1) Load dataset

```
# Read the dataset into a pandas DataFrame
import pandas as pd
import os

# Update this path if needed (using raw strings to handle backslashes correctly)
if os.path.exists(r'/content/student-por.csv'):
    df = pd.read_csv(r'/content/student-por.csv', sep=';')
elif os.path.exists(r'/content/student-mat.csv'):
    df = pd.read_csv(r'/content/student-mat.csv', sep=';')
else:
    # Try to find any csv in working dir
    csvs = [f for f in os.listdir('.') if f.lower().endswith('.csv')]
    if csvs:
        df = pd.read_csv(csvs[0], sep=';')
    else:
        df = pd.DataFrame()
        print('No CSV found in working directory. Please upload or set the correct path.')

print('Data shape:', df.shape)
df.head()
```

```
Data shape: (649, 33)
```

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	freetime	goout
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4	3	4
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	5	3	3
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	4	3	2
3	GP	F	15	U	GT3	T	4	2	health	services	...	3	2	2
4	GP	F	16	U	GT3	T	3	3	other	other	...	4	3	2

5 rows × 33 columns

2) Exploratory Data Analysis (EDA)

Inspect distributions, missing values, correlations, and class balance.

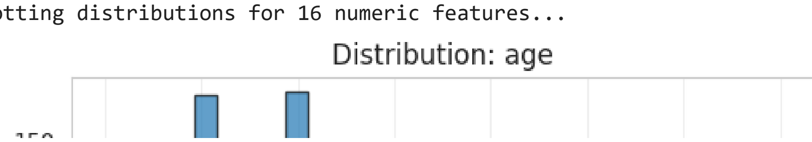
```
# Basic EDA
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

# Set visualization style
sns.set_style('whitegrid')
plt.rcParams['figure.dpi'] = 100

print('\n--- Dataset Info ---')
display(df.info())

print('\n--- Descriptive Statistics (numeric) ---')
display(df.describe())

print('\n--- Missing Values ---')
```

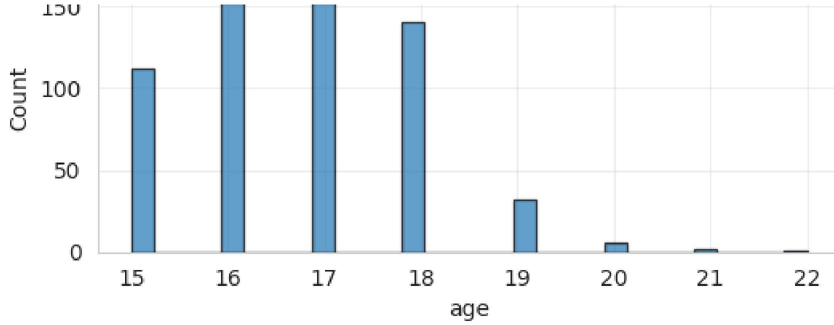
```
missing = df.isnull().sum()
if missing.sum() > 0:
    display(missing[missing > 0])
else:
    print('No missing values found!')

# Plot distributions for numeric features (one chart per numeric column)
num_cols = df.select_dtypes(include=[np.number]).columns.tolist()
print(f'\nPlotting distributions for {len(num_cols)} numeric features...')
for col in num_cols[:15]:  # Limit to first 15 to avoid too many plots
    plt.figure(figsize=(6,2.5))
    plt.hist(df[col].dropna(), bins=30, edgecolor='black', alpha=0.7)
    plt.title(f'Distribution: {col}')
    plt.xlabel(col)
    plt.ylabel('Count')
    plt.grid(True, alpha=0.3)
    plt.show()

# Plot categorical value counts for top categorical columns
cat_cols = df.select_dtypes(include=['object','category']).columns.tolist()
print(f'\nPlotting value counts for {min(len(cat_cols), 6)} categorical features...')
for col in cat_cols[:6]:
    plt.figure(figsize=(6,2.5))
    df[col].value_counts().plot(kind='bar', edgecolor='black', alpha=0.7)
    plt.title(f'Value Counts: {col}')
    plt.xlabel(col)
    plt.ylabel('Count')
    plt.xticks(rotation=45, ha='right')
    plt.grid(True, alpha=0.3, axis='y')
    plt.tight_layout()
    plt.show()
```

```
--- Dataset Info ---
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 649 entries, 0 to 648
Data columns (total 33 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   school      649 non-null    object
 1   sex         649 non-null    object
 2   age         649 non-null    int64 
 3   address     649 non-null    object
 4   famsize     649 non-null    object
 5   Pstatus     649 non-null    object
 6   Medu        649 non-null    int64 
 7   Fedu        649 non-null    int64 
 8   Mjob        649 non-null    object
 9   Fjob        649 non-null    object
 10  reason      649 non-null    object
 11  guardian    649 non-null    object
 12  traveltime  649 non-null    int64 
 13  studytime   649 non-null    int64 
 14  failures    649 non-null    int64 
 15  schoolsup   649 non-null    object
 16  famsup      649 non-null    object
 17  paid        649 non-null    object
 18  activities  649 non-null    object
 19  nursery     649 non-null    object
 20  higher      649 non-null    object
 21  internet    649 non-null    object
 22  romantic    649 non-null    object
 23  famrel      649 non-null    int64 
 24  freetime    649 non-null    int64 
 25  goout       649 non-null    int64 
 26  Dalc        649 non-null    int64 
 27  Walc        649 non-null    int64 
 28  health      649 non-null    int64 
 29  absences    649 non-null    int64 
 30  G1          649 non-null    int64 
 31  G2          649 non-null    int64 
 32  G3          649 non-null    int64 
dtypes: int64(16), object(17)
memory usage: 167.4+ KB
None

--- Descriptive Statistics (numeric) ---
```

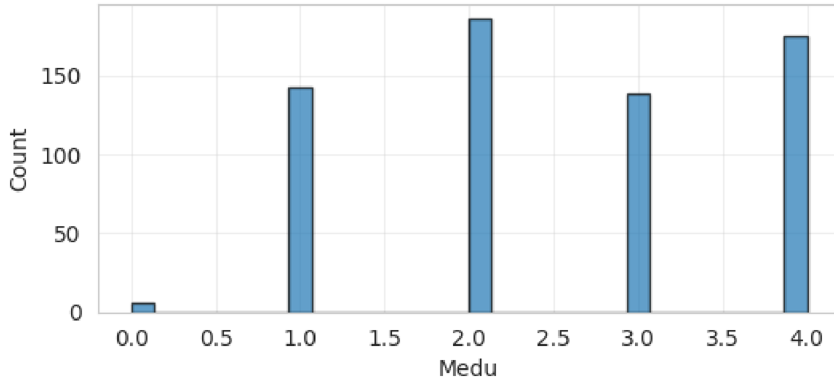
	age	Medu	Fedu	traveltime	studytime	failures	famrel	freetime	
count	649.000000	649.000000	649.000000	649.000000	649.000000	649.000000	649.000000	649.000000	649.0
mean	16.744222	2.514638	2.306626	1.568567	1.930663	0.221880	3.930663	3.180277	3.1
std	1.218138	1.134552	1.099931	0.748660	0.829510	0.593235	0.955717	1.051093	1.1
min	15.000000	0.000000	0.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.0
25%	16.000000	2.000000	1.000000	1.000000	1.000000	0.000000	4.000000	3.000000	2.0
50%	17.000000	2.000000	2.000000	1.000000	2.000000	0.000000	4.000000	3.000000	3.0
75%	18.000000	4.000000	3.000000	2.000000	2.000000	0.000000	5.000000	4.000000	4.0
max	22.000000	4.000000	4.000000	4.000000	4.000000	3.000000	5.000000	5.000000	5.0

```
--- Missing Values ---
No missing values found!

Plotting distributions for 16 numeric features...
```

