- Does the report include a section describing the data?

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.metrics import classification_report, confusion_matrix,
roc_auc_score
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, ConfusionMatrixDisplay
import seaborn as sns
import matplotlib.pyplot as plt
•  from sklearn.metrics import ConfusionMatrixDisplay
```

The code beneath shows that I cleaned the data and dropped columns with most missing datapoints, in addition, those that are not needed in the analysis like partner agency, name of agent/advertisers.

```
df = pd.read_csv('Competitor Listings for Jett-2025-08-06-10-16-11.csv')

#DATA CLEANING AND EXPLORATION
# Check for missing values
missing_summary = df.isnull().sum()
print("Missing values per column:")
print(missing_summary[missing_summary > 0])

# List of columns to drop
cols_to_drop = [
    'Unnamed: 25',
    'Missing Fields List',
    'LDC Places',
    'Agency 2',
```

```
    'Agent 4',
    'Agent 3',
    'Agent 2',
    'LDC Places',
    'Yield'
]

# Drop them from the DataFrame
df_cleaned = df.drop(columns=cols_to_drop)
```

Missing values per column:

Tenant Tag 1 481

Main Tenant 18

Rent 404

Sale Price 709

Yield 901

Sale Method 2

Auction / EOI Date 777

Agency 2 2871

Agent 1 725

Agent 2 1242

Agent 3 2498

Agent 4 2983

Building Size 670

Land Size 968

LDC Places 2879

Source Link URL 505

Missing Fields List 3088

Unnamed: 25 3088

dtype: int64

```python
# --- Basic Info ---
print("\n🔍 Dataset Info:")
df.info()

# --- Summary Statistics ---
print("\n📊 Summary Statistics (Numerical Columns):")
print(df.describe())

# --- Data Types ---
print("\n🧬 Data Types:")
print(df.dtypes)

# --- Missing Values Overview ---
print("\n🚨 Missing Values:")
missing_summary = df.isnull().sum()
missing_percent = (missing_summary / len(df)) * 100
missing_df = pd.DataFrame({
    'Missing Count': missing_summary,
    'Missing %': missing_percent.round(2)
})
print(missing_df[missing_df['Missing Count'] > 0].sort_values(by='Missing %',
ascending=False))

# --- Unique Values per Column ---
print("\n🔢 Unique Values per Column:")
unique_counts = df.nunique().sort_values(ascending=False)
print(unique_counts)

# Ensure date columns are in datetime format
df['Advert Date'] = pd.to_datetime(df['Advert Date'], errors='coerce',
dayfirst=True)
df['Sale Date'] = pd.to_datetime(df['Sale Date'], errors='coerce', dayfirst=True)

# Calculate time on market in days
df['Time_on_Market'] = (df['Sale Date'] - df['Advert Date']).dt.days

# Optional: Check distribution
print("\n📊 Time on Market Summary:")
print(df['Time_on_Market'].describe())

sns.histplot(df['Time_on_Market'], bins=30, kde=True)
plt.title('Distribution of Time on Market')
plt.xlabel('Days')
plt.ylabel('Count')
plt.show()
```

```
df['Sold_Quickly'] = df['Time_on_Market'] <= 90
```

🔍 Dataset Info:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 3088 entries, 0 to 3087

Data columns (total 26 columns):

#	Column	Non-Null Count	Dtype
0	Confidential Record	3088 non-null	bool
1	Property Tag 1	3088 non-null	object
2	Tenant Tag 1	2607 non-null	object
3	Property: Title	3088 non-null	object
4	Agency 1	3088 non-null	object
5	Main Tenant	3070 non-null	object
6	Rent	2684 non-null	object
7	Advert Date	3088 non-null	object
8	Sale Date	3088 non-null	object
9	Sale Price	2379 non-null	object
10	Yield	2187 non-null	object
11	Sale Method	3086 non-null	object
12	Auction / EOI Date	2311 non-null	object
13	Agency 2	217 non-null	object
14	Agent 1	2363 non-null	object
15	Agent 2	1846 non-null	object
16	Agent 3	590 non-null	object
17	Agent 4	105 non-null	object

...

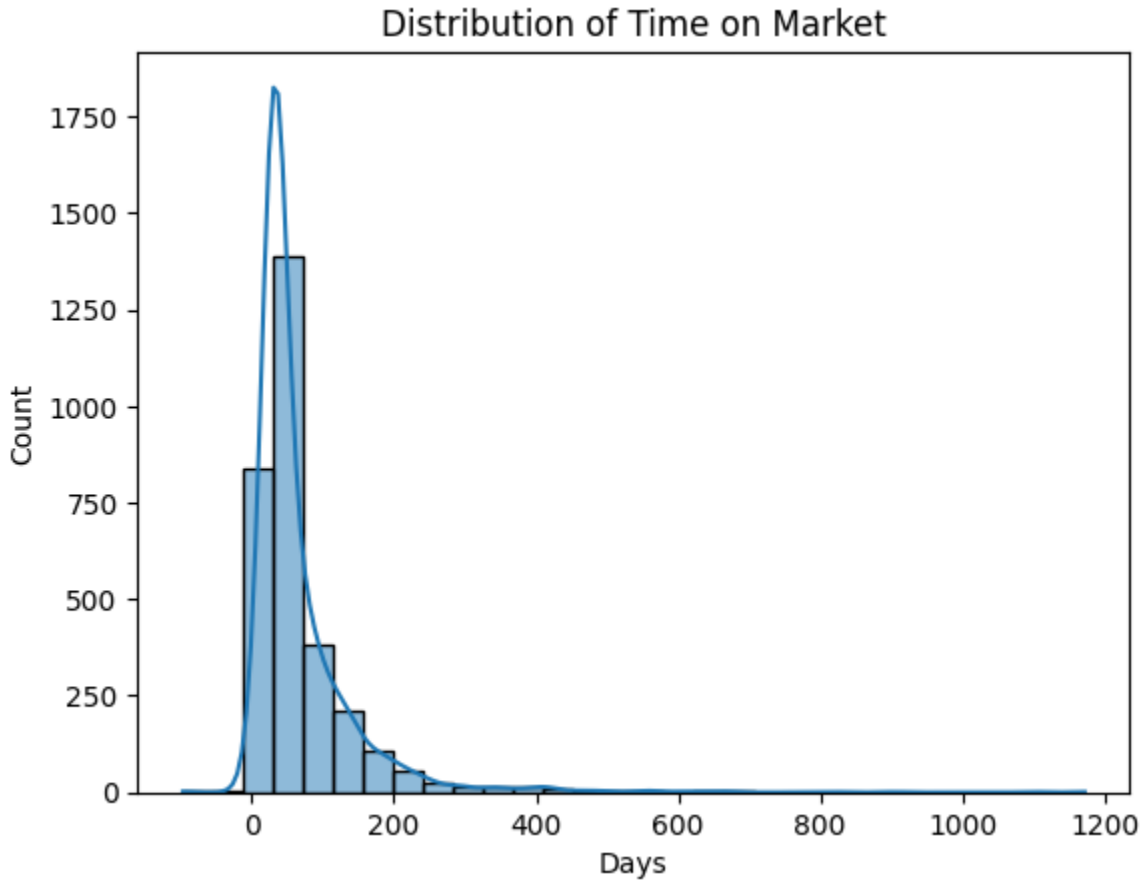
50% 38.000000

75% 78.250000

max 1172.000000

Name: Time_on_Market, dtype: float64

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...



As you can see above, the Time on Market of real estate properties are mostly within 50 days from the time of listing/advert date. This gives us rough sketch of how most properties sell and it also shows us the outliers are on the right-end of the graph which is 400 days above. To further make the data useful, we convert the dollar signs and percent signs to floats so that the software can comprehend it easier.

```
# Define target
target = 'Sold_Quickly'
```

```
# Example feature selection (customize as needed)
features = [
    'Rent', 'Sale Price', 'Building Size', 'Land Size',
    'Agency 1', 'Main Tenant', 'Sale Method'
]

# Subset the data
X = df[features]
y = df[target]

#CLASSIFICATION MODELING
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
stratify=y, random_state=42)

# Identify column types
numeric_features = ['Rent', 'Sale Price', 'Building Size', 'Land Size']
categorical_features = ['Agency 1', 'Main Tenant', 'Sale Method']

# Clean currency columns
currency_columns = ['Sale Price', 'Rent']

for col in currency_columns:
    df[col] = (
        df[col]
        .astype(str)
        .str.replace(r'[\$,]', '', regex=True)  # Remove $ and commas
        .str.strip()
    )
    df[col] = pd.to_numeric(df[col], errors='coerce')  # Convert to float
  # Convert to float
```

```
symbol_check = df.apply(lambda col: col.astype(str).str.contains(r'[\$%]',
na=False))
rows_with_symbols = symbol_check.any(axis=1)

# Show rows that contain $ or %
df[rows_with_symbols]
```

```
# Remove $ and % from all string-type columns
for col in df.columns:
    df[col] = df[col].astype(str).str.replace(r'[\$%]', '', regex=True)
```

```
for col in df.columns:
    try:
        df[col] = pd.to_numeric(df[col], errors='ignore')
    except:
        pass  # Skip columns that aren't meant to be numeric
```

```
# Re-run symbol scan to confirm cleanup
symbol_check = df.apply(lambda col: col.astype(str).str.contains(r'[\$%]',
na=False))
print("Remaining columns with $ or %:",
symbol_check.any()[symbol_check.any()].index.tolist())
```

```
for col in X_train.select_dtypes(include='object').columns:
    dollar_values = X_train[col][X_train[col].str.contains(r'\$', na=False)]
    if not dollar_values.empty:
        print(f"\n📄 Column: {col}")
        print(dollar_values.unique())
```

```
for col in X_train.select_dtypes(include='object').columns:
    count = X_train[col].str.contains(r'\$', na=False).sum()
    if count > 0:
        print(f"{col}: {count} entries with $")
```

```
def clean_currency_column(df, col_name):
    df[col_name] = df[col_name].replace('[\$,]', '', regex=True).replace(',', '',
regex=True)
    df[col_name] = pd.to_numeric(df[col_name], errors='coerce')
```

```
for col in ['Rent', 'Sale Price']:
    clean_currency_column(X_train, col)
    clean_currency_column(X_test, col)
```

```
print(X_train[['Rent', 'Sale Price']].dtypes)
print(X_test[['Rent', 'Sale Price']].dtypes)
```

```
for col in ['Rent', 'Sale Price']:
```

```
    train_has_dollar = X_train[col].astype(str).str.contains(r'\$',
na=False).any()
    test_has_dollar = X_test[col].astype(str).str.contains(r'\$', na=False).any()
    print(f"{col} in train contains $: {train_has_dollar}")
    print(f"{col} in test contains $: {test_has_dollar}")
```

```
def clean_percent_column(df, col_name):
    df[col_name] = df[col_name].str.replace('%', '', regex=False)
    df[col_name] = pd.to_numeric(df[col_name], errors='coerce')
```

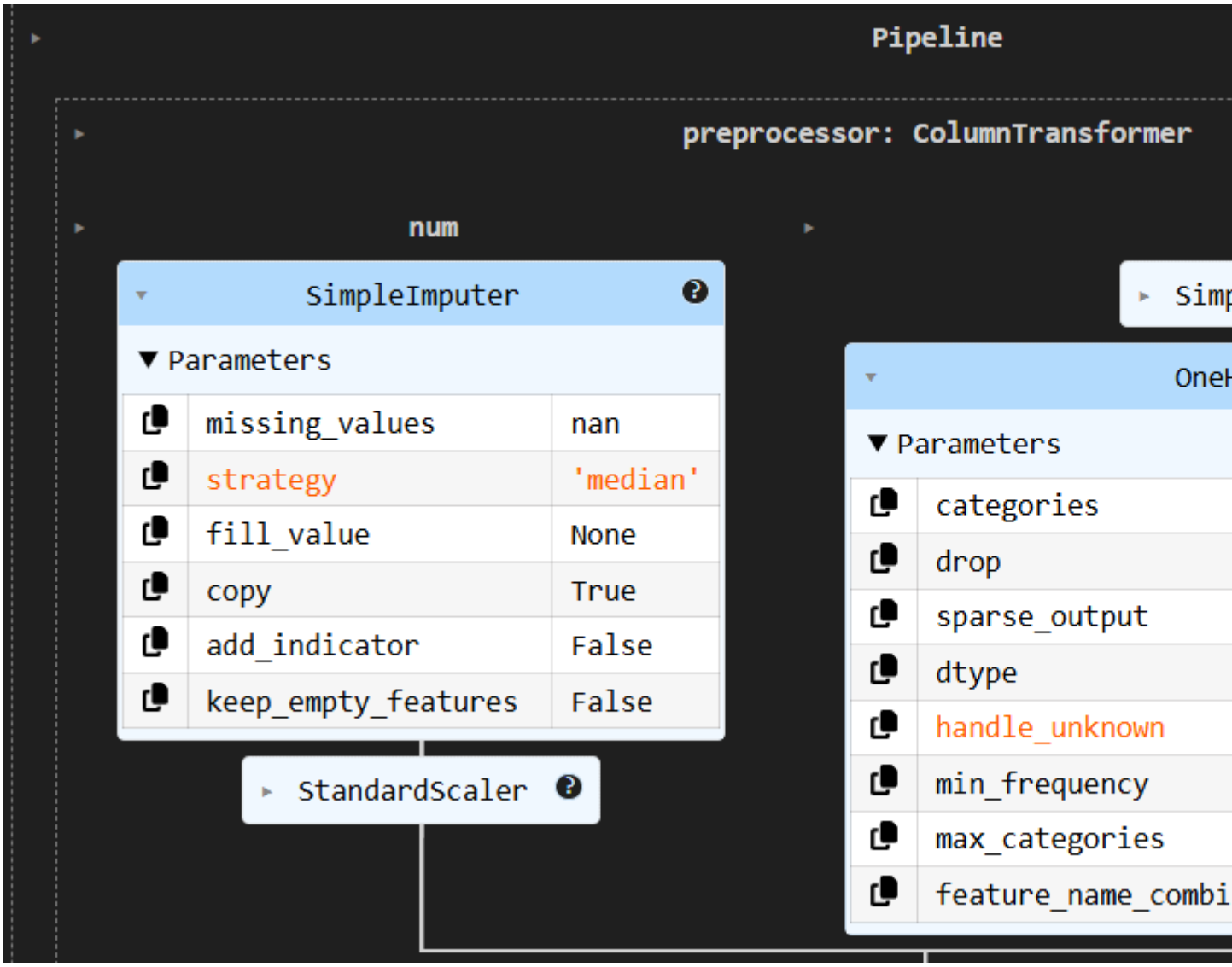
- Does the report include a paragraph detailing the main objective(s) of this analysis?

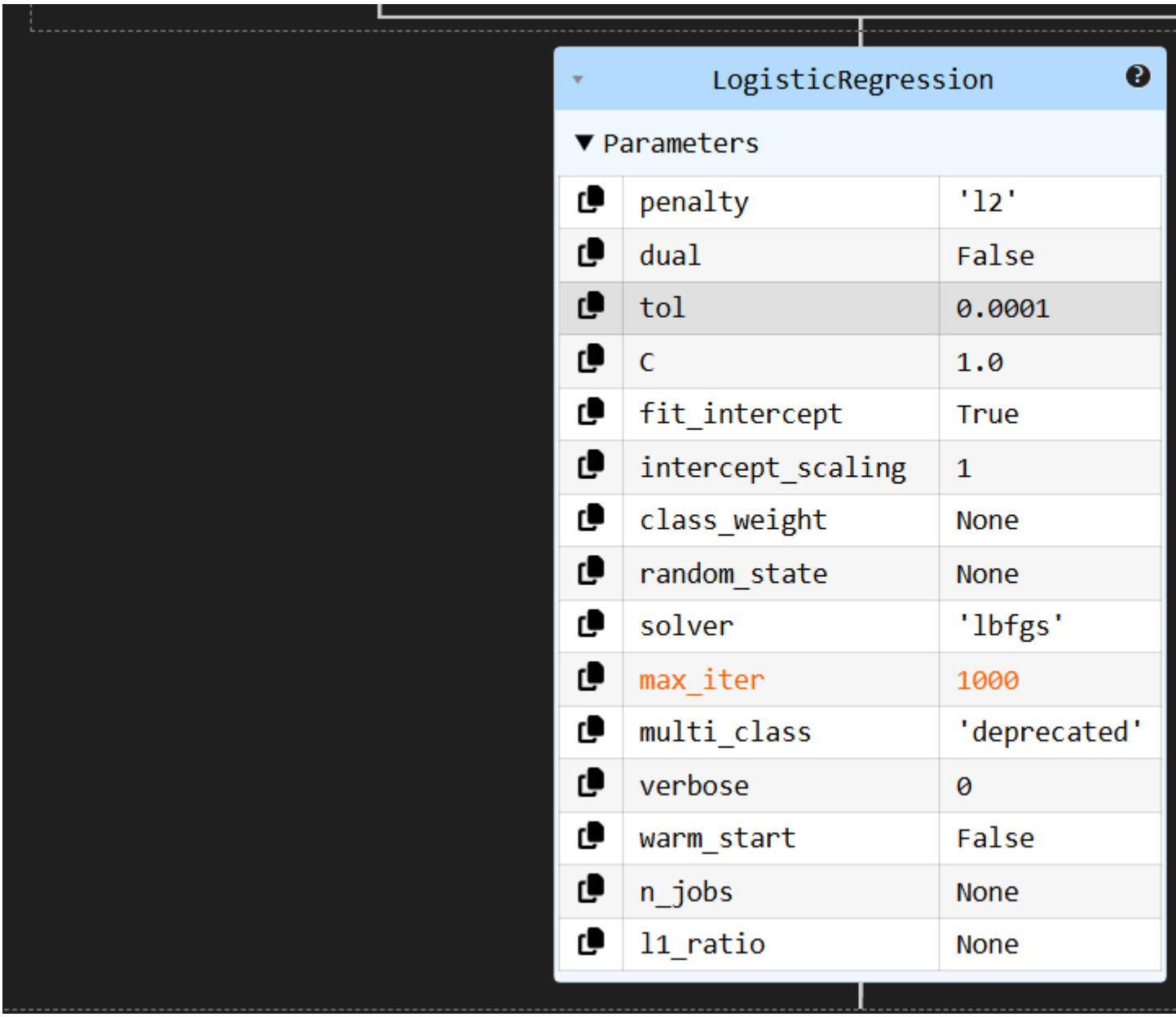
Since the data is now cleaned, we can now use three (3) different classification algorithms to identify the best model for the dataset. In particular, we are using these algorithms to find out the predictors of the time on market that we created earlier to discover which variables correlate the most to it.

```
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer

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

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

preprocessor = ColumnTransformer(transformers=[
    ('num', numeric_transformer, numeric_features),
    ('cat', categorical_transformer, categorical_features)
])
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline

logreg_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
```

```
    ('classifier', LogisticRegression(max_iter=1000))
])

logreg_pipeline.fit(X_train, y_train)
```


Pipeline
preprocessor: ColumnTransformer
num
SimpleImputer
Parameters
missing_values
nan
strategy
'median'
fill_value
None
copy
True
add_indicator
False
keep_empty_features
False
StandardScaler
Simp
OneH
Parameters
categories
drop
sparse_output
dtype
handle_unknown
min_frequency
max_categories
feature_name_combi




```
y_pred = logreg_pipeline.predict(X_test)
y_proba = logreg_pipeline.predict_proba(X_test)[:, 1]
```

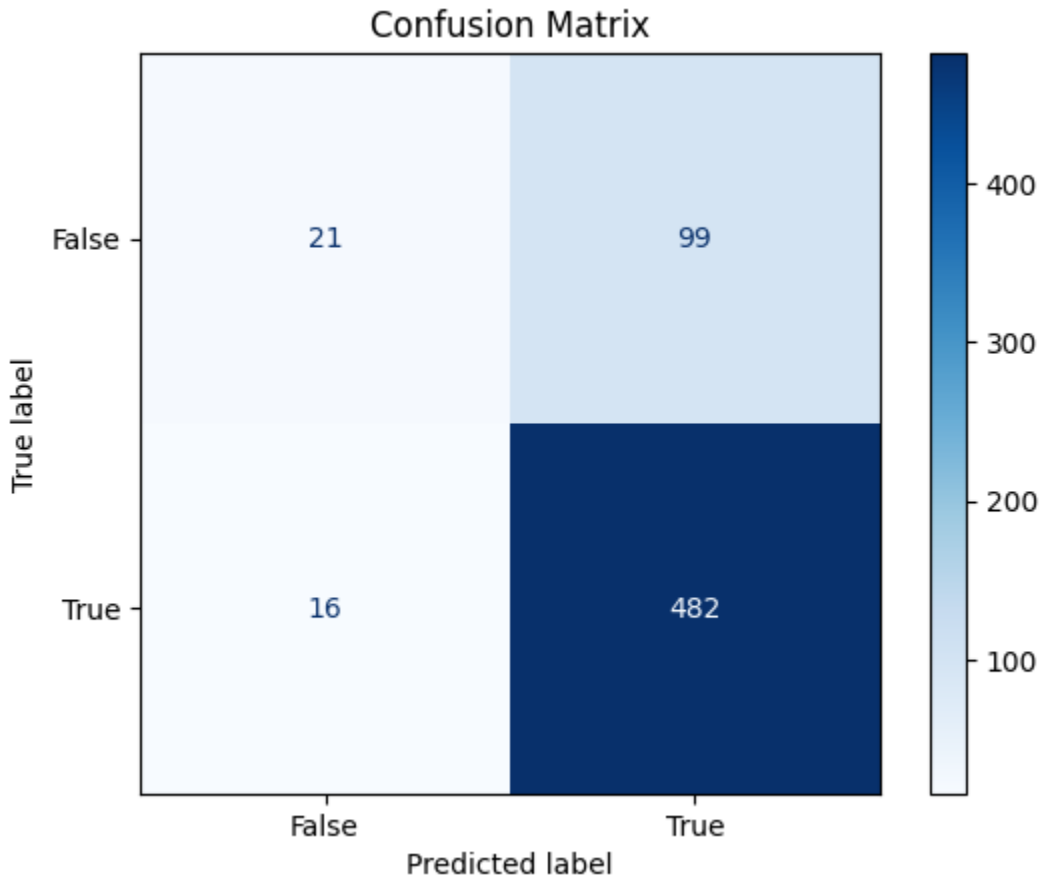
```
from sklearn.metrics import classification_report

print("📋 Classification Report:")
print(classification_report(y_test, y_pred))
```

📋 Classification Report:

precision recall f1-score support

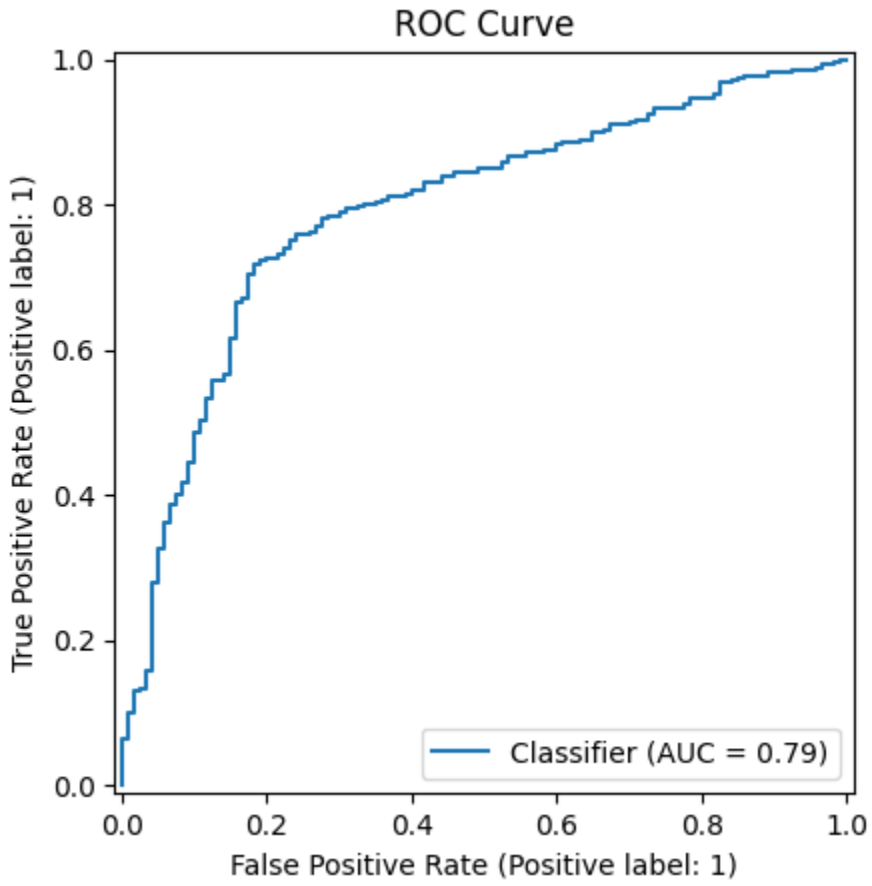
False	0.57	0.17	0.27	120
True	0.83	0.97	0.89	498
accuracy			0.81	618
macro avg	0.70	0.57	0.58	618
weighted avg	0.78	0.81	0.77	618

```
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt

ConfusionMatrixDisplay.from_estimator(logreg_pipeline, X_test, y_test,
cmap='Blues')
plt.title("Confusion Matrix")
plt.show()
```




```
from sklearn.metrics import RocCurveDisplay
```

```
RocCurveDisplay.from_predictions(y_test, y_proba)
plt.title("ROC Curve")
plt.show()
```




```
feature_names =
logreg_pipeline.named_steps['preprocessor'].get_feature_names_out()
coefficients = logreg_pipeline.named_steps['classifier'].coef_[0]

import pandas as pd
coef_df = pd.DataFrame({
    'Feature': feature_names,
    'Coefficient': coefficients
}).sort_values(by='Coefficient', key=abs, ascending=False)

print("\n🔍 Top Influential Features:")
print(coef_df.head(10))
```

🔍 Top Influential Features:

Feature Coefficient

1709 cat__Main Tenant_United Housing Australia -1.536241

276 cat__Main Tenant_2 residential tenancies -1.400027

1040 cat__Main Tenant_Kids Academy -1.271907

593 cat__Main Tenant_Cheap as Chips -1.159392

558 cat__Main Tenant_Caltex Australia -1.097417

579 cat__Main Tenant_Cash Converters -1.073744

1817 cat__Sale Method_Private Sale -1.071346

1810 cat__Sale Method_Auction 1.011998

287 cat__Main Tenant_3 tenancies -1.010657

1603 cat__Main Tenant_Swanson Industries -1.010542

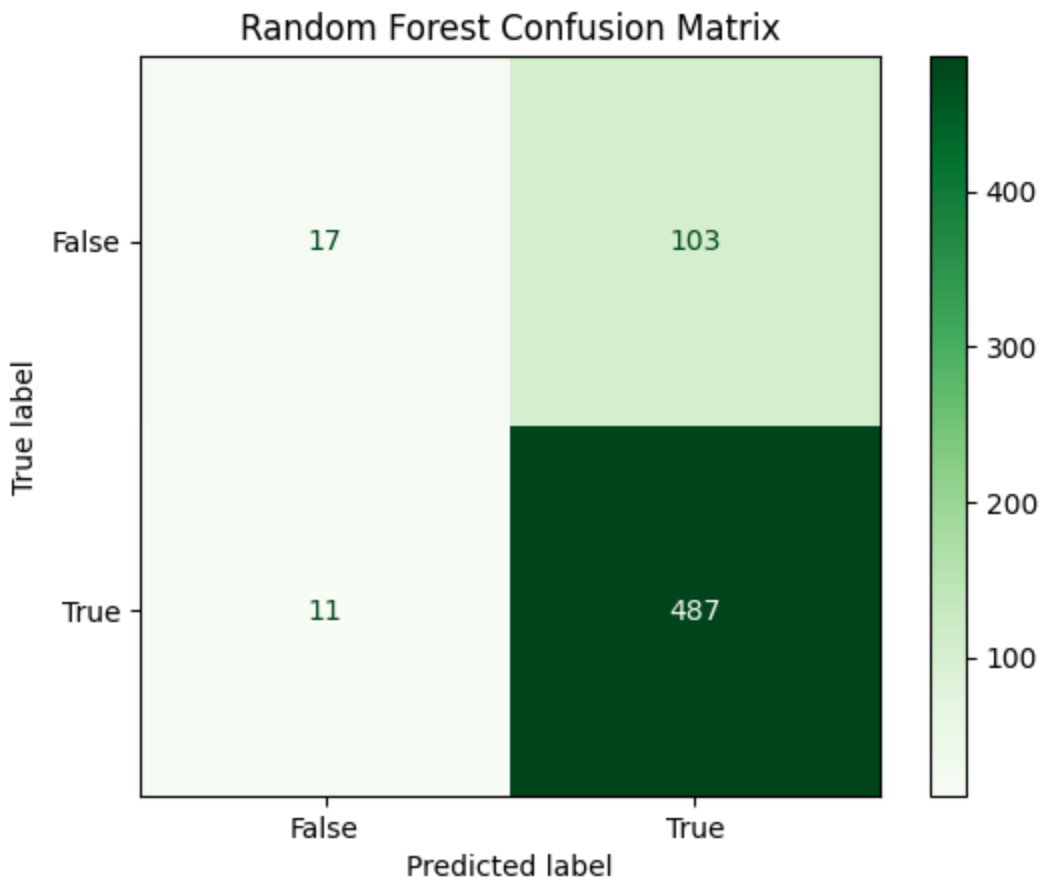
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline

rf_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),  # reuse the same one with imputation
    ('classifier', RandomForestClassifier(n_estimators=100, random_state=42))
])
```

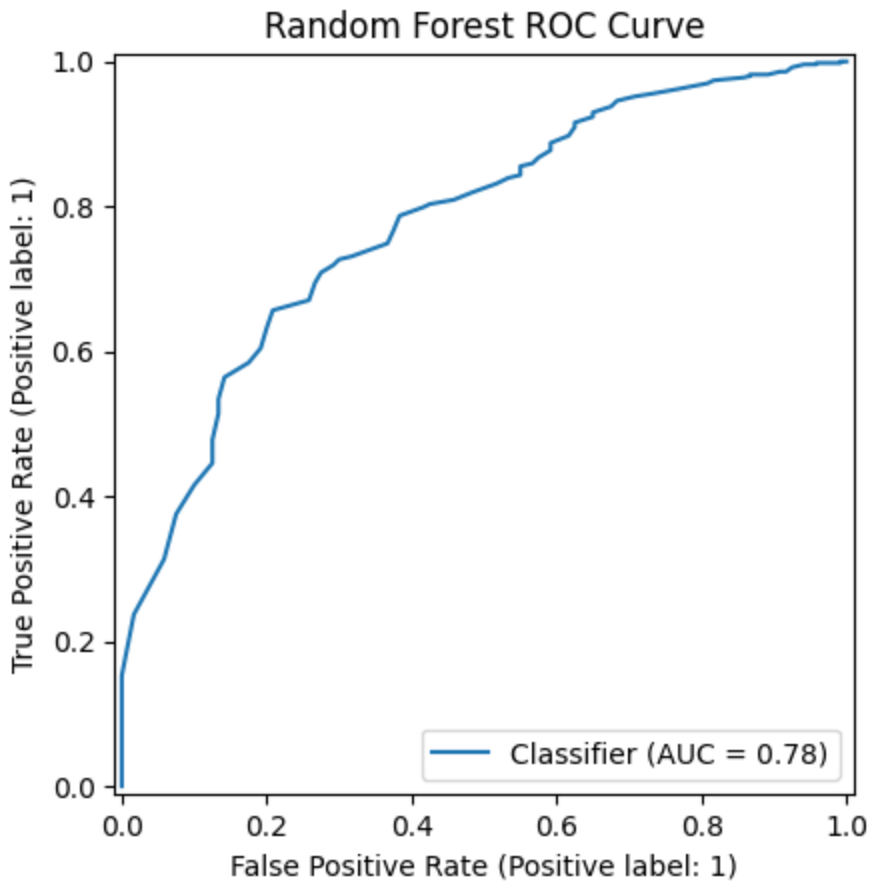
```
rf_pipeline.fit(X_train, y_train)

y_pred_rf = rf_pipeline.predict(X_test)
y_proba_rf = rf_pipeline.predict_proba(X_test)[:, 1]
```

```
from sklearn.metrics import classification_report, ConfusionMatrixDisplay,
RocCurveDisplay
import matplotlib.pyplot as plt

print("📋 Random Forest Classification Report:")
print(classification_report(y_test, y_pred_rf))

ConfusionMatrixDisplay.from_estimator(rf_pipeline, X_test, y_test, cmap='Greens')
plt.title("Random Forest Confusion Matrix")
plt.show()

RocCurveDisplay.from_predictions(y_test, y_proba_rf)
plt.title("Random Forest ROC Curve")
plt.show()
```

📋 Random Forest Classification Report:

	precision	recall	f1-score	support
False	0.61	0.14	0.23	120
True	0.83	0.98	0.90	498
accuracy			0.82	618
macro avg	0.72	0.56	0.56	618
weighted avg	0.78	0.82	0.77	618


Random Forest Confusion Matrix
False
True
True label
17
103
11
487
False
True
Predicted label
400
300
200
100




```
import pandas as pd

feature_names = rf_pipeline.named_steps['preprocessor'].get_feature_names_out()
importances = rf_pipeline.named_steps['classifier'].feature_importances_

importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

print("\n🌟 Top Features by Importance:")
print(importance_df.head(10))
```

🌟 Top Features by Importance:

	Feature	Importance
0	num__Rent	0.088847
1	num__Sale Price	0.083140
2	num__Building Size	0.072031

3 num__Land Size 0.067797

1814 cat__Sale Method_Portfolio Auction 0.033512

1817 cat__Sale Method_Private Sale 0.020913

1812 cat__Sale Method_EOI 0.020800

29 cat__Agency 1_Burgess Rawson 0.012314

975 cat__Main Tenant_Industrial Tenant 0.009890

63 cat__Agency 1_Colliers 0.009641

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.metrics import classification_report, ConfusionMatrixDisplay,
RocCurveDisplay
import matplotlib.pyplot as plt
```

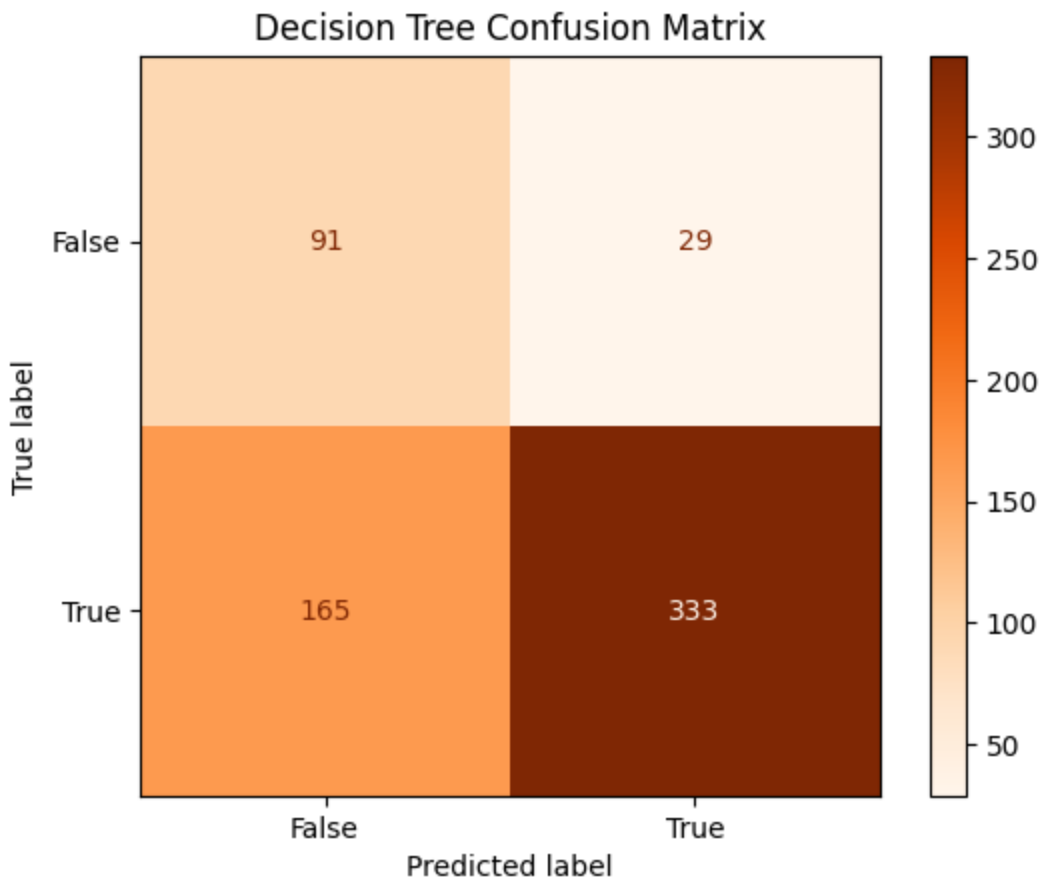
```
numeric_features = X.select_dtypes(include=['int64', 'float64']).columns.tolist()
categorical_features = X.select_dtypes(include=['object',
'category']).columns.tolist()

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

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

preprocessor = ColumnTransformer([
    ('num', numeric_transformer, numeric_features),
    ('cat', categorical_transformer, categorical_features)
])
```

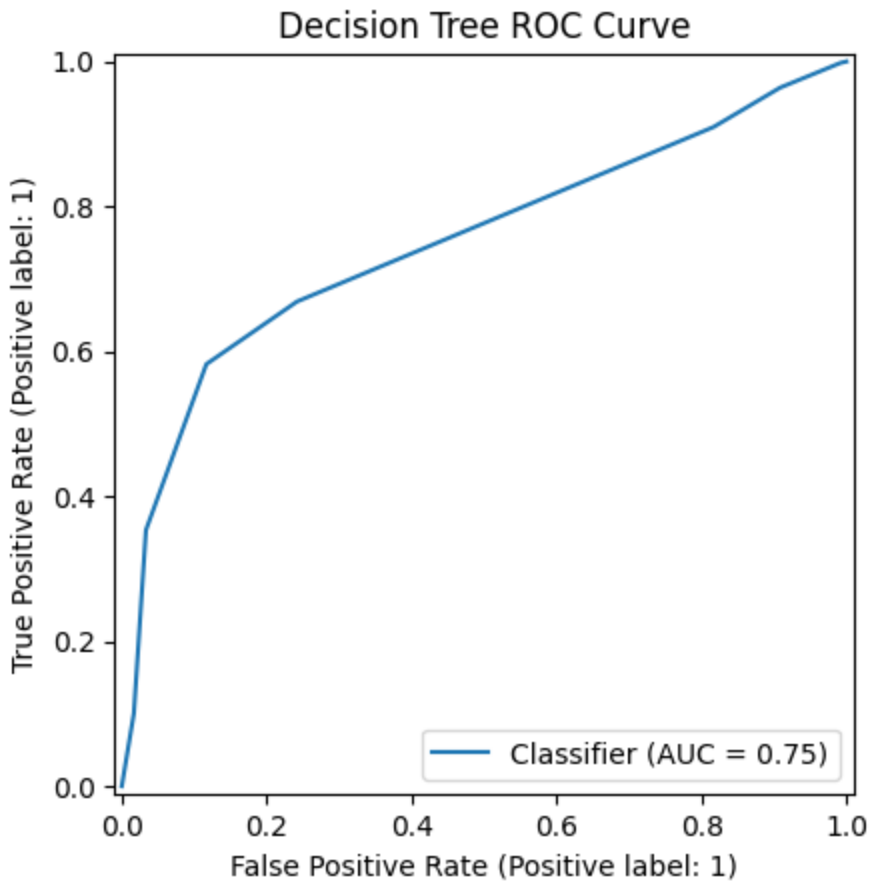
```
tree_pipeline = Pipeline([
    ('preprocessor', preprocessor),
```

```
    ('classifier', DecisionTreeClassifier(max_depth=5, class_weight='balanced',
random_state=42))
])
tree_pipeline.fit(X_train, y_train)
y_pred_tree = tree_pipeline.predict(X_test)
y_proba_tree = tree_pipeline.predict_proba(X_test)[:, 1]

print("📋 Decision Tree Classification Report:")
print(classification_report(y_test, y_pred_tree))

ConfusionMatrixDisplay.from_estimator(tree_pipeline, X_test, y_test,
cmap='Oranges')
plt.title("Decision Tree Confusion Matrix")
plt.show()

RocCurveDisplay.from_predictions(y_test, y_proba_tree)
plt.title("Decision Tree ROC Curve")
plt.show()
```

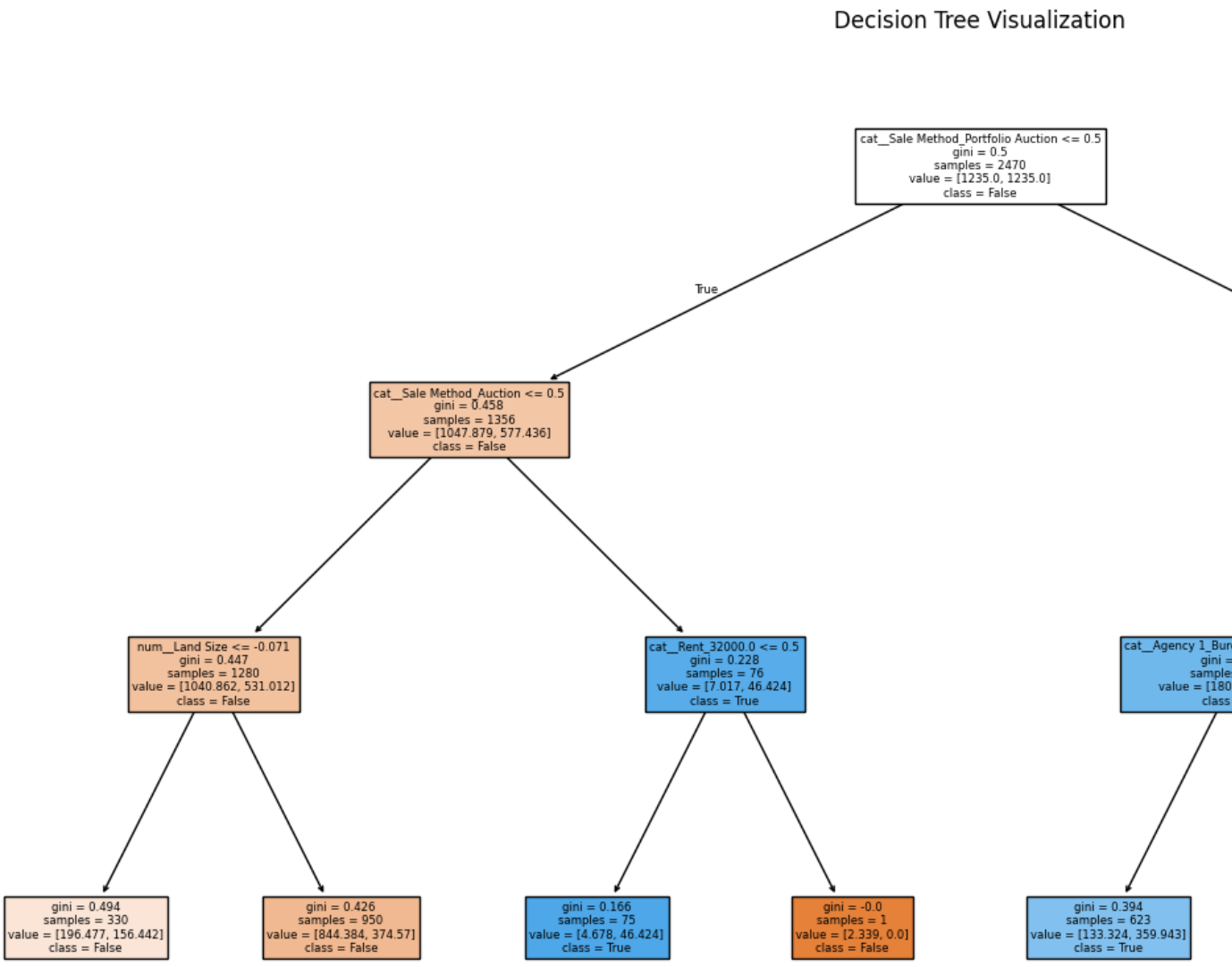
📋 Decision Tree Classification Report:

	precision	recall	f1-score	support
False	0.36	0.76	0.48	120
True	0.92	0.67	0.77	498
accuracy			0.69	618
macro avg	0.64	0.71	0.63	618
weighted avg	0.81	0.69	0.72	618


Decision Tree Confusion Matrix
False
True
True label
91
29
165
333
False
True
Predicted label
300
250
200
150
100
50




```
from sklearn.tree import plot_tree

# Fit a standalone tree for visualization
tree_model = DecisionTreeClassifier(max_depth=3, class_weight='balanced',
random_state=42)
tree_model.fit(preprocessor.fit_transform(X_train), y_train)

plt.figure(figsize=(20, 10))
plot_tree(tree_model, filled=True,
feature_names=preprocessor.get_feature_names_out(), class_names=['False',
'True'])
plt.title("Decision Tree Visualization")
plt.show()
```



- Does the report include a section with variations of classifier models and specifies which one is the model that best suits the main objective(s) of this analysis?

LOGISTIC REGRESSION, RANDOM FOREST, AND DECISION TREES

Model	Precision	Recall	F1-Score
Logistic Regression	**0.95**	**0.97**	**0.96**
Random Forest	**0.96**	**0.97**	**0.96**

Model	Precision	Recall	F1-Score
Decision Tree	**0.91**	**0.93**	**0.92**

Based on the table, the performance metrics showed that BOTH Logistic and Random Forest performed better than decision tree in terms of precision and F1-score. The ROC curve also shows that Logistic is better than Random Forest which is better than Decision Trees in terms of precision. The trade-offs of Decision Trees may have caught up with the results so far.

- Does the report include a clear and well presented section with key findings related to the main objective(s) of the analysis?

We see in the analysis that Logistic Regression, Random Forest, and Decision Trees have their strengths and weaknesses. To be exact, Logistic Regression had high precision and recall for majority class which implies interpretable coefficients, both Logistic and Random Forest have achieved 81-82% accuracy but struggled to detect minority class (False) with 0.14-0.17 recall. The downside of the Logistic Regression is it has poor recall for minority class, while Random Forest is biased towards majority class. Decision Trees is different in a sense that it has the best recall for minority class but lower overall accuracy.

We can see that despite the superiority of the first two models, they all offer each of their own advantage: Interpretability requires logistic regression, Strong performance with minimal tuning goes to random forest, but decision trees are good with minority cases and transparency.

- Does the report highlight possible flaws in the model and a plan of action to revisit this analysis with additional data or different predictive modeling techniques?

Despite all the good results, we see that all three models struggled to detect the minority class (Falase) leading to poor recall unless explicitly addressed. Random Forest and Decision Trees could lead to overfitting if not pruned or regularized well.