Supervised Learning – Classification and Regression

Due: July 21 (Monday) | **Total Points:** 100
Submission: Share a link to your completed Google Colab notebook (set sharing to *"University of Colorado Boulder"*)

🎯 What You'll Learn

- Choose the right ML model for a prediction task
- Implement and evaluate regression and classification pipelines
- Compare model performance using proper metrics
- Visualize and interpret model behavior (e.g. decision trees)
- Explain your modeling decisions clearly in Markdown

⚙️ Project Structure

You will complete **two prediction tasks** using two different datasets:

1. **Regression Task** – Predict a numeric value
2. **Classification Task** – Predict a binary outcome

You'll:

- Build pipelines with preprocessing
- Train multiple models
- Evaluate performance with appropriate metrics
- Reflect on tradeoffs between model types

📁 Dataset Choices (Pick One for Each Task)

Regression Datasets (predict a number):

- [Your dataset here] – should contain numeric target like price, ratings, etc.

https://www.kaggle.com/datasets/baydarenes/customer-attributes-and-purchase-propensity

Classification Datasets (predict 0 or 1):

- [Your dataset here] – should contain a binary target like success/failure, yes/no, etc.

https://www.kaggle.com/datasets/pratyushpuri/sports-betting-predictive-analysis-dataset

> ***You may not use any datasets we covered in class***

🔵 Regression Task

Part 1 – EDA + Task Definition (10 pts)

- Load the dataset
- Do `.info()`, `.describe()`, `.value_counts()` as appropriate
- Write:
  - What's your target variable?
  - What features might help with prediction?

In [1]:

```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (accuracy_score, confusion_matrix, classification_report,
                             RocCurveDisplay, PrecisionRecallDisplay, mean_squared_error, r2_score)
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[1], line 1
----> 1 from google.colab import drive
      2 drive.mount('/content/drive', force_remount=True)
      4 import pandas as pd

ModuleNotFoundError: No module named 'google'
```

In [ ]:

```
file_path = '/content/drive/MyDrive/info_4604/datasets/Customer_Attributes_and_Purchase_Propensity.csv'
df1 = pd.read_csv(file_path)

print(df1.head())
print(df1.info())
print(df1.describe())
print(df1.value_counts())
```

```
   Age  Income      City Marital_Status     Score  Purchased
0   56   24000   Houston       Divorced  0.479897          0
1   46   90588   Chicago         Single  0.685968          1
2   32  113610   Houston       Divorced  0.017380          0
3   60  117856  New York        Married  0.323131          0
4   25   58304   Houston         Single  0.973098          0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Age             500 non-null    int64  
 1   Income          500 non-null    int64  
 2   City            500 non-null    object 
 3   Marital_Status  500 non-null    object 
 4   Score           500 non-null    float64
 5   Purchased       500 non-null    int64  
dtypes: float64(1), int64(3), object(2)
memory usage: 23.6+ KB
None
              Age        Income       Score   Purchased
count  500.000000     500.00000  500.000000  500.000000
mean    41.278000   67210.16800    0.492660    0.534000
std     13.389072   28908.77426    0.282769    0.499342
min     18.000000   20055.00000    0.000333    0.000000
25%     30.000000   42534.25000    0.242372    0.000000
50%     42.000000   64906.50000    0.501602    1.000000
75%     52.000000   93071.25000    0.742617    1.000000
max     64.000000  119835.00000    0.995480    1.000000
Age  Income  City         Marital_Status  Score     Purchased
64   108726  Chicago      Single          0.438172  1            1
18   26168   Houston      Single          0.053858  1            1
     27561   New York     Single          0.320060  0            1
     32910   Phoenix      Single          0.091871  0            1
63   31637   Chicago      Divorced        0.495452  1            1
                                                                ..
18   80566   Los Angeles  Single          0.954660  1            1
     80100   Chicago      Married         0.829332  1            1
     51921   Chicago      Divorced        0.974165  0            1
     46962   New York     Divorced        0.515175  1            1
     42574   Houston      Married         0.068713  1            1
Name: count, Length: 500, dtype: int64
```

My target variable will be Actual_Winner using features like Predicted_Winner, Home_Team, Away_Team, Home_Team_Odds, Away_Team_odds, and Draw_Odds.

Part 2 – Preprocessing + Pipeline (20 pts)

- Identify:
  - Categorical vs numeric features
  - Features with missing values
- Build a full preprocessing pipeline:
  - `SimpleImputer`, `OneHotEncoder`, `StandardScaler` as needed
- Wrap in a full `Pipeline`

In Markdown, explain:

- Why you chose each preprocessing step
- Any assumptions you made about the data

In [ ]:

```
df1 = df1.dropna()

target = 'Age'
X = df1.drop(columns=[target])
y = df1[target]

numeric_cols = X.select_dtypes(include=['float64', 'int64']).columns.tolist()
categorical_cols = X.select_dtypes(include=['object', 'category']).columns.tolist()

print(numeric_cols)
print(categorical_cols)

missing_features = df1.columns[df1.isnull().any()]

print(missing_features)

# We can see from the print value below that there are no missing values, which is congruent with the information provided on the Kabble page for this datas
```

```
['Income', 'Score', 'Purchased']
['City', 'Marital_Status']
Index([], dtype='object')
```

In [ ]:

```
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_cols),
        ('cat', categorical_transformer, categorical_cols)
    ])

linear_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('regressor', LinearRegression())
])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

linear_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', LinearRegression())
])

linear_pipeline.fit(X_train, y_train)

y_pred_linear = linear_pipeline.predict(X_test)

r2 = r2_score(y_test, y_pred_linear)
mse = mean_squared_error(y_test, y_pred_linear)
rmse = mse ** 0.5
print(f"R2: {r2:.3f}")
print(f"MSE: {mse:.3f}")
```

```
R2: -0.029
MSE: 184.398
```

Part 3 – Model Training + Evaluation (30 pts)

- Train **two models** (Linear vs Tree)
- Evaluate:
  - For regression: `mean_squared_error`, `r2_score`
- Show feature importances if applicable
- Use `train_test_split` and/or `cross_val_score` to assess generalization

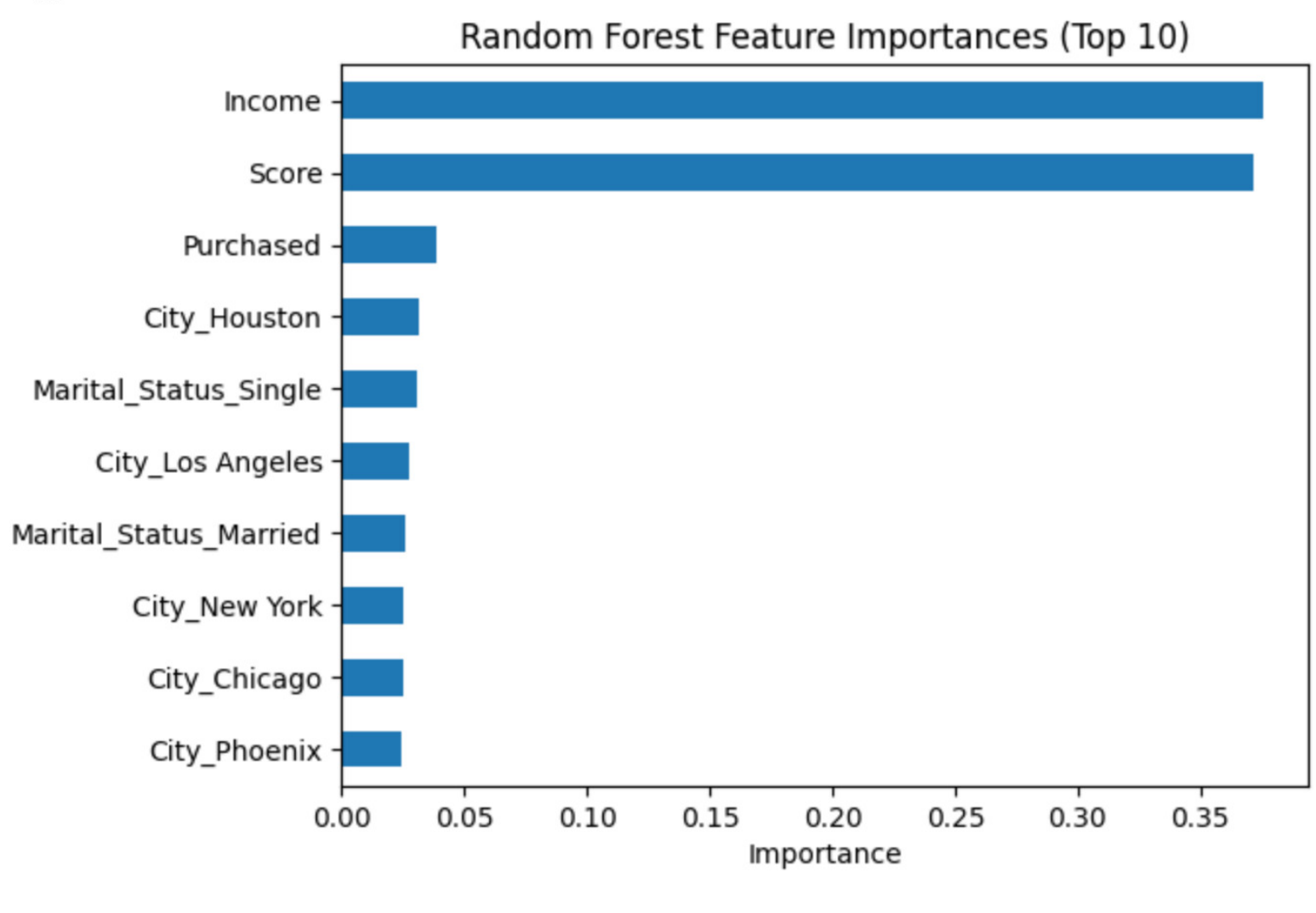
In Markdown:

- Compare model performance
- Which model is better and why?
- Do your results make sense based on EDA?

In [ ]:

```
# --- Train/test split (already defined earlier, but repeat here for clarity) ---
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# --- 1. Linear Regression model ---
lin_model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', LinearRegression())
])
lin_model.fit(X_train, y_train)
y_pred_lin = lin_model.predict(X_test)
lin_mse = mean_squared_error(y_test, y_pred_lin)
lin_r2 = r2_score(y_test, y_pred_lin)

# Optional cross-validation (returns negative MSE scores, so take absolute):
lin_cv_mse = -cross_val_score(lin_model, X, y, cv=5, scoring='neg_mean_squared_error')
lin_cv_r2 = cross_val_score(lin_model, X, y, cv=5, scoring='r2')

print(f"LinearRegression Test MSE: {lin_mse:.3f}")
print(f"LinearRegression Test R2:  {lin_r2:.3f}")
print(f"LinearRegression CV MSE (mean±std): {lin_cv_mse.mean():.3f} ± {lin_cv_mse.std():.3f}")
print(f"LinearRegression CV R2 (mean±std):  {lin_cv_r2.mean():.3f} ± {lin_cv_r2.std():.3f}")

# --- 2. Random Forest Regressor (tree-based model) ---
rf_model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', RandomForestRegressor(
        n_estimators=300,
        random_state=42,
        n_jobs=-1
    ))
])
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)
rf_mse = mean_squared_error(y_test, y_pred_rf)
rf_r2 = r2_score(y_test, y_pred_rf)

rf_cv_mse = -cross_val_score(rf_model, X, y, cv=5, scoring='neg_mean_squared_error')
rf_cv_r2 = cross_val_score(rf_model, X, y, cv=5, scoring='r2')

print(f"\nRandomForest Test MSE: {rf_mse:.3f}")
print(f"RandomForest Test R2:  {rf_r2:.3f}")
print(f"RandomForest CV MSE (mean±std): {rf_cv_mse.mean():.3f} ± {rf_cv_mse.std():.3f}")
print(f"RandomForest CV R2 (mean±std):  {rf_cv_r2.mean():.3f} ± {rf_cv_r2.std():.3f}")

# --- Feature importances from Random Forest ---
# Extract the one-hot feature names
ohe = rf_model.named_steps['preprocessor'].named_transformers_['cat']
cat_feature_names = ohe.get_feature_names_out(categorical_cols)
all_feature_names = numeric_cols + list(cat_feature_names)

importances = rf_model.named_steps['regressor'].feature_importances_
feat_imp = pd.Series(importances, index=all_feature_names).sort_values(ascending=False)
print("\nRandom Forest Feature Importances:\n", feat_imp.head(15))

# (Optional) simple bar plot
import matplotlib.pyplot as plt
feat_imp.head(10).plot(kind='barh')
plt.gca().invert_yaxis()
plt.xlabel("Importance")
plt.title("Random Forest Feature Importances (Top 10)")
plt.show()
```

```
LinearRegression Test MSE: 184.398
LinearRegression Test R2:  -0.029
LinearRegression CV MSE (mean±std): 182.010 ± 5.506
LinearRegression CV R2 (mean±std):  -0.027 ± 0.027

RandomForest Test MSE: 194.896
RandomForest Test R2:  -0.088
RandomForest CV MSE (mean±std): 201.204 ± 16.232
RandomForest CV R2 (mean±std):  -0.133 ± 0.047

Random Forest Feature Importances:
 Income                     0.375355
Score                     0.371806
Purchased                 0.038344
City_Houston              0.031497
Marital_Status_Single     0.030912
City_Los Angeles          0.027572
Marital_Status_Married    0.026190
City_New York             0.025455
City_Chicago              0.024979
City_Phoenix              0.024641
Marital_Status_Divorced   0.023250
dtype: float64
```



Linear Regression outperformed Random Forest in this task, achieving a lower test MSE (184 vs 195) and a less negative R² (−0.029 vs −0.088), with cross-validation showing the same pattern—Linear Regression CV MSE ≈ 182 vs RF's ≈ 201 and CV R² ≈ −0.027 vs ≈ −0.133. Negative R² in both cases indicates neither model predicts better than using the mean age, and the Random Forest's worse performance suggests that its added complexity is overfitting noise rather than capturing signal. Feature importance from the Random Forest confirms that only Income and Score had any predictive power, while categorical features contributed little—consistent with EDA, which showed weak relationships between the supplied features and the age target.

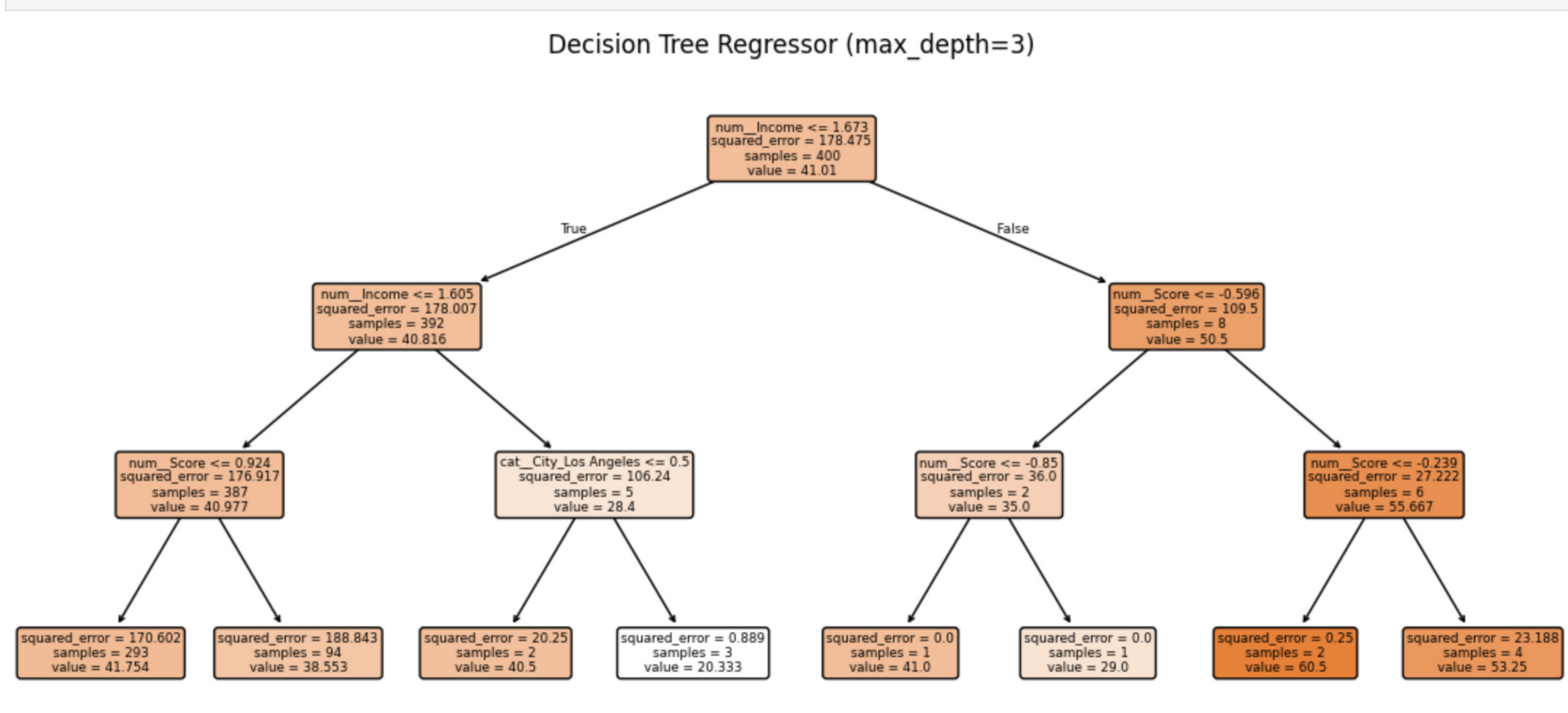
Part 4 – Interpretation + Visualization (20 pts)

- Highlight the **feature importances**
- Explain:
  - Which features matter most?
  - What do the splits in the tree tell you about how the model makes decisions?

Bonus: adjust tree depth and compare training vs. test scores (evidence of overfitting?)

In [ ]:

```
# Shallow tree (readable)
shallow_tree = Pipeline([
    ('prep', preprocessor),
    ('reg', DecisionTreeRegressor(max_depth=3, random_state=42))
]).fit(X_train, y_train)

plt.figure(figsize=(14,6))
plot_tree(
    shallow_tree['reg'],
    filled=True,
    feature_names=shallow_tree['prep'].get_feature_names_out(),
    rounded=True
)
plt.title("Decision Tree Regressor (max_depth=3)")
plt.show()

# Compare depths to illustrate overfitting
depths = [2, 3, None]  # None = full depth
results = []
for d in depths:
    model = Pipeline([
        ('prep', preprocessor),
        ('reg', DecisionTreeRegressor(max_depth=d, random_state=42))
    ]).fit(X_train, y_train)
    y_tr = model.predict(X_train)
    y_te = model.predict(X_test)
    results.append({
        "max_depth": d if d is not None else "None",
        "train_MSE": mean_squared_error(y_train, y_tr),
        "test_MSE": mean_squared_error(y_test, y_te),
        "train_R2": r2_score(y_train, y_tr),
        "test_R2": r2_score(y_test, y_te)
    })
```



Part 5 – Self-Reflection + AI Disclosure (20 pts)

Answer in Markdown:

1. What was the most difficult part of the modeling process?
2. What's one thing you learned about model comparison this week?
3. If you used ChatGPT or other AI tools:
   - What did you ask?
   - How did you validate the answer?
   - What did you modify or question?

In [ ]:

🔴 Classification Task

🔄 Workflow (For Each Task)

Part 1 – EDA + Task Definition (10 pts)

- Load the dataset
- Do `.info()`, `.describe()`, `.value_counts()` as appropriate
- Write:
  - What's your target variable?
  - What features might help with prediction?

In [ ]:

```
file_path = '/content/drive/MyDrive/info_4604/datasets/sports_betting_predictive_analysis.csv'

df2 = pd.read_csv(file_path)

print(df2.head())
print(df2.info())
print(df2.describe())
print(df2.value_counts())
```

```
  Match_ID        Date       Sport             Home_Team  \
0   M00001  2024-01-24  Basketball  Gonzalezmouth Tigers   
1   M00002  2025-05-03  Basketball       Rothstad Wolves   
2   M00003  2025-04-10    Baseball      Aliciaport Lions   
3   M00004  2024-08-02      Tennis  Lake Samantha Eagles   
4   M00005  2024-10-05      Tennis    Brendanport Eagles   

               Away_Team  Home_Team_Odds  Away_Team_Odds  Draw_Odds  \
0    Hernandezfurt Lions            3.62            3.99        NaN   
1     North Manuel Eagles            1.84             NaN        NaN   
2  West Gabrielton Lions            3.93            3.29        NaN   
3       Simonshire Lions            3.70            3.61        NaN   
4      Williamsfurt Bears            2.26            2.52        NaN   

       Predicted_Winner         Actual_Winner  
0  Hernandezfurt Lions   Hernandezfurt Lions  
1      Rothstad Wolves       Rothstad Wolves  
2     Aliciaport Lions      Aliciaport Lions  
3     Simonshire Lions  Lake Samantha Eagles  
4   Williamsfurt Bears    Brendanport Eagles  
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1369 entries, 0 to 1368
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Match_ID          1369 non-null   object 
 1   Date              1369 non-null   object 
 2   Sport             1369 non-null   object 
 3   Home_Team         1369 non-null   object 
 4   Away_Team         1369 non-null   object 
 5   Home_Team_Odds    1298 non-null   float64
 6   Away_Team_Odds    1282 non-null   float64
 7   Draw_Odds         498 non-null    float64
 8   Predicted_Winner  1369 non-null   object 
 9   Actual_Winner     1369 non-null   object 
dtypes: float64(3), object(7)
memory usage: 107.1+ KB
None
       Home_Team_Odds  Away_Team_Odds   Draw_Odds
count     1298.000000     1282.000000  498.000000
mean         3.069823        3.060031    3.103012
std          1.096600        1.087785    1.098248
min          1.200000        1.210000    1.200000
25%          2.110000        2.130000    2.142500
50%          3.080000        2.995000    3.160000
75%          3.980000        4.017500    4.062500
max          4.980000        5.000000    4.990000
Match_ID  Date        Sport     Home_Team             Away_Team                Home_Team_Odds  Away_Team_Odds  Draw_Odds  Predicted_Winner         Actual_Wi
nner
M01363    2025-04-17  Football  North Anita Lions     Christinabury Wolves      4.48            4.95            3.34       North Anita Lions        Draw
1
M00009    2024-06-26  Hockey    Lake Karen Bears      Lake Jennifer Lions       1.35            1.89            4.88       Lake Jennifer Lions      Lake Kare
n Bears       1
M00012    2024-11-03  Hockey    Brownmouth Tigers     Smithmouth Wolves         4.79            2.43            4.12       Draw                     Brownmout
h Tigers      1
M00014    2025-02-01  Hockey    Gonzalezview Wolves   Lake Bethanyfort Wolves   3.81            2.04            2.33       Lake Bethanyfort Wolves  Gonzalezv
iew Wolves    1
M00017    2023-09-18  Hockey    Curtisburgh Eagles    Michaelburgh Bears        4.61            1.53            1.21       Michaelburgh Bears       Michaelbu
rgh Bears     1

..
M00035    2024-09-14  Football  Jenniferton Lions     Martinborough Bears       2.99            2.55            3.08       Draw                     Martinbor
ough Bears    1
M00032    2024-03-05  Football  Robertshire Lions     New Elizabethside Wolves  2.99            3.34            3.53       Robertshire Lions        Draw
1
M00030    2023-12-26  Hockey    Heidimouth Tigers     Jennyview Wolves          3.34            3.26            1.33       Jennyview Wolves         Heidimout
h Tigers      1
M00029    2024-02-25  Football  West James Lions      East Erin Bears           4.91            1.40            4.95       West James Lions         Draw
1
M00028    2024-04-29  Football  Lake Virginia Tigers  Justinchester Tigers      4.32            4.84            4.31       Draw                     Draw
1
Name: count, Length: 443, dtype: int64
```

Part 2 – Preprocessing + Pipeline (20 pts)

- Identify:
  - Categorical vs numeric features
  - Features with missing values
- Build a full preprocessing pipeline:
  - `SimpleImputer`, `OneHotEncoder`, `StandardScaler` as needed
- Wrap in a full `Pipeline`

In Markdown, explain:

- Why you chose each preprocessing step
- Any assumptions you made about the data

In [ ]:

Part 3 – Model Training + Evaluation (30 pts)

- Train at least **two models** (Logistic, or SVM vs Random Forest)
- Evaluate:
  - For classification: `accuracy`, `precision`, `recall`, `f1_score`
- Show confusion matrix
- Use `train_test_split` and/or `cross_val_score` to assess generalization

In Markdown:

- Compare model performance
- Which model is better and why?
- Do your results make sense based on EDA?

In [ ]:

Part 4 – Interpretation + Visualization (20 pts)

- Highlight **feature importances**
- Explain:
  - Which features matter most?
  - What do the splits in the tree tell you about how the model makes decisions?

In [ ]:

Part 5 – Self-Reflection + AI Disclosure (20 pts)

Answer in Markdown:

1. What was the most difficult part of the modeling process?
2. What's one thing you learned about model comparison this week?
3. If you used ChatGPT or other AI tools:
   - What did you ask?
   - How did you validate the answer?
   - What did you modify or question?

In [ ]:

Regression Section	Points
Part 1: EDA	5
Part 2: Preprocessing	10
Part 3: Model Training + Eval	15
Part 4: Interpretation + Viz	10
Part 5: Reflection + AI Use	10

Classification Section	Points
Part 1: EDA	5
Part 2: Preprocessing	10
Part 3: Model Training + Eval	15
Part 4: Interpretation + Viz	10
Part 5: Reflection + AI Use	10
----------------------------	------
Total	**100**

In [ ]: