OSVALDO_GONZALEZ_TESIS.

CASO DE ESTUDIO DE ACCIONES DE COPA AIRLINES

El primer paso es obtener los datos, fueron descargados desde: Yahoo Finance website. Es una sitio lleno de información financiera de diversas empresas, seleccionamos Copa Airlines, los años serán desde el 2006 hasta el 2023. Se descargará un archivo csv que es la fuente de los datos.

Responderemos las siguientes preguntas planteadas:

- 1.) ¿Cuál fue el cambio en el precio de las acciones a lo largo del tiempo?
- 2.) ¿Cuál fue el promedio de movimiento de las distintas acciones?
- 3.) ¿Cuál fue el rendimiento diario de las acciones en promedio?
- 4.) ¿Cuánto valor ponemos en riesgo al invertir en una acción en particular?
- 5.) ¿Valor más bajo y alto de las acciones durante el 2020?
- 6.) ¿Cómo podemos intentar predecir el comportamiento futuro de las acciones? (Predicción del precio de cierre de las acciones de Copa Airlines utilizando LSTM)

Importamos las librerías con sus correspondientes llamados para cada vez que la usemos

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


#Cargar el archivo CSV
stock_data = pd.read_csv('CPA.csv')

#Mostrar el df importado
stock_data
```

Out[2]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2006-01-01	27.000000	27.100000	21.799999	24.000000	16.522408	6386300
1	2006-02-01	24.000000	24.150000	20.400000	23.049999	15.868400	2664500
2	2006-03-01	23.100000	23.790001	20.309999	22.850000	15.730710	4122800
3	2006-04-01	22.920000	23.340000	21.200001	22.350000	15.386494	1918800
4	2006-05-01	22.150000	24.040001	20.950001	23.799999	16.384724	4046200
...	...	...	...	...	...	...	...
210	2023-07-01	113.580002	121.199997	107.599998	118.019997	116.923500	16819500
211	2023-08-01	117.019997	118.220001	93.919998	101.699997	100.755127	16410900
212	2023-09-01	101.959999	103.379997	86.570000	89.120003	88.292007	9293700
213	2023-10-01	88.790001	89.669998	82.220001	84.940002	84.940002	2081800
214	2023-10-06	82.879997	85.510002	82.220001	84.940002	84.940002	359036

215 rows × 7 columns

Con la función "describe()" nos permite visualizar y describir nuestro df

In [3]:
```
stock_data.describe()
```

Out[3]:

	Open	High	Low	Close	Adj Close	Volume
count	215.000000	215.000000	215.000000	215.000000	215.000000	2.150000e+02
mean	76.790093	83.138233	70.458837	76.881023	65.569367	8.803659e+06
std	33.352497	34.330323	31.781764	33.039397	30.678603	4.046253e+06
min	22.150000	23.340000	18.000000	22.170000	15.386494	3.590360e+05
25%	50.850001	56.415001	46.005001	50.865000	38.913048	6.076050e+06
50%	75.879997	83.209999	68.500000	75.529999	68.113297	7.893100e+06
75%	99.595001	108.604999	92.229999	99.099998	89.424213	1.089265e+07
max	158.899994	162.830002	145.789993	160.110001	127.377693	2.657910e+07

Para datos numéricos, el índice del resultado incluirá recuento, media, estándar, mínimo, máximo y percentiles inferior, 50 y superior. De forma predeterminada, el percentil inferior es 25 y el percentil superior es 75. El percentil 50 es el mismo que la mediana.

Continuamos explorando los datos con el método "info()" este nos permite ver el tipo de dato que tienen las columnas, valores nulos y memoria usada.

In [4]:
```
stock_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 215 entries, 0 to 214
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Date       215 non-null    object 
 1   Open       215 non-null    float64
 2   High       215 non-null    float64
 3   Low        215 non-null    float64
 4   Close      215 non-null    float64
 5   Adj Close  215 non-null    float64
 6   Volume     215 non-null    int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 11.9+ KB
```

Indica de forma booleana los objetos con valor NA

In [5]:
```
stock_data.isna().sum()
```

Out[5]:
```
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

En este caso, no hay valores faltantes en ninguna de las columnas del DataFrame stock_data. Cada columna tiene 0 valores faltantes. Esto indica que los datos están completos y no es necesario realizar imputaciones o limpieza adicional debido a valores faltantes en este DataFrame.

El df se encuentra sin errores por la cual podemos seguir sin realizar correcciones

1.) ¿Cuál fue el cambio en el precio de las acciones a lo largo del tiempo?

La biblioteca matplotlib.ticker que se utiliza para personalizar la ubicación de las etiquetas en los ejes de un gráfico. Esto es útil cuando se necesita un control más preciso sobre cómo se muestran las etiquetas de los ejes en un gráfico. En este caso, la función MaxNLocator de matplotlib.ticker se usó para ajustar la ubicación de las etiquetas en el eje x y mostrar solo los años de cinco en cinco para que el gráfico sea más legible.

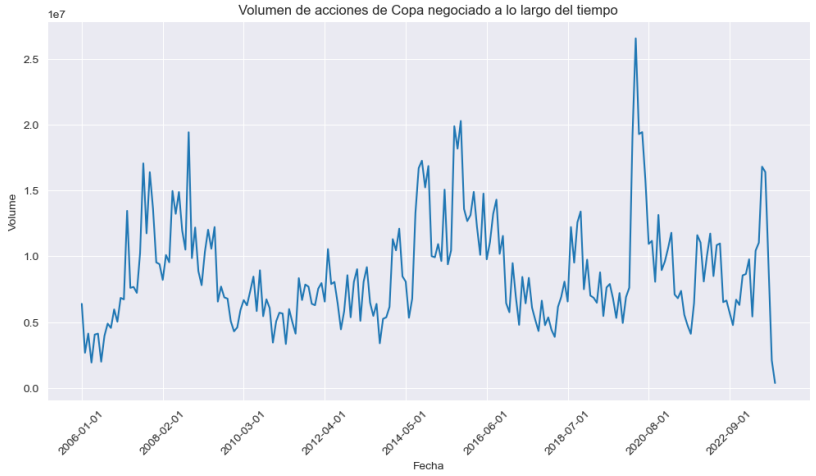
In [6]:
```
import matplotlib.ticker as ticker

# Esto configura el estilo de fondo de las gráficas
sns.set_style("darkgrid")
# Se crea una figura de matplotlib con un tamaