Credit Card Spending in India - Analysis and prediction modelling

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

plt.rcParams['figure.figsize'] = (20, 10)
plt.style.use('ggplot')
```

Exploratory Data Analysis

In [ ]:
```
df = pd.read_csv("data.csv")
```

In [ ]:
```
# get stats of the dataset
print(df["Card Type"].unique())
print(df["Exp Type"].unique())
print(df["Gender"].unique())
print("The number of unique cities - " + str(len(df["City"].unique())))
df.describe()
```

```
['Gold' 'Platinum' 'Silver' 'Signature']
['Bills' 'Food' 'Entertainment' 'Grocery' 'Fuel' 'Travel']
['F' 'M']
The number of unique cities - 986
```

Out[ ]:

	index	Amount
count	26052.000000	26052.000000
mean	13025.500000	156411.537425
std	7520.708943	103063.254287
min	0.000000	1005.000000
25%	6512.750000	77120.250000
50%	13025.500000	153106.500000
75%	19538.250000	228050.000000
max	26051.000000	998077.000000

In [ ]:
```
# add an year column
df["Year"] = df["Date"].apply(lambda x: "20" + x[-2:])
df.head()
```

Out[ ]:

	index	City	Date	Card Type	Exp Type	Gender	Amount	Year
0	0	Delhi, India	29-Oct-14	Gold	Bills	F	82475	2014
1	1	Greater Mumbai, India	22-Aug-14	Platinum	Bills	F	32555	2014
2	2	Bengaluru, India	27-Aug-14	Silver	Bills	F	101738	2014
3	3	Greater Mumbai, India	12-Apr-14	Signature	Bills	F	123424	2014
4	4	Bengaluru, India	5-May-15	Gold	Bills	F	171574	2015

In [ ]:

```python
# checking null values
df.isna().sum()
```

Out[ ]:

```
index        0
City         0
Date         0
Card Type    0
Exp Type     0
Gender       0
Amount       0
Year         0
dtype: int64
```

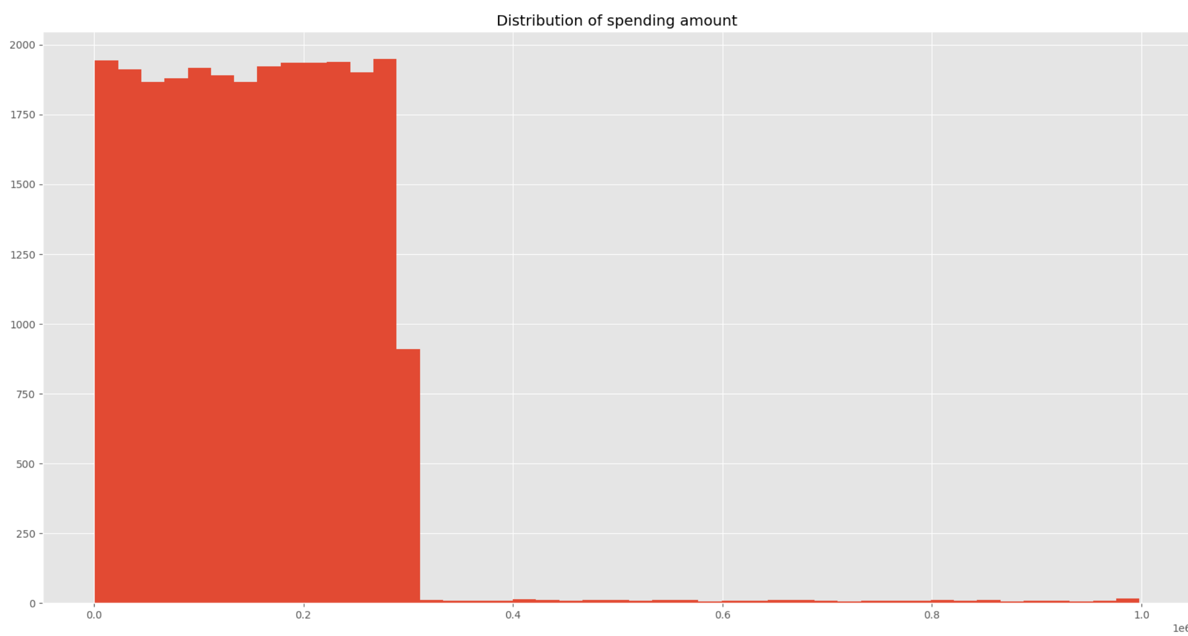
In [ ]:

```python
plt.hist(df['Amount'], bins=int(45/1))

plt.title('Distribution of spending amount')
```

Out[ ]:

```
Text(0.5, 1.0, 'Distribution of spending amount')
```

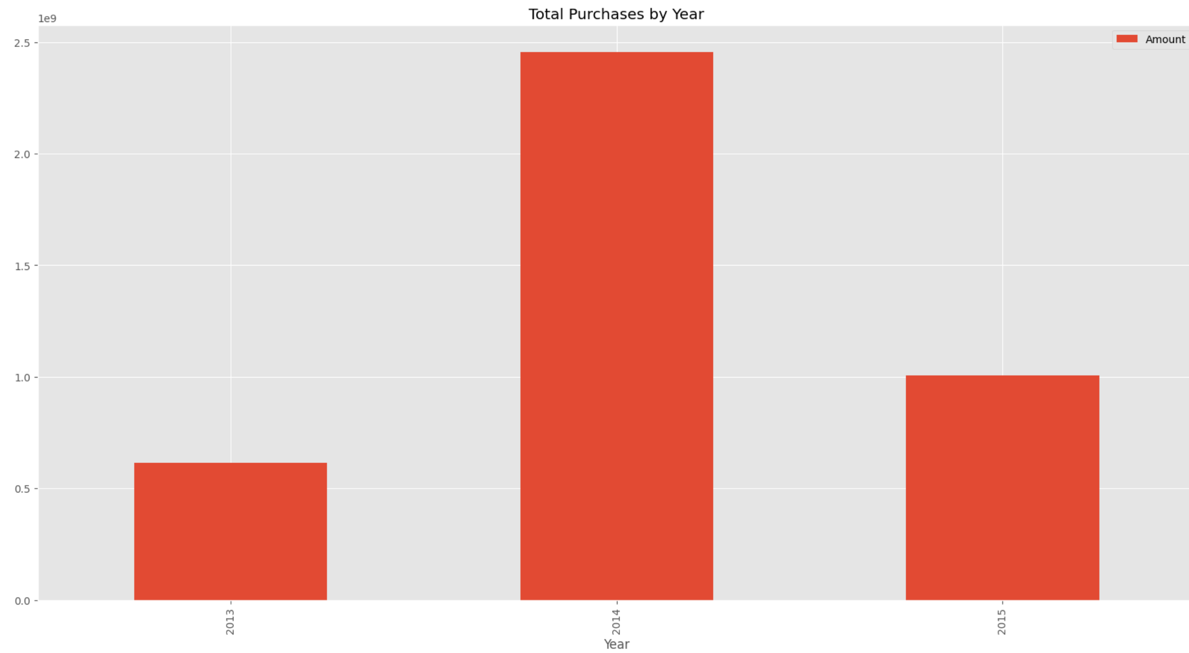


In [ ]:

```python
fig = df[['Year','Amount']].groupby('Year').sum()
fig = fig.sort_values(by='Year', ascending=True)
fig.plot(kind='bar', title='Total Purchases by Year')
```

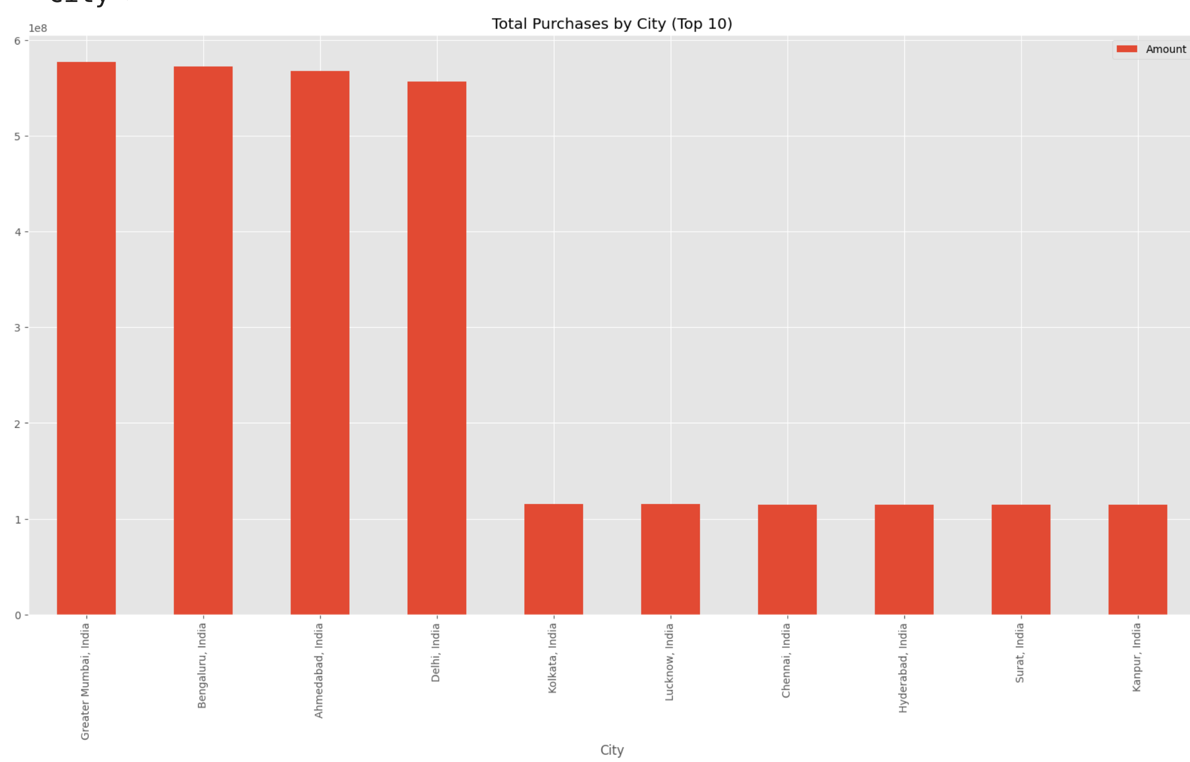
Out[ ]:

```
<AxesSubplot: title={'center': 'Total Purchases by Year'}, xlabel='Year'>
```



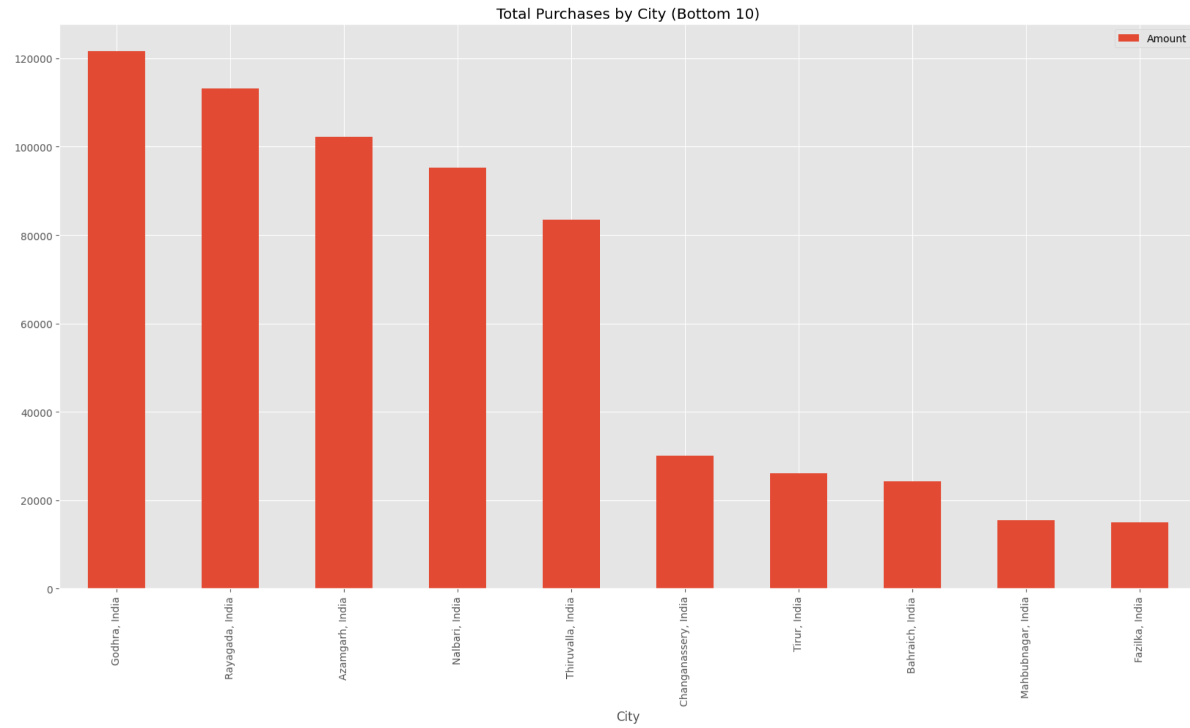
In [ ]:
```
fig = df[['City','Amount']].groupby('City').sum()
fig = fig.sort_values(by='Amount', ascending=False)[0:10]
fig.plot(kind='bar', title='Total Purchases by City (Top 10)')
```

Out[ ]: <AxesSubplot: title={'center': 'Total Purchases by City (Top 10)'}, xlabel ='City'>



In [ ]:
```
fig = df[['City','Amount']].groupby('City').sum()
fig = fig.sort_values(by='Amount', ascending=False)[-10:]
fig.plot(kind='bar', title='Total Purchases by City (Bottom 10)')
```

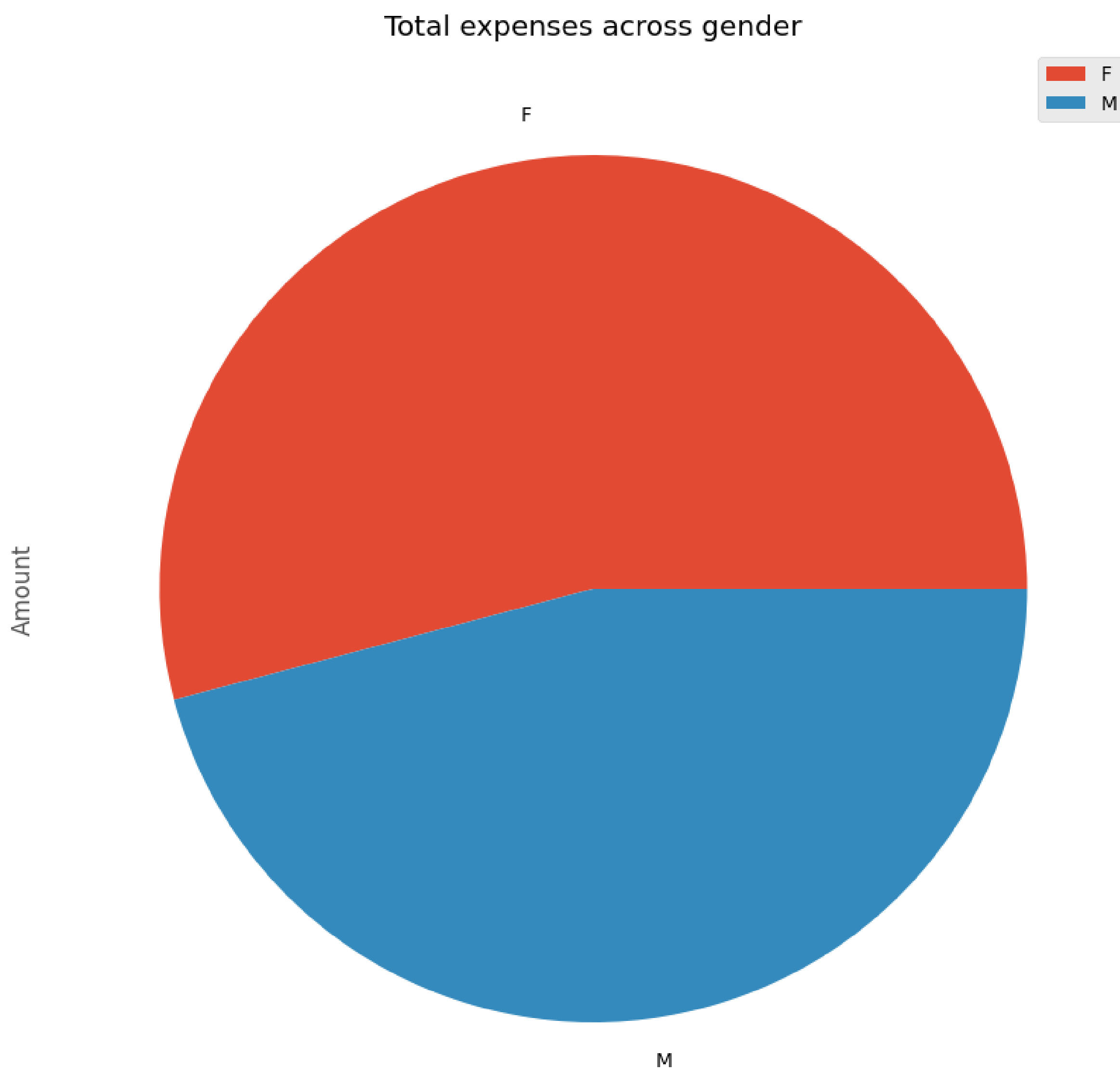
Out[ ]: <AxesSubplot: title={'center': 'Total Purchases by City (Bottom 10)'}, xlab el='City'>



In [ ]:
```python
df.groupby(['Gender']).sum().plot(kind='pie', y='Amount', title='Total exper
```

```
/tmp/ipykernel_4790/3826420844.py:1: FutureWarning: The default value of nu
meric_only in DataFrameGroupBy.sum is deprecated. In a future version, nume
ric_only will default to False. Either specify numeric_only or select only
columns which should be valid for the function.
  df.groupby(['Gender']).sum().plot(kind='pie', y='Amount', title='Total ex
penses across gender')
```

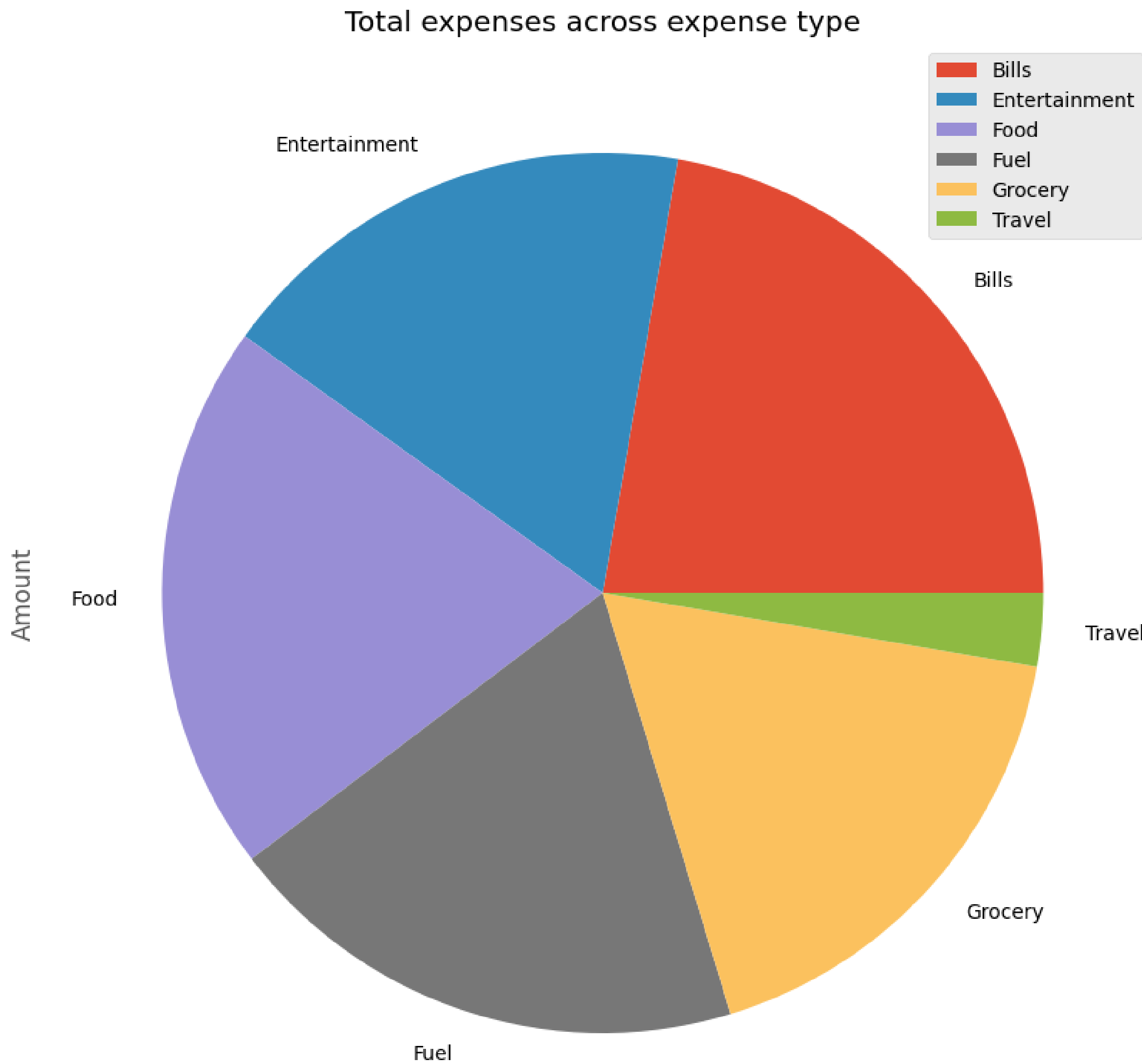
Out[ ]:
```
<AxesSubplot: title={'center': 'Total expenses across gender'}, ylabel='Amo
unt'>
```



In [ ]:
```
df.groupby(['Exp Type']).sum().plot(kind='pie', y='Amount', title='Total exp
```

```
/tmp/ipykernel_4790/3725335283.py:1: FutureWarning: The default value of nu
meric_only in DataFrameGroupBy.sum is deprecated. In a future version, nume
ric_only will default to False. Either specify numeric_only or select only
columns which should be valid for the function.
  df.groupby(['Exp Type']).sum().plot(kind='pie', y='Amount', title='Total
expenses across expense type')
```

Out[ ]:
```
<AxesSubplot: title={'center': 'Total expenses across expense type'}, ylabe
l='Amount'>
```



Preprocess and feature extraction

In [ ]:

```
# Encoding Categorical Data
from sklearn.preprocessing import OrdinalEncoder

ord_enc = OrdinalEncoder()

df["Card Type"] = ord_enc.fit_transform(df[["Card Type"]])
df["Exp Type"] = ord_enc.fit_transform(df[["Exp Type"]])
df["Gender"] = ord_enc.fit_transform(df[["Gender"]])
df["City"] = ord_enc.fit_transform(df[["City"]])
df["Year"] = ord_enc.fit_transform(df[["Year"]])


df.head()
```

Out[ ]:

	index	City	Date	Card Type	Exp Type	Gender	Amount	Year
0	0	126.0	29-Oct-14	0.0	0.0	0.0	82475	1.0
1	1	170.0	22-Aug-14	1.0	0.0	0.0	32555	1.0
2	2	71.0	27-Aug-14	3.0	0.0	0.0	101738	1.0
3	3	170.0	12-Apr-14	2.0	0.0	0.0	123424	1.0
4	4	71.0	5-May-15	0.0	0.0	0.0	171574	2.0

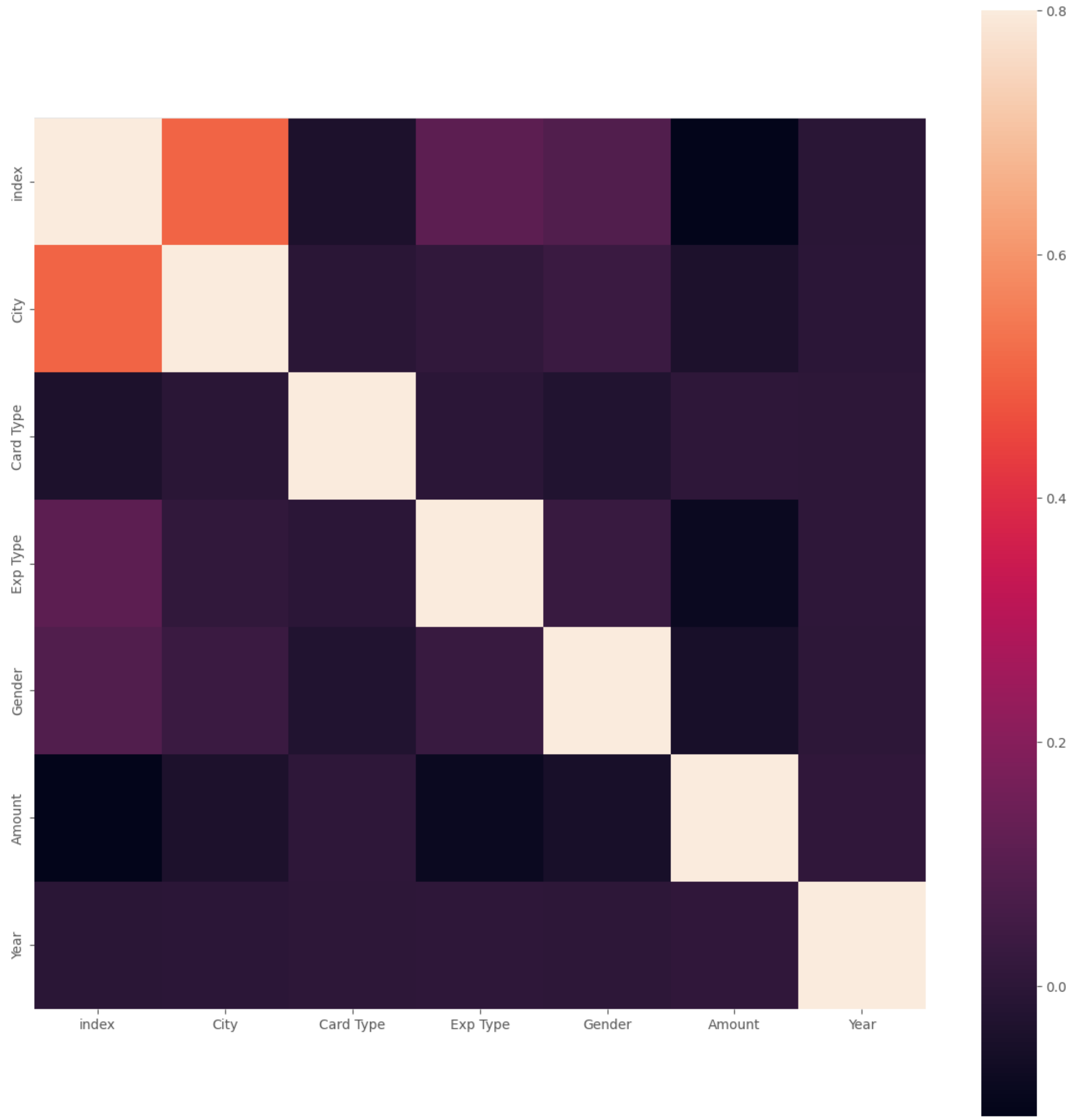
In [ ]:

```
# Testing correlation between features

C_mat = df.corr()
fig = plt.figure(figsize = (15,15))

sns.heatmap(C_mat, vmax = .8, square = True)
plt.show()
```

```
/tmp/ipykernel_4790/3164886862.py:3: FutureWarning: The default value of nu
meric_only in DataFrame.corr is deprecated. In a future version, it will de
fault to False. Select only valid columns or specify the value of numeric_o
nly to silence this warning.
  C_mat = df.corr()
```



In [ ]:
```
# splitting train and test and normalizing categorical data

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X = df[["Year", "City", "Card Type", "Gender"]]
y = df[["Amount"]]

scaler = StandardScaler()
X_norm = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)

X_train, X_test, y_train, y_test= train_test_split(X_norm, y, test_size= 0.2
```

In [ ]:
```
X_train.head()
```

Out[ ]:

	Year	City	Card Type	Gender
1144	-1.759796	-0.955110	1.309596	-0.950992
4274	-0.148840	-0.290037	0.419819	-0.950992
8557	-0.148840	-0.955110	-0.469957	-0.950992
14608	-0.148840	1.044217	-1.359734	-0.950992
10401	-0.148840	-0.470674	0.419819	-0.950992

Regression model

In [ ]:

```
from lazypredict.Supervised import LazyRegressor
from sklearn.utils import all_estimators

# taking a subset of estimators due to memory issues
estimators = [
    "SGDClassifier",
    "KNeighborsClassifier",
    "DecisionTreeClassifier",
]

lazy_estimators = [e for e in all_estimators() if e[0] in estimators]

reg = LazyRegressor(
    verbose=0,
    ignore_warnings=True,
    custom_metric=None,
    predictions=False,
    random_state=42,
    regressors=lazy_estimators,
)

models, predictions = reg.fit(X_train, X_test, y_train, y_test)
```

```
'tuple' object has no attribute '__name__'
Invalid Regressor(s)
100%|██████████| 3/3 [01:37<00:00, 32.35s/it]
```

In [ ]:

```
predictions
```

Out[ ]:

Model	Adjusted R-Squared	R-Squared	RMSE	Time Taken
SGDClassifier	-0.41	-0.41	127184.22	91.18
KNeighborsClassifier	-1.02	-1.01	152145.56	0.15
DecisionTreeClassifier	-1.69	-1.69	175748.08	5.72

The SGDClassifier is the best one out of the three regression models

In [ ]:

```
import numpy as np
from sklearn import linear_model
```

```
from sklearn.metrics import mean_squared_error

SGDClf = linear_model.SGDClassifier(max_iter = 1000, tol=1e-3, penalty = "el
SGDClf.fit(X_train, y_train)
sgd_pred = SGDClf.predict(X_test)
```

In [ ]:

```
li1 = list(zip(range(1, len(y_test.values)), y_test.values))
li2 = list(zip(range(1, len(sgd_pred)), sgd_pred))

plt.ylabel('Label Value')
plt.xlabel('Sample')

plt.scatter(*zip(*li1), s=1, label='Actual')
plt.scatter(*zip(*li2), s=1, label='Predicted')

plt.legend(bbox_to_anchor=(1.15, 1), loc="upper right")
plt.title("Actual vs Predicted")

plt.show()
```