Machine Learning for Determining Project Team Size

- Eric Scrivens
- Nicole Nageli

CSU CS-345: Machine Learning Foundations and Practice

Fall 2025

Introduction

Our goal with this project was to build machine learning models that can predict team size based on project characteristics.

We're evaluating multiple approaches, comparing them against a naive baseline, then evaluating and understanding which features and models contribute the most to predictive accuracy.

Dataset

As mentioned in our project proposal, the Project Management Risk Raw dataset contains 4000 simulated project management data points and 51 input variables (26 categorical, 25 numeric) that describe demographic, financial, and risk-related features.

Because the dataset contains many categorical variables, one of the big challenges with this project was with encoding and feature selection. When Eric and I were looking through the 51 features, we found many predictors that were vague, subjective, and environmental factors that would not have a direct relationship with our target variable team size. Examples of features we dropped were 'documentation quality' (subjective), 'risk management maturity' which had values like 'basic' or 'formal' which are vague, and 'industry volatility.'

Cited Sources and dataset:

- Adam, Tarek. “Project Management Risk Raw.” Kaggle, 24 May 2025, www.kaggle.com/datasets/ka66ledata/project-management-risk-raw/data.

Methodology

Our workflow was to

1. load and clean data through preprocessing

2. exploratory analysis
3. categorical encoding
4. variance filtering of weak features
5. standardize numeric features
6. model training & hyperparameter tuning
7. cross-validation and grid search
8. performance evaluation

The models we tested are

- naive baseline
- linear regression
- lasso regression (L1)
- ridge regression (L2)
- support vector machine

We used Mean Absolute Error to evaluate since team size is measured in units of people.

Imports

In [ ]:
```
import numpy as np
import pandas as pd
from pandas.core.arrays import categorical
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn.svm import SVR
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split, cross_val_score, KFold, GridS
from sklearn.preprocessing import OneHotEncoder, PolynomialFeatures, StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_absolute_error
from sklearn.feature_selection import VarianceThreshold
from sklearn.linear_model import Lasso
```

Data Loader

In [ ]:
```
def load_data():
    filename='data/project_risk.csv'
    data = pd.read_csv(filename, na_values='?', sep=',')
    X = data.drop(columns=['Team_Size'])
    y = data['Team_Size'].astype(float)  # explicitly picking target column
    numeric_cols = X.select_dtypes(include=np.number).columns.tolist()
    categorical_cols = X.select_dtypes(exclude=np.number).columns.tolist()
    return X, y, numeric_cols, categorical_cols

X, y, numeric_cols, categorical_cols = load_data()
```

```
# tests
print(len(numeric_cols), numeric_cols)
print(len(categorical_cols), categorical_cols)
```

```
18 ['Project_Budget_USD', 'Estimated_Timeline_Months', 'Complexity_Score', 'Change_R
equest_Frequency', 'Team_Turnover_Rate', 'Communication_Frequency', 'Geographical_Di
stribution', 'Integration_Complexity', 'Resource_Availability', 'Previous_Delivery_S
uccess_Rate', 'Current_Phase_Duration_Months', 'Team_Experience_Level(0-3)', 'Regula
tory_Compliance_Level', 'Stakeholder_Engagement_Level', 'Priority_Level', 'Data_Secu
rity_Requirements', 'Risk_Level', 'Project_Manager_Experience']
3 ['Project_Type', 'Methodology_Used', 'Team_Colocation']
```

- We have performed basic data preprocessing to the import file to ensure readability, and formatting for our data loader. We then use a method to import the data and returns:
  - X: features
  - y: labels
  - categorical_cols: a list of the features that are categorical in nature. These will be used for further processing during the data analysis.

Train Test Split

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                    test_size=0.3, random_state=42)
```

Preprocess and Scale

In [ ]:
```
# ColumnTransformer for preprocessing
preprocessor = ColumnTransformer(transformers=[
    ("num", "passthrough", numeric_cols),  # keep numeric as-is for now
    ("cat", OneHotEncoder(handle_unknown="ignore", sparse_output=False), categorica
])

# Function to scale numeric features
def scaler(X, numeric_count=len(numeric_cols)):
    """Assumes numeric features are first columns."""
    X_scaled = X.copy()
    X_scaled[:, :numeric_count] = StandardScaler().fit_transform(X_scaled[:, :numer
    return X_scaled

scale_numeric_transformer = FunctionTransformer(scaler)
```

- Since we are creating multiple machine learning models utilizing scikit-learn pipelines, we will create a preprocessor that can be utilized within our pipelines.
  - First a ColumnTransformer is used to keep the numeric columns unchanged while applying one-hot encoding to the categorical columns.

- Next, a custom scaling step is added. The scaler() applies StandardScaler() to the numeric features, and leaves the encoded categorical features unchanged. This allows us to scale numerical data after one-hot encoding and variance filtering, while still keeping categorical features unmodified.

Naive Model as a Baseline

In [ ]:
```
# Naive Regression Model
def naive_regression(y_train, y_test):
    y_mean = np.mean(y_train)
    # create a baseline array
    # all elements being the same value (y_mean) and shape of y_test
    y_pred_naive = np.full(y_test.shape[0], y_mean)

