In [1]:
```
# From: https://www.kaggle.com/datasets/naddamuhhamed/sleepy-driver-eeg-brai
# Download data as zip: "acquiredDataset.csv"
```

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
import pandas as pd
import seaborn as sns

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import StandardScaler, OneHotEncoder

from scipy.stats import ttest_ind
```

Data Overview (`acquiredDataset.csv`)

- The dataset contains EEG brainwave data collected from people in both awake and asleep states using a NeuroSky MindWave sensor
- Features include attention scores, meditation (calmness) levels, and various brainwave frequencies (delta, theta, alpha, beta, gamma)
- Binary classification: 0 = Awake, 1 = Sleepy

Analysis Steps & Findings:

In [3]:
```
data = pd.read_csv('acquiredDataset.csv')
```

In [4]:
```
data.head()
```

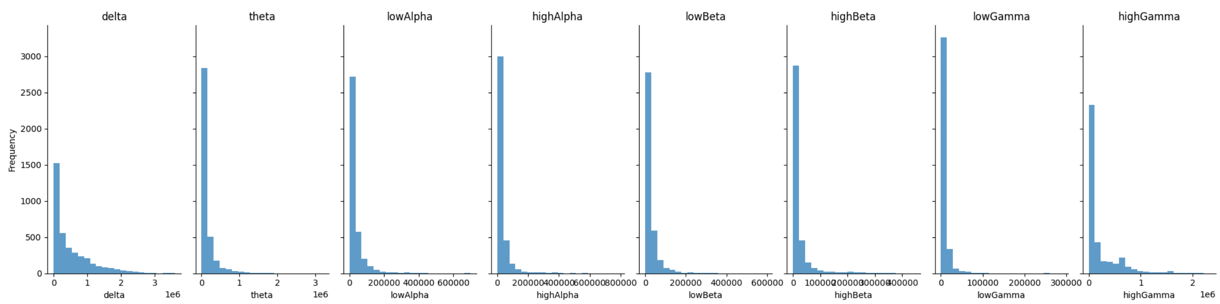
Out[4]:

	attention	meditation	delta	theta	lowAlpha	highAlpha	lowBeta	highBeta	lo
0	26	34	960462	277180	26575	27356	26575	13056	
1	29	54	39145	28225	20172	39551	20172	9933	
2	40	48	75410	43144	8601	13564	8601	11663	
3	66	47	16057	41211	2534	34254	2534	27663	
4	81	67	10304	47239	33158	47349	33158	16328	

1. Initial Data Exploration

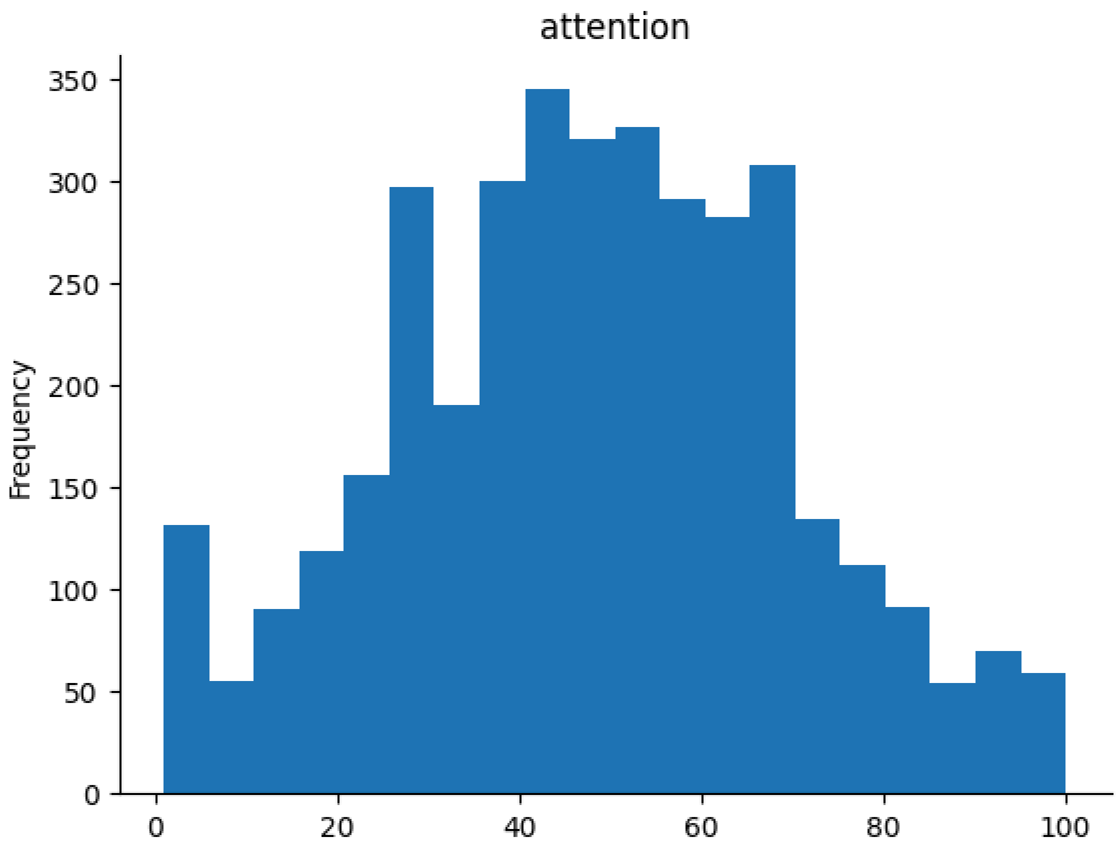
In [5]:

```
# @title EEG waveform frequency
columns = ['delta', 'theta', 'lowAlpha', 'highAlpha', 'lowBeta', 'highBeta',

# Create subplots
fig, axes = plt.subplots(1, len(columns), figsize=(20, 5), sharey=True)

# Plot each column
for ax, col in zip(axes, columns):
    data[col].plot(kind='hist', bins=20, ax=ax, title=col, alpha=0.7)
    ax.spines[['top', 'right']].set_visible(False)
    ax.set_xlabel(col)
    ax.set_ylabel('Frequency')

