue')
    ax.set_yticks(range(len(top_features)))
    ax.set_yticklabels(top_features['feature'])
    ax.invert_yaxis()
    ax.set_xlabel('Importance', fontsize=11)
    ax.set_title(f"{r['model_name']}", fontsize=11, fontweight='bold')
    ax.grid(axis='x', alpha=0.3)

```

```

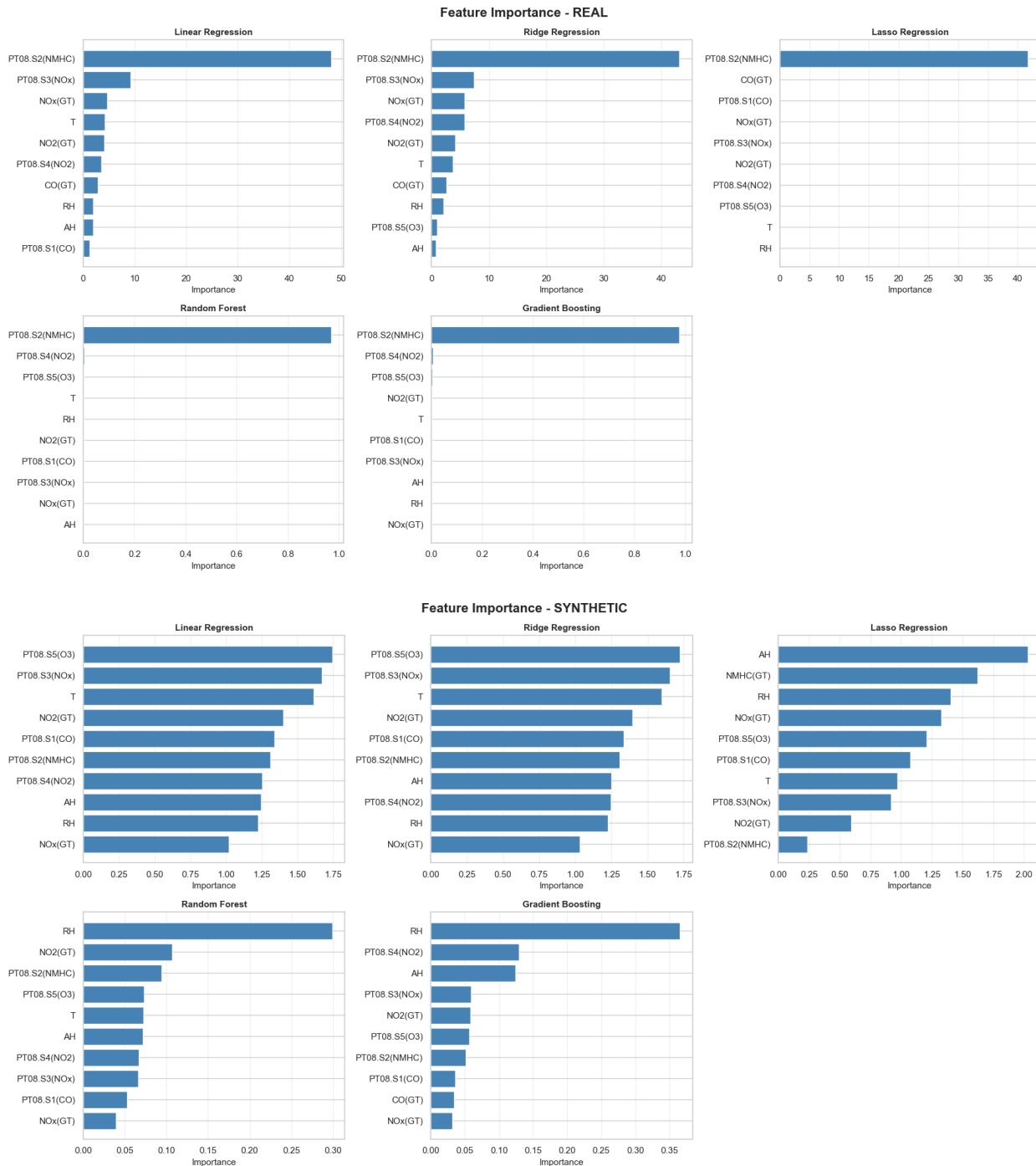
# Save to CSV
os.makedirs("results", exist_ok=True)
for r in models_with_importance:
    safe_name = r['model_name'].replace(' ', '_').replace('/', '_')
    filename = f"results/feature_importance_{safe_name}_{dataset_name}.csv"
    r['feature_importance'].to_csv(filename, index=False)

print(f"Feature importance plotted and saved for {len(models_with_importance)} models ({dataset_name})")

# Call the function with correct feature names
if results_real:
    plot_feature_importance(results_real, "real", features_real)
if results_synth:
    plot_feature_importance(results_synth, "synthetic", features_synth)

```

تصاویر نمودار ها در صفحه بعدی:



این بخش تحلیل و مصوّر سازی توزیع خطاها پیشینی (Error Distribution) مدل های ONNX را انجام می دهد.

نحوه عملکرد:

- برای هر مدل، خطای مطلق بین مقادیر واقعی و پیشینی های ONNX محاسبه می شود :

$$\text{Error} = | Y(\text{test}) - Y(\text{pred}) |$$

- نمودار هیستوگرام خطاها ترسیم می شود تا پراکندگی و میزان خطا در کل نمونه ها مشاهده شود.
- خطوط عمودی روی نمودار قرار می گیرند:
 - خط قرمز بیان گر: میانگین خطا
 - خط سبز بیان گر : میانه خطا

ساختار نمودارها:

- برای هر دیتا ست، یک شبکه با ۲ ردیف و ۳ ستون ایجاد می شود که هر نمودار مربوط به یک مدل است.

هدف نمودارها :

- شناسایی مدل هایی با توزیع خطا فشرده و پایدار
- مقایسه عملکرد مدل ها از نظر پراکندگی و مقدار متوسط خطا

6-5. Error Distribution

```
print("SECTION 6.5: VISUALIZATION - ERROR DISTRIBUTION")

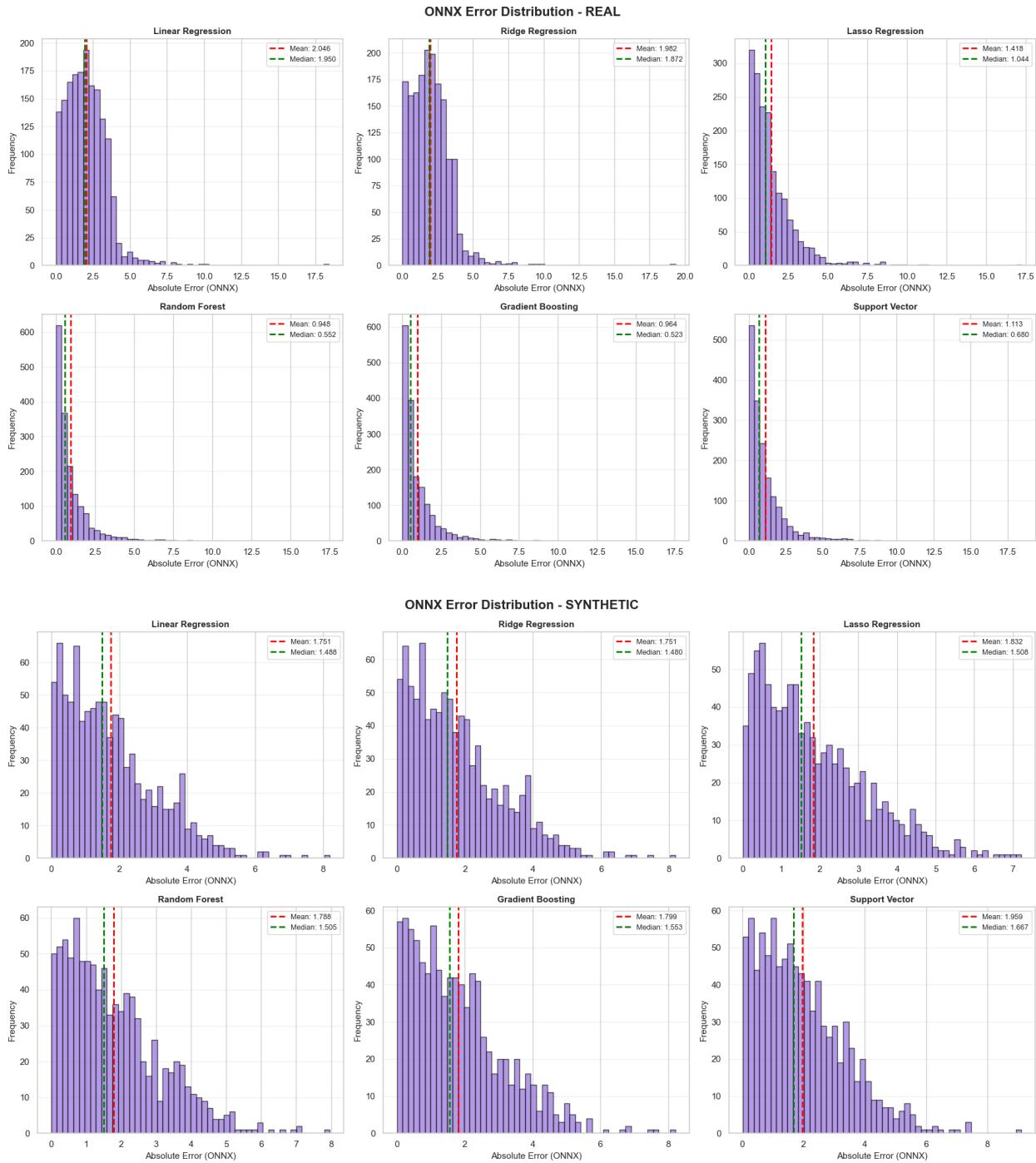
def plot_error_distribution(results, dataset_name):
    fig, axes = plt.subplots(2, 3, figsize=(18, 10))
    axes = axes.flatten()

    for i, r in enumerate(results):
        ax = axes[i]
        # use onnx predictions for errors
        errors = np.abs(r['y_test'] - r['y_pred_onnx'])
        mean_error = np.mean(errors)
        median_error = np.median(errors)

        ax.hist(errors, bins=50, color='mediumpurple', edgecolor='black', alpha=0.7)
        ax.axvline(mean_error, color='red', linestyle='--', linewidth=2,
                   label=f'Mean: {mean_error:.3f}')
        ax.axvline(median_error, color='green', linestyle='--', linewidth=2,
                   label=f'Median: {median_error:.3f}')

        ax.set_xlabel('Absolute Error (ONNX)', fontsize=11)
        ax.set_ylabel('Frequency', fontsize=11)
        ax.set_title(f'{r["model_name"]}', fontsize=11, fontweight='bold')
        ax.legend(fontsize=9)
        ax.grid(axis='y', alpha=0.3)

    plt.suptitle(f'ONNX Error Distribution - {dataset_name.upper()}', fontsize=16, fontweight='bold')
    plt.tight_layout()
    plt.savefig(f"visualizations/onnx_error_distribution_{dataset_name}.png", dpi=300)
    plt.show()
    plt.close()
```



این بخش به تحلیل ROC Curves (Receiver Operating Characteristic) برای مدل های ONNX اختصاص دارد تا توانایی مدل ها در تفکیک نمونه های بالاتر و پایین تر از یک Threshold هدف بررسی شود.

نحوه عملکرد:

- ابتدا مقادیر واقعی تست باینری (y_{test_bin}) با پیشビینی های مدل مقایسه می شوند.
- برای هر مدل، True Positive Rate (TPR) و False Positive Rate (FPR) محاسبه شده و نمودار ROC رسم می شود.
- AUC (Area Under Curve) هر مدل نیز محاسبه و در برچسب نمودار نشان داده می شود.
- یک خط مورب مشکی با $AUC=0.5$ به عنوان مرجع تصادفی رسم می شود.

هدف نمودارها:

- بررسی توانایی مدل ها در تفکیک نمونه ها (مانند تشخیص مقادیر بالاتر و پایین تر از میانه)
- مقایسه مدل ها از نظر AUC
- ارزیابی کیفیت پیشビینی مدل ها پس از تبدیل به ONNX ، حتی در قالب باینری

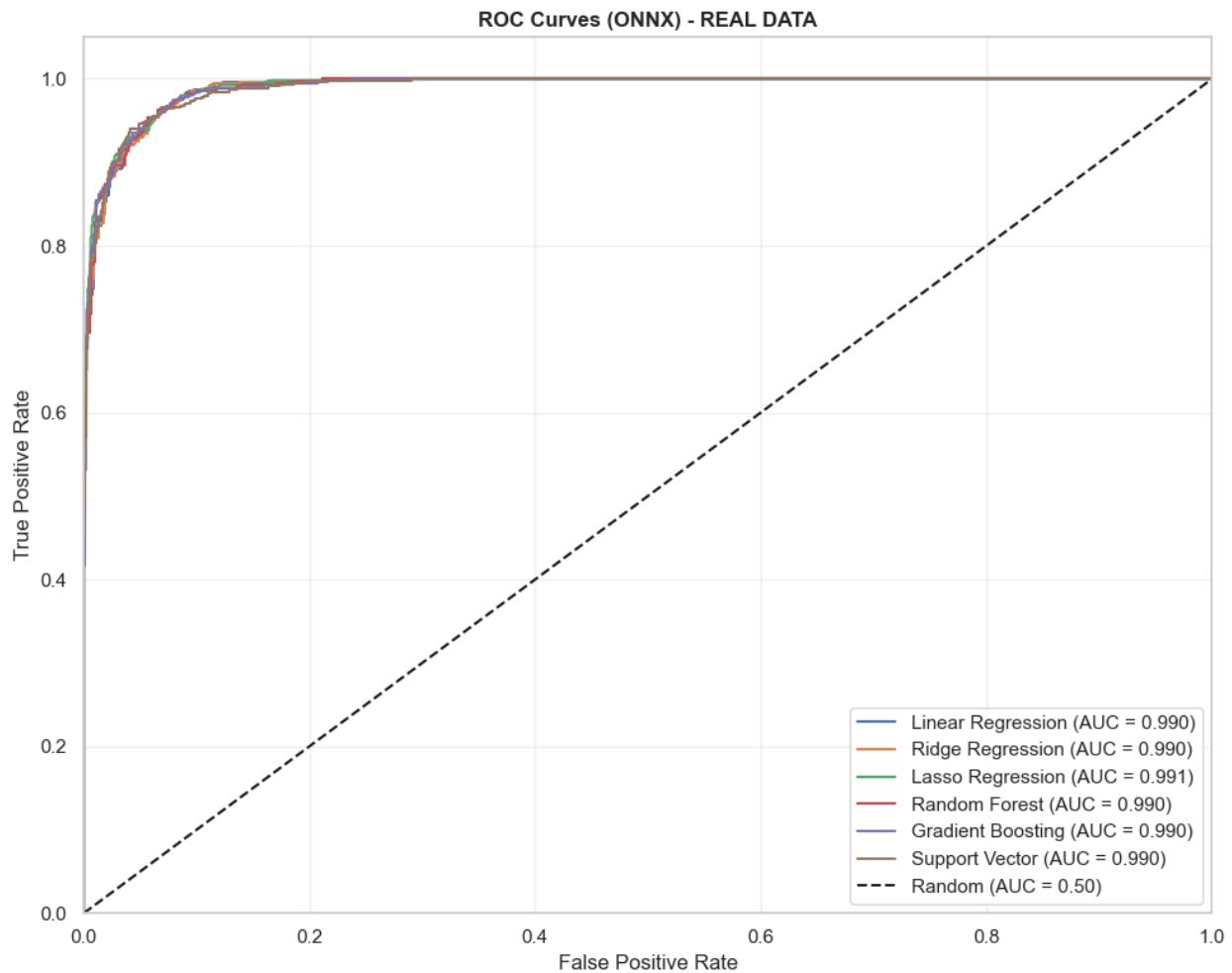
6-6. Comparison ROC Curves Using ONNX Predictions

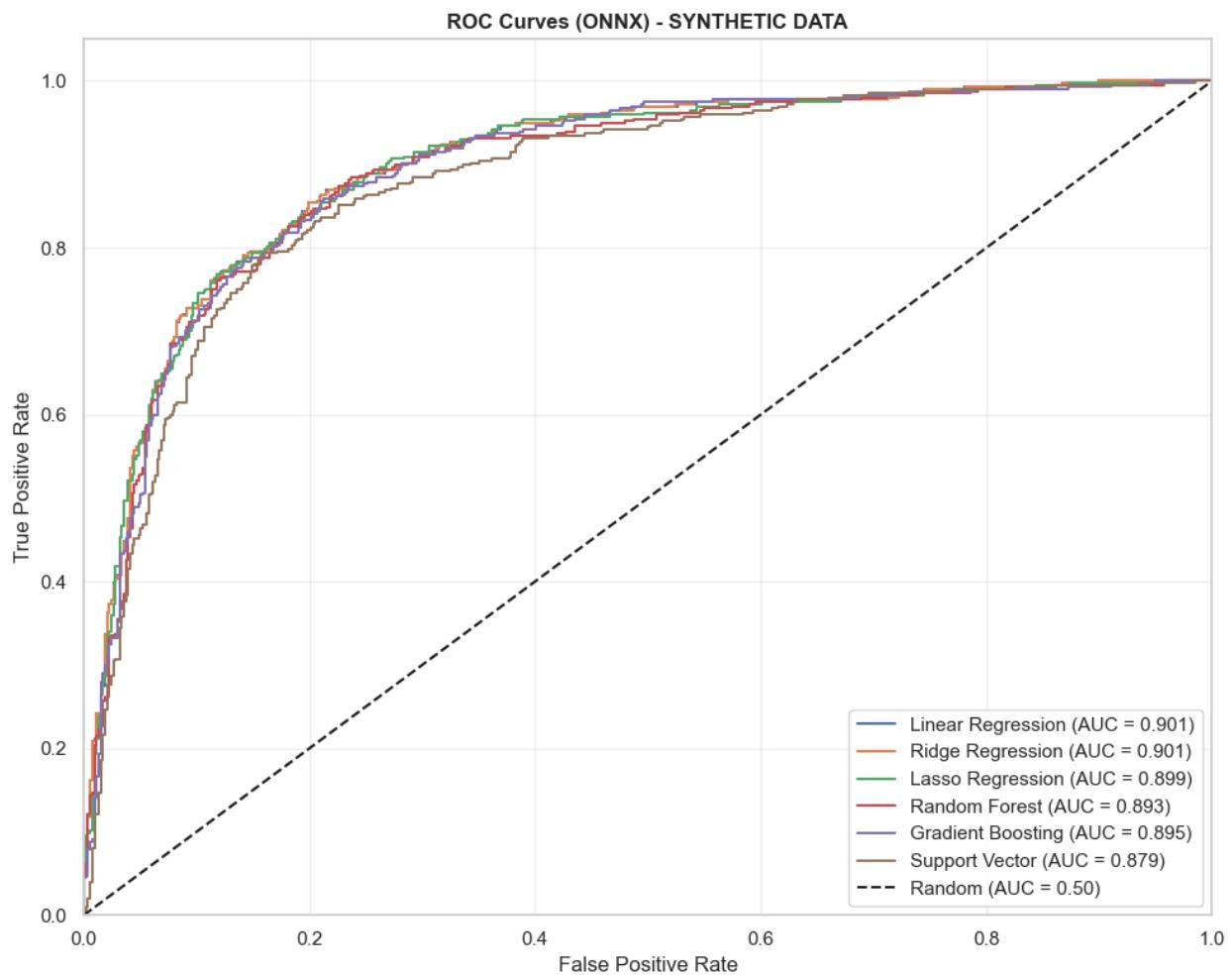
```
print("SECTION 6.6: COMPARISON ROC CURVES (ONNX PREDICTIONS)")

def plot_roc_comparison(results, y_test_bin, dataset_name):
    plt.figure(figsize=(10, 8))
    for r in results:
        #use onnx predictions for roc
        fpr, tpr, _ = roc_curve(y_test_bin, r['y_pred_onnx'])
        roc_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, label=f"{r['model_name']} (AUC = {roc_auc:.3f})")

    plt.plot([0, 1], [0, 1], 'k--', label='Random (AUC = 0.50)')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curves (ONNX) - {dataset_name.upper()} DATA', fontweight='bold')
    plt.legend(loc="lower right")
    plt.grid(alpha=0.3)
    plt.tight_layout()
    plt.savefig(f"visualizations/onnx_roc_comparison_{dataset_name}.png", dpi=300)
    plt.show()
    plt.close()

if results_real:
    plot_roc_comparison(results_real, y_test_bin_real, "real")
if results_synth:
    plot_roc_comparison(results_synth, y_test_bin_synth, "synthetic")
print(" ONNX ROC comparison plots saved")
```





این بخش به تحلیل Precision-Recall (PR) Curves مدل های ONNX اختصاص دارد تا کیفیت پیشビینی ها در تفکیک نمونه های مثبت و منفی بررسی شود، به ویژه در شرایط عدم تعادل داده.

نحوه عملکرد:

- برای هر مدل، مقادیر Precision و Recall از پیشビینی های ONNX نسبت به برچسب های باینری (y_{test_bin}) محاسبه می شوند.
- همچنین Average Precision (AP) برای هر مدل محاسبه و در برچسب نمودار نمایش داده می شود.
- نمودار Precision-Recall رسم می شود تا عملکرد مدل ها در بازیابی نمونه های مثبت و حفظ دقت آنها مشاهده شود.

هدف نمودار ها:

- ارزیابی عملکرد مدل ها در شرایطی که توزیع کلاس ها نامتوازن است.
- مقایسه مدل ها از نظر توازن بین دقت و بازیابی.
- تمکیل تحلیل عملکرد مدل های ONNX در کنار ROC ، RMSE و سایر معیارها.

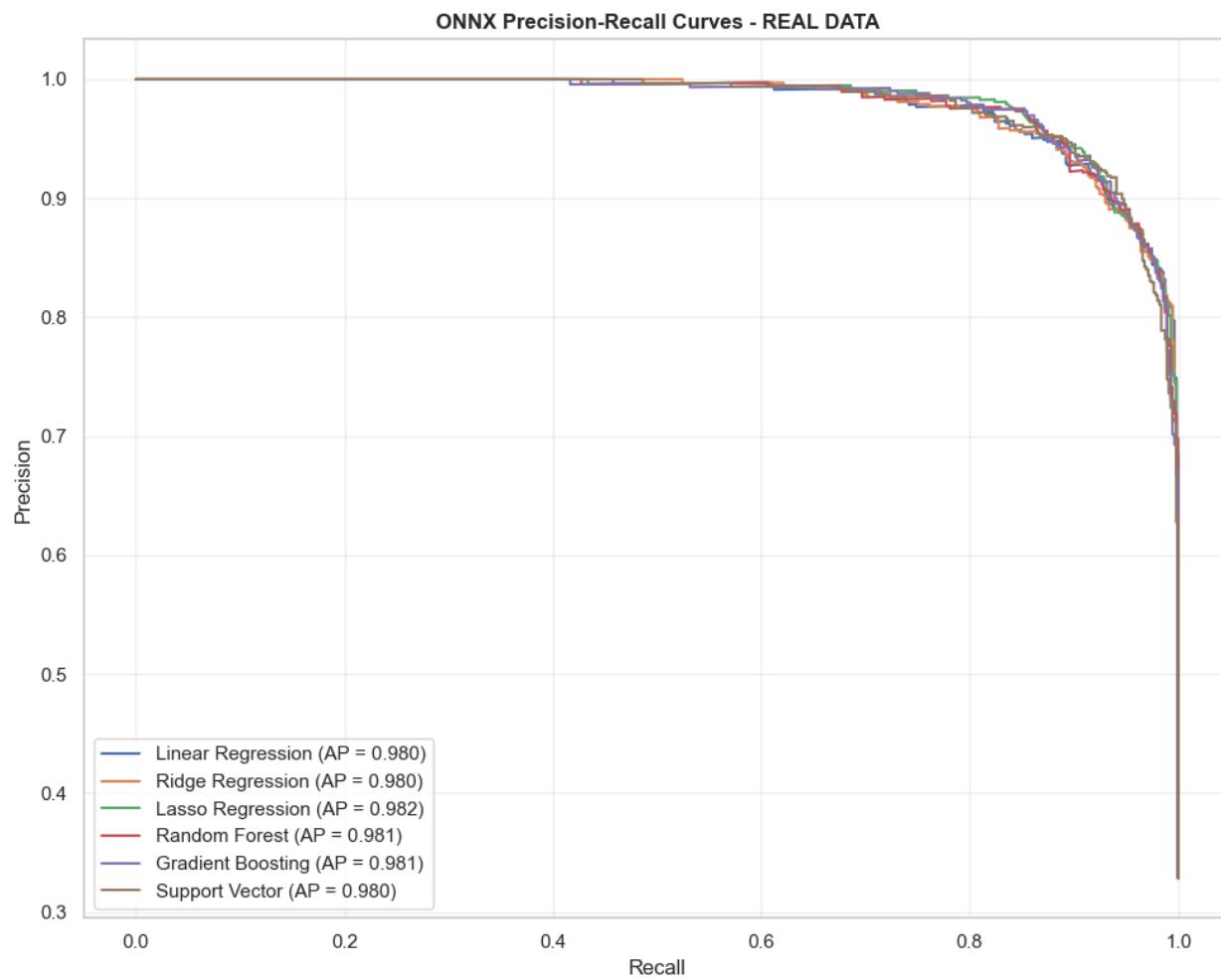
6-7. Comparison Precision-Recall Curves ONNX Predictions

```
print("SECTION 6.8: COMPARISON PRECISION-RECALL CURVES")

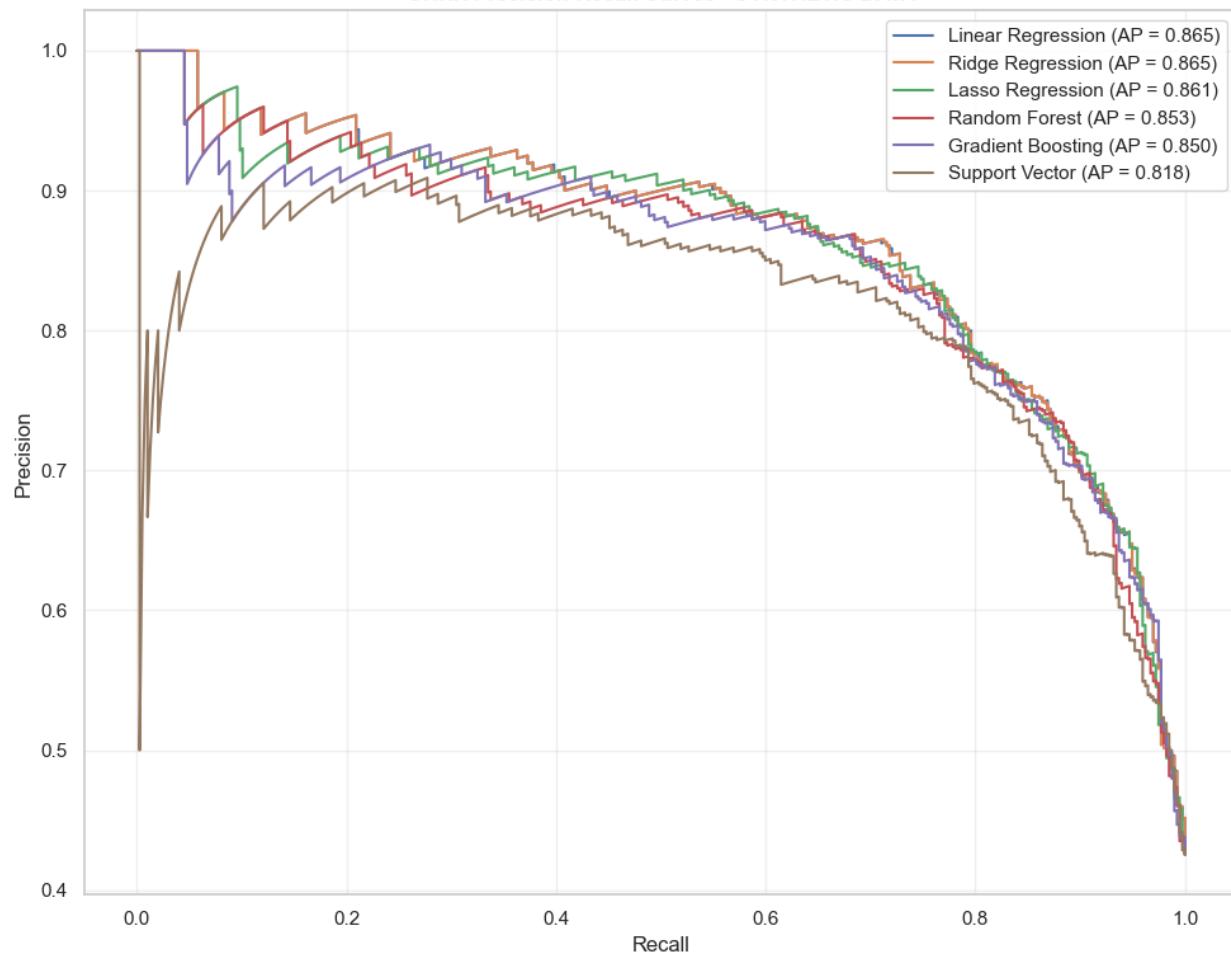
def plot_pr_comparison(results, y_test_bin, dataset_name):
    plt.figure(figsize=(10, 8))
    for r in results:
        #use onnx predictions for PR curve
        precision, recall, _ = precision_recall_curve(y_test_bin, r['y_pred_onnx'])
        ap = average_precision_score(y_test_bin, r['y_pred_onnx'])
        plt.plot(recall, precision, label=f"{r['model_name']} (AP = {ap:.3f})")

    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title(f'ONNX Precision-Recall Curves - {dataset_name.upper()} DATA', fontweight='bold')
    plt.legend(loc="best")
    plt.grid(alpha=0.3)
    plt.tight_layout()
    plt.savefig(f"visualizations/onnx_pr_comparison_{dataset_name}.png", dpi=300)
    plt.show()
    plt.close()

if results_real:
    plot_pr_comparison(results_real, y_test_bin_real, "real")
if results_synth:
    plot_pr_comparison(results_synth, y_test_bin_synth, "synthetic")
print(" ONNX Precision-Recall comparison plots saved")
```



ONNX Precision-Recall Curves - SYNTHETIC DATA



این بخش مربوط به بررسی Overfitting مدل ها است و هدف آن مقایسه عملکرد مدل روی داده های آموزش و تست است.

عملکرد کد:

- برای هر مدل، ابتدا دوباره روی داده های آموزش بازآموزی می شود و RMSE و R^2 آموزش محاسبه می شوند.
- دو نمودار کنار هم رسم می شوند:
 - نمودار اول : RMSE آموزش و تست
 - نمودار دوم : R^2 آموزش و تست
- هر مدل با دو میله کنار هم نشان داده شده تا اختلاف عملکرد آموزش و تست مشخص باشد.

هدف نمودار:

- مشاهده مدل هایی که روی داده های آموزش عملکرد بسیار خوبی دارند ولی روی داده های تست ضعیف عمل می کنند نشانه Overfitting است.
- کمک به انتخاب مدل مناسب برای تعیین بهتر روی داده های واقعی یا مصنوعی

6-8. Overfitting

```
print("SECTION 6.8: OVERTFITTING COMPARISON VISUALIZATION")

def plot_overfitting_comparison(results, dataset_name):
    if not results:
        print(f"No results for {dataset_name} - skipping overfitting plot")
        return

    models_list = [r['model_name'] for r in results]
    x = np.arange(len(models_list))
    width = 0.35

    # Lists to store values
    train_rmse_list = []
    test_rmse_list = []
    train_r2_list = []
    test_r2_list = []

    # Select correct training data
    X_train = X_train_real if dataset_name == "real" else X_train_synth
    y_train = y_train_real if dataset_name == "real" else y_train_synth

    for r in results:
        model_name = r['model_name']

        # Retrain fresh sklearn model to get training metrics
        model_template = models[model_name] # from global models dict
        model = clone(model_template)
        model.fit(X_train, y_train)
```

```
# Training predictions and metrics
y_train_pred = model.predict(X_train)
train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred))
train_r2 = r2_score(y_train, y_train_pred)

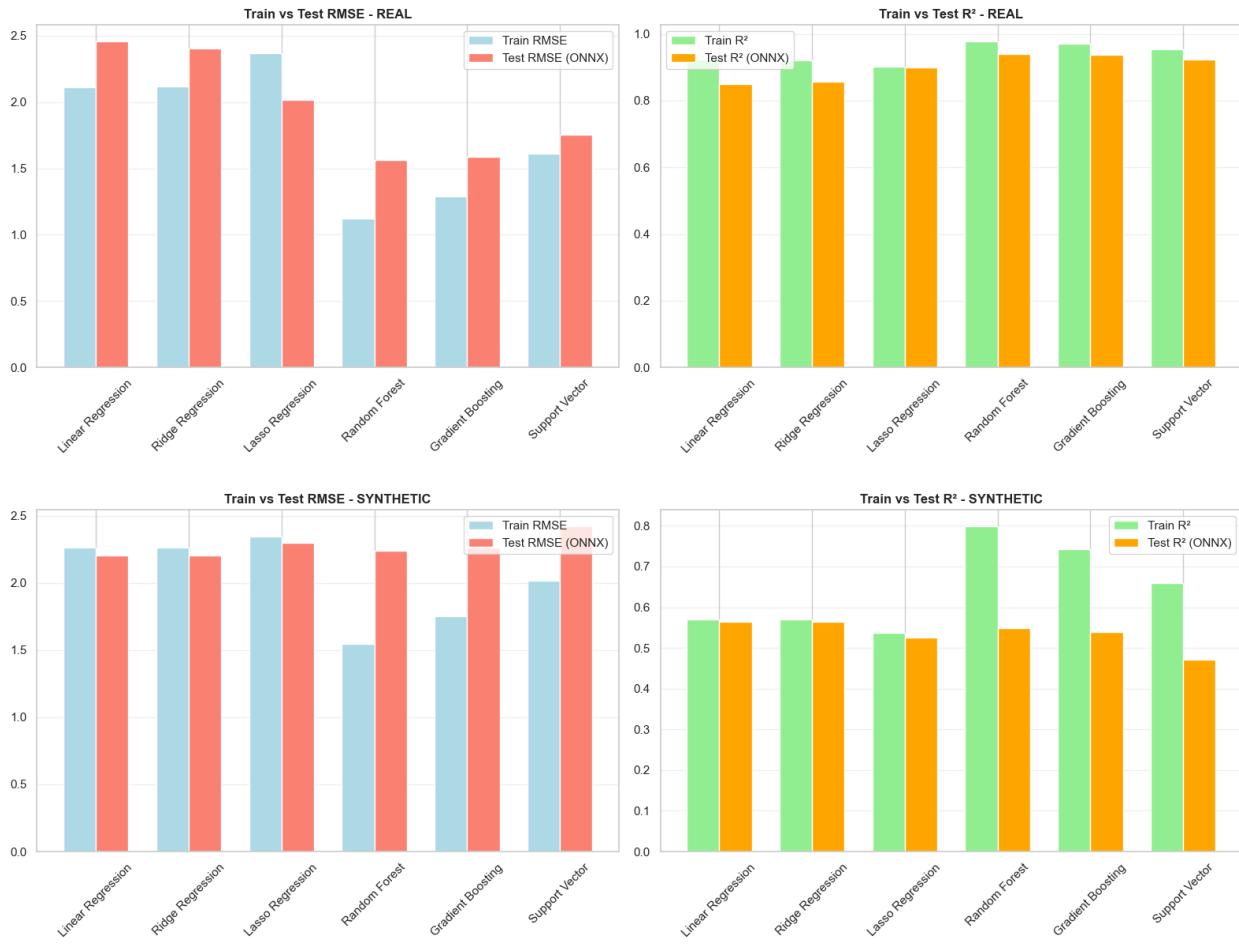
# Test metrics (from ONNX - already in results)
test_rmse = r['test_rmse']
test_r2 = r['test_r2']

train_rmse_list.append(train_rmse)
test_rmse_list.append(test_rmse)
train_r2_list.append(train_r2)
test_r2_list.append(test_r2)

# Plotting
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# RMSE Plot
axes[0].bar(x - width/2, train_rmse_list, width, label='Train RMSE', color='lightblue')
axes[0].bar(x + width/2, test_rmse_list, width, label='Test RMSE (ONNX)', color='salmon')
axes[0].set_title(f'Train vs Test RMSE - {dataset_name.upper()}', fontweight='bold')
axes[0].set_xticks(x)
axes[0].set_xticklabels(models_list, rotation=45)
axes[0].legend()
axes[0].grid(axis='y', alpha=0.3)

# R2 Plot
axes[1].bar(x - width/2, train_r2_list, width, label='Train R2', color='lightgreen')
axes[1].bar(x + width/2, test_r2_list, width, label='Test R2 (ONNX)', color='orange')
axes[1].set_title(f'Train vs Test R2 - {dataset_name.upper()}', fontweight='bold')
axes[1].set_xticks(x)
axes[1].set_xticklabels(models_list, rotation=45)
axes[1].legend()
axes[1].grid(axis='y', alpha=0.3)
```



این بخش مربوط به تحلیل توزیع نمرات Cross-Validation مدل‌ها است و هدف آن بررسی پایداری و ثبات عملکرد مدل‌ها در folds مختلف است.

برای هر مدل، داده‌ها به پنج بخش تقسیم شده و مدل پنج بار آموزش و تست می‌شود (5-fold CV). در هر بار، مقدار خطای پیش‌بینی (RMSE) و ضریب برازش (R^2) محاسبه می‌شود و پنج نمره برای هر مدل به دست می‌آید که نشان دهنده عملکرد مدل روی قسمت‌های مختلف داده است.

برای نمایش این توزیع، از Boxplot استفاده شده است:

- میانه (Median) با خط قرمز مشخص است.
- جعبه (Box) محدوده بین چارک اول و سوم (Q1–Q3) را نشان می‌دهد.
- خطوط بیرونی (Whiskers) و نقاط دورافتاده پراکندگی نمرات را نمایش می‌دهند.

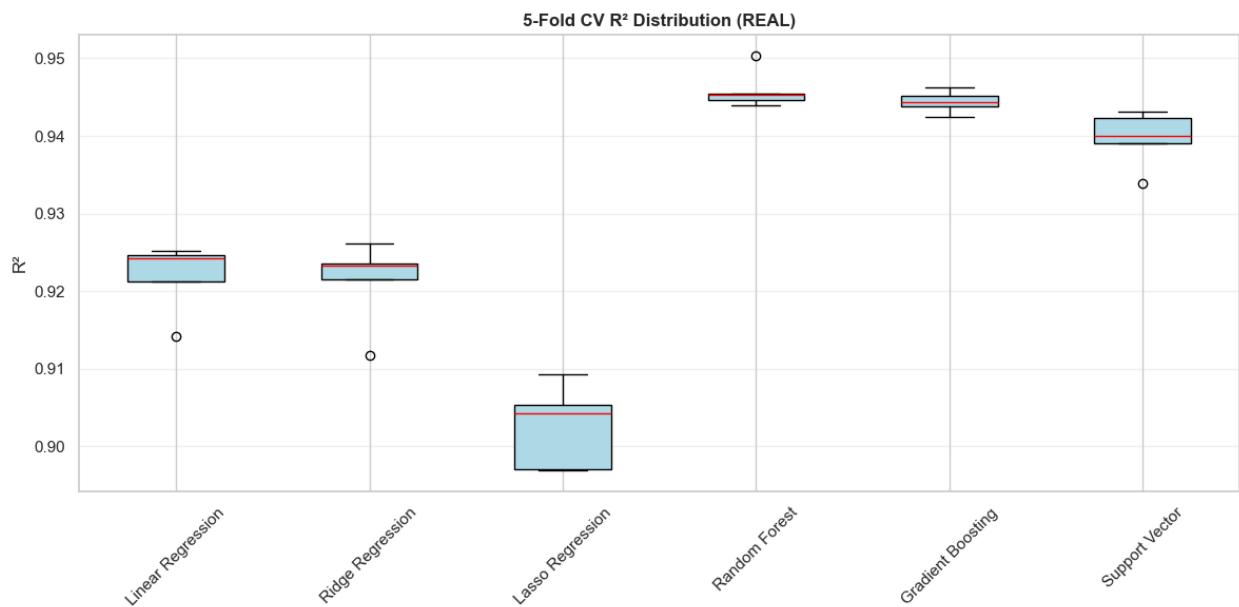
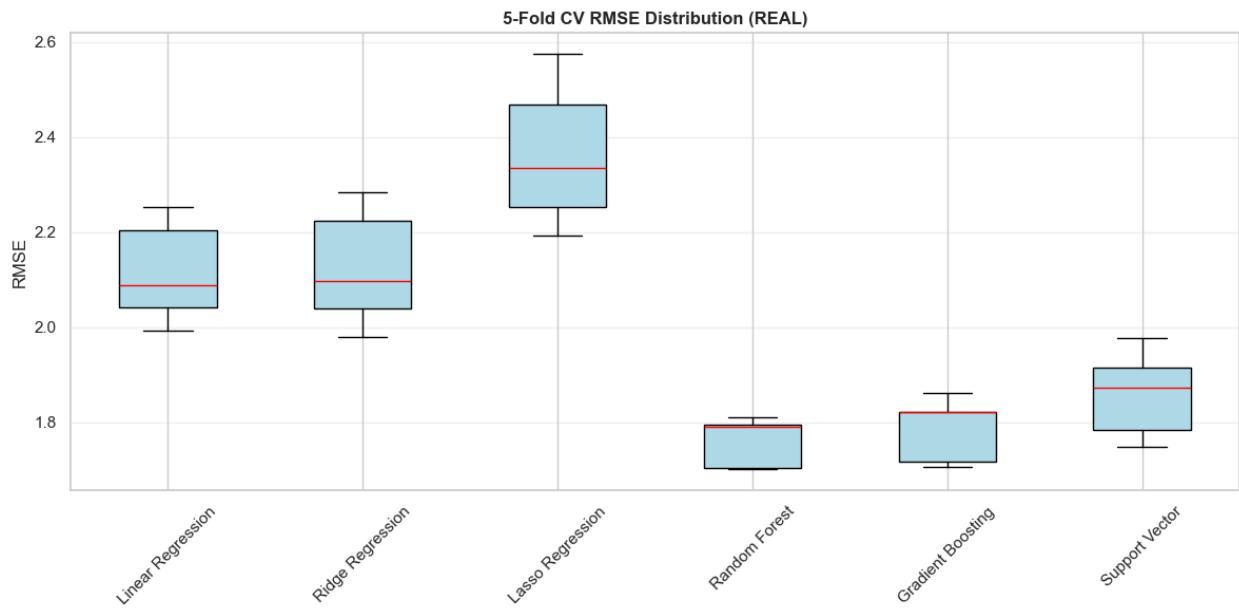
این نمودارها کمک می‌کنند تا مدل‌هایی که پایدار و کم نوسان هستند را از مدل‌هایی که نوسان بالایی دارند تشخیص دهیم. خروجی‌ها برای هر دیتاست (واقعی و مصنوعی) و هر مدل جدایگانه ذخیره و قابل مشاهده هستند و دید خوبی از ثبات عملکرد مدل‌ها می‌دهند.

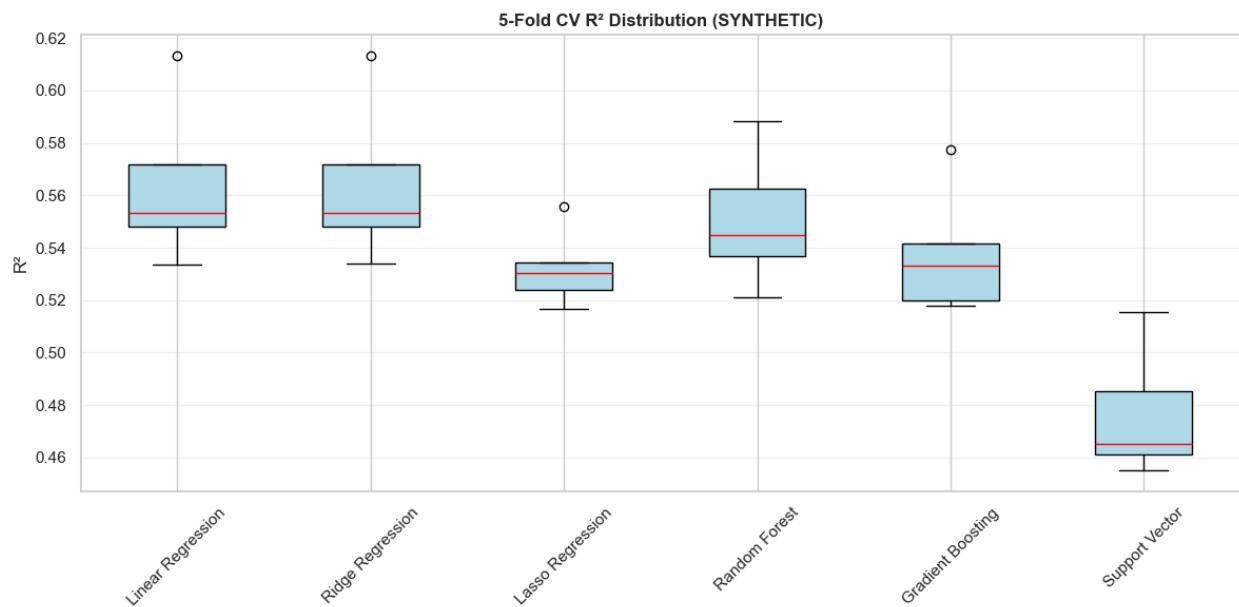
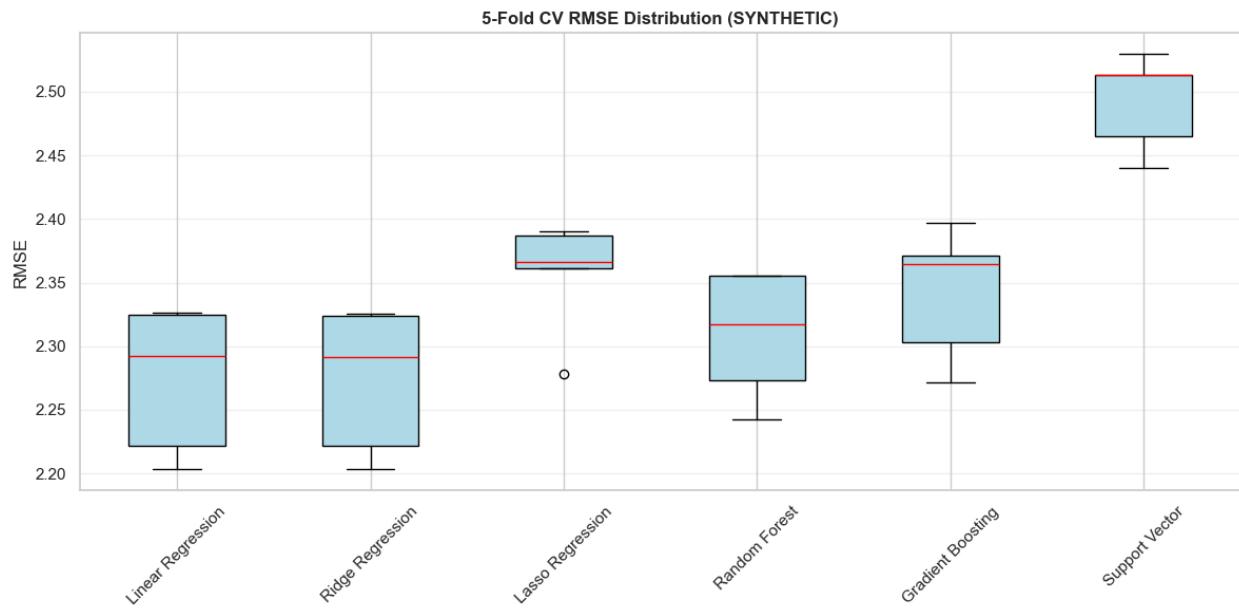
6-9. Cross-Valiation Score Distributaion

