Predictive Analysis for Vehicle Transmission Detection

This project aims to develop a predictive model that estimates the likelihood of a car having manual transmission based on various automotive attributes. The dataset used for this project is sourced from the 1974 Motor Trend US magazine, encompassing essential factors like fuel efficiency and 10 distinct aspects of vehicle design and performance.

Data Source

The data was extracted from the 1974 Motor Trend US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models). Henderson and Velleman (1981), Building multiple regression models interactively. Biometrics, 37, 391–411.

Format

A data frame with 32 observations on 11 (numeric) variables.

- [, 1] mpg Miles/(US) gallon
- [, 2] cyl Number of cylinders
- [, 3] disp Displacement (cu.in.)
- [, 4] hp Gross horsepower
- [, 5] drat Rear axle ratio
- [, 6] wt Weight (1000 lbs)
- [, 7] qsec 1/4 mile time
- [, 8] vs Engine (0 = V-shaped, 1 = straight)
- [, 9] am Transmission (0 = automatic, 1 = manual)
- [,10] gear Number of forward gears
- [,11] carb Number of carburetors

Table of Contents

1. Imports

In [ ]:
```
#basic libraries for linear algebra and data processing
import numpy as np
import pandas as pd

#visualization
import matplotlib.pyplot as plt
import seaborn as sns

#data preprocesing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

#models
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neural_network import MLPClassifier
from xgboost import XGBClassifier

#evaluation
import random
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_validate, cross_val_score
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score, f
1_score, classification_report, confusion_matrix, roc_curve, auc, precision_recall_curve,
roc_auc_score, average_precision_score

#time and warnings
import time
import warnings

#settings
warnings.filterwarnings("ignore")
%matplotlib inline
sns.set_context('poster', font_scale=0.5)
```

2. Data

In [ ]:
```
#loading the dataset as a pandas dataframe
mtcars = pd.read_csv('mtcars.csv')
```

3. Initial Data Exploration

In [ ]:
```
#first 5 rows of the dataset
mtcars.head()
```

Out[ ]:

	Unnamed: 0	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

In [ ]:

```
#checking for missing values and data types
mtcars.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32 entries, 0 to 31
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Unnamed: 0  32 non-null     object 
 1   mpg         32 non-null     float64
 2   cyl         32 non-null     int64  
 3   disp        32 non-null     float64
 4   hp          32 non-null     int64  
 5   drat        32 non-null     float64
 6   wt          32 non-null     float64
 7   qsec        32 non-null     float64
 8   vs          32 non-null     int64  
 9   am          32 non-null     int64  
 10  gear        32 non-null     int64  
 11  carb        32 non-null     int64  
dtypes: float64(5), int64(6), object(1)
memory usage: 3.1+ KB
```

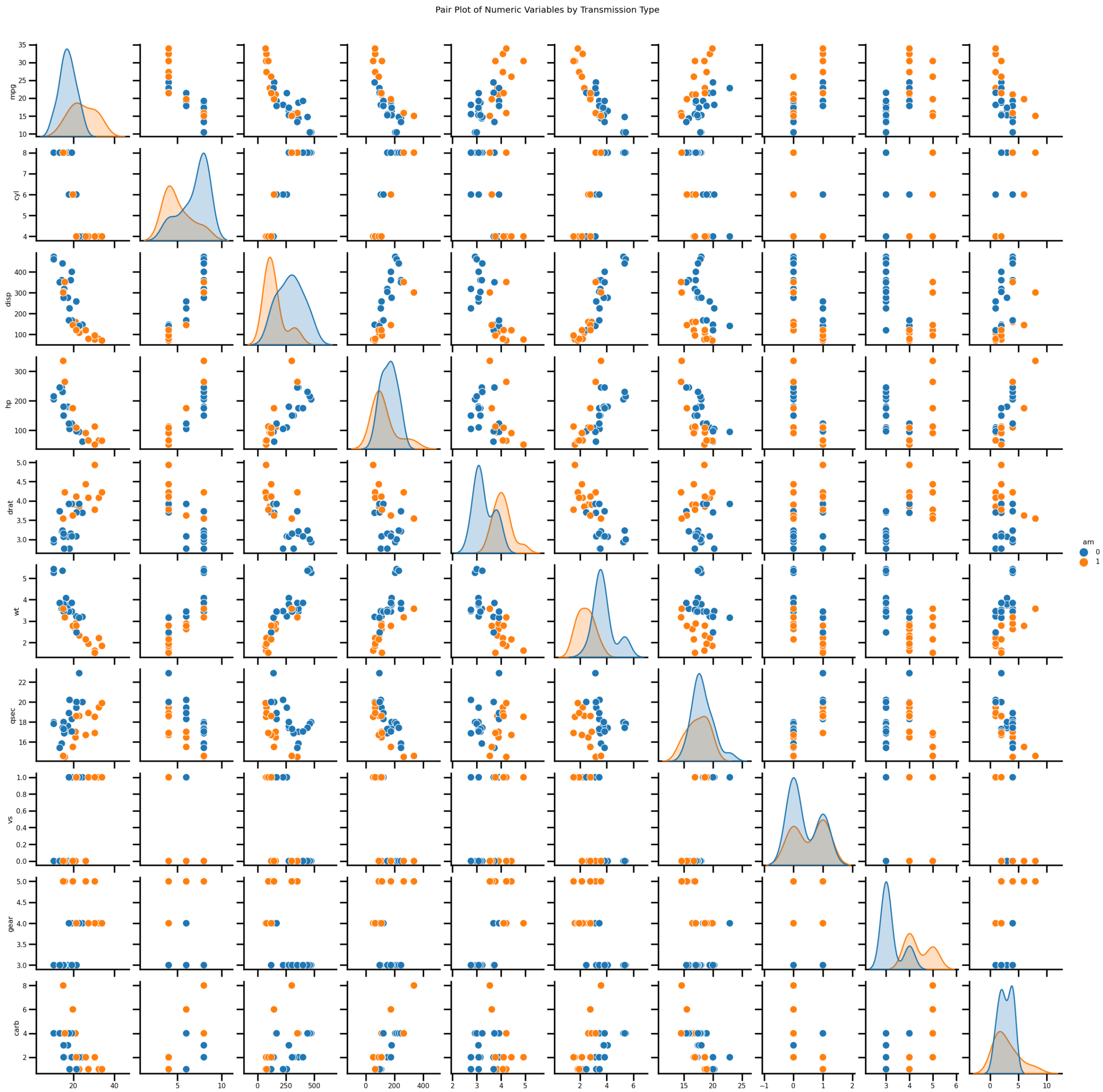
In [ ]:

```
#brief statistics of data
mtcars.describe()
```

Out[ ]:

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	
count	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32
mean	20.090625	6.187500	230.721875	146.687500	3.596563	3.217250	17.848750	0.437500	0.406250	3.687500	2
std	6.026948	1.785922	123.938694	68.562868	0.534679	0.978457	1.786943	0.504016	0.498991	0.737804	1
min	10.400000	4.000000	71.100000	52.000000	2.760000	1.513000	14.500000	0.000000	0.000000	3.000000	1
25%	15.425000	4.000000	120.825000	96.500000	3.080000	2.581250	16.892500	0.000000	0.000000	3.000000	2
50%	19.200000	6.000000	196.300000	123.000000	3.695000	3.325000	17.710000	0.000000	0.000000	4.000000	2
75%	22.800000	8.000000	326.000000	180.000000	3.920000	3.610000	18.900000	1.000000	1.000000	4.000000	4
max	33.900000	8.000000	472.000000	335.000000	4.930000	5.424000	22.900000	1.000000	1.000000	5.000000	8

In [ ]:
```
#pairplot
sns.pairplot(mtcars, hue='am')
plt.suptitle('Pair Plot of Numeric Variables by Transmission Type', y=1.02)
plt.show()
```



In [ ]:
```
#rare occasion of having a visual inspection of the entire dataset
mtcars
```

Out[ ]:

	Unnamed: 0	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
5	Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
6	Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
7	Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
8	Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
9	Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
10	Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
11	Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
12	Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
13	Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
14	Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
15	Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
16	Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
17	Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
18	Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
19	Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
20	Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
21	Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
22	AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
23	Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
24	Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
25	Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
26	Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
27	Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
28	Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
29	Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
30	Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
31	Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2

Initial Assumptions

- Data has only 32 points per variable
- The target variable is 'am'
- There are no missing values
- Visual inspection of the entire dataset showed no significant anomalies
- Visualizing numerical points with pairplot showed no significant anomalies and outliers

4. Data Cleaning

In [ ]:
```
#renaming the first column to car_brand
mtcars.rename(columns={'Unnamed: 0': 'car_brand'}, inplace=True)
```

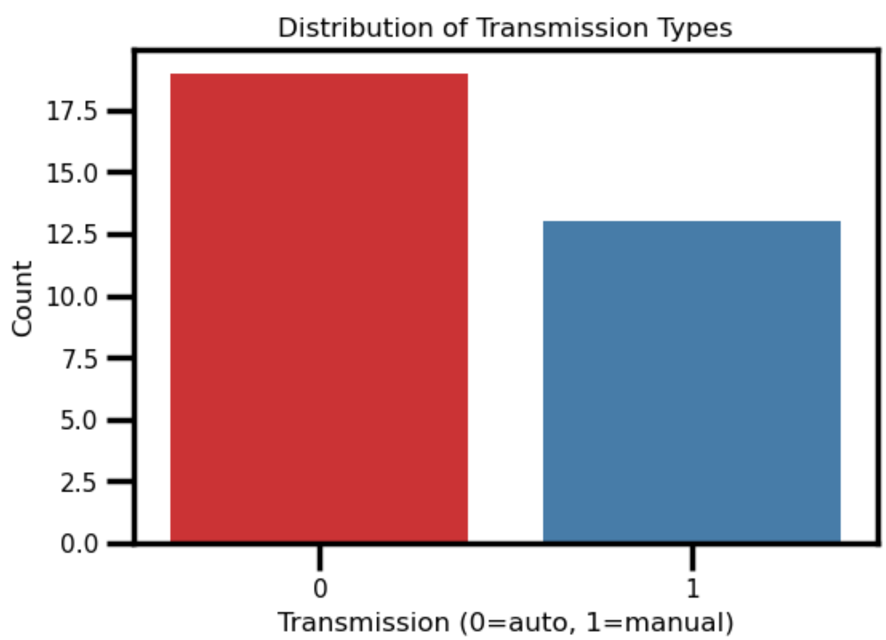
5. Data Visualizations

There are couple of relationships that I would like to explore in order to get a sense of the dataset, and base my modeling desicions on that.

Visualizations I want to explore are:

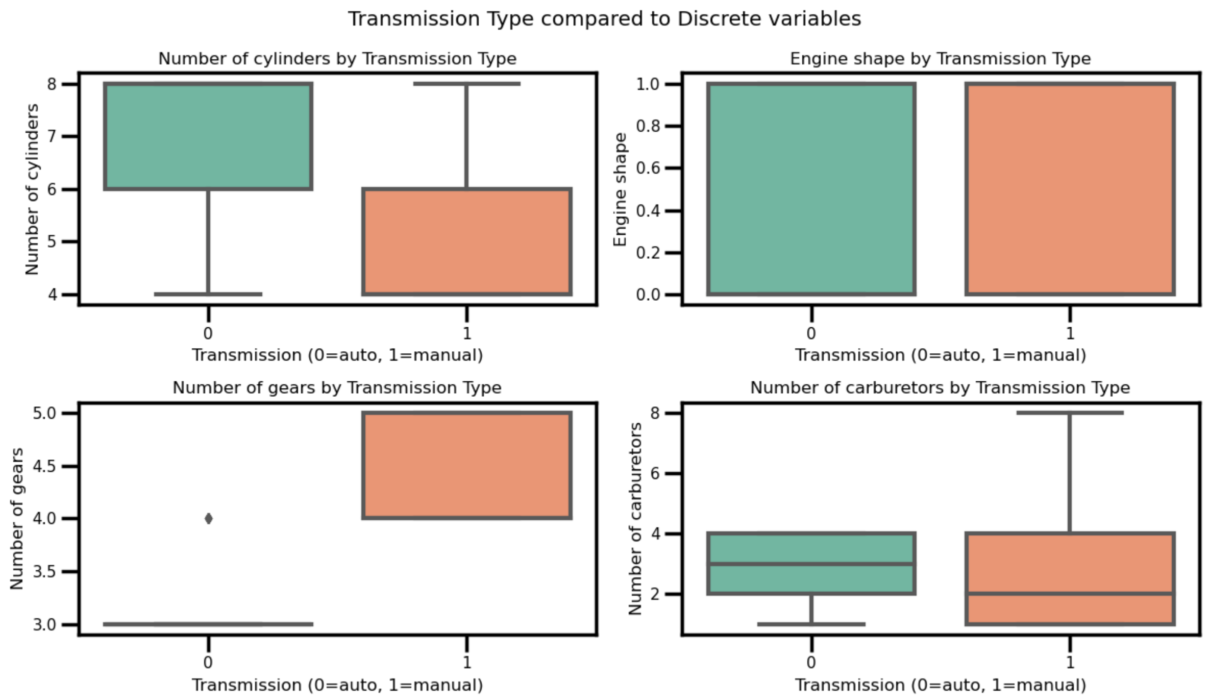
- Target variable 'am' count
- Target variable 'am' in comparison to dicrete variables: number of cylinders, engine shape, number of gears, and number of carburetors
- Target variable 'am' visualized in comparison to fuel efficiency and engine displacement as a scatter plot

In [ ]:
```
#transmission type count
plt.figure(figsize=(6, 4))
sns.countplot(data=mtcars, x='am', palette='Set1')

plt.xlabel('Transmission (0=auto, 1=manual)')
plt.ylabel('Count')
plt.title('Distribution of Transmission Types')
plt.show()
```

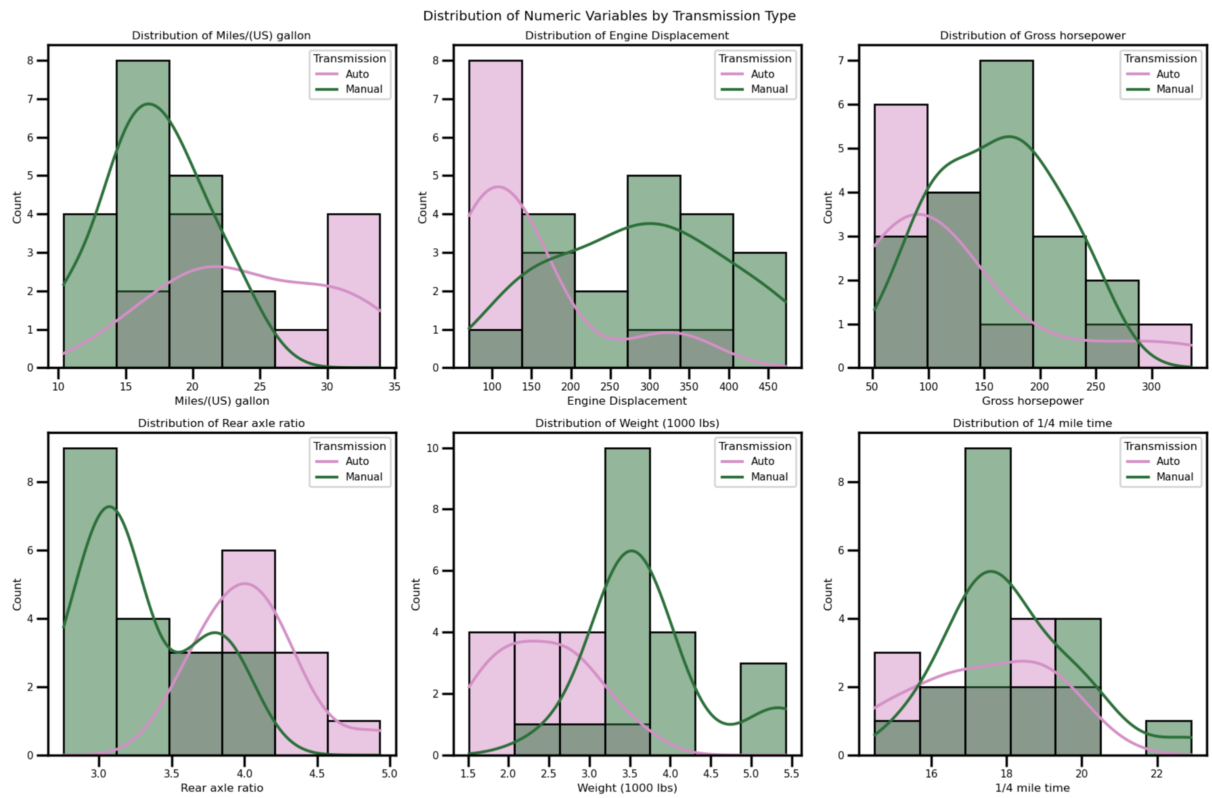


In [ ]:

```
#transmission type against discrete variables
#variables
boxplot_vars = ['cyl', 'vs', 'gear', 'carb']
boxplot_labels = ['Number of cylinders', 'Engine shape', 'Number of gears', 'Number of ca
rburetors']

#subplots
fig, axes = plt.subplots(2, 2, figsize=(12, 7))
fig.suptitle('Transmission Type compared to Discrete variables')

#creating the boxplot
for var, label, ax in zip(boxplot_vars, boxplot_labels, axes.flatten()):
    sns.boxplot(x='am', y=var, data=mtcars, ax=ax, palette='Set2')
    ax.set_xlabel('Transmission (0=auto, 1=manual)')
    ax.set_ylabel(label)
    ax.set_title(f'{label} by Transmission Type')

plt.tight_layout()
plt.show();
```

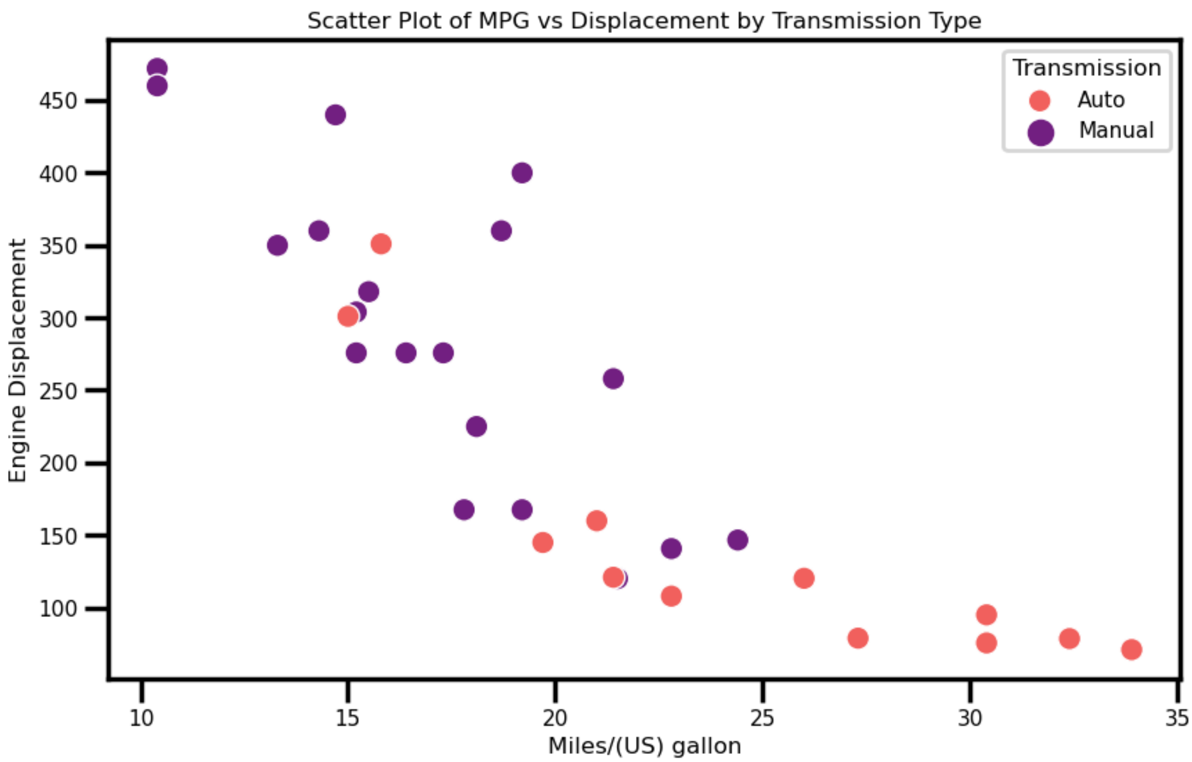


In [ ]:
```python
#transmission type against continuous variables
#creating subplots
fig, axes = plt.subplots(2, 3, figsize=(18, 12))
fig.suptitle('Distribution of Numeric Variables by Transmission Type')

#numeric variables
renamed_vars = {'mpg': 'Miles/(US) gallon',
                'disp': 'Engine Displacement',
                'hp': 'Gross horsepower',
                'drat': 'Rear axle ratio',
                'wt': 'Weight (1000 lbs)',
                'qsec': '1/4 mile time'}

#creating histograms
for var, label, ax in zip(renamed_vars.keys(), renamed_vars.values(), axes.flatten()):
    sns.histplot(data=mtcars, x=var, hue='am', kde=True, ax=ax, palette='cubehelix')
    ax.set_title(f'Distribution of {label}')
    ax.set_xlabel(label)
    ax.set_ylabel('Count')
    ax.legend(title='Transmission', labels=['Auto', 'Manual'])

plt.tight_layout()
plt.show()
```

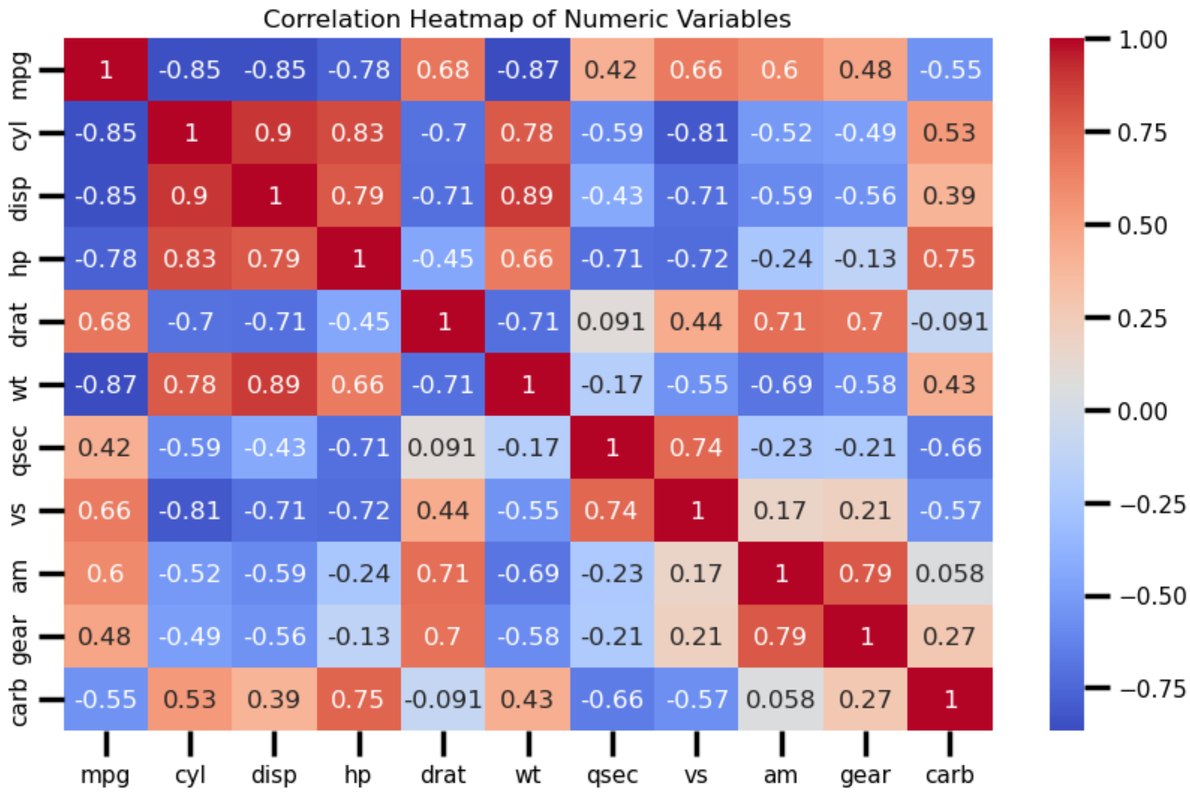


In [ ]:

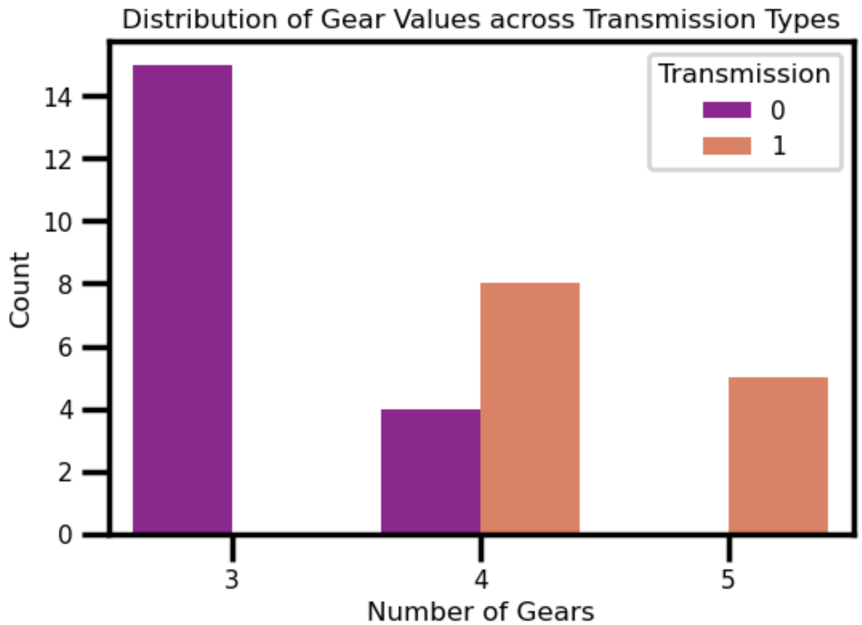
```
#transmission type in a scatter plot with milegae per gallon and engine displacement
plt.figure(figsize=(10, 6))
sns.scatterplot(data=mtcars, x='mpg', y='disp', hue='am', palette='magma')

plt.title('Scatter Plot of MPG vs Displacement by Transmission Type')
plt.xlabel('Miles/(US) gallon')
plt.ylabel('Engine Displacement')
plt.legend(title='Transmission', labels=['Auto', 'Manual'])
plt.show()
```



In [ ]:
```
#correlation heatmap
plt.figure(figsize=(10, 6))
correlation_matrix = mtcars.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')

plt.title('Correlation Heatmap of Numeric Variables')
plt.show()
```

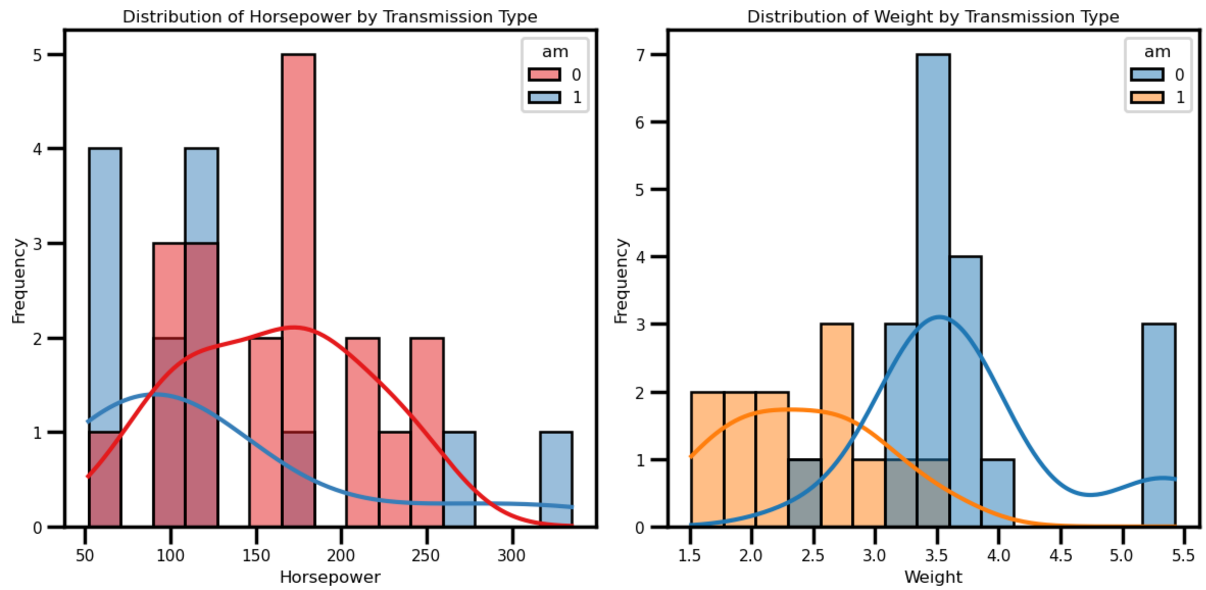


In [ ]:
```
# gear distribution across transmission types
plt.figure(figsize=(6, 4))
sns.countplot(data=mtcars, x='gear', hue='am', palette='plasma')

plt.xlabel('Number of Gears')
plt.ylabel('Count')
plt.title('Distribution of Gear Values across Transmission Types')
plt.legend(title='Transmission')
plt.show()
```



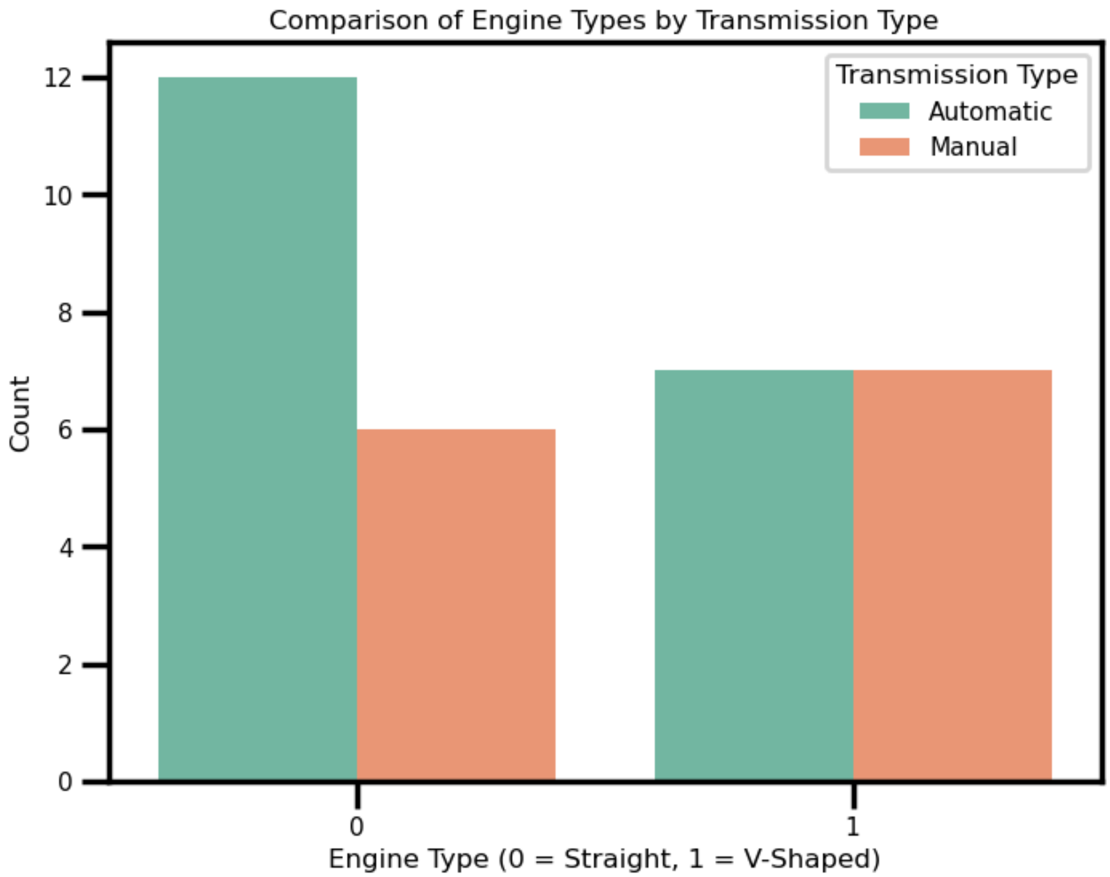
In [ ]:

```
#distribution of horsepower and vehicle weight by transmission type
plt.figure(figsize=(12, 6))

#horsepower histogram (with kde)
plt.subplot(1, 2, 1)
sns.histplot(data=mtcars, x='hp', hue='am', bins=15, kde=True, palette='Set1')
plt.xlabel('Horsepower')
plt.ylabel('Frequency')
plt.title('Distribution of Horsepower by Transmission Type')

#vehicle weight histogram (with kde)
plt.subplot(1, 2, 2)
sns.histplot(data=mtcars, x='wt', hue='am', bins=15, kde=True)
plt.xlabel('Weight')
plt.ylabel('Frequency')
plt.title('Distribution of Weight by Transmission Type')

plt.tight_layout()
plt.show()
```



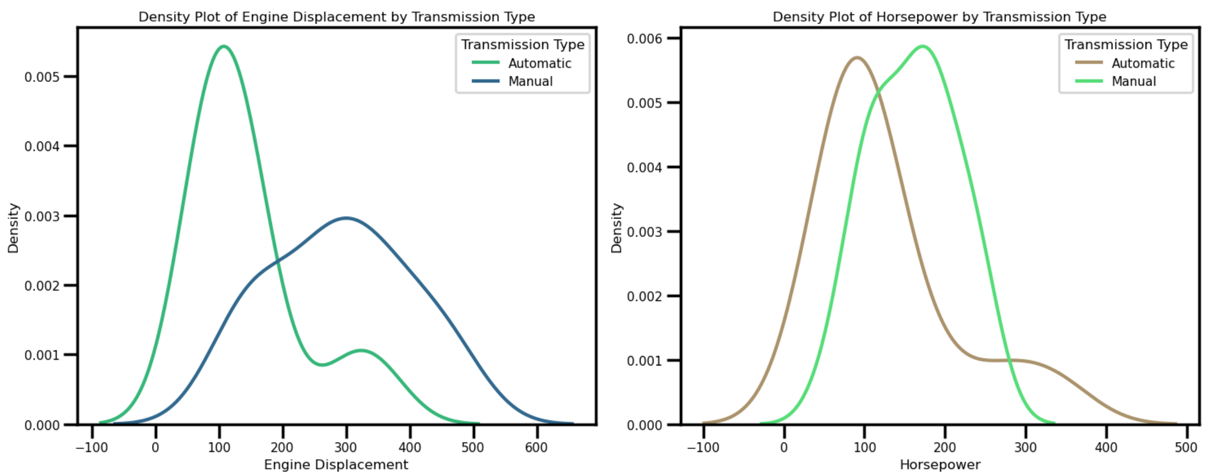
In [ ]:

```
#comparing engine types across transmission types
plt.figure(figsize=(8, 6))
sns.countplot(data=mtcars, x='vs', hue='am', palette='Set2')
plt.xlabel('Engine Type (0 = Straight, 1 = V-Shaped)')
plt.ylabel('Count')
plt.title('Comparison of Engine Types by Transmission Type')
plt.legend(title='Transmission Type', loc='upper right', labels=['Automatic', 'Manual'])
plt.show()
```

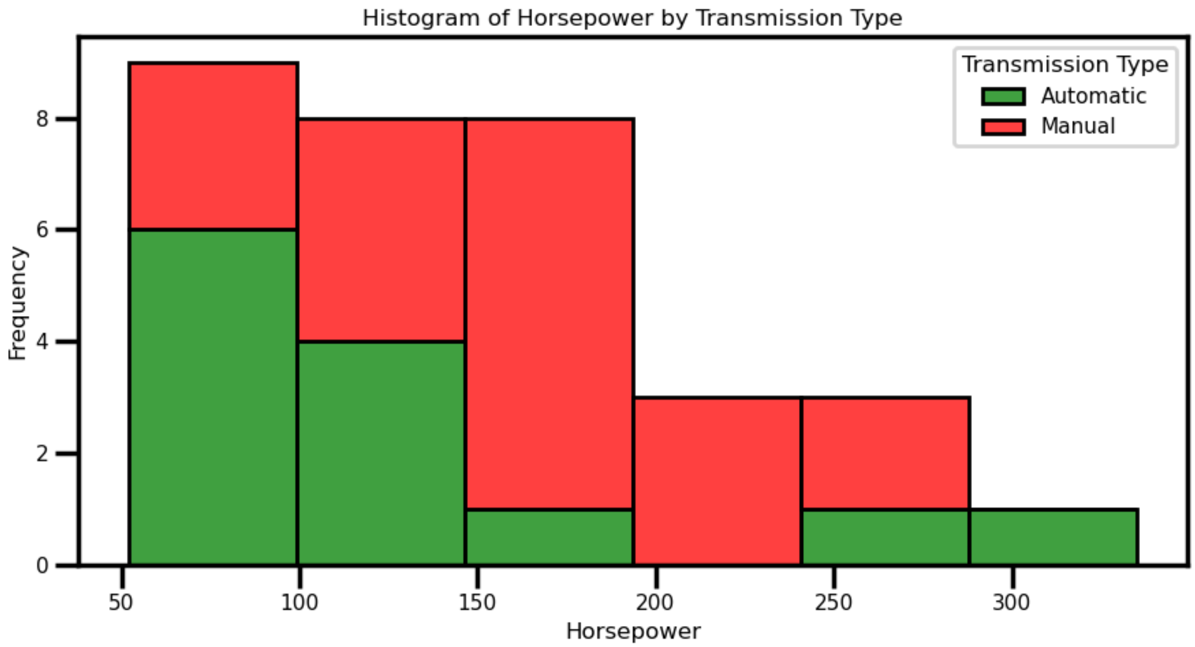


In [ ]:

```
#density plots for engine displacement and horsepower by transmission type
#subplots
fig, axes = plt.subplots(1, 2, figsize=(15, 6))

#engine displacement
sns.kdeplot(data=mtcars, x='disp', hue='am', palette='viridis', common_norm=False, ax=axe
s[0])
axes[0].set_xlabel('Engine Displacement')
axes[0].set_ylabel('Density')
axes[0].set_title('Density Plot of Engine Displacement by Transmission Type')
axes[0].legend(title='Transmission Type', labels=['Automatic', 'Manual'])

#horsepower
sns.kdeplot(data=mtcars, x='hp', hue='am', palette='terrain', common_norm=False, ax=axes
[1])
axes[1].set_xlabel('Horsepower')
axes[1].set_ylabel('Density')
axes[1].set_title('Density Plot of Horsepower by Transmission Type')
axes[1].legend(title='Transmission Type', labels=['Automatic', 'Manual'])


plt.tight_layout()
plt.show()
```



In [ ]:
```
#horsepower by transmission type

plt.figure(figsize=(9, 5))
palette_color = {0: 'red', 1: 'green'}
sns.histplot(data=mtcars, x='hp', hue='am', palette=palette_color, multiple='stack')
plt.xlabel('Horsepower')
plt.ylabel('Frequency')
plt.title('Histogram of Horsepower by Transmission Type')
plt.legend(title='Transmission Type', labels=['Automatic', 'Manual'])

plt.tight_layout()
plt.show()
```



6. Data Modeling

In [ ]:
```
#selecting features and target variable
X = mtcars[['mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'gear', 'carb']]
y = mtcars['am']

#splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [ ]:
```
#storing the original indices of the test set for later evaluation
test_indices = X_test.index
X_test_array = X_test.to_numpy()
```

In [ ]:
```
#scaling data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [ ]:
```
#checking the shape of the data
print(X_train.shape)
print(y_train.shape)
```

```
(25, 10)
(25,)
```

```
In [ ]: #initializing models for classification
models = [
    ('Logistic Regression', LogisticRegression()),
    ('Random Forest', RandomForestClassifier()),
    ('Decision Tree', DecisionTreeClassifier()),
    ('SVM', SVC()),
    ('KNN', KNeighborsClassifier()),
    ('Naive Bayes', GaussianNB()),
    ('Gradient Boosting', GradientBoostingClassifier()),
    ('Neural Network', MLPClassifier(max_iter=1000)),
    ('XGBoost', XGBClassifier())
]

#results dataframe
results_df = pd.DataFrame(columns=['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score
'])
```

```
In [ ]: #evaluation dict
scoring = {
    'Accuracy': make_scorer(accuracy_score),
    'Precision': make_scorer(precision_score),
    'Recall': make_scorer(recall_score),
    'F1 Score': make_scorer(f1_score)
}
```

```
In [ ]: #fitting and performing cross validation
for name, model in models:
    scores = cross_validate(model, X, y, cv=5, scoring=scoring)
    results_df = results_df.append({
        'Model': name,
        'Accuracy': scores['test_Accuracy'].mean(),
        'Precision': scores['test_Precision'].mean(),
        'Recall': scores['test_Recall'].mean(),
        'F1 Score': scores['test_F1 Score'].mean()
    }, ignore_index=True)

print(results_df)
```

```
                 Model  Accuracy  Precision    Recall  F1 Score
0  Logistic Regression  0.847619   0.850000  0.766667  0.771429
1        Random Forest  0.728571   0.785714  0.666667  0.653333
2        Decision Tree  0.700000   0.585714  0.600000  0.553333
3                  SVM  0.785714   0.785714  0.800000  0.753333
4                  KNN  0.695238   0.785714  0.600000  0.613333
5          Naive Bayes  0.785714   0.785714  0.800000  0.753333
6    Gradient Boosting  0.733333   0.619048  0.700000  0.613333
7       Neural Network  0.790476   0.700000  0.566667  0.600000
8              XGBoost  0.785714   0.785714  0.800000  0.753333
```

7. Model Evaluation

Logistic Regression

In [ ]:
```
#logistic regression instance
log_reg = LogisticRegression()

#parameters grid
param_grid = {
    'C': np.logspace(-4, 4, 9),
    'penalty': ['l1', 'l2', 'elasticnet', 'none'],
    'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
}

#searching for best parameters
grid_search = GridSearchCV(log_reg, param_grid, scoring='accuracy', cv=5)
grid_search.fit(X_train, y_train)

#tuned logistic regression
best_log_reg = grid_search.best_estimator_
```

In [ ]:
```
#checking the models parameters
best_log_reg
```

Out[ ]:
```
LogisticRegression(C=0.1, penalty='none', solver='sag')
```

In [ ]:
```
#selcting a single random sample index from the test set
random_sample_index = random.choice(test_indices)
sample_features_scaled = X_test[test_indices.get_loc(random_sample_index)].reshape(1, -1)
sample_actual_label = y_test.loc[random_sample_index]

#predicting the target variable for the single sample
sample_predicted_label = best_log_reg.predict(sample_features_scaled)

#results
print('Logistic Regression')
print(f'Randomly Selected Sample Index: {random_sample_index}')
print(f'Sample Features (scaled):\n{sample_features_scaled}')
print(f'Actual Label: {sample_actual_label}')
print(f'Predicted Label: {sample_predicted_label[0]}')
```

```
Logistic Regression
Randomly Selected Sample Index: 17
Sample Features (scaled):
[[ 2.10337761 -1.21233579 -1.26340899 -1.21810835  0.86474409 -0.98522213
   1.12305825  1.12815215  0.57735027 -1.29881326]]
Actual Label: 1
Predicted Label: 1
```

XGBoost

In [ ]:
```
#xgboost instance
xgb = XGBClassifier()

#parameter grid for gridsearch
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 4, 5],
    'learning_rate': [0.01, 0.1, 0.2]
}

grid_search = GridSearchCV(xgb, param_grid, scoring='accuracy', cv=5)
grid_search.fit(X_train, y_train)

#tuned xgb
best_xgb = grid_search.best_estimator_
```

In [ ]:
```
#xbg best params
best_xgb
```

Out[ ]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.01, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=3, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=50, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

In [ ]:
```
#selecting a single random sample index from the test set
random_sample_index = random.choice(test_indices)
sample_features_scaled = X_test[test_indices.get_loc(random_sample_index)].reshape(1, -1)
sample_actual_label = y_test.loc[random_sample_index]

#predicting the target variable for the single sample
sample_predicted_label = best_xgb.predict(sample_features_scaled)

#evaluating the results
print('XGBoost')
print(f'Randomly Selecteds Sample Index: {random_sample_index}')
print(f'Sample Features (scaled):\n{sample_features_scaled}')
print(f'Actual Label: {sample_actual_label}')
print(f'Predicted Label: {sample_predicted_label[0]}')
```

```
XGBoost
Randomly Selecteds Sample Index: 24
Sample Features (scaled):
[[-0.16645434  1.03273049  1.4628053   0.56648587 -0.926358    0.73503686
  -0.52489286 -0.88640526 -0.8660254  -0.44433085]]
Actual Label: 0
Predicted Label: 0
```

8. Future Model Improvements and Recommendations

In [ ]:
```
#predictions using the tuned XGBoost model
xgb_predictions = best_xgb.predict(X_test)

#predictions using the best logistic regression model
log_reg_predictions = best_log_reg.predict(X_test)

#classification report for XGBoost
print("XGBoost Classification Report:")
print(classification_report(y_test, xgb_predictions))

#classification report for Logistic Regression
print("Logistic Regression Classification Report:")
print(classification_report(y_test, log_reg_predictions))
```

```
XGBoost Classification Report:
              precision    recall  f1-score   support

           0       0.60      0.75      0.67         4
           1       0.50      0.33      0.40         3

    accuracy                           0.57         7
   macro avg       0.55      0.54      0.53         7
weighted avg       0.56      0.57      0.55         7

Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         4
           1       1.00      1.00      1.00         3

    accuracy                           1.00         7
   macro avg       1.00      1.00      1.00         7
weighted avg       1.00      1.00      1.00         7
```

In [ ]:
```
#initializing the log_reg_eval_model
log_reg_eval_model = LogisticRegression()

#cross-validation
cv_scores = cross_val_score(log_reg_eval_model, X_train, y_train, cv=5, scoring='accuracy
')

#fitting the model
log_reg_eval_model.fit(X_train, y_train)

#prediction
y_pred = log_reg_eval_model.predict(X_test)

#cross-validation scores
print("Cross-Validation Scores:", cv_scores)
print("Mean CV Score:", cv_scores.mean())
print("Standard Deviation of CV Scores:", cv_scores.std())

print()
#coefficients and intercept
coefficients = log_reg_eval_model.coef_[0]
intercept = log_reg_eval_model.intercept_[0]
print("Coefficients:", coefficients)
print("Intercept:", intercept)

print()
#classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# ROC AUC and Precision-Recall AUC
y_pred_prob = log_reg_eval_model.predict_proba(X_test)[:, 1]
roc_auc = roc_auc_score(y_test, y_pred_prob)
precision_recall_auc = average_precision_score(y_test, y_pred_prob)
print("ROC AUC:", roc_auc)
print("Precision-Recall AUC:", precision_recall_auc)
```

```
Cross-Validation Scores: [1.  1.  1.  0.8 1. ]
Mean CV Score: 0.96
Standard Deviation of CV Scores: 0.07999999999999999

Coefficients: [ 0.4925669  -0.37241137 -0.5292893  -0.12575401  0.81832278 -0.81099424
 -0.82452522 -0.30261624  0.91972965 -0.07023265]
Intercept: -0.9860865027948342

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         4
           1       1.00      1.00      1.00         3

    accuracy                           1.00         7
   macro avg       1.00      1.00      1.00         7
weighted avg       1.00      1.00      1.00         7

ROC AUC: 1.0
Precision-Recall AUC: 1.0
```