Data Intake Report Cloud and API deployment

Project Name: Cloud and API deployment

Report Date: 06/04/2024

Internship Batch: LISUM33

Version: 1.0

Data Intake by: Marina Tsvetkova

Data Intake Reviewer: Data Glacier

Data Storage Location:

https://github.com/Marinatsv07/Data_Glacier_Internship/tree/main/Week_5

Model Setup

df = pd.read_parquet('https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_2024-01.parquet')

In [6]:

```
# Calculate 'duration' as the difference between drop-off and pick-up times
df['duration'] = (df['tpep_dropoff_datetime'] - df['tpep_pickup_datetime'])

# Convert duration from timedelta to minutes
df['duration_min'] = df['duration'].dt.total_seconds() / 60

# Calculate the standard deviation of trip durations in January (in minutes)
std_duration_january = df['duration_min'].std()
print("Standard deviation of trips duration in January (in minutes):", std_duration_january)

# Filtering df to remove outliers, keeping only durations between 1 and 60 minutes
original_count = len(df)
df_filtered = df[(df['duration_min'] >= 1) & (df['duration_min'] <= 60)].copy()
filtered_count = len(df_filtered)

# Determine what fraction of records remains after dropping outliers
fraction_left = filtered_count / original_count
print("Fraction of records left after dropping outliers:", fraction_left)
```

```
# Convert location IDs to strings
df_filtered['PULocationID'] = df_filtered['PULocationID'].astype(str)
df_filtered['DOLocationID'] = df_filtered['DOLocationID'].astype(str)

# Use only the filtered duration in minutes for further processing
df_filtered['duration'] = df_filtered['duration_min']

# Convert DataFrame to a list of dictionaries for feature encoding
data_dicts = df_filtered[['PULocationID', 'DOLocationID']].to_dict(orient='records')

# Create and apply DictVectorizer
vectorizer = DictVectorizer(sparse=True)
feature_matrix = vectorizer.fit_transform(data_dicts)

# Number of columns in the feature matrix
print("Dimensionality of the feature matrix (number of columns):", feature_matrix.shape[1])

# Target variable
y = df_filtered['duration']

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(feature_matrix, y, test_size=0.2, random_state=42)

# Create and train a linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the training data
y_train_pred = model.predict(X_train)

# Calculate RMSE on the training data
rmse_train = np.sqrt(mean_squared_error(y_train, y_train_pred))
print("RMSE on the train data:", rmse_train)
```

```
Standard deviation of trips duration in January (in minutes): 34.851053592192876
Fraction of records left after dropping outliers: 0.9778326020432945
Dimensionality of the feature matrix (number of columns): 518
RMSE on the train data: 7.948999308349224
```

In [7]:

```
# Save the trained model using pickle
model_filename = 'linear_regression_model.pkl'
with open(model_filename, 'wb') as file:
    pickle.dump(model, file)

print(f"Model saved as {model_filename}")
```

```
Model saved as linear_regression_model.pkl
```

In [8]:

```
#save the vectorizer too
vectorizer_filename = 'vectorizer.pkl'
with open(vectorizer_filename, 'wb') as file:
    pickle.dump(vectorizer, file)

print(f"Vectorizer saved as {vectorizer_filename}")
```

```
Vectorizer saved as vectorizer.pkl
```

In [9]:

```
# Load the saved model
with open(model_filename, 'rb') as file:
    loaded_model = pickle.load(file)

# Load the saved vectorizer
with open(vectorizer_filename, 'rb') as file:
    loaded_vectorizer = pickle.load(file)
```

In [6]:

```
def preprocess_data(data_path: str) -> pd.DataFrame:
    """
    Preprocess the data from the given Parquet file.

    Parameters:
    data_path (str): Path to the input data file (Parquet format).

    Returns:
    pd.DataFrame: Preprocessed DataFrame.
    """
    print("Loading data...")
    df = pd.read_parquet(data_path)
    print("Data loaded. Calculating duration...")

    df['duration'] = (df['tpep_dropoff_datetime'] - df['tpep_pickup_datetime'])
    df['duration_min'] = df['duration'].dt.total_seconds() / 60

    print("Filtering outliers...")
    df_filtered = df[(df['duration_min'] >= 1) & (df['duration_min'] <= 60)].copy()

    print("Converting location IDs to strings...")
    df_filtered['PULocationID'] = df_filtered['PULocationID'].astype(str)
    df_filtered['DOLocationID'] = df_filtered['DOLocationID'].astype(str)

    print("Data preprocessing complete.")
    return df_filtered
```

```
data_path = 'https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_2024-02.parquet'
df_filtered = preprocess_data(data_path)
print(df_filtered.head())
```

```
Loading data...
Data loaded. Calculating duration...
Filtering outliers...
Converting location IDs to strings...
Data preprocessing complete.
   VendorID tpep_pickup_datetime tpep_dropoff_datetime  passenger_count  \
0         2  2024-02-01 00:04:45   2024-02-01 00:19:58              1.0
1         2  2024-02-01 00:56:31   2024-02-01 01:10:53              1.0
2         2  2024-02-01 00:07:50   2024-02-01 00:43:12              2.0
3         1  2024-02-01 00:01:49   2024-02-01 00:10:47              1.0
4         1  2024-02-01 00:37:35   2024-02-01 00:51:15              1.0

   trip_distance  RatecodeID store_and_fwd_flag PULocationID DOLocationID  \
0           4.39         1.0                  N          68          236
1           7.71         1.0                  N          48          243
2          28.69         2.0                  N         132          261
3           1.10         1.0                  N         161          163
4           2.60         1.0                  N         246           79

   payment_type  ...  extra  mta_tax  tip_amount  tolls_amount  \
0             1  ...    1.0      0.5        1.28          0.00
1             1  ...    1.0      0.5        9.00          0.00
2             2  ...    0.0      0.5        0.00          6.94
3             1  ...    3.5      0.5        2.85          0.00
4             2  ...    3.5      0.5        0.00          0.00

   improvement_surcharge  total_amount  congestion_surcharge  Airport_fee  \
0                    1.0         26.78                   2.5         0.00
1                    1.0         45.00                   2.5         0.00
2                    1.0         82.69                   2.5         1.75
3                    1.0         17.15                   2.5         0.00
4                    1.0         20.60                   2.5         0.00

        duration duration_min
0 0 days 00:15:13    15.216667
1 0 days 00:14:22    14.366667
2 0 days 00:35:22    35.366667
3 0 days 00:08:58     8.966667
4 0 days 00:13:40    13.666667

[5 rows x 21 columns]
```

In [7]:

```
def load_vectorizer(vectorizer_path: str) -> DictVectorizer:
    """
```

```
    Load the DictVectorizer from the given file.

    Parameters:
    vectorizer_path (str): Path to the saved DictVectorizer file (Pickle format).

    Returns:
    DictVectorizer: Loaded DictVectorizer.
    """
    print("Loading vectorizer...")
    with open(vectorizer_path, 'rb') as file:
        vectorizer = pickle.load(file)
    print("Vectorizer loaded.")
    return vectorizer


vectorizer_path = 'vectorizer.pkl'
vectorizer = load_vectorizer(vectorizer_path)

data_dicts = df_filtered[['PULocationID', 'DOLocationID']].to_dict(orient='records')
print("Transforming data...")
X = vectorizer.transform(data_dicts)
print("Data transformed. Shape:", X.shape)
```

```
Loading vectorizer...
Vectorizer loaded.
Transforming data...
Data transformed. Shape: (2938060, 518)
```

In [8]:

```
def load_model(model_path: str) -> LinearRegression:
    """
    Load the model from the given file.

    Parameters:
    model_path (str): Path to the saved model file (Pickle format).

    Returns:
    LinearRegression: Loaded model.
    """
    print("Loading model...")
    with open(model_path, 'rb') as file:
        model = pickle.load(file)
    print("Model loaded.")
    return model

# Example usage
model_path = 'linear_regression_model.pkl'
model = load_model(model_path)
```

```
y = df_filtered['duration_min']
print("Making predictions...")
y_pred = model.predict(X)
print("Predictions made. First 10 predictions:", y_pred[:10])
```

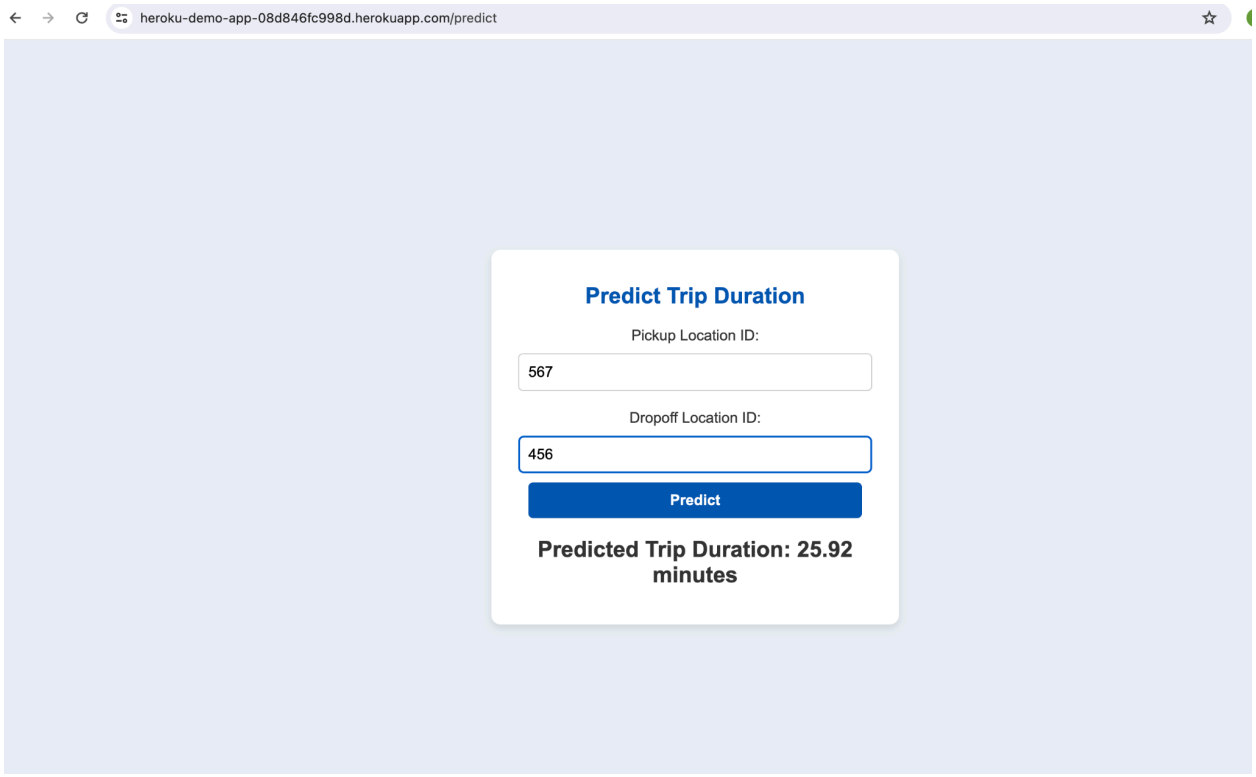
```
Loading model...
Model loaded.
Making predictions...
Predictions made. First 10 predictions: [13.24544287 20.48169216 37.42164098 13.46284077
13.12928572 12.15326858
 12.69852248 13.03233581 13.81530867 10.55141448]
```

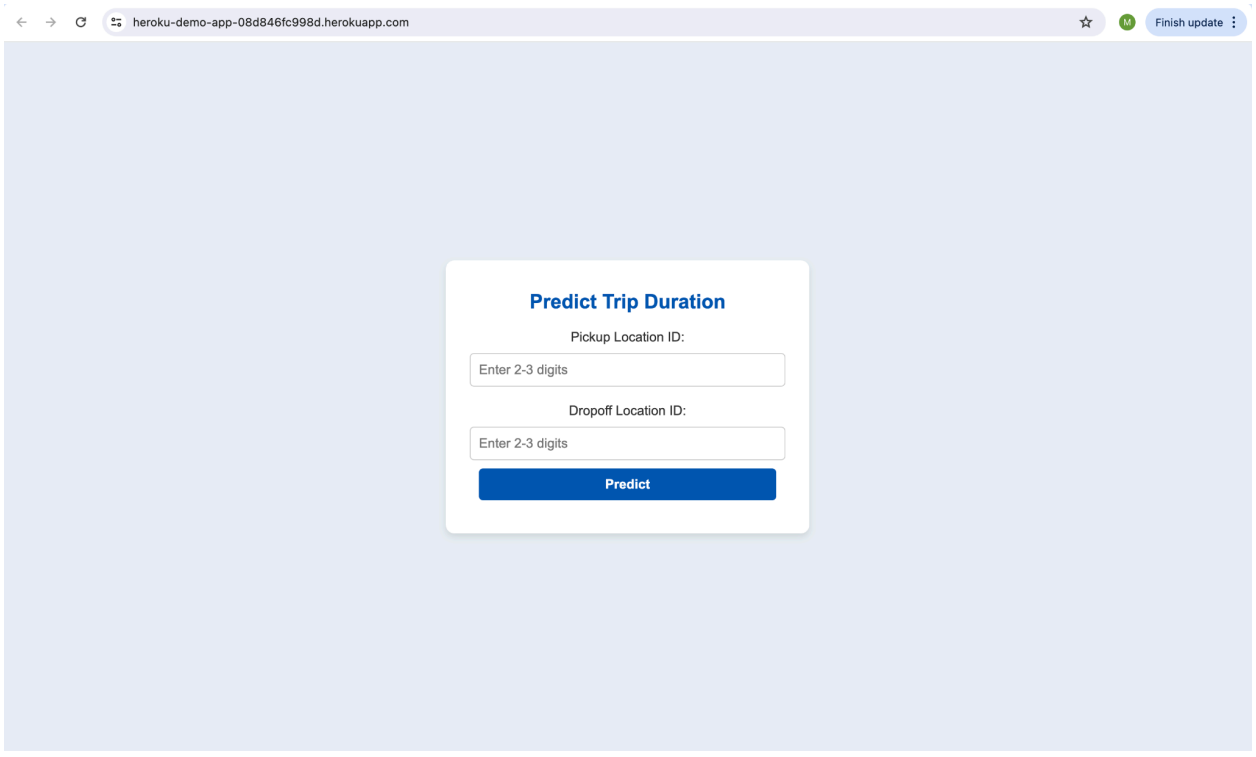
In [9]:

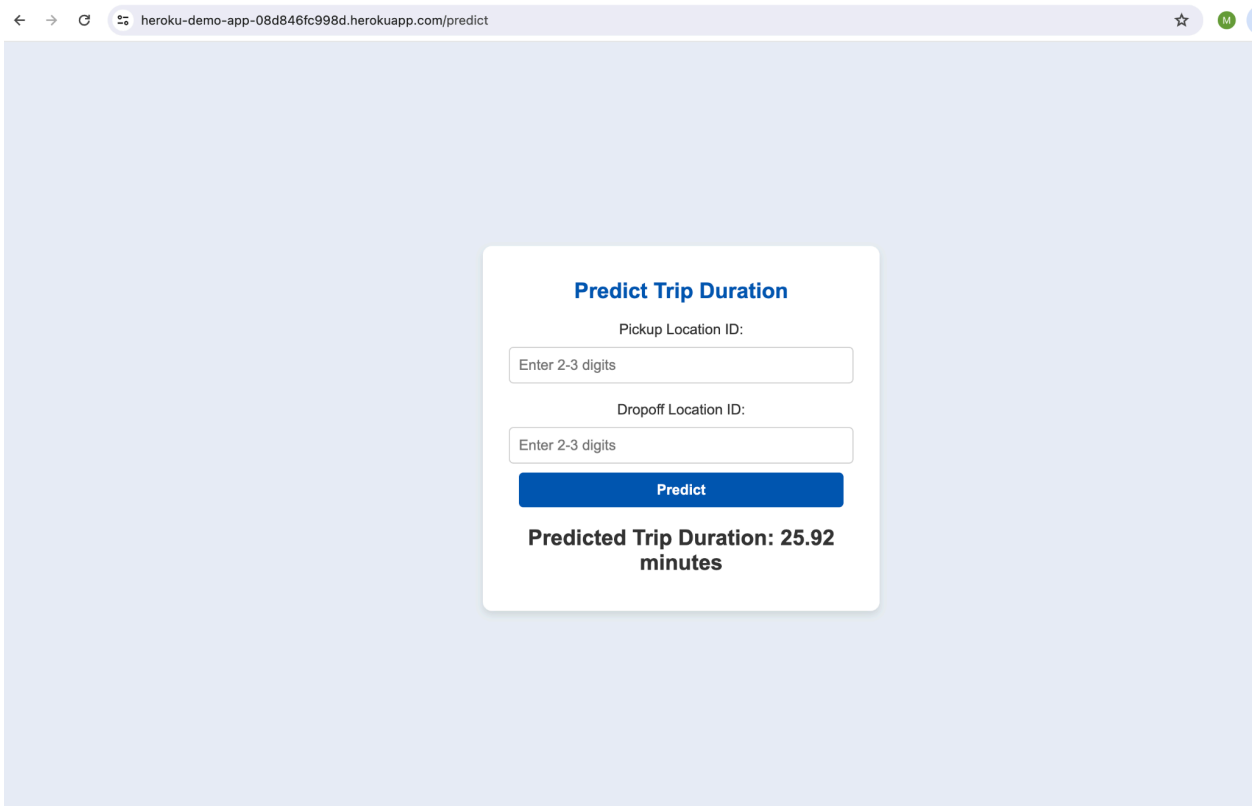
```
def calculate_rmse(y_true: pd.Series, y_pred: np.ndarray) -> float:
    """
    Calculate the Root Mean Squared Error (RMSE).

    Parameters:
    y_true (pd.Series): True values.
    y_pred (np.ndarray): Predicted values.

    Returns:
    float: RMSE value.
    """
    print("Calculating RMSE...")
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    print("RMSE calculated:", rmse)
    return rmse

# Example usage
rmse = calculate_rmse(y, y_pred)
```

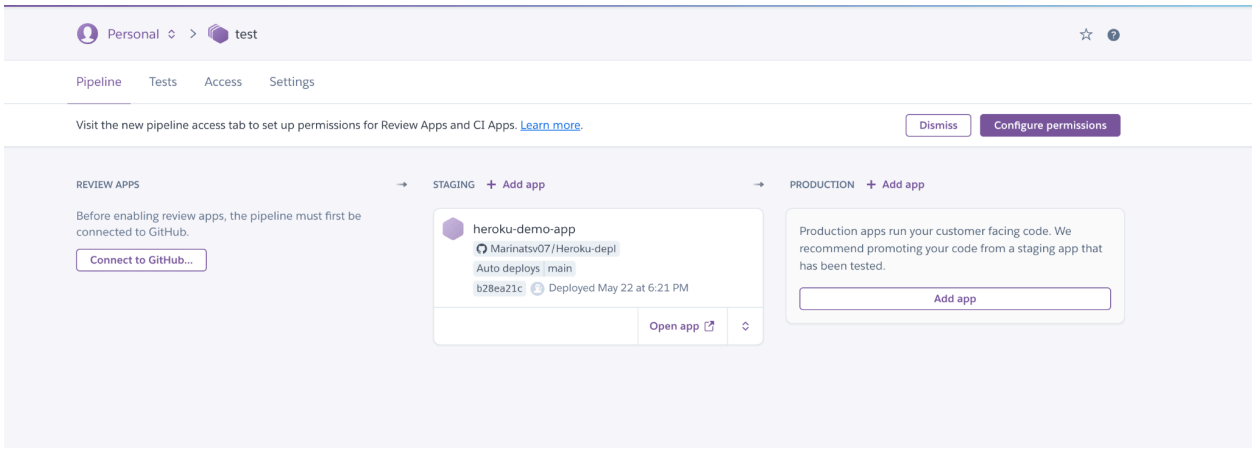
```
Calculating RMSE...
RMSE calculated: 8.124017135620836
```

Heroku Deployment


heroku-demo-app-08d846fc998d.herokuapp.com/predict
Predict Trip Duration
Pickup Location ID:
567
Dropoff Location ID:
456
Predict
Predicted Trip Duration: 25.92 minutes


heroku-demo-app-08d846fc998d.herokuapp.com
Finish update
Predict Trip Duration
Pickup Location ID:
Enter 2-3 digits
Dropoff Location ID:
Enter 2-3 digits
Predict


heroku-demo-app-08d846fc998d.herokuapp.com/predict
Predict Trip Duration
Pickup Location ID:
Enter 2-3 digits
Dropoff Location ID:
Enter 2-3 digits
Predict
Predicted Trip Duration: 25.92 minutes


Personal
test
Pipeline
Tests
Access
Settings
Visit the new pipeline access tab to set up permissions for Review Apps and CI Apps. Learn more.
Dismiss
Configure permissions
REVIEW APPS
Before enabling review apps, the pipeline must first be connected to GitHub.
Connect to GitHub...
STAGING
Add app
heroku-demo-app
Marinatsv07/Heroku-depl
Auto deploys
main
b28ea21c
Deployed May 22 at 6:21 PM
Open app
PRODUCTION
Add app
Production apps run your customer facing code. We recommend promoting your code from a staging app that has been tested.
Add app