Golf Course Manager

ML-Powered Course Management with Big Data Processing

Thomas Kulch
DS5110 - Final Project

Introduction

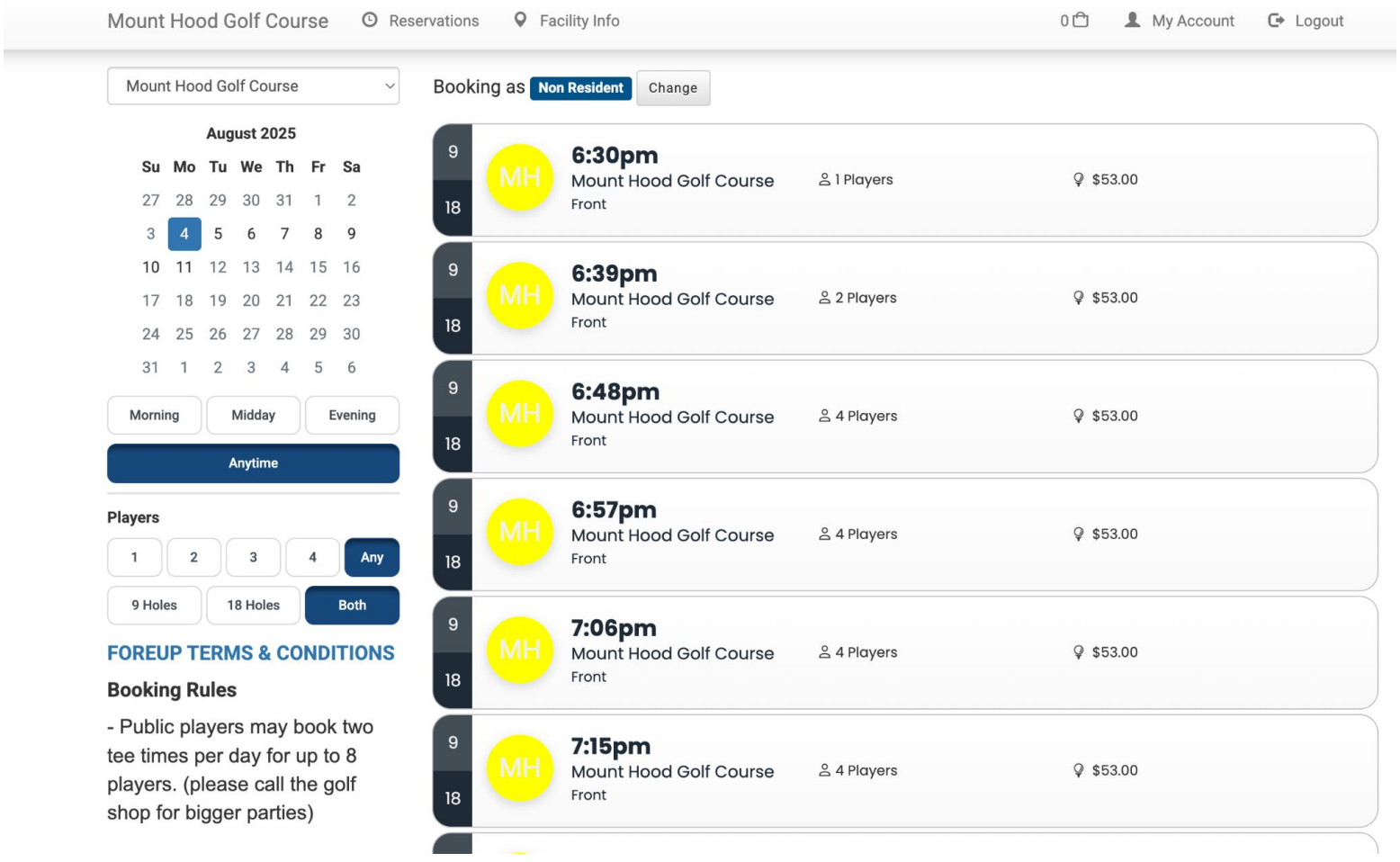
Objectives and Goals

- Build end-to-end data pipeline for managing a golf course
- Develop ML models for score prediction
- Demonstrate big data processing with Spark
- Create interactive web application for bookings and analytics

Scope

- 6000+ golf records and 10 years of weather data from Kaggle
- Booking system with dynamic pricing and score prediction
- Exploratory analysis in pandas
- Statistical analysis in R for feature selection
- PostgreSQL database with triggers and functions
    - 4 tables: players, rounds, bookings, weather

Typical Golf Booking Page



Architecture Overview

Technologies

- Pandas
- Plotly
- R
- PySpark
- PostgreSQL
- Scikit-learn
- Flask
- HTML

```
golf-analytics-platform/
├── data/
│   ├── raw/                    # Original datasets
│   ├── processed/              # Cleaned data used for analysis
│   └── models/                 # Trained ML models
├── documents/                  # Project reports/presentation
├── database/
│   ├── create_database.sql     #  Database create script
│   ├── init.sql                # Database Schema and Functions
│   ├── queries.sql             # Queries for Database
│   ├── user_permissions.sql    # Database user permissions
├── notebooks/
│   ├── 01_eda.ipynb
│   ├── 02_sda.rmd
│   ├── 03_feature_eng.ipynb
│   └── 04_model_dev.ipynb
├── src/
│   ├── templates/
│       ├── index.html
│       └── dashboard.html
│   ├── app.py                 # Flask web application
│   ├── data_processing.py     # Spark ETL pipeline
│   ├── database.py            # Database operations
│   └── round_booking.py       # Booking system logic
├── scripts/
│   ├── conversions.py          # Simple conversions used in EDA
│   ├── feature_engineering.py # Feature engineering functions
│   └── run_pipeline.py         # Script that runs Spark pipeline
├── requirements.txt
└── README.md
```

Full Data Pipeline

Pseudo Code: Predictions and Pricing

```
# Pseudo Code - Score Prediction
def predict_score(player_features, weather_data):
    features = feature_engineering(player_features, weather_data)
    scaled_features = scaler.transform(features)
    prediction = linear_model.predict(scaled_features)
    return prediction

# Dynamic Pricing Algorithm
def calculate_price(base_price, weather, demand, player_skill):
    price = base_price
    price += weather_adjustment(weather)
    price += demand_multiplier(day_of_week, time)
    price += skill_based_discount(player_skill)
    return max(price, minimum_price)
```

Pseudo Code: PySpark

```
# PySpark Data Processing Pipeline
def spark_etl_pipeline():
    spark = SparkSession.builder.appName("Golf").getOrCreate()

    # Raw Data Reading
    golf_df = spark.read.csv("golf_data.csv", header=True)
    weather_df = spark.read.csv("weather_data.csv", header=True)

    # Data Cleaning
    cleaned_golf = golf_df.filter(col("score") > 0) \
                          .withColumn("round_date", to_date(col("date")))

    # Write to PostgreSQL
    player_stats.write.jdbc(url=postgres_url, table="player_summary")
```

Pseudo Code: DB Functions and Triggers

```
-- Trigger: Auto-update player handicap after new round
CREATE OR REPLACE FUNCTION calculate_handicap_for_player(p_player_id INTEGER)
RETURNS FLOAT AS $$
DECLARE
    average_differential FLOAT;
BEGIN

    -- calculate average differential from most recent rounds
    SELECT AVG(ROUND(((score - 67.3) * 113.0 / 119.0)::NUMERIC, 1)) INTO average_differential
    FROM (
        SELECT score
        FROM rounds
        WHERE player_id = p_player_id
        ORDER BY round_date DESC
        LIMIT 20
    ) recent_scores;

    RETURN ROUND((average_differential::NUMERIC * 0.96), 1);

END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER round_inserted_trigger
    AFTER INSERT ON rounds
    FOR EACH ROW EXECUTE FUNCTION update_player_handicap();
```

Hours Spent

Week 1: Data exploration and Statistical Analysis (10 hours)

Week 2-3: Spark pipeline (20 hours)

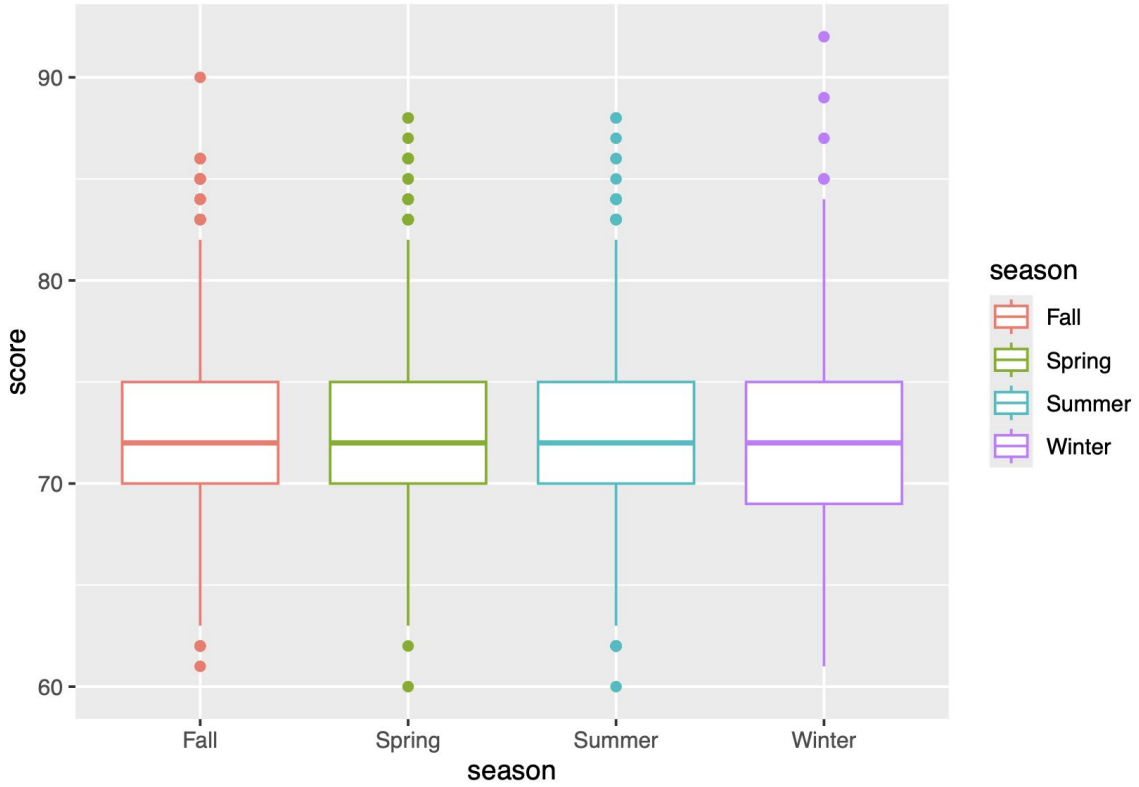
Week 4: Database integration (15 hours)

Week 5-6: Feature engineering and ML model development (12 hours)

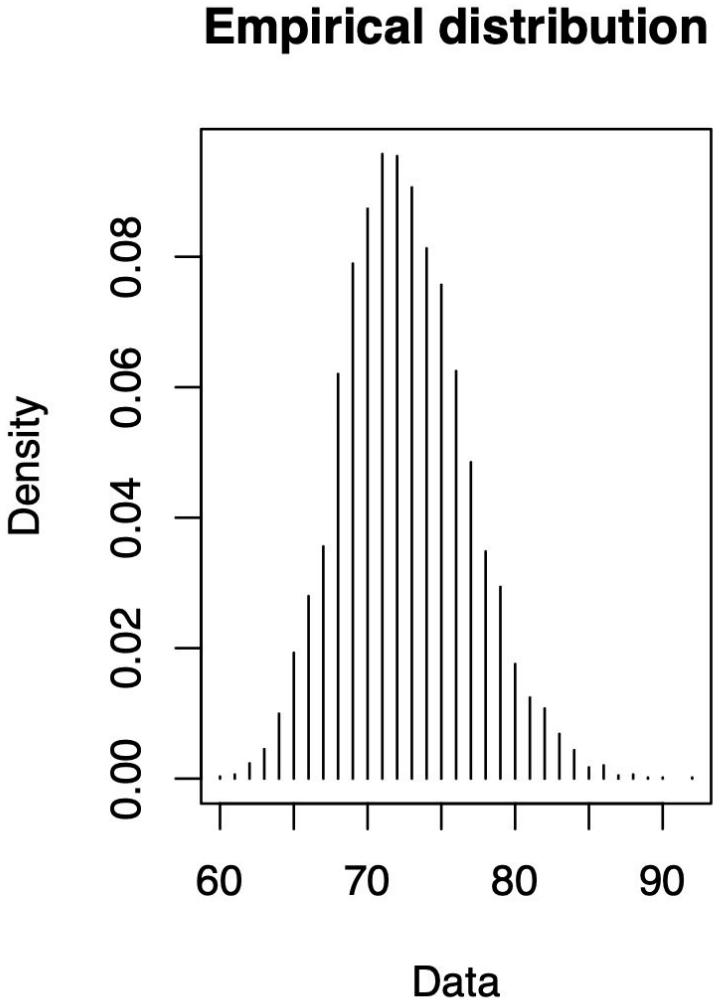
Week 7: Flask app (20 hours)

Week 8: Documentation (6 hours)

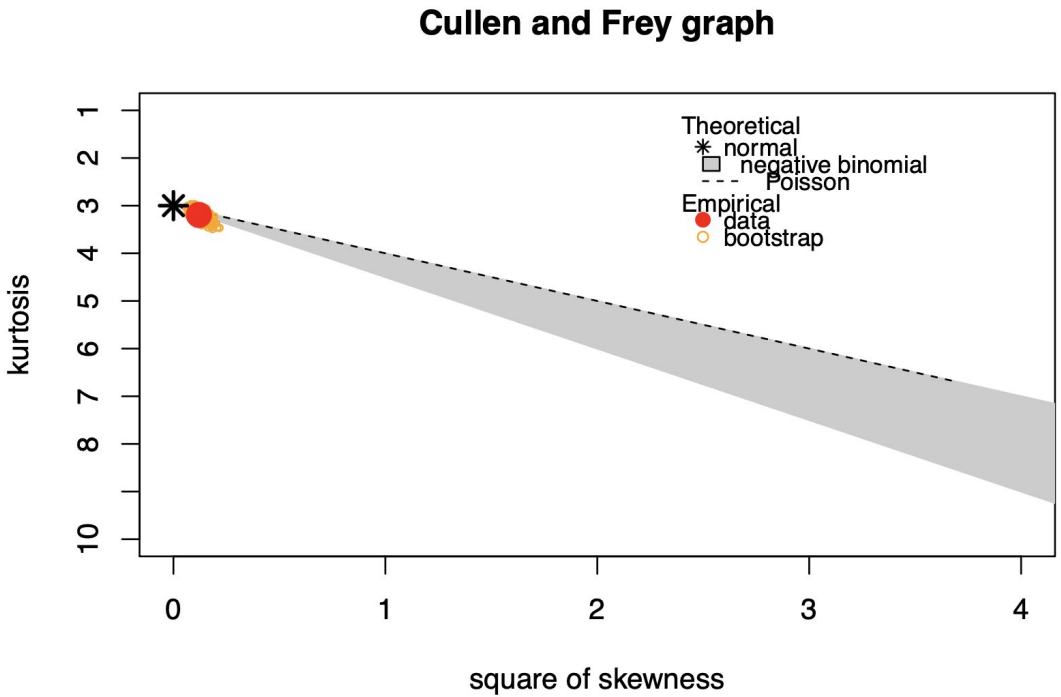
Findings: Scores by Season



Scoring Distribution



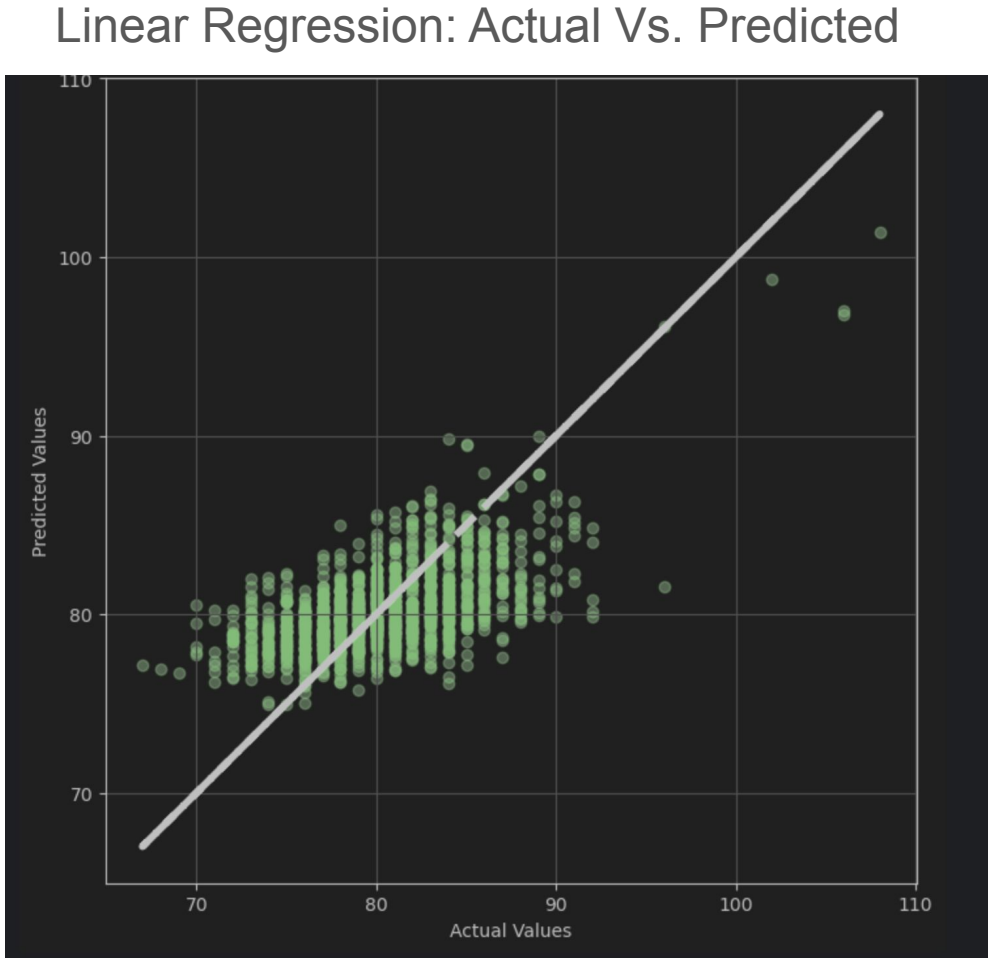
Model Selection



- Confirms Normal distribution
- Linear Regression would likely be optimal model

Model Development

- Linear Regression had 35% accuracy
- Tested models: Ridge, RF, GB
- Gradient Boosting Regressor had a 46% accuracy, beating out other models
- GBR was used as final model



Limitations

- Limited Golf Data
- Needed to manipulate data to get any result
- Feature Limitations, resulting in low ML model accuracy
- Requires consistent data collection for effectiveness
- Model trained on modified professional golfer data

Achievements

- Successfully built end-to-end big data pipeline
- Built cohesive relational db with triggers
- Created web application
- Trained and deployed a ML model

Future Work

- Allow players to enter rounds on Web Platform
- Hypertuning features for more accurate ML model
- Secure Logins for Web App
- Host Web App and Database on Cloud
- Expand ETL to allow data streaming
- Make dashboards interactive

App Demo

Welcome back, Thomas Kulch!

Book a Tee Time

Date:

mm/dd/yyyy

Tee Time:

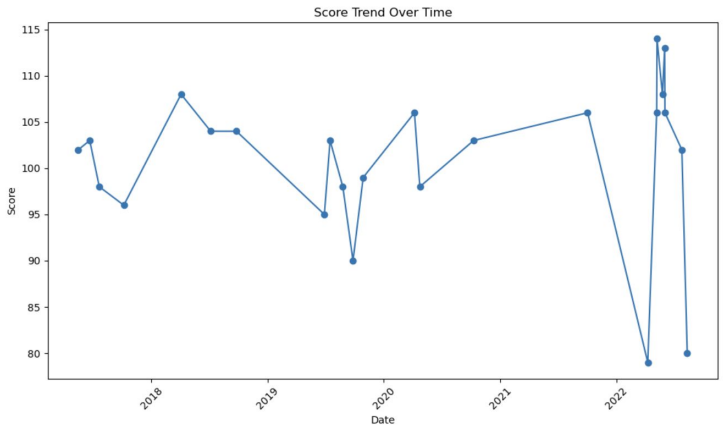
6:00 AM

Add Golf Cart (+$20)

Book Tee Time

Your Performance Analysis

Score Trend Over Time



Questions?