Report for ridesharing application

DataFrame:

vendor_id: Identifier for the transportation service provider.

pickup_datetime: Date and time when the trip started.

dropoff_datetime: Date and time when the trip ended.

trip_duration: Duration of the trip in seconds.

dist_meters: Distance of the trip in meters.

wait_sec: Waiting time in seconds.

dist_long_lat: Distance between longitude and latitude coordinates.

driver_avg_speed: Average speed of the driver during the trip.

slope: Slope of the terrain during the trip.

trip_duration_hours: Duration of the trip in hours.

dist_kilometers: Distance of the trip in kilometers.

vendor_id	pickup_datetime	dropoff_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude
Quito	2016-09-20 05:57:48	2016-09-20 05:59:55	0.545969	0.168200	-78.484300	-0.171597
Default Fare	2016-09-18 12:03:19	2016-09-18 12:03:52	0.437893	0.539475	-78.442144	-0.244787
Quito	2016-09-18 08:55:16	2016-09-18 08:55:52	0.682279	0.009253	-78.527340	-0.090939
Quito	2016-09-19 07:16:39	2016-09-19 07:17:47	0.498026	0.140398	-78.475512	-0.162803
Panama Uber	2016-09-19 04:14:21	2016-09-21 12:39:50	0.750012	0.002848	-79.525941	8.983723
...	...	...	...	...	...	...
Quito	2017-04-21 09:35:05	2017-04-21 10:31:34	0.443849	0.791983	-78.515515	-0.237873
Monterrey	2016-09-19 11:22:06	2016-09-19 11:59:09	0.168254	0.601672	-100.312213	25.777738
Quito	2016-09-18 11:06:58	2016-09-18 11:53:59	0.685477	0.543913	-78.361078	-0.124627
Quito	2016-09-19 12:43:55	2016-09-19 02:46:04	0.431389	0.098833	-78.417412	-0.136554
Monterrey UberX	2016-09-13 09:23:52	2016-09-14 07:41:36	0.137811	0.610079	-99.891755	26.742812

trip_duration	dist_meters	wait_sec	dist_long_lat	driver_avg_speed	slope	trip_duration_hours	dist_kilometers
127	21	107	16941.545561	0.595276	0.004300	0.035278	0.021
33	37	14	17284.263136	4.036364	0.009942	0.009167	0.037
36	51	24	15727.807319	5.100000	0.001265	0.010000	0.051
69	70	94	17285.793585	3.652174	0.003839	0.019167	0.070
159929	75	1370	10954.925266	0.001688	-0.111875	44.424722	0.075
...	...	...	...	...	...	...	...
3389	36422	516	15827.389254	38.689643	0.013043	0.941389	36.422
2223	39386	3712	380.109085	63.782996	-0.250557	0.617500	39.386
2821	46418	302	15991.837039	59.236016	0.008458	0.783611	46.418
7330	59823	2513	18051.341048	29.381010	0.002985	2.036111	59.823
37065	135871	3738	6342.897727	13.196698	-0.261250	10.295833	135.871

Functions:

1. **Random Shuffle**

The **random_shuffle** function is designed to randomly shuffle the rows of an input DataFrame.

2. **Quicksort Algorithm for DataFrame Sorting**

The **quicksort** algorithm presented here is a randomized divide and conquer approach to sorting a DataFrame based on the 'dist_meters' column and selecting a random pivot element from the 'dist_meters' column.

3. **Linear Regression and SVM time prediction**

This function, using the standardized features compares the performance of two regression models, **Linear Regression** and **Support Vector Machine** (SVM), in predicting the expected duration and arrival time of transportation trips. The analysis includes evaluating Mean Squared Error (MSE) and Deterministic Coefficient (R-squared) for both models.

From the output of the function, we can acknowledge that linear regression in this case gives much better output than SVM with lesser MSE, positive R-squared and more accurate time prediction.

```
Minimum Trip Duration: 0.25 minutes
Mean Trip Duration: 8.778499819037277 hours
Maximum Trip Duration: 979.8622222222223 hours

 ------- regression_results -------
Mean Squared Error: 73.21149802515569
Deterministic Coefficient (R-squared): 0.997606486406999
Expected trip duration in minutes:  817.9754507382711
Expected date and time of arrival: 2016-09-17 18:23:54.527044352-05:00

 -------- SVM result ----------
Mean Squared Error: 33905.721218315346
Deterministic Coefficient (R-squared): -0.10848441577652501
Expected trip duration in minutes:  4110.407044311364
Expected date and time of arrival: 2016-09-20 01:16:20.422658560-05:00
```

4. Principal Component Analysis (PCA) Calculation

The provided code includes two functions: **calculate_eigenvalues** and **PCA**. These functions are dedicated to performing Principal Component Analysis (PCA) on a given DataFrame, extracting eigenvalues and eigenvectors to understand the variance in the data.

“Wait_sec” - captures the most variance among the features.

```
Explained Variance by eigenvalues:
Eigenvalue for dist_meters: 141.33
Eigenvalue for dist_long_lat: 54.46
Eigenvalue for driver_avg_speed: 2.82
Eigenvalue for trip_duration: 7.58
Eigenvalue for wait_sec: 1759.12
```

5. Vendor Selection Algorithm

The provided code includes three functions: **calculate_travel_time**, **find_closest_floats**, and **calculate_best_vendor_greedy**. Together, these functions aim to find the best vendor for a customer at a given location using a greedy approach. The algorithm considers the distance, average speed of drivers, and historical data to estimate the pickup time.

As output, we get vendor id with shortest waiting time.

```
For the customer at position: 0.375639519207419, 0.752236590542026
The best vendor is Cuenca with a pickup time of 35.89 minutes.
```

6. Fuel Cost Calculation and Optimization

The provided code comprises three functions: **cost_function**, **calculate_fuel_coeff**, and **dynamic_calculate_min_fuel_cost**. These functions are designed to calculate the cost function for a trip, dynamically determine the fuel cost coefficient based on terrain slope.

As the output we get optimized total fuel cost for every unique vendor

	vendor_id	minimum_total_fuel_cost
0	Quito	415.927394
1	Guadalajara Easy Taxi	1114.144124
2	Monterrey	10316.120150
3	Manta	22.806651
4	Aguascalientes	1071.615968
5	Guadalajara	473.664690
6	Cali	28.347370
7	Ecuador	208.461837
8	Bogota	309.864481
9	Medellin	68.067152
10	Mexico DF Taxi de Sitio	1014.332456
11	Mexico DF Taxi Libre	705.861902
12	Default Fare	139.408184
13	Panama UberX	233.620639
14	Panama UberSUV	43.184989
15	Puebla Easy Taxi	32.737404
16	Mexico DF Radio Taxi	515.657918
17	Cuenca	12275.151877
18	Ruminahui	28.325116

7. **Cost Minimization using Gradient Descent**

The provided code consists of two functions: **gradient_descent** and **minimize_cost**. These functions are designed to perform gradient descent to minimize the cost function with respect to driver speed, and subsequently, to minimize the total cost for each trip in a DataFrame using gradient descent.

As the output we add to DataFrame highest optimal speed to total cost of journey.

optimal_speed	total_cost
43.043396	49.248266
27.371245	44.429541
20.361943	13.543835
23.854490	184.395037
26.713155	37.798914
...	...
9.949538	0.183978
34.828713	223.869949
34.020444	2.419680
48.342529	0.633881
14.956387	79.965338

8. **Calculate Travel Route Greedy**

The **calculate_travel_route_greedy** function employs a greedy algorithm to find the most efficient travel route, starting from a specified location and reaching the destination.

The **find_nearest_point** is responsible for identifying the nearest point considering both distance and a heuristic score. In summary, these functions work in tandem to create an efficient and optimized travel route, considering both distance and a heuristic score.

As the output we get travel route from starting point blue to end point red.