In [3]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [4]:
```
pd.options.display.max_columns = None
```

In [5]:
```
df = pd.read_csv('df.csv')
```

In [6]:
```
df
```

Out[6]:

	Age	Gender	Location	Debt	Owns Property	Platform	Total Time Spent	Video Category	Engagement	F
0	56	Male	Pakistan	True	True	Instagram	less	Pranks	high	
1	46	Female	Mexico	False	True	Instagram	high	Pranks	moderate	
2	32	Female	United States	False	True	Facebook	less	Vlogs	high	
3	60	Male	Barzil	True	False	YouTube	less	Vlogs	less	
4	25	Male	Pakistan	False	True	TikTok	moderate	Gaming	moderate	
...	...	...	...	...	...	...	...	...	...	
995	22	Male	India	True	True	TikTok	moderate	Gaming	moderate	
996	40	Female	Pakistan	False	False	Facebook	high	Life Hacks	less	
997	27	Male	India	True	True	TikTok	moderate	Pranks	high	
998	61	Male	Pakistan	True	False	YouTube	moderate	Life Hacks	less	
999	19	Male	India	True	True	YouTube	moderate	Pranks	high	

1000 rows × 16 columns

In [7]:
```
temp_df = df.drop(columns=['ProductivityLoss','Self Control'])
```

In [8]:

```python
import pickle
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report

# Assuming df is already defined
X = temp_df.drop('Addiction Level', axis=1)
y = temp_df['Addiction Level']

# Splitting the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.

# Identifying the categorical and numerical columns
categorical_cols = ['Gender', 'Location', 'Platform', 'Video Category'
numerical_cols = ['Age', 'Satisfaction']

# Column transformer to handle different data types
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_co
    ])

# Defining pipelines for different models
logreg_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(max_iter=1000))
])

rf_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier())
])

svc_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', SVC())
])

# List of pipelines to iterate over
pipelines = [('Logistic Regression', logreg_pipeline),
             ('Random Forest', rf_pipeline),
             ('SVM', svc_pipeline)]

# Training and saving models
for name, pipeline in pipelines:
    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)
    print(f"{name} Classification Report:")
    print(classification_report(y_test, y_pred))

print("Models saved successfully.")
```

```
Logistic Regression Classification Report:
              precision    recall  f1-score   support

     extreme       0.89      0.74      0.81        23
        high       0.92      0.89      0.91        76
         low       0.94      1.00      0.97        87
    moderate       0.90      0.90      0.90        58
no addiction       0.98      0.98      0.98        56

    accuracy                           0.93       300
   macro avg       0.93      0.90      0.91       300
weighted avg       0.93      0.93      0.93       300

Random Forest Classification Report:
              precision    recall  f1-score   support

     extreme       0.95      0.91      0.93        23
        high       0.96      0.95      0.95        76
         low       0.97      0.98      0.97        87
    moderate       0.95      0.95      0.95        58
no addiction       0.96      0.98      0.97        56

    accuracy                           0.96       300
   macro avg       0.96      0.95      0.96       300
weighted avg       0.96      0.96      0.96       300

SVM Classification Report:
              precision    recall  f1-score   support

     extreme       1.00      0.74      0.85        23
        high       0.93      0.97      0.95        76
         low       0.97      0.99      0.98        87
    moderate       0.95      0.93      0.94        58
no addiction       0.98      1.00      0.99        56

    accuracy                           0.96       300
   macro avg       0.96      0.93      0.94       300
weighted avg       0.96      0.96      0.96       300

Models saved successfully.
```

In [8]:

```
temp_df['Addiction Level'].value_counts()
```

Out[8]:

```
Addiction Level
low             308
high            262
moderate        195
no addiction    180
extreme          55
Name: count, dtype: int64
```

In [7]:

```
temp_df.to_csv('temp_df.csv', index=False)
```

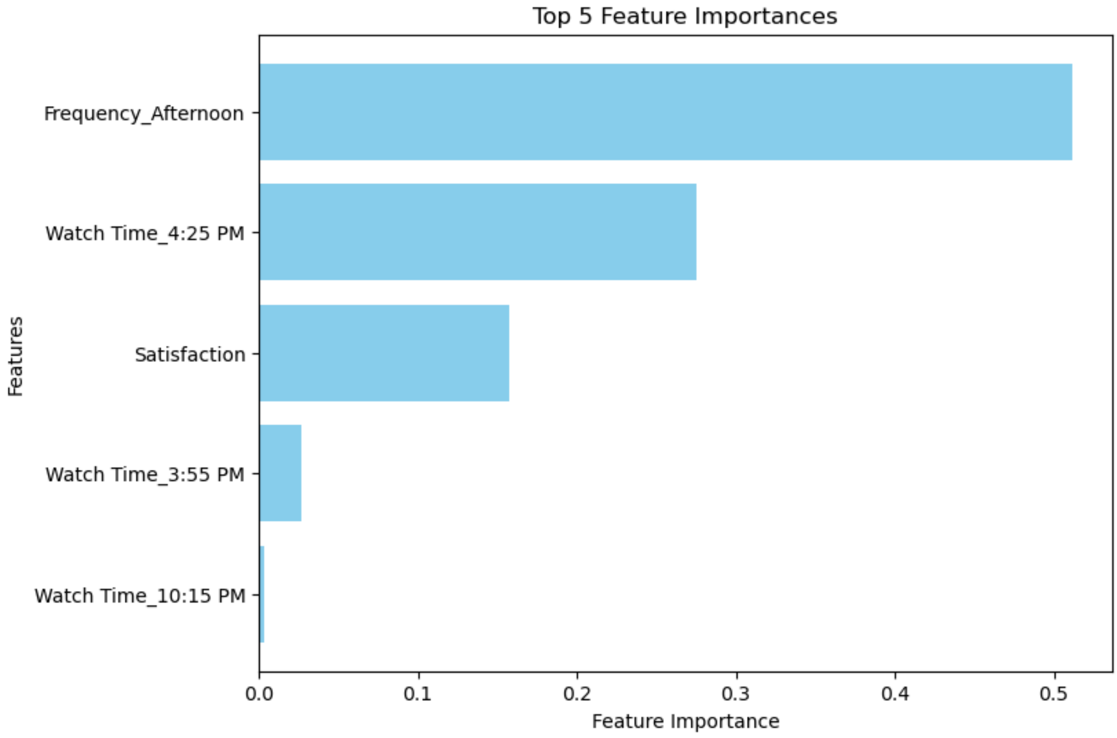
In [6]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pickle
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder, Label
from sklearn.metrics import accuracy_score, classification_report
from xgboost import XGBClassifier