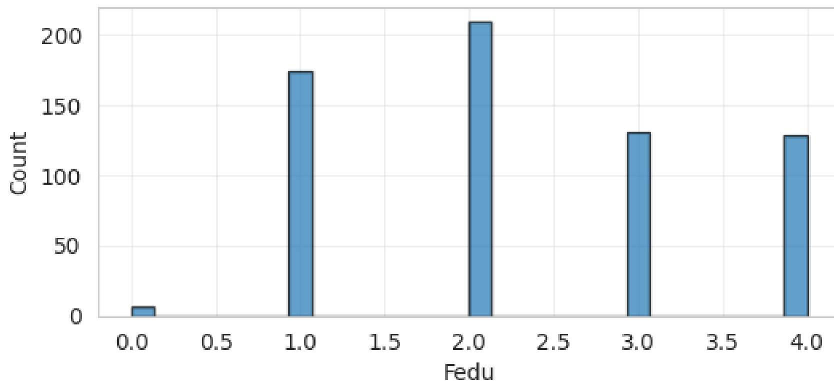




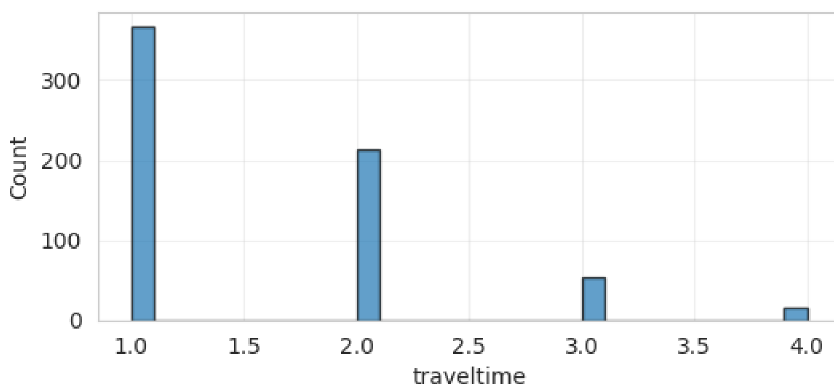
Distribution: Medu



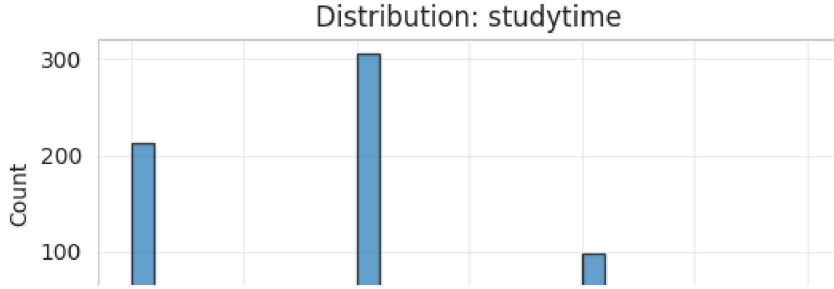
Distribution: Fedu

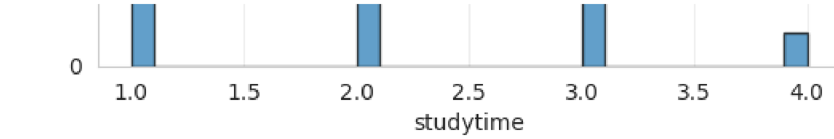


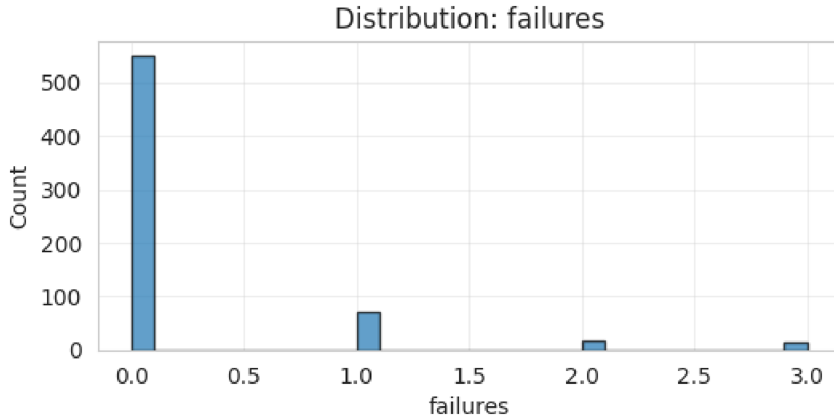
Distribution: traveltime

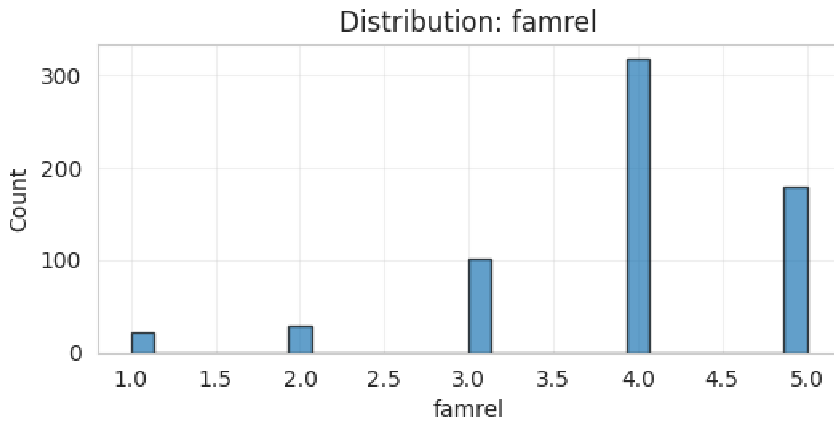


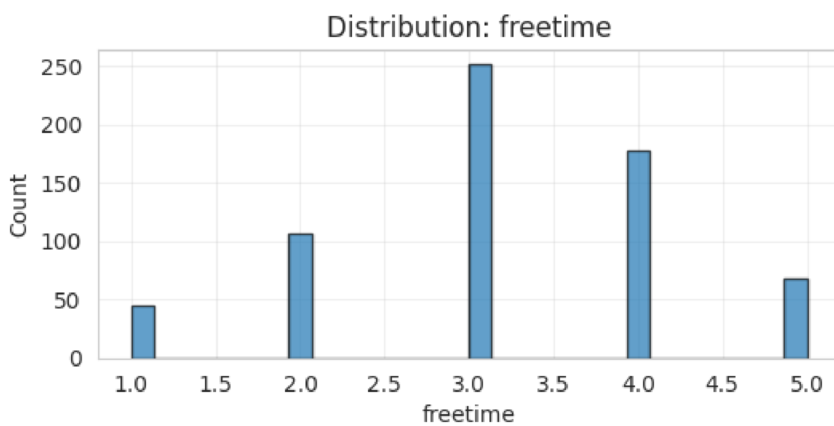
Distribution: studytime

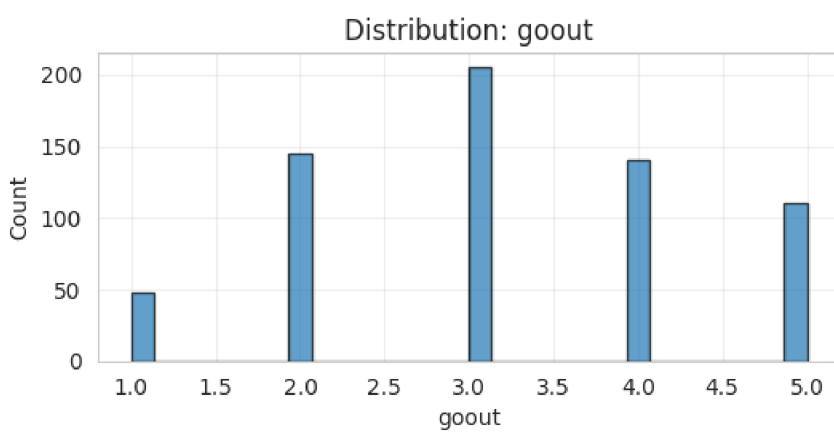




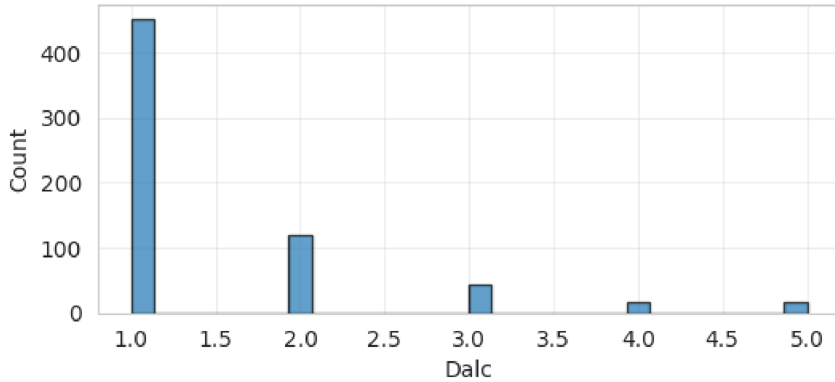




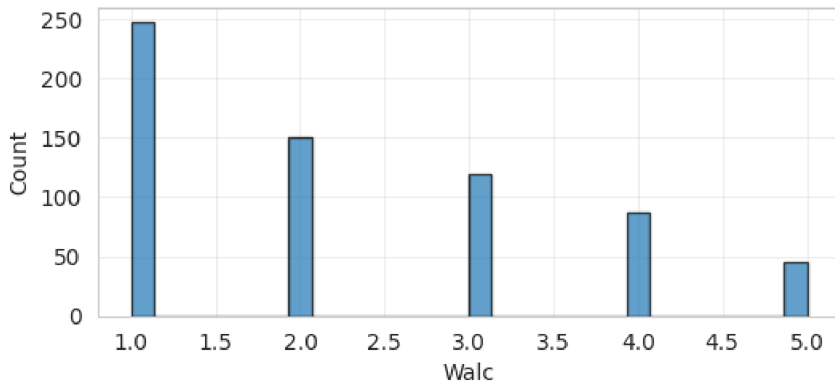




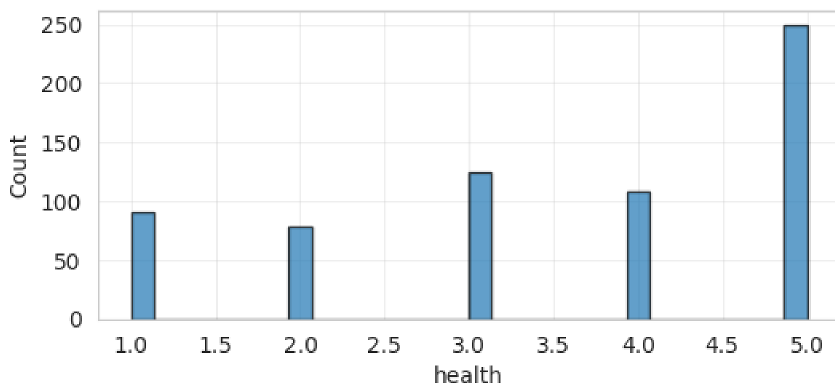
Distribution: Dalc



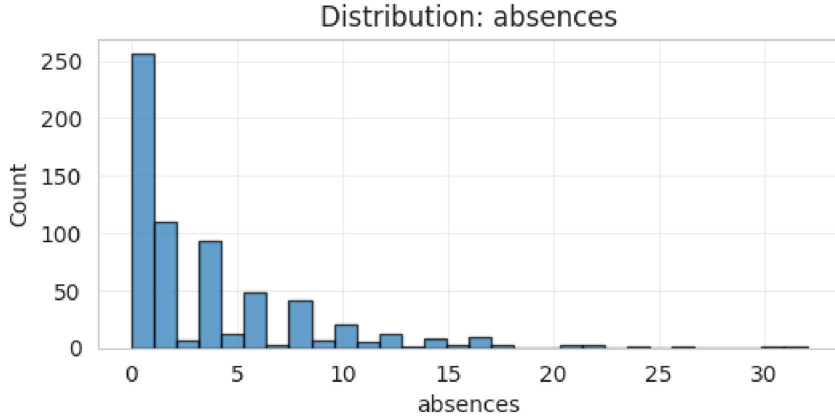
Distribution: Walc



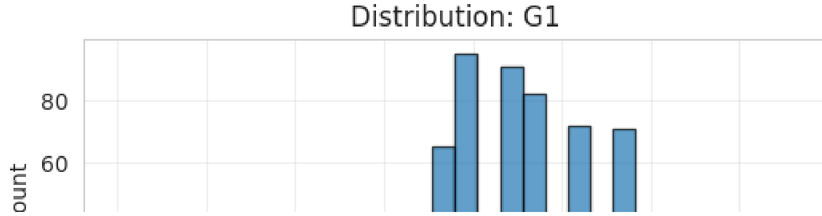
Distribution: health

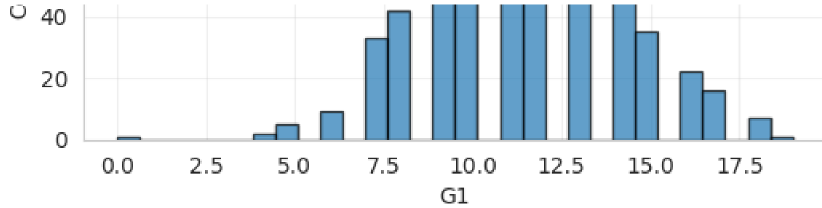


Distribution: absences

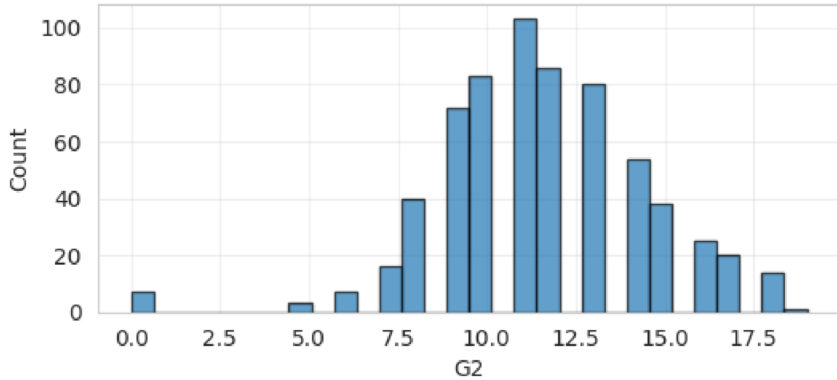


Distribution: G1



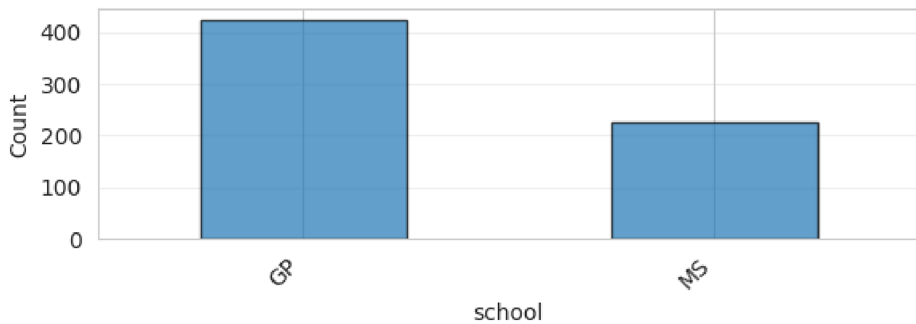


Distribution: G2

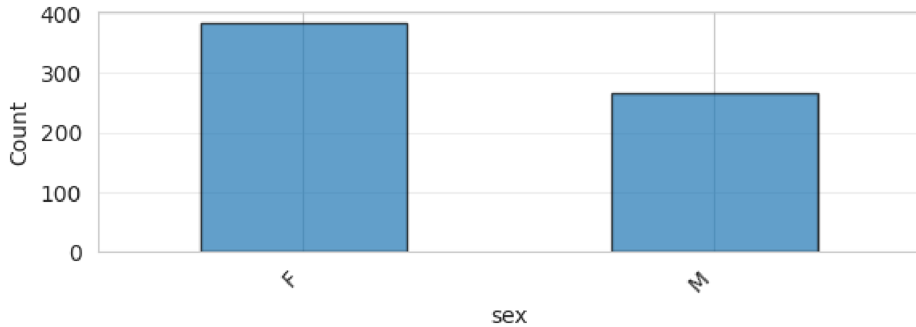



```
Plotting value counts for 6 categorical features...
```

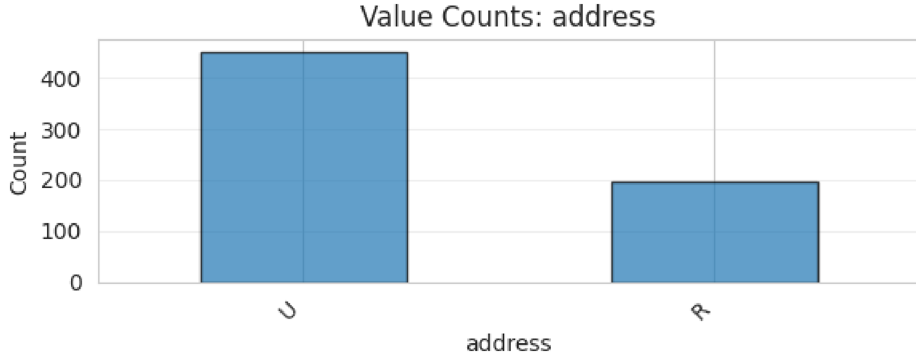
Value Counts: school



Value Counts: sex

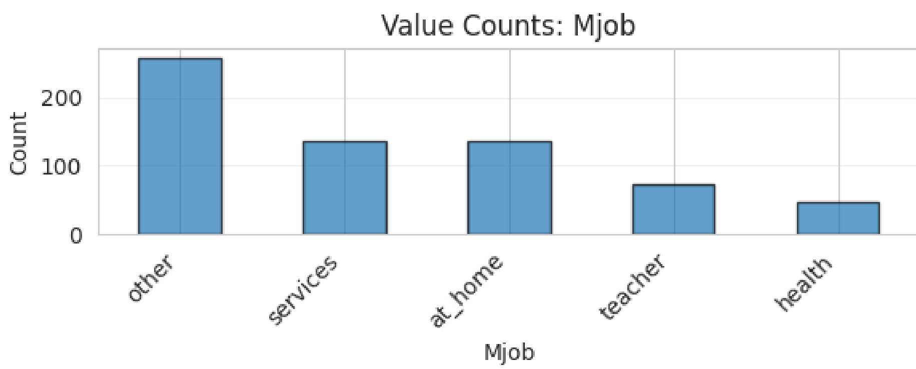


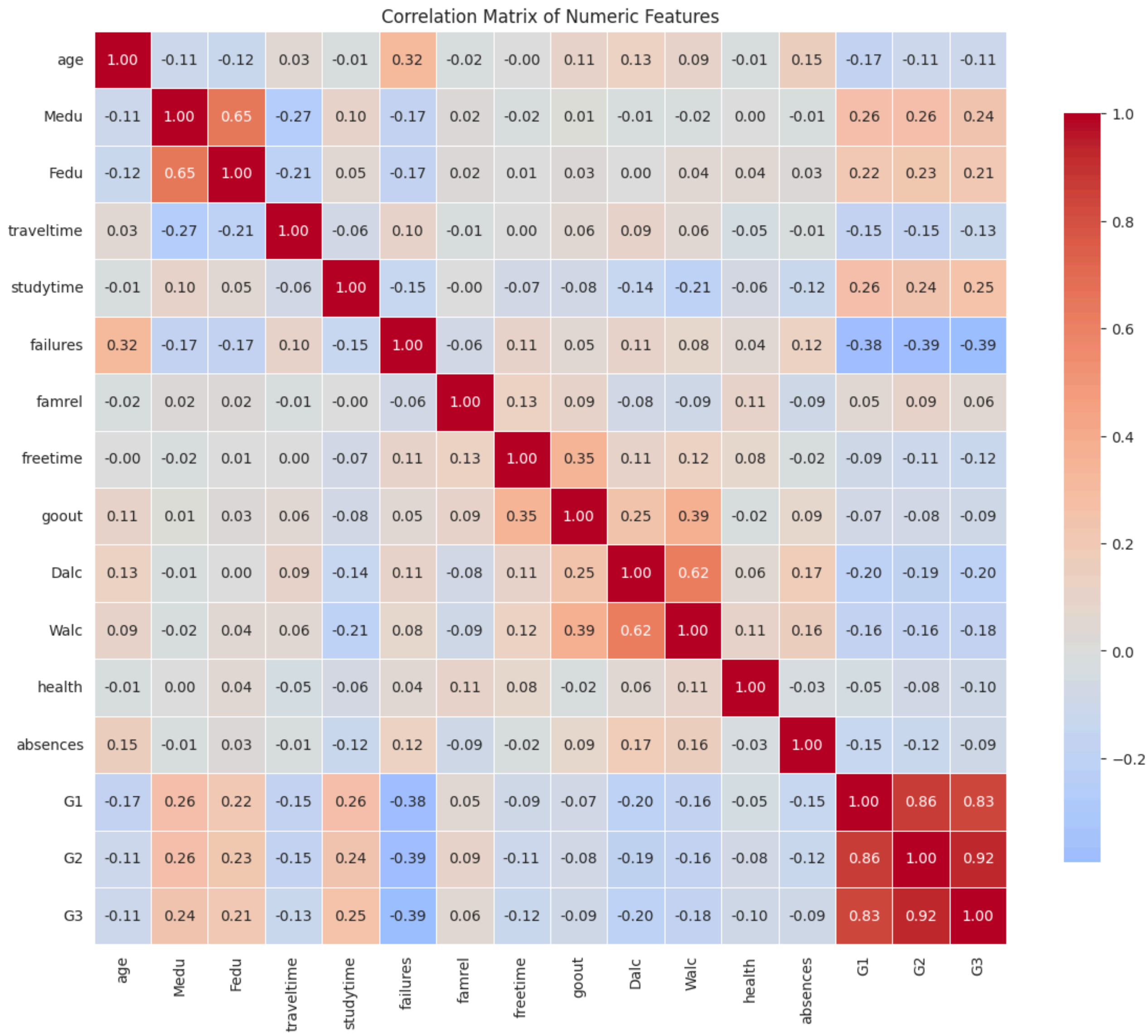
Value Counts: address



Value Counts: famsize

```python
# Correlation analysis for numeric features
numeric_df = df.select_dtypes(include=[np.number])
if len(numeric_df.columns) > 1:
    plt.figure(figsize=(12, 10))
    correlation_matrix = numeric_df.corr()
    sns.heatmap(correlation_matrix, annot=True, fmt='.2f', cmap='coolwarm', center=0,
                square=True, linewidths=0.5, cbar_kws={"shrink": 0.8})
    plt.title('Correlation Matrix of Numeric Features')
    plt.tight_layout()
    plt.show()

    # Show features most correlated with G3 (final grade)
    if 'G3' in correlation_matrix.columns:
        print('\nFeatures most correlated with Final Grade (G3):')
        g3_corr = correlation_matrix['G3'].sort_values(ascending=False)
        print(g3_corr.to_string())
else:
    print('Not enough numeric columns for correlation analysis.')
```

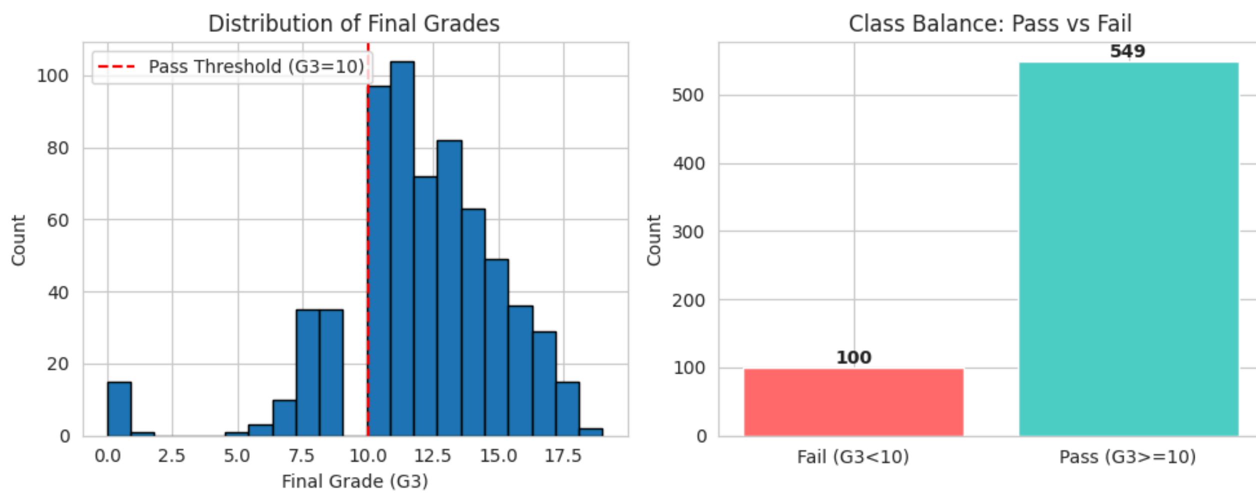





```
Features most correlated with Final Grade (G3):
G3            1.000000
G2            0.918548
G1            0.826387
studytime     0.249789
Medu          0.240151
Fedu          0.211800
famrel        0.063361
goout        -0.087641
absences     -0.091379
health       -0.098851
age          -0.106505
freetime     -0.122705
traveltime   -0.127173
Walc         -0.176619
Dalc         -0.204719
failures     -0.393316
```

```
# Check class balance for target variable (G3 grades)
if 'G3' in df.columns:
    # Show distribution of final grades
    plt.figure(figsize=(10, 4))

    plt.subplot(1, 2, 1)
```

```
    plt.hist(df['G3'], bins=21, edgecolor='black')
    plt.xlabel('Final Grade (G3)')
    plt.ylabel('Count')
    plt.title('Distribution of Final Grades')
    plt.axvline(x=10, color='r', linestyle='--', label='Pass Threshold (G3=10)')
    plt.legend()

    plt.subplot(1, 2, 2)
    pass_count = (df['G3'] >= 10).sum()
    fail_count = (df['G3'] < 10).sum()
    plt.bar(['Fail (G3<10)', 'Pass (G3>=10)'], [fail_count, pass_count], color=['#ff6b6b', '#4ecdc4'])
    plt.ylabel('Count')
    plt.title('Class Balance: Pass vs Fail')
    for i, v in enumerate([fail_count, pass_count]):
        plt.text(i, v + 5, str(v), ha='center', fontweight='bold')

    plt.tight_layout()
    plt.show()

    print(f'Pass rate: {pass_count}/{len(df)} = {pass_count/len(df)*100:.1f}%')
    print(f'Fail rate: {fail_count}/{len(df)} = {fail_count/len(df)*100:.1f}%')
```




```
Pass rate: 549/649 = 84.6%
Fail rate: 100/649 = 15.4%
```

3) Data Cleaning & Preprocessing

Handle missing values, encode categorical variables, and create target variable.

Example target: predict final grade pass/fail (G3 >= 10 -> pass). Adjust to your project's objective.

```
# Example preprocessing pipeline
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder
from sklearn.impute import SimpleImputer

# Make a copy
data = df.copy()

# Example: create binary target 'pass' from final grade column G3 if present
if 'G3' in data.columns:
    data['pass'] = (data['G3'] >= 10).astype(int)
    target_col = 'pass'
```

```
else:
    # If no numeric grade present, user should set target manually
    print('No G3 column found. Please define your target_col manually.')
    target_col = None

# Drop columns unlikely to be helpful (example: drop G1 and G2 to avoid leakage)
# G1 and G2 are period grades that directly predict G3, so excluding them makes the problem more realis
drop_cols = ['G1', 'G2']
for c in drop_cols:
    if c in data.columns:
        data.drop(columns=c, inplace=True)

# Separate features and target
if target_col:
    X = data.drop(columns=[target_col])
    y = data[target_col]
else:
    X = data.copy()
    y = None

# Simple handling: numeric columns fillna with median, categorical with mode, one-hot encode categorica
num_cols = X.select_dtypes(include=[np.number]).columns.tolist()
cat_cols = X.select_dtypes(include=['object','category']).columns.tolist()

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

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

preproc = ColumnTransformer([('num', num_pipe, num_cols),
                             ('cat', cat_pipe, cat_cols)], remainder='drop')

# Note: For supervised learning, preproc will be fit on training data only (see Cell 11)
# For unsupervised learning (clustering), we'll fit on the full dataset
print('Feature columns identified:')
print(f'  - Numeric: {len(num_cols)} columns')
print(f'  - Categorical: {len(cat_cols)} columns')
```

```
Feature columns identified:
  - Numeric: 14 columns
  - Categorical: 17 columns
```

```
# Extract column names after OneHotEncoder for interpretability
# Fit preproc on full X temporarily just to get feature names
try:
    preproc_temp = ColumnTransformer([('num', num_pipe, num_cols),
                                      ('cat', cat_pipe, cat_cols)], remainder='drop')
    preproc_temp.fit(X)

    ohe = None
    for name, trans, cols in preproc_temp.transformers_:
        if name == 'cat':
            ohe = trans.named_steps['onehot']
            cat_in_cols = cols
    feature_names = []
    # numeric names
    feature_names.extend(num_cols)
    # onehot names
    if ohe is not None:
        ohe_names = ohe.get_feature_names_out(cat_in_cols)
        feature_names.extend(list(ohe_names))
    print('Number of features after preprocessing:', len(feature_names))
```

```
except Exception as e:
    feature_names = None
    print('Could not extract feature names automatically.', e)

feature_names[:50] if feature_names else None
```

```
Number of features after preprocessing: 57
['age',
 'Medu',
 'Fedu',
 'traveltime',
 'studytime',
 'failures',
 'famrel',
 'freetime',
 'goout',
 'Dalc',
 'Walc',
 'health',
 'absences',
 'G3',
 'school_GP',
 'school_MS',
 'sex_F',
 'sex_M',
 'address_R',
 'address_U',
 'famsize_GT3',
 'famsize_LE3',
 'Pstatus_A',
 'Pstatus_T',
 'Mjob_at_home',
 'Mjob_health',
 'Mjob_other',
 'Mjob_services',
 'Mjob_teacher',
 'Fjob_at_home',
 'Fjob_health',
 'Fjob_other',
 'Fjob_services',
 'Fjob_teacher',
 'reason_course',
 'reason_home',
 'reason_other',
 'reason_reputation',
 'guardian_father',
 'guardian_mother',
 'guardian_other',
 'schoolsup_no',
 'schoolsup_yes',
 'famsup_no',
 'famsup_yes',
 'paid_no',
 'paid_yes',
 'activities_no',
 'activities_yes',
 'nursery_no']
```

4) Predictive Modeling (Supervised)

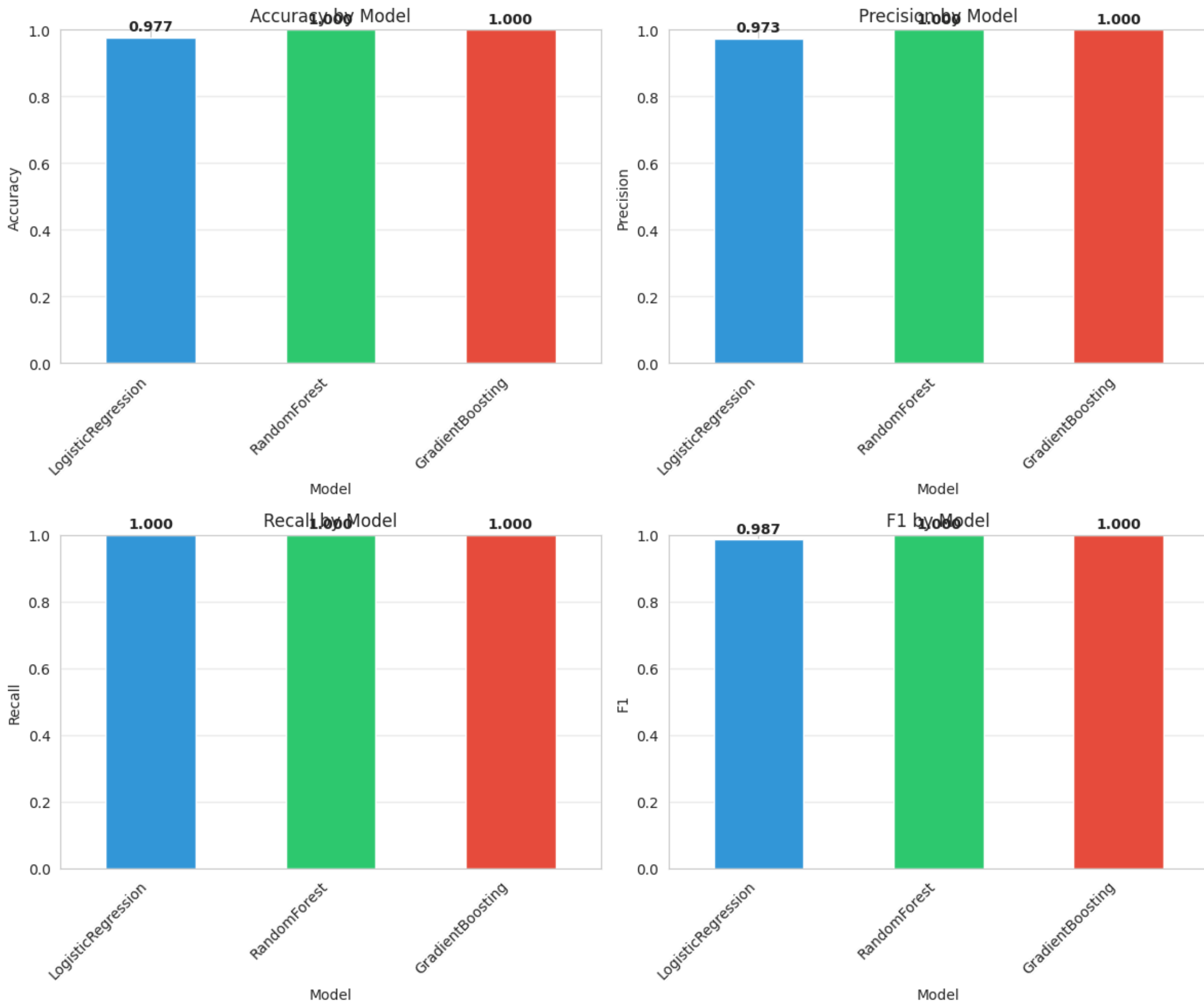
Three models: Logistic Regression, Random Forest, and Gradient Boosting.

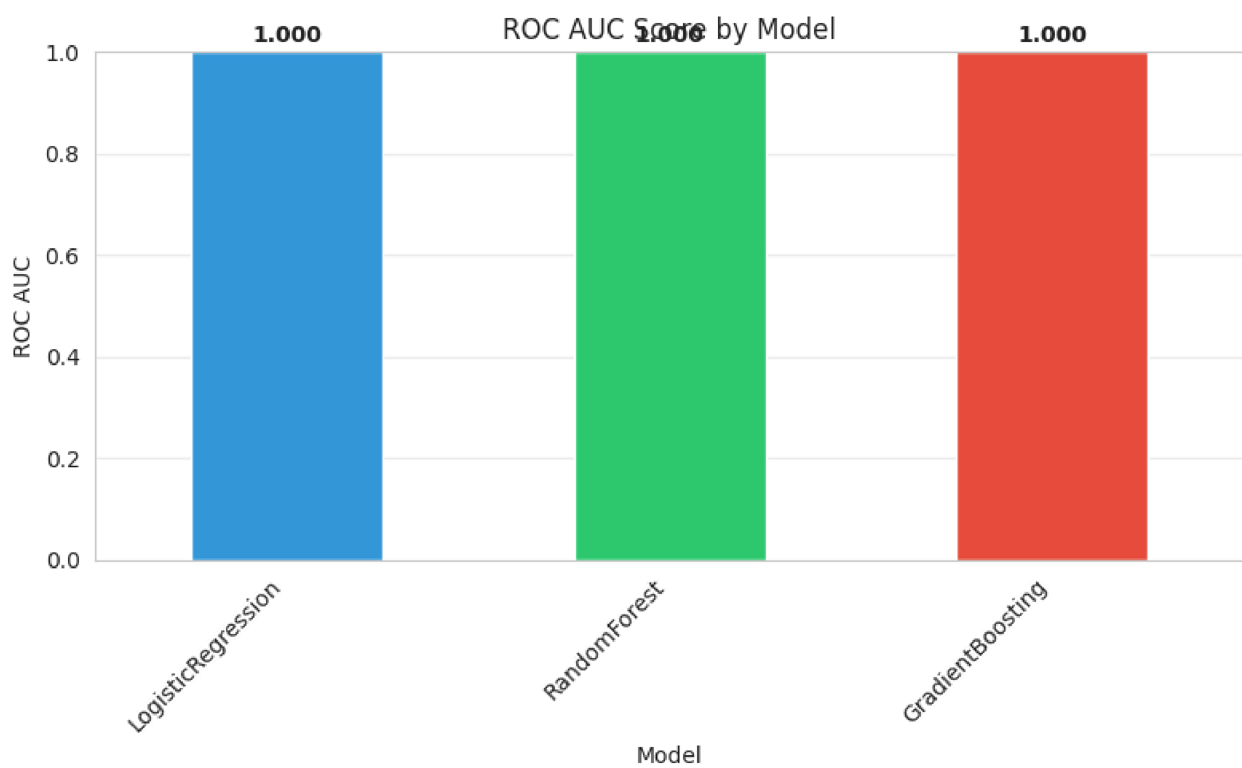
```
# Modeling: train/test split and basic training pipeline
from sklearn.model_selection import train_test_split, cross_val_score, StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, conf
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, conf

# Only proceed if we have a target
if y is None:
    print('No target variable defined. Define y to proceed with supervised modeling.')
else:
    X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2, random_state=42)
    # Fit preprocessor ONLY on training data to avoid data leakage
    X_train_pre = preproc.fit_transform(X_train)
    X_test_pre = preproc.transform(X_test)

    models = {
        'LogisticRegression': LogisticRegression(max_iter=1000, random_state=42),
        'RandomForest': RandomForestClassifier(n_estimators=200, random_state=42),
        'GradientBoosting': GradientBoostingClassifier(n_estimators=200, random_state=42)
    }

    results = {}
    for name, model in models.items():
        model.fit(X_train_pre, y_train)
        y_pred = model.predict(X_test_pre)
        y_proba = model.predict_proba(X_test_pre)[:,1] if hasattr(model, 'predict_proba') else None
        res = {
            'accuracy': accuracy_score(y_test, y_pred),
            'precision': precision_score(y_test, y_pred, zero_division=0),
            'recall': recall_score(y_test, y_pred, zero_division=0),
            'f1': f1_score(y_test, y_pred, zero_division=0),
        }
        if y_proba is not None:
            res['roc_auc'] = roc_auc_score(y_test, y_proba)
        results[name] = res
        print(f'--- {name} ---')
        print(classification_report(y_test, y_pred))
    print('\nSummary results:')
    import pandas as pd
    display(pd.DataFrame(results).T)
```

```
--- LogisticRegression ---
              precision    recall  f1-score   support

           0       1.00      0.85      0.92        20
           1       0.97      1.00      0.99       110

    accuracy                           0.98       130
   macro avg       0.99      0.93      0.95       130
weighted avg       0.98      0.98      0.98       130
```

```
# Visualize model performance comparison
if y is not None and len(results) > 0:
    import pandas as pd
    results_df = pd.DataFrame(results).T

    # Plot metrics comparison
    fig, axes = plt.subplots(2, 2, figsize=(12, 10))
    metrics = ['accuracy', 'precision', 'recall', 'f1']

    for idx, metric in enumerate(metrics):
        ax = axes[idx // 2, idx % 2]
        if metric in results_df.columns:
            results_df[metric].plot(kind='bar', ax=ax, color=['#3498db', '#2ecc71', '#e74c3c'])
            ax.set_title(f'{metric.capitalize()} by Model')
            ax.set_ylabel(metric.capitalize())
            ax.set_xlabel('Model')
            ax.set_ylim([0, 1])
            ax.grid(True, alpha=0.3, axis='y')
            ax.set_xticklabels(results_df.index, rotation=45, ha='right')

            # Add value labels on bars
            for i, v in enumerate(results_df[metric]):
                ax.text(i, v + 0.02, f'{v:.3f}', ha='center', fontweight='bold')

    plt.tight_layout()
    plt.show()

    # Display ROC AUC comparison if available
    if 'roc_auc' in results_df.columns:
        plt.figure(figsize=(8, 5))
        results_df['roc_auc'].plot(kind='bar', color=['#3498db', '#2ecc71', '#e74c3c'])
        plt.title('ROC AUC Score by Model')
        plt.ylabel('ROC AUC')
        plt.xlabel('Model')
        plt.ylim([0, 1])
        plt.grid(True, alpha=0.3, axis='y')
        plt.xticks(rotation=45, ha='right')

        # Add value labels
        for i, v in enumerate(results_df['roc_auc']):
            plt.text(i, v + 0.02, f'{v:.3f}', ha='center', fontweight='bold')

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


Accuracy by Model
0.977
1.000
1.000
Accuracy
Model
LogisticRegression
RandomForest
GradientBoosting
Precision by Model
0.973
1.000
1.000
Precision
Model
Recall by Model
1.000
1.000
1.000
Recall
Model
F1 by Model
0.987
1.000
1.000
F1
Model


ROC AUC Score by Model
1.000
1.000
1.000
ROC AUC
LogisticRegression
RandomForest
GradientBoosting
Model

```
# Cross-validation (Stratified K-Fold) for each model
# Use Pipeline to avoid data leakage - preprocessor fits only on training folds
if y is not None:
    from sklearn.pipeline import Pipeline
    skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
    cv_results = {}

    # Create pipelines for each model to ensure proper cross-validation
    for name, model in models.items():
        # Create a fresh preprocessor for CV to avoid leakage
        from sklearn.compose import ColumnTransformer
        num_pipe_cv = Pipeline([('imputer', SimpleImputer(strategy='median'))])
        cat_pipe_cv = Pipeline([('imputer', SimpleImputer(strategy='most_frequent')),
                               ('onehot', OneHotEncoder(handle_unknown='ignore', sparse_output=False))])
        preproc_cv = ColumnTransformer([('num', num_pipe_cv, num_cols),
                                       ('cat', cat_pipe_cv, cat_cols)], remainder='drop')

        # Create pipeline with preprocessing and model
        pipeline = Pipeline([('preprocess', preproc_cv), ('classifier', model)])
        scores = cross_val_score(pipeline, X, y, cv=skf, scoring='f1')
        cv_results[name] = scores
    print('Cross-val F1 scores:')
    display(pd.DataFrame(cv_results))
```

Cross-val F1 scores:

	LogisticRegression	RandomForest	GradientBoosting
0	1.000000	1.000000	1.0
1	0.995475	1.000000	1.0
2	1.000000	1.000000	1.0
3	1.000000	1.000000	1.0
4	0.986425	0.995434	1.0

Feature importance (for tree-based models)

Show top features from RandomForest.

```
# Feature importances from RandomForest
import numpy as np
if 'RandomForest' in models and feature_names is not None:
    rf = models['RandomForest']
    importances = rf.feature_importances_
    idx = np.argsort(importances)[::-1]

    print('Top 20 Most Important Features:')
    print('-' * 50)
    for i, feat_idx in enumerate(idx[:20], 1):
        print(f'{i:2d}. {feature_names[feat_idx]:40s} {importances[feat_idx]:.4f}')

    # Visualize top features
    plt.figure(figsize=(10, 6))
    top_n = 15
    top_idx = idx[:top_n]
    plt.barh(range(top_n), importances[top_idx])
    plt.yticks(range(top_n), [feature_names[i] for i in top_idx])
    plt.xlabel('Feature Importance')
    plt.title('Top 15 Feature Importances (Random Forest)')
    plt.gca().invert_yaxis()
    plt.tight_layout()
```