New York Taxi Fare Prediction Data Analysis & Modelling

Name: Murigi Gracious Ng'ang'a

The end product will be a mock app that helps to predict the amount of fare you are likely to incur when moving from one place to another in New York

In [1]:

```
# Importing the necessary libraries

# 1. Data Manipulation & numerical calculations
import pandas as pd
import numpy as np
import math

# 2. Data preprocessing and modelling
from sklearn.model_selection import train_test_split, KFold, cross_val_score

# 3. Machine Learning Linear Regression model
from sklearn.linear_model import LinearRegression

# 4. Evaluation metrics
from sklearn.metrics import mean_squared_error

# 5. Data Visualization
import matplotlib.pyplot as plt
import seaborn as sns

# 6. Saving our model
import joblib
```

In [2]:

```
# Load the data - Reading the first 1M rows
fare_df= pd.read_csv('../Datasets/train.csv', nrows=1_000_000)

# View the first few rows
fare_df.head(3)
```

Out[2]:

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_long
0	2009-06-15 17:26:21.0000001	4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	-73.84
1	2010-01-05 16:52:16.0000002	16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	-73.97
2	2011-08-18 00:35:00.00000049	5.7	2011-08-18 00:35:00 UTC	-73.982738	40.761270	-73.9

In [3]:
```
fare_df.tail(3)
```

Out[3]:

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropo
999997	2013-04-26 14:03:00.00000118	10.5	2013-04-26 14:03:00 UTC	-73.978423	40.751135	
999998	2011-07-08 00:29:00.00000099	6.9	2011-07-08 00:29:00 UTC	-73.980317	40.759482	
999999	2009-12-31 14:30:00.00000021	4.1	2009-12-31 14:30:00 UTC	-74.006635	40.741598	

In [4]:
```
# View the shape of data
fare_df.shape
```

Out[4]:
```
(1000000, 8)
```

In [5]:
```
# Concise summary of the data
fare_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Data columns (total 8 columns):
 #   Column             Non-Null Count    Dtype  
---  ------             --------------    -----  
 0   key                1000000 non-null  object 
 1   fare_amount        1000000 non-null  float64
 2   pickup_datetime    1000000 non-null  object 
 3   pickup_longitude   1000000 non-null  float64
 4   pickup_latitude    1000000 non-null  float64
 5   dropoff_longitude  999990 non-null   float64
 6   dropoff_latitude   999990 non-null   float64
 7   passenger_count    1000000 non-null  int64  
dtypes: float64(5), int64(1), object(2)
memory usage: 61.0+ MB
```

In [6]:
```
# Number of null values in each column
fare_df.isna().sum()
```

Out[6]:
```
key                   0
fare_amount           0
pickup_datetime       0
pickup_longitude      0
pickup_latitude       0
dropoff_longitude    10
dropoff_latitude     10
passenger_count       0
dtype: int64
```

In [7]:
```
# Drop the missing values since our dataset is fairly large and very few rows are
print(f"{len(fare_df) - len(fare_df.dropna())} rows with missing values have been
fare_df.dropna(inplace=True)
```

```
10 rows with missing values have been dropped!
```

In [8]:
```python
# Dropping the key column for identification of duplicate rows
# The 'key' column  has been used as a unique id for each row
fare_df.drop(inplace=True, columns=['key'])
fare_df.head()
```

Out[8]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitu
0	4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	-73.841610	40.712
1	16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	-73.979268	40.782
2	5.7	2011-08-18 00:35:00 UTC	-73.982738	40.761270	-73.991242	40.750
3	7.7	2012-04-21 04:30:42 UTC	-73.987130	40.733143	-73.991567	40.758
4	5.3	2010-03-09 07:51:00 UTC	-73.968095	40.768008	-73.956655	40.783

In [9]:
```python
# Dropping all the duplicate rows
print(f"{len(fare_df) - len(fare_df.drop_duplicates())} duplicate rows found and
fare_df.drop_duplicates(inplace=True)
```

```
0 duplicate rows found and dropped!
```

In [10]:
```python
# View data types of the columns
fare_df.dtypes
```

Out[10]:
```
fare_amount          float64
pickup_datetime       object
pickup_longitude     float64
pickup_latitude      float64
dropoff_longitude    float64
dropoff_latitude     float64
passenger_count        int64
dtype: object
```

In [11]:
```
# statistical data for numerical columns
fare_df.describe()
```

Out[11]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passen
count	999990.000000	999990.000000	999990.000000	999990.000000	999990.000000	9999
mean	11.347953	-72.526699	39.929040	-72.527860	39.919954	
std	9.821790	12.057778	7.626087	11.324494	8.201418	
min	-44.900000	-3377.680935	-3116.285383	-3383.296608	-3114.338567	
25%	6.000000	-73.992060	40.734965	-73.991385	40.734046	
50%	8.500000	-73.981792	40.752695	-73.980135	40.753166	
75%	12.500000	-73.967094	40.767154	-73.963654	40.768129	
max	500.000000	2522.271325	2621.628430	45.581619	1651.553433	2

Data Cleaning Process. The following rows are to be dropped:

1. Zero-passenger trips

2. Where the fare is 0 or -ve

3. Outliers (Absurd values)

4. Invalid coordinates --> Logitude(180,-180), Logitude(90,-90),

5. Not within the New York coordinates --> Longitude = (-71.4725,-79.4554), Latitude = (40.2940,45.0042)

In [12]:
```
fare_df.shape
```

Out[12]: (999990, 7)

In [13]:
```
# Dropping rows where the fare is less than or equal to zero
fare_df = fare_df[fare_df['fare_amount'] > 0]
fare_df.shape
```

Out[13]: (999923, 7)

In [14]:
```
# Dropping rows with zero-passenger trips
fare_df = fare_df[fare_df['passenger_count'] > 0]
fare_df.shape
```

Out[14]: (996368, 7)

In [15]:
```python
# Sorting the passenger count column values
sorted_counts = np.sort(fare_df['passenger_count'])
sorted_counts[-10:]
```

Out[15]:
```
array([  6,   6,   6,   6,   6,   6,   6,   6,   6, 208], dtype=int64)
```

In [16]:
```python
# Check if there are outliers -- greater than 6
fare_df = fare_df[fare_df['passenger_count'] <= 6]

fare_df.shape
```

Out[16]:
```
(996367, 7)
```

In [17]:
```python
# Dropping rows with invalid coordinates (Steps 4 & 5)
fare_df = fare_df[fare_df['pickup_longitude'] <= -71.4725]
fare_df = fare_df[fare_df['pickup_longitude'] >= -79.4554]

fare_df = fare_df[fare_df['pickup_latitude'] <= 45.0042]
fare_df = fare_df[fare_df['pickup_latitude'] >= 40.2940]

fare_df = fare_df[fare_df['dropoff_longitude'] <= -71.4725]
fare_df = fare_df[fare_df['dropoff_longitude'] >= -79.4554]

fare_df = fare_df[fare_df['dropoff_latitude'] <= 45.0042]
fare_df = fare_df[fare_df['dropoff_latitude'] >= 40.2940]

fare_df.shape
```

Out[17]:
```
(975710, 7)
```

In [18]:
```python
# Statistical data
fare_df.describe()
```

Out[18]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passen
count	975710.000000	975710.000000	975710.000000	975710.000000	975710.000000	9757
mean	11.342525	-73.975205	40.751102	-73.974291	40.751445	
std	9.756582	0.041462	0.030181	0.040029	0.033605	
min	0.010000	-78.669423	40.300233	-78.669423	40.300223	
25%	6.000000	-73.992272	40.736573	-73.991573	40.735582	
50%	8.500000	-73.982083	40.753410	-73.980591	40.753872	
75%	12.500000	-73.968312	40.767581	-73.965316	40.768419	
max	500.000000	-72.702870	43.213962	-72.196091	44.600000	

In [19]:
```
# changing 'pickup_datetime' to datetime data type
fare_df['pickup_datetime'] = pd.to_datetime(fare_df['pickup_datetime'], format =
fare_df.dtypes
```

Out[19]:
```
fare_amount                      float64
pickup_datetime      datetime64[ns, UTC]
pickup_longitude                 float64
pickup_latitude                  float64
dropoff_longitude                float64
dropoff_latitude                 float64
passenger_count                    int64
dtype: object
```

In [20]:
```
# Creating new columns from the datetime object
fare_df['year'] = fare_df['pickup_datetime'].dt.strftime('%Y')
fare_df['month'] = fare_df['pickup_datetime'].dt.strftime('%m')
fare_df['day'] = fare_df['pickup_datetime'].dt.strftime('%d')
fare_df['hour'] = fare_df['pickup_datetime'].dt.strftime('%H')
fare_df['minute'] = fare_df['pickup_datetime'].dt.strftime('%M')
fare_df
```

Out[20]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff
0	4.5	2009-06-15 17:26:21+00:00	-73.844311	40.721319	-73.841610	4
1	16.9	2010-01-05 16:52:16+00:00	-74.016048	40.711303	-73.979268	4
2	5.7	2011-08-18 00:35:00+00:00	-73.982738	40.761270	-73.991242	4
3	7.7	2012-04-21 04:30:42+00:00	-73.987130	40.733143	-73.991567	4
4	5.3	2010-03-09 07:51:00+00:00	-73.968095	40.768008	-73.956655	4
...	...	...	...	...	...	
999995	7.0	2014-09-13 21:44:38+00:00	-73.976676	40.785630	-73.959196	4
999996	7.3	2010-09-20 14:50:37+00:00	-73.992103	40.671385	-73.992103	4
999997	10.5	2013-04-26 14:03:00+00:00	-73.978423	40.751135	-73.993397	4
999998	6.9	2011-07-08 00:29:00+00:00	-73.980317	40.759482	-73.976832	4
999999	4.1	2009-12-31 14:30:00+00:00	-74.006635	40.741598	-73.999450	4

975710 rows × 12 columns

In [21]:
```
# Dropping the pickup_datetime column
fare_df.drop(columns=['pickup_datetime'], inplace=True)
fare_df.head(3)
```

Out[21]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_cc
0	4.5	-73.844311	40.721319	-73.841610	40.712278	
1	16.9	-74.016048	40.711303	-73.979268	40.782004	
2	5.7	-73.982738	40.761270	-73.991242	40.750562	

In [22]:
```
# Making the 'time' columns numeric
fare_df[['year', 'month','day', 'hour', 'minute']] = fare_df[['year', 'month','da
fare_df.dtypes
```

Out[22]:
```
fare_amount          float64
pickup_longitude     float64
pickup_latitude      float64
dropoff_longitude    float64
dropoff_latitude     float64
passenger_count        int64
year                   int64
month                  int64
day                    int64
hour                   int64
minute                 int64
dtype: object
```

The formula used to calculate the distance between longitudes and latitudes:

[Haversine Formula (https://www.sisense.com/blog/latitude-longitude-distance-calculation-explained/)](https://www.sisense.com/blog/latitude-longitude-distance-calculation-explained/)

$a = sin^2(\Delta\varphi/2) + cos\varphi1 \cdot cos\varphi2 \cdot sin^2(\Delta\lambda/2)$

$c = 2 * atan2(\sqrt{a}, \sqrt{(1-a)})$

$d = R * c$

Where φ represents the latitudes, λ the longitudes and R the radius of the earth

In [23]:
```python
# Function to calculate the distance in kilometers between the latitudes and Long

def haversine(lon1, lat1, lon2, lat2):
    """
    Calculates the distance between two locations given the latitudes and longitu

    lat1: The start latitude
    lat2: The end latitude
    lon1: The start longitude
    lon2: The end longitude
    """
    earth_radius = 6371

    #Convert the angles from degrees to radians
    lon1, lat1, lon2, lat2 = map(math.radians, [lon1, lat1, lon2, lat2])

    lat_diff = lat2 - lat1
    lon_diff = lon2 - lon1

    a = pow(math.sin(lat_diff/2), 2) + math.cos(lat1) * math.cos(lat1) * pow(math
    # c = 2 * math.asin(math.sqrt(a))
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))
    distance = earth_radius * c

    return distance
```

In [24]:
```python
# Adding the distance column to our dataframe
distances = []

for i in range(fare_df.shape[0]):                    # or range(len(fare_df))
    pick_lon = fare_df.iloc[i]['pickup_longitude']
    pick_lat = fare_df.iloc[i]['pickup_latitude']
    drop_lon = fare_df.iloc[i]['dropoff_longitude']
    drop_lat = fare_df.iloc[i]['dropoff_latitude']
    distance = haversine(pick_lon, pick_lat, drop_lon, drop_lat)
    distances.append(distance)

fare_df['distance in kilometres'] = distances
fare_df.head(3)
```

Out[24]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_cc
0	4.5	-73.844311	40.721319	-73.841610	40.712278	
1	16.9	-74.016048	40.711303	-73.979268	40.782004	
2	5.7	-73.982738	40.761270	-73.991242	40.750562	

In [25]:
```python
# Adding the longitude_difference and latitude_difference columns to our data fro
fare_df['longitude_difference'] = (fare_df['dropoff_longitude'] - fare_df['pickup
fare_df['latitude_difference'] = (fare_df['dropoff_latitude'] - fare_df['pickup_l
```

In [26]:
```
fare_df.describe()
```

Out[26]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passen
count	975710.000000	975710.000000	975710.000000	975710.000000	975710.000000	9757
mean	11.342525	-73.975205	40.751102	-73.974291	40.751445	
std	9.756582	0.041462	0.030181	0.040029	0.033605	
min	0.010000	-78.669423	40.300233	-78.669423	40.300223	
25%	6.000000	-73.992272	40.736573	-73.991573	40.735582	
50%	8.500000	-73.982083	40.753410	-73.980591	40.753872	
75%	12.500000	-73.968312	40.767581	-73.965316	40.768419	
max	500.000000	-72.702870	43.213962	-72.196091	44.600000	

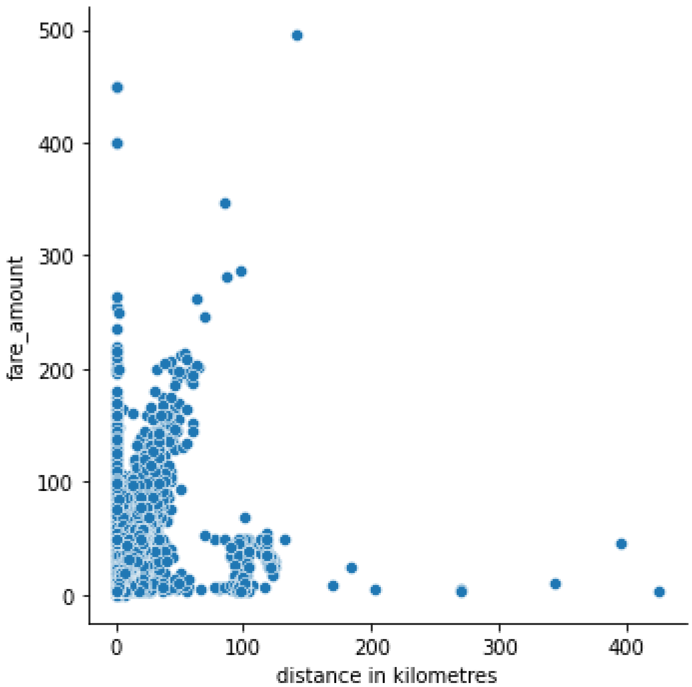
In [27]:
```
# Drop rows where the distance is zero
fare_df = fare_df[fare_df['distance in kilometres'] > 0]
fare_df.describe()
```

Out[27]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passen
count	965221.000000	965221.000000	965221.000000	965221.000000	965221.000000	9652
mean	11.345486	-73.975520	40.750993	-73.974596	40.751340	
std	9.678684	0.038662	0.028379	0.037112	0.032034	
min	0.010000	-78.650908	40.300233	-78.659447	40.300223	
25%	6.000000	-73.992284	40.736591	-73.991579	40.735597	
50%	8.500000	-73.982107	40.753424	-73.980624	40.753890	
75%	12.500000	-73.968457	40.767583	-73.965480	40.768424	
max	495.000000	-72.702870	43.183332	-72.196091	44.600000	

In [28]:
```
#Plotting distances in kilometers vs fare amount
sns.relplot(data = fare_df, x = 'distance in kilometres', y = 'fare_amount')
```

Out[28]: <seaborn.axisgrid.FacetGrid at 0x2012139bd30>



Feature Exploration with aggregation

In [29]:
```python
def plot_group(x_unique, group_means, x_label, title, palette):
    """
    Plots bar plots of fare_amount against the various forms of grouped data
    x_unique: Unique values for the x-axis
    group_means: A list of the means for each unique group
    palette: The color palette to be used on the plot
    """
    plt.figure(figsize=(10,5))
    sns.barplot(x = x_unique, y = group_means['fare_amount'], palette=palette)
    plt.xlabel(x_label)
    plt.title(title, pad=25)
```

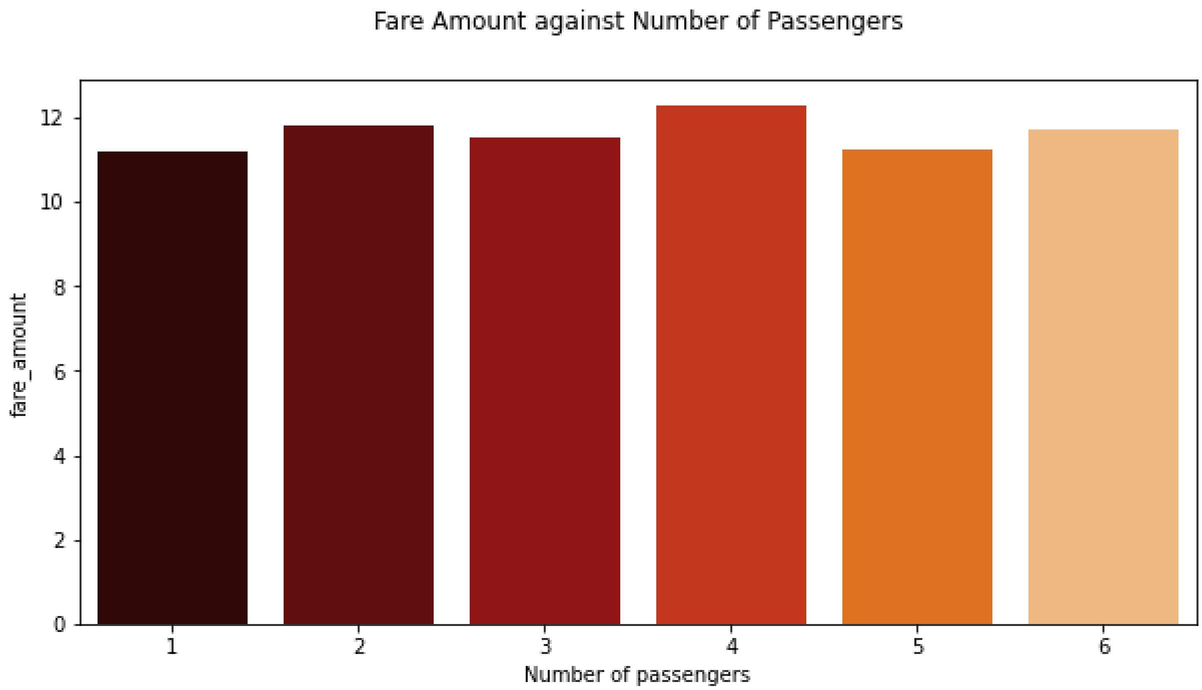
Passengers

In [30]:
```python
# Grouping the data by passengers
passenger_group_means = fare_df.groupby(['passenger_count']).mean()
passenger_group_means
```

Out[30]:

passenger_count	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude
1	11.207628	-73.975515	40.751302	-73.974558	40.751529
2	11.808544	-73.975420	40.749979	-73.974457	40.750714
3	11.507638	-73.976502	40.749825	-73.975723	40.750671
4	11.711905	-73.976867	40.749248	-73.976559	40.750427
5	11.223965	-73.975040	40.751344	-73.974075	40.751545
6	12.289971	-73.974620	40.751043	-73.974267	40.751187

In [31]:
```
# Plotting the grouped data
plot_group(x_unique=fare_df['passenger_count'].unique(),
                 group_means=passenger_group_means,
                 x_label='Number of passengers',
                 title='Fare Amount against Number of Passengers',
                 palette='gist_heat')
```



Year

In [32]:
```
# Grouping the data by years
year_group_means = fare_df.groupby(['year']).mean()
year_group_means
```

Out[32]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenge
year						
2009	10.027271	-73.975592	40.751915	-73.974579	40.751853	1
2010	10.163705	-73.975931	40.751331	-73.974887	40.751463	1
2011	10.441670	-73.976500	40.750992	-73.975348	40.751384	1
2012	11.192015	-73.975620	40.750728	-73.974673	40.751158	1
2013	12.599509	-73.975123	40.750481	-73.974528	40.750978	1
2014	12.905375	-73.975051	40.750554	-73.974014	40.751018	1
2015	13.012596	-73.973978	40.750909	-73.973571	40.751748	1

```
In [33]: # Plotting the grouped data
plot_group(x_unique=fare_df['year'].unique(),
                 group_means=year_group_means,
                 x_label='Year',
                 title='Fare Amount against Years',
                 palette='rainbow')
```

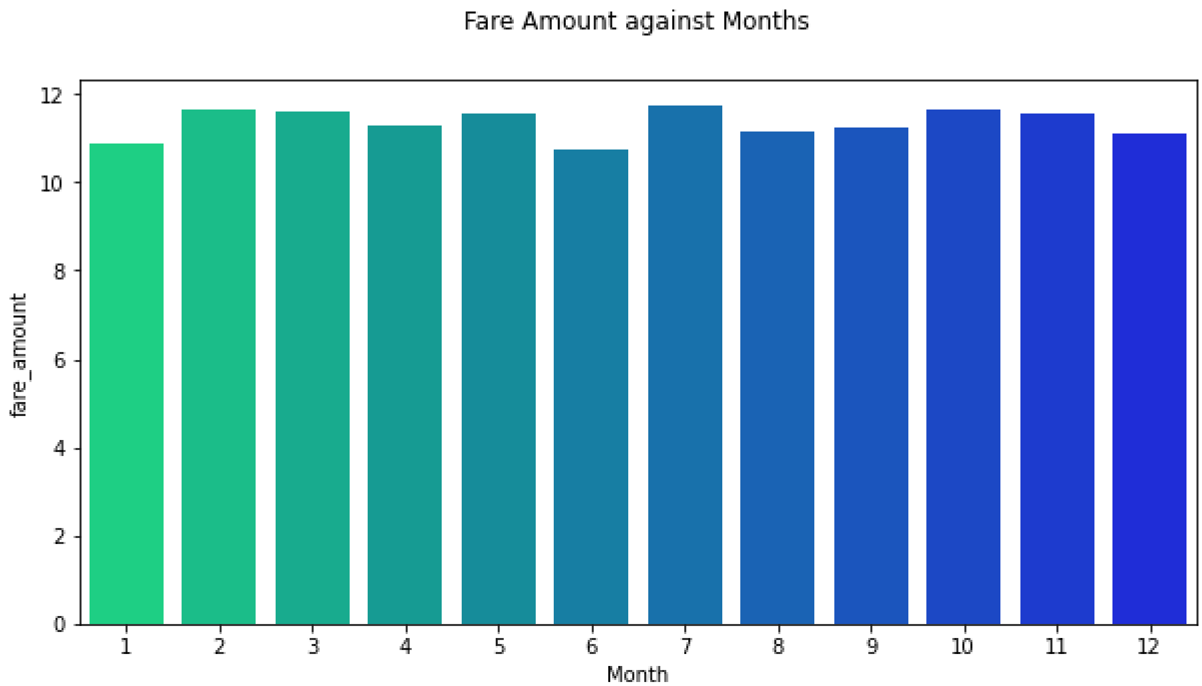
Month

In [34]:
```python
# Grouping the data by months
month_group_means = fare_df.groupby(['month']).mean()
month_group_means
```

Out[34]:

month	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passeng
1	10.741288	-73.975548	40.751500	-73.974876	40.751739	
2	10.894426	-73.976548	40.751115	-73.975150	40.751697	
3	11.138471	-73.976103	40.751170	-73.974754	40.751427	
4	11.297897	-73.975677	40.751225	-73.974681	40.751610	
5	11.629131	-73.975037	40.751100	-73.973883	40.751493	
6	11.546246	-73.975118	40.750952	-73.974453	40.751377	
7	11.126462	-73.975575	40.750283	-73.974881	40.750607	
8	11.234443	-73.975448	40.750055	-73.974710	40.750404	
9	11.745191	-73.975491	40.750459	-73.974501	40.750742	
10	11.671659	-73.975425	40.750943	-73.974792	40.751249	
11	11.561637	-73.975150	40.751350	-73.974388	40.751735	
12	11.637575	-73.975031	40.751498	-73.974113	40.751717	

In [35]:
```
# Plotting the grouped data
plot_group(x_unique=fare_df['month'].unique(),
                 group_means=month_group_means,
                 x_label='Month',
                 title='Fare Amount against Months',
                 palette='winter_r')
```



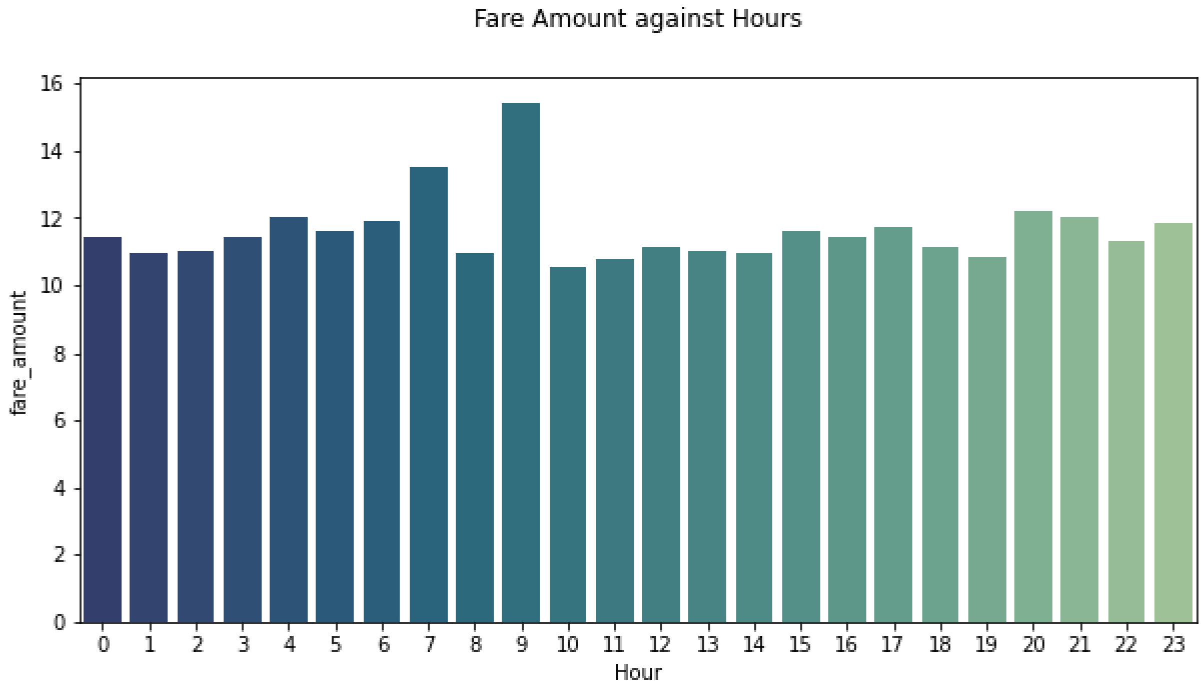
Hour

In [36]:
```python
# Grouping the data by hours
hour_group_means = fare_df.groupby(['hour']).mean()
hour_group_means
```

Out[36]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenge
hour						
0	11.717357	-73.978900	40.743901	-73.973508	40.746093	1
1	11.424954	-73.981521	40.742525	-73.971938	40.745235	1
2	11.450369	-73.983197	40.741704	-73.970609	40.745591	1
3	12.013269	-73.984192	40.741631	-73.968255	40.746004	1
4	13.521498	-73.980923	40.744385	-73.959845	40.745817	1
5	15.421365	-73.969920	40.747179	-73.955097	40.747231	1
6	12.208275	-73.971297	40.750995	-73.967032	40.751106	1
7	10.983889	-73.972621	40.753822	-73.972943	40.753886	1
8	10.922314	-73.974504	40.754128	-73.976370	40.752855	1
9	10.839921	-73.975597	40.754060	-73.977689	40.752345	1
10	10.940560	-73.974572	40.754620	-73.976502	40.753389	1
11	11.121389	-73.974613	40.754616	-73.975902	40.753704	1
12	11.145677	-73.975196	40.754234	-73.975824	40.753391	1
13	11.618821	-73.974300	40.753425	-73.974573	40.753040	1
14	11.876154	-73.973897	40.753078	-73.973769	40.753030	1
15	12.033399	-73.972528	40.753247	-73.972819	40.753985	1
16	11.861820	-73.971178	40.753175	-73.973212	40.754383	1
17	11.407867	-73.972845	40.753052	-73.974676	40.754254	1
18	10.940091	-73.975000	40.753067	-73.976562	40.753134	1
19	10.562280	-73.976166	40.751639	-73.977392	40.751679	1
20	10.797137	-73.976364	40.749691	-73.977283	40.749891	1
21	10.995093	-73.976834	40.748929	-73.976535	40.749610	1
22	11.293823	-73.977181	40.748047	-73.975706	40.748948	1
23	11.594215	-73.977185	40.746597	-73.974859	40.747640	1

In [37]:
```
# Plotting the grouped data
plot_group(x_unique=fare_df['hour'].unique(),
                 group_means=hour_group_means,
                 x_label='Hour',
                 title='Fare Amount against Hours',
                 palette='crest_r')
```



According to the 4 immediate graphs above, we can deduce that in the year period of 2009 - 2015:

1. An average of 10-15 dollars were spent per hour on Taxi Fare in New York

2. An average of 10-12 dollars were spent per month

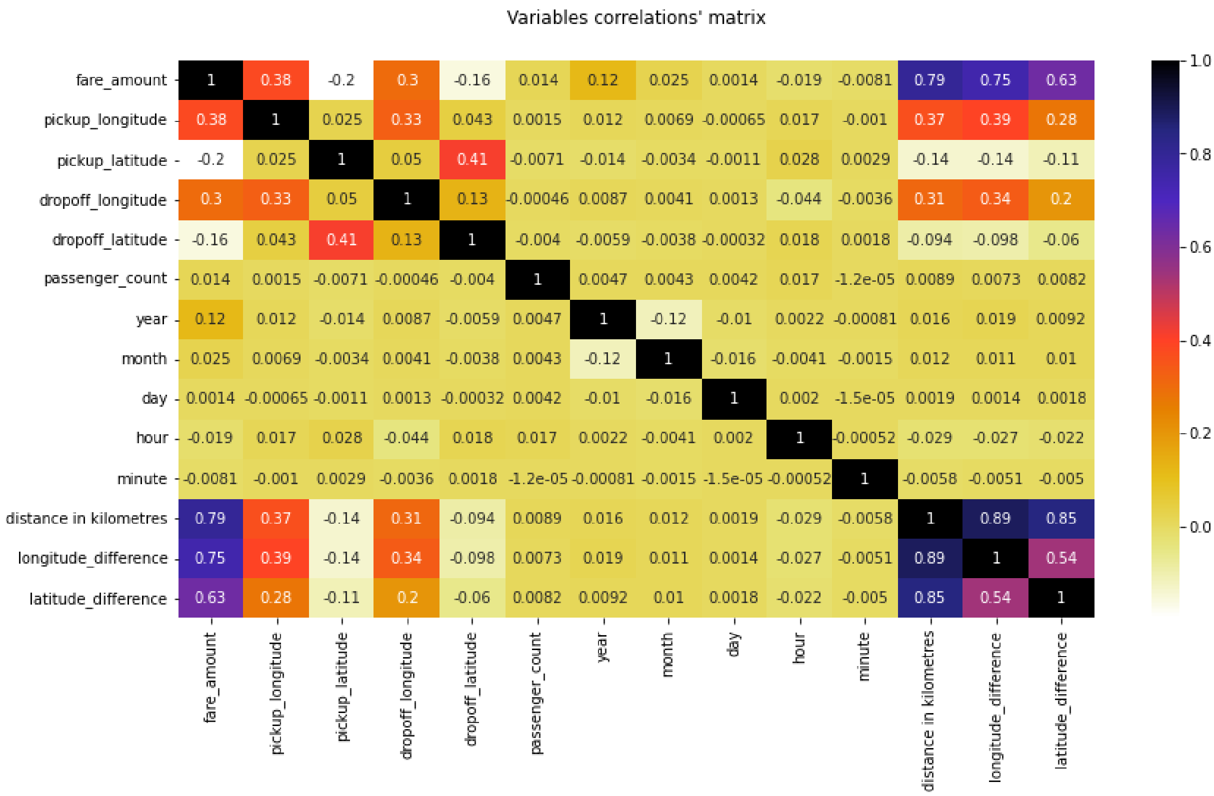
3. Fare has been on the increase over the years (owing to population growth and industrialization) and we could say that this could continue in the years after 2015.

4. For every trip, an average of 11-12 dollars was spent on Taxi Fare

Feature exploration with the correlation matrix

```
In [38]: corr_matrix = fare_df.corr()
         plt.figure(figsize=(14,7))
         sns.heatmap(corr_matrix, cmap=plt.cm.CMRmap_r, annot=True)
         plt.title("Variables correlations' matrix", pad=25)
```

```
Out[38]: Text(0.5, 1.0, "Variables correlations' matrix")
```




```
In [39]: # Correlations of features with the target variable
         corr_target = abs(corr_matrix['fare_amount'])
         corr_target
```

```
Out[39]: fare_amount               1.000000
         pickup_longitude          0.380424
         pickup_latitude           0.197608
         dropoff_longitude         0.299773
         dropoff_latitude          0.157974
         passenger_count           0.013706
         year                      0.117924
         month                     0.025349
         day                       0.001423
         hour                      0.019097
         minute                    0.008093
         distance in kilometres    0.794570
         longitude_difference      0.746860
         latitude_difference       0.631502
         Name: fare_amount, dtype: float64
```

The **distance in kilometers** feature has the highest correlation with the **'day'** feature having the lowest correlation with the target variable **'fare_amount'**

In [40]:
```
fare_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 965221 entries, 0 to 999999
Data columns (total 14 columns):
 #   Column                 Non-Null Count   Dtype  
---  ------                 --------------   -----  
 0   fare_amount            965221 non-null  float64
 1   pickup_longitude       965221 non-null  float64
 2   pickup_latitude        965221 non-null  float64
 3   dropoff_longitude      965221 non-null  float64
 4   dropoff_latitude       965221 non-null  float64
 5   passenger_count        965221 non-null  int64  
 6   year                   965221 non-null  int64  
 7   month                  965221 non-null  int64  
 8   day                    965221 non-null  int64  
 9   hour                   965221 non-null  int64  
 10  minute                 965221 non-null  int64  
 11  distance in kilometres  965221 non-null  float64
 12  longitude_difference   965221 non-null  float64
 13  latitude_difference    965221 non-null  float64
dtypes: float64(8), int64(6)
memory usage: 142.7 MB
```

Feature Selection

Out of the 13 features, we could select the most relevant features using the correlation values

In [41]:
```
relevant_features = corr_target[corr_target > 0.5]
relevant_features
```

Out[41]:
```
fare_amount               1.000000
distance in kilometres    0.794570
longitude_difference      0.746860
latitude_difference       0.631502
Name: fare_amount, dtype: float64
```

The distance in kilometers, longitude_difference and latitude_difference are our most relevant features

In [42]:
```
# Create a new DataFrame with only the relevant features
data_with_relevant = fare_df[['fare_amount', 'distance in kilometres', 'longitude
data_with_relevant
```

Out[42]:

	fare_amount	distance in kilometres	longitude_difference	latitude_difference
0	4.5	1.030761	0.002701	0.009041
1	16.9	8.450738	0.036780	0.070701
2	5.7	1.389495	0.008504	0.010708
3	7.7	2.799280	0.004437	0.024949
4	5.3	1.999212	0.011440	0.015754
...	...	...	...	...
999994	20.0	6.433821	0.076395	0.000935
999995	7.0	1.879672	0.017480	0.010516
999997	10.5	1.761816	0.014974	0.011062
999998	6.9	1.842677	0.003485	0.016360
999999	4.1	0.758066	0.007185	0.004104

965221 rows × 4 columns

Separating the features from the output (target) variable

In [43]:
```
X1 = data_with_relevant.drop(columns=['fare_amount'])
y1 = data_with_relevant['fare_amount']
X1, y1
```

Out[43]:
```
(        distance in kilometres  longitude_difference  latitude_difference
 0                     1.030761              0.002701             0.009041
 1                     8.450738              0.036780             0.070701
 2                     1.389495              0.008504             0.010708
 3                     2.799280              0.004437             0.024949
 4                     1.999212              0.011440             0.015754
 ...                        ...                   ...                  ...
 999994                6.433821              0.076395             0.000935
 999995                1.879672              0.017480             0.010516
 999997                1.761816              0.014974             0.011062
 999998                1.842677              0.003485             0.016360
 999999                0.758066              0.007185             0.004104

 [965221 rows x 3 columns],
 0          4.5
 1         16.9
 2          5.7
 3          7.7
 4          5.3
          ...
 999994    20.0
 999995     7.0
 999997    10.5
 999998     6.9
 999999     4.1
 Name: fare_amount, Length: 965221, dtype: float64)
```

Machine Learning model

In [44]:
```
model1 = LinearRegression()
X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, test_size=0.3, ra
model1.fit(X1_train, y1_train)
kfold1 = KFold(n_splits=10, random_state=42, shuffle=True)
cv_results1 = -(cross_val_score(model1, X1_test, y1_test, cv=kfold1, scoring='neg
print(f"Linear Regression Error : {np.mean(cv_results1)}")
```

```
Linear Regression Error : 36.736948145095155
```

Splitting data into X2 and y2, where features had not been dropped to create model2

In [45]:
```python
X2 = fare_df.drop(columns=['fare_amount'])
y2 = fare_df['fare_amount']
X2, y2
```

Out[45]:
```
(        pickup_longitude  pickup_latitude  dropoff_longitude  \
 0              -73.844311        40.721319         -73.841610   
 1              -74.016048        40.711303         -73.979268   
 2              -73.982738        40.761270         -73.991242   
 3              -73.987130        40.733143         -73.991567   
 4              -73.968095        40.768008         -73.956655   
 ...                   ...              ...                ...   
 999994         -73.872919        40.774106         -73.796524   
 999995         -73.976676        40.785630         -73.959196   
 999997         -73.978423        40.751135         -73.993397   
 999998         -73.980317        40.759482         -73.976832   
 999999         -74.006635        40.741598         -73.999450   

         dropoff_latitude  passenger_count  year  month  day  hour  minute  \
 0              40.712278                1  2009      6   15    17      26   
 1              40.782004                1  2010      1    5    16      52   
 2              40.750562                2  2011      8   18     0      35   
 3              40.758092                1  2012      4   21     4      30   
 4              40.783762                1  2010      3    9     7      51   
 ...                  ...              ...   ...    ...  ...   ...     ...   
 999994         40.775041                2  2014      6   29    22      57   
 999995         40.775114                1  2014      9   13    21      44   
 999997         40.762197                5  2013      4   26    14       3   
 999998         40.743122                1  2011      7    8     0      29   
 999999         40.745702                1  2009     12   31    14      30   

         distance in kilometres  longitude_difference  latitude_difference  
 0                     1.030761              0.002701             0.009041  
 1                     8.450738              0.036780             0.070701  
 2                     1.389495              0.008504             0.010708  
 3                     2.799280              0.004437             0.024949  
 4                     1.999212              0.011440             0.015754  
 ...                        ...                   ...                  ...  
 999994                6.433821              0.076395             0.000935  
 999995                1.879672              0.017480             0.010516  
 999997                1.761816              0.014974             0.011062  
 999998                1.842677              0.003485             0.016360  
 999999                0.758066              0.007185             0.004104  

 [965221 rows x 13 columns],
 0          4.5
 1         16.9
 2          5.7
 3          7.7
 4          5.3
          ... 
 999994    20.0
 999995     7.0
 999997    10.5
 999998     6.9
 999999     4.1
 Name: fare_amount, Length: 965221, dtype: float64)
```

Training another model (model2) based on the dataset where features had not been dropped

In [46]:
```
model2 = LinearRegression()
X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, test_size=0.3, ra
model2.fit(X2_train, y2_train)
kfold2 = KFold(n_splits=10, random_state=42, shuffle=True)
cv_results2 = -(cross_val_score(model2, X2_test, y2_test, cv=kfold2, scoring='neg
print(f"Linear Regression Error : {np.mean(cv_results2)}")
```

```
Linear Regression Error : 32.98854262303762
```

From the results, we can see that model2 has better performance. probably due to [the assumption in Linear algebra (https://towardsdatascience.com/feature-selection-with-pandas-e3690ad8504b#14d9)](https://towardsdatascience.com/feature-selection-with-pandas-e3690ad8504b#14d9) that : independent variables need to be uncorrelated with one another.

In [47]:
```
# Correlation between the chosen relevant features
print(fare_df[["distance in kilometres","latitude_difference"]].corr())
print(fare_df[["longitude_difference","distance in kilometres"]].corr())
print(fare_df[["longitude_difference","latitude_difference"]].corr())
```

```
                        distance in kilometres  latitude_difference
distance in kilometres                1.000000             0.848074
latitude_difference                   0.848074             1.000000
                      longitude_difference  distance in kilometres
longitude_difference              1.000000                0.890916
distance in kilometres            0.890916                1.000000
                      longitude_difference  latitude_difference
longitude_difference              1.000000             0.539039
latitude_difference               0.539039             1.000000
```

The relevant features had a high correlation with one another

We can also say that some of dropped features had some relevance to some extent

In this case, I would opt to use model2 (the model trained on all features) for deployment

Saving the model

In [48]:
```
joblib.dump(model2, 'ny_taxifare_predictor.joblib')
```

Out[48]:
```
['ny_taxifare_predictor.joblib']
```

In [49]:
```
fare_df.describe()
```

Out[49]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passen
count	965221.000000	965221.000000	965221.000000	965221.000000	965221.000000	9652
mean	11.345486	-73.975520	40.750993	-73.974596	40.751340	
std	9.678684	0.038662	0.028379	0.037112	0.032034	
min	0.010000	-78.650908	40.300233	-78.659447	40.300223	
25%	6.000000	-73.992284	40.736591	-73.991579	40.735597	
50%	8.500000	-73.982107	40.753424	-73.980624	40.753890	
75%	12.500000	-73.968457	40.767583	-73.965480	40.768424	
max	495.000000	-72.702870	43.183332	-72.196091	44.600000	