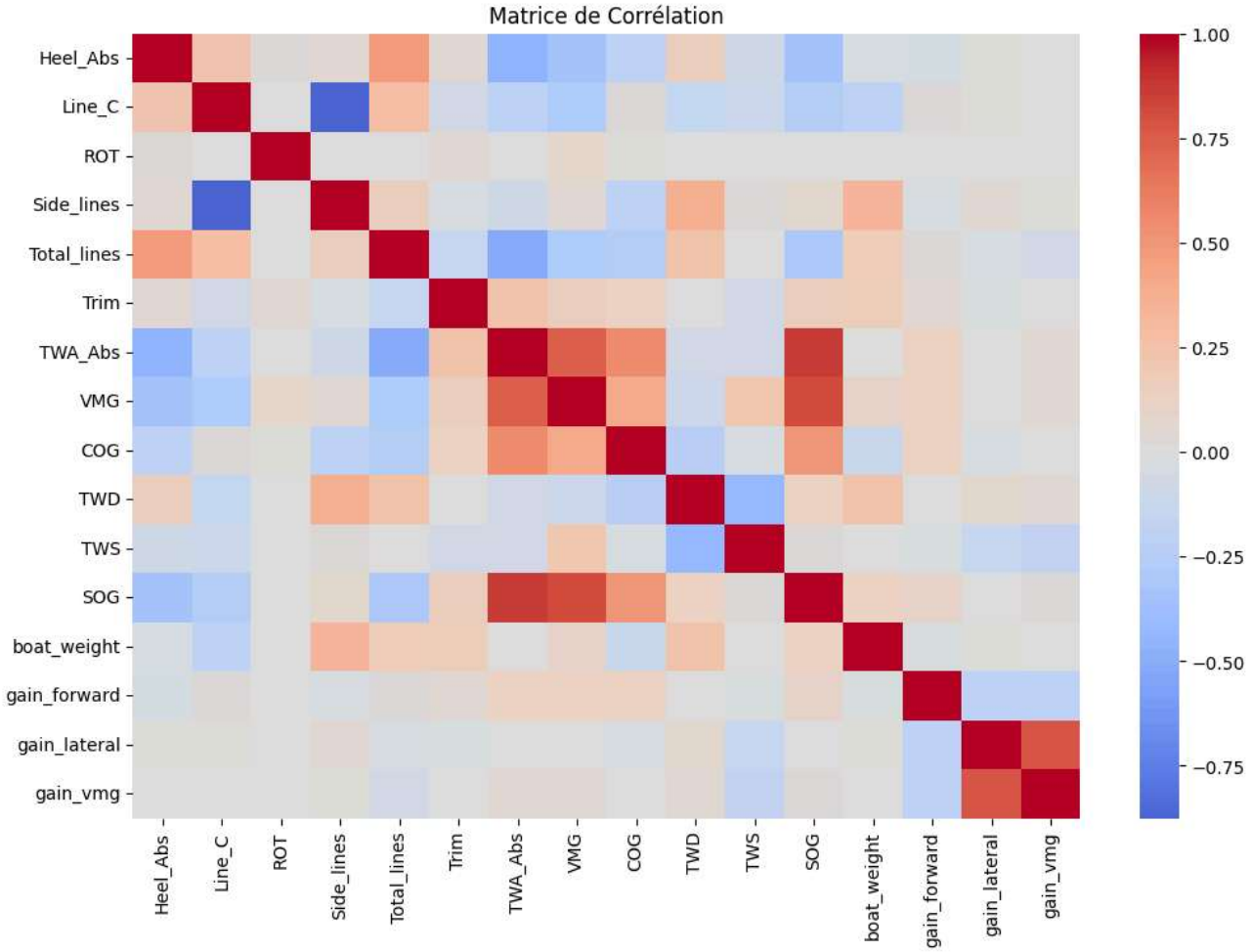
# Analyse Avancée des Données VMG (Formula Kite)

Ce notebook propose une analyse enrichie à partir du dataset `all_data.csv`, en s'inspirant de méthodes vues dans les notebooks précédents.

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from statsmodels.stats.outliers_influence import variance_inflation_factor

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

from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.metrics import r2_score
import statsmodels.formula.api as smf
import statsmodels.api as sm

import warnings
warnings.filterwarnings("ignore")

import warnings
warnings.filterwarnings("ignore")

df = pd.read_csv("all_data.csv")
```

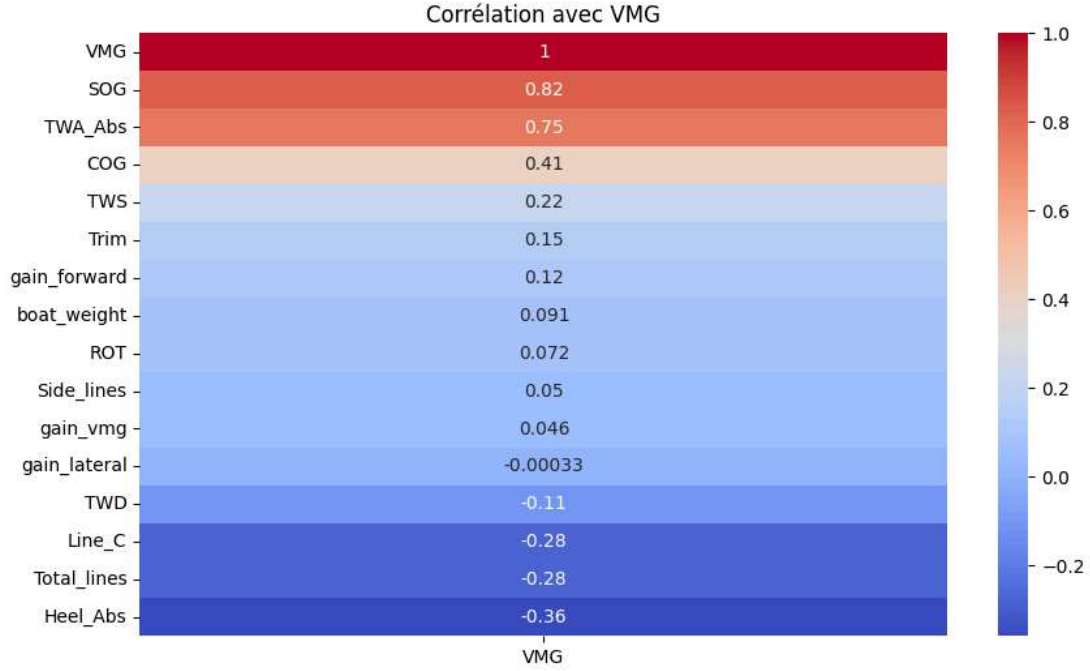
In [2]:
```
def afficher_matrice_correlation(dataframe, taille_figure=(12, 8), cmap="coolwarm"):
    plt.figure(figsize=taille_figure)
    sns.heatmap(dataframe.corr(), cmap=cmap, center=0)
    plt.title("Matrice de Corrélation")
    plt.show()

df_numeric = df.select_dtypes(include=["float64", "int64"]).copy()
drop_cols = [
    "TWA", "TimeUTC", "SecondsSince1970", "ISODateTimeUTC",
    "Lat", "LatBow", "LatCenter", "LatStern", "Lon", "LonBow", "LonCenter", "LonStern",
    "Leg", "Log", "LogAlongCourse", "MagneticVariation", "Rank", "TimeLocal",
    "DistanceToLeader", "interval_id", "boat_name", "interval_duration",
    "Heel", "Heel_Lwd", "Line_R", "Line_L", "BelowLineCalc", "VMC","XTE"
]
df_numeric.drop(columns=[c for c in drop_cols if c in df_numeric.columns], inplace=True)
df_numeric.dropna(subset=["gain_vmg"], inplace=True)
print(f"Variables utilisées:", df_numeric.columns.tolist())
afficher_matrice_correlation(df_numeric)
```

```
Variables utilisées: ['Heel_Abs', 'Line_C', 'ROT', 'Side_lines', 'Total_lines', 'Trim', 'TWA_Abs', 'VMG', 'COG', 'TWD', 'TWS', 'SOG', 'boat_weight', 'gain_forward', 'gain_lateral', 'gain_vmg']
```




In [3]:
```
def afficher_correlation_vmg(df, variable="VMG", taille_figure=(10, 6)):
    corr_with_vmg = df.corr()[variable].sort_values(ascending=False)
    print(f"Corrélation avec {variable} :")
    print(corr_with_vmg)

    plt.figure(figsize=taille_figure)
    sns.heatmap(corr_with_vmg.to_frame(), annot=True, cmap="coolwarm")
    plt.title(f"Corrélation avec {variable}")
    plt.show()

afficher_correlation_vmg(df_numeric)
```

```
Corrélation avec VMG :
VMG             1.000000
SOG             0.821369
TWA_Abs         0.753505
COG             0.407729
TWS             0.220702
Trim            0.149725
gain_forward    0.119680
boat_weight     0.091306
ROT             0.071569
Side_lines      0.049695
gain_vmg        0.046101
gain_lateral   -0.000335
TWD            -0.111722
Line_C         -0.276445
Total_lines    -0.280905
Heel_Abs       -0.359779
Name: VMG, dtype: float64
```




In [4]:
```python
def calculer_anova_quadratique(df, target="VMG"):
    cols = df.columns.drop(target)
    formula = f"{target} ~ " + " + ".join(
        list(cols) +
        [f"I({col}**2)" for col in cols]
    )
    model_anova = smf.ols(formula=formula, data=df.dropna(subset=cols)).fit()
    anova_table = sm.stats.anova_lm(model_anova, typ=2)
    return anova_table.sort_values("F", ascending=False)

