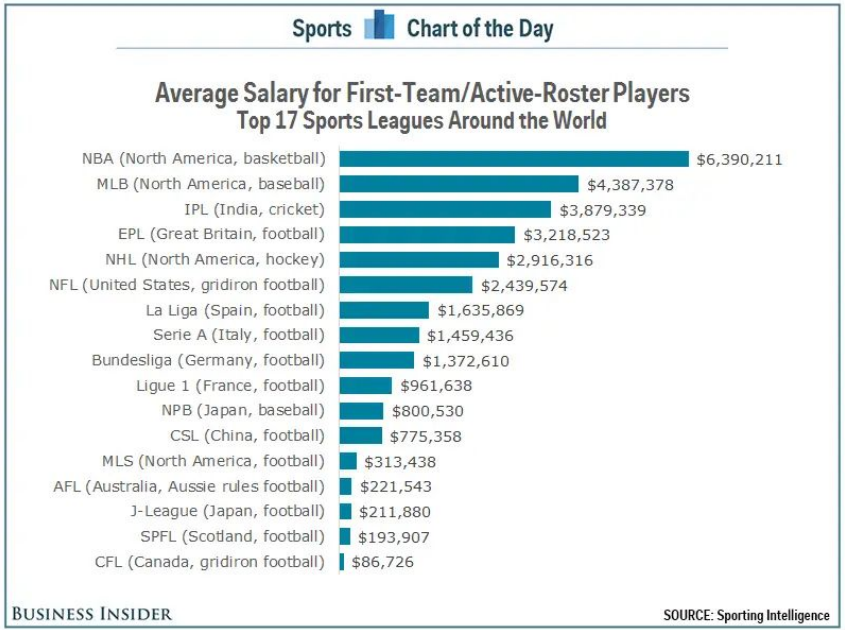
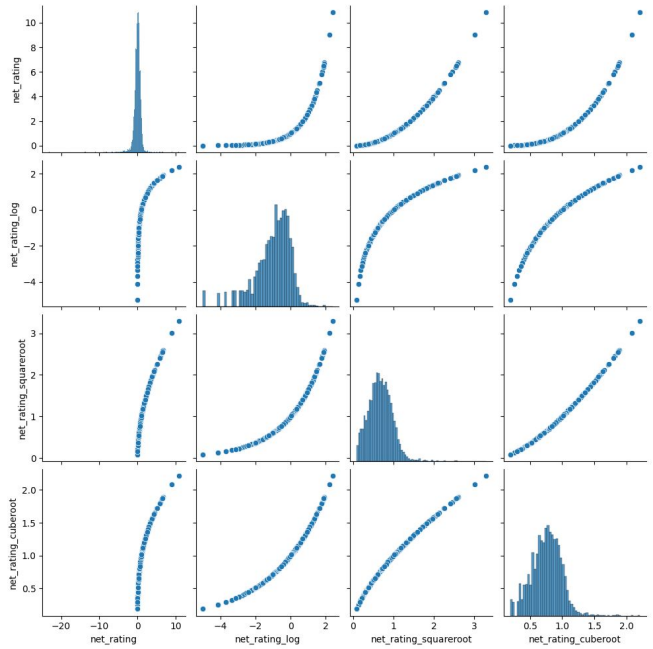
Predicting NBA Player Net Ratings and Draft Positions (Group 19)

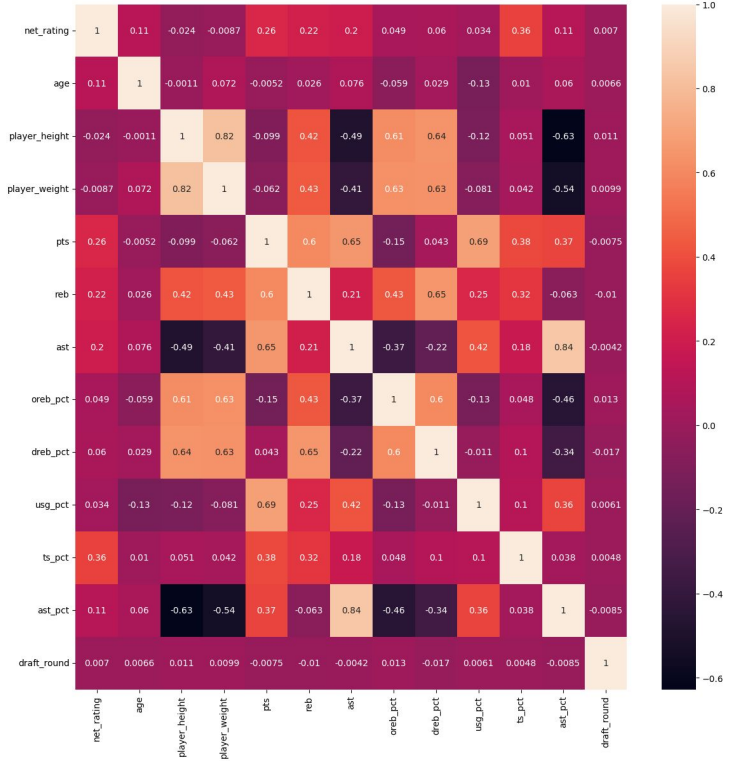
Raghuram Palaniappan
Jeff Lee
Koby Lieu
Jasper Liu

Introduction



Exploratory Data Analysis





Key questions

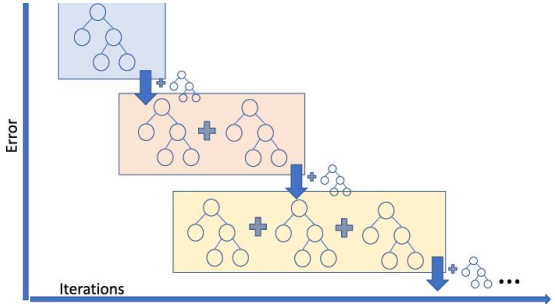
1. How can we predict net rating from offensive and defensive metrics?
2. Does the draft position of a player correlate with their performance?

Methodology

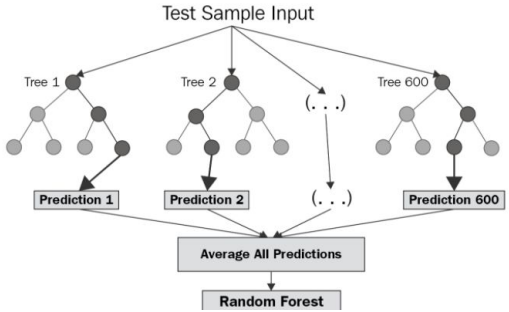
Regression Models

1. Linear Regression
2. Support Vector Regression
3. Random Forest Regression
4. Gradient Boosting Regression
5. Lasso Regression
6. Ridge Regression

Gradient Boosting Regression



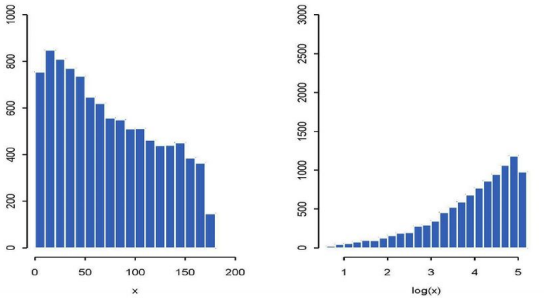
Random Forest Regression



Should we remove outliers?

Should we transform the data?

Log Transformed Data
Example ->



Results

Best Model: Random Forest Regression

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

# Set up the parameter grid to search over
param_grid = {'n_estimators': [100, 200, 500],
              'max_depth': [None, 10, 20],
              'max_features': ['auto', 'sqrt']}

# Instantiate a random forest regression model
rf_model = RandomForestRegressor(random_state=42)

# Create a GridSearchCV object to find the best hyperparameters
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=5)

# Fit the GridSearchCV object to the training data
grid_search.fit(X_train, y_train)

# Best GridSearchCV Model
y_pred = grid_search.predict(X_test)

# Calculate the model's performance metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared: {r2:.2f}")
```

MSE and R^2

```
Mean Squared Error: 45.68
R-squared: 0.73
```

We want to implement the model through Flask

Results

Random Forest Model Function

```
# Random Forest Model Function
def predict_net_rating(age, player_height, player_weight, pts, reb, ast, oreb_pct, dreb_pct, usg_pct, ts_pct, ast_pct):
    # Load the data
    X = players_df[["age", "player_height", "player_weight", "pts", "reb", "ast",
                    "oreb_pct", "dreb_pct", "usg_pct", "ts_pct", "ast_pct"]]
    y = players_df["net_rating"]

    # Split 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)

    # Scale and normalize the data
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Set up the parameter grid to search over
    param_grid = {'n_estimators': [100, 200, 500],
                  'max_depth': [None, 10, 20],
                  'max_features': ['auto', 'sqrt']}

    # Instantiate a random forest regression model
    rf_model = RandomForestRegressor(random_state=42)

    # Create a GridSearchCV object to find the best hyperparameters
    grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=5)

    # Fit the GridSearchCV object to the training data
    grid_search.fit(X_train, y_train)

    # Make a prediction using the trained model
    input_data = [[age, player_height, player_weight, pts, reb, ast, oreb_pct, dreb_pct, usg_pct, ts_pct, ast_pct]]
    input_data_scaled = scaler.transform(input_data)
    net_rating = grid_search.predict(input_data_scaled)[0]

    return net_rating
```

Thank you, Questions?