In [1]:
```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer  # Advanced imputation
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.svm import SVR
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import accuracy_score, classification_report, mean_squared_
```

In [3]:
```
# Step 1: Load and combine multiple CSV files from URLs
file_urls = [
    'https://raw.githubusercontent.com/hayatu4islam/Automotive_Diagnostics/main/
    'https://raw.githubusercontent.com/hayatu4islam/Automotive_Diagnostics/main/
    'https://raw.githubusercontent.com/hayatu4islam/Automotive_Diagnostics/main/
    'https://raw.githubusercontent.com/hayatu4islam/Automotive_Diagnostics/main/
    'https://raw.githubusercontent.com/hayatu4islam/Automotive_Diagnostics/main/
    'https://raw.githubusercontent.com/hayatu4islam/Automotive_Diagnostics/main/
    'https://raw.githubusercontent.com/hayatu4islam/Automotive_Diagnostics/main/
    'https://raw.githubusercontent.com/hayatu4islam/Automotive_Diagnostics/main/
    'https://raw.githubusercontent.com/hayatu4islam/Automotive_Diagnostics/main/
    'https://raw.githubusercontent.com/hayatu4islam/Automotive_Diagnostics/main/
    'https://raw.githubusercontent.com/hayatu4islam/Automotive_Diagnostics/main/
    'https://raw.githubusercontent.com/hayatu4islam/Automotive_Diagnostics/main/
]
dfs = [pd.read_csv(url) for url in file_urls]
df = pd.concat(dfs, ignore_index=True)
```

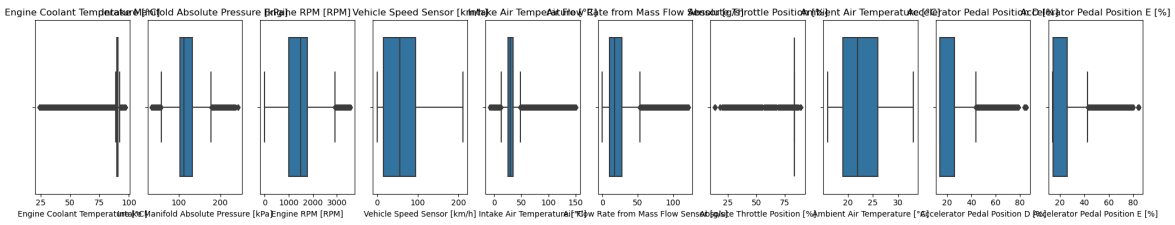
In [4]:
```
# Clean column names (remove unwanted characters and strip whitespace)
df.columns = df.columns.str.replace('Â', '', regex=False)
df.columns = df.columns.str.strip()
```

In [5]:
```
# Step 2: Initial inspection & EDA
print(df.info())
print(df.describe())
print("Missing values in each column:\n", df.isna().sum())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 368341 entries, 0 to 368340
Data columns (total 11 columns):
 #   Column                                    Non-Null Count   Dtype
---  ------                                    --------------   -----
 0   Time                                      368341 non-null  object
 1   Engine Coolant Temperature [°C]           368341 non-null  int64
 2   Intake Manifold Absolute Pressure [kPa]   368329 non-null  float64
 3   Engine RPM [RPM]                          368317 non-null  float64
 4   Vehicle Speed Sensor [km/h]               368305 non-null  float64
 5   Intake Air Temperature [°C]               368293 non-null  float64
 6   Air Flow Rate from Mass Flow Sensor [g/s] 368281 non-null  float64
 7   Absolute Throttle Position [%]            368269 non-null  float64
 8   Ambient Air Temperature [°C]              368257 non-null  float64
 9   Accelerator Pedal Position D [%]          368245 non-null  float64
 10  Accelerator Pedal Position E [%]          368233 non-null  float64
dtypes: float64(9), int64(1), object(1)
memory usage: 30.9+ MB
None
       Engine Coolant Temperature [°C]  \
count                    368341.000000
mean                         84.588770
std                          14.872193
min                          24.000000
25%                          90.000000
50%                          91.000000
75%                          91.000000
max                          97.000000

       Intake Manifold Absolute Pressure [kPa]  Engine RPM [RPM]  \
count                             368329.000000     368317.000000
mean                                 123.747424       1431.849087
std                                   31.450095        546.891641
min                                   36.000000          0.000000
25%                                  103.000000       1013.000000
50%                                  113.000000       1487.000000
75%                                  133.000000       1776.000000
max                                  242.000000       3563.000000

       Vehicle Speed Sensor [km/h]  Intake Air Temperature [°C]  \
count                368305.000000                  368293.00000
mean                     58.062880                      33.27672
std                      44.482831                      17.59453
min                       0.000000                      -8.00000
25%                      16.000000                      26.00000
50%                      55.000000                      30.00000
75%                      94.000000                      35.00000
max                     211.000000                     150.00000

       Air Flow Rate from Mass Flow Sensor [g/s]  \
count                                368281.000000
mean                                     21.384516
std                                      16.466133
min                                       0.000000
25%                                      10.190000
50%                                      17.020000
75%                                      27.270000
max                                     121.080000

       Absolute Throttle Position [%]  Ambient Air Temperature [°C]  \
```

```
count                    368269.000000                368257.000000
mean                         80.802209                    22.723587
std                          12.488784                     4.350579
min                          13.700000                    16.000000
25%                          83.500000                    19.000000
50%                          83.500000                    22.000000
75%                          83.500000                    26.000000
max                          89.000000                    33.000000

       Accelerator Pedal Position D [%]  Accelerator Pedal Position E [%]
count                     368245.000000                     368233.000000
mean                          21.330466                         21.649893
std                           12.670384                         12.681027
min                           14.100000                         14.100000
25%                           14.100000                         14.500000
50%                           14.100000                         14.500000
75%                           25.900000                         25.900000
max                           85.100000                         84.300000
Missing values in each column:
 Time                                          0
Engine Coolant Temperature [°C]                0
Intake Manifold Absolute Pressure [kPa]       12
Engine RPM [RPM]                              24
Vehicle Speed Sensor [km/h]                   36
Intake Air Temperature [°C]                   48
Air Flow Rate from Mass Flow Sensor [g/s]     60
Absolute Throttle Position [%]                72
Ambient Air Temperature [°C]                  84
Accelerator Pedal Position D [%]              96
Accelerator Pedal Position E [%]             108
dtype: int64
```

In [6]:
```
# Visualize distributions and outliers of numeric columns (limit to first 10 to
num_cols = df.select_dtypes(include=np.number).columns.tolist()
plt.figure(figsize=(20, 4))
for i, col in enumerate(num_cols[:10]):
    plt.subplot(1, 10, i + 1)
    sns.boxplot(x=df[col])
    plt.title(col)
plt.tight_layout()
plt.show()
```



In [7]:
```
# Step 3: Outlier treatment based on domain knowledge (clip coolant temp)
coolant_col = 'Engine Coolant Temperature [°C]'
if coolant_col in df.columns:
    df[coolant_col] = df[coolant_col].clip(60, 130)
```

In [8]:
```
# Step 4: Feature engineering
df['overheating'] = (df[coolant_col] > 100).astype(int)
rpm_col = 'Engine RPM [RPM]'
df['rpm_rolling_mean'] = df[rpm_col].rolling(window=5, min_periods=1).mean()
df['rpm_rolling_std'] = df[rpm_col].rolling(window=5, min_periods=1).std().filln
```

In [9]:
```
# Select features for modeling (ensure they exist)
features = [
    rpm_col,
    'Vehicle Speed Sensor [km/h]',
    coolant_col,
    'Absolute Throttle Position [%]',
    'Air Flow Rate from Mass Flow Sensor [g/s]',
    'rpm_rolling_mean',
    'rpm_rolling_std'
]
features = [f for f in features if f in df.columns]
X = df[features]
y_class = df['overheating']
```

In [10]:
```
# Step 5: Scale features (no PCA to avoid NaN issues)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [13]:
```
# Step 7: Regression modeling with RandomForestRegressor (fast, handles NaNs if
regression_target = 'Air Flow Rate from Mass Flow Sensor [g/s]'
regression_features = [
    'Vehicle Speed Sensor [km/h]',
    rpm_col,
    'Absolute Throttle Position [%]'
]
if regression_target in df.columns and all(f in df.columns for f in regression_f
    X_reg = df[regression_features]
    y_reg = df[regression_target]

    # Minimal imputation if needed (simple mean for any remaining NaNs, to keep
    from sklearn.impute import SimpleImputer
    imputer_reg = SimpleImputer(strategy='mean')
    X_reg = imputer_reg.fit_transform(X_reg)
```

In [14]:
```
    # Scale features
    scaler_reg = StandardScaler()
    X_reg_scaled = scaler_reg.fit_transform(X_reg)

    # Split dataset
    X_train_r, X_test_r, y_train_r, y_test_r = train_test_split(X_reg_scaled, y_
```

In [19]:
```

    # Minimal imputation if needed (simple mean for any remaining NaNs, to keep
    from sklearn.impute import SimpleImputer
    imputer_reg = SimpleImputer(strategy='mean')
    X_reg = imputer_reg.fit_transform(X_reg)
```

In [24]:
```
# Step 7: Regression modeling with RandomForestRegressor (fast, handles NaNs in
regression_target = 'Air Flow Rate from Mass Flow Sensor [g/s]'
regression_features = [
    'Vehicle Speed Sensor [km/h]',
    'Engine RPM [RPM]',
    'Absolute Throttle Position [%]'
]
if regression_target in df.columns and all(f in df.columns for f in regression_f
    X_reg = df[regression_features]
```

```
y_reg = df[regression_target]
```

In [27]:
```
# Minimal imputation for X_reg (handles any remaining NaNs in features)
from sklearn.impute import SimpleImputer
imputer_reg = SimpleImputer(strategy='mean')
X_reg = imputer_reg.fit_transform(X_reg)

# Handle NaNs in target y_reg: Drop rows where y_reg is NaN (avoids model er
mask = pd.notna(y_reg)  # Create a mask for non-NaN targets
X_reg = X_reg[mask]     # Filter X_reg to match
y_reg = y_reg[mask]     # Filter y_reg
print(f"After dropping NaNs: X_reg shape: {X_reg.shape}, y_reg shape: {y_reg
```

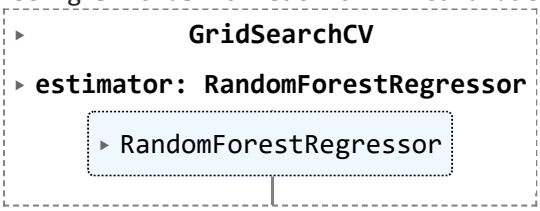
```
After dropping NaNs: X_reg shape: (368281, 3), y_reg shape: (368281,)
```

In [28]:
```
# Scale features
scaler_reg = StandardScaler()
X_reg_scaled = scaler_reg.fit_transform(X_reg)

# Split dataset
X_train_r, X_test_r, y_train_r, y_test_r = train_test_split(X_reg_scaled, y_
```

In [29]:
```
# Regression with RandomForestRegressor + hyperparameter tuning
reg = RandomForestRegressor(random_state=42)
param_grid_reg = {
    'n_estimators': [100, 200],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5]
}
grid_reg = GridSearchCV(reg, param_grid_reg, cv=3, n_jobs=-1, verbose=2)
grid_reg.fit(X_train_r, y_train_r)  # Should now work without NaN errors
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
```

Out[29]:

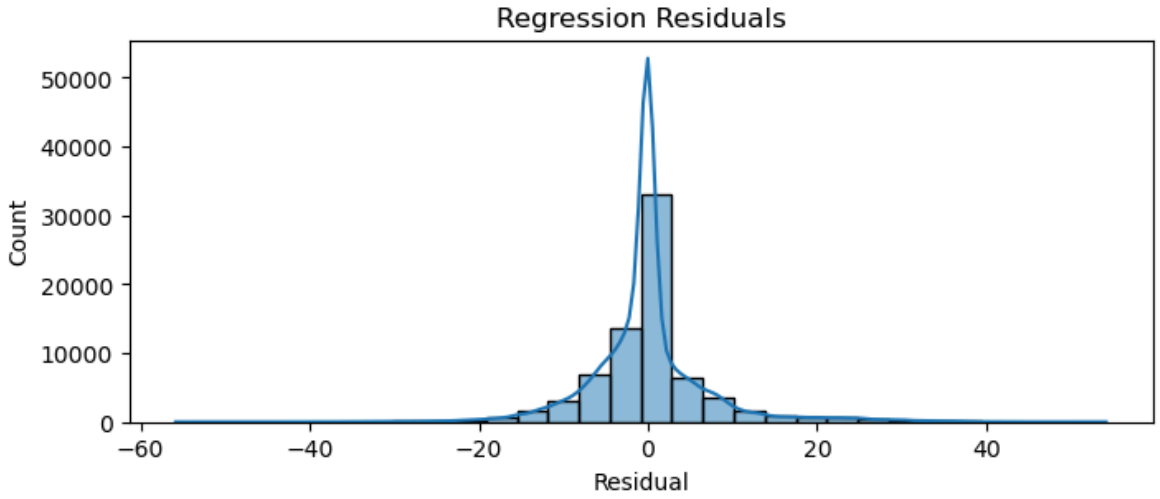


In [ ]:
```
if regression_target in df.columns and all(f in df.columns for f in regression


    y_pred_r = grid_reg.predict(X_test_r)
    rmse = mean_squared_error(y_test_r, y_pred_r, squared=False)
    print("Best regression params:", grid_reg.best_params_)
    print(f"Regression RMSE: {rmse:.4f}")

    # Residuals plot (insert the fixed code here)
    if len(y_test_r) > 0 and len(y_pred_r) > 0:
        residuals = y_test_r - y_pred_r
        if residuals.isna().sum() == 0:
            plt.figure(figsize=(8, 3))
            sns.histplot(residuals, bins=30, kde=True)
            plt.title('Regression Residuals')
            plt.xlabel('Residual')
            plt.show()
        else:
            print("Warning: Residuals contain NaNs; skipping plot.")
```

```
        else:
            print("No test data available for residuals plot.")
    else:
        print(f"Regression target '{regression_target}' or features not found. Ski
```

```
Best regression params: {'max_depth': None, 'min_samples_split': 5, 'n_estimator
s': 200}
Regression RMSE: 7.0326
```

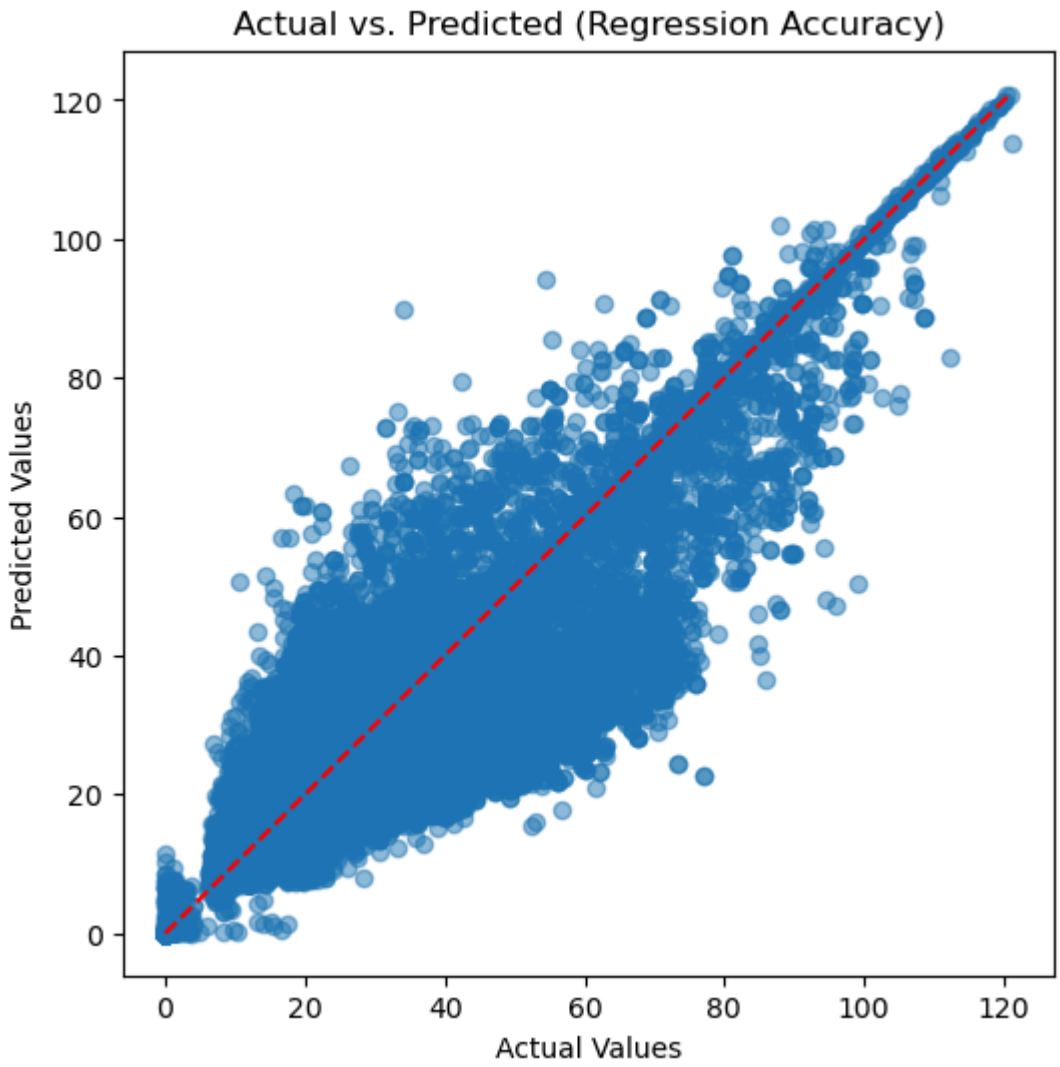
```
c:\Users\Neha\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarnin
g: use_inf_as_na option is deprecated and will be removed in a future version. Co
nvert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



In [34]:

```
from sklearn.metrics import r2_score, mean_absolute_error

# Existing RMSE
rmse = mean_squared_error(y_test_r, y_pred_r, squared=False)
print(f"Regression RMSE: {rmse:.4f}")

# Additional accuracy checks
r2 = r2_score(y_test_r, y_pred_r)
mae = mean_absolute_error(y_test_r, y_pred_r)
print(f"R² Score: {r2:.4f} (closer to 1 is better)")
print(f"Mean Absolute Error (MAE): {mae:.4f}")

# Scatter plot for visual accuracy check
plt.figure(figsize=(6, 6))
plt.scatter(y_test_r, y_pred_r, alpha=0.5)
plt.plot([y_test_r.min(), y_test_r.max()], [y_test_r.min(), y_test_r.max()], '
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs. Predicted (Regression Accuracy)')
plt.show()
```

```
Regression RMSE: 7.0326
R² Score: 0.8164 (closer to 1 is better)
Mean Absolute Error (MAE): 4.1858
```



In [35]:
```
# Regression accuracy checks
rmse = mean_squared_error(y_test_r, y_pred_r, squared=False)
r2 = r2_score(y_test_r, y_pred_r)
mae = mean_absolute_error(y_test_r, y_pred_r)
print("Best regression params:", grid_reg.best_params_)
print(f"Regression RMSE: {rmse:.4f}")
print(f"R² Score: {r2:.4f} (closer to 1 is better)")
print(f"Mean Absolute Error (MAE): {mae:.4f}")
```

```
Best regression params: {'max_depth': None, 'min_samples_split': 5, 'n_estimator
s': 200}
Regression RMSE: 7.0326
R² Score: 0.8164 (closer to 1 is better)
Mean Absolute Error (MAE): 4.1858
```

In [ ]: