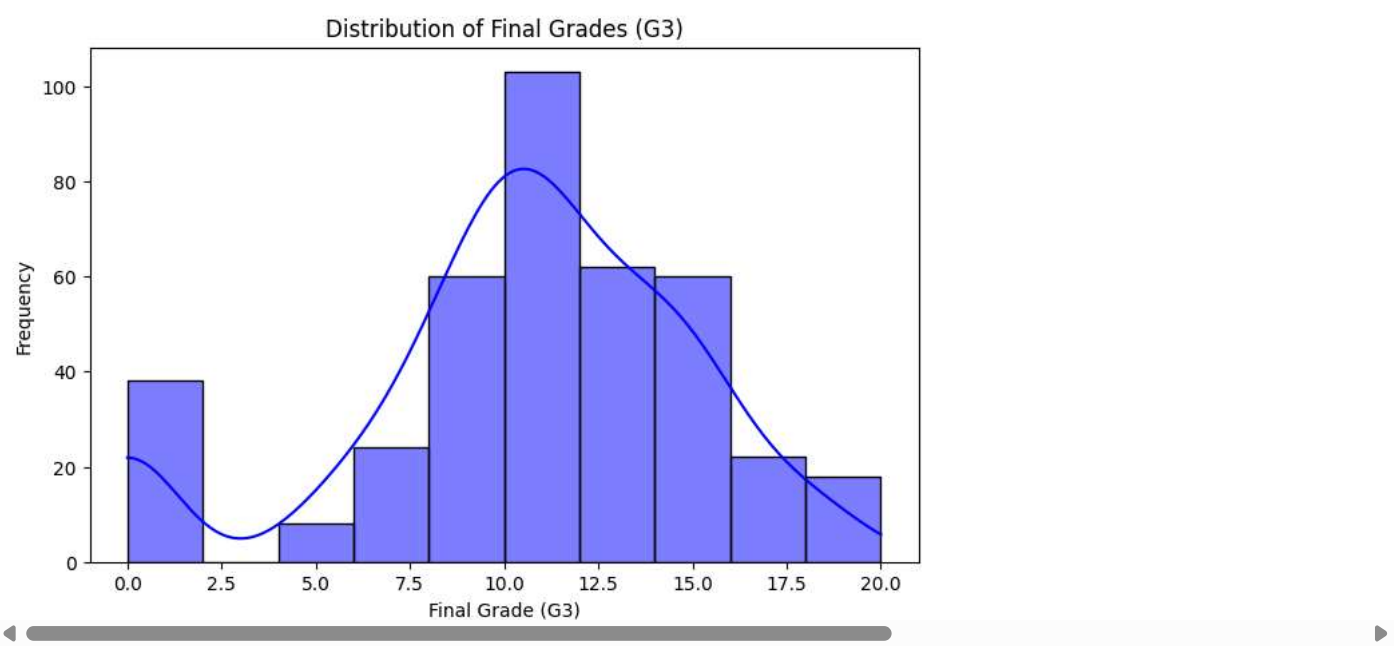
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score


df = pd.read_csv('data.csv')
df.drop_duplicates(inplace=True)
df.dropna()
```

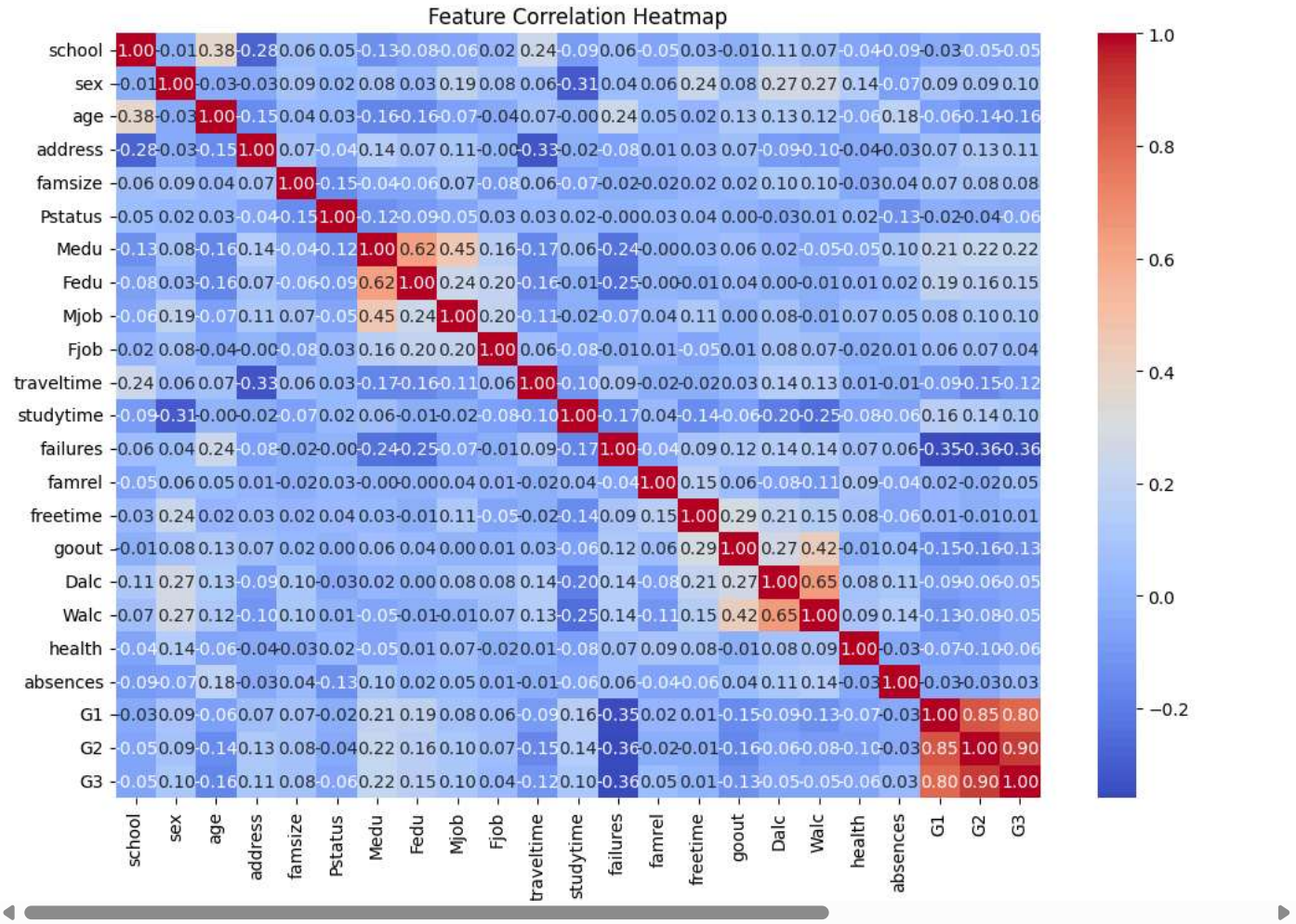
	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	freetime	goout	Dalc	Walc	health	absences
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4	3	4	1	1	3	6
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	5	3	3	1	1	3	4
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	4	3	2	2	3	3	10
3	GP	F	15	U	GT3	T	4	2	health	services	...	3	2	2	1	1	5	2
4	GP	F	16	U	GT3	T	3	3	other	other	...	4	3	2	1	2	5	4
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
390	MS	M	20	U	LE3	A	2	2	services	services	...	5	5	4	4	5	4	11
391	MS	M	17	U	LE3	T	3	1	services	services	...	2	4	5	3	4	2	3
392	MS	M	21	R	GT3	T	1	1	other	other	...	5	5	3	3	3	3	3
393	MS	M	18	R	LE3	T	3	2	services	other	...	4	4	1	3	4	5	0
394	MS	M	19	U	LE3	T	1	1	other	at_home	...	3	2	3	3	3	5	5

395 rows × 33 columns

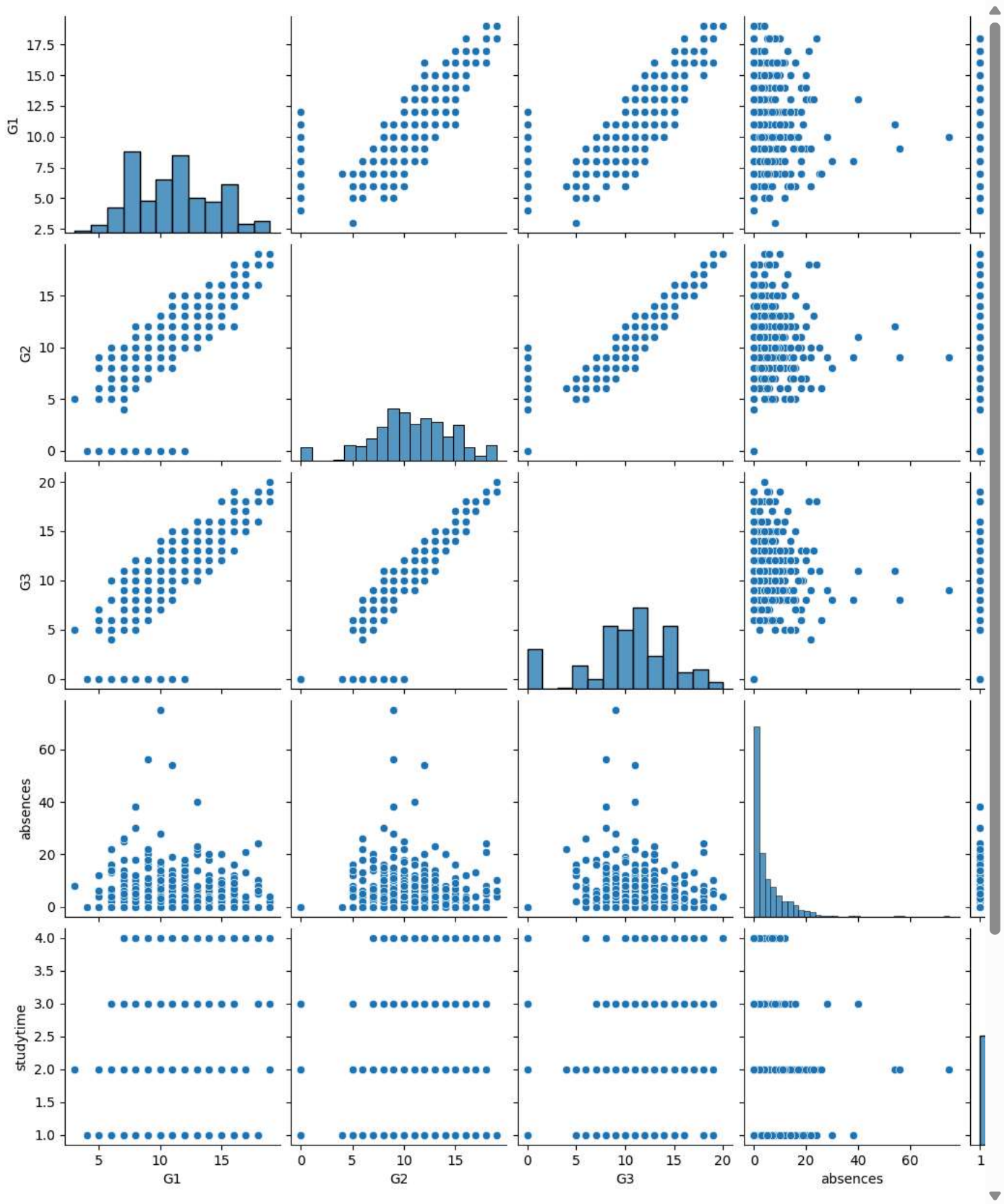
```
categorical_columns = ['school', 'sex', 'address', 'famsize', 'Pstatus', 'Mjob', 'Fjob']
numeric_columns = [col for col in df.columns if col not in categorical_columns + ['G3']]


label_encoders = {}
for col in categorical_columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le


label_encoders = {}
for col in categorical_columns:
    df[col] = df[col].astype(str)
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le



X = df.drop(columns=['G3'])
y = df['G3']


numeric_columns = X.select_dtypes(include=['int64', 'float64']).columns.tolist()
scaler = StandardScaler()
X[numeric_columns] = scaler.fit_transform(X[numeric_columns])



plt.figure(figsize=(8,5))
sns.histplot(df['G3'], bins=10, kde=True, color='blue')
plt.title('Distribution of Final Grades (G3)')
plt.xlabel('Final Grade (G3)')
plt.ylabel('Frequency')
plt.show()
```




```
df_numeric = df.select_dtypes(include=['int64', 'float64'])
plt.figure(figsize=(12,8))
sns.heatmap(df_numeric.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Feature Correlation Heatmap')
plt.show()
```



```
sns.pairplot(df[['G1', 'G2', 'G3', 'absences', 'studytime']])
plt.show()
```




```
X_train = X_train.apply(pd.to_numeric, errors='coerce')
X_test = X_test.apply(pd.to_numeric, errors='coerce')
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
```

```
from xgboost import XGBRegressor


models = {
    'Linear Regression': LinearRegression(),
    'Random Forest': RandomForestRegressor(n_estimators=100, random_state=42),
    'Support Vector Regressor': SVR(kernel='rbf'),
    'XGBoost': XGBRegressor(objective='reg:squarederror', random_state=42)
}


from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline


X = df.drop(columns=['G3'])
y = df['G3']
numeric_features = X.select_dtypes(include=np.number).columns
categorical_features = X.select_dtypes(include=['object']).columns

numeric_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler()),
])

categorical_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(sparse_output=False, handle_unknown='ignore')),
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_pipeline, numeric_features),
        ('cat', categorical_pipeline, categorical_features),
    ]
)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


X_train = preprocessor.fit_transform(X_train)
X_test = preprocessor.transform(X_test)



from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score, accuracy_score, precision_score
results = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    accuracy = accuracy_score(y_test.round(), y_pred.round()) if name != 'Support Vector Regressor' else None
    precision = precision_score(y_test.round(), y_pred.round(), average='weighted', zero_division=1) if name != 'Support Vector Regressor' e
    results[name] = {'MAE': mae, 'MSE': mse, 'R2 Score': r2, 'Accuracy': accuracy, 'Precision': precision}


for name, metrics in results.items():
    print(f"Model: {name}")
    for metric, value in metrics.items():
        if value is not None:
            print(f" {metric}: {value:.4f}")
    print("\n")
```

```
 Model: Linear Regression
  MAE: 1.5634
  MSE: 5.3530
  R2 Score: 0.7389
  Accuracy: 0.2025
  Precision: 0.3112


 Model: Random Forest
  MAE: 1.1549
  MSE: 3.7844
  R2 Score: 0.8154
  Accuracy: 0.4051
  Precision: 0.5311
```

```
Model: Support Vector Regressor
 MAE: 1.5915
 MSE: 6.1411
 R2 Score: 0.7005


Model: XGBoost
 MAE: 1.1783
 MSE: 4.6592
 R2 Score: 0.7728
 Accuracy: 0.4177
 Precision: 0.5388
```

```
results_df = pd.DataFrame(results).T
print(results_df)

results_df[['MAE', 'MSE', 'R2 Score', 'Accuracy', 'Precision']].plot(kind='bar', figsize=(12,6))
plt.title('Model Performance Comparison')
plt.xlabel('Model')
plt.ylabel('Error Metrics')
plt.xticks(rotation=45)
plt.legend()
plt.show()
```

```
                            MAE       MSE  R2 Score  Accuracy  Precision
Linear Regression      1.563443  5.352984  0.738943  0.202532   0.311181
Random Forest          1.154937  3.784367  0.815442  0.405063   0.531133
Support Vector Regressor  1.591549  6.141110  0.700507       NaN        NaN
XGBoost                1.178280  4.659227  0.772777  0.417722   0.538765
```