```
print("SECTION 6.9: CROSS-VALIDATION SCORE DISTRIBUTION")
def plot_cv_scores(results, dataset_name, metric='RMSE'):
    models_list = [r['model_name'] for r in results]
    if metric == 'RMSE':
        data = [r['cv_rmse_scores'] for r in results]
        title = f'5-Fold CV RMSE Distribution ({dataset_name.upper()})'
        ylabel = 'RMSE'
    else:
        data = [r['cv_r2_scores'] for r in results]
        title = f'5-Fold CV R2 Distribution ({dataset_name.upper()})'
        ylabel = 'R2'

    fig, ax = plt.subplots(figsize=(12, 6))
    ax.boxplot(data, labels=models_list, patch_artist=True,
               boxprops=dict(facecolor='lightblue'), medianprops=dict(color='red'))
    ax.set_title(title, fontweight='bold')
    ax.set_ylabel(ylabel)
    ax.tick_params(axis='x', rotation=45)
    ax.grid(axis='y', alpha=0.3)
    plt.tight_layout()
    plt.savefig(f"visualizations/cv_{metric.lower()}_{dataset_name}.png", dpi=300)
    plt.show()
    plt.close()

if results_real:
    plot_cv_scores(results_real, "real", 'RMSE')
    plot_cv_scores(results_real, "real", 'R2')
if results_synth:
    plot_cv_scores(results_synth, "synthetic", 'RMSE')
    plot_cv_scores(results_synth, "synthetic", 'R2')
print("CV distribution plots saved")
```





این بخش خلاصه نهایی عملکرد مدل ها و خروجی ONNX را ارائه می دهد.

- برای هر دیتاست (واقعی و مصنوعی)، بهترین مدل ها بر اساس R^2 بالاترین و RMSE پایین ترین شناسایی شده و نمایش داده می شوند.
- همچنین تعداد مدل هایی که با موفقیت به فرمت ONNX صادر شده اند و تعداد پیشビینی هایی که تولید شده اند گزارش می شود.

FINAL SUMMARY

```
print("FINAL SUMMARY")
def find_best_models(results, dataset_name):
    if not results:
        print(f"No successful ONNX models for {dataset_name}")
        return

    best_r2 = max(results, key=lambda x: x['test_r2'])
    best_rmse = min(results, key=lambda x: x['test_rmse'])

    print(f"\n{dataset_name.upper()} - BEST ONNX PERFORMERS:")
    print(f" Best R2: {best_r2['model_name']} (R2={best_r2['test_r2']:.4f})")
    print(f" Best RMSE: {best_rmse['model_name']} (RMSE={best_rmse['test_rmse']:.4f})")

    if results_real:
        find_best_models(results_real, "real")
    if results_synth:
        find_best_models(results_synth, "synthetic")

    # Count successful ONNX exports
    num_onnx_real = len([r for r in results_real if r['onnx_file'] is not None])
    num_onnx_synth = len([r for r in results_synth if r['onnx_file'] is not None])

    print("\n")
    print(f"\n Trained: {len(results_real) + len(results_synth)} models")
    print(f" ONNX models exported & loaded: {num_onnx_real + num_onnx_synth}")
    print(f" ONNX predictions generated: {len(results_real) + len(results_synth)} test sets")
```

FINAL SUMMARY

REAL - BEST ONNX PERFORMERS:

Best R²: Random Forest (R²=0.9397)

Best RMSE: Random Forest (RMSE=1.5615)

SYNTHETIC - BEST ONNX PERFORMERS:

Best R²: Ridge Regression (R²=0.5634)

Best RMSE: Ridge Regression (RMSE=2.2018)

Trained: 12 models

ONNX models exported & loaded: 12

ONNX predictions generated: 12 test sets

توضیحات منابع:

سورس کد این پروژه تلفیقی از منابع مختلف است. بخش هایی از آن مستقیماً بر اساس اسلایدهای درس و مفاهیم آموزش داده شده در کلاس طراحی شده اند تا اصول پیش پردازش داده و آموزش مدل ها رعایت شود. برای الگوریتم ها، ساختار مدل ها و شیوه های تحلیل، از پروژه ها و ریپازیتوری های GitHub که قبلاً روی دیتابست UCI Air Quality کار کرده اند، ایده گرفته شده است. علاوه بر این، برای رفع برخی اشکالات، بهینه سازی پیش پردازش و ارائه پیشنهادات مفهومی، از هوش مصنوعی و ابزار های یادگیری ماشین جهت تحلیل خودکار و پیشنهاد رامحل های مناسب استفاده شده است.