# Adjust layout
plt.tight_layout()
plt.show()
```



In [6]:

```
# @title attention

from matplotlib import pyplot as plt
data['attention'].plot(kind='hist', bins=20, title='attention')
plt.gca().spines[['top', 'right',]].set_visible(False)
```



Recap of 1. **Initial Data Exploration**

- Visualized distribution of brainwave frequencies
- Examined attention scores distribution

---

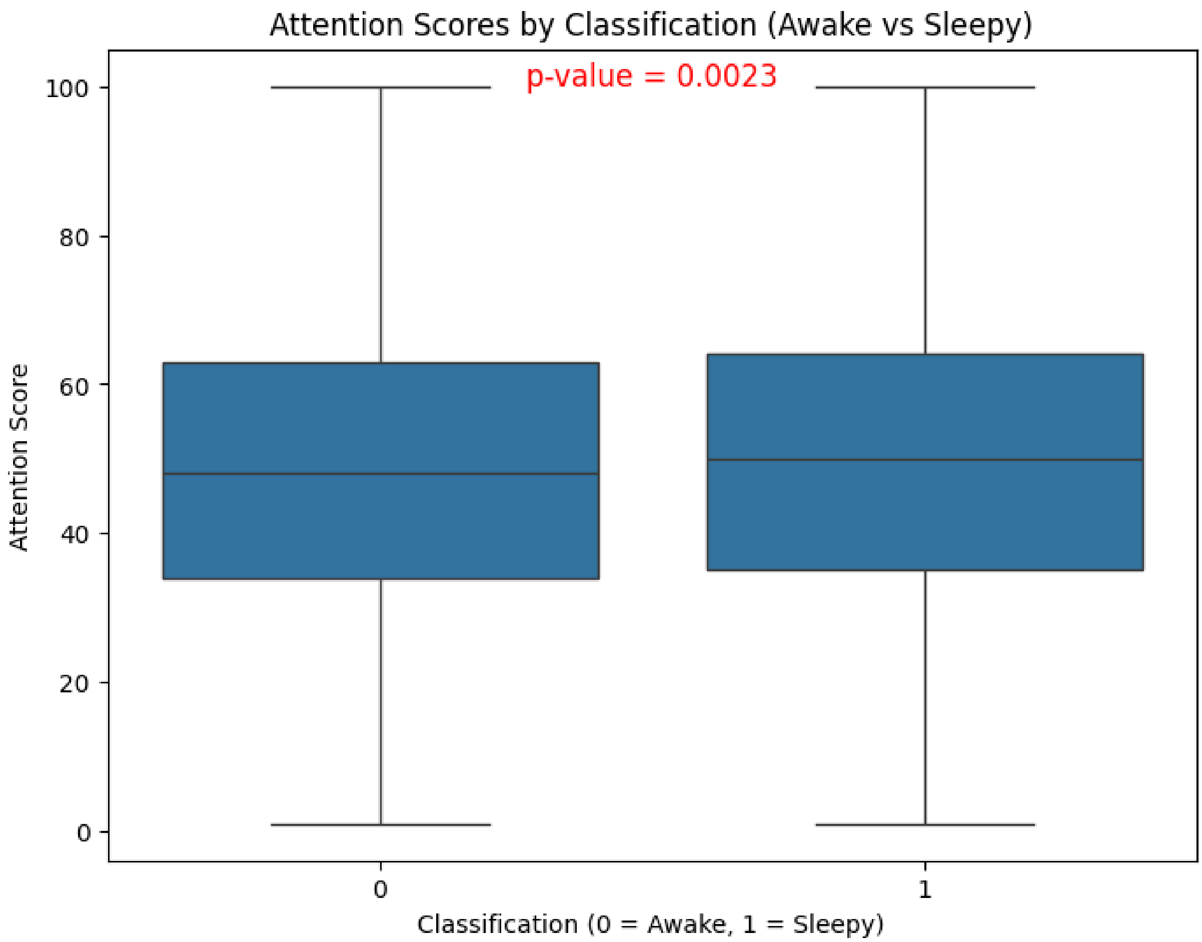
2. Statistical Analysis

Are sleep and attention related?

- We will separate the data according to sleepy vs awake - according to our project's hypothesis, the sleepy drivers should have lower attention scores due to poor sleep quality

In [7]:
```
group_awake = data[data['classification'] == 0]['attention']
group_sleepy = data[data['classification'] == 1]['attention']

t_stat, p_val = ttest_ind(group_awake, group_sleepy, equal_var=False)

plt.figure(figsize=(8, 6))
sns.boxplot(x='classification', y='attention', data=data)
plt.title("Attention Scores by Classification (Awake vs Sleepy)")

plt.text(0.5, max(data['attention']), f"p-value = {p_val:.4f}",
```

```
        horizontalalignment='center', fontsize=12, color='red')
plt.xlabel("Classification (0 = Awake, 1 = Sleepy)")
plt.ylabel("Attention Score")
plt.show()
```

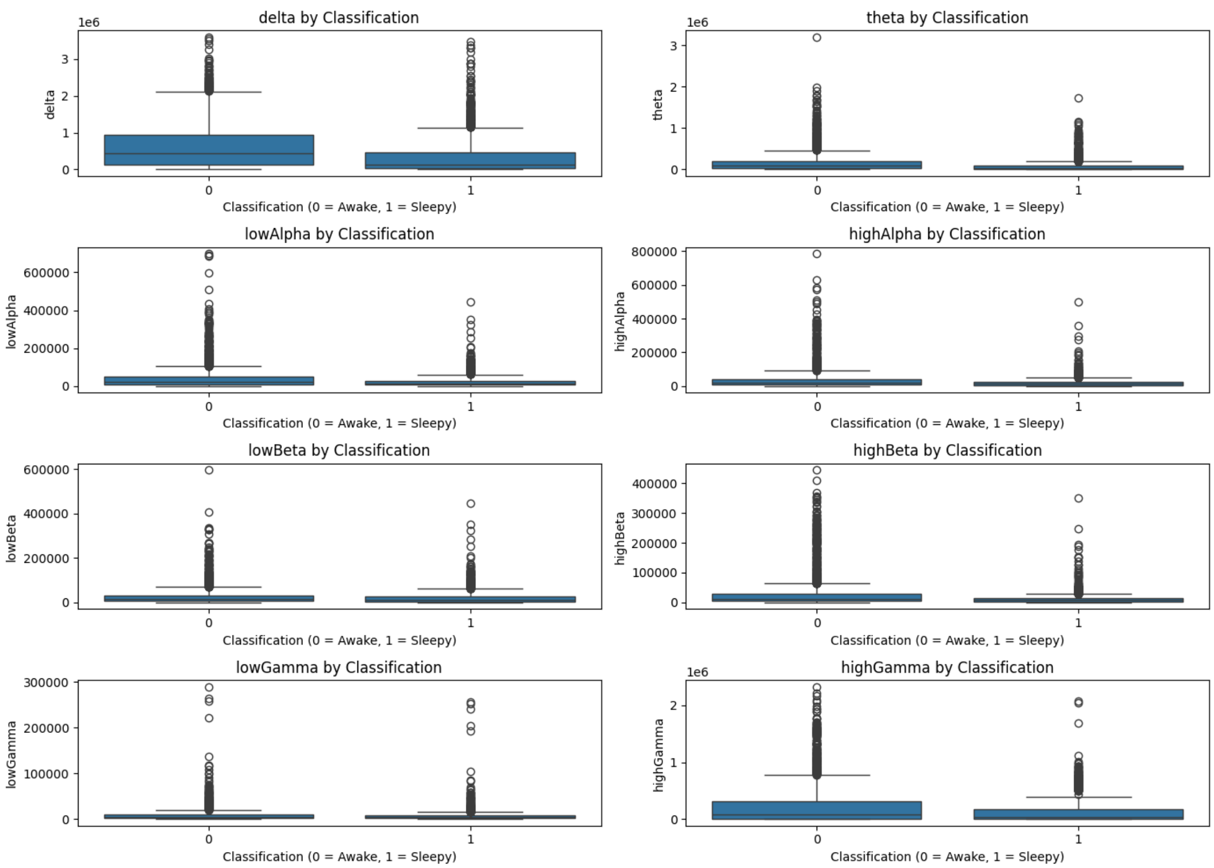


Since p=0.0023 < 0.05, there is a statistically significant difference between the two groups.

Now, will the brainwave data differ statistically between the two groups?

In [8]:
```
eeg = ["delta", "theta", "lowAlpha", "highAlpha", "lowBeta", "highBeta", "lo


plt.figure(figsize=(14, 10))
for i, col in enumerate(eeg, 1):
    plt.subplot(4, 2, i)
    sns.boxplot(x='classification', y=col, data=data)
    plt.title(f"{col} by Classification")
    plt.xlabel("Classification (0 = Awake, 1 = Sleepy)")
    plt.ylabel(col)
plt.tight_layout()
plt.show()

results = {}
for col in eeg:
    group_awake = data[data['classification'] == 0][col]
```

```python
    group_sleepy = data[data['classification'] == 1][col]

    t_stat, p_val = ttest_ind(group_awake, group_sleepy, equal_var=False)
    results[col] = {'t_stat': t_stat, 'p_val': p_val}

print("Statistical Test Results:")
for feature, result in results.items():
    print(f"{feature}: t-stat = {result['t_stat']:.4f}, p-value = {result['p
```




```
Statistical Test Results:
delta: t-stat = 14.8369, p-value = 2.0308e-48
theta: t-stat = 13.8615, p-value = 1.4658e-42
lowAlpha: t-stat = 11.7861, p-value = 1.9027e-31
highAlpha: t-stat = 13.1178, p-value = 2.7364e-38
lowBeta: t-stat = 4.0104, p-value = 6.1816e-05
highBeta: t-stat = 14.9164, p-value = 1.7063e-48
lowGamma: t-stat = 3.0203, p-value = 2.5433e-03
highGamma: t-stat = 6.6961, p-value = 2.4624e-11
```

Based on statistical analysis, it seems as though **high beta and delta waves** are the most statisically different amongst sleepy vs. awake individuals, closely followed by **delta and theta waves**. This validates our literature review which states "Fast frequencies correspond to beta (13 to 25) and gamma (25 to 60 Hz) waves. They are associated with a high state of vigilance, or cognitive activity. Slow waves correspond to theta (4 to 8 Hz) and delta waves (1 to 4 Hz), and are associated with a state of drowsiness and sleep, respectively." (Terlow 2016).

Recap of 2. **Statistical Analysis**

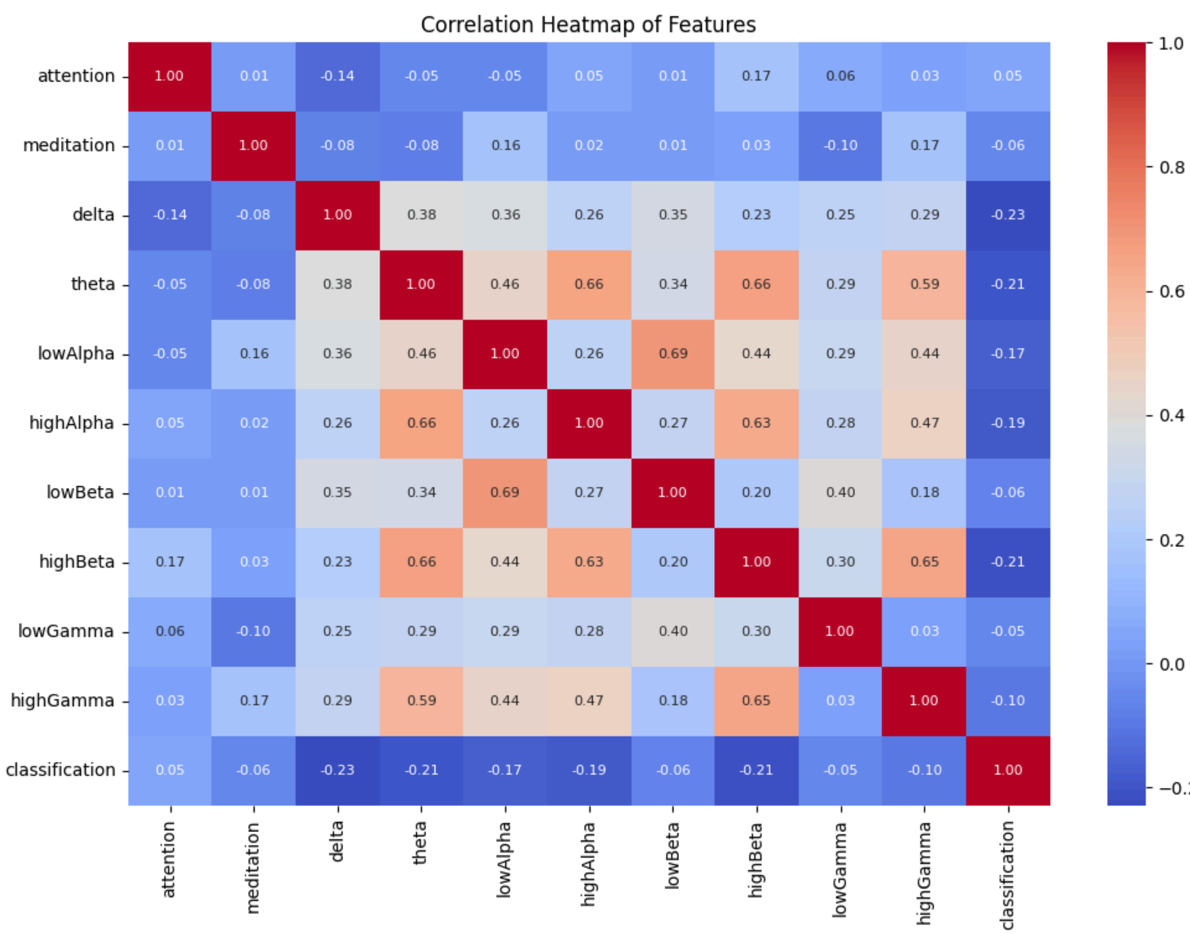
- Conducted t-tests between awake/sleepy groups
- Found significant differences in attention scores (p=0.0023)
- High beta and delta waves showed strongest statistical differences between states
- Results aligned with literature: fast frequencies (beta/gamma) indicate vigilance, slow waves (theta/delta) indicate drowsiness

---

3. Feature Relationships

In [9]:

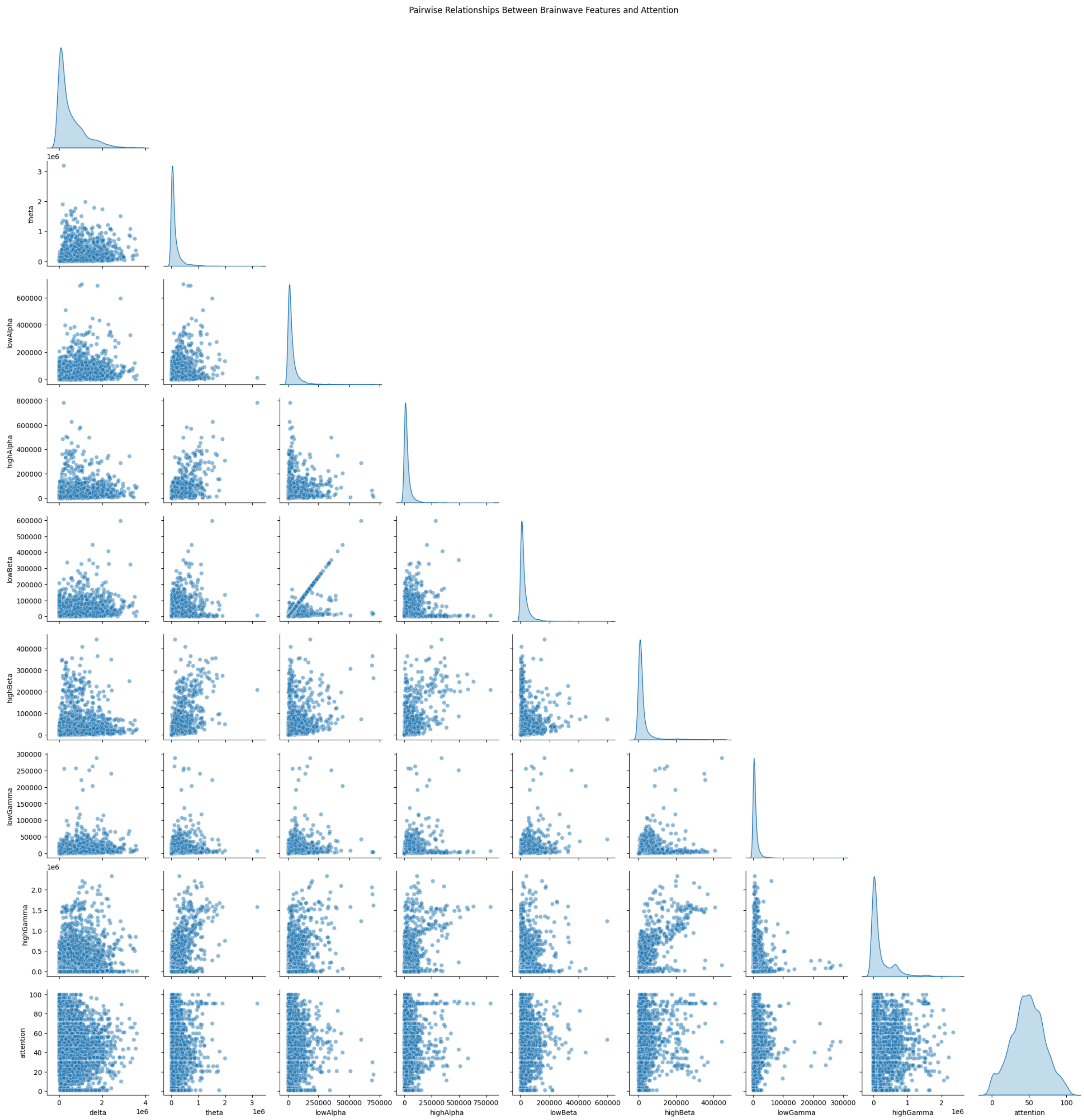
```
correlation_matrix = data.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap="coolwarm", cbar
plt.title("Correlation Heatmap of Features")
plt.show()
```



In [10]:

```
selected_columns = [
    "delta", "theta", "lowAlpha", "highAlpha",
    "lowBeta", "highBeta", "lowGamma", "highGamma", "attention"
]

missing_columns = [col for col in selected_columns if col not in data.columr
if missing_columns:
    print(f"Missing columns in dataset: {missing_columns}")
else:
```

```
    sns.pairplot(data[selected_columns], diag_kind="kde", corner=True, plot_
    plt.suptitle("Pairwise Relationships Between Brainwave Features and Atte
    plt.show()
```



Recap of 3. **Feature Relationships**

- Created correlation heatmap
- Generated pairwise plots to visualize relationships between brainwaves and attention

---

4. Predictive Modeling

Can we Predict Attention (Cognitive Performace) based on EEG signals from sleep?

- We will train a few regression models as well as a neural network and compare their performaces.
- Since we proved strong statistically significant differences between brainwaves and sleep state, we will train the model usign the classification column: it will function as a one hot encoder.

In [11]:
```
eeg = data[['delta', 'theta', 'lowAlpha', 'highAlpha', 'lowBeta', 'highBeta'
target = data['attention']
encoder = OneHotEncoder(drop='first', sparse_output=False)  # drop='first' a
classification_encoded = encoder.fit_transform(data[['classification']])
classification_df = pd.DataFrame(classification_encoded, columns=['classific
eeg_encode = pd.concat([eeg, classification_df], axis=1)  # Concatenate data
```

In [12]:
```
X_train, X_test, y_train, y_test = train_test_split(eeg_encode, target, test
```

In [13]:
```
# scaler = StandardScaler()
# X_train_scaled = scaler.fit_transform(X_train)
# X_test_scaled = scaler.transform(X_test)
```

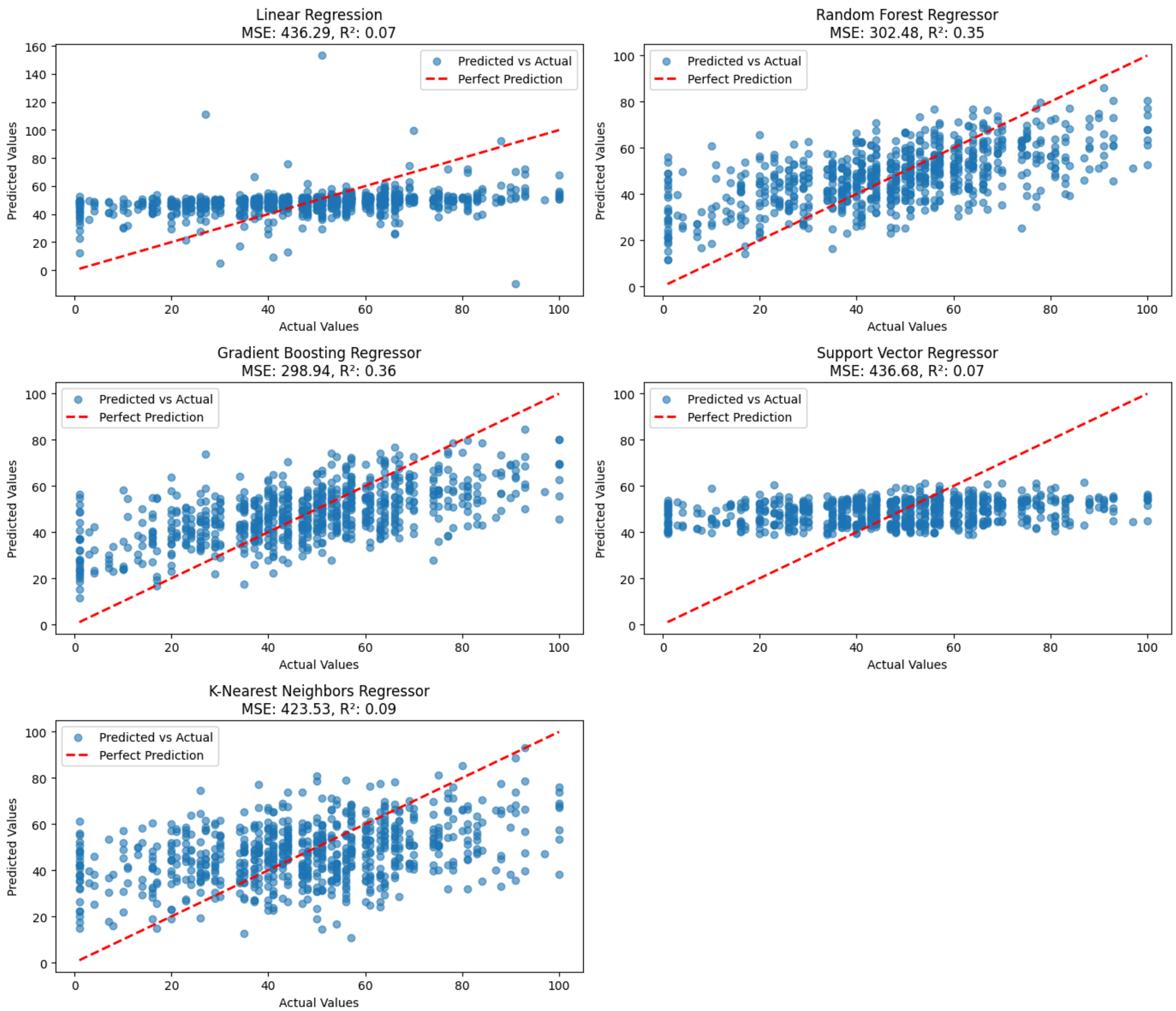
In [14]:
```
models = {
    "Linear Regression": LinearRegression(),
    "Random Forest Regressor": RandomForestRegressor(random_state=42),
    "Gradient Boosting Regressor": GradientBoostingRegressor(random_state=42
    "Support Vector Regressor": SVR(kernel='rbf'),
    "K-Nearest Neighbors Regressor": KNeighborsRegressor()
}
```

In [15]:
```
results = {}
```

In [16]:
```
for model_name, model in models.items():

    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    results[model_name] = {"MSE": mse, "R²": r2, "Predictions": y_pred}
```

In [17]:
```
fig, axes = plt.subplots(3, 2, figsize=(14, 12))
axes = axes.flatten()

for i, (model_name, result) in enumerate(results.items()):
    ax = axes[i]
    ax.scatter(y_test, result["Predictions"], alpha=0.6, label='Predicted vs
    ax.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--
    ax.set_title(f"{model_name}\nMSE: {result['MSE']:.2f}, R²: {result['R²']
    ax.set_xlabel("Actual Values")
    ax.set_ylabel("Predicted Values")
```

```
    ax.legend()

plt.tight_layout()
for j in range(len(results), len(axes)):
    axes[j].axis('off')

plt.show()
```



Model Performance Summary

Model	MSE	R^2
Linear Regression	436.29	0.07
Random Forest	302.48	0.35
Gradient Boosting	298.94	0.36
SVR	436.68	0.07
KNN	423.53	0.09

Best performing model: Gradient Boosting Regressor

→ The Gradient Boosting Regressor performed the highest with a high R^2 score and lowest MSE.

Neural Network

In [18]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

In [19]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [20]:
```
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train_scaled.shape[1],)),
    Dropout(0.2),
    Dense(32, activation='relu'),
    Dense(1)
])

model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])
model.summary()
```

```
/Users/raviriley/Library/Caches/pypoetry/virtualenvs/sleep--5W40_9x-py3.10/l
ib/python3.10/site-packages/keras/src/layers/core/dense.py:87: UserWarning:
Do not pass an `input_shape`/`input_dim` argument to a layer. When using Seq
uential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Par
dense (Dense)	(None, 64)	
dropout (Dropout)	(None, 64)	
dense_1 (Dense)	(None, 32)	2
dense_2 (Dense)	(None, 1)	

Total params: 2,753 (10.75 KB)

Trainable params: 2,753 (10.75 KB)

Non-trainable params: 0 (0.00 B)

In [21]:
```
history = model.fit(
    X_train_scaled, y_train,
    validation_split=0.2,
```

```
    epochs=50,
    batch_size=32,
    verbose=1
)
```

```
Epoch 1/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 2735.3459 - mae: 47.4645 - va
l_loss: 2258.6592 - val_mae: 42.4439
Epoch 2/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 922us/step - loss: 1957.2992 - mae: 38.6921 -
val_loss: 886.0980 - val_mae: 24.3095
Epoch 3/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 889us/step - loss: 748.3942 - mae: 22.1919 - v
al_loss: 634.3730 - val_mae: 20.1251
Epoch 4/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 876us/step - loss: 628.5749 - mae: 19.8538 - v
al_loss: 562.7963 - val_mae: 18.8466
Epoch 5/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 881us/step - loss: 573.1982 - mae: 18.8778 - v
al_loss: 515.5413 - val_mae: 18.0104
Epoch 6/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 863us/step - loss: 487.1070 - mae: 17.5834 - v
al_loss: 484.9637 - val_mae: 17.4665
Epoch 7/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 893us/step - loss: 497.5269 - mae: 17.6226 - v
al_loss: 462.9465 - val_mae: 17.1019
Epoch 8/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 845us/step - loss: 471.6473 - mae: 17.2226 - v
al_loss: 448.3519 - val_mae: 16.8259
Epoch 9/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 442.9675 - mae: 16.7267 - val
_loss: 435.7083 - val_mae: 16.6388
Epoch 10/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 888us/step - loss: 448.9631 - mae: 16.9308 - v
al_loss: 426.8712 - val_mae: 16.5239
Epoch 11/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 854us/step - loss: 430.4935 - mae: 16.5378 - v
al_loss: 420.9412 - val_mae: 16.4028
Epoch 12/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 919us/step - loss: 427.4755 - mae: 16.6559 - v
al_loss: 415.5627 - val_mae: 16.2951
Epoch 13/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 820us/step - loss: 418.5377 - mae: 16.2259 - v
al_loss: 411.4618 - val_mae: 16.2289
Epoch 14/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 869us/step - loss: 417.7725 - mae: 16.2187 - v
al_loss: 406.9445 - val_mae: 16.1383
Epoch 15/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 875us/step - loss: 414.7763 - mae: 16.2908 - v
al_loss: 405.0710 - val_mae: 16.0758
Epoch 16/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 946us/step - loss: 436.0999 - mae: 16.6309 - v
al_loss: 401.6443 - val_mae: 16.0344
Epoch 17/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 890us/step - loss: 429.8110 - mae: 16.6514 - v
al_loss: 399.4407 - val_mae: 15.9647
Epoch 18/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 927us/step - loss: 398.1504 - mae: 15.8413 - v
al_loss: 397.5006 - val_mae: 15.9337
Epoch 19/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 969us/step - loss: 409.2348 - mae: 16.1791 - v
```

```
al_loss: 399.6777 - val_mae: 16.0000
Epoch 20/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 937us/step - loss: 404.3939 - mae: 16.0681 - v
al_loss: 395.2018 - val_mae: 15.9041
Epoch 21/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 943us/step - loss: 408.3559 - mae: 16.0066 - v
al_loss: 393.7819 - val_mae: 15.8819
Epoch 22/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 915us/step - loss: 400.5138 - mae: 15.9738 - v
al_loss: 391.8469 - val_mae: 15.8006
Epoch 23/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 876us/step - loss: 376.3546 - mae: 15.6291 - v
al_loss: 393.3839 - val_mae: 15.7893
Epoch 24/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 894us/step - loss: 419.7965 - mae: 16.2438 - v
al_loss: 389.7506 - val_mae: 15.7591
Epoch 25/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 952us/step - loss: 402.2542 - mae: 15.9773 - v
al_loss: 389.6898 - val_mae: 15.7816
Epoch 26/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 946us/step - loss: 378.0498 - mae: 15.5387 - v
al_loss: 390.7380 - val_mae: 15.7264
Epoch 27/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 950us/step - loss: 397.7905 - mae: 15.8997 - v
al_loss: 387.1385 - val_mae: 15.6956
Epoch 28/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 883us/step - loss: 403.6649 - mae: 15.9662 - v
al_loss: 387.7490 - val_mae: 15.7545
Epoch 29/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 954us/step - loss: 382.8876 - mae: 15.4865 - v
al_loss: 385.8104 - val_mae: 15.6824
Epoch 30/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 933us/step - loss: 411.3086 - mae: 16.1755 - v
al_loss: 383.9679 - val_mae: 15.6294
Epoch 31/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 922us/step - loss: 397.2262 - mae: 15.8384 - v
al_loss: 384.1627 - val_mae: 15.6072
Epoch 32/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 894us/step - loss: 402.7354 - mae: 16.0752 - v
al_loss: 381.7791 - val_mae: 15.5853
Epoch 33/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 880us/step - loss: 412.0653 - mae: 16.1625 - v
al_loss: 380.7258 - val_mae: 15.5835
Epoch 34/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 897us/step - loss: 420.3224 - mae: 16.3728 - v
al_loss: 379.0074 - val_mae: 15.5341
Epoch 35/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 402.5019 - mae: 16.0123 - val
_loss: 378.3146 - val_mae: 15.5175
Epoch 36/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 883us/step - loss: 399.3326 - mae: 15.8930 - v
al_loss: 378.7667 - val_mae: 15.5098
Epoch 37/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 933us/step - loss: 381.1002 - mae: 15.6498 - v
al_loss: 377.5035 - val_mae: 15.4959
Epoch 38/50
```

```
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 907us/step - loss: 385.8171 - mae: 15.7196 - v
al_loss: 375.6226 - val_mae: 15.4525
Epoch 39/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 902us/step - loss: 368.4117 - mae: 15.3309 - v
al_loss: 372.4229 - val_mae: 15.3894
Epoch 40/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 898us/step - loss: 383.2478 - mae: 15.6257 - v
al_loss: 372.6351 - val_mae: 15.4003
Epoch 41/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 911us/step - loss: 413.9254 - mae: 16.3524 - v
al_loss: 371.8273 - val_mae: 15.3822
Epoch 42/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 372.6580 - mae: 15.2969 - val
_loss: 371.3007 - val_mae: 15.3657
Epoch 43/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 975us/step - loss: 391.9673 - mae: 15.8021 - v
al_loss: 371.5344 - val_mae: 15.3765
Epoch 44/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 965us/step - loss: 383.0977 - mae: 15.4571 - v
al_loss: 371.2492 - val_mae: 15.3377
Epoch 45/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 371.2884 - mae: 15.3169 - val
_loss: 369.0620 - val_mae: 15.2968
Epoch 46/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 905us/step - loss: 381.9666 - mae: 15.5146 - v
al_loss: 365.8521 - val_mae: 15.2631
Epoch 47/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 878us/step - loss: 380.6219 - mae: 15.7021 - v
al_loss: 365.5903 - val_mae: 15.2376
Epoch 48/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 876us/step - loss: 374.9511 - mae: 15.4957 - v
al_loss: 364.6068 - val_mae: 15.2180
Epoch 49/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 911us/step - loss: 376.6045 - mae: 15.4717 - v
al_loss: 364.9396 - val_mae: 15.2102
Epoch 50/50
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 866us/step - loss: 365.2273 - mae: 15.1515 - v
al_loss: 364.3828 - val_mae: 15.1714
```

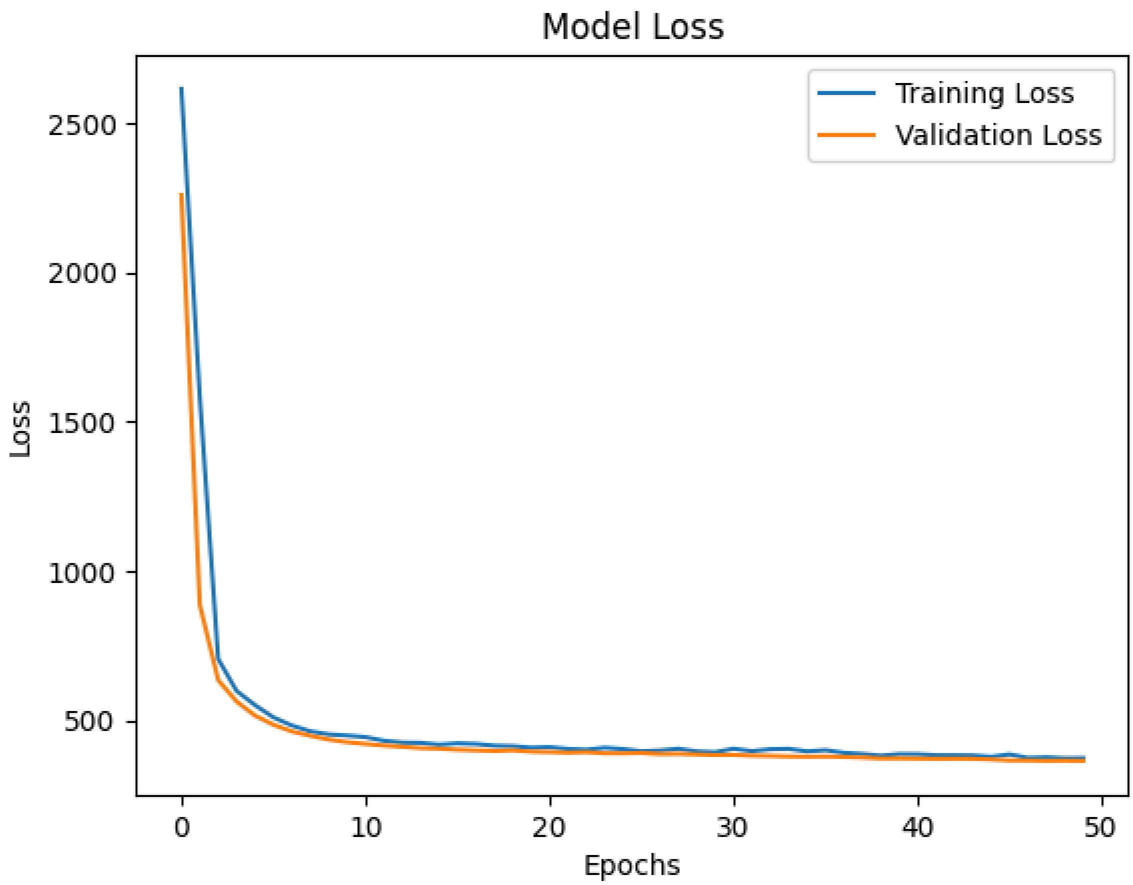
In [22]:

```python
test_loss, test_mae = model.evaluate(X_test_scaled, y_test, verbose=0)
print(f"Test Loss (MSE): {test_loss}, Test MAE: {test_mae}")
```

```
Test Loss (MSE): 356.1883239746094, Test MAE: 14.772906303405762
```

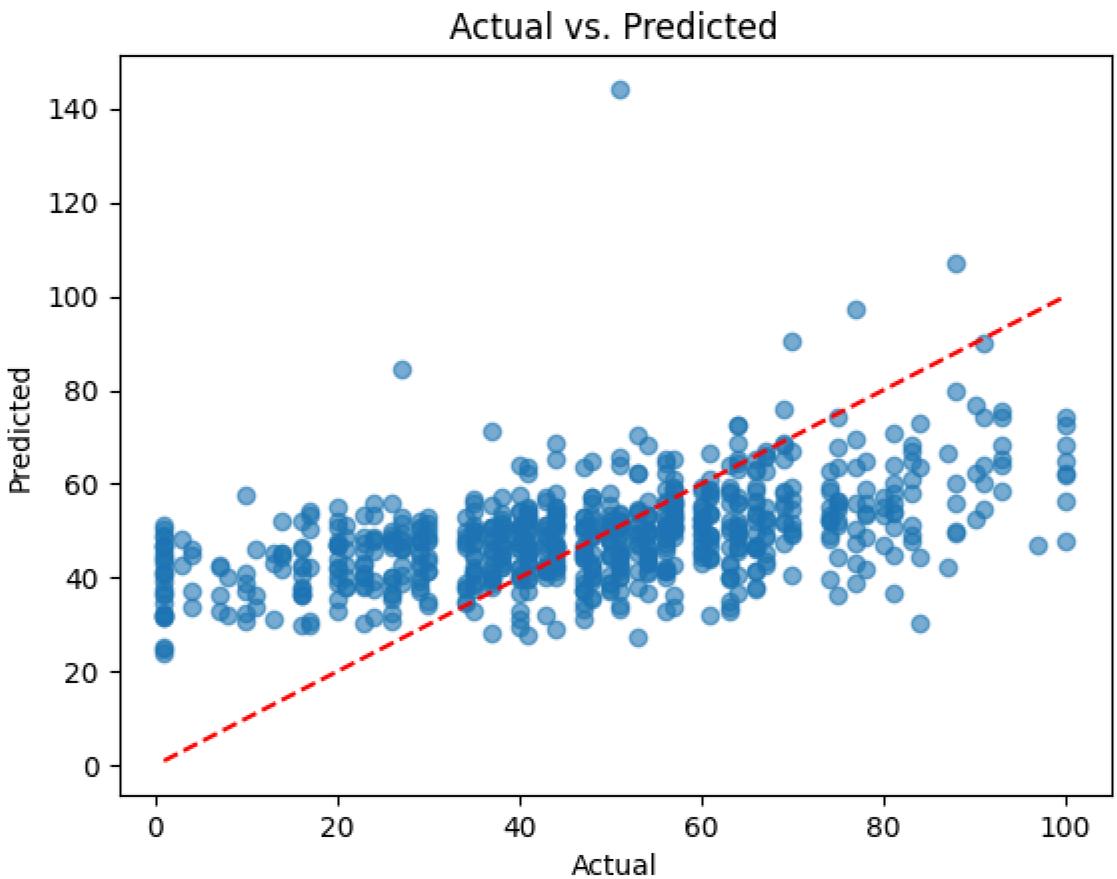
In [23]:

```python
import matplotlib.pyplot as plt

plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



In [24]:
```
y_pred = model.predict(X_test_scaled)


plt.scatter(y_test, y_pred, alpha=0.6)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual vs. Predicted')
plt.show()
```

```
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```



Recap of 4. **Predictive Modeling**

- Goal: Predict attention based on EEG signals
- Tested multiple regression models:
  - Linear Regression
  - Random Forest
  - Gradient Boosting (best performer)
  - SVR
  - KNN
- Implemented Neural Network:
  - 2-layer architecture with dropout
  - Used StandardScaler for feature normalization
  - Monitored training/validation loss

Conclusion

Key Findings

1. Significant difference in attention scores between awake and sleepy states ($p=0.0023$)

2. Gradient Boosting achieved best prediction performance

Limitations

1. Single-channel EEG data
2. Limited sample size
3. Controlled environment

Future Work

1. Collect multi-channel EEG data
2. Could add feature importance analysis
3. Could do cross-validation for more robust model evaluation
4. Might benefit from hyperparameter tuning
5. Could explore more advanced signal processing techniques for EEG data