I. Business Understanding

Introduction:

As a leading data mining and statistical computation company, Chris Tech specializes in transforming raw data into valuable insights. Signature Realtors, recognizing our expertise, has entrusted us with a strategic assignment. Our collaborative venture aims to develop a sophisticated price estimation model for houses, leveraging data extracted from jiji.com. This report delineates the meticulous approach we have adopted in the initial phases of the CRISP-DM framework to ensure the success of the project.

Business Objectives:

Our first objective in this collaboration is to create a precise price estimation model for houses, a task that aligns seamlessly with our core competencies in data mining and statistical computation. Through in-depth discussions with Signature Realtors, we've established clear success criteria, emphasizing accurate price predictions, model interpretability, and seamless integration into their existing systems. Our commitment to understanding the business objectives sets the stage for a purposeful and effective data mining project.

The Situation:

We recognize the importance of a comprehensive understanding of the project's context. Assessing the situation involves identifying the required resources, project requirements, potential risks, and conducting a cost-benefit analysis. We've meticulously evaluated the availability of human resources with expertise in data scraping and model development. Simultaneously, we've assessed the technological requirements and time commitments necessary for the successful completion of the project. Our risk assessment has identified potential challenges such as legal issues related to data scraping, data quality concerns, and model interpretability challenges. This detailed situational analysis, including a cost-benefit evaluation, ensures that our collaboration with Signature Realtors is grounded in a robust foundation.

Data Mining Goals:

The technical objectives of our data mining process have been clearly defined. These include specifying data cleaning processes, feature engineering techniques, model selection criteria, and the metrics by which we'll evaluate the success of our models. Additionally, we've outlined the data requirements, explicitly specifying the types of features needed for our price estimation model—factors such as size, bedrooms, bathrooms, neighborhood characteristics, and country-specific attributes. Our meticulous approach to determining data mining goals ensures that we have a roadmap for technical success aligned with the overall business objectives.

Project Plan:

Selecting appropriate technologies and tools is paramount to the success of our project. We've chosen tools that facilitate efficient data scraping, robust data preprocessing, and streamlined model development. Our detailed project plan encompasses timelines and milestones for each phase of the project, providing clarity on tasks related to data scraping, preprocessing, model development, and evaluation. Additionally, we've established a communication plan to ensure seamless collaboration with Signature Realtors, fostering transparency and mutual understanding throughout the project lifecycle.

II. Data Understanding

The Data:

The first task involves acquiring the necessary data and loading it into our analysis tool. Given that our project involves scraping data from jiji.com, this task is crucial for kickstarting the data mining process. The data collected should encompass the features essential for our price estimation model, such as house size, bedrooms, bathrooms, neighborhood details, and country-specific attributes. This task sets the stage for subsequent analysis, ensuring that we have the raw materials needed for our modeling endeavors.

for more information checkout: 'DATA\data_description.txt'

and the scrapper notebook : "APP\scrapper.ipynb"

```
# import the dependencies
import numpy as np
import pandas as pd
from xgboost import XGBRegressor
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import FunctionTransformer, OneHotEncoder,
StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
from sklearn.linear_model import Lasso, Ridge, LinearRegression
from sklearn.model_selection import GridSearchCV, train_test_split,
ShuffleSplit, cross_val_score
from sklearn.metrics import r2_score, mean_squared_error,
mean_absolute_error, make_scorer
from sklearn.ensemble import RandomForestRegressor
from joblib import dump


# preferences

import warnings
warnings.filterwarnings('ignore')

pd.set_option('display.max_columns', None)

# load the data
data = pd.read_csv('DATA\\finalData.csv')
```

```
def clean_dataframe(df):
    # Drop rows with null values
    df_cleaned = df.dropna()

    # Drop duplicate rows
    df_cleaned = df_cleaned.drop_duplicates()

    # Reset the index after dropping rows
    df_cleaned = df_cleaned.reset_index(drop=True)

    return df_cleaned

data = clean_dataframe(data)

#chck the top ten raws
data.head(10)
```

```
             Cost
Title  \
0   KSh 7,500,000   3bdrm House in Tabasamu Annex, Kalimoni for Sale

1  KSh 16,500,000       4bdrm Maisonette in Milimani Estate for Sale

2   KSh 4,325,000  2bdrm Block of Flats in Estate, Old Junction f...

3   KSh 4,500,000              Studio Apartment in Kilimani for sale

4  KSh 35,000,000  Furnished 4bdrm Mansion in Palm Tree, Ukunda f...

5   KSh 8,800,000    2bdrm Apartment in Kcc Estate, Umoja I for sale

6   KSh 6,500,000                   3bdrm Bungalow in Rimpa for Sale

7  KSh 42,000,000  4bdrm Townhouse/Terrace in Loresho Ridge Estat...

8  KSh 15,000,000  4bdrm Maisonette in Transview, Mombasa Road fo...

9   KSh 7,300,000      1bdrm Apartment in Kileleshwa Estate for sale


                                         Description
Location  \
0  3 Bedroom Houses available for viewing. \nOthe...
Kiambu, Juja
1  I am selling a 4bedroom massionate sitted on a...           Kiambu,
Ruiru
2  Affordable Housing in Ruiru\nMost of the block...           Kiambu,
Ruiru
3  Property description\n* modern studio apartmen...       Nairobi,
Kilimani
4  Four bedroom mansion in diani! @\nlooking for ...           Kwale,
```

```
Ukunda
5  3bedroomed house for sale.* \nPlus a one room\...          Nairobi, 
Umoja
6  Newly built 3 bedroom bungalow in a gated comm...  Kajiado, Ongata 
Rongai
7  Property type: House \nOffer type: For sale \n...      Nairobi, 
Westlands
8  4bedroom town house for sale. With SQ With a k...   Nairobi, 
Mombasa Road
9  Purchase these very spacious and very luxuriou...     Nairobi, 
Kileleshwa

     Bedrooms    Bathrooms       Furnished    Space
0  3 bedrooms  2 bathrooms     Unfurnished   300sqm
1  4 bedrooms  3 bathrooms     Unfurnished   165sqm
2  2 bedrooms  2 bathrooms     Unfurnished    75sqm
3   1 bedroom   1 bathroom     Unfurnished    35sqm
4  4 bedrooms  5 bathrooms       Furnished  1000sqm
5  2 bedrooms  3 bathrooms     Unfurnished    10sqm
6  3 bedrooms  2 bathrooms  Semi-Furnished   200sqm
7  4 bedrooms  5 bathrooms     Unfurnished   245sqm
8  4 bedrooms  4 bathrooms     Unfurnished   450sqm
9   1 bedroom  2 bathrooms     Unfurnished    78sqm
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8838 entries, 0 to 8837
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   Cost         8838 non-null   object
 1   Title        8838 non-null   object
 2   Description  8838 non-null   object
 3   Location     8838 non-null   object
 4   Bedrooms     8838 non-null   object
 5   Bathrooms    8838 non-null   object
 6   Furnished    8838 non-null   object
 7   Space        8838 non-null   object
dtypes: object(8)
memory usage: 552.5+ KB
```

```
data.columns
```

```
Index(['Cost', 'Title', 'Description', 'Location', 'Bedrooms', 
'Bathrooms',
       'Furnished', 'Space'],
      dtype='object')
```

```
data.describe().T
```

```
            count unique                                              
top  \
Cost         8838    539                                     KSh 
7,500,000  
Title        8838   4964               2bdrm Apartment in Kilimani for 
sale  
Description  8838   7234  BUNGALOW HOUSE ON SALE ALONG THIKA RD RUIRU 
KI...   
Location     8838    214                                 Nairobi, 
Kilimani   
Bedrooms     8838     13                                       3 
bedrooms   
Bathrooms    8838     13                                      2 
bathrooms   
Furnished    8838      3                                      
Unfurnished   
Space        8838    621                                            
500sqm   

             freq  
Cost          249  
Title         100  
Description     6  
Location      766  
Bedrooms     3021  
Bathrooms    2283  
Furnished    7433  
Space         397  

data.shape

(8838, 8)
```

III. Data Preparation

Data:

The first task in the Data Preparation phase involves determining which data sets will be used for modeling and documenting the reasons for inclusion/exclusion. In the context of our project, this task is pivotal for selecting the relevant features and ensuring that the chosen data aligns with the objectives of creating a price estimation model. Decisions made during this task directly impact the effectiveness and accuracy of the subsequent modeling steps.

ETL Data_Pipeline

This ETL process encompasses a series of functions tailored for preprocessing real estate listing data. The 'split_location,' 'process_bedrooms,' 'process_bathrooms,' 'process_space,' 'remove_ksh,' 'process_title,' and 'process_description' functions each address specific columns, extracting, cleaning, and transforming information. The ColumnTransformer, known as ETL, applies these functions to relevant columns, producing a new DataFrame named preprocessed_data. The resulting dataset includes essential features such as 'County,' 'Neighborhood,' 'Bedroom,' 'Bathroom,' 'Size,' 'Type,' 'Price,' 'Quality,' and 'Furnished.' Columns not explicitly transformed are retained. This final preprocessed dataset provides a structured and cleaned foundation for subsequent analyses, particularly in the context of real estate pricing based on diverse features.

```
def split_location(df):
    df[['County', 'Neighborhood']] = df['Location'].str.split(', ',
n=1, expand=True)
    df['County'] = df['County'].str.strip()
    df['Neighborhood'] = df['Neighborhood'].str.strip()
    df = df.drop('Location', axis=1)
    return df

def process_bedrooms(df):
    df['Bedrooms'] = df['Bedrooms'].apply(lambda x: x.split()[0] if
isinstance(x, str) else x)
    return df[['Bedrooms']]

def process_bathrooms(df):
    df['Bathrooms'] = df['Bathrooms'].apply(lambda x: x.split()[0] if
isinstance(x, str) else x)
    return df[['Bathrooms']]

def process_space(df):
    df['Space'] = df['Space'].apply(lambda x: x.replace('sqm',
'').strip() if isinstance(x, str) else x)
    return df[['Space']]

def remove_ksh(df):
    df['Cost'] = df['Cost'].apply(lambda x: x.replace('KSh',
'').replace(',', '').strip() if isinstance(x, str) else x)
    return df[['Cost']]

def process_title(df):
    keywords = ['Apartment', 'Villa', 'Bungalow', 'Maisonette',
'House', 'Mansion', 'flat']
    df['Type'] = df['Title'].apply(lambda title: next((word for word
in keywords if word.lower() in title.lower()), np.nan))
    return df[['Type']]

def process_description(df):
    keywords = ['luxury', 'executive', 'magnificent',
                'modern', 'ensuite', 'opulent', 'newly',
                'Spacious', 'Residential', 'equipped',
                ]
    df['Quality'] = df['Description'].apply(lambda desc: 'good' if
```

```
any(keyword in desc.lower() for keyword in keywords) else 'moderate')
    return df[['Quality']]

ETL = ColumnTransformer(
    transformers=[
        ('county_and_Neighborhood',
FunctionTransformer(split_location, validate=False), ['Location']),
        ('process_bedrooms', FunctionTransformer(process_bedrooms,
validate=False), ['Bedrooms']),
        ('process_bathrooms', FunctionTransformer(process_bathrooms,
validate=False), ['Bathrooms']),
        ('process_space', FunctionTransformer(process_space,
validate=False), ['Space']),
        ('process_title', FunctionTransformer(process_title,
validate=False), ['Title']),
        ('remove_ksh', FunctionTransformer(remove_ksh,
validate=False), ['Cost']),
        ('process_description',
FunctionTransformer(process_description, validate=False),
['Description']),
    ],
    remainder='passthrough'  # Include other columns as-is
)

preprocessed_data = ETL.fit_transform(data)
preprocessed_data = pd.DataFrame(preprocessed_data)
preprocessed_data.columns = ['County','Neighborhood',  'Bedroom',
'Bathroom', 'Size', 'Type', 'Price', 'Quality', 'Furnished']
preprocessed_data.head(10)
```

	County	Neighborhood	Bedroom	Bathroom	Size	Type	Price \
0	Kiambu	Juja	3	2	300	House	7500000
1	Kiambu	Ruiru	4	3	165	Maisonette	16500000
2	Kiambu	Ruiru	2	2	75	flat	4325000
3	Nairobi	Kilimani	1	1	35	Apartment	4500000
4	Kwale	Ukunda	4	5	1000	Mansion	35000000
5	Nairobi	Umoja	2	3	10	Apartment	8800000
6	Kajiado	Ongata Rongai	3	2	200	Bungalow	6500000
7	Nairobi	Westlands	4	5	245	House	42000000
8	Nairobi	Mombasa Road	4	4	450	Maisonette	15000000
9	Nairobi	Kileleshwa	1	2	78	Apartment	7300000

```
   Quality        Furnished
0  moderate     Unfurnished
1  moderate     Unfurnished
2  moderate     Unfurnished
3      good     Unfurnished
4  moderate       Furnished
5  moderate     Unfurnished
6      good  Semi-Furnished
7  moderate     Unfurnished
8  moderate     Unfurnished
9  moderate     Unfurnished
```

Split The Data

```
X = preprocessed_data.drop('Price', axis=1)
y = preprocessed_data['Price']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

Feature Engineering

The transformations are organized into a ColumnTransformer named feature_engineering, which applies distinct processes to different subsets of features. The 'One_Hot_Encode' transformation utilizes one-hot encoding on categorical columns ('Neighborhood', 'County', 'Type', 'Quality', 'Furnished'), dropping the first category to avoid multicollinearity and ignoring unknown values. Additionally, it incorporates a frequency threshold of 20 to handle less frequent categories. The 'Scaller' transformation standardizes numerical features ('Size', 'Bathroom', 'Bedroom') using StandardScaler. This ensures that these features are on a consistent scale, preventing one from dominating the others during model training. The 'passthrough' remainder parameter retains any columns not explicitly transformed. Overall, this feature engineering setup prepares the input data for machine learning by encoding categorical variables and standardizing numerical features, optimizing them for subsequent model training and evaluation.

```
# Specify transformations

feature_transformations = [
                   ('One_Hot_Encode',
OneHotEncoder(sparse_output=False,

handle_unknown='ignore',
                                                    drop='first',
                                                    min_frequency=20
                                                    ),
                    ['Neighborhood','County', 'Type', 'Quality',
'Furnished']),
```

```
                   ('Scaller', StandardScaler(), ['Size', 'Bathroom',
'Bedroom']),

]
# Create ColumnTransformer
feature_engineering =
ColumnTransformer(transformers=feature_transformations,
remainder='passthrough')
```

IV. Modeling

Modeling

The Modeling phase stands as the juncture where data science transforms data into actionable insights. Through the tasks of selecting modeling techniques, generating test designs, building models, and assessing their performance, data scientists contribute to the creation of a model that aligns with the project's objectives. While the CRISP-DM Guide encourages iterative refinement until the best model is found, the practical reality often involves achieving a "good enough" model, advancing through the lifecycle, and continuously improving the model in subsequent iterations. This flexible approach ensures that the project remains adaptable to changing needs and ever-evolving datasets.

Base Model

```
# Define the pipeline
lr_transformations = Pipeline([
    ('feature_engineering', feature_engineering),
    ('pca', PCA(n_components=60, whiten=True)),
    ('regr', LinearRegression())
])

# Fit the model
lr_transformations.fit(X_train, y_train)

# Predict using the trained model
y_pred = lr_transformations.predict(X_test)

# Evaluate the model using cross-validation and ShuffleSplit
cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=42)

# R2 score
r2_scores = cross_val_score(lr_transformations, X_train, y_train,
cv=cv, scoring='r2')
print(f'R2 scores: {r2_scores}')
print(f'Mean R2 score: {r2_scores.mean()}')

# Mean Absolute Error (MAE)
```

```
mae_scores = cross_val_score(lr_transformations, X_train, y_train,
cv=cv, scoring='neg_mean_absolute_error')
mae_scores = -mae_scores  # Take the negative and convert to positive
print(f'MAE scores: {mae_scores}')
print(f'Mean MAE: {mae_scores.mean()}')

# Root Mean Squared Error (RMSE)
rmse_scores = cross_val_score(lr_transformations, X_train, y_train,
cv=cv, scoring='neg_root_mean_squared_error')
rmse_scores = -rmse_scores  # Take the negative and convert to
positive
print(f'RMSE scores: {rmse_scores}')
print(f'Mean RMSE: {rmse_scores.mean()}')
```

```
R2 scores: [0.28988038 0.28067574 0.3443933  0.32960179 0.30887318]
Mean R2 score: 0.31068487781860565
MAE scores: [16749520.03254678 17001833.04615616 17108410.26626476
15731998.76657406
 17289835.32797788]
Mean MAE: 16776319.487903928
RMSE scores: [34191901.12617593 37763612.33388431 35452583.16581678
31569276.8294648
 38858630.49605058]
Mean RMSE: 35567200.79027848
```

```
dump(lr_transformations, 'lr_model.joblib')
```

```
['lr_model.joblib']
```

Model Selection and Tunning

RandomForest regressor

```
# Define the pipeline
rf_transformations = Pipeline([
    ('feature_engineering', feature_engineering),
    ('pca', PCA(n_components=60, whiten=True)),
    ('regr', RandomForestRegressor())
])

# Fit the model
rf_transformations.fit(X_train, y_train)

# Predict using the trained model
y_pred = rf_transformations.predict(X_test)

# Evaluate the model using cross-validation and ShuffleSplit
cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=42)

# R2 score
```

```
r2_scores = cross_val_score(rf_transformations, X_train, y_train, 
cv=cv, scoring='r2')
print(f'R2 scores: {r2_scores}')
print(f'Mean R2 score: {r2_scores.mean()}')

# Mean Absolute Error (MAE)
mae_scores = cross_val_score(rf_transformations, X_train, y_train, 
cv=cv, scoring='neg_mean_absolute_error')
mae_scores = -mae_scores  # Take the negative and convert to positive
print(f'MAE scores: {mae_scores}')
print(f'Mean MAE: {mae_scores.mean()}')

# Root Mean Squared Error (RMSE)
rmse_scores = cross_val_score(rf_transformations, X_train, y_train, 
cv=cv, scoring='neg_root_mean_squared_error')
rmse_scores = -rmse_scores  # Take the negative and convert to 
positive
print(f'RMSE scores: {rmse_scores}')
print(f'Mean RMSE: {rmse_scores.mean()}')
```

```
R2 scores: [0.23560641 0.00256675 0.25876112 0.22173936 0.23621453]
Mean R2 score: 0.190977632582947
MAE scores: [16126242.57586013 17187355.36349905 16994306.73782054 
15833687.09498284
 17666750.77230712]
Mean MAE: 16761668.508893937
RMSE scores: [35605604.10135386 45812202.84707737 37829264.05918647 
34213206.84472663
 40840664.77074244]
Mean RMSE: 38860188.52461735
```

```
dump(rf_transformations, 'rf_model.joblib')
```

```
['rf_model.joblib']
```

XG boostRegressor

```
# XGBoost transformations pipeline
xg_transformations = Pipeline([
    ('feature_engineering', feature_engineering),
    ('pca', PCA()),
    ('regr', XGBRegressor())
])

# Parameter grid for PCA components and XGBoost hyperparameters
param_grid = {
    'pca__n_components': [25, 50, 75, 100],
    'regr__n_estimators': [50, 100, 150, 200],
    'regr__learning_rate': [0.01, 0.1, 0.2, 0.3],
    'regr__max_depth': [3, 5, 7, 9],
```

```
}

# Perform GridSearchCV for XGBoost
xg_grid_search = GridSearchCV(xg_transformations, param_grid,
scoring='neg_mean_squared_error', cv=5, n_jobs=-1)
xg_grid_search.fit(X_train, y_train)

# Get the best components and hyperparameters from the grid search
best_components_xg = xg_grid_search.best_params_['pca__n_components']
best_params_xg = xg_grid_search.best_params_

# Set the best components and hyperparameters to the pipeline
best_params_xg_regr = best_params_xg.get('regr', {})  # Retrieve
'regr' key safely
xg_transformations.set_params(pca__n_components=best_components_xg,

regr__n_estimators=best_params_xg_regr.get('n_estimators', 100),

regr__learning_rate=best_params_xg_regr.get('learning_rate', 0.1),

regr__max_depth=best_params_xg_regr.get('max_depth', 3))

# Evaluate the model using cross-validation and ShuffleSplit
cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=42)

# R2 score
r2_scores = cross_val_score(xg_transformations, X_train, y_train,
cv=cv, scoring='r2')
print(f'R2 scores: {r2_scores}')
print(f'Mean R2 score: {r2_scores.mean()}')

# Mean Absolute Error (MAE)
mae_scores = cross_val_score(xg_transformations, X_train, y_train,
cv=cv, scoring='neg_mean_absolute_error')
mae_scores = -mae_scores  # Take the negative and convert to positive
print(f'MAE scores: {mae_scores}')
print(f'Mean MAE: {mae_scores.mean()}')

# Root Mean Squared Error (RMSE)
rmse_scores = cross_val_score(xg_transformations, X_train, y_train,
cv=cv, scoring='neg_root_mean_squared_error')
rmse_scores = -rmse_scores  # Take the negative and convert to
positive
print(f'RMSE scores: {rmse_scores}')
print(f'Mean RMSE: {rmse_scores.mean()}')

# Fit the best model on the training data
xg_transformations.fit(X_train, y_train)

# Predict using the best model
```

```
y_pred_xg = xg_transformations.predict(X_test)

# Print the best components, hyperparameters, and performance metrics
print(f'Best PCA Components for XGBoost: {best_components_xg}')
print(f'Best XGBoost Hyperparameters: {best_params_xg}')
r2_xg = r2_score(y_test, y_pred_xg)
mae_xg = mean_absolute_error(y_test, y_pred_xg)
rmse_xg = np.sqrt(mean_squared_error(y_test, y_pred_xg))
print(f'r2_score (R2) for XGBoost: {r2_xg * 100}')
print(f'Mean Absolute Error (MAE) for XGBoost: {mae_xg}')
print(f'Root Mean Squared Error (RMSE) for XGBoost: {rmse_xg}')
```

```
R2 scores: [0.2647094  0.22921734 0.31855206 0.27093384 0.28032906]
Mean R2 score: 0.27274834107233226
MAE scores: [15853768.14091231 16074667.91089109 15989463.25304986
14522650.64382956
 16334909.62164074]
Mean MAE: 15755091.914064711
RMSE scores: [34633200.75620892 38822053.30292863 36220977.91826513
32738165.36578636
 39717687.76296422]
Mean RMSE: 36426417.02123065
Best PCA Components for XGBoost: 50
Best XGBoost Hyperparameters: {'pca__n_components': 50,
'regr__learning_rate': 0.1, 'regr__max_depth': 3,
'regr__n_estimators': 50}
r2_score (R2) for XGBoost: 29.54779654291594
Mean Absolute Error (MAE) for XGBoost: 15440150.9270362
Root Mean Squared Error (RMSE) for XGBoost: 38421357.60644769
```

```
dump(xg_transformations, 'Xg_model.joblib')
```

```
['Xg_model.joblib']
```

Lasso and Ridge

```
# Lasso pipeline
lasso_pipeline = Pipeline([
    ('feature_engineering', feature_engineering),
    ('pca', PCA()),
    ('lasso', Lasso())
])

# Ridge pipeline
ridge_pipeline = Pipeline([
    ('feature_engineering', feature_engineering),
    ('pca', PCA()),
    ('ridge', Ridge())
])
```

```
# Parameter grids for Lasso and Ridge
lasso_param_grid = {
    'pca__n_components': np.arange(1, 101),  # PCA components
    'lasso__alpha': [0.001, 0.01, 0.1, 1.0]  # Lasso alpha values
}

ridge_param_grid = {
    'pca__n_components': np.arange(1, 101),  # PCA components
    'ridge__alpha': [0.001, 0.01, 0.1, 1.0]  # Ridge alpha values
}

# Define scoring function (you can use R2, MAE, or RMSE)
scoring = make_scorer(mean_squared_error, greater_is_better=False)

# Perform GridSearchCV for Lasso
lasso_grid_search = GridSearchCV(lasso_pipeline, lasso_param_grid,
scoring=scoring, cv=5)
lasso_grid_search.fit(X_train, y_train)

# Perform GridSearchCV for Ridge
ridge_grid_search = GridSearchCV(ridge_pipeline, ridge_param_grid,
scoring=scoring, cv=5)
ridge_grid_search.fit(X_train, y_train)

# Get the best components and hyperparameters from the grid search for
Lasso
best_components_lasso =
lasso_grid_search.best_params_['pca__n_components']
best_alpha_lasso = lasso_grid_search.best_params_['lasso__alpha']

# Get the best components and hyperparameters from the grid search for
Ridge
best_components_ridge =
ridge_grid_search.best_params_['pca__n_components']
best_alpha_ridge = ridge_grid_search.best_params_['ridge__alpha']

# Set the best components and hyperparameters to the pipelines
lasso_pipeline.set_params(pca__n_components=best_components_lasso,
lasso__alpha=best_alpha_lasso)
ridge_pipeline.set_params(pca__n_components=best_components_ridge,
ridge__alpha=best_alpha_ridge)

# Evaluate Lasso model using cross-validation and ShuffleSplit
cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=42)

# Lasso - Mean Squared Error
lasso_scores = cross_val_score(lasso_pipeline, X_train, y_train,
cv=cv, scoring='neg_mean_squared_error')
lasso_scores = -lasso_scores  # Take the negative and convert to
positive
```

```
print(f'Mean Squared Error for Lasso: {lasso_scores.mean()}')

# Evaluate Ridge model using cross-validation and ShuffleSplit
# Ridge - Mean Squared Error
ridge_scores = cross_val_score(ridge_pipeline, X_train, y_train,
cv=cv, scoring='neg_mean_squared_error')
ridge_scores = -ridge_scores  # Take the negative and convert to
positive
print(f'Mean Squared Error for Ridge: {ridge_scores.mean()}')
```

```
Mean Squared Error for Lasso: nan
Mean Squared Error for Ridge: nan
```

```
dump(lasso_pipeline, 'lasso_model.joblib')
dump(ridge_pipeline, 'ridge_model.joblib')
```

```
['ridge_model.joblib']
```

RandomForestRegressor Tuning

```
# Define parameter grid for RandomForestRegressor
param_grid = {
    'pca__n_components': [50, 100, 150],
    'regr__n_estimators': [50, 100, 150],
    'regr__max_depth': [None, 10, 20],
    'regr__min_samples_split': [2, 5, 10]
}

# RandomForestRegressor transformations pipeline
rf_transformations_tuned = Pipeline([
    ('feature_engineering', feature_engineering),
    ('pca', PCA(whiten=True)),
    ('regr', RandomForestRegressor())
])

# Use ShuffleSplit for cross-validation
cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=42)

# Perform GridSearchCV for RandomForestRegressor
grid_search = GridSearchCV(rf_transformations_tuned, param_grid,
scoring='neg_mean_squared_error', cv=cv, n_jobs=-1)
grid_search.fit(X_train, y_train)

# Get the best parameters from the grid search
best_params = grid_search.best_params_

# Set the best parameters to the pipeline and fit on the training data
rf_transformations_tuned.set_params(**best_params)
rf_transformations_tuned.fit(X_train, y_train)
```

```
# Evaluate RandomForestRegressor model using cross-validation
rf_scores = cross_val_score(rf_transformations_tuned, X_train,
y_train, cv=cv, scoring='neg_mean_squared_error')
rf_scores = -rf_scores  # Take the negative and convert to positive

# Predict using the best model
y_pred = rf_transformations_tuned.predict(X_test)

# Print the best parameters and performance metrics
print(f'Best Parameters: {best_params}')
print(f'Mean Squared Error for RandomForestRegressor:
{rf_scores.mean()}')

r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print(f'r2_score (R2): {r2 * 100}')
print(f'Mean Absolute Error (MAE): {mae}')
print(f'Root Mean Squared Error (RMSE): {rmse}')

dump(rf_transformations_tuned, 'rf_model_tuned.joblib')
```

V. Evaluation

Base Model

The base model, which incorporates feature engineering, dimensionality reduction via PCA (Principal Component Analysis) with 60 components, and linear regression, has been evaluated using various performance metrics. The R2 score, indicating the proportion of variance in the dependent variable explained by the model, is approximately 28.60%. The Mean Absolute Error (MAE) is calculated to be approximately 16,576,773.67, representing the average absolute difference between the predicted and actual values. The Root Mean Squared Error (RMSE), measuring the standard deviation of the residuals, is approximately 38,677,757.14. These metrics provide insights into the model's predictive accuracy and precision.

RandomForest Regressor

The Random Forest Regressor model, implemented through the rf_transformations pipeline, has been evaluated on the provided metrics. The R2 score, a measure of the model's goodness of fit, is approximately 12.86%, indicating that the model explains a modest proportion of the variance in the target variable. The Mean Absolute Error (MAE) is found to be approximately 17,382,498.84, representing the average absolute difference between predicted and actual values. This metric provides insight into the model's accuracy, with lower values indicating better performance. Finally, the Root Mean Squared Error (RMSE) is approximately 42,729,236.45, serving as a measure of the model's prediction error, with lower values indicating better predictive accuracy.

VI. Deployment