resultats_anova = calculer_anova_quadratique(df_numeric)
resultats_anova
```

Out[4]:

| | sum_sq | df | F | PR(>F) |
|---|---|---|---|---|
| **I(TWA_Abs ** 2)** | 195870.824295 | 1.0 | 683611.510579 | 0.000000e+00 |
| **TWA_Abs** | 182376.832057 | 1.0 | 636515.939042 | 0.000000e+00 |
| **COG** | 1328.923320 | 1.0 | 4638.093913 | 0.000000e+00 |
| **I(COG ** 2)** | 1199.349376 | 1.0 | 4185.866076 | 0.000000e+00 |
| **I(SOG ** 2)** | 831.460278 | 1.0 | 2901.891175 | 0.000000e+00 |
| **I(gain_forward ** 2)** | 180.617498 | 1.0 | 630.375663 | 1.714631e-138 |
| **gain_lateral** | 166.092962 | 1.0 | 579.683376 | 1.462680e-127 |
| **I(gain_vmg ** 2)** | 165.326075 | 1.0 | 577.006853 | 5.527907e-127 |
| **Total_lines** | 106.056608 | 1.0 | 370.149655 | 2.840297e-82 |
| **I(gain_lateral ** 2)** | 80.614106 | 1.0 | 281.352423 | 5.066337e-63 |
| **SOG** | 43.997313 | 1.0 | 153.555640 | 3.154867e-35 |
| **TWS** | 40.001673 | 1.0 | 139.610398 | 3.476611e-32 |
| **I(TWS ** 2)** | 27.850400 | 1.0 | 97.201070 | 6.483389e-23 |
| **ROT** | 23.257515 | 1.0 | 81.171380 | 2.120343e-19 |
| **I(Total_lines ** 2)** | 22.978053 | 1.0 | 80.196025 | 3.471503e-19 |
| **gain_forward** | 21.410460 | 1.0 | 74.724946 | 5.523095e-18 |
| **Line_C** | 10.507375 | 1.0 | 36.671938 | 1.404890e-09 |
| **TWD** | 6.681889 | 1.0 | 23.320553 | 1.374171e-06 |
| **I(TWD ** 2)** | 5.593723 | 1.0 | 19.522731 | 9.955927e-06 |
| **I(Side_lines ** 2)** | 5.563257 | 1.0 | 19.416402 | 1.052571e-05 |
| **I(Heel_Abs ** 2)** | 5.116586 | 1.0 | 17.857470 | 2.383851e-05 |
| **Heel_Abs** | 4.740644 | 1.0 | 16.545388 | 4.755154e-05 |
| **gain_vmg** | 1.605032 | 1.0 | 5.601743 | 1.794533e-02 |
| **I(Line_C ** 2)** | 1.435227 | 1.0 | 5.009105 | 2.521748e-02 |
| **I(ROT ** 2)** | 0.977516 | 1.0 | 3.411641 | 6.474227e-02 |
| **Side_lines** | 0.971159 | 1.0 | 3.389455 | 6.561889e-02 |
| **boat_weight** | 0.499727 | 1.0 | 1.744105 | 1.866239e-01 |
| **I(boat_weight ** 2)** | 0.406069 | 1.0 | 1.417228 | 2.338642e-01 |
| **Trim** | 0.055179 | 1.0 | 0.192580 | 6.607792e-01 |
| **I(Trim ** 2)** | 0.009903 | 1.0 | 0.034563 | 8.525148e-01 |
| **Residual** | 19976.997754 | 69722.0 | NaN | NaN |

In [5]:
```python
def analyser_modele_polynomial(df, target="VMG", degree=2, top_coefs=30):
    """
    Analyse un modèle polynomial avec standardisation et régression linéaire
```

```
    Paramètres:
    - df: DataFrame contenant les données
    - cols: Liste des colonnes features
    - target: Variable cible (par défaut "VMG")
    - degree: Degré polynomial (par défaut 2)
    - top_coefs: Nombre de coefficients à afficher (par défaut 30)
    """
    # Préparation des données
    cols = df.columns.drop(target)
    X = df[cols].dropna()
    y = df.loc[X.index, target]

    # Création du pipeline
    poly_model = make_pipeline(
        PolynomialFeatures(degree=degree, include_bias=False),
        StandardScaler(),
        LinearRegression()
    )

    # Entraînement du modèle
    poly_model.fit(X, y)

    # Calcul et affichage du R²
    y_pred = poly_model.predict(X)
    print(f"R² (sur toutes les données) : {r2_score(y, y_pred):.3f}")

    # Extraction des coefficients
    model = poly_model.named_steps['linearregression']
    poly = poly_model.named_steps['polynomialfeatures']

    coef_df = pd.DataFrame({
        "feature": poly.get_feature_names_out(cols),
        "coefficient": model.coef_
    }).sort_values(by="coefficient", key=abs, ascending=False)

    return coef_df.head(top_coefs)

# Appel de la fonction

resultats_coefs = analyser_modele_polynomial(df_numeric)
display(resultats_coefs.head(30))
```

```
R² (sur toutes les données) : 0.986
```

| | feature | coefficient |
|---|---|---|
| **6** | TWA_Abs | -27.007227 |
| **90** | TWA_Abs^2 | 24.381093 |
| **11** | boat_weight | -9.296602 |
| **125** | boat_weight^2 | 8.517708 |
| **94** | TWA_Abs SOG | 7.352104 |
| **65** | Side_lines boat_weight | -4.532806 |
| **120** | SOG^2 | -3.381033 |
| **91** | TWA_Abs COG | -3.338670 |
| **99** | COG^2 | 2.590537 |
| **3** | Side_lines | 2.409791 |
| **1** | Line_C | -2.244464 |
| **39** | Line_C SOG | -2.195226 |
| **64** | Side_lines SOG | -2.025573 |
| **96** | TWA_Abs gain_forward | 1.917429 |
| **12** | gain_forward | -1.915238 |
| **38** | Line_C TWS | 1.793856 |
| **8** | TWD | -1.732586 |
| **102** | COG SOG | 1.688771 |
| **121** | SOG boat_weight | 1.610453 |
| **62** | Side_lines TWD | 1.570711 |
| **75** | Total_lines SOG | 1.538874 |
| **133** | gain_lateral gain_vmg | -1.478938 |
| **134** | gain_vmg^2 | 1.468066 |
| **40** | Line_C boat_weight | 1.466204 |
| **74** | Total_lines TWS | -1.389595 |
| **63** | Side_lines TWS | 1.370655 |
| **100** | COG TWD | -1.347580 |
| **124** | SOG gain_vmg | 1.184808 |
| **10** | SOG | 1.161407 |
| **9** | TWS | -1.075644 |

## Feature Importance Analysis with Random Forest

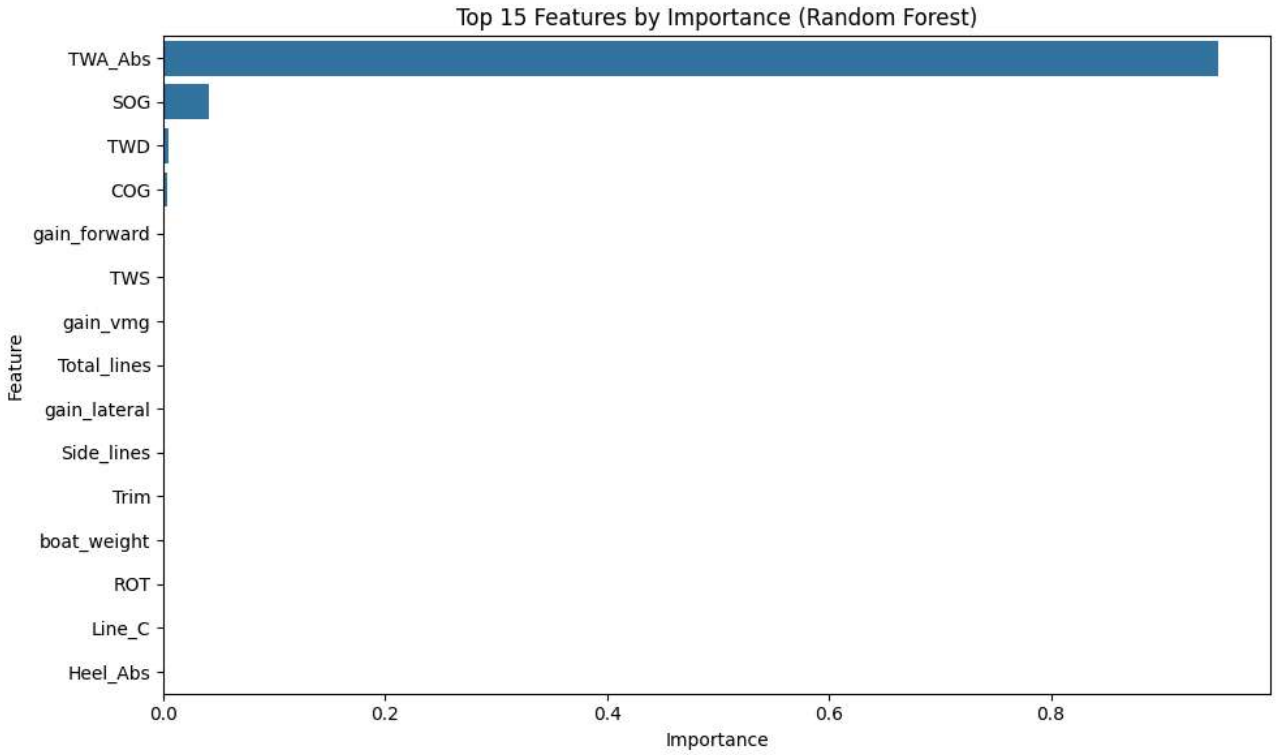
In [6]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

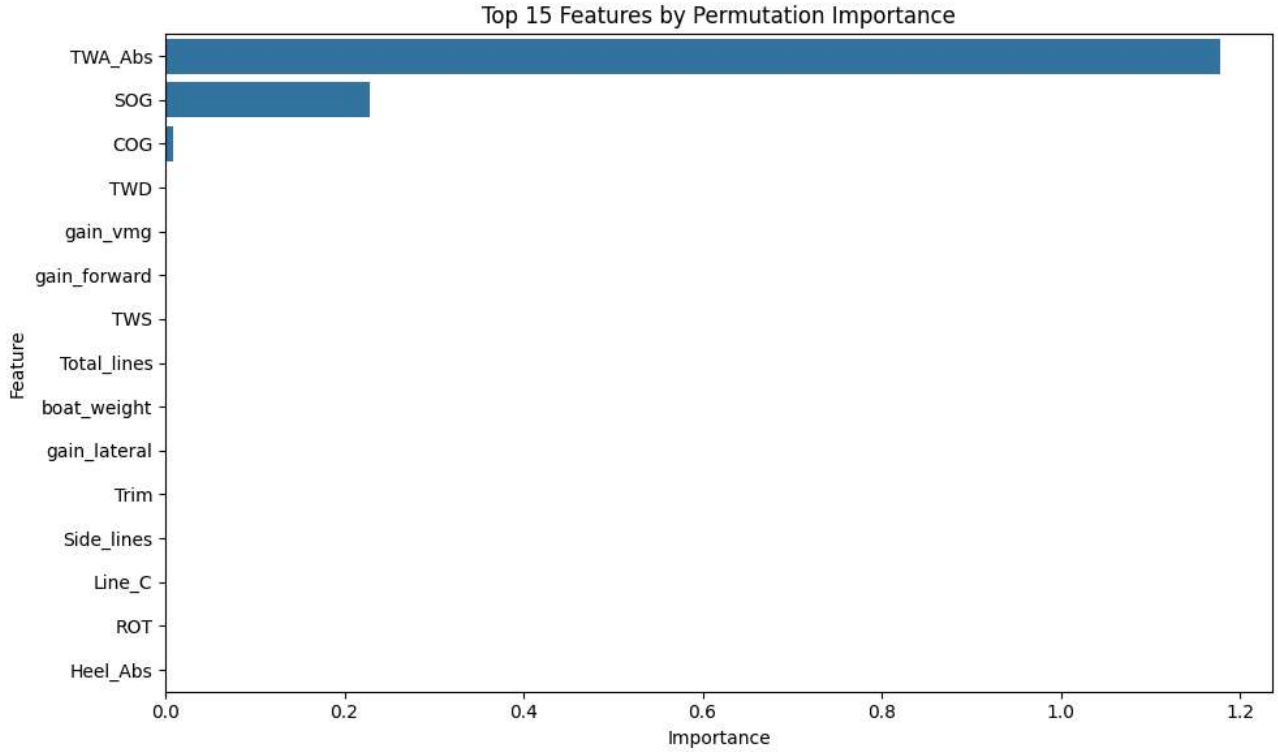
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.impute import SimpleImputer
from sklearn.pipeline import make_pipeline
from sklearn.inspection import permutation_importance
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

def analyze_feature_importance(df, target='VMG', n_estimators=100, top_features=15):
    """
    Analyze and visualize feature importance using Random Forest and Permutation Importance

    Parameters:
    - df: DataFrame containing the data
    - target: Target variable name
    - n_estimators: Number of trees in Random Forest
    - top_features: Number of top features to display
    """
    # Prepare data
    X = df.drop(columns=[target]).copy()
    y = df[target].copy()

    # Create and fit pipeline
    rf_pipeline = make_pipeline(
        SimpleImputer(strategy='median'),
        RandomForestRegressor(n_estimators=n_estimators, random_state=42)
    )
    rf_pipeline.fit(X, y)

    # Get feature importances
    rf = rf_pipeline.named_steps['randomforestregressor']
    importances = rf.feature_importances_
    feature_importance = pd.DataFrame({
        'Feature': X.columns,
        'Importance': importances
    }).sort_values('Importance', ascending=False)

    # Plot feature importance
    plt.figure(figsize=(10, 6))
    sns.barplot(x='Importance', y='Feature', data=feature_importance.head(top_features))
    plt.title(f'Top {top_features} Features by Importance (Random Forest)')
    plt.tight_layout()
    plt.show()

    # Permutation importance
    result = permutation_importance(rf_pipeline, X, y, n_repeats=10, random_state=42)
    perm_importance = pd.DataFrame({
        'Feature': X.columns,
        'Importance': result.importances_mean,
        'Std': result.importances_std
    }).sort_values('Importance', ascending=False)

    plt.figure(figsize=(10, 6))
    sns.barplot(x='Importance', y='Feature', data=perm_importance.head(top_features))
    plt.title(f'Top {top_features} Features by Permutation Importance')
    plt.tight_layout()
    plt.show()

    return feature_importance, perm_importance

def perform_clustering(df, features=['VMG', 'Side_lines', 'Line_C', 'COG', 'boat_weight'],
                       max_clusters=5, optimal_clusters=5):
    """
```

```
    Perform clustering analysis with automatic elbow method and visualization

    Parameters:
    - df: DataFrame containing the data
    - features: List of features to use for clustering
    - max_clusters: Maximum number of clusters to test for elbow method
    - optimal_clusters: Predefined number of clusters (None to use elbow method)
    """
    X_cluster = df[features].copy()

    # Create and fit pipeline
    cluster_pipeline = make_pipeline(
        SimpleImputer(strategy='median'),
        StandardScaler()
    )
    X_scaled = cluster_pipeline.fit_transform(X_cluster)

    # Determine optimal number of clusters if not provided
    if optimal_clusters is None:
        wcss = []
        for i in range(1, max_clusters+1):
            kmeans = KMeans(n_clusters=i, random_state=42)
            kmeans.fit(X_scaled)
            wcss.append(kmeans.inertia_)

        plt.figure(figsize=(10, 6))
        plt.plot(range(1, max_clusters+1), wcss, marker='o')
        plt.title('Elbow Method for Optimal Cluster Number')
        plt.xlabel('Number of clusters')
        plt.ylabel('WCSS')
        plt.show()


    # Apply KMeans clustering
    kmeans = KMeans(n_clusters=optimal_clusters, random_state=42)
    clusters = kmeans.fit_predict(X_scaled)

    # Visualize clusters with PCA
    pca = PCA(n_components=2)
    X_pca = pca.fit_transform(X_scaled)

    plt.figure(figsize=(10, 6))
    sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=clusters, palette='viridis', alpha=0.6)
    plt.title('Cluster Visualization (PCA-reduced space)')
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.show()

    # Analyze cluster characteristics
    X_cluster['Cluster'] = clusters
    cluster_stats = X_cluster.groupby('Cluster').mean()

    print("\nCluster Characteristics:")
    display(cluster_stats)

    return X_cluster, cluster_stats

def visualize_vmg_twa(df):
    """
    Visualize VMG vs True Wind Angle by Cluster

    Parameters:
    - df: DataFrame containing 'Cluster', 'TWA_Abs', and 'VMG' columns
```

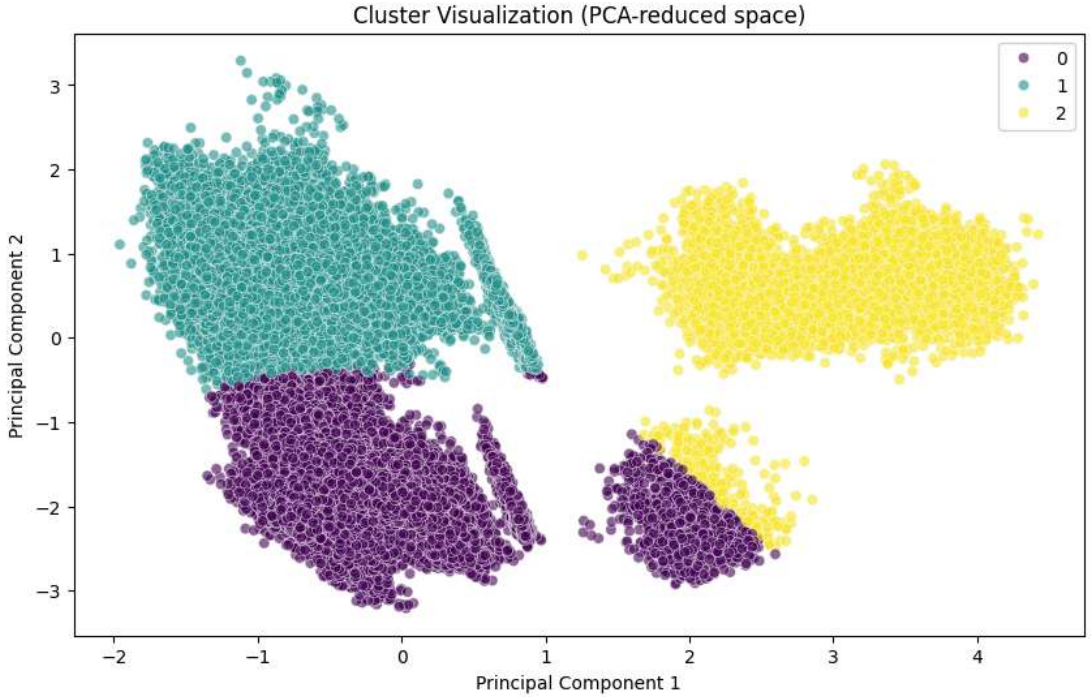
```
    """
    plt.figure(figsize=(12, 6))
    sns.scatterplot(data=df, x='boat_weight', y='VMG', hue='Cluster', palette='viridis', alpha=0.6)
    plt.title('VMG vs True Wind Angle by Cluster')
    plt.xlabel('True Wind Angle (TWA)')
    plt.ylabel('VMG')
    plt.show()
```

In [7]:
```
feature_imp, perm_imp = analyze_feature_importance(df_numeric)
```



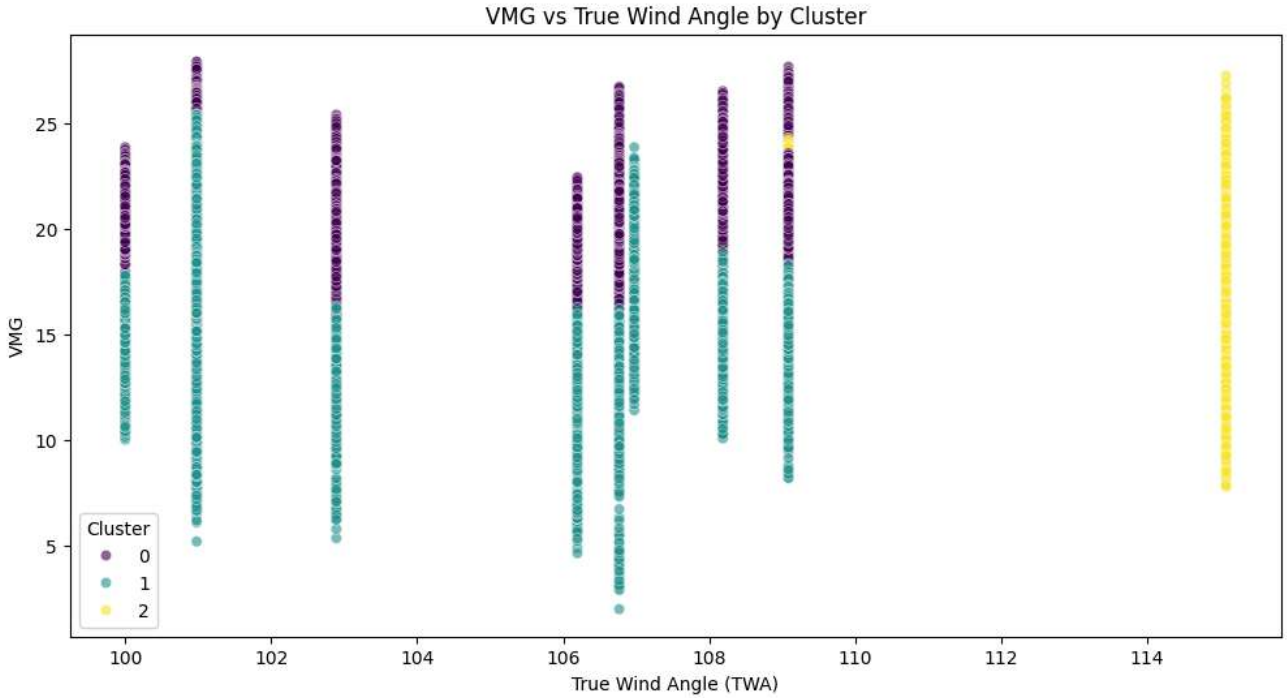



In [8]:
```
clustered_df, cluster_stats = perform_clustering(df_numeric, optimal_clusters=3)
visualize_vmg_twa(clustered_df)
```




Cluster Characteristics:

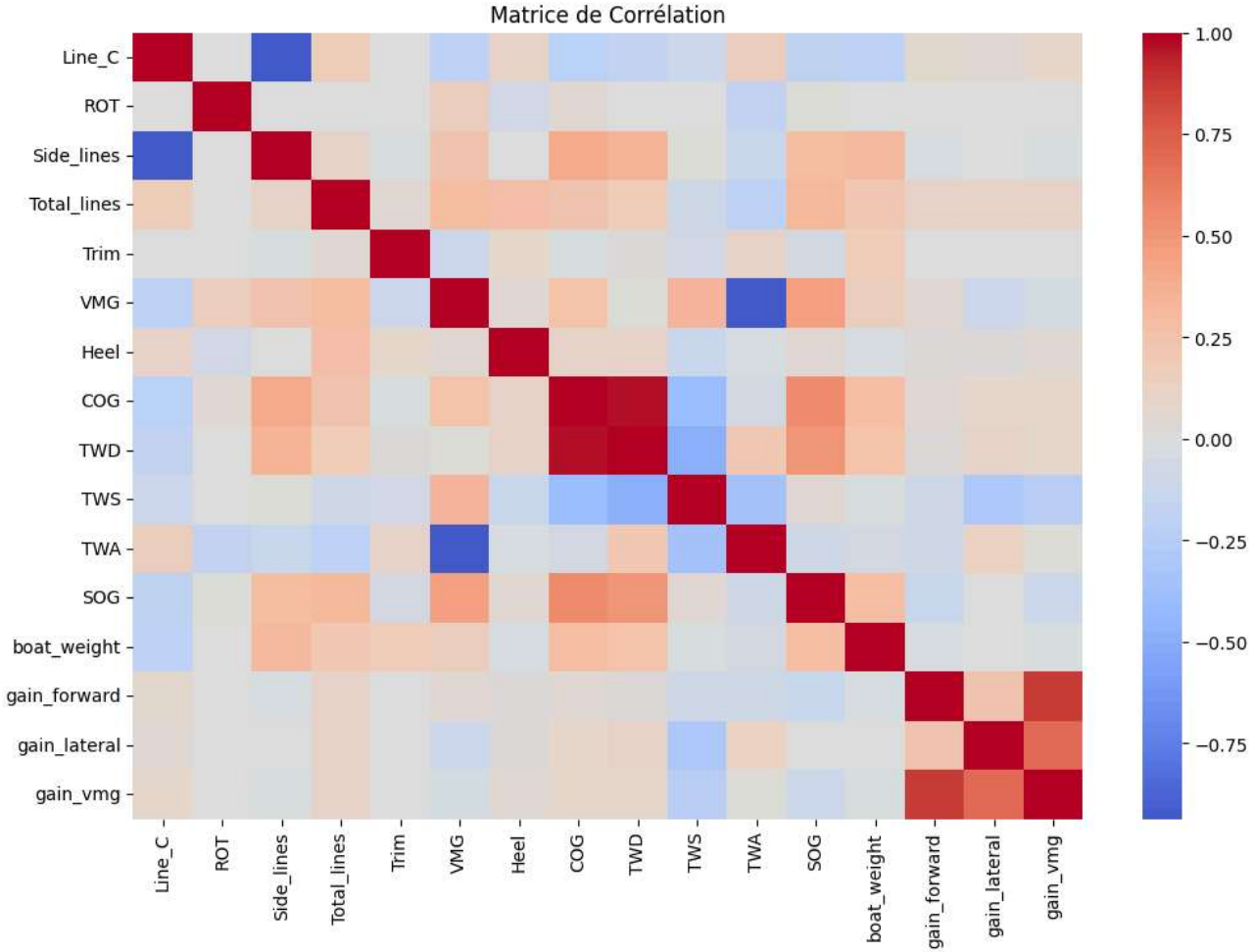
| Cluster | VMG | Side_lines | Line_C | COG | boat_weight |
|---|---|---|---|---|---|
| **0** | 20.904479 | 16.936972 | 85.209799 | 331.760911 | 104.762133 |
| **1** | 14.664049 | 13.127825 | 111.129091 | 145.727275 | 104.668802 |
| **2** | 17.148050 | 117.834902 | 5.860067 | 131.020179 | 109.590380 |




## Upwind:

In [9]:
```
# Same analysis, but for downwind and upwind separately
upwind_data = df[df['TWA'] >= 0]

# Create a numeric-only version of upwind_data (not df)
df_numeric_upwind = upwind_data.select_dtypes(include=["float64", "int64"]).copy()

drop_cols = [
    "TWA_Abs", "TimeUTC", "SecondsSince1970", "ISODateTimeUTC",
    "Lat", "LatBow", "LatCenter", "LatStern", "Lon", "LonBow", "LonCenter", "LonStern",
    "Leg", "Log", "LogAlongCourse", "MagneticVariation", "Rank", "TimeLocal",
    "DistanceToLeader", "interval_id", "boat_name", "interval_duration",
    "Heel_Abs", "Heel_Lwd", "Line_R", "Line_L", "BelowLineCalc", "VMC","XTE"
]
df_numeric_upwind.drop(columns=[c for c in drop_cols if c in df_numeric_upwind.columns], inplace=True)
df_numeric_upwind.dropna(subset=["gain_vmg"], inplace=True)
print(f"Variables utilisées:", df_numeric_upwind.columns.tolist())

afficher_matrice_correlation(df_numeric_upwind)
```

```
Variables utilisées: ['Line_C', 'ROT', 'Side_lines', 'Total_lines', 'Trim', 'VMG', 'Heel', 'COG', 'TWD', 'TWS', 'TWA', 'SOG', 'boat_weight', 'gain_forward', 'gain_lateral', 'gain_vmg']
```




In [10]:
```
afficher_correlation_vmg(df_numeric_upwind)
```

```
Corrélation avec VMG :
VMG             1.000000
SOG             0.442597
TWS             0.331718
Total_lines     0.294161
COG             0.254933
Side_lines      0.228875
boat_weight     0.152040
ROT             0.147572
Heel            0.055646
gain_forward    0.049845
TWD             0.015006
gain_vmg       -0.047593
Trim           -0.114488
gain_lateral   -0.114560
Line_C         -0.198375
TWA            -0.929339
Name: VMG, dtype: float64
```


In [11]:
```
resultats_anova_upwind = calculer_anova_quadratique(df_numeric)
resultats_anova_upwind
```

Out[11]:

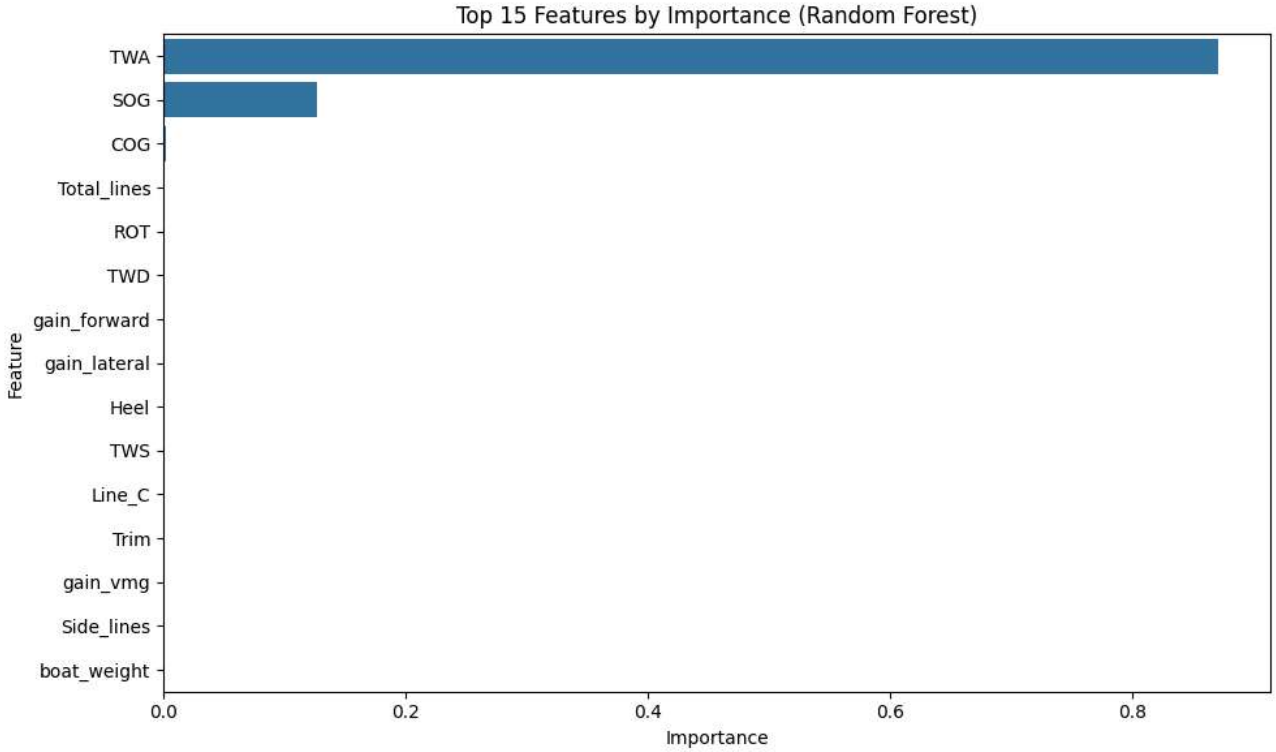
| | sum_sq | df | F | PR(>F) |
|---|---|---|---|---|
| **I(TWA_Abs ** 2)** | 195870.824295 | 1.0 | 683611.510579 | 0.000000e+00 |
| **TWA_Abs** | 182376.832057 | 1.0 | 636515.939042 | 0.000000e+00 |
| **COG** | 1328.923320 | 1.0 | 4638.093913 | 0.000000e+00 |
| **I(COG ** 2)** | 1199.349376 | 1.0 | 4185.866076 | 0.000000e+00 |
| **I(SOG ** 2)** | 831.460278 | 1.0 | 2901.891175 | 0.000000e+00 |
| **I(gain_forward ** 2)** | 180.617498 | 1.0 | 630.375663 | 1.714631e-138 |
| **gain_lateral** | 166.092962 | 1.0 | 579.683376 | 1.462680e-127 |
| **I(gain_vmg ** 2)** | 165.326075 | 1.0 | 577.006853 | 5.527907e-127 |
| **Total_lines** | 106.056608 | 1.0 | 370.149655 | 2.840297e-82 |
| **I(gain_lateral ** 2)** | 80.614106 | 1.0 | 281.352423 | 5.066337e-63 |
| **SOG** | 43.997313 | 1.0 | 153.555640 | 3.154867e-35 |
| **TWS** | 40.001673 | 1.0 | 139.610398 | 3.476611e-32 |
| **I(TWS ** 2)** | 27.850400 | 1.0 | 97.201070 | 6.483389e-23 |
| **ROT** | 23.257515 | 1.0 | 81.171380 | 2.120343e-19 |
| **I(Total_lines ** 2)** | 22.978053 | 1.0 | 80.196025 | 3.471503e-19 |
| **gain_forward** | 21.410460 | 1.0 | 74.724946 | 5.523095e-18 |
| **Line_C** | 10.507375 | 1.0 | 36.671938 | 1.404890e-09 |
| **TWD** | 6.681889 | 1.0 | 23.320553 | 1.374171e-06 |
| **I(TWD ** 2)** | 5.593723 | 1.0 | 19.522731 | 9.955927e-06 |
| **I(Side_lines ** 2)** | 5.563257 | 1.0 | 19.416402 | 1.052571e-05 |
| **I(Heel_Abs ** 2)** | 5.116586 | 1.0 | 17.857470 | 2.383851e-05 |
| **Heel_Abs** | 4.740644 | 1.0 | 16.545388 | 4.755154e-05 |
| **gain_vmg** | 1.605032 | 1.0 | 5.601743 | 1.794533e-02 |
| **I(Line_C ** 2)** | 1.435227 | 1.0 | 5.009105 | 2.521748e-02 |
| **I(ROT ** 2)** | 0.977516 | 1.0 | 3.411641 | 6.474227e-02 |
| **Side_lines** | 0.971159 | 1.0 | 3.389455 | 6.561889e-02 |
| **boat_weight** | 0.499727 | 1.0 | 1.744105 | 1.866239e-01 |
| **I(boat_weight ** 2)** | 0.406069 | 1.0 | 1.417228 | 2.338642e-01 |
| **Trim** | 0.055179 | 1.0 | 0.192580 | 6.607792e-01 |
| **I(Trim ** 2)** | 0.009903 | 1.0 | 0.034563 | 8.525148e-01 |
| **Residual** | 19976.997754 | 69722.0 | NaN | NaN |

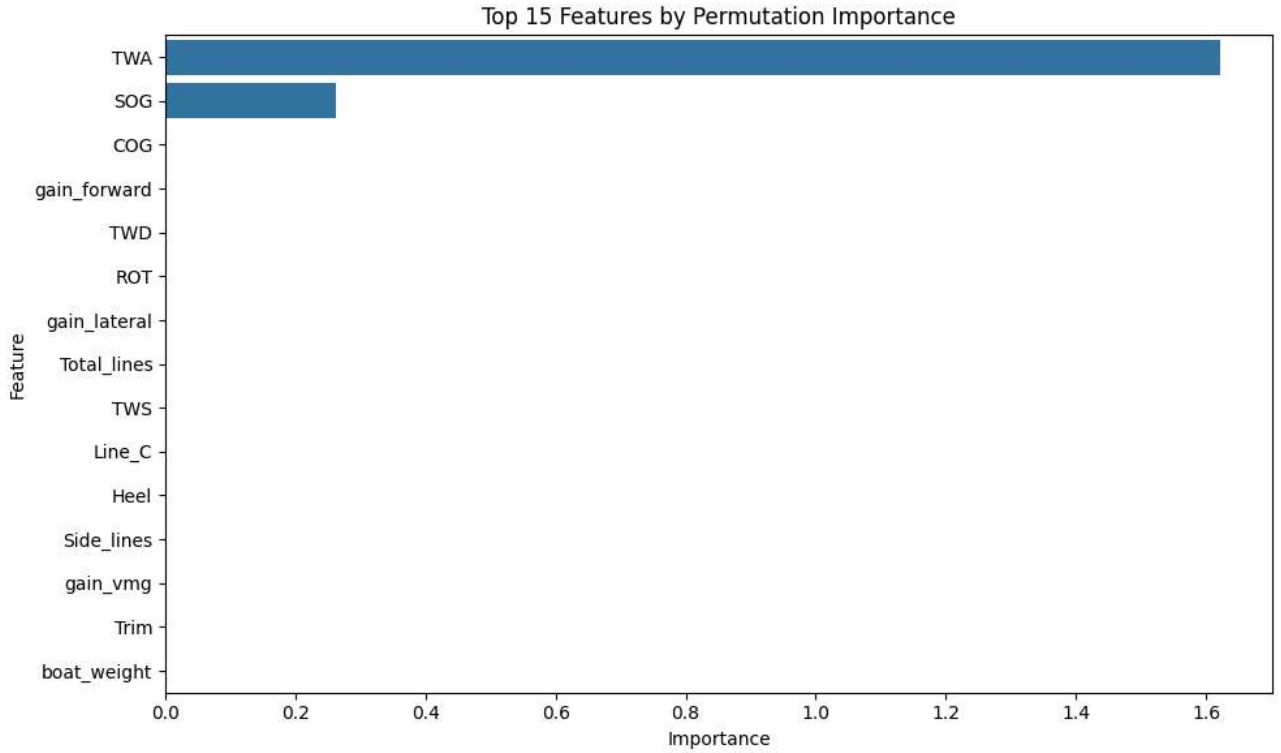
In [12]:
```
resultats_coefs_upwind = analyser_modele_polynomial(df_numeric_upwind)
display(resultats_coefs_upwind.head(30))
```

```
R² (sur toutes les données) : 1.000
```

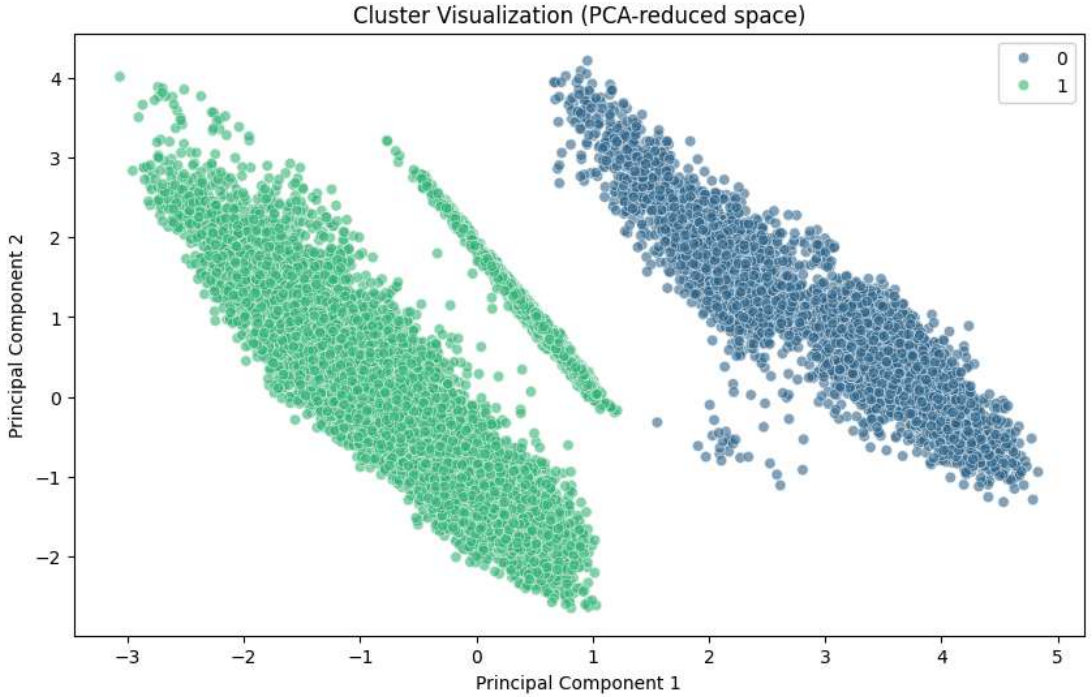
| | feature | coefficient |
|---|---|---|
| **91** | COG TWD | 7.832949e+06 |
| **99** | TWD^2 | -4.486192e+06 |
| **90** | COG^2 | -3.400250e+06 |
| **101** | TWD TWA | 1.585162e+06 |
| **93** | COG TWA | -1.258796e+06 |
| **114** | TWA^2 | -2.584012e+05 |
| **106** | TWD gain_vmg | 9.320012e+02 |
| **104** | TWD gain_forward | -7.211720e+02 |
| **98** | COG gain_vmg | -6.943658e+02 |
| **49** | Side_lines TWD | -6.125845e+02 |
| **102** | TWD SOG | -5.310672e+02 |
| **96** | COG gain_forward | 5.292414e+02 |
| **94** | COG SOG | 4.977636e+02 |
| **48** | Side_lines COG | 4.849549e+02 |
| **22** | Line_C TWD | -4.636752e+02 |
| **105** | TWD gain_lateral | -4.097133e+02 |
| **21** | Line_C COG | 3.587234e+02 |
| **61** | Total_lines TWD | 3.558074e+02 |
| **103** | TWD boat_weight | 3.117143e+02 |
| **97** | COG gain_lateral | 3.102983e+02 |
| **60** | Total_lines COG | -3.034358e+02 |
| **95** | COG boat_weight | -3.002878e+02 |
| **119** | TWA gain_vmg | -2.419997e+02 |
| **7** | TWD | -2.075688e+02 |
| **6** | COG | 2.064458e+02 |
| **117** | TWA gain_forward | 1.951706e+02 |
| **100** | TWD TWS | 1.377211e+02 |
| **51** | Side_lines TWA | 1.299360e+02 |
| **92** | COG TWS | -1.238393e+02 |
| **24** | Line_C TWA | 1.144389e+02 |

In [13]:
```
feature_imp_upwind, perm_imp_upwind = analyze_feature_importance(df_numeric_upwind)
```


Top 15 Features by Importance (Random Forest)
TWA
SOG
COG
Total_lines
ROT
TWD
gain_forward
gain_lateral
Heel
TWS
Line_C
Trim
gain_vmg
Side_lines
boat_weight
Feature
0.0
0.2
0.4
0.6
0.8
Importance

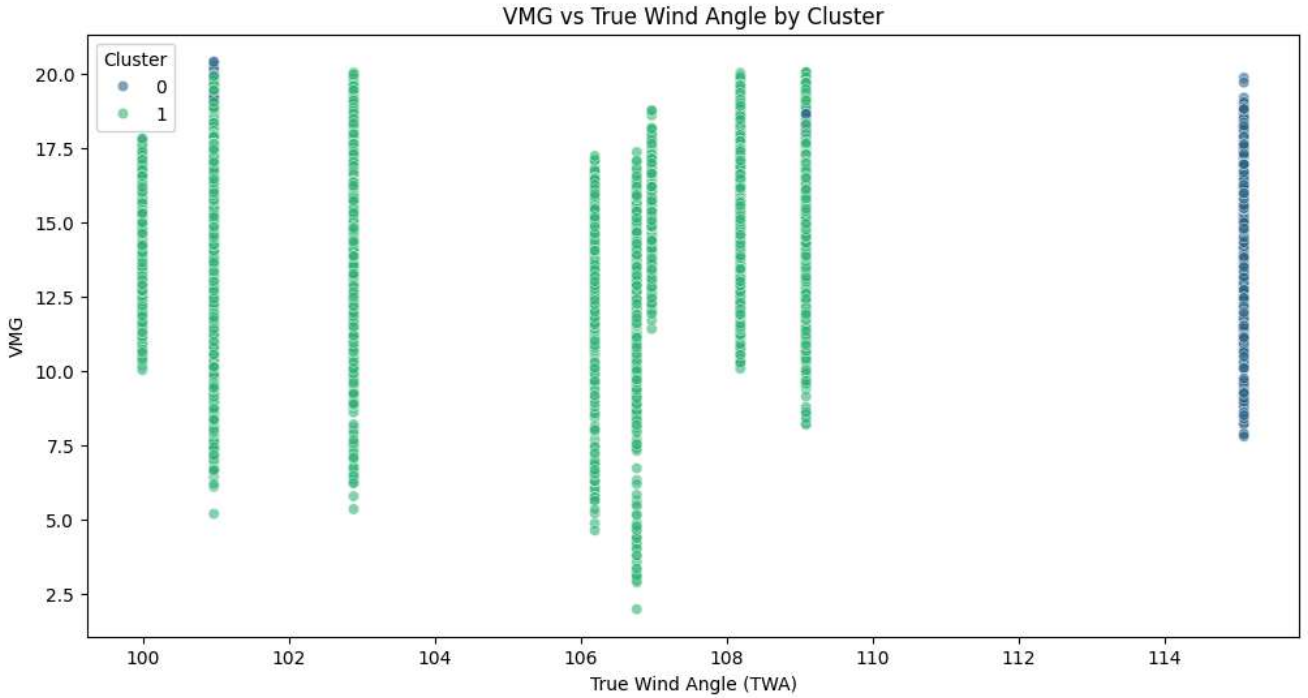



In [14]:
```
clustered_df_upwind, cluster_stats_upwind = perform_clustering(df_numeric_upwind, optimal_clusters=2)
visualize_vmg_twa(clustered_df_upwind)
```




Cluster Characteristics:

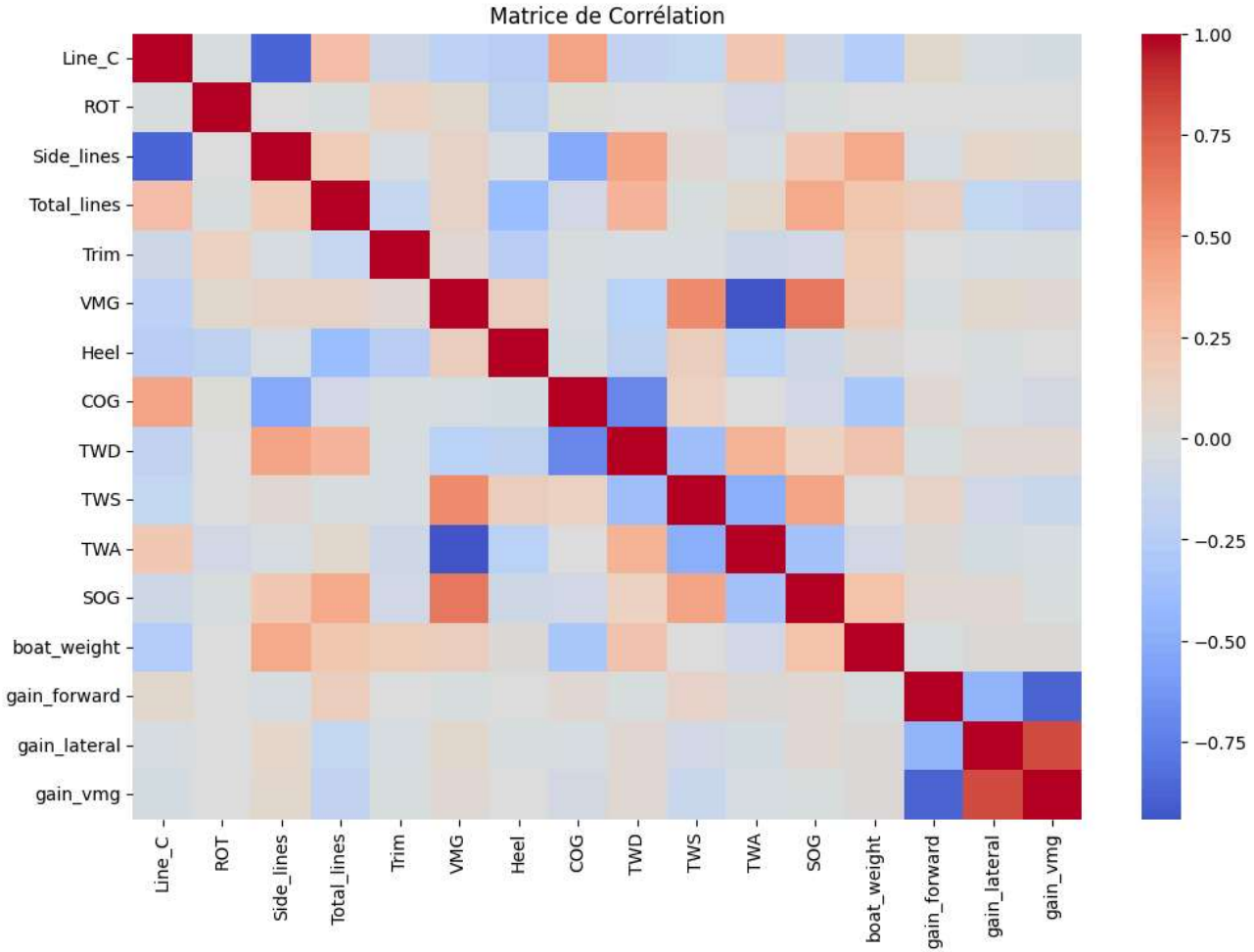
| Cluster | VMG | Side_lines | Line_C | COG | boat_weight |
|---|---|---|---|---|---|
| **0** | 15.427403 | 124.768379 | 6.004333 | 173.622030 | 108.488561 |
| **1** | 14.317719 | 13.234537 | 113.206068 | 149.023227 | 104.836522 |




## Downwind

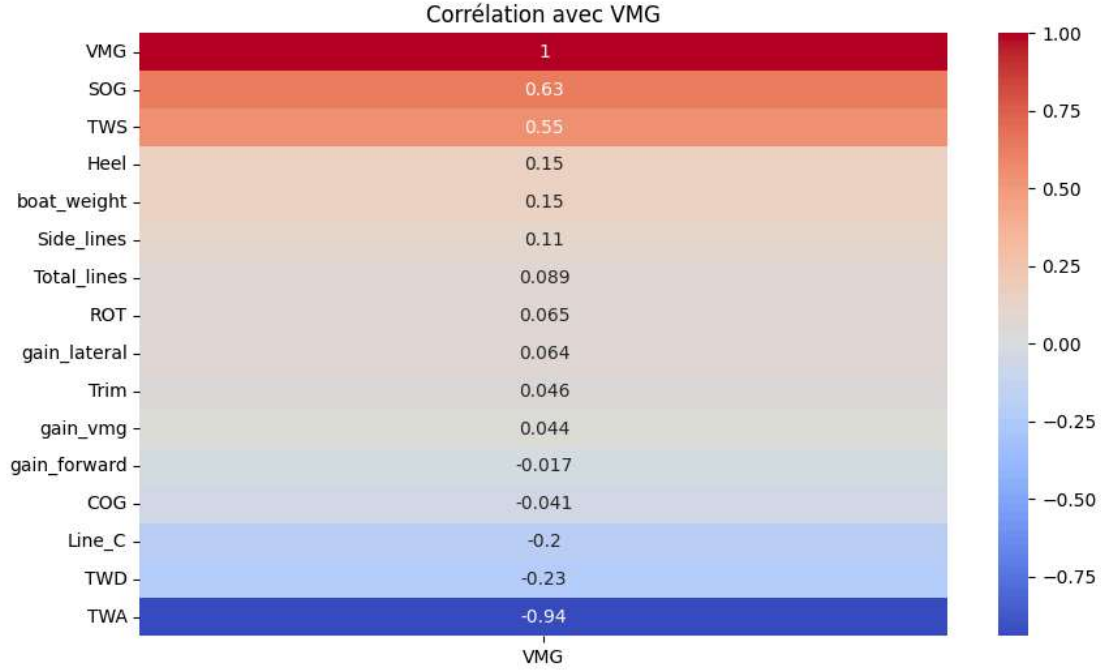
In [15]:
```
# Same analysis, but for downwind and upwind separately
downwind_data = df[df['TWA'] <= 0]

# Create a numeric-only version of upwind_data (not df)
df_numeric_downwind = downwind_data.select_dtypes(include=["float64", "int64"]).copy()

drop_cols = [
    "TWA_Abs", "TimeUTC", "SecondsSince1970", "ISODateTimeUTC",
    "Lat", "LatBow", "LatCenter", "LatStern", "Lon", "LonBow", "LonCenter", "LonStern",
    "Leg", "Log", "LogAlongCourse", "MagneticVariation", "Rank", "TimeLocal",
    "DistanceToLeader", "interval_id", "boat_name", "interval_duration",
    "Heel_Abs", "Heel_Lwd", "Line_R", "Line_L", "BelowLineCalc", "VMC","XTE"
]
df_numeric_downwind.drop(columns=[c for c in drop_cols if c in df_numeric_downwind.columns], inplace=True)
df_numeric_downwind.dropna(subset=["gain_vmg"], inplace=True)
print(f"Variables utilisées:", df_numeric_downwind.columns.tolist())

afficher_matrice_correlation(df_numeric_downwind)
```

```
Variables utilisées: ['Line_C', 'ROT', 'Side_lines', 'Total_lines', 'Trim', 'VMG', 'Heel', 'COG', 'TWD', 'TWS', 'TWA', 'SOG', 'boat_weight', 'gain_forward', 'gain_lateral', 'gain_vmg']
```




In [16]:
```
afficher_correlation_vmg(df_numeric_downwind)
```

```
Corrélation avec VMG :
VMG             1.000000
SOG             0.634229
TWS             0.549965
Heel            0.154422
boat_weight     0.146242
Side_lines      0.106729
Total_lines     0.089235
ROT             0.064886
gain_lateral    0.064343
Trim            0.046333
gain_vmg        0.043519
gain_forward   -0.016645
COG            -0.040604
Line_C         -0.200301
TWD            -0.226256
TWA            -0.940760
Name: VMG, dtype: float64
```




In [17]:
```
resultats_anova_downwind = calculer_anova_quadratique(df_numeric_downwind)
resultats_anova_downwind
```

Out[17]:

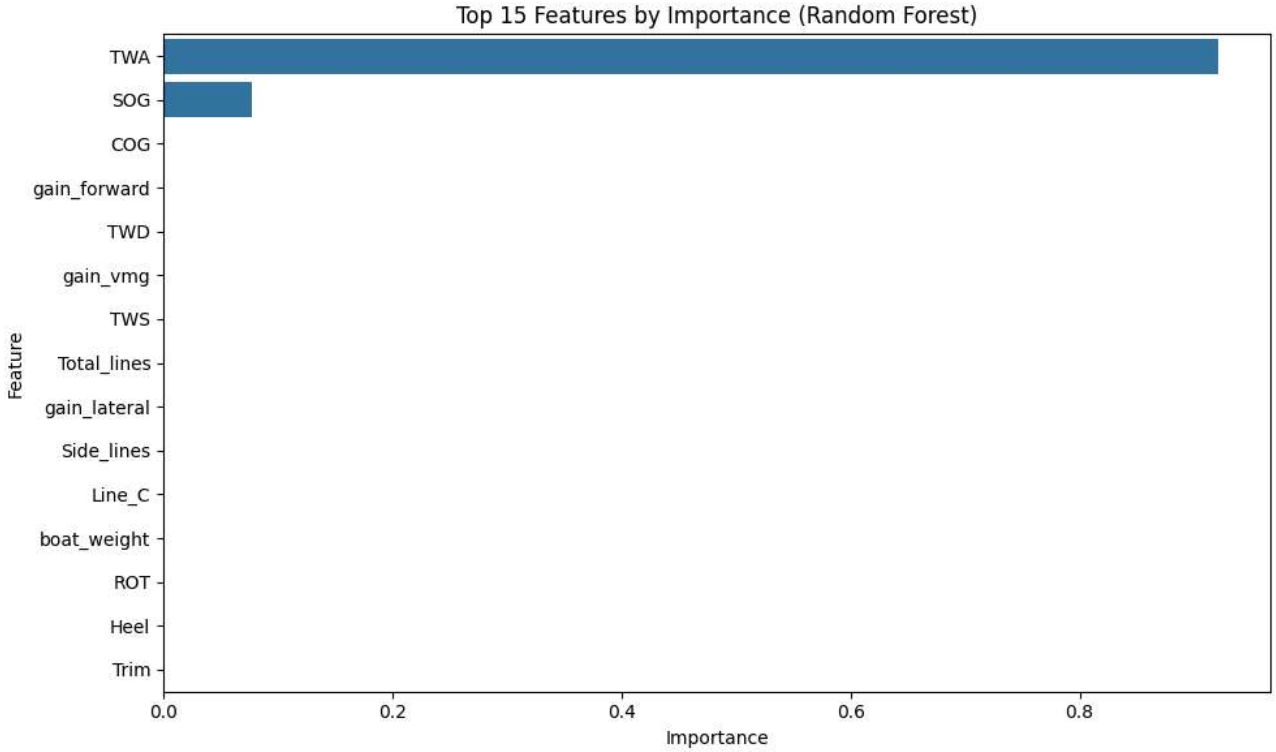
| | sum_sq | df | F | PR(>F) |
|---|---|---|---|---|
| **TWA** | 5276.057688 | 1.0 | 330144.760839 | 0.000000e+00 |
| **I(TWA ** 2)** | 2720.232975 | 1.0 | 170216.233019 | 0.000000e+00 |
| **I(SOG ** 2)** | 84.653123 | 1.0 | 5297.096186 | 0.000000e+00 |
| **TWD** | 31.305338 | 1.0 | 1958.904552 | 0.000000e+00 |
| **I(TWD ** 2)** | 29.884247 | 1.0 | 1869.980999 | 0.000000e+00 |
| **gain_forward** | 26.052039 | 1.0 | 1630.183908 | 0.000000e+00 |
| **gain_vmg** | 24.591264 | 1.0 | 1538.777111 | 0.000000e+00 |
| **gain_lateral** | 22.282049 | 1.0 | 1394.280022 | 1.620284e-297 |
| **I(COG ** 2)** | 21.473583 | 1.0 | 1343.690947 | 4.647449e-287 |
| **COG** | 19.410727 | 1.0 | 1214.609536 | 2.738799e-260 |
| **Line_C** | 15.299520 | 1.0 | 957.354254 | 1.474098e-206 |
| **SOG** | 14.955489 | 1.0 | 935.826840 | 4.875292e-202 |
| **Side_lines** | 8.079376 | 1.0 | 505.560001 | 6.276600e-111 |
| **I(Line_C ** 2)** | 6.589672 | 1.0 | 412.343027 | 5.509846e-91 |
| **Total_lines** | 4.861287 | 1.0 | 304.190812 | 9.556296e-68 |
| **I(boat_weight ** 2)** | 3.050777 | 1.0 | 190.899744 | 2.849881e-43 |
| **boat_weight** | 3.021036 | 1.0 | 189.038716 | 7.214097e-43 |
| **I(TWS ** 2)** | 1.796496 | 1.0 | 112.414169 | 3.269062e-26 |
| **TWS** | 1.787247 | 1.0 | 111.835460 | 4.371801e-26 |
| **I(Side_lines ** 2)** | 1.489974 | 1.0 | 93.233816 | 5.048146e-22 |
| **Heel** | 0.614002 | 1.0 | 38.420630 | 5.785960e-10 |
| **I(Heel ** 2)** | 0.571786 | 1.0 | 35.779023 | 2.238228e-09 |
| **I(gain_lateral ** 2)** | 0.178033 | 1.0 | 11.140281 | 8.458838e-04 |
| **I(gain_vmg ** 2)** | 0.111598 | 1.0 | 6.983144 | 8.232888e-03 |
| **I(Total_lines ** 2)** | 0.105611 | 1.0 | 6.608486 | 1.015475e-02 |
| **ROT** | 0.086367 | 1.0 | 5.404336 | 2.009425e-02 |
| **I(gain_forward ** 2)** | 0.030495 | 1.0 | 1.908220 | 1.671727e-01 |
| **Trim** | 0.020246 | 1.0 | 1.266870 | 2.603648e-01 |
| **I(ROT ** 2)** | 0.014693 | 1.0 | 0.919391 | 3.376437e-01 |
| **I(Trim ** 2)** | 0.013539 | 1.0 | 0.847164 | 3.573642e-01 |
| **Residual** | 427.109361 | 26726.0 | NaN | NaN |

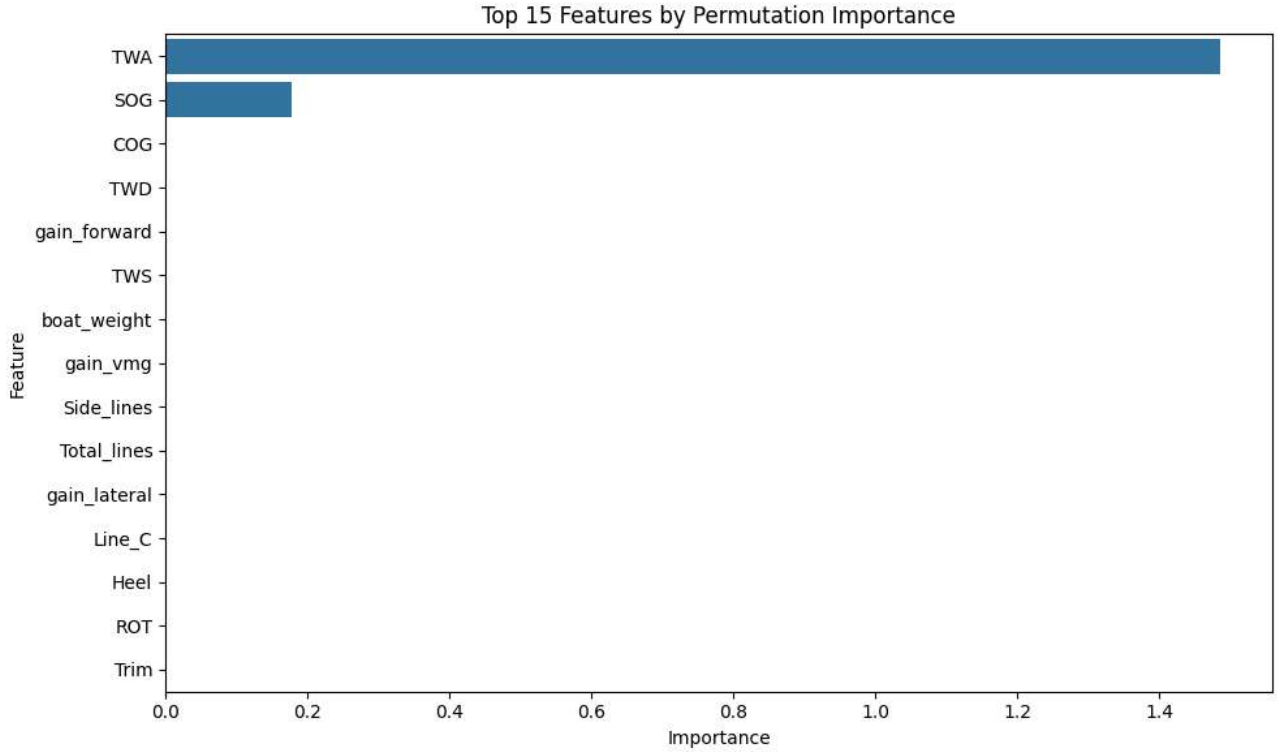
In [18]:
```
resultats_coefs_downwind = analyser_modele_polynomial(df_numeric_downwind)
display(resultats_coefs_downwind.head(30))
```

```
R² (sur toutes les données) : 1.000
```

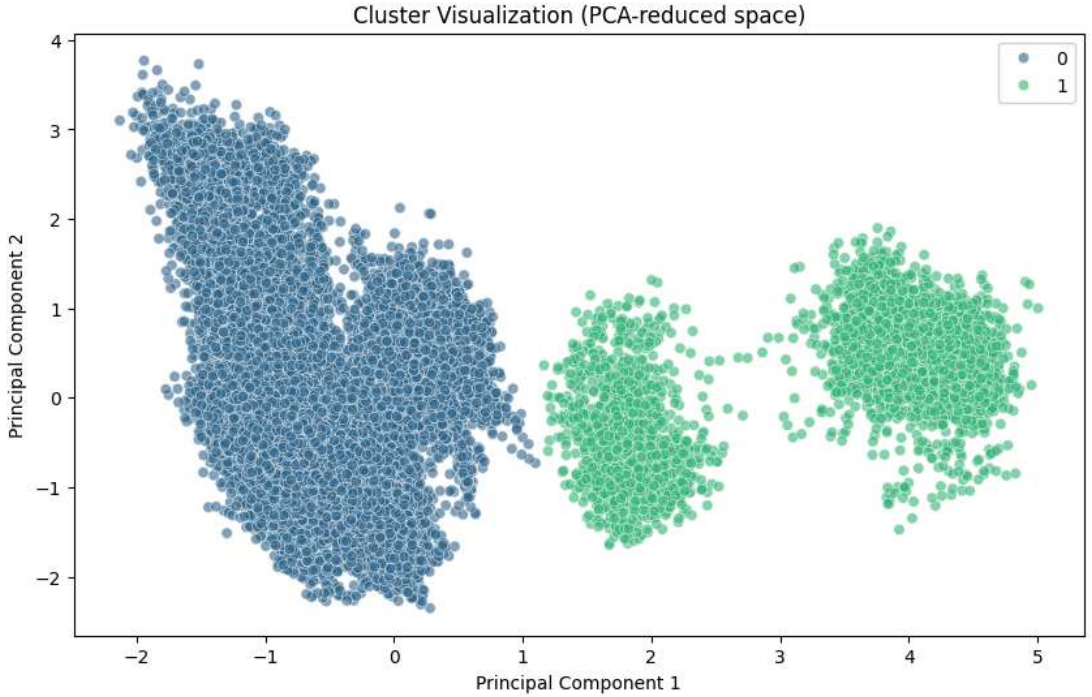
| | feature | coefficient |
|---|---|---|
| **91** | COG TWD | -27.205295 |
| **6** | COG | 25.469906 |
| **90** | COG^2 | 24.807386 |
| **93** | COG TWA | 20.239052 |
| **114** | TWA^2 | -6.705346 |
| **9** | TWA | -6.094274 |
| **99** | TWD^2 | 5.601602 |
| **7** | TWD | -5.165609 |
| **115** | TWA SOG | -4.258562 |
| **14** | gain_vmg | -4.247254 |
| **101** | TWD TWA | -4.043864 |
| **12** | gain_forward | -3.173090 |
| **106** | TWD gain_vmg | 1.973923 |
| **13** | gain_lateral | 1.930213 |
| **124** | SOG gain_vmg | 1.702934 |
| **0** | Line_C | 1.363891 |
| **104** | TWD gain_forward | 1.330989 |
| **10** | SOG | -1.260004 |
| **2** | Side_lines | 1.222105 |
| **122** | SOG gain_forward | 1.210389 |
| **105** | TWD gain_lateral | -0.971627 |
| **24** | Line_C TWA | 0.951422 |
| **3** | Total_lines | -0.939219 |
| **51** | Side_lines TWA | 0.793460 |
| **22** | Line_C TWD | -0.664731 |
| **63** | Total_lines TWA | -0.646292 |
| **125** | boat_weight^2 | 0.636542 |
| **123** | SOG gain_lateral | -0.597539 |
| **49** | Side_lines TWD | -0.581861 |
| **11** | boat_weight | -0.553719 |

In [19]:
```
feature_imp_downwind, perm_imp_downwind = analyze_feature_importance(df_numeric_downwind)
```


Top 15 Features by Importance (Random Forest)
TWA
SOG
COG
gain_forward
TWD
gain_vmg
TWS
Total_lines
gain_lateral
Side_lines
Line_C
boat_weight
ROT
Heel
Trim
Feature
0.0
0.2
0.4
0.6
0.8
Importance

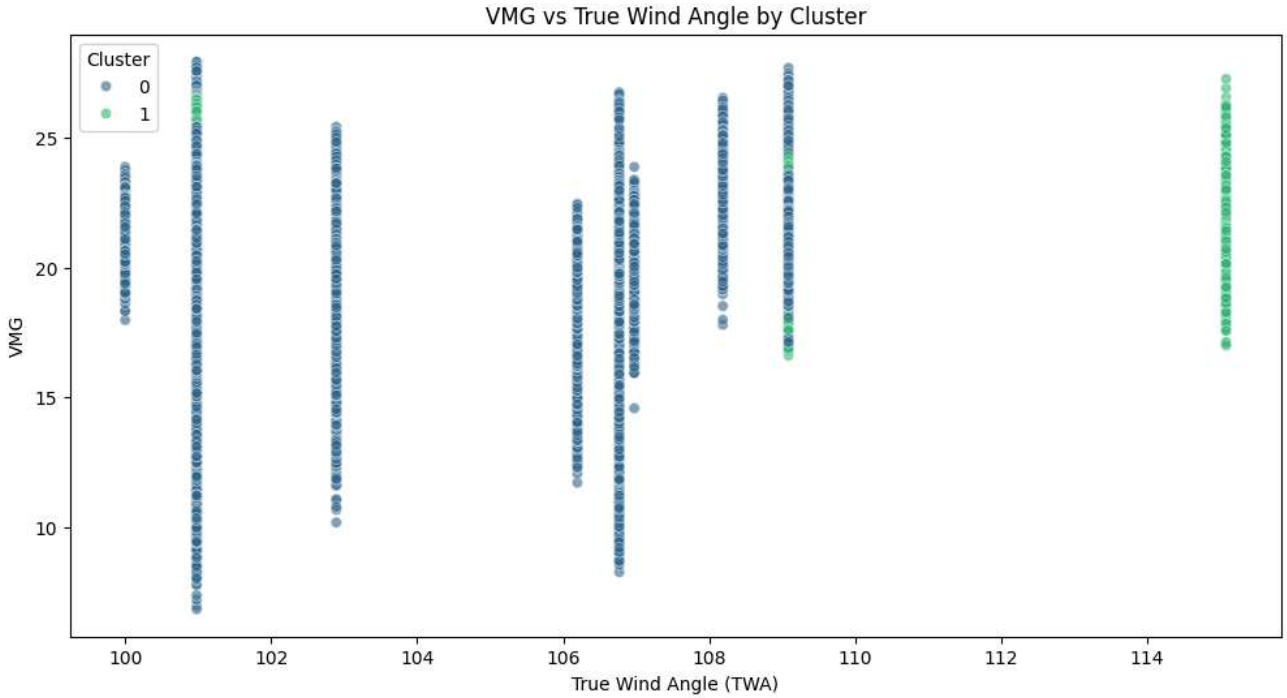



In [20]:
```
clustered_df_downwind, cluster_stats_downwind = perform_clustering(df_numeric_downwind, optimal_clusters=2)
visualize_vmg_twa(clustered_df_downwind)
```




Cluster Characteristics:

| Cluster | VMG | Side_lines | Line_C | COG | boat_weight |
|---|---|---|---|---|---|
| **0** | 20.167006 | 12.436882 | 90.228821 | 291.542612 | 104.641060 |
| **1** | 21.451826 | 99.765070 | 5.873776 | 106.598957 | 109.413162 |


VMG vs True Wind Angle by Cluster
Cluster
0
1
VMG
True Wind Angle (TWA)
10
15
20
25
100
102
104
106
108
110
112
114

In [ ]: