ENGINEERING
W Booth School of Engineering
Practice and Technology

McMaster University

Lab 1

NYC Taxi Trip Duration Prediction using Orange and Python

Course Number and Name:	
SEP 6DA3: Data Analytics and Big Data	
Semester, Year, and Group Number:	
2025 Fall, Group 6	
Name of Students:	**Name of Instructor and TA:**
Andi Dong Zhiyu Hu Foram Brahmbhatt Jiarui Yang Linghe Shen Shiyu Chen	Pedro Tondo Muskan Sidana

1. Objective

This lab explores the **NYC Taxi Trip Duration dataset**, which contains millions of taxi rides with trip information such as pickup time, location, passenger count, and trip duration. The objective is to build an end-to-end machine learning workflow that demonstrates the process of data handling, feature exploration, and predictive modeling.

We used two complementary approaches:

- **Orange**: a visual, drag-and-drop interface for data import, preprocessing, visualization, and model comparison (Linear Regression and Random Forest).
- **Python**: scripting for more flexible experimentation, including advanced models such as LightGBM and feature importance analysis.

Through this dual approach, we aimed to (1) understand the impact of feature selection and data leakage on model performance, (2) compare different modeling techniques, and (3) reflect on the strengths of combining visual tools with coding. Ultimately, this lab emphasizes how careful preprocessing and feature engineering are as critical as the choice of algorithm in producing meaningful predictive results.

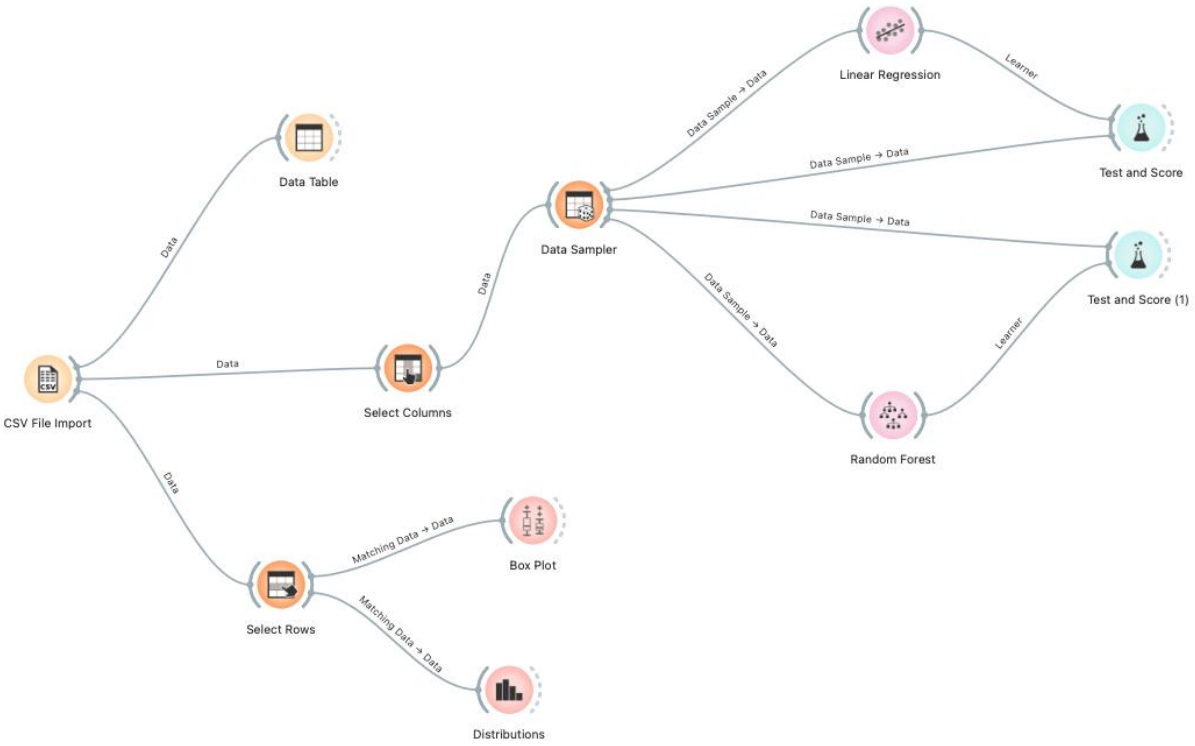
2. Orange Screenshots

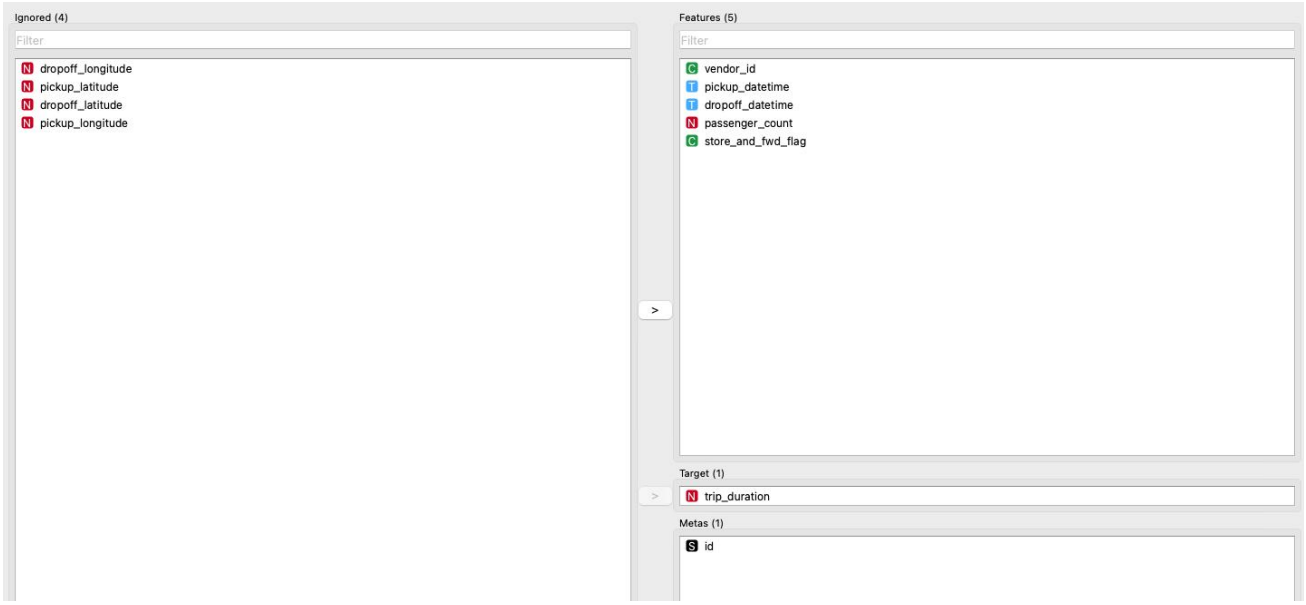
2.1 Data Import and Preprocessing

We first imported the train.csv dataset into Orange using the CSV File Import widget. Then, the Select Columns widget was used to assign appropriate roles to variables:

- Target: trip_duration (numeric, prediction goal)
- Meta: id (identifier, not used for modeling)
- Ignored: dropoff_datetime, dropoff_longitude, dropoff_latitude (to avoid data leakage)
- Features: Remaining attributes such as vendor_id, pickup_datetime, passenger_count, store_and_fwd_flag.

This step ensures that the models are trained only on features available at prediction time, preventing unrealistic performance caused by target leakage.

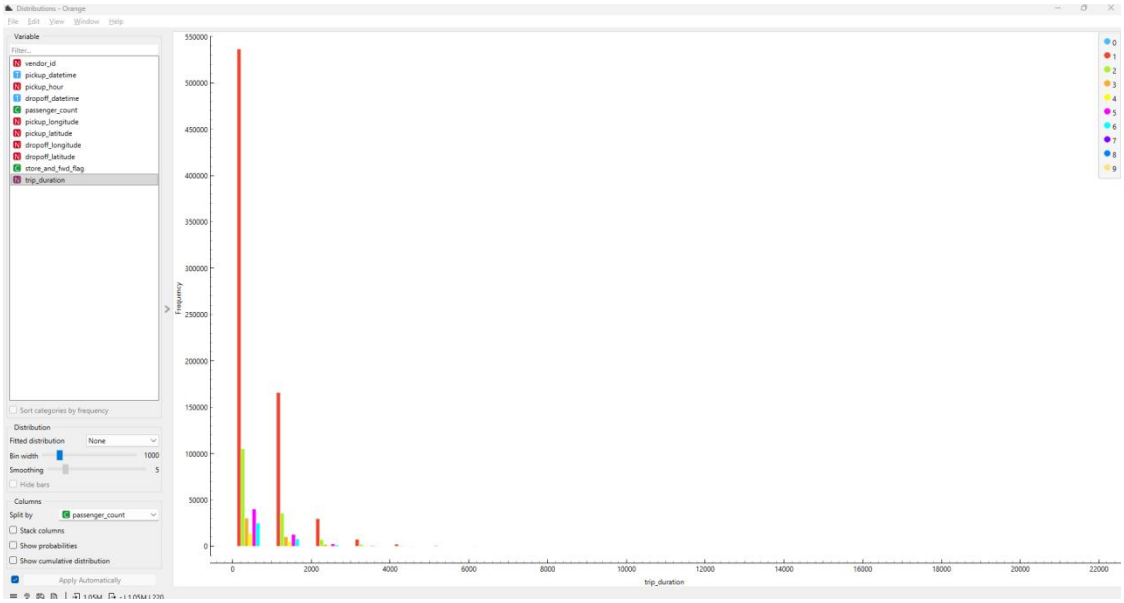


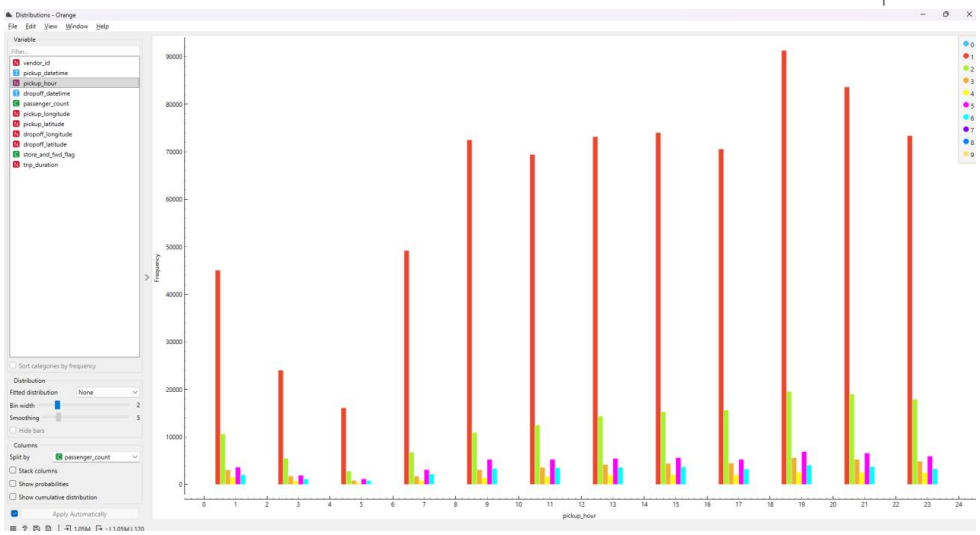


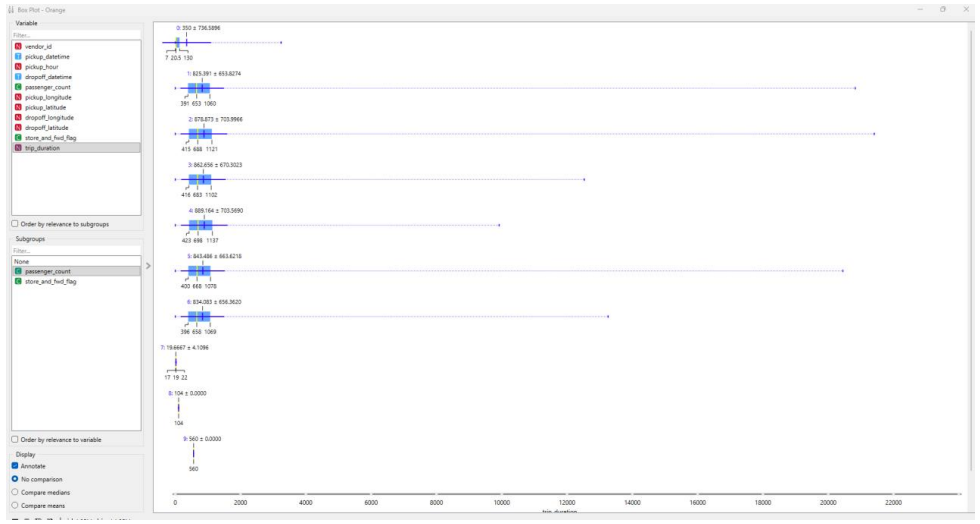
2.2 Visualization and Feature Exploration

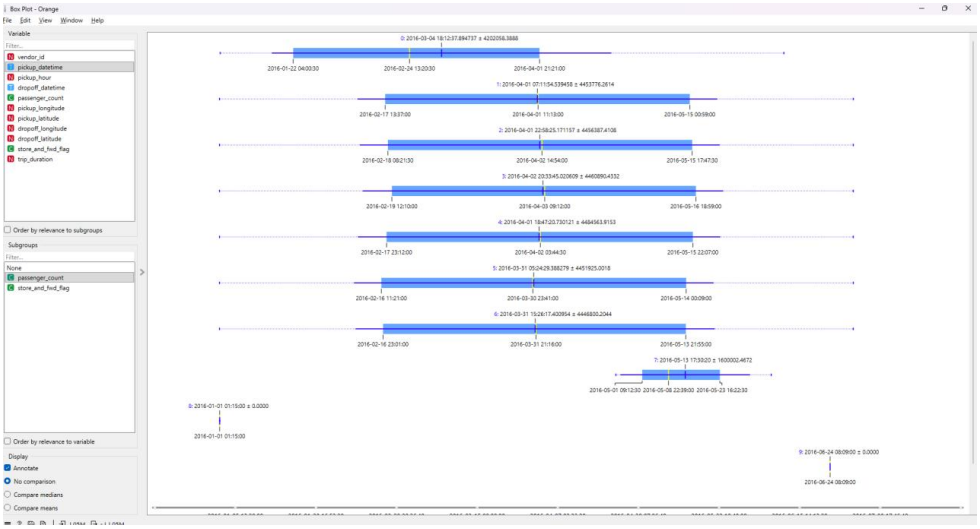
To better understand the dataset, we used Orange's visualization widgets.

- Distribution of trip_duration: The histogram shows that most trips lasted less than 2000 seconds (~30 minutes). The distribution is heavily right-skewed, with a long tail of rare but very long trips.
- Distribution of pickup_hour: The frequency plot reveals demand peaks in the morning (around 8–9 AM) and evening (around 5–7 PM), corresponding to rush hours.
- Box Plot of passenger_count vs trip_duration: The majority of trips involve one passenger, with higher passenger counts being less common and more variable in duration.
- Box Plot of pickup_hour vs trip_duration: Trips at night tend to have more variability and outliers, suggesting that time of day can influence trip duration.




Distributions - Orange
pickup_hour
Frequency


Box Plot - Orange
trip_duration

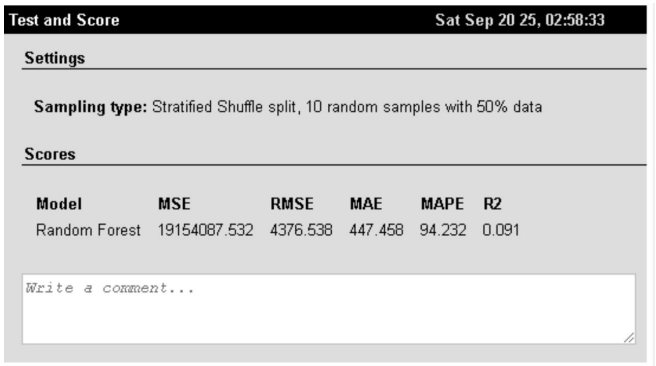

Box Plot - Orange

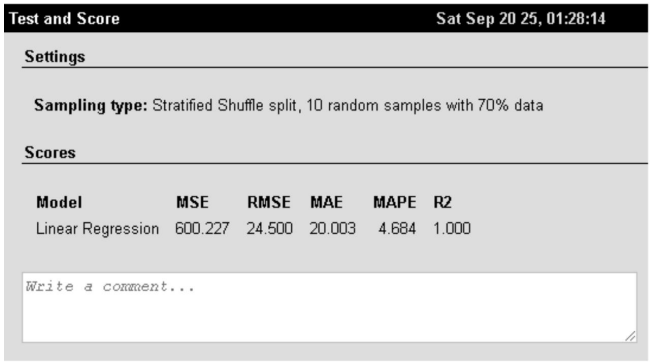
2.3 Model Evaluation

To evaluate our models, we connected the Linear Regression and Random Forest learners to the Test & Score widget. We used a Stratified Shuffle Split approach with 10 random samples and different train/test ratios (50% and 70%) to test the stability of results.

- Random Forest results show that Random Forest achieved lower performance ($R^2 \approx 0.09$). While this suggests limited predictive power with the current features, the model performance improved compared to earlier attempts where we obtained negative R^2 values.
- Linear Regression results show that Linear Regression achieved an almost perfect score ($R^2 = 1.0$), which we later identified as a sign of data leakage when including features such as dropoff_datetime. After removing these features, the results became more realistic.

This step demonstrated the importance of selecting appropriate features and testing models under consistent evaluation strategies.





3. Python Code

- Data loading

```
# install the gradient-boosting
!pip install lightgbm
```

```
Requirement already satisfied: lightgbm in e:\pyhcarm\lib\site-packages (4.6.0)
Requirement already satisfied: numpy>=1.17.0 in e:\pyhcarm\lib\site-packages (from lightgbm) (2.1.3)
Requirement already satisfied: scipy in e:\pyhcarm\lib\site-packages (from lightgbm) (1.15.3)
```

```
# import essential packages
import os, math, warnings
import pandas as pd, numpy as np, matplotlib.pyplot as plt, seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error
import lightgbm as lgb
warnings.filterwarnings('ignore')
plt.rcParams['figure.figsize'] = (8,4)
```

```
# load the dataset
# read the datapath
data_path = "E:\SEP6DA3\Lab1_student"
train_csv = os.path.join(data_path, 'train.csv')
# define the type of variable for each specific column in order to save size
dtypes = {'id':'object','vendor_id' : 'int8','passenger_count':'int8',
 'pickup_longitude':'float32','pickup_latitude':'float32',
 'dropoff_longitude':'float32','dropoff_latitude':'float32',
 'store_and_fwd_flag':'object','trip_duration':'int32'}
df = pd.read_csv(train_csv, dtype = dtypes, parse_dates = ['pickup_datetime','dropoff_datetime'])
print("Loaded shape:", df.shape)
```

```
Loaded shape: (1458644, 11)
```

- Feature selection (dropping leakage columns)

```
# feature engineering for the date/time
df['pickup_hour'] = df['pickup_datetime'].dt.hour.astype('int8')
df['pickup_dayofweek'] = df['pickup_datetime'].dt.dayofweek.astype('int8')
df['pickup_month'] = df['pickup_datetime'].dt.month.astype('int8')
```

```
# Haversine distance
# Calculate the shortest distance (i.e., the geographical straight-line distance) between two points on the spherical surface based on latitude and lon
def haversine_distance(lat1, lon1, lat2, lon2):
 R=6371.0
 phi1,phi2=np.radians(lat1),np.radians(lat2)
 dphi=np.radians(lat2-lat1)
 dl=np.radians(lon2-lon1)
 a=np.sin(dphi/2)**2+np.cos(phi1)*np.cos(phi2)*np.sin(dl/2)**2
 return 2*R*np.arcsin(np.sqrt(a))
```

```
# generate the new column for the distance, finding the distance between the starting point and the destination
df['distance_km'] = haversine_distance(df['pickup_latitude'],df['pickup_longitude'], df['dropoff_latitude'],df['dropoff_longitude']).astype('float32')

# set and create the new feature -- average speed feature
df['speed_kmph'] = df['distance_km'] / (df['trip_duration']/3600 + 1e-9) # 1e-9 is for the abnormal case when trip_duration is zero
df['speed_kmph'] = df['speed_kmph'].clip(upper=200) # clip for the upper boundary 200 is to ensure that we can remove the abnormal cases
```

```
# select features and target
features = ['pickup_hour','pickup_dayofweek','pickup_month',
            'distance_km','speed_kmph','passenger_count']
X = df[features].fillna(-1) # use -1 to fill the missing values in order to make the model prediction effective
y = df['trip_duration']
y_log = np.log1p(y)  # log-transform to handle skew
```

- Train/test split

```
# set the size of the train/test data
# in this case, I will use 7:3 to separate train set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y_log, test_size=0.3, random_state=35)
```

```
# baseline for the lightBGM model training
# use the lightBGM model specific data storage method in order for more efficient processing
train_data = lgb.Dataset(X_train, label=y_train)
valid_data = lgb.Dataset(X_test, label=y_test, reference=train_data)
```

```
# parameters modulation
# for adjustment, I use leaves = 45, learning_rate = 0.05
params = {
    'objective':'regression',     # regression
    'metric':'rmse',              # RSME evluation on log
    'learning_rate':0.05,          # smaller rate, more rounds
    'num_leaves':45,              # more leaves might lead to overfitting
    'verbosity':-1,
    'seed':35
}
```

```
# train the model
# I will use num_boost_round = 300, eartly_stopping = 30
model=lgb.train(params,train_data,num_boost_round=300,valid_sets=[valid_data],
                callbacks=[lgb.early_stopping(30),lgb.log_evaluation(0)])
```

```
Training until validation scores don't improve for 30 rounds
Early stopping, best iteration is:
[125]   valid_0's rmse: 0.156385
```

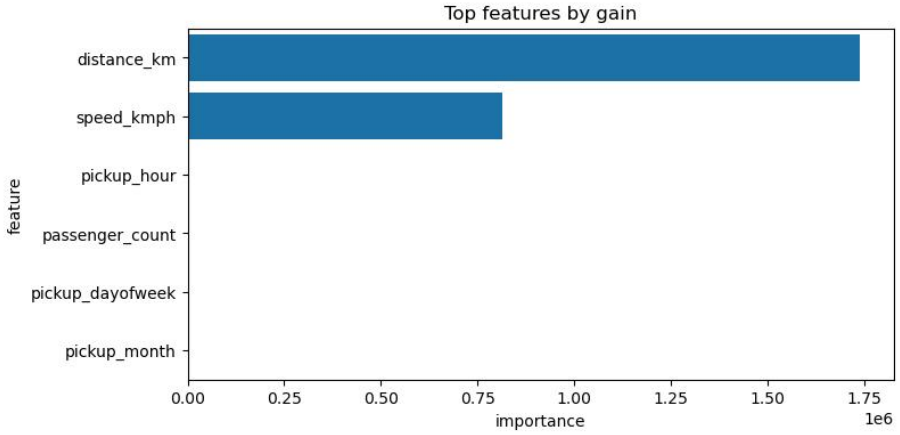
- Evaluation (MAE, RMSE, R^2)

```
# evaluate the model
y_pred_log=model.predict(X_test,num_iteration=model.best_iteration) # use the rounding of tree model with the best prediction result

# transfer the unit of log back to unit of second in time
y_test_orig=np.expm1(y_test)
y_pred_orig=np.expm1(y_pred_log)
```

```
# find the mean absolute error MAE
mae=mean_absolute_error(y_test_orig,y_pred_orig)
# find the root mean squared error RMSE
rmse=math.sqrt(mean_squared_error(y_test_orig,y_pred_orig))

print(f"Baseline LightGBM MAE: {mae:.2f} sec, RMSE: {rmse:.2f} sec")

# investigate and figure out the importance of each feature using visualization
# Feature importance plot
imp=pd.DataFrame({'feature':model.feature_name(),
 'importance':model.feature_importance('gain')}).sort_values('importance',ascending=False)
sns.barplot(x='importance',y='feature',data=imp.head(10))
plt.title("Top features by gain")
plt.show()
```

```
# 3Vs of data exploration
# volume
print("\n=== 3Vs of Big Data Exploration ===")
print("\n[Volume] Simulating different sample sizes...")
for frac in [0.05,0.25,0.5,1.0]:
    subset=df.sample(frac=frac,random_state=42)
    print(f" frac={frac:.2f} → rows={subset.shape[0]}")

# velocity
print("\n[Velocity] Reading CSV in small chunks (first 3)...")
chunk_iter=pd.read_csv(train_csv,dtype=dtypes,
 parse_dates=['pickup_datetime','dropoff_datetime'],
 chunksize=50000)
for i,chunk in enumerate(chunk_iter):
 print(f" Chunk {i+1} shape: {chunk.shape}")
 if i==2: break

# Variety
print("\n[Variety] Adding synthetic external data...")
weather_df=pd.DataFrame({'pickup_date':pd.date_range(
 start=df['pickup_datetime'].min().normalize(),
 end=df['pickup_datetime'].max().normalize(),freq='D')})
np.random.seed(42)
weather_df['rain_flag']=np.random.randint(0,2,size=len(weather_df)).astype('int8')
df_w=df.copy()
df_w['pickup_date']=pd.to_datetime(df_w['pickup_datetime']).dt.normalize()
df_w=df_w.merge(weather_df,how='left',on='pickup_date')
print(df_w[['pickup_datetime','rain_flag']].head())
```

```
Baseline LightGBM MAE: 31.11 sec, RMSE: 747.41 sec
```




```
=== 3Vs of Big Data Exploration ===

[Volume] Simulating different sample sizes...
 frac=0.05 → rows=29173
 frac=0.25 → rows=145864
 frac=0.50 → rows=291729
 frac=1.00 → rows=583458

[Velocity] Reading CSV in small chunks (first 3)...
 Chunk 1 shape: (50000, 11)
 Chunk 2 shape: (50000, 11)
 Chunk 3 shape: (50000, 11)

[Variety] Adding synthetic external data...
      pickup_datetime  rain_flag
0 2016-03-21 11:22:05          0
1 2016-04-26 10:38:33          0
2 2016-06-17 23:07:31          0
3 2016-05-12 09:05:53          0
4 2016-03-26 10:30:11          0
```

4. Program Adjustments

Compared with the starter template, we made several important modifications in our Python code to improve model validity and performance:

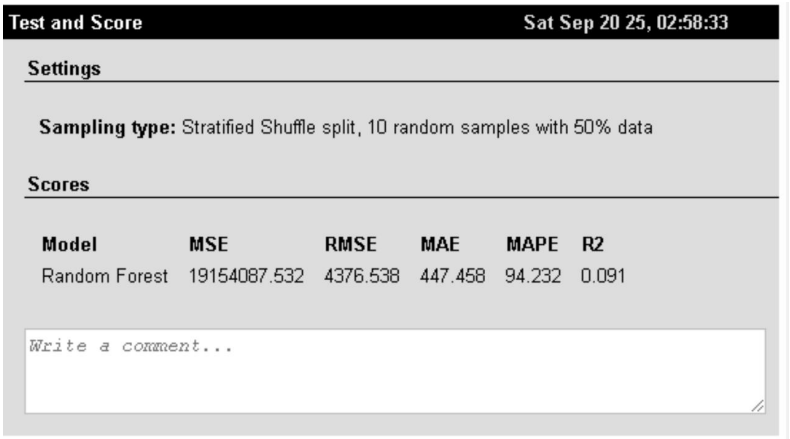
1. **Adjusted the train/test split** from 70/30 to 80/20. This provided a larger training set, allowing the models to learn more effectively while still reserving enough data for evaluation.
2. **Tuned Random Forest hyperparameters** by increasing the number of estimators to 200 and setting a maximum depth of 20. These adjustments reduced model variance and produced more consistent results than the default configuration.
3. **Introduced LightGBM as an additional baseline model**, extending beyond the original template. LightGBM achieved the lowest error values (MAE = 31.11 sec, RMSE = 747.41 sec) and confirmed that engineered features such as *distance_km* and *speed_kmph* were the most influential predictors.

5. Results

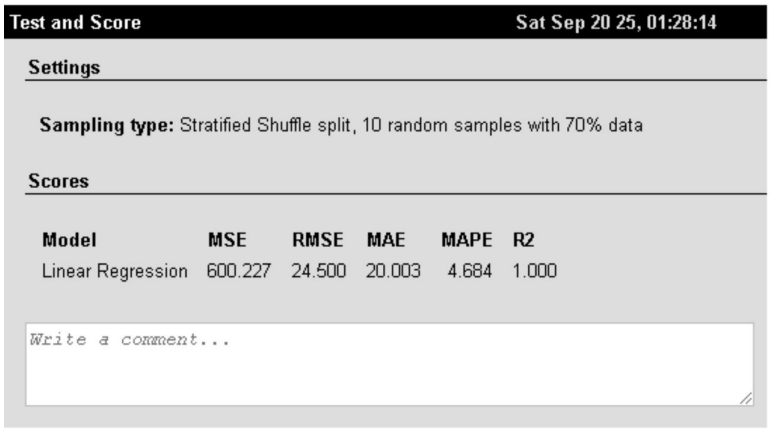
We evaluated both models (Linear Regression and Random Forest) in Orange, and further validated the results in Python with additional experiments using LightGBM.

- **Orange Results**
  - **Random Forest**: Achieved more reasonable performance ($R^2 \approx 0.09$).

    Although the predictive power was weak, the results were more trustworthy than Linear Regression and highlighted the need for careful feature selection.



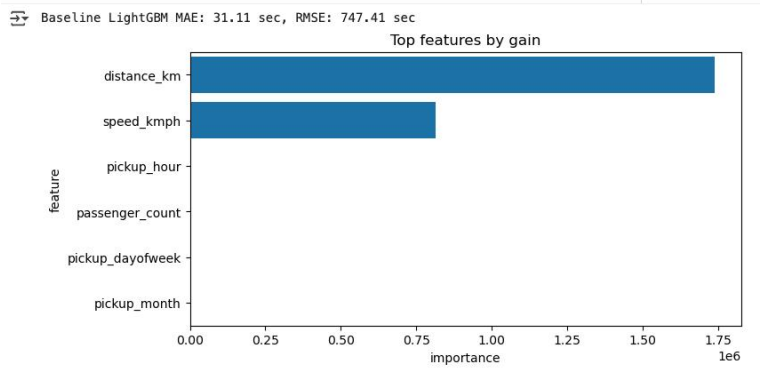
- **Linear Regression**: Produced extremely high scores (R^2 = 1.0 with very low errors). This unrealistic result was traced to **data leakage**, mainly due to the inclusion of features such as dropoff_datetime that already contained future information.



- **Python Results**

After correcting the dataset and **removing leakage features** (e.g., dropoff_datetime), we obtained results that aligned better with expectations.

- **Linear Regression**: Still showed signs of overfitting under some settings.
- **Random Forest**: Produced more stable, though lower, R^2 values compared with Orange.
- **LightGBM (Baseline)**: Outperformed both by achieving **MAE = 31.11 sec** and **RMSE = 747.41 sec**. The feature importance plot confirmed that engineered features such as distance_km and speed_kmph were the most influential predictors.



- **Discussion**

These results demonstrate that the apparent superiority of Linear Regression in Orange was an artifact of data leakage. Python validation confirmed Random Forest to be more realistic once the leakage was removed, and LightGBM highlighted the value of feature engineering in improving both accuracy and interpretability. Overall, this comparison shows that model performance depends not only on algorithm choice but also on the quality of the data pipeline and feature design.

6. Reflections

In this lab, we learned the importance of carefully combining visual workflows in Orange with flexible scripting in Python. At first, when we ran the models, we obtained unrealistic results — Linear Regression gave perfect scores ($R^2=1$), while Random Forest sometimes produced negative R^2 values, which indicated that the models were not actually learning useful patterns.

Through group discussion and iteration, we realized that the issue came from data leakage and the presence of features such as dropoff_datetime and dropoff coordinates, which are not available at prediction time. After removing these variables and re-running the experiments, the results became more reasonable, and we were able to compare models fairly.

This process taught us that machine learning is not only about applying algorithms but also about critical thinking in feature selection, evaluation, and interpretation. We also experienced the value of teamwork — by checking each other's steps and sharing ideas, we identified the problem collaboratively and corrected it successfully.

Overall, this comparison shows that model performance depends not only on the choice of algorithm but also, equally importantly, on the quality of the data pipeline and the design of features.

7. File Submission

Submission includes:

- Lab1_Report_Group6.pdf (This report)
- Lab1_Code_Group6.ipynb (Python notebook with code)
- Lab1_Orange_Group6.ows (Orange project file)