The hackathon data analysis is a summary of the EDA activities you have completed. This should include data cleaning, pre-processing, exploratory analysis, and insights you have generated from your data. Each group will submit and demonstrate one notebook (in ipynb and pdf) that comprises the following sections:

1.Domain: Explain the domain/subdomain of this project (use images)

2.Data: Go over 1 row of the data explaining relevant columns

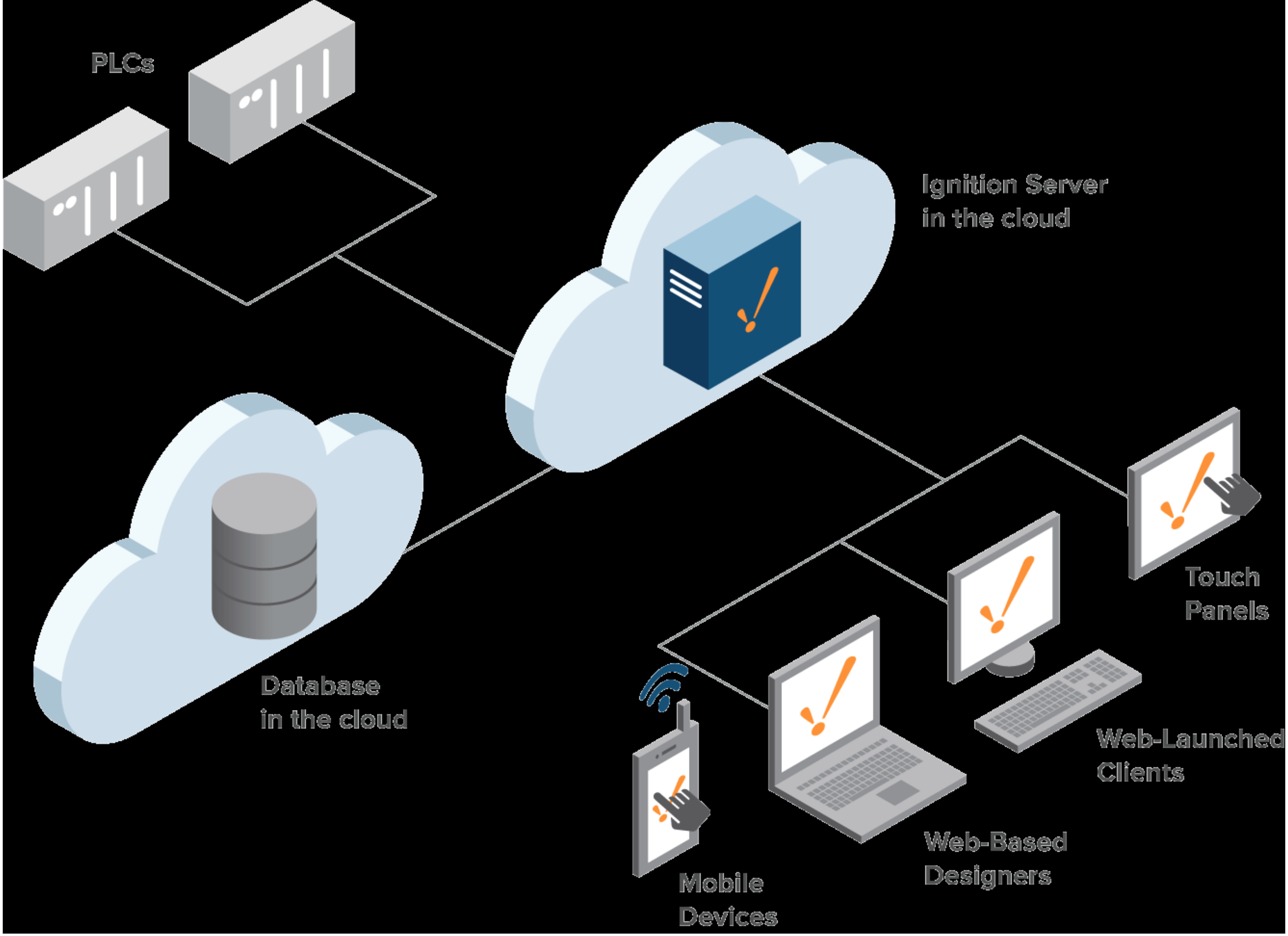
3.Insights: Show intuitive and non-intuitive insights generated from your EDA

4.Expected Objective: Sample input and output

Domain

Cloud Computing

Cloud Computing (NIST): a model for enabling on-demand network access to a shared pool of computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.



5 Key Characteristics of Cloud Computing

> 1. Broad Network Access
> 2. Measured Service
> 3. On Demand Self Service

> 4. Shared Resource Pooling
> 5. Rapid Elasticity

3 Service Models

> 1. **Software as a Service (SaaS)**: Provider's applications running on a cloud infrastructure
>
> 2. **Platform as a Service (PaaS)**: Consumer-created or acquired applications created using tools supported by the provider
>
> 3. **Infrastructure as a Service (IaaS)**: Consumer is able to deploy and run arbitrary software on storage, networks, etc.

Leading Cloud Service Providers

1. **Amazon Web Services (AWS)**

1. **Microsoft Azure**

1. Google Cloud Platform

Cloud Migration: the process of moving a company's digital assets, services, databases, IT resources, and applications either partially, or wholly, into the cloud.

Cloud Management

MultiCloudX: Austin based start-up that functions as a third-party cloud manager to manage cloud storage, operations, and costs

MultiCloudX

Unleash your business potential with the cloud

Data

In [1]:
```
import numpy as np
import pandas as pd

import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
pd.set_option('display.max_columns', None)
```

In [3]:
```
dfCCO = pd.read_csv(r".\Data\cco_cost_monthly.csv")
```

In [4]:
```
dfPYCO = pd.read_csv(r".\Data\pyco_cost_monthly.csv", low_memory=False)
```

In [5]:
```
dfXCO = pd.read_csv(r".\Data\xco_cost_monthly.csv")
```

In [6]:
```
dfCCO.head()
```

Out[6]:

	cloud_resource_id	usage_account_id	billing_account_id	provider_code	usage_amount	currency_code	tax	sub_total	total_cost	system_cur
0	NaN	751355800400	751355800400	aws	1.0	USD	0.0	0.0	0.03	
1	NaN	751355800400	751355800400	aws	2.0	USD	0.0	0.0	0.07	
2	NaN	751355800400	751355800400	aws	1.0	USD	0.0	0.0	0.00	
3	arn:aws:kms:us-east-1:751355800400:key/5fedb8a...	751355800400	751355800400	aws	18.0	USD	0.0	0.0	0.00	
4	arn:aws:cognito-idp:us-east-1:751355800400:use...	751355800400	751355800400	aws	2.0	USD	0.0	0.0	0.00	

cloud_resource_id	usage_account_id	billing_account_id	provider_code	usage_amount	currency_code	tax	sub_total	total_cost	system_cur

Useful: total_cost, service_name Maybe: usage_amount, product_name, usage_type, location_id, cost_type, instance_type Not Useful:cloud_resource_id, usage_account_id, billing_account_id, provider_code, currency_code, tax, sub_total, system_currency_code, conversion_rate, converted_total_cost (redundant), service_code, product_sku, availability_zone, location_id, usage_type_group,

Insights

EDA

Now that the functions that we will use have been created, let's look at some key features of our data. First, we combine the dataframes to get general ideas about the full data we were provided.

In [7]:

```
dfTotal = pd.concat([dfCCO, dfPYCO, dfXCO], axis=0)
dfTotal.shape
```

Out[7]:

```
(984010, 44)
```

So, almost 1 million rows and 44 columns! We will need to aggregate this data further to get a better sense of what it means. Let's now look at the general variability of our columns by seeing how many unique values each has.

In [8]:

```
dfTotal.nunique()
```

Out[8]:

```
cloud_resource_id       279088
usage_account_id             4
billing_account_id           3
provider_code                1
usage_amount            111997
currency_code                1
tax                          1
sub_total                    1
total_cost              132600
system_currency_code         1
conversion_rate              1
converted_total_cost    132600
product_name                47
service_code                41
service_name                40
product_sku                598
availability_zone           13
location_id                 18
usage_type                 525
usage_type_group            61
cost_type                    6
```

```
instance_type                52
blended_rate                479
unblended_rate              143
unblended_cost           127870
category                     60
clock_speed                  11
from_location                24
to_location                  28
transfer_type                 6
pricing_rate_id            2866
public_ondemand_cost      66222
public_ondemand_rate        138
pricing_term                  2
pricing_unit                 56
memory                       17
operating_system              3
vpcu                          6
phsyical_processor           11
volume_type                  10
storage                       6
storage_class                 3
storage_media                 6
invoice_month                19
dtype: int64
```

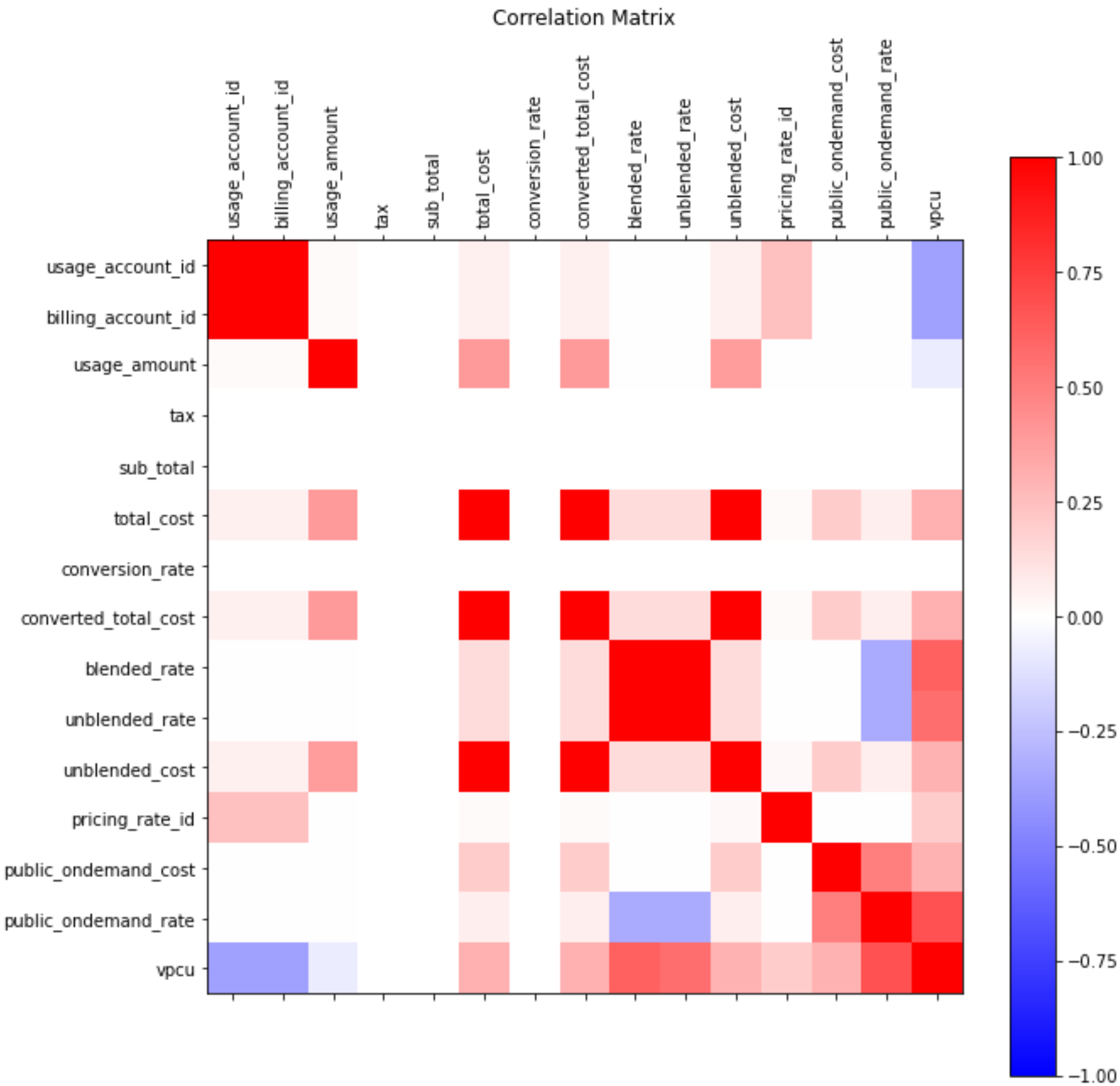
Since features like "conversion_rate" and "provider_code" are unvaried, they can be discarded as features as invariate features are worthless and can be dropped. Next, let's look at a quick correlation matrix of our full dataset to see if anything has a high level of correlation with the total_cost that isn't fully redundant.

In [9]:

```
def plot_corr(dataframe,size=10):
    """
    Plots a correlation matrix as a heat map
    """
    corr = dataframe.corr()
    fig, ax = plt.subplots(figsize=(size, size))
    im = ax.matshow(corr,vmin = -1.0, vmax = 1.0, cmap = "bwr")
    plt.xticks(range(len(corr.columns)), corr.columns, rotation = 90);
    plt.yticks(range(len(corr.columns)), corr.columns);
    plt.colorbar(im, orientation = 'vertical')
    plt.title('Correlation Matrix')
```

In [10]:

```
plot_corr(dfTotal)
```



It looks like there's some light correlation to usage and the vcpu count... nothing too useful, although we may need to make another correlation matrix after some data augmentation. Next, let's look at the %NaN for each column to see which columns are generally unused in the dataset.

In [11]:

```
dfTotal.isna().sum() / dfTotal.shape[0]
```

Out[11]:

```
cloud_resource_id       0.003146
usage_account_id        0.000000
billing_account_id      0.000000
```

```
provider_code            0.000000
usage_amount             0.000000
currency_code            0.000000
tax                      0.000000
sub_total                0.000000
total_cost               0.000000
system_currency_code     0.000000
conversion_rate          0.000000
converted_total_cost     0.000000
product_name             0.000000
service_code             0.084868
service_name             0.084906
product_sku              0.000605
availability_zone        0.995690
location_id              0.000605
usage_type               0.084868
usage_type_group         0.989718
cost_type                0.000000
instance_type            0.996030
blended_rate             0.000000
unblended_rate           0.000000
unblended_cost           0.000000
category                 0.000000
clock_speed              0.996988
from_location            0.577076
to_location              0.577076
transfer_type            0.577076
pricing_rate_id          0.000605
public_ondemand_cost     0.000000
public_ondemand_rate     0.000000
pricing_term             0.097949
pricing_unit             0.000605
memory                   0.996018
operating_system         0.996992
vpcu                     0.996012
phsyical_processor       0.996180
volume_type              0.996372
storage                  0.996179
storage_class            0.998962
storage_media            0.985840
invoice_month            0.000000
dtype: float64
```

It looks like the 3 columns for transfers all need one another to exist. Then, the columns like "instance_type", "volume_type", and "operating_system" all likely point towards the amount of line items that are actual VMs that have been spun up rather than associated costs or abstracted cloud services. It could perhaps be possible to use the amount of rows we have for each of those columns as advisors of fixed costs as certain server instances such as domain controllers, FTP servers, and application servers require high availability.

In [12]:

```
dfPYCO.sort_values("total_cost", ascending=False)[["invoice_month", "product_name", "total_cost"]]
```

Out[12]:

	invoice_month	product_name	total_cost

	invoice_month	product_name	total_cost
65279	2019-09-01	Amazon Elastic Compute Cloud	3284.000000
28934	2019-09-01	AWS Support (Business)	1000.000000
96649	2019-10-01	AWS Support (Business)	1000.000000
114403	2019-10-01	Amazon Elastic File System	879.020880
29088	2019-09-01	Amazon Elastic File System	861.036975
...	...	...	...
567528	2019-12-01	Amazon Elastic Compute Cloud	-3.050000
96842	2019-10-01	Amazon Elastic Compute Cloud	-100.000000
28932	2019-09-01	AWS Support (Business)	-130.774159
96648	2019-10-01	AWS Support (Business)	-708.380000
28935	2019-09-01	AWS Support (Business)	-1000.000000

Here, we see some interesting behaviour where it looks like there were certain charges put forward that then had to be fixed (look at the indices of the largest negative charges and how they're +1 of the large charges).

Plotting

Now, let's make some figures to get an idea of how the big hitter services are affecting costs month over month.

In [13]:

```
def hitter_monthly(df, product):
    """
    Grabs the total cost incurred by the big hitter (product) for each month of the DataFrame, df
    """
    dfFin = df.loc[df["product_name"] == product].groupby("invoice_month").agg({"total_cost": "sum"})
    return dfFin
```

In [39]:

```python
def plot_n_hitters(n_hitters):
    """
    Plots the N biggest overall hitters and the aggregate cost
    """
    top_hitters = dfTotal.groupby("product_name").agg({"total_cost": "sum"}).sort_values("total_cost", ascending=False).i
    company_dfs = [dfXCO, dfPYCO, dfCCO]
    company_names = ["XCO", "PYCO", "CCO"]
    final_df_list = []
    fig, ax = plt.subplots(3,2, figsize=(14, 8.5))
    for index in range(len(company_names)):
        inter_list = []

        for hitter in range(len(top_hitters)):
            new_monthly_df = hitter_monthly(company_dfs[index], top_hitters[hitter])
            ax[index][0].plot(new_monthly_df)
            ax[index][0].tick_params("x", rotation=90)
            inter_list.append(new_monthly_df)


        ax[index][0].set_title(str(company_names[index]) + " big hitters monthly cost")
        ax[index][0].legend(top_hitters, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
        final_df_list.append(inter_list)
        agg_df = company_dfs[index].groupby("invoice_month").agg({"total_cost": "sum"})
        ax[index][1].plot(agg_df, color="red")
        ax[index][1].set_title(str(company_names[index]) + " total monthly cost")
        ax[index][1].tick_params("x", rotation=90)

    fig.tight_layout()
    fig.savefig(r".\figures\graphs_mod" + str(n_hitters) + ".png")
    return final_df_list
```

In [40]:

```
def plot_n_hitters_bycorp(n_hitters):
    """
    Plots the top N big hitters on a company-by-company basis (possibly different for each company)
    """
    company_dfs = [dfXCO, dfPYCO, dfCCO]
    company_names = ["XCO", "PYCO", "CCO"]
    final_df_list = []
    fig, ax = plt.subplots(3,2, figsize=(14, 8.5))
    for index in range(len(company_names)):
        inter_list = []
        top_hitters = company_dfs[index].groupby("product_name").agg({"total_cost": "sum"}).sort_values("total_cost", asc

        for hitter in range(len(top_hitters)):
            new_monthly_df = hitter_monthly(company_dfs[index], top_hitters[hitter])
            ax[index][0].plot(new_monthly_df)
            ax[index][0].tick_params("x", rotation=90)
            inter_list.append(new_monthly_df)


        ax[index][0].set_title(str(company_names[index]) + " big hitters monthly cost")
        ax[index][0].legend(top_hitters, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
        final_df_list.append(inter_list)
        agg_df = company_dfs[index].groupby("invoice_month").agg({"total_cost": "sum"})
        ax[index][1].plot(agg_df, color="red")
        ax[index][1].set_title(str(company_names[index]) + " total monthly cost")
        ax[index][1].tick_params("x", rotation=90)

    fig.tight_layout()
    fig.savefig(r".\figures\company_hitters_" + str(n_hitters) + ".png")
```

The first plot here is a demonstration of how the top 5 overall "big hitters" affect the aggregate cost (right) for each company. We can see that the top 5 big hitters aren't necessarily the biggest cost drivers for each company. Another interesting insight is how for XCO, the graphical shape seems to be driven more by Amazon DocumentDB even though it's not the largest charge most months, it is the biggest contributor of variance.

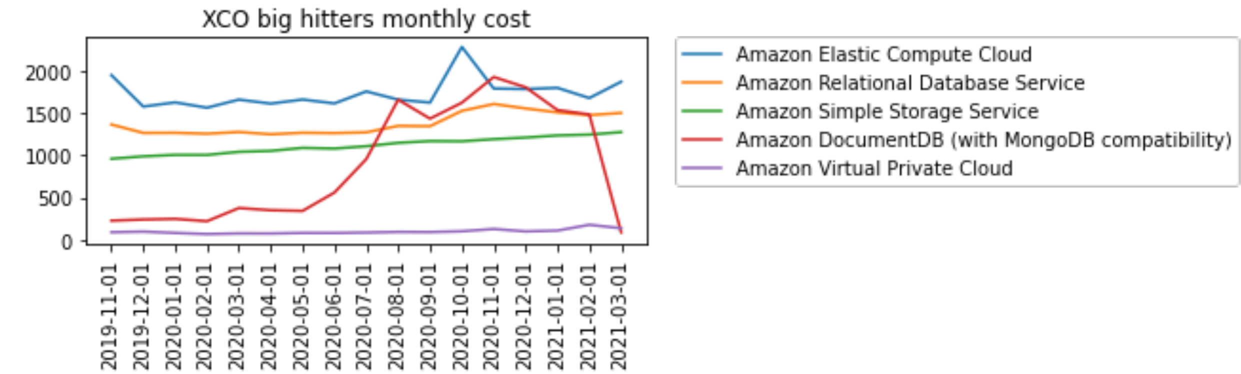
In [41]:

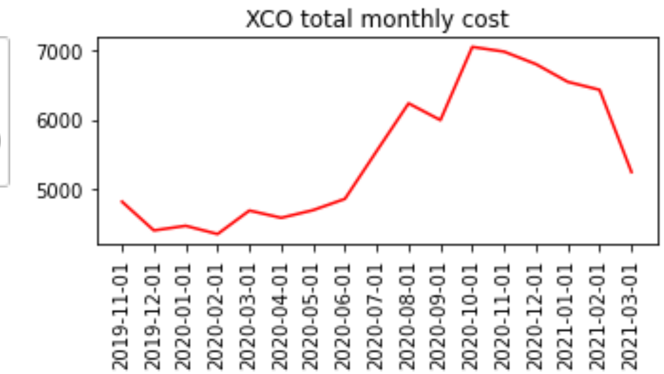
```
top_5 = plot_n_hitters(5)
```

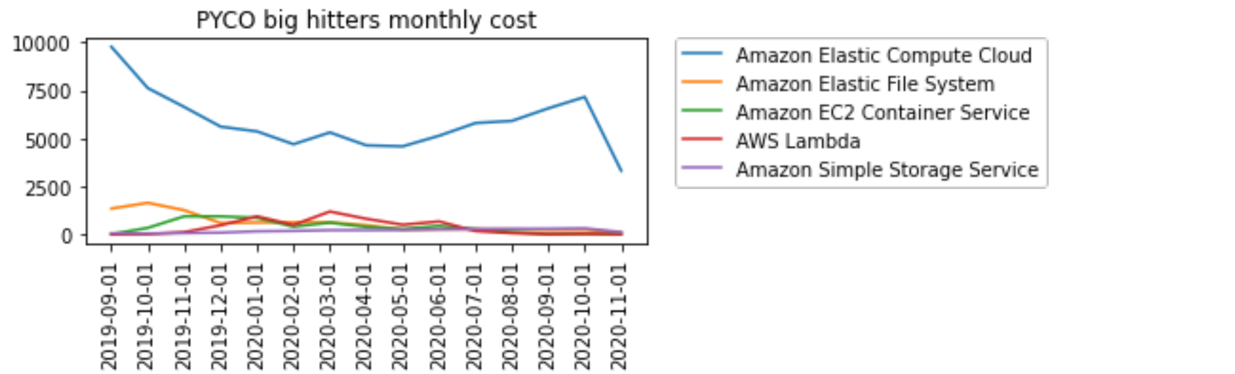
Here, we look at the top 5 top "big hitters" and see that there were a lot of costs that matter to company with a smaller cloud footprint, like CCO that don't necessarily affect a relatively larger corporation like PYCO.

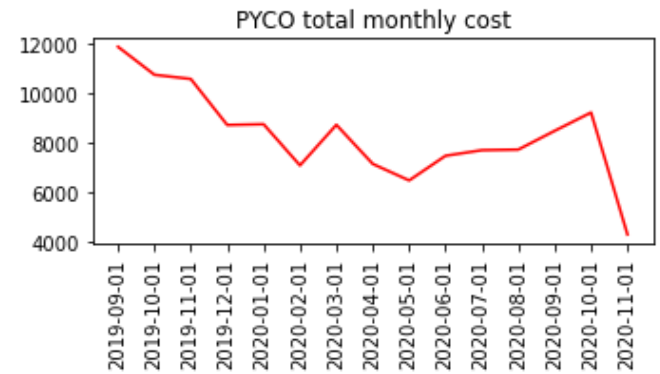
In [42]:

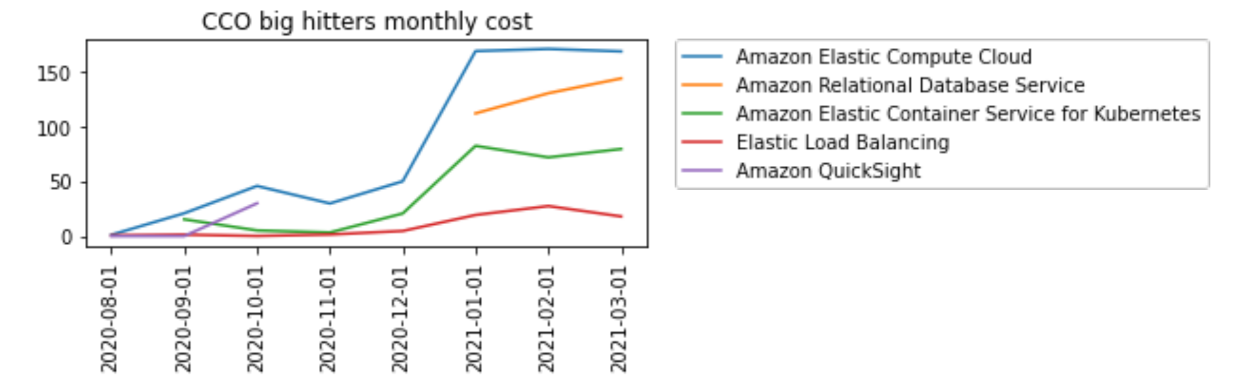
```
plot_n_hitters_bycorp(5)
```

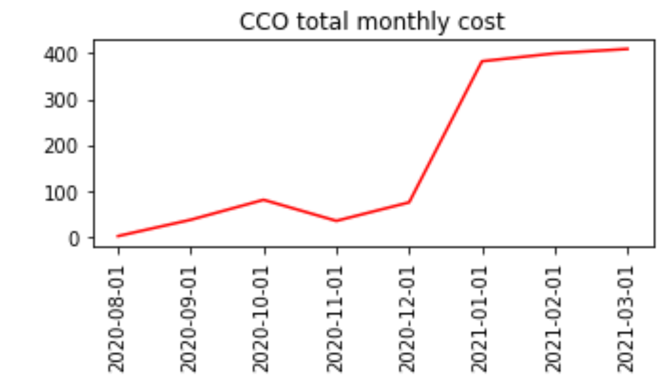












Data Processing

Now, we will engineer our features to be used in the final model. We are taking the usage and total cost of each product name and taking their sum total for each month. This creates a huge amount of features that can then be later cut down.

Other possible datum:

- # of Windows/Linux/RHEL machines
- Amount of VCPUs/Storage Type/Memory used

In [18]:

```python
# First start the creation of our final dataframe, starting with total monthly cost
def process_dataframe(df, fillna=True):
    """
    Processes a dataFrame from what's initially given to a usable form for our model.
    """
    dfProcessed = df.groupby("invoice_month").agg({"total_cost": "sum"})
    products = dfTotal["product_name"].unique()
    # Loops through all of the unique product names
    for product in products:
        # Grabs the monthly total cost and usage for each product
        df_product_monthly = df.loc[df["product_name"] == product].groupby("invoice_month").agg({"total_cost": "sum", "us
        #Concatenates that onto the total dataframe we have
        dfProcessed = pd.concat([dfProcessed, df_product_monthly], axis=1)
    if fillna==True:
        dfProcessed = dfProcessed.fillna(0)
    return dfProcessed
```

In [19]:

```python
dfCCO_processed = process_dataframe(dfCCO, fillna=False)
dfCCO_processed
```

Out[19]:

invoice_month	total_cost	Elastic Load Balancing cost	Elastic Load Balancing usage	Amazon Elastic Compute Cloud cost	Amazon Elastic Compute Cloud usage	Amazon Simple Storage Service cost	Amazon Simple Storage Service usage	AWS Key Management Service cost	AWS Key Management Service usage	Amazon Cognito cost	Amazon Cognito usage	Amazon(
2020-08-01	1.717943	0.547634	24.021073	1.147767	24.566741	0.022542	543.011748	0.0	18.0	0.0	2.0	
2020-09-01	37.258579	1.273340	54.133633	20.755028	824.133404	0.021228	2575.908828	0.0	20.0	0.0	1.0	
2020-10-01	80.664613	0.122506	6.000797	45.572186	1189.248625	0.013111	761.534243	0.0	9.0	NaN	NaN	
2020-11-01	34.867281	1.417530	60.004739	29.653838	1292.703062	0.014404	2149.542143	0.0	167.0	0.0	1.0	
2020-12-01	75.113704	4.633152	194.110072	49.838250	1637.684729	0.016703	3426.625776	0.0	233.0	0.0	1.0	
2021-01-01	382.368784	19.185234	803.845366	168.243566	5142.464178	0.025501	5776.833469	0.0	239.0	0.0	57.0	
2021-02-01	399.512312	27.303523	1141.367505	170.194027	5240.705849	0.018575	3482.610250	0.0	269.0	0.0	7.0	
2021-03-01	409.219731	17.845686	747.001553	167.964576	5917.144565	0.019282	3840.988064	0.0	220.0	0.0	9.0	

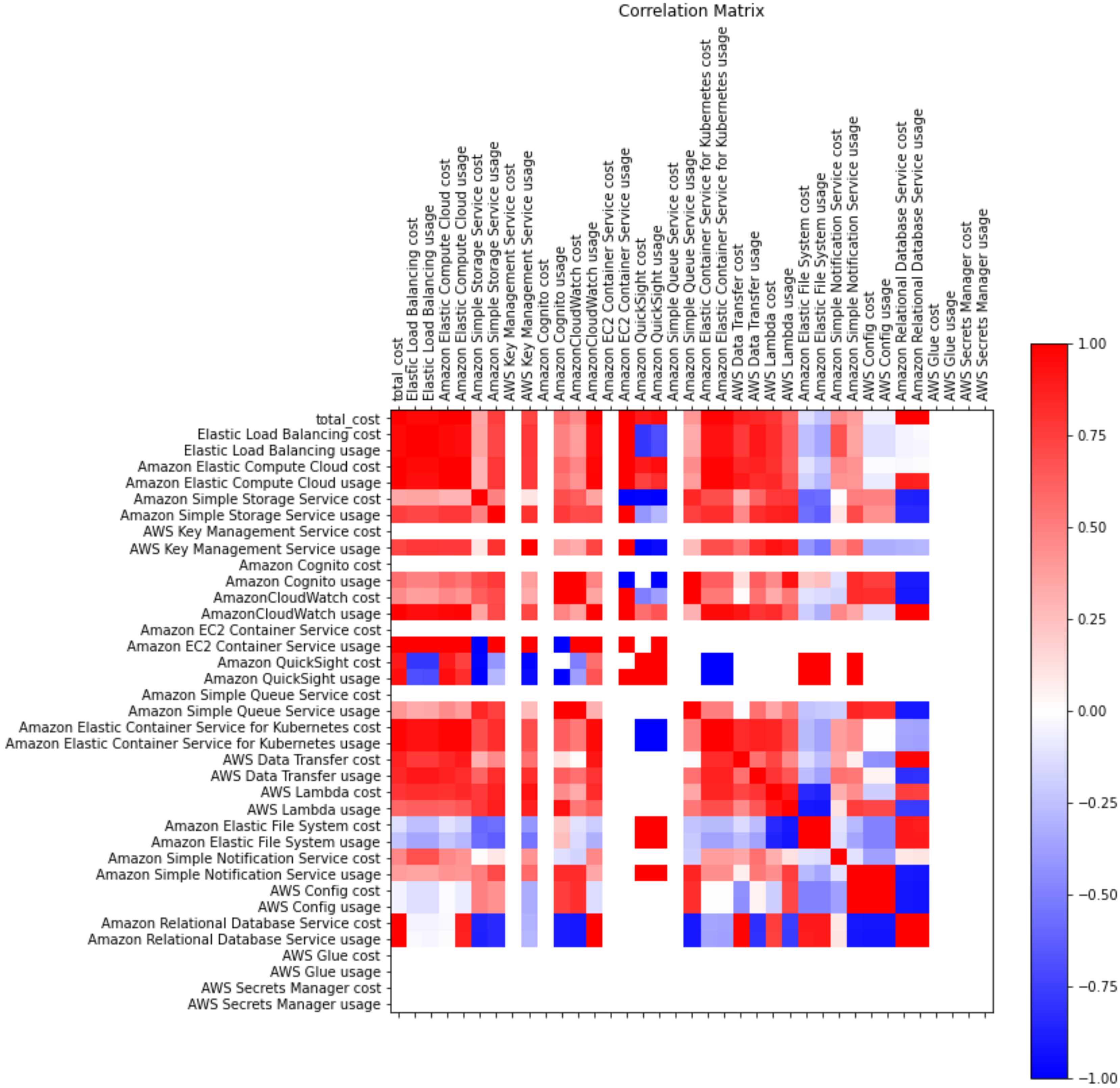
CCO

Interesting things to note:

- The only negative correlations lie with the Amazon Elastic File System
- Cost variance seems to be driven strongly by many different services. Likely points to a ramping up of cloud costs, which was reflected in our earlier plots

In [20]:

```
# CCO Correlation Matrix
plot_corr(dfCCO_processed.dropna(axis=1, how="all"))
```


Correlation Matrix
total_cost
Elastic Load Balancing cost
Elastic Load Balancing usage
Amazon Elastic Compute Cloud cost
Amazon Elastic Compute Cloud usage
Amazon Simple Storage Service cost
Amazon Simple Storage Service usage
AWS Key Management Service cost
AWS Key Management Service usage
Amazon Cognito cost
Amazon Cognito usage
AmazonCloudWatch cost
AmazonCloudWatch usage
Amazon EC2 Container Service cost
Amazon EC2 Container Service usage
Amazon QuickSight cost
Amazon QuickSight usage
Amazon Simple Queue Service cost
Amazon Simple Queue Service usage
Amazon Elastic Container Service for Kubernetes cost
Amazon Elastic Container Service for Kubernetes usage
AWS Data Transfer cost
AWS Data Transfer usage
AWS Lambda cost
AWS Lambda usage
Amazon Elastic File System cost
Amazon Elastic File System usage
Amazon Simple Notification Service cost
Amazon Simple Notification Service usage
AWS Config cost
AWS Config usage
Amazon Relational Database Service cost
Amazon Relational Database Service usage
AWS Glue cost
AWS Glue usage
AWS Secrets Manager cost
AWS Secrets Manager usage
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

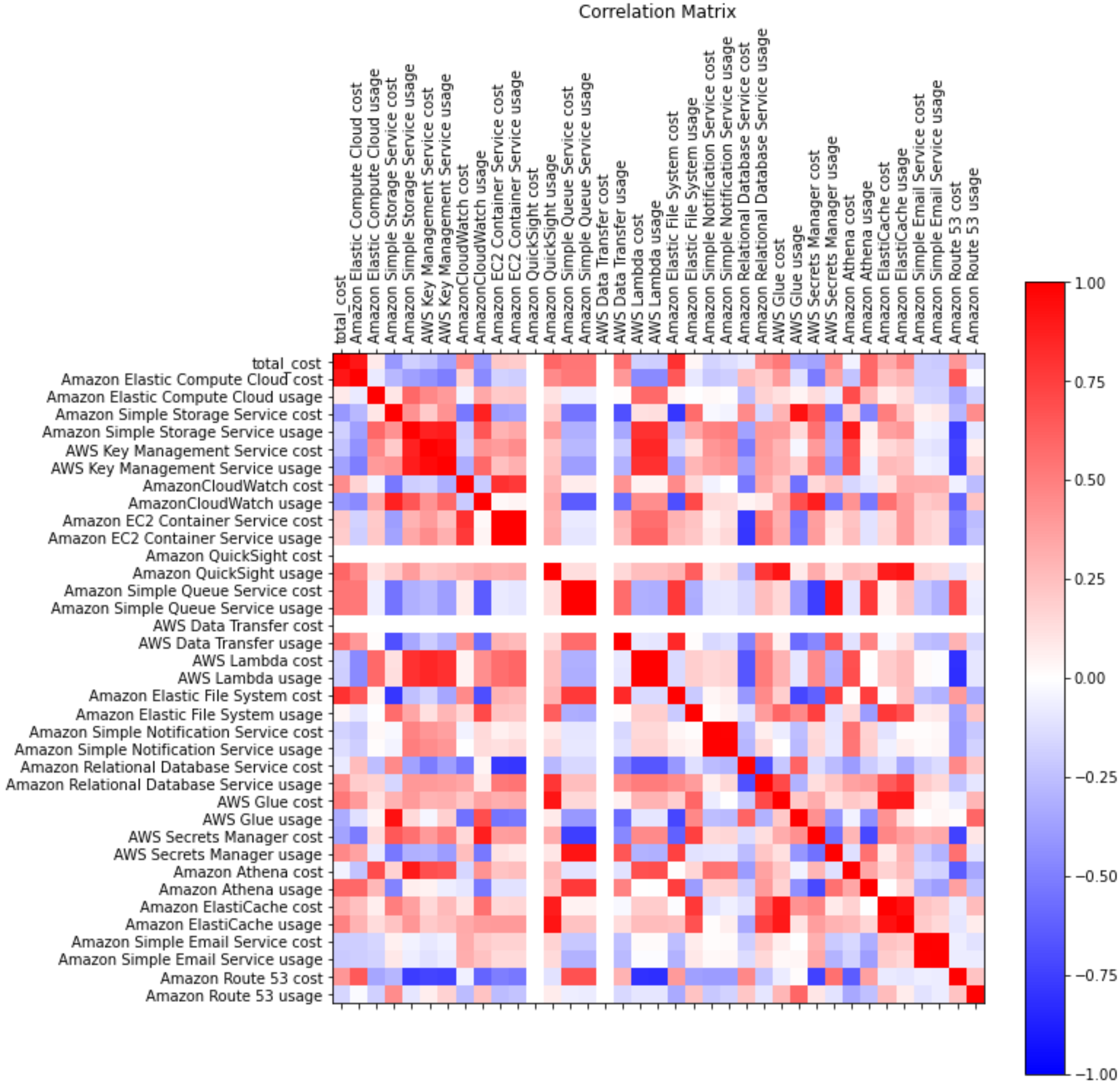
PYCO

Interesting things to note:

- There's a somewhat large amount of negative correlations for cost, but none are very strong.
- The strongest correlation seems to lie with Amazon Elastic File System costs

In [21]:

```
dfPYCO_processed = process_dataframe(dfPYCO, fillna=False)
# PYCO Correlation Matrix
plot_corr(dfPYCO_processed.dropna(axis=1))
```


Correlation Matrix
total_cost
Amazon Elastic Compute Cloud cost
Amazon Elastic Compute Cloud usage
Amazon Simple Storage Service cost
Amazon Simple Storage Service usage
AWS Key Management Service cost
AWS Key Management Service usage
AmazonCloudWatch cost
AmazonCloudWatch usage
Amazon EC2 Container Service cost
Amazon EC2 Container Service usage
Amazon QuickSight cost
Amazon QuickSight usage
Amazon Simple Queue Service cost
Amazon Simple Queue Service usage
AWS Data Transfer cost
AWS Data Transfer usage
AWS Lambda cost
AWS Lambda usage
Amazon Elastic File System cost
Amazon Elastic File System usage
Amazon Simple Notification Service cost
Amazon Simple Notification Service usage
Amazon Relational Database Service cost
Amazon Relational Database Service usage
AWS Glue cost
AWS Glue usage
AWS Secrets Manager cost
AWS Secrets Manager usage
Amazon Athena cost
Amazon Athena usage
Amazon ElastiCache cost
Amazon ElastiCache usage
Amazon Simple Email Service cost
Amazon Simple Email Service usage
Amazon Route 53 cost
Amazon Route 53 usage
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

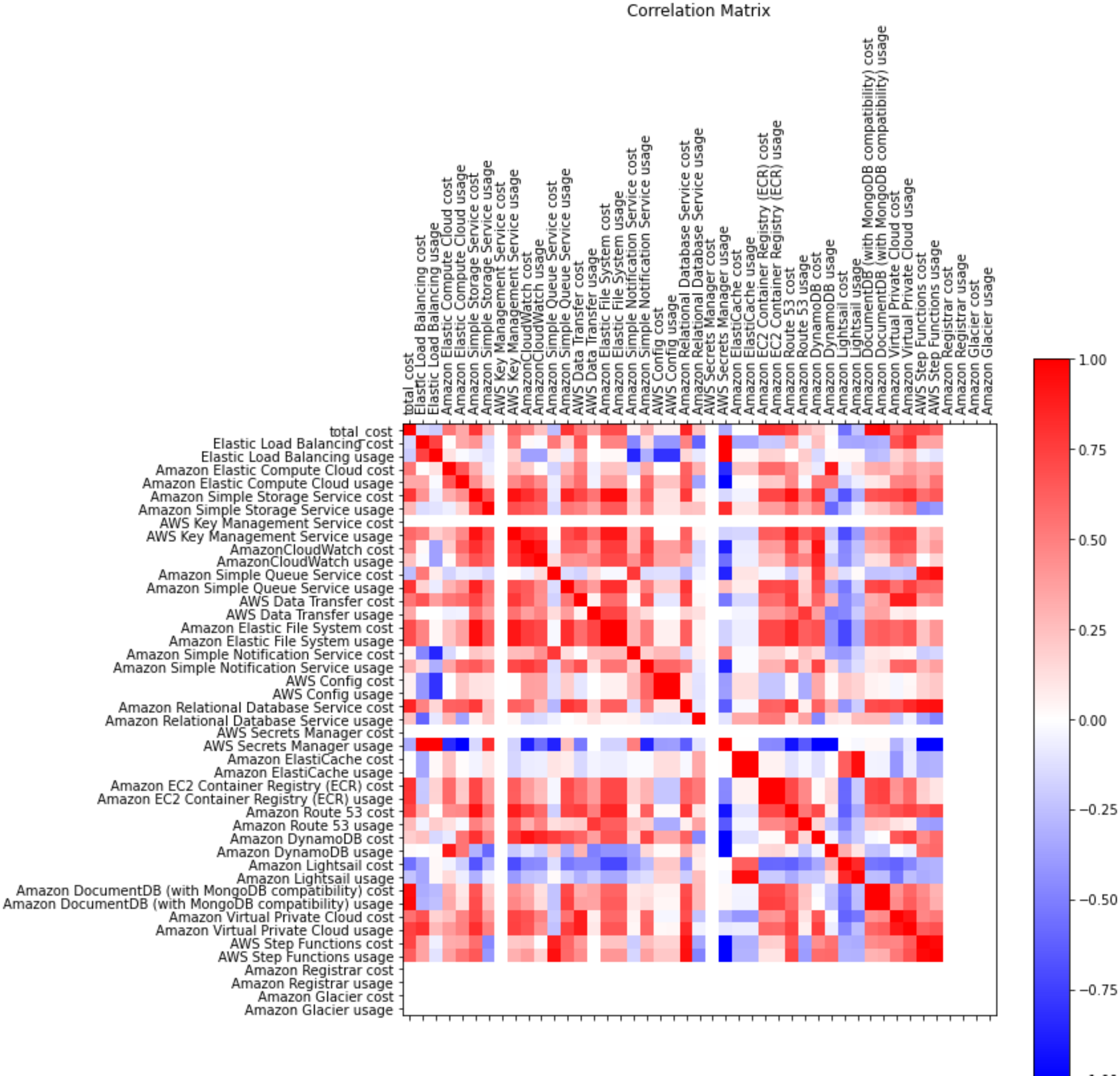
XCO

Interesting points to note:

- Amazon Lightsail cost has a fairly strong negative correlation with total cost.
- Amazon Relational DB Cost, Amazon simple storage, and Amazon DocumentDB are cost variance drivers. This seems to be a company that intakes a good amount of data and the data intake seems to correlate with increasing costs.

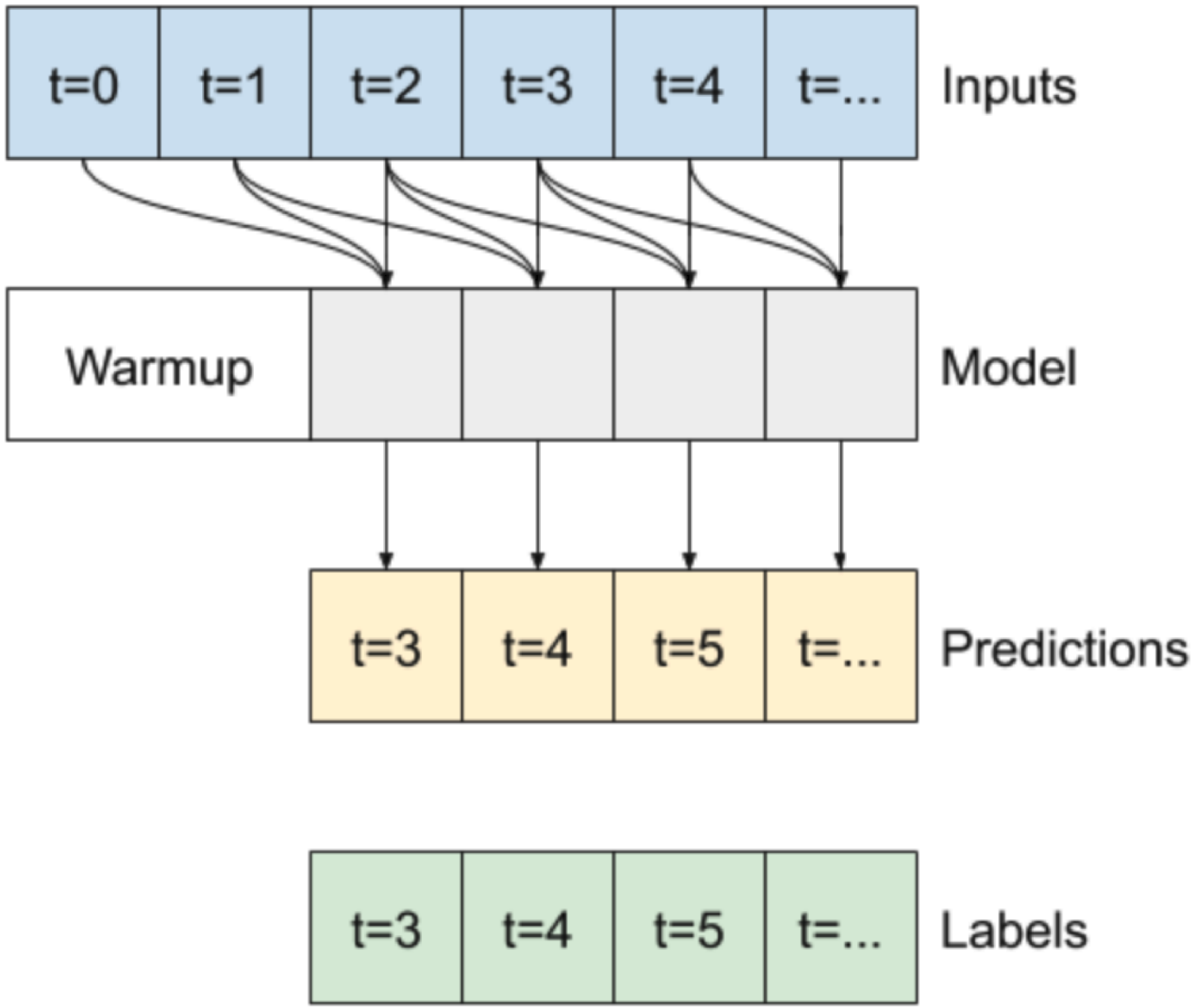
In [22]:

```
dfXCO_processed = process_dataframe(dfXCO, fillna=False)
plot_corr(dfXCO_processed.dropna(axis=1, how="all"))
```


Correlation Matrix
total_cost
Elastic Load Balancing cost
Elastic Load Balancing usage
Amazon Elastic Compute Cloud cost
Amazon Elastic Compute Cloud usage
Amazon Simple Storage Service cost
Amazon Simple Storage Service usage
AWS Key Management Service cost
AWS Key Management Service usage
AmazonCloudWatch cost
AmazonCloudWatch usage
Amazon Simple Queue Service cost
Amazon Simple Queue Service usage
AWS Data Transfer cost
AWS Data Transfer usage
Amazon Elastic File System cost
Amazon Elastic File System usage
Amazon Simple Notification Service cost
Amazon Simple Notification Service usage
AWS Config cost
AWS Config usage
Amazon Relational Database Service cost
Amazon Relational Database Service usage
AWS Secrets Manager cost
AWS Secrets Manager usage
Amazon ElastiCache cost
Amazon ElastiCache usage
Amazon EC2 Container Registry (ECR) cost
Amazon EC2 Container Registry (ECR) usage
Amazon Route 53 cost
Amazon Route 53 usage
Amazon DynamoDB cost
Amazon DynamoDB usage
Amazon Lightsail cost
Amazon Lightsail usage
Amazon DocumentDB (with MongoDB compatibility) cost
Amazon DocumentDB (with MongoDB compatibility) usage
Amazon Virtual Private Cloud cost
Amazon Virtual Private Cloud usage
AWS Step Functions cost
AWS Step Functions usage
Amazon Registrar cost
Amazon Registrar usage
Amazon Glacier cost
Amazon Glacier usage
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

Expected Objective/Modelling

Time series modelling is not well modelled by traditional linear regression or decision trees due to the stochastic nature of the data. Because of this, we have to use modelling that has so-far not been used in this class. The current popular models for time series data are ARIMA (autoregressive integrated moving average) and LTSM Neural Networks (long short-term memory). We would like to put in a the features created in the last section and have an output of the total cost for the next month.


t=0
t=1
t=2
t=3
t=4
t=...
Inputs
Warmup
Model
t=3
t=4
t=5
t=...
Predictions
t=3
t=4
t=5
t=...
Labels

In [6]:

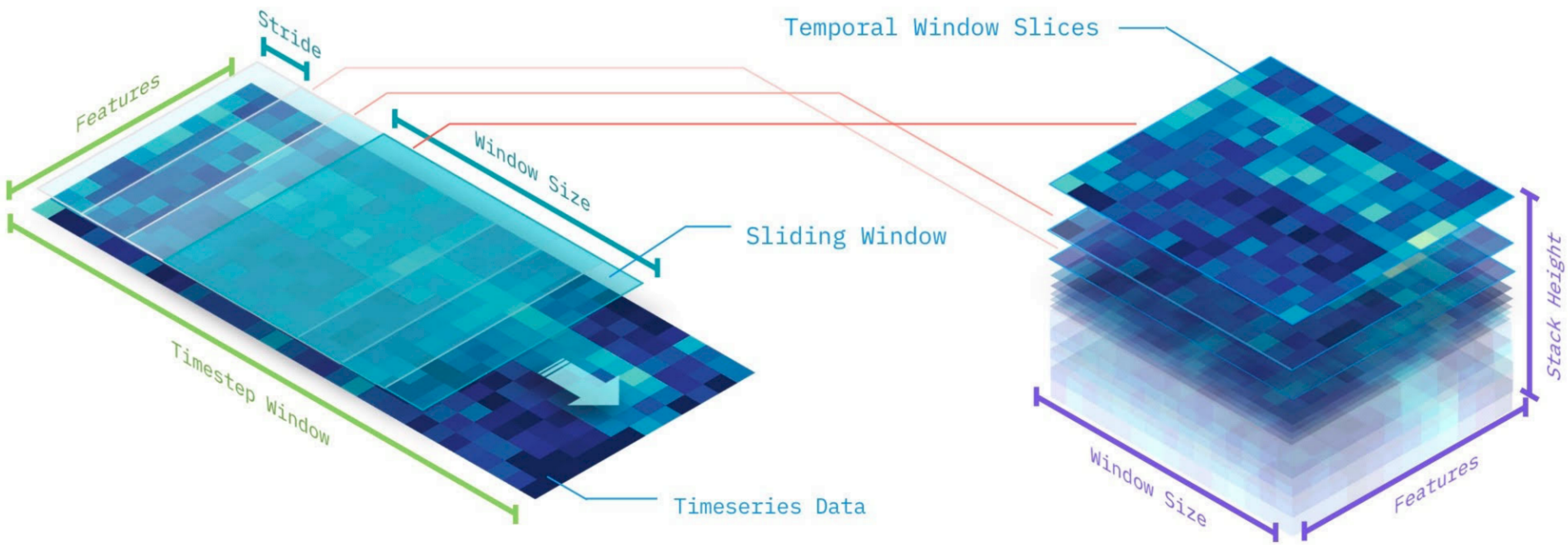
```
def to_supervised (df_for_training, n_future, n_past):
    trainX = []
    trainY = []

    for i in range(n_past, len(df_for_training) - n_future +1):
        trainX.append(df_for_training[i - n_past:i, 0:df_for_training.shape[1]])
        trainY.append(df_for_training[i + n_future - 1:i + n_future, 0])

    trainX, trainY = np.array(trainX), np.array(trainY)

    print('trainX shape == {}.'.format(trainX.shape))
    print('trainY shape == {}.'.format(trainY.shape))

    return trainX, trainY
```



In [7]:

```
def build_model (df_for_training, n_future, n_past):

    trainX, trainY = to_supervised (df_for_training, n_future, n_past)

    model = Sequential()
    model.add(LSTM(64, activation='relu', input_shape=(trainX.shape[1], trainX.shape[2]), return_sequences=True))
    model.add(LSTM(32, activation='relu', return_sequences=False))
    model.add(Dropout(0.2))
    model.add(Dense(trainY.shape[1]))

    model.compile(optimizer='adam', loss='mse') #custom loss function, l2/l1 regularization
    model.summary()

    es = EarlyStopping(monitor='val_loss', min_delta=1e-10, patience=10, verbose=1)
    rlr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=10, verbose=1)
    mcp = ModelCheckpoint(filepath='weights.h5', monitor='val_loss', verbose=1, save_best_only=True, save_weights_only=Tr

    tb = TensorBoard('logs')

    history = model.fit(trainX, trainY, shuffle=True, epochs=30, callbacks=[es, rlr, mcp, tb], validation_split=0.2, verb
    plt.plot(history.history['loss'], label='Training loss')
    plt.plot(history.history['val_loss'], label='Validation loss')
    plt.legend()

    return model, trainX, trainY
```