Machine Learning-Based Flight Delay Prediction

intro

Issue in Air Transportation:

- Flight delays significantly impact consumers and airports.
- Causes include weather, traffic, and scheduling inefficiencies.

Objective of the Study:

- Examine JFK Airport flight takeoff data.
- Identify primary reasons for flight delays.

Approach:

- Utilize machine learning models to predict departure delays.
- Compare performance of various regression models.

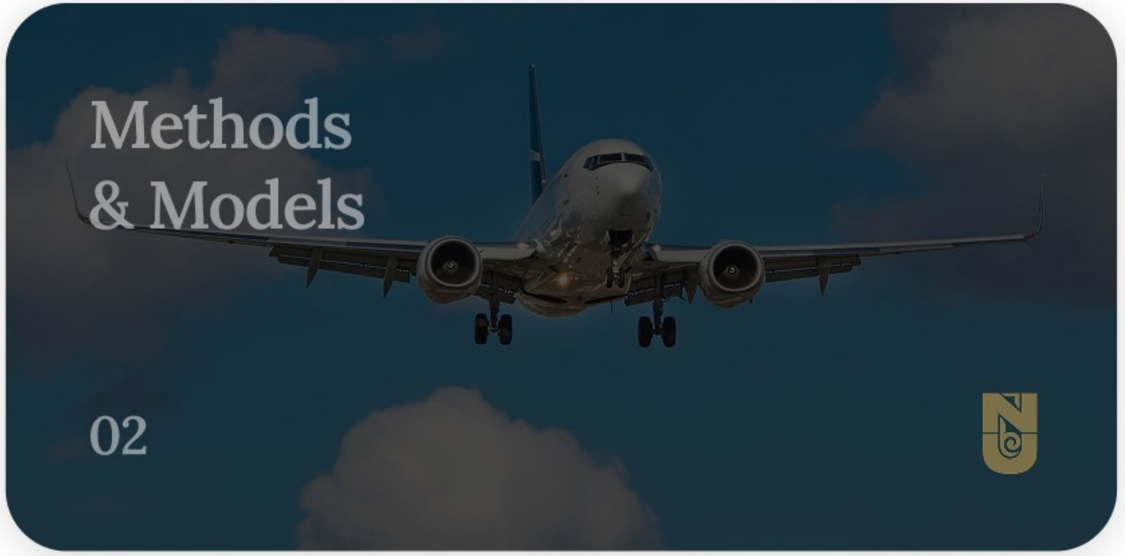
Research Goal:

- Determine the most important variables impacting flight departures.
- Assess the effectiveness of different machine learning algorithms.

Features & Preprocessing

01

Methods & Models



02

Results

03

about Dataset

> The dataset used for this study is sourced from Kaggle, titled "**Flight Take Off Data – JFK Airport**", created by Deepankur Kansal. It comprises **28,821 rows** and **23 features** related to flight take-off details from JFK Airport. The data is primarily aimed at predicting flight delays, particularly those influenced by weather conditions.

28,821 rows
23 features

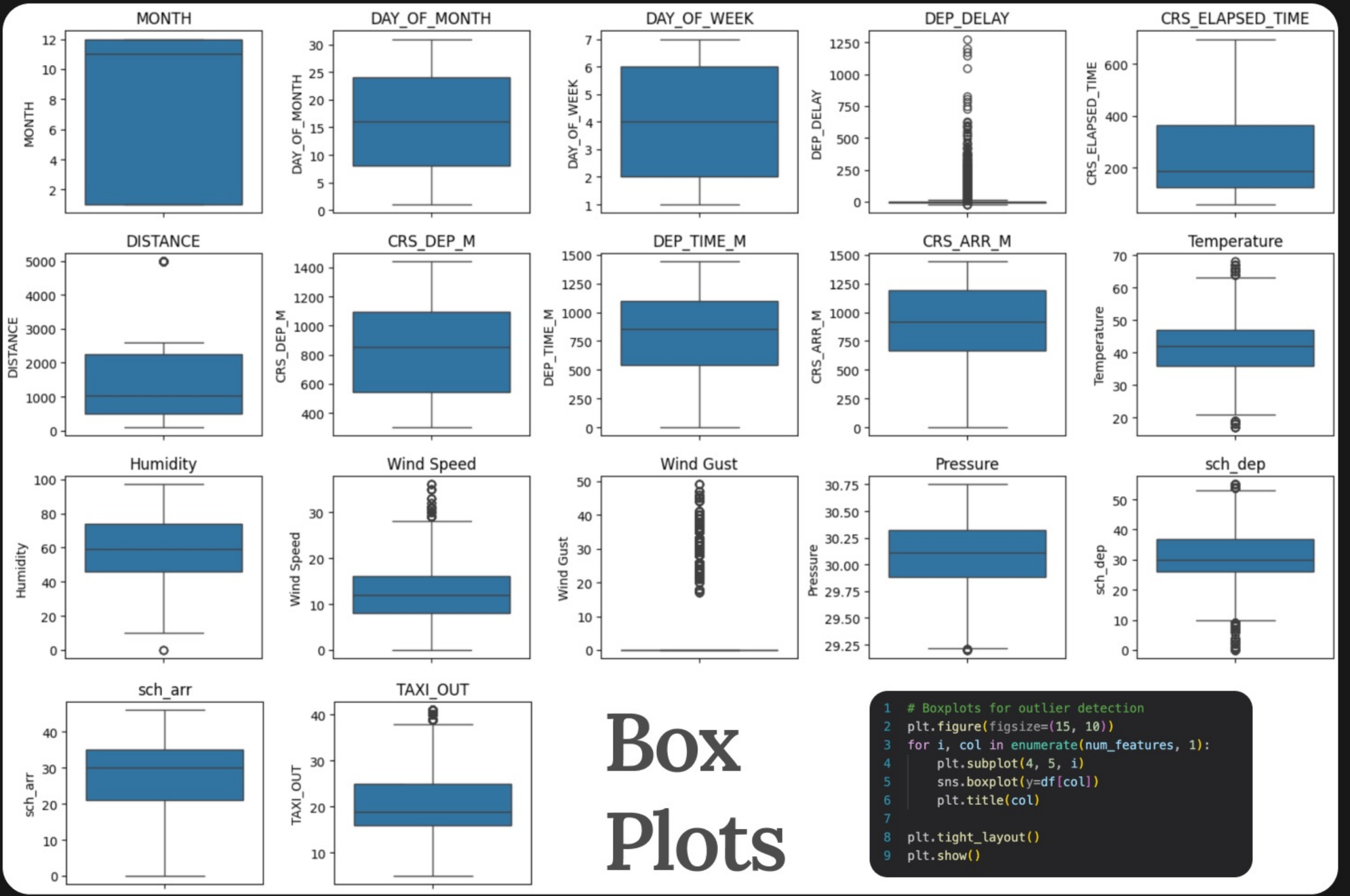
Weather and flight related data

published 4y ago
27.5k views in last month

flights between Nov 2019 - Dec 2020

Features

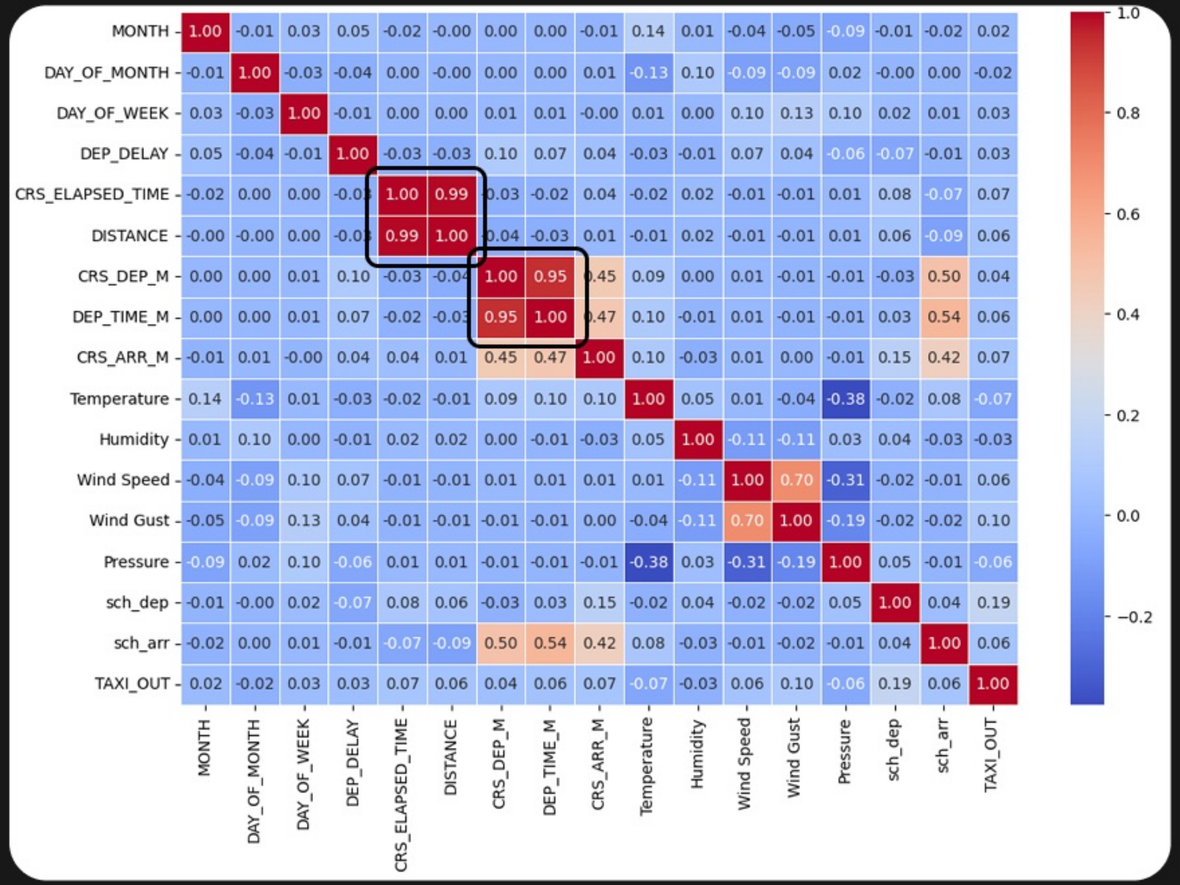
target feature - **DEP_DELAY**



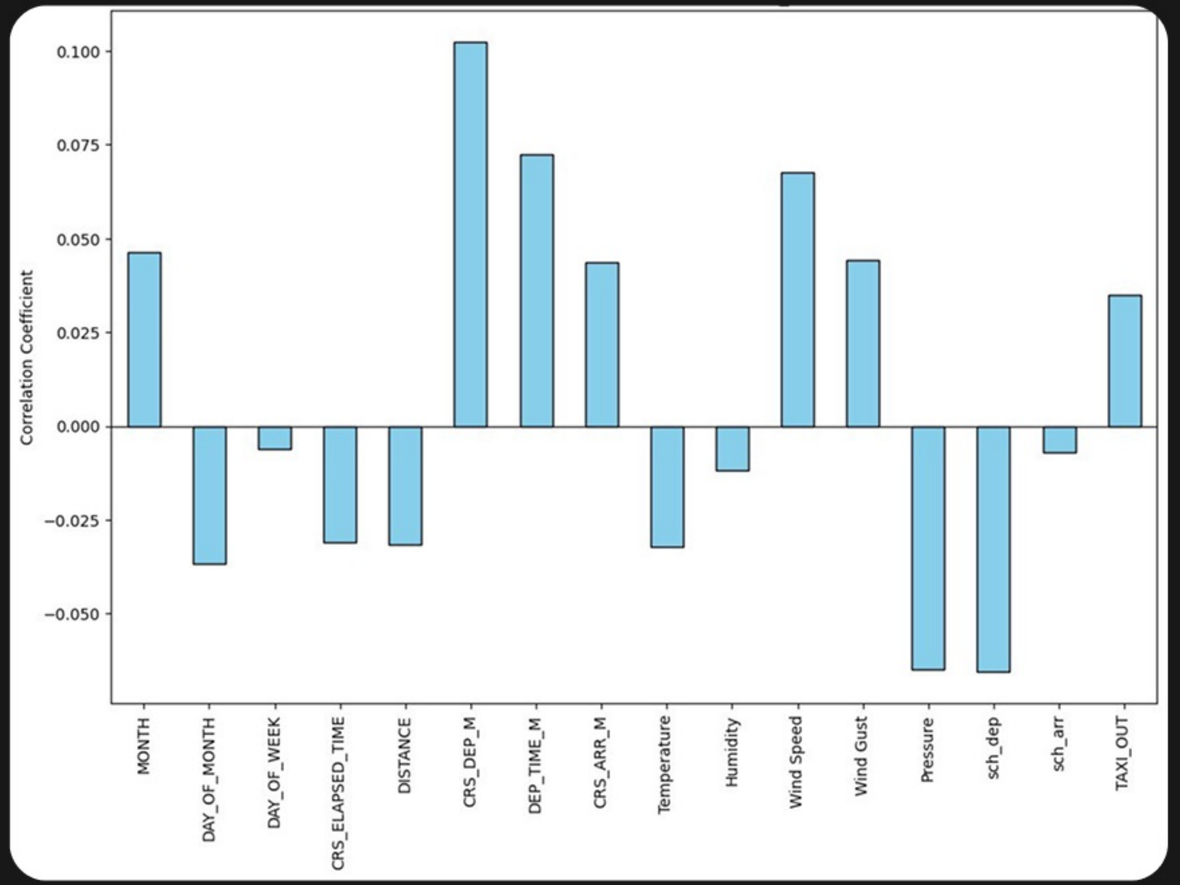
Box Plots

```python
# Boxplots for outlier detection
plt.figure(figsize=(15, 10))
for i, col in enumerate(num_features, 1):
    plt.subplot(4, 5, i)
    sns.boxplot(y=df[col])
    plt.title(col)

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

Correlation matrix of Numerical Features



Correlation of Other Features with 'DEP DELAY'

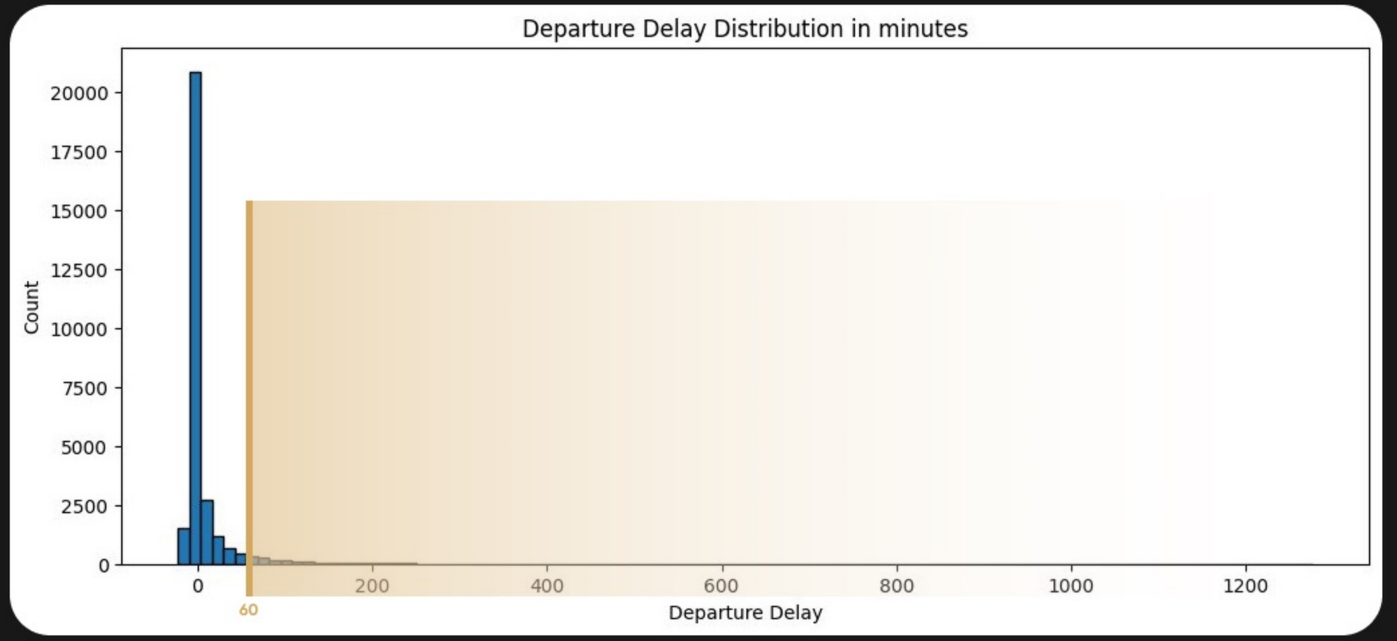


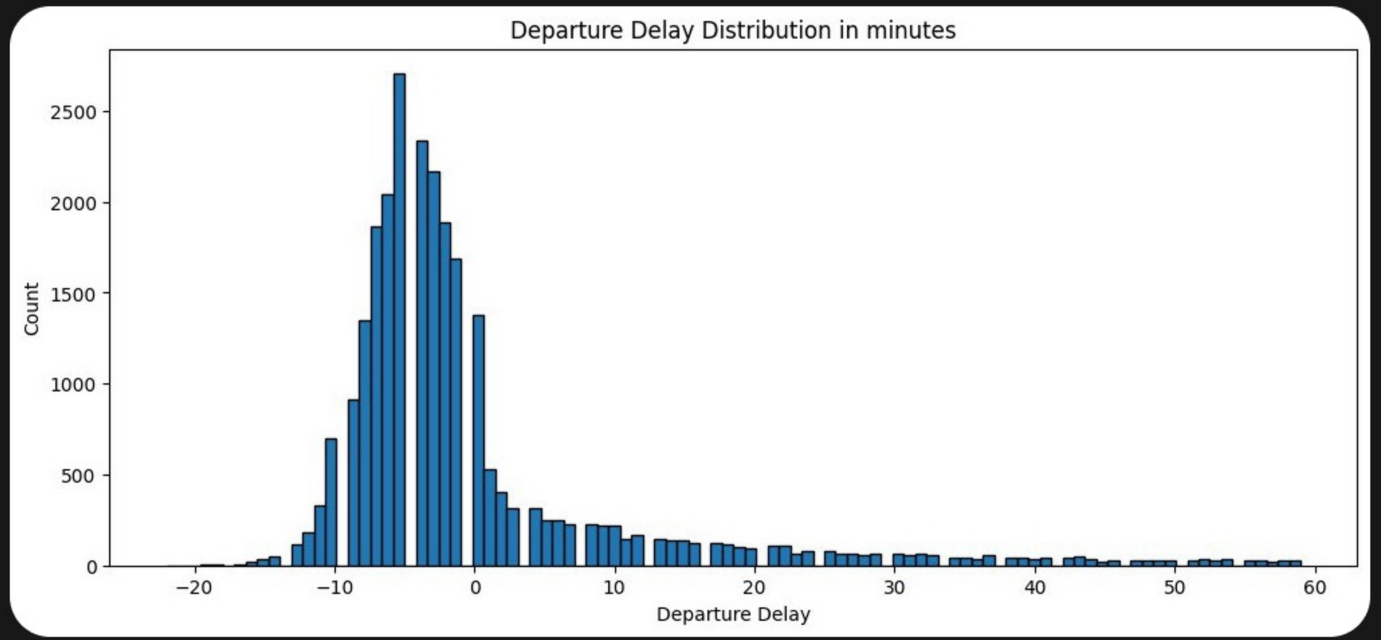
data preprocessing

Outlier Handling

> Since regression models are sensitive to outliers, 'DEP_DELAY' values greater than 60 minutes were removed.

28,821 rows →
27,473 data points





data preprocessing

Handling Corrupted Variables

'Dew Point' feature contained corrupted values:

- Converted the values to numerical and removed data points with NULL values

27,473 rows →
25,818 data points

```
[12]
1 dataset['Dew Point'].unique()
```

```
array(['34', '32', '33', '31', '30', '29', '67', '35', '36', '38', '39',
       '40', '41', '42', '28', '27', '37', '44', '45', '52', '53', '54',
       '56', '57', '59', '58', '50', '46', '43', '48', '49', '47', '21',
       '20', '22', '24', '23', '19', '18', '16', '13', '12', '14', '15',
       '17', '26', '25', '51', '55', '9\xa0', '6\xa0', '5\xa0', '8\xa0',
       '10', '4\xa0', '7\xa0', '3\xa0', '2\xa0', '11', '0\xa0', '-1',
       '1\xa0', '-2', '-3'], dtype=object)
```

```
[13]
1 dataset['Dew Point'] = pd.to_numeric(dataset['Dew Point'], errors='coerce')
2 dataset.isna().sum()['Dew Point']
```

```
1654
```

```
[14]
1 dataset = dataset.dropna()
2 dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 25818 entries, 0 to 28819
```

data preprocessing

Encoding Categorical Values

> The **'Condition'** and **'Wind'** features had **24** and **18 unique values**, respectively.
>
> - **Converted** these features to **categorical values**

17 features → 57 features

```
1  dataset['Condition'].nunique()
```

```
24
```

```
1  dataset['Wind'].nunique()
```

```
18
```

```
1  dataset = pd.get_dummies(dataset, columns=['Condition', 'Wind'])
2  dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 25818 entries, 0 to 28819
Data columns (total 57 columns):
```

Features &
Preprocessing

01

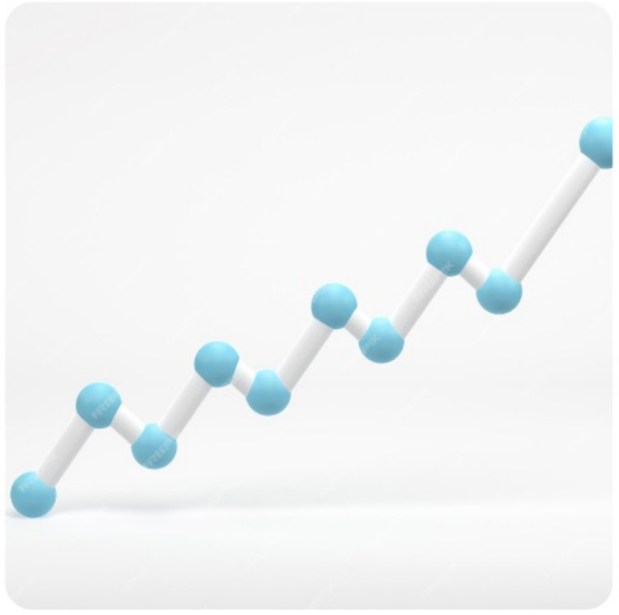
Methods
& Models

02

Results

03

Model Types used



Linear models

Ordinary Least Squares (OLS), Polynomial (Degree = 3)

Tree models

Decision Tree, Random Forest, Gradient Boosting (XGBoost)

Non-linear models

Support Vector Machines (SVM) and k-Nearest Neighbors (k-NN)

Methods

Settings

Train 80% test 20%

Linear regression models with this split

Coefficient of Determination

Measures the proportion of variance explained by the model

Root Mean Squared Error

Simulate and check correctness

Mean Absolute Error

Represents the average absolute difference between predicted and actual delays

OLS and 2-Polynomial Regression

```python
linear_models = {
    # "LinearRegression w/o Scaling": (make_pipeline(LinearRegression()), {}),
    "LinearRegression w/ Scaling": (make_pipeline(StandardScaler(), LinearRegression()), {}),
    "LinearRegression w/ Scaling & Poly": (make_pipeline(StandardScaler(), PolynomialFeatures(), LinearRegression()), {}),
    # "Ridge w/o Scaling": (make_pipeline(Ridge()), {"ridge__alpha": [0.01, 0.1, 1, 10, 100, 1000]}),
    # "Lasso w/o Scaling": (make_pipeline(Lasso()), {"lasso__alpha": [0.01, 0.1, 1, 10, 100, 1000]}),
    "Ridge w/ Scaling & Poly": (make_pipeline(StandardScaler(), PolynomialFeatures(), Ridge()), {"ridge__alpha": [0.01, 0.1, 1, 10, 100, 1000]}
    "Lasso w/ Scaling & Poly": (make_pipeline(StandardScaler(), PolynomialFeatures(), Lasso()), {"lasso__alpha": [0.01, 0.1, 1, 10, 100, 1000]}
}

```

Linear Models

```python
best_linear_models = model_tuning(linear_models)
```

```
Best hyperparams for 'LinearRegression w/ Scaling': {}

Best hyperparams for 'Ridge w/ Scaling': {'ridge__alpha': 1000}

Best hyperparams for 'Lasso w/ Scaling': {'lasso__alpha': 0.1}
```

Regularized Linear Regression

To mitigate overfitting and enhance generalization.

Ridge (L2 Regularization)
Introduces a penalty on large coefficients, improving stability and reducing variance

Lasso (L1 Regularization)
Encourages sparsity by shrinking some coefficients to zero, effectively performing feature selection

Capture nonlinear relationships

```python
tree_ensemble_models = {
    "DecisionTree": (DecisionTreeRegressor(), {"max_depth": [3, 5, 10, 15]}),
    "RandomForest": (RandomForestRegressor(), {"n_estimators": [50, 100, 200], "max_depth": [3, 5, 10, 15]}),
    "BaggingRegressor": (
        BaggingRegressor(estimator=DecisionTreeRegressor(), n_estimators=50),
        {"n_estimators": [50, 100, 200], "estimator__max_depth": [3, 5, 10, 15]}
    ),
    "GradientBoosting": (GradientBoostingRegressor(), {"n_estimators": [50, 100, 200], "learning_rate": [0.01, 0.1, 0.2]}),
    "AdaBoost": (AdaBoostRegressor(), {"n_estimators": [50, 100, 200], "learning_rate": [0.01, 0.1, 0.2]}),
}
```

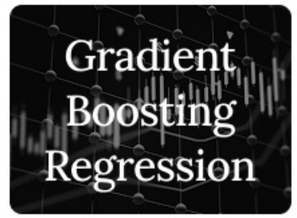
Tree-based and Ensemble Models

```python
best_tree_ensemble_models = model_tuning(tree_ensemble_models)
```

```
Best hyperparams for 'DecisionTree': {'max_depth': 5}

Best hyperparams for 'RandomForest': {'max_depth': 15, 'n_estimators': 200}

Best hyperparams for 'BaggingRegressor': {'estimator__max_depth': 15, 'n_estimators': 200}

Best hyperparams for 'GradientBoosting': {'learning_rate': 0.2, 'n_estimators': 200}

Best hyperparams for 'AdaBoost': {'learning_rate': 0.01, 'n_estimators': 50}
```

A non-parametric model that recursively splits data based on feature values

An ensemble learning method that reduces overfitting by averaging multiple decision trees



building decision trees in a multi-stage fashion, correcting errors of the previous stages using gradient descent

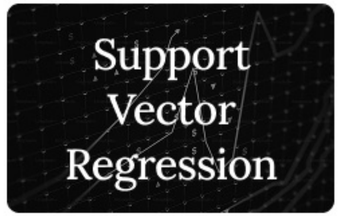
ensemble learning method combining multiple models trained on different subsets of the dataset

capturing nonlinearity in flight delays

Other models

```python
other_models = {
    # "Linear SVR w/o Scaling": (make_pipeline(StandardScaler(), SVR(kernel="linear")), {"svr__C": [0.1, 1, 10, 100, 1000],}),
    "Linear SVR w/ Scaling": (make_pipeline(StandardScaler(), SVR(kernel="linear")), {"svr__C": [0.1, 1, 10, 100, 1000],}),
    # "RBF SVR w/o Scaling": (make_pipeline(StandardScaler(), SVR(kernel="rbf")), {"svr__C": [0.1, 1, 10, 100, 1000],}),
    "RBF SVR w/ Scaling": (make_pipeline(StandardScaler(), SVR(kernel="rbf")), {"svr__C": [0.1, 1, 10, 100, 1000],}),
    # "KNN w/o Scaling": (make_pipeline(StandardScaler(), KNeighborsRegressor()), {"kneighborsregressor__n_neighbors": [3, 5, 7, 9]}),
    "KNN w/ Scaling": (make_pipeline(StandardScaler(), KNeighborsRegressor()), {"kneighborsregressor__n_neighbors": [3, 5, 7, 9]}),
}
```

```
Test Metrics for 'Linear SVR w/ Scaling' with params {'svr__C': 1}
MSE: 1.42e+02   RMSE: 11.9      MAE: 6.43      R2: -0.0554

Test Metrics for 'RBF SVR w/ Scaling' with params {'svr__C': 100}
MSE: 1.33e+02   RMSE: 11.5      MAE: 6.48      R2: 0.00978

Test Metrics for 'KNN w/o Scaling' with params {'kneighborsregressor__n_neighbors': 9}
MSE: 1.32e+02   RMSE: 11.5      MAE: 7.47      R2: 0.0161

Test Metrics for 'KNN w/ Scaling' with params {'kneighborsregressor__n_neighbors': 9}
MSE: 1.32e+02   RMSE: 11.5      MAE: 7.47      R2: 0.0161
```



Finds a hyperplane that best fits the data while minimizing errors and maximizing margins using support points

Predicts delays based on the average outcome of the k most similar observations

Preprocessing approaches

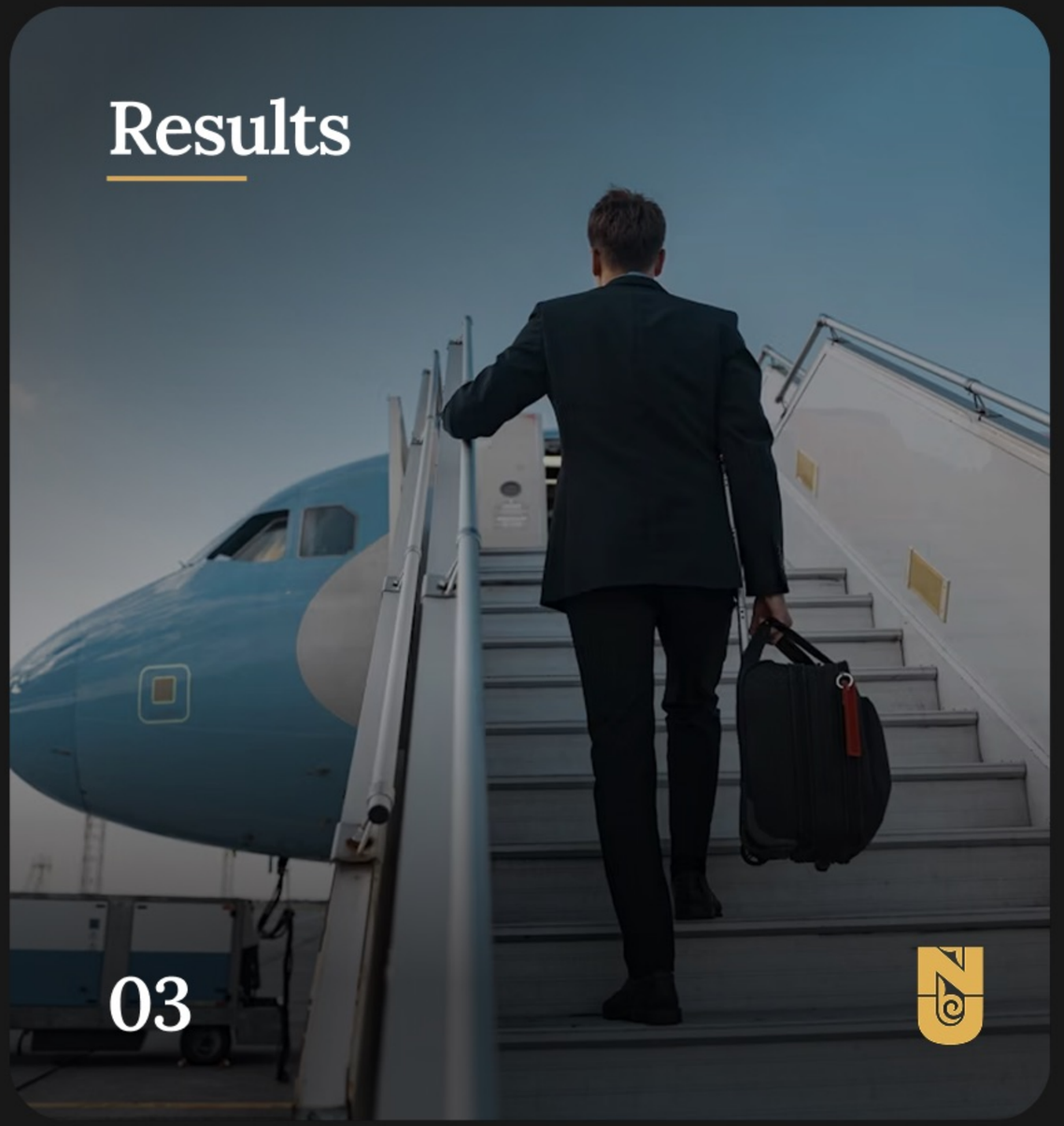
01

Models & Results

FAKE NEWS

02

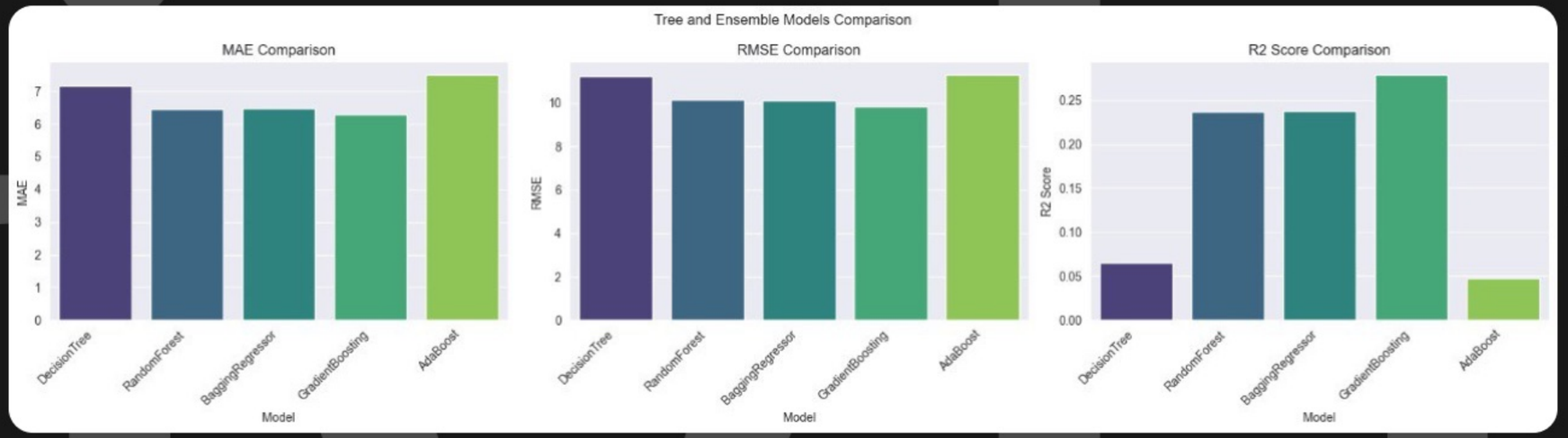
Results

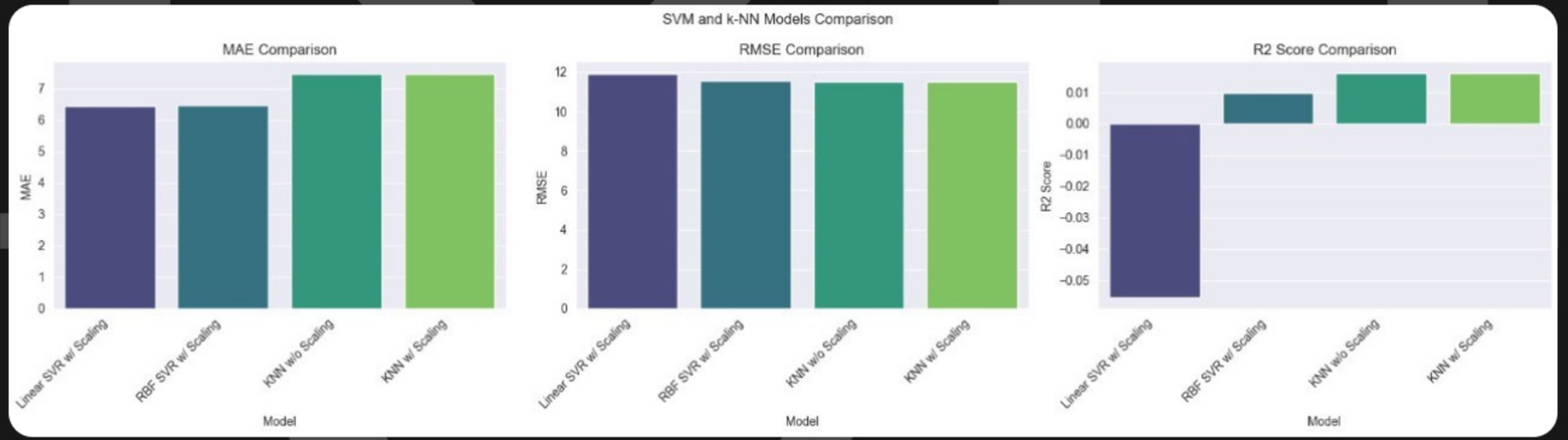


03

model results

Results





Results

Model	Our R^2 Score	Research Paper R^2 Score
Linear Regression	0.0395	0.0214
Ridge	0.0546	0.0212
Lasso	0.062	0.0083
Decision Tree	0.0651	-0.9243
Random Forest	0.238	-0.1021
Bagging	0.243	-
Gradient Boosting	0.278	-
AdaBoost	0.0464	-
Linear SVM	-0.0554	-0.0297
RBF SVM	0.00978	-
K-Nearest Neighbors	0.0161	-

TABLE I: Comparison of R^2 Scores Between Our Study and Research Paper after pre-processing

Model	Our R^2 Score	Research Paper R^2 Score
Linear Regression	0.7523	-
Ridge	0.7548	-
Lasso	0.7421	-
Decision Tree	0.7210	0.8986
Random Forest	0.8327	0.9629
Gradient Boosting	0.8456	-
XGBoosting	0.9018	0.9864
AdaBoost	0.8012	-
Polynomial SVM	0.04	-
K-Nearest Neighbors	0.6894	-

TABLE II: Comparison of R^2 Scores Between Our Study and Research Paper with all features

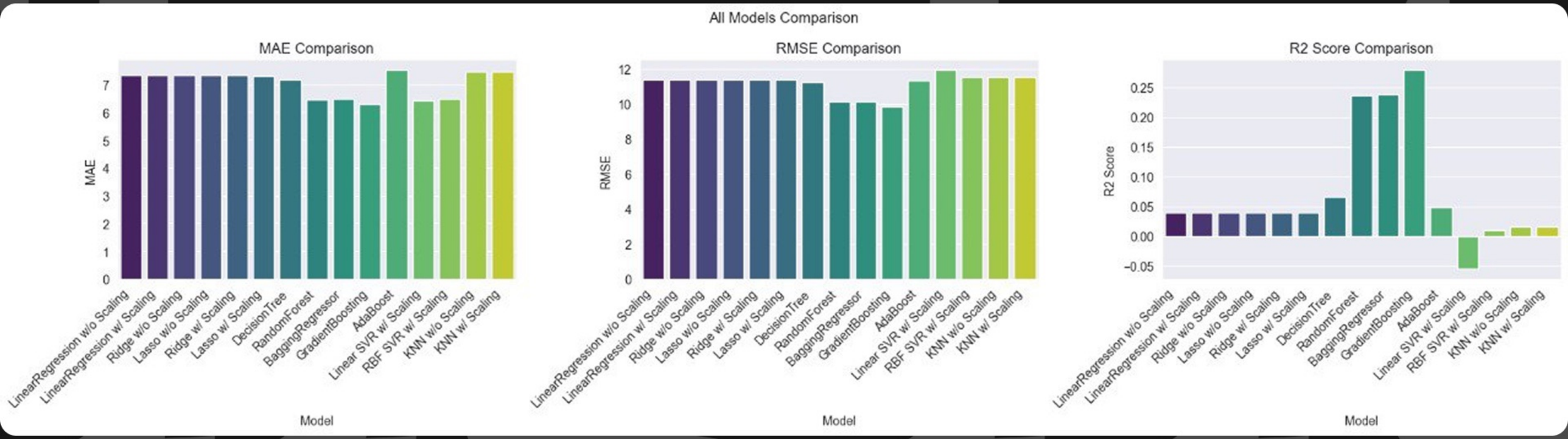
Trees total (Decision tree fastest, random forest slowest, ~17m, xgboost fast): 25m 4.5s

Linear total (all fast): 17.3s

kNN total: 17.1s

SVM total (linear kernel slowest, ~70m, poly kernel fast, RBF kernel mid, 15-20m): 95m 7.5s

Results


All Models Comparison
MAE Comparison
MAE
Model
RMSE Comparison
RMSE
Model
R2 Score Comparison
R2 Score
Model
LinearRegression w/o Scaling
LinearRegression w/ Scaling
Ridge w/o Scaling
Lasso w/o Scaling
Ridge w/ Scaling
Lasso w/ Scaling
DecisionTree
RandomForest
BaggingRegressor
GradientBoosting
AdaBoost
Linear SVR w/ Scaling
RBF SVR w/ Scaling
KNN w/o Scaling
KNN w/ Scaling

Results

Model	Our R^2 Score	Research Paper R^2 Score
Linear Regression	0.0395	0.0214
Ridge	0.0546	0.0212
Lasso	0.062	0.0083
Decision Tree	0.0651	-0.9243
Random Forest	0.238	-0.1021
Bagging	0.243	-
Gradient Boosting	0.278	-
AdaBoost	0.0464	-
Linear SVM	-0.0554	-0.0297
RBF SVM	0.00978	-
K-Nearest Neighbors	0.0161	-

TABLE I: Comparison of R^2 Scores Between Our Study and Research Paper after pre-processing

Model	Our R^2 Score	Research Paper R^2 Score
Linear Regression	0.7523	-
Ridge	0.7548	-
Lasso	0.7421	-
Decision Tree	0.7210	0.8986
Random Forest	0.8327	0.9629
Gradient Boosting	0.8456	-
XGBoosting	0.9018	0.9864
AdaBoost	0.8012	-
Polynomial SVM	0.04	-
K-Nearest Neighbors	0.6894	-

TABLE II: Comparison of R^2 Scores Between Our Study and Research Paper with all features

Trees total (Decision tree fastest, random forest slowest, ~17m, xgboost fast): 25m 4.5s

Linear total (all fast): 17.3s

kNN total: 17.1s

SVM total (linear kernel slowest, ~70m, poly kernel fast, RBF kernel mid, 15-20m): 95m 7.5s

with all the features

Results