    # Compute baseline error
    mae_naive = mean_absolute_error(y_test, y_pred_naive)
    return mae_naive

mae_naive = naive_regression(y_train, y_test)
print(mae_naive)
```

```
7.359576190476191
```

Naive Baseline Evaluation

- The large MAE of ~7.36 shows that blindly predicting team size doesn't give usable project-specific patterns. This is the reference that we will use to determine if our models can improve upon.

SVR pipeline regression baseline and Grid search

In [ ]:
```
# SVR cross-validation setup
svr_cv = KFold(n_splits=5, shuffle=True, random_state=42)

# SVR Pipeline - baseline
def baseline_SVR(X_train, y_train, preprocessor, svr_cv):
    pipeline = Pipeline([
        ("preprocess", preprocessor),
        ("svr", SVR(kernel="rbf", C=1, gamma=0.01))  # simple baseline parameters
    ])

    scores = cross_val_score(pipeline, X_train, y_train, cv=svr_cv, scoring="neg_me
    baseline = -np.mean(scores)

    return baseline


# SVR Pipeline - Gridsearch
def pipeline_SVR(X_train, X_test, y_train, y_test, preprocessor, cv):
    pipeline = Pipeline(steps=[
        ("preprocess", preprocessor),
        ("var_filter", VarianceThreshold(threshold=2)),
```

```
        ("scale_numeric", scale_numeric_transformer),
        ("svr", SVR())
    ])

    param_grid = {
        "svr__C": [0.1, 1, 10, 100],
        "svr__gamma": [0.001, 0.01, 0.1],
        "svr__kernel": ["rbf"]
    }

    grid = GridSearchCV(
        estimator=pipeline,
        param_grid=param_grid,
        scoring="neg_mean_absolute_error",
        cv=cv,
        n_jobs=-1
    )

    grid.fit(X_train, y_train)
    best_params = grid.best_params_
    mae_train = -grid.best_score_

    y_pred = grid.best_estimator_.predict(X_test)
    mae_test = mean_absolute_error(y_test, y_pred)

    # ---------------------------
    # Handle feature name extraction
    # ---------------------------
    if preprocessor is None:
        all_features = list(X_train.columns)
        var_mask = grid.best_estimator_.named_steps["var_filter"].get_support()
        final_features = [f for f, keep in zip(all_features, var_mask) if keep]

    else:
        preprocess = grid.best_estimator_.named_steps["preprocess"]
        ohe = preprocess.named_transformers_["cat"]
        ohe_feature_names = ohe.get_feature_names_out(categorical_cols)

        all_features = numeric_cols + list(ohe_feature_names)
        var_mask = grid.best_estimator_.named_steps["var_filter"].get_support()
        final_features = [f for f, keep in zip(all_features, var_mask) if keep]

    print("\nFinal feature names:")
    for f in final_features:
        print(" -", f)

    # ---------------------------
    # Extract X_final for plotting
    # ---------------------------
    if preprocessor is None:
        X_after_pre = X_train
    else:
        X_after_pre = preprocess.transform(X_train)

    X_final = grid.best_estimator_.named_steps["var_filter"].transform(X_after_pre)
```

```
    return best_params, mae_train, mae_test, grid, final_features, X_final
```

SVR Calculations

In [ ]:
```
mae_naive = naive_regression(y_train, y_test)
svr_baseline = baseline_SVR(X_train, y_train, preprocessor, svr_cv)
svr_best_params, svr_mae_train, svr_mae_test, svr_grid, final_features, X_final = p
                                X_train, X_test, y_train, y_test, preprocessor, svr

print(f"\nNaive model MAE: {mae_naive:.3f}")
print(f"SVR - Baseline MAE: {svr_baseline:.3f}")
print("SVR - Best Params:", svr_best_params)
print(f"SVR - Best Train MAE: {svr_mae_train:.3f}")
print(f"SVR - Test MAE: {svr_mae_test:.3f}")
```

```
Final feature names:
 - Project_Budget_USD
 - Estimated_Timeline_Months
 - Complexity_Score
 - Communication_Frequency
 - Integration_Complexity
 - Current_Phase_Duration_Months

Naive model MAE: 7.360
SVR - Baseline MAE: 6.895
SVR - Best Params: {'svr__C': 100, 'svr__gamma': 0.01, 'svr__kernel': 'rbf'}
SVR - Best Train MAE: 2.246
SVR - Test MAE: 2.247
```

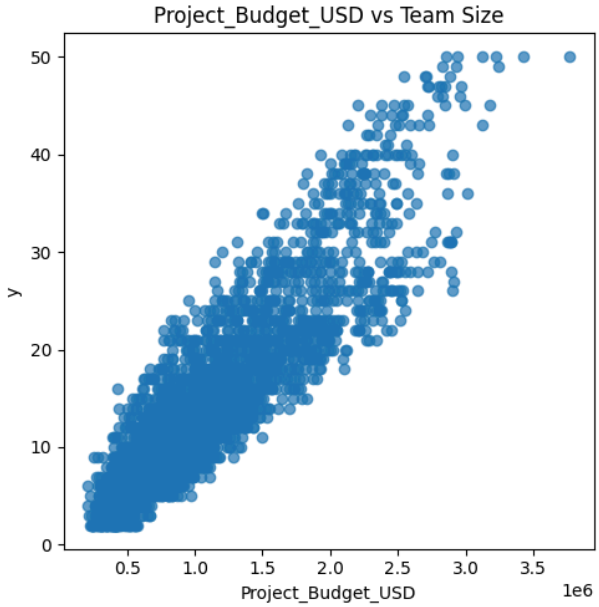
SVR Initial Analysis:

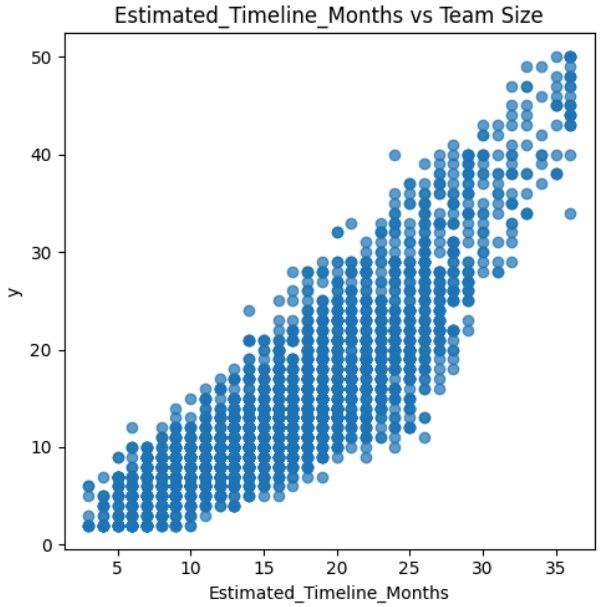
- Utilizing the pipeline VarianceThreshold(threshold=2) yeilded the best results. This narrowed the top features down to six key features that will be visualized later. The final SVR model was trained using six key features related to project budget, timeline, complexity, communication, integration difficulty, and current phase duration.
- A naïve baseline model produced a mean absolute error (MAE) of 7.360, while the SVR baseline marginally improved performance to an MAE of 6.895. After applying GridSearchCV, the best SVR configuration used an RBF kernel with C = 100 and γ = 0.01. This reduced the training error to 2.246 MAE. When evaluated on the held-out test set, the optimized model achieved a nearly identical 2.247 MAE, indicating strong generalization and stable performance on unseen data.

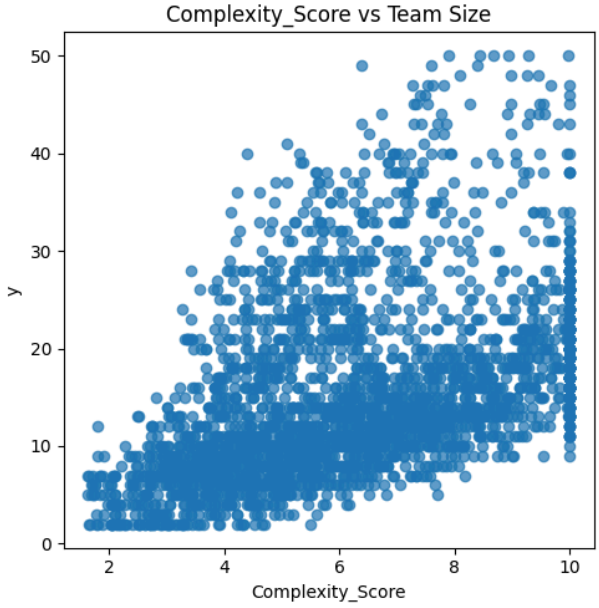
Top Features Visualized

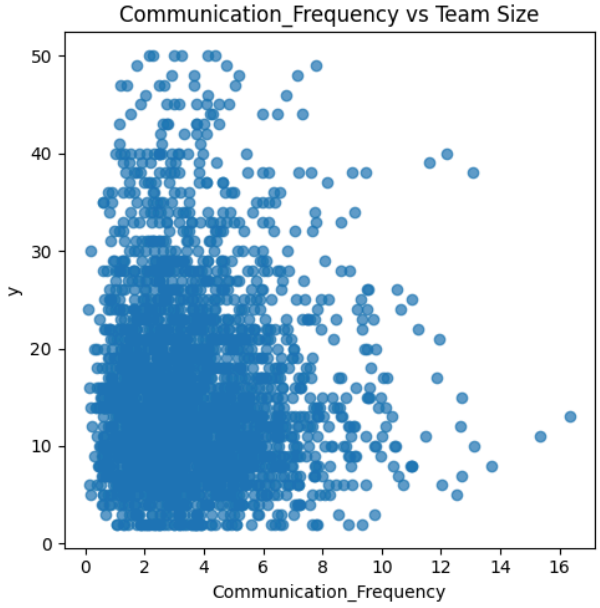
In [ ]:
```
n_features = len(final_features)
rows = 3
cols = 2

plt.figure(figsize=(10, 15))
```

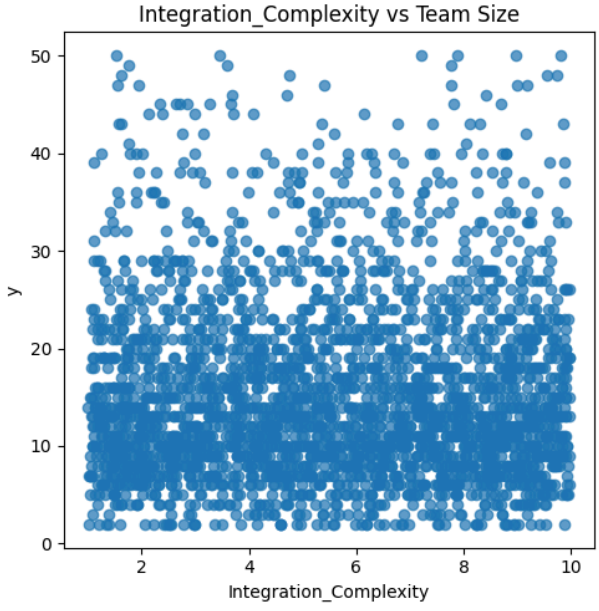
```
for i in range(n_features):
    plt.subplot(rows, cols, i + 1)
    plt.scatter(X_final[:, i], y_train, alpha=0.7)
    plt.xlabel(final_features[i])
    plt.ylabel("y")
    plt.title(f"{final_features[i]} vs Team Size")

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

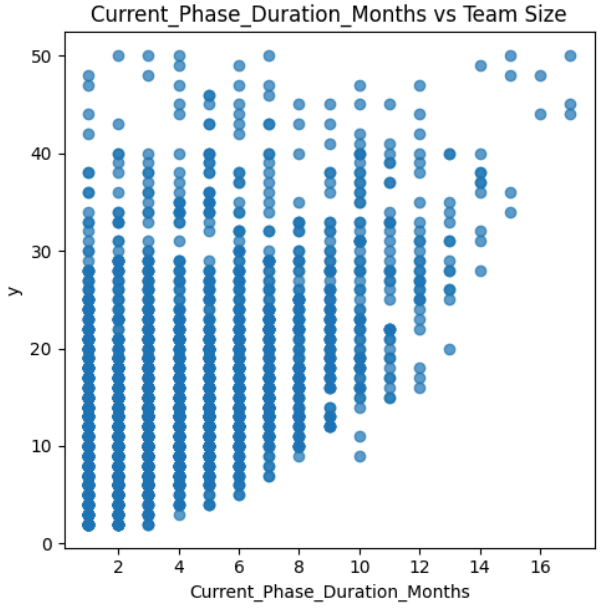

Project_Budget_USD vs Team Size
y
Project_Budget_USD
1e6


Estimated_Timeline_Months vs Team Size
y
Estimated_Timeline_Months


Complexity_Score vs Team Size
y
Complexity_Score


Communication_Frequency vs Team Size
y
Communication_Frequency


Integration_Complexity vs Team Size
y
Integration_Complexity


Current_Phase_Duration_Months vs Team Size
y
Current_Phase_Duration_Months

Top Six Features Visualized:

- In order to get a greater understanding of how our features may correspond to finding the ideal team size, we plotted them against their corresponding labels. These are the features as defined by our VarianceThreshold. While several showed little noticable correlations such as Integration Complexity, there were several features that showed a strong correlation to team size. For the SVR models these included:
  - Project Budget USD
  - Estimated Timeline Months
  - Complexity Score

Expanding SVR to further improve performance

In [ ]:
```
selected_features = ["Project_Budget_USD", "Estimated_Timeline_Months", "Complexity

X_train_sel = X_train[selected_features]
X_test_sel  = X_test[selected_features]

svr_best_params, svr_mae_train_sel, svr_mae_test_sel, svr_grid_sel, final_features_
    X_train_sel, X_test_sel, y_train, y_test,
    preprocessor=None,
    cv=svr_cv
)

print(f"SVR - Best Train MAE: {svr_mae_train_sel:.3f}")
print(f"SVR - Test MAE: {svr_mae_test_sel:.3f}")
```

```
Final feature names:
 - Project_Budget_USD
 - Estimated_Timeline_Months
 - Complexity_Score
SVR - Best Train MAE: 2.162
SVR - Test MAE: 2.132
```

SVR Optimized:

- As noted, Project Budget USD, Estimated Timeline Months, and Complexity Score showed a strong visual correlation to the ideal project team size. In an attempt to further extract a better result, SVR was again ran through the pipeline utilizing only these three features.

- While the best test MAE utilizing the six visualized features was 2.247, we were able to extract slightly better results utilizing only the top three features with a MAE of 2.132.

Correlation Matrix

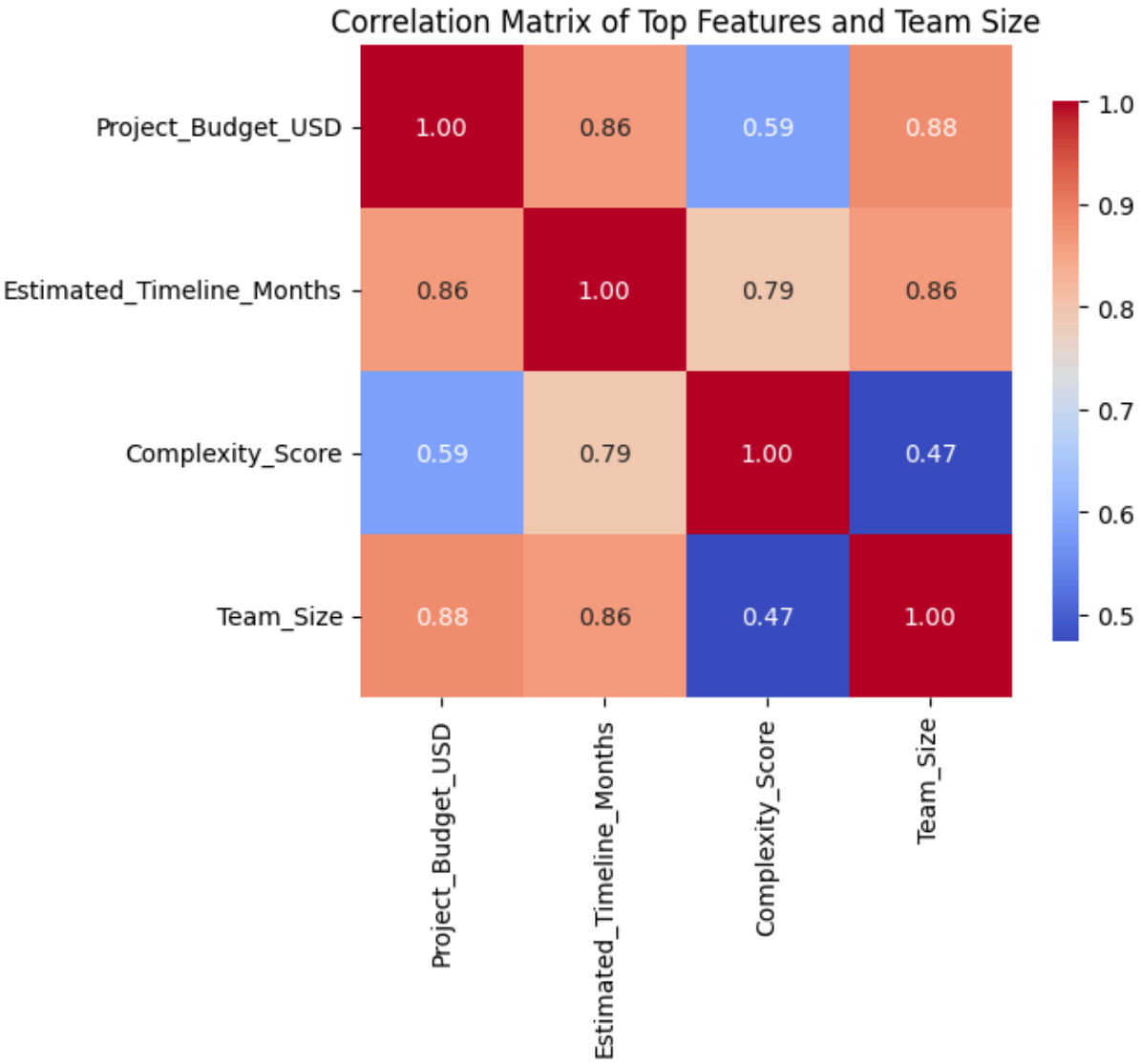
In [ ]:
```
# after preprocessing
def correlation_matrix(X_final, feature_names, y):
  df = pd.DataFrame(X_final, columns=feature_names)
```

```
  df['Team_Size'] = y.values  # target
  corr = df.corr()
  return corr

corr = correlation_matrix(X_final, selected_features, y_train)
print(corr)
```

```
                          Project_Budget_USD  Estimated_Timeline_Months  \
Project_Budget_USD                  1.000000                   0.862899
Estimated_Timeline_Months           0.862899                   1.000000
Complexity_Score                    0.587451                   0.794465
Team_Size                           0.884291                   0.863824

                          Complexity_Score  Team_Size
Project_Budget_USD                0.587451   0.884291
Estimated_Timeline_Months         0.794465   0.863824
Complexity_Score                  1.000000   0.473248
Team_Size                         0.473248   1.000000
```

Correlation Matrix Visualizations

In [ ]:
```
plt.figure(figsize=(6, 5))
sns.heatmap(corr, annot=True, fmt=".2f", cmap="coolwarm", square=True, cbar_kws={'s
plt.title("Correlation Matrix of Top Features and Team Size")
plt.show()
# positive correlations are red and negative are blue
```

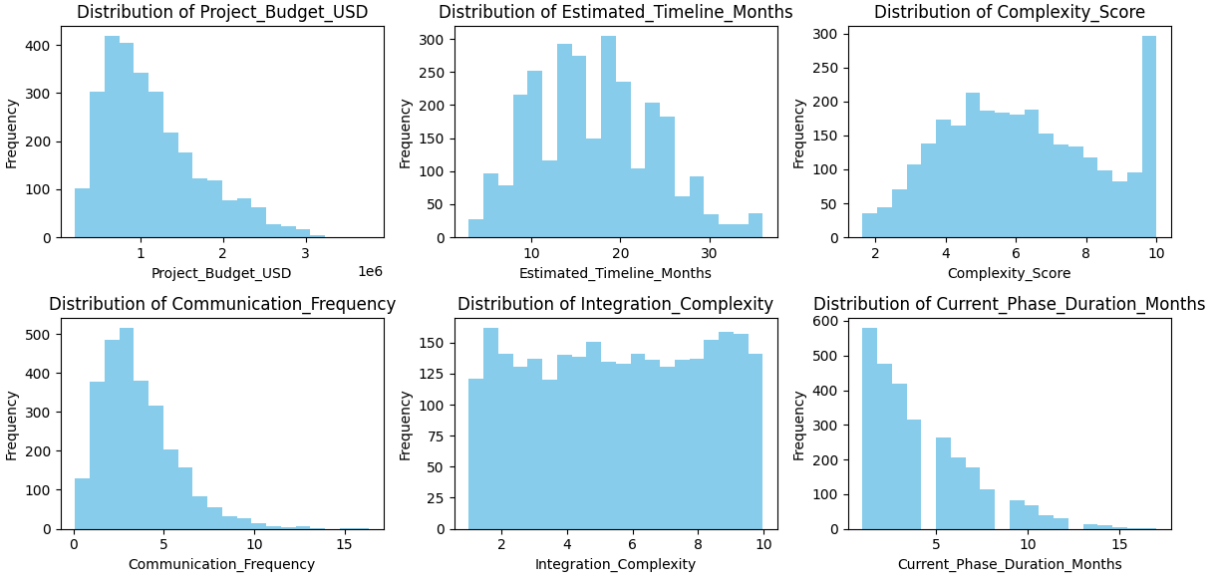


Top Feature's distributions across the dataset

In [ ]:

```
svr_best_params, svr_mae_train, svr_mae_test, svr_grid, final_features, X_final_all
    X_train, X_test, y_train, y_test, preprocessor, svr_cv
)

n_features = len(final_features)
rows = 2
cols = 3

plt.figure(figsize=(12, 6))
for i, feature in enumerate(final_features):
    plt.subplot(rows, cols, i + 1)
    plt.hist(X_final_all[:, i], bins=20, color='skyblue')
    plt.title(f'Distribution of {feature}')
    plt.xlabel(feature)
    plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```

```
Final feature names:
 - Project_Budget_USD
 - Estimated_Timeline_Months
 - Complexity_Score
 - Communication_Frequency
 - Integration_Complexity
 - Current_Phase_Duration_Months
```



Feature Engineering & Feature Importance Discussion

Feature selection matters immensely in a dataset with a high-dimensional categorical space. Preproccessing and feature selection was critical so we can reduce dimensionality and improve our models. The dangers of not taking these into account mean that our models can overfit to the data, so we needed to make sure the features selected were justified.

At this point, we had already dropped vague unhelpful features. Eric did preprocessing through one-hot encoding and VarianceThreshold filtering. Because one-hot encoding converts each categorical value into multiple binary indicators, the dimensionality greatly increases, which is why feature filtering is important. Variance Threshold removes features that have low variation that don't meaningfully tell us anything about team size.

After this, there were 6 features that remained. Ranked from highest:

1. Project Budget
2. Estimated Timeline (Months)
3. Complexity Score
4. Communication Frequency
5. Integration_Complexity
6. Current Phase_Duration (Months)

Why are these the most predictive features of team size?

My interpretation:

1. Larger budgets = larger teams
2. Longer timelines may correlate with project scope.
3. Projects with higher complexity might need more people to handle tasks.
4. The more people there are, the more communication needs to happen.
5. More integration means more coordination among additional people
6. A long current phase could reflect the volume of work.

Conceptually, what all these have in common is project scale. The correlation matrix I made confirms this: 1 is perfect positive correlation, -1 is opposite direction negative correlation, 0 is no correlation. Project_Budget_USD and Estimated_Timeline_Months are highly correlated with team size (0.88 and 0.86), while Complexity_Score also shows moderate positive correlation (0.47). Features are also inter-correlated, reflecting their shared underlying relationship with project scale. Positive correlations are highlighted in red and negative correlations in blue.

The histograms also confirm that larger budgets, longer timelines, and higher complexity projects tend to correspond with larger teams, though there are diminishing returns at the high end.

You can see that Eric's SVR model performance significantly improved. The naive baseline's MAE was 7.36 while SVR was 2.13. This supports what we theorized in the project proposal: removing weak features dramatically reduces model error, which suggests that the most meaningful and useful indicators of team size are concentrated in a small set of these technical project variables. While we started with 51 original features, the most important predictors came down to these 6 features. This demonstrates that dependence on technical project attributes matters far more than organizational or contextual factors.

The scatter plots of the top features against team size confirm that team size increases with project scope and budget, though with diminishing returns at higher values.

Linear Regression Baseline Model

In [ ]:
```
def linear_regression_baseline(X_train, y_train, preprocessor, cv):
  # lin reg pipeline
  pipeline = Pipeline([
        ("preprocess", preprocessor),
        ("var_filter", VarianceThreshold(threshold=2)),
        ("scale_numeric", scale_numeric_transformer),
        ("linear", LinearRegression())
    ])

  scores = cross_val_score(pipeline, X_train, y_train, cv=cv, scoring="neg_mean_abs
  baseline_mae = -np.mean(scores)
  return baseline_mae
```

```
linreg_baseline_mae = linear_regression_baseline(X_train, y_train, preprocessor, sv
print("Linear Regression Baseline MAE:", linreg_baseline_mae)
```

```
Linear Regression Baseline MAE: 2.372246306538005
```

Linear Regression Evaluation

Using a standard linear regression model and 5 fold cross-validation for,the MAE is around 2.25.

This is a large improvement compared to the naive model (7.36), similar to SVR with best parameters (2.25). This means that a linear relationship between our best features and team size captures most of the variance in the data.

L1 Regularization

In [ ]:
```
def pipeline_lasso(X_train, X_test, y_train, y_test, preprocessor, cv):
    pipeline = Pipeline(steps=[
        ("preprocess", preprocessor),
        ("var_filter", VarianceThreshold(threshold=2)),
        ("scale_numeric", scale_numeric_transformer),
        ("lasso", Lasso(max_iter=5000))
    ])

    param_grid = {"lasso__alpha": [0.001, 0.01, 0.1, 1, 10]}

    grid = GridSearchCV(
        estimator=pipeline,
        param_grid=param_grid,
        scoring="neg_mean_absolute_error",
        cv=cv,
        n_jobs=-1
    )

    grid.fit(X_train, y_train)
    best_params = grid.best_params_
    mae_train = -grid.best_score_

    best_lasso = grid.best_estimator_.named_steps["lasso"]
    coefficients = best_lasso.coef_

    y_pred = grid.best_estimator_.predict(X_test)
    mae_test = mean_absolute_error(y_test, y_pred)

    return best_params, mae_train, mae_test, coefficients

lasso_best_params, lasso_mae_train, lasso_mae_test, coefficients=pipeline_lasso(
    X_train, X_test, y_train, y_test, preprocessor, svr_cv
)
feature_coef = dict(zip(final_features, coefficients))
print("Lasso coefficients:", feature_coef)
```

```
print("Lasso best alpha:", lasso_best_params)
print("Lasso MAE (train/test):", lasso_mae_train, lasso_mae_test)
```

```
Lasso coefficients: {'Project_Budget_USD': np.float64(3.303217166483341), 'Estimated
_Timeline_Months': np.float64(8.53200154504801), 'Complexity_Score': np.float64(-4.3
686321503533945), 'Communication_Frequency': np.float64(0.0), 'Integration_Complexit
y': np.float64(-0.06313101715526485), 'Current_Phase_Duration_Months': np.float64(0.
0)}
Lasso best alpha: {'lasso__alpha': 0.01}
Lasso MAE (train/test): 2.371498138921697 2.3548489391921077
```

Feature Names and Lasso Coefficients Table

In [ ]:
```
lasso_table = pd.DataFrame(list(feature_coef.items()), columns=["Feature", "Lasso C

# sorting by absolute value for most important features
lasso_table["Abs_Coeff"] = lasso_table["Lasso Coefficient"].abs()
lasso_table = lasso_table.sort_values(by="Abs_Coeff", ascending=False).drop(columns

print(lasso_table)
```

```
                       Feature  Lasso Coefficient
1      Estimated_Timeline_Months           8.532002
2               Complexity_Score          -4.368632
0             Project_Budget_USD           3.303217
4          Integration_Complexity          -0.063131
3        Communication_Frequency           0.000000
5  Current_Phase_Duration_Months           0.000000
```

L1 Evaluation

Lasso regression drives the coefficients of less important features to zero. Grid search was used to find the best alpha.

The results show that the best alpha is 0.01, and L1 has a MAE of 2.3. Features with coefficients near zero like communication frequency and current phase duration are ignored, which means they don't have much predictive value. The strongest predictors are Estimated Timeline, Project Budget, and Complexity score which confirms our earlier feature engineering analysis.

Ridge Regression with GridSearch

In [ ]:
```
def pipeline_ridge(X_train, X_test, y_train, y_test, preprocessor, cv):
    pipeline = Pipeline(steps=[
        ("preprocess", preprocessor),
        ("var_filter", VarianceThreshold(threshold=2)),
        ("scale_numeric", scale_numeric_transformer),
        ("ridge", Ridge())
    ])

    param_grid = {"ridge__alpha": [0.01, 0.1, 1, 10, 100]}
```

```
    grid = GridSearchCV(
        estimator=pipeline,
        param_grid=param_grid,
        scoring="neg_mean_absolute_error",
        cv=cv,
        n_jobs=-1
    )

    grid.fit(X_train, y_train)
    best_params = grid.best_params_
    mae_train = -grid.best_score_

    y_pred = grid.best_estimator_.predict(X_test)
    mae_test = mean_absolute_error(y_test, y_pred)

    return best_params, mae_train, mae_test

ridge_best_params, ridge_mae_train, ridge_mae_test = pipeline_ridge(
    X_train, X_test, y_train, y_test, preprocessor, svr_cv
)

print("Ridge best alpha:", ridge_best_params)  # regularization strength
print("Ridge MAE (train/test):", ridge_mae_train, ridge_mae_test)
```

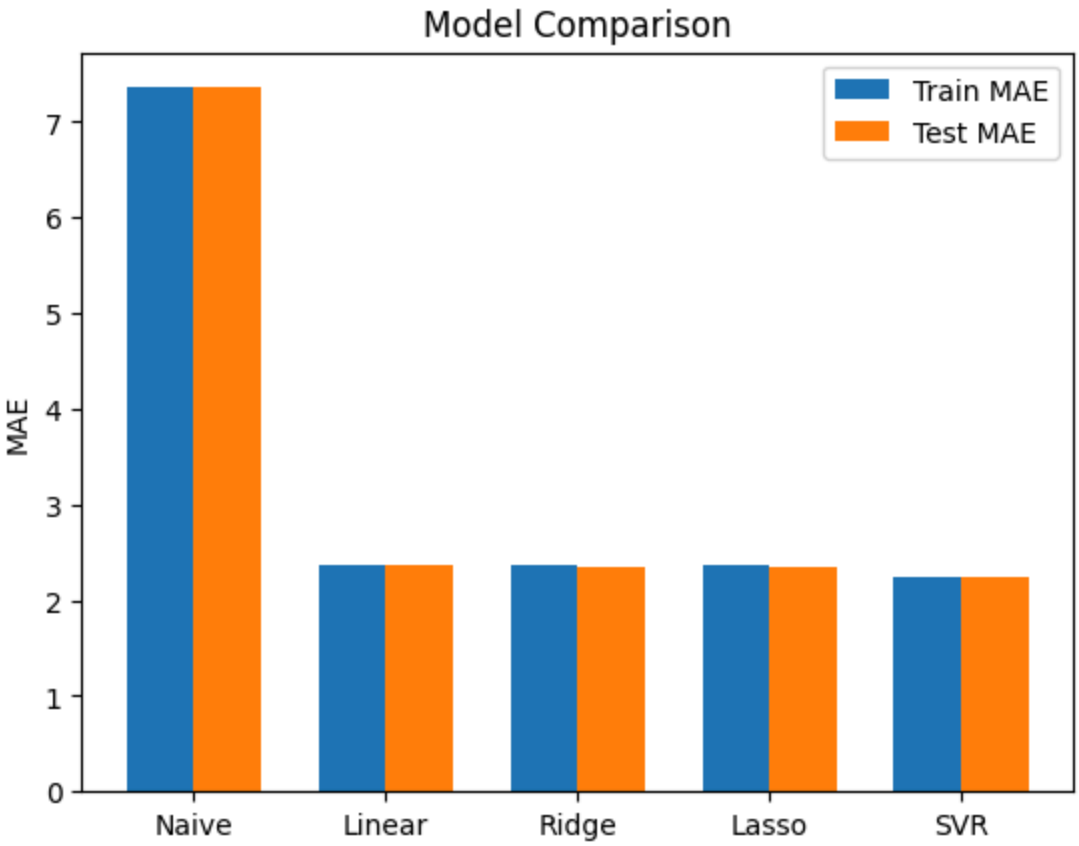
```
Ridge best alpha: {'ridge__alpha': 1}
Ridge MAE (train/test): 2.37212581390125 2.3545922992952235
```

L2 Evaluation

Ridge regression punishes large coefficients in order to reduce overfitting. The best paramaters for alpha was 1 and the MAE was around 2.3, which is identical to linear regression. This tells us that we're not overfitting on the dataset.

All Models Comparison Visualizations

In [ ]:
```
models = ["Naive", "Linear", "Ridge", "Lasso", "SVR"]
train_mae = [mae_naive, linreg_baseline_mae, ridge_mae_train, lasso_mae_train, svr_
test_mae = [mae_naive, linreg_baseline_mae, ridge_mae_test, lasso_mae_test, svr_mae

x = np.arange(len(models))
width = 0.35

fig, ax = plt.subplots()
ax.bar(x - width/2, train_mae, width, label="Train MAE")
ax.bar(x + width/2, test_mae, width, label="Test MAE")
ax.set_xticks(x)
ax.set_xticklabels(models)
ax.set_ylabel("MAE")
ax.set_title("Model Comparison")
ax.legend()
plt.show()
```



Overall Evaluation

We can see that feature engineering dramatically reduced MAE from the baseline of 7.5 to ~2.3 for all the models, which means we were successful.

Our models improved greatly using grid search and pipelines reflecting the need for extensive hyper-parameter tuning. Additionally, our accuracy on the dataset prior to standardizing the features performed poorly due to various scale and size factors in each feature set. For example costs could be many thousands of dollars, while one-hot encoded values were always limited to 1. We performed baseline testing prior to standardizing features simply as a sanity check compared to our mean-baseline.

Linear models capture most of the variance, lasso regularization confirms our most predictive features, ridge regression slightly reduces overfitting, and SVR has the lowest MAE since it's able to capture minor non-linear relationships.

We also found that performing additional tuning of the model based on the most expressive features added a lot of compute time while delivering very little actual improvement to our results (2.247 improved to 2.132).

Even though SVR offers marginally better predictions, the consistent low MAEs across linear and regularized models suggest that the selected technical project features strongly determine team size and can be represented in a linear relationship. Organizational or environmental variables are far less predictive in this dataset.

The strongest predictive features were structurally meaningful, suggesting that complex, large-scope projects genuinely require more staff. This aligns with the real-world expectations: larger, more complex projects require more team members, confirming that our model captures true project dynamics rather than noise.

Project Responsibilites

Nicole:

- Introduction
- Correlation Matrix and Visualization
- Top Feature's distributions across the dataset
- Feature Engineering & Feature Importance Discussion
- Linear Regression Baseline Model
- L1 Regularization
- Feature Names and Lasso Coefficients Table
- Ridge Regression with GridSearch
- All Models Comparison Visualizations
- Overall Model Evaluation

Eric:

- Data Format and Cleaning for import
- Data Loader
- Preprocess and Scale
- Naive Model
- Pipeline Creation
- SVR pipeline regression baseline and Grid search
- SVR - call models
- SVR top 3 features visually
- Expanding SVR to further improve performance
- Overall Model Evaluation