# Assuming df is already defined
# Replace this with your actual dataset loading code
# Example:
# df = pd.read_csv("your_dataset.csv")

# Define features and target
# Replace `temp_df` with your actual DataFrame name
X = temp_df.drop('Addiction Level', axis=1)
y = temp_df['Addiction Level']

# Encode target labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# Splitting the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test

# Identifying the categorical and numerical columns
categorical_cols = ['Gender', 'Location', 'Platform', 'Video Category'
numerical_cols = ['Age', 'Satisfaction']

# Column transformer to handle different data types
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_co
    ])

# Preprocessing function
def preprocess_data(X, fit=True):
    if fit:
        return preprocessor.fit_transform(X)
    else:
        return preprocessor.transform(X)

# Preprocess train and test data
X_train_transformed = preprocess_data(X_train, fit=True)
X_test_transformed = preprocess_data(X_test, fit=False)

# Medium-Level Hyperparameter Tuning with GridSearchCV
param_grid = {
    'n_estimators': [100, 200],
    'learning_rate': [0.05, 0.1, 0.2],
    'max_depth': [3, 5, 7],
    'subsample': [0.8, 1.0]
}

xgb = XGBClassifier(random_state=42, use_label_encoder=False, eval_met
grid_search = GridSearchCV(estimator=xgb, param_grid=param_grid, cv=3,
grid_search.fit(X_train_transformed, y_train)

```

```python
# Best Model after Tuning
best_model = grid_search.best_estimator_
print("Best Parameters:", grid_search.best_params_)

# Make predictions
preds = best_model.predict(X_test_transformed)

# Evaluate the model
accuracy = accuracy_score(y_test, preds)
print("Model accuracy:", accuracy)

# Classification Report
report = classification_report(y_test, preds, target_names=label_encod
print("Classification Report:\n", report)

# Feature Importance Analysis
feature_names = (numerical_cols +
                 list(preprocessor.named_transformers_['cat'].get_feat

feature_importances = best_model.feature_importances_

# Extract top 5 features
top_5_idx = np.argsort(feature_importances)[-5:]
top_5_features = np.array(feature_names)[top_5_idx]
top_5_importances = feature_importances[top_5_idx]

# Display top 5 features
print("Top 5 Features:")
for feature, importance in zip(top_5_features[::-1], top_5_importances
    print(f"{feature}: {importance:.4f}")

# Plot top 5 feature importances
plt.figure(figsize=(8, 6))
plt.barh(top_5_features, top_5_importances, color="skyblue")
plt.xlabel('Feature Importance')
plt.ylabel('Features')
plt.title('Top 5 Feature Importances')
plt.show()

# Save the model to a .pkl file
with open('xgboost_model.pkl', 'wb') as model_file:
    pickle.dump(best_model, model_file)

# Save the preprocessor to a .pkl file
with open('preprocessor.pkl', 'wb') as preprocessor_file:
    pickle.dump(preprocessor, preprocessor_file)

print("Model and preprocessor saved as 'xgboost_model.pkl' and 'prepro
```

```
Fitting 3 folds for each of 36 candidates, totalling 108 fits

C:\Users\TIRTH PATEL\anaconda3\Lib\site-packages\xgboost\core.py:158: Use
rWarning: [20:04:39] WARNING: C:\buildkite-agent\builds\buildkite-windows
-cpu-autoscaling-group-i-0015a694724fa8361-1\xgboost\xgboost-ci-windows\s
rc\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

```
Best Parameters: {'learning_rate': 0.05, 'max_depth': 3, 'n_estimators':
100, 'subsample': 0.8}
Model accuracy: 0.9966666666666667
Classification Report:
               precision    recall  f1-score   support

     extreme       1.00      1.00      1.00        23
        high       1.00      1.00      1.00        76
         low       1.00      1.00      1.00        87
    moderate       1.00      0.98      0.99        58
no addiction       0.98      1.00      0.99        56

    accuracy                           1.00       300
   macro avg       1.00      1.00      1.00       300
weighted avg       1.00      1.00      1.00       300

Top 5 Features:
Frequency_Afternoon: 0.5114
Watch Time_4:25 PM: 0.2756
Satisfaction: 0.1579
Watch Time_3:55 PM: 0.0270
Watch Time_10:15 PM: 0.0034
```



```
Model and preprocessor saved as 'xgboost_model.pkl' and 'preprocessor.pk
l'
```