Institute of Technology of Cambodia
Department Information Technology and Communication

REPORT PROJECT : Public Transportation

LECTURER : UN Lykong

SUBJECT : AI

GROUP : 03

NAME	ID
KONG VONGPISITH	e20190457
ROTHA DAPRAVITH	e20190915
YORNG TONGHY	e20191313

2023-2024

Table of Contents

I. Introduction

A recent study employed machine-learning tools like Random Forest and Decision Trees, Linear Regression to enhance public transportation. The research, using a diverse dataset, applied linear regression to reveal connections between factors and transit times. Random Forest for both classification and regression tasks, and Decision Trees provided clear insights into variables affecting route efficiency. Using Python, the study addressed both regression and classification issues, predicting aspects like effective routes and travel delays. This marks a significant move towards improving urban sustainability through data-driven optimizations in public transportation systems.

II. Background and Problem Statement

The objective of the project is to use machine learning to improve public transportation networks. Its main objective is to estimate passenger loads, improve routes, and predict delays by evaluating past transit data. Increasing schedule dependability, decreasing congestion, and effectively managing resources are the main obstacles. The method provides insights for more intelligent, effective public transportation operations in metropolitan settings by utilizing algorithms like Decision Trees, Random Forest, and Linear Regression.

III. Proposed Methodology and Algorithms

In this section, we will discuss the methodology and algorithms that we use throughout the project. There are a few steps follow:

1. Technologies

- Python programming language.
- Scikit learn library for train dataset model.
- Kiggle for collection data.

2. Machine learning algorithms

- **Linear regression** is a statistical modeling technique used to predict a continuous outcome variable based on one or more predictor variables
- **Random Forest** builds multiple decision trees during training and combines their predictions. It improves accuracy and robustness, making it effective for

various tasks like classification and regression

- **Decision tree** is a popular supervised machine learning algorithm used for both classification and regression tasks

3. Method and flow process

In order to build the model, there are several steps to follow:

- Data collection for apply dataset.
- Exploring and analysis data.
- Preparing data.
- Training model and Evaluate performance.
- Comparison between models.

IV. Implementation and Result

1. Exploring and analysis data

a. Data exploring

This dataset contains 1043534 documents and 11 features with 1 classification result. All documents in the dataset contain no non-value andit has 11 features type as integer and 1 as float shown in Figure 1.

```
1  data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1043534 entries, 0 to 1043533
Data columns (total 12 columns):
 #   Column       Non-Null Count    Dtype
---  ------       --------------    -----
 0   Unnamed: 0   1043534 non-null  int64
 1   insert_date  1043534 non-null  object
 2   origin       1043534 non-null  object
 3   destination  1043534 non-null  object
 4   start_date   1043534 non-null  datetime64[ns]
 5   end_date     1043534 non-null  object
 6   train_type   1043534 non-null  object
 7   price        970506 non-null   float64
 8   train_class  1040288 non-null  object
 9   fare         1040288 non-null  object
 10  month        1043534 non-null  int64
 11  year         1043534 non-null  int64
dtypes: datetime64[ns](1), float64(1), int64(3), object(7)
memory usage: 95.5+ MB
```

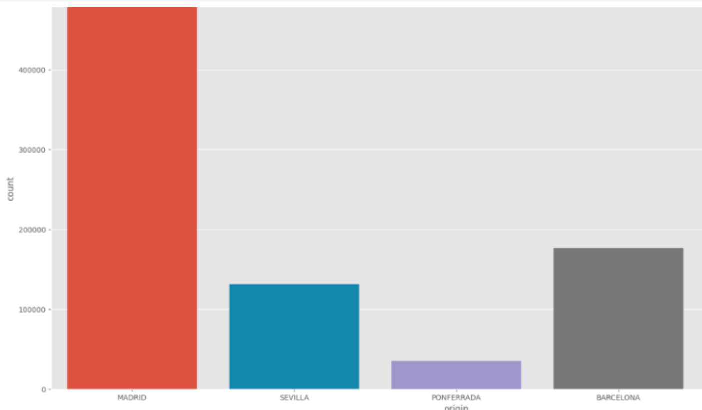
```
1  data.describe()
```

	Unnamed: 0	price	month	year
count	1.043534e+06	970506.000000	1.043534e+06	1043534.0
mean	5.217665e+05	62.854765	4.880514e+00	2019.0
std	3.012425e+05	25.723718	4.785939e-01	0.0
min	0.000000e+00	16.600000	4.000000e+00	2019.0
25%	2.608832e+05	43.550000	5.000000e+00	2019.0
50%	5.217665e+05	60.300000	5.000000e+00	2019.0
75%	7.826498e+05	76.300000	5.000000e+00	2019.0
max	1.043533e+06	214.200000	6.000000e+00	2019.0

b. Data analysis

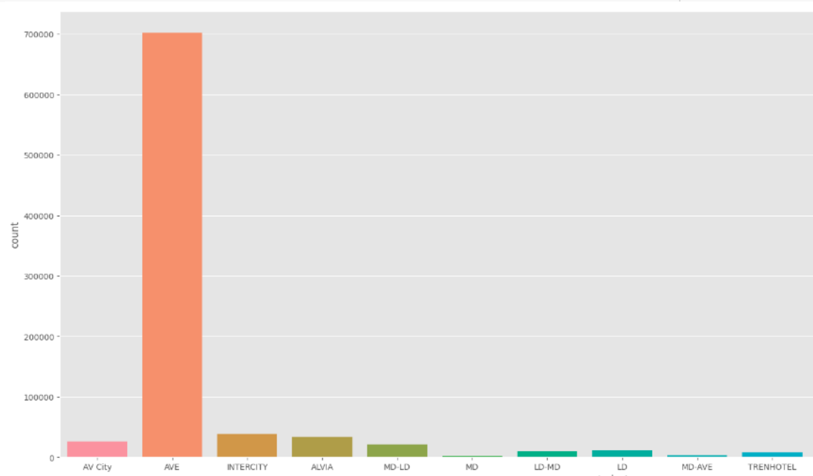
❖ This figure is showing about the graph that counts for 'origin' column.

```
[ ] # The people finish trip from this stations.
    import matplotlib.pyplot as plt
    import seaborn as sns

    # Assuming 'df' is your DataFrame and it's already defined.

    plt.figure(figsize=(20, 10))  # Set the size of the figure
    sns.countplot(x='origin', data=df)  # Create a count plot for the 'origin' column
    plt.show()  # Corrected to 'plt.show()' to actually display the plot
```



❖ Another figure is showing the train type visualization

```
# the train type more popular there/ more using

import matplotlib.pyplot as plt
import seaborn as sns

# Ensure 'df' is your DataFrame and it's already defined.
# Also, ensure 'df' contains a column named 'train_type'.

plt.figure(figsize=(25, 10))  # Set the size of the figure
sns.countplot(x='train_type', data=df)  # Create a count plot for the 'train_type' column
plt.show()  # Call plt.show() to display the plot
```



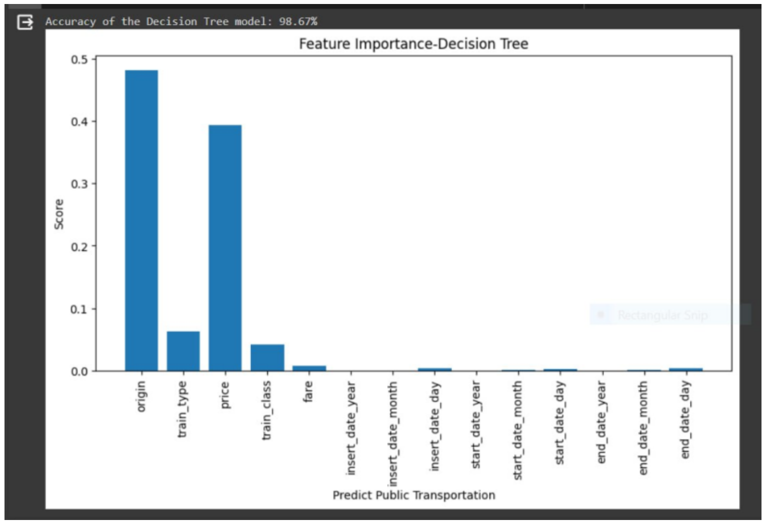
2. Training model and evaluation

For training, we use 3 algorithms for testing

- Decision Tree

We are trying to find the accuracy of the decision tree model and feature importance

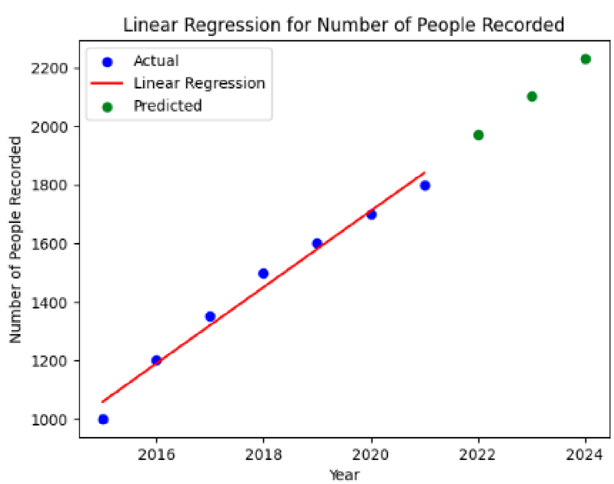
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler, LabelEncoder

# Load the dataset
file_path = '/content/public_transportation_data.csv'
data = pd.read_csv(file_path)

# Preprocess the Data
# Convert date columns to datetime
data['insert_date'] = pd.to_datetime(data['insert_date'])
data['start_date'] = pd.to_datetime(data['start_date'])
data['end_date'] = pd.to_datetime(data['end_date'])

# Extract features from date columns (e.g., year, month, day)
data['insert_date_year'] = data['insert_date'].dt.year
data['insert_date_month'] = data['insert_date'].dt.month
data['insert_date_day'] = data['insert_date'].dt.day
data['start_date_year'] = data['start_date'].dt.year
data['start_date_month'] = data['start_date'].dt.month
data['start_date_day'] = data['start_date'].dt.day
data['end_date_year'] = data['end_date'].dt.year
data['end_date_month'] = data['end_date'].dt.month
data['end_date_day'] = data['end_date'].dt.day

# Drop the original date columns
data = data.drop(columns=['insert_date', 'start_date', 'end_date'])

# Handling missing values
for col in data.columns:
    if data[col].dtype == 'object':
        data[col] = data[col].fillna(data[col].mode().iloc[0])
    else:
        data[col] = data[col].fillna(data[col].mean())

# Encoding categorical features
label_encoder = LabelEncoder()
for column in data.select_dtypes(include=['object']).columns:
    data[column] = label_encoder.fit_transform(data[column])

# Selecting features and target
features = ['origin', 'train_type', 'price', 'train_class', 'fare',
            'insert_date_year', 'insert_date_month', 'insert_date_day',
            'start_date_year', 'start_date_month', 'start_date_day',
            'end_date_year', 'end_date_month', 'end_date_day']
X = data[features]
y = data['destination']

# Splitting the dataset into the Training set and Test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scaling the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize the Decision Tree Classifier
decision_tree = DecisionTreeClassifier(random_state=42)

# Fit the model to the training data
decision_tree.fit(X_train, y_train)

# Predicting the Test set results
y_pred = decision_tree.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
accuracy_percentage = accuracy * 100

# Print the accuracy as a percentage
print(f'Accuracy of the Decision Tree model: {accuracy_percentage:.2f}%')

# Plotting a graph (e.g., feature importance)
feature_importance = decision_tree.feature_importances_
plt.figure(figsize=(10, 5))

# plt.bar(range(len(feature_importance)), feature_importance)
plt.bar(range(len(features)), feature_importance)
plt.xticks(range(len(features)), features, rotation=90)
plt.xlabel('Predict Public Transportation')
plt.ylabel('Score')
plt.title('Feature Importance-Decision Tree')
plt.show()
```



❖ Linear regression

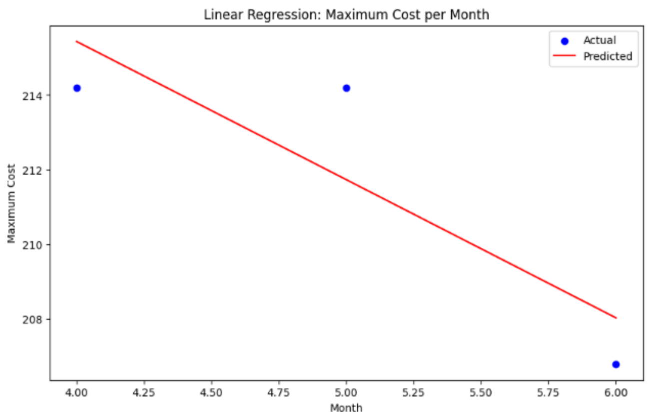
- We are trying to predict the amount of people record in each year.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Historical data
years = [2015, 2016, 2017, 2018, 2019, 2020, 2021]  # Years
people = [1000, 1200, 1350, 1500, 1600, 1700, 1800]  # Number of people recordec

# Convert the data to numpy arrays
X = np.array(years).reshape(-1, 1)  # Reshape to a 2D array
y = np.array(people)

# Create and fit the linear regression model
model = LinearRegression()
model.fit(X, y)

# Predict the number of people for future years
future_years = np.array([2022, 2023, 2024]).reshape(-1, 1)
predicted_people = model.predict(future_years)

# Plot the historical data and the linear regression line
plt.scatter(years, people, color='b', label='Actual')
plt.plot(years, model.predict(X), color='r', label='Linear Regression')
plt.scatter(future_years, predicted_people, color='g', label='Predicted')

plt.xlabel('Year')
plt.ylabel('Number of People Recorded')
plt.title('Linear Regression for Number of People Recorded')
plt.legend()
plt.show()
```



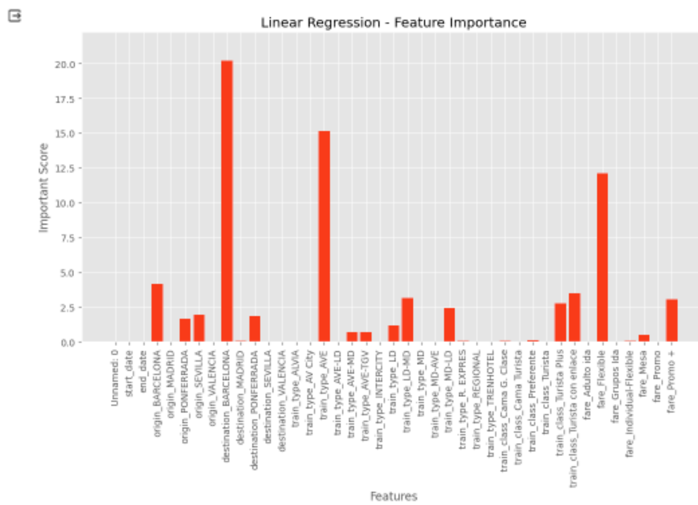
- We are trying to predict maximum cost per month.

```python
# Load the dataset
data = pd.read_csv('/content/public_transportation_data.csv')

# Convert the 'start_date' column to datetime format
data['start_date'] = pd.to_datetime(data['start_date'])

# Extract the month and year from the 'start_date' column
data['month'] = data['start_date'].dt.month
data['year'] = data['start_date'].dt.year

# Calculate the maximum cost for each month
max_costs = data.groupby(['year', 'month'])['price'].max().reset_index()

# Perform linear regression
X = max_costs['month'].values.reshape(-1, 1)
y = max_costs['price'].values.reshape(-1, 1)
regressor = LinearRegression()
regressor.fit(X, y)

# Predict the maximum cost for every month
predicted_costs = regressor.predict(X)

# Plot the predicted costs
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='blue', label='Actual')
plt.plot(X, predicted_costs, color='red', label='Predicted')
plt.xlabel('Month')
plt.ylabel('Maximum Cost')
plt.title('Linear Regression: Maximum Cost per Month')
plt.legend()
plt.show()
```



- Linear Regression – Feature Importance

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import OneHotEncoder

# Load the dataset
data = pd.read_csv('/content/public_transportation_data.csv')  # Replace 'your_dataset.csv' with t

# Exclude 'insert_date' column
data = data.drop(['insert_date'], axis=1)

# Convert datetime columns to numeric representation
date_columns = ['start_date', 'end_date']
for column in date_columns:
    data[column] = pd.to_datetime(data[column]).astype(int)

# Separate the features (X) and target variable (y)
X = data.drop(['price'], axis=1)  # Exclude the target variable 'price'
y = data['price']

# Remove rows with missing values in the target variable
X = X[~np.isnan(y)]
y = y[~np.isnan(y)]

# Perform one-hot encoding on categorical columns
categorical_cols = ['origin', 'destination', 'train_type', 'train_class', 'fare']
X_encoded = pd.get_dummies(X, columns=categorical_cols)

# Fit the linear regression model
model = LinearRegression()
model.fit(X_encoded, y)

# Retrieve the coefficients
coefficients = model.coef_

# Retrieve the feature names
feature_names = X_encoded.columns.tolist()

# Create a bar plot of feature importance
plt.figure(figsize=(12, 6))
plt.bar(feature_names, coefficients)
plt.xlabel('Features')
plt.ylabel('Important Score')
plt.title('Linear Regression - Feature Importance')
plt.xticks(rotation=90)
plt.ylim(bottom=0)  # Set the lower limit of the y-axis to zero
plt.show()
```



❖ Random Forest

Preparing the value

```
# Preparing the features (X) and target variable (Y)
X = df.drop(['price'], axis=1)
Y = df['price']

# Display the first few rows of X and Y to verify
print(X.head())
print(Y.head())
```

```
   Unnamed: 0          insert_date  origin destination           start_date  \
0           0  2019-04-19 05:31:43  MADRID     SEVILLA  2019-05-29 06:20:00
1           1  2019-04-19 05:31:43  MADRID     SEVILLA  2019-05-29 07:00:00
2           2  2019-04-19 05:31:43  MADRID     SEVILLA  2019-05-29 07:30:00
3           3  2019-04-19 05:31:43  MADRID     SEVILLA  2019-05-29 08:00:00
4           4  2019-04-19 05:31:43  MADRID     SEVILLA  2019-05-29 08:30:00

              end_date train_type train_class   fare
0  2019-05-29 09:16:00    AV City     Turista  Promo
1  2019-05-29 09:32:00        AVE     Turista  Promo
2  2019-05-29 09:51:00        AVE     Turista  Promo
3  2019-05-29 10:32:00        AVE  Preferente  Promo
4  2019-05-29 11:14:00      ALVIA     Turista  Promo
0    38.55
1    53.40
2    47.30
3    69.40
4      NaN
Name: price, dtype: float64
```

- Split dataset

```
[13] X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
```

Step3: Preapring the variables

```
[12] # Preparing the features (X) and target variable (Y)
     X = df.drop(['price'], axis=1)
     Y = df['price']

     # Display the first few rows of X and Y to verify
     print(X.head())
     print(Y.head())
```

```
   Unnamed: 0          insert_date  origin destination           start_date  \
0           0  2019-04-19 05:31:43  MADRID     SEVILLA  2019-05-29 06:20:00
1           1  2019-04-19 05:31:43  MADRID     SEVILLA  2019-05-29 07:00:00
2           2  2019-04-19 05:31:43  MADRID     SEVILLA  2019-05-29 07:30:00
3           3  2019-04-19 05:31:43  MADRID     SEVILLA  2019-05-29 08:00:00
4           4  2019-04-19 05:31:43  MADRID     SEVILLA  2019-05-29 08:30:00

              end_date train_type train_class   fare
0  2019-05-29 09:16:00    AV City     Turista  Promo
1  2019-05-29 09:32:00        AVE     Turista  Promo
2  2019-05-29 09:51:00        AVE     Turista  Promo
3  2019-05-29 10:32:00        AVE  Preferente  Promo
4  2019-05-29 11:14:00      ALVIA     Turista  Promo
0    38.55
1    53.40
2    47.30
3    69.40
4      NaN
Name: price, dtype: float64
```

- Train Model

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Assuming the best_random_state has been determined in a previous step of your analysis
best_random_state = 100

# Initialize the RandomForestClassifier with the best found random state
rf = RandomForestClassifier(random_state=best_random_state, n_jobs=-1, max_depth=7,
                            min_samples_leaf=1, min_samples_split=5,
                            n_estimators=50, oob_score=True)

# Train the model
rf.fit(X_train, y_train)

# Predict on the test set
y_pred = rf.predict(X_test)

# Calculate the accuracy and classification report
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

# Print the accuracy as a percentage with 2 decimal places
print("Accuracy: {:.2f}%".format(accuracy * 100))
print("Classification Report:\n", report)
```

```
Accuracy: 92.02%
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.89      0.93     34736
           1       1.00      1.00      1.00    106241
           2       0.79      0.65      0.72      6300
           3       0.70      0.97      0.81     30726
           4       0.92      0.68      0.78     30704

    accuracy                           0.92    208707
   macro avg       0.88      0.84      0.85    208707
weighted avg       0.93      0.92      0.92    208707
```

- Data Processing

```python
[5] # Drop rows with NaN values as a quick fix
    X_train = X_train.dropna()
    y_train = y_train.dropna()

    # Ensure y_train is of type integer if it's a classification label
    y_train = y_train.astype(int)
```

```python
# Check for null values
print(X_train.isnull().sum())
print(y_train.isnull().sum())
```

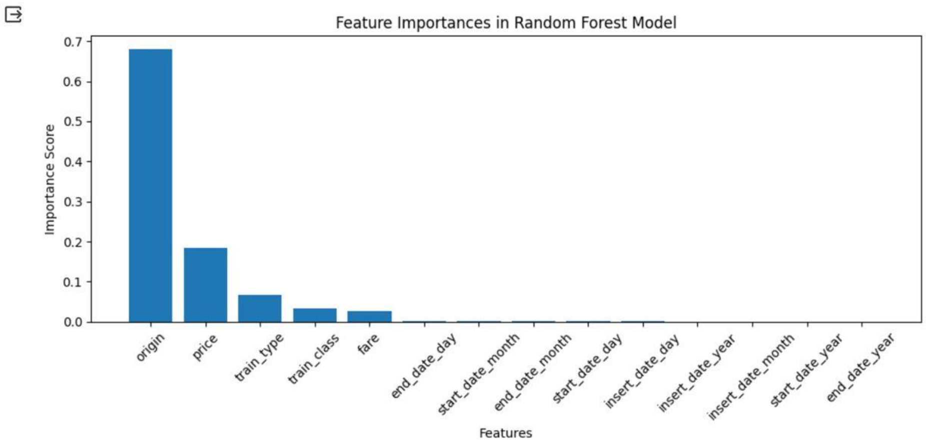
```
Unnamed: 0       0
insert_date      0
origin           1
destination      1
start_date       1
end_date         1
train_type       1
train_class    741
fare           741
dtype: int64
9822
```

```
[ ] Start coding or generate with AI.
```

❖ Feature Importance

Plot Graph

```python
[215]
import matplotlib.pyplot as plt

# Feature importances from the model
feature_importances = rf.feature_importances_

# Assuming X includes the preprocessed features of your dataset
feature_names = X.columns  # Ensure X has the correct columns after preprocessing
feature_importance_dict = dict(zip(feature_names, feature_importances))

# Sort features by importance
sorted_features = sorted(feature_importance_dict.items(), key=lambda x: x[1], reverse=True)

# Create a bar chart for feature importances
plt.figure(figsize=(10, 5))
plt.bar(range(len(sorted_features)), [val[1] for val in sorted_features], tick_label=[val[0] for val in sorted_features])
plt.xticks(rotation=45)
plt.xlabel('Features')
plt.ylabel('Importance Score')
plt.title('Feature Importances in Random Forest Model')
plt.tight_layout()
plt.show()
```



❖ Compare between Models

From we got so far, we can see that **Decision Tree** is more accurate than **Random Forest** since **Linear Regression** cannot make comparison because it is designed for predicting continuous numeric values, not classification tasks with accuracy percentages. It models the relationship between independent and dependent variables using a linear equation, suitable for predicting a continuous outcome.

V. Conclusion

1. Summary what we have done
2. Compare each model
3. Draw results of each model

VI. Reference

https://www.kaggle.com/code/qusaybtoush1990/spain-public-transportation#Make-group-by-and-fitter

VII. Annex

The train type more popular there/ more using:

- AVE 70%
- ALVIA 7%
- REGIONAL 5%
- Other train less 5 %

```
# any relationship between train type and price ?

plt.figure(figsize=(20,10))
sns.lineplot(data=df,x="train_type",y="price")
```

- The people prefer train class Turista because faster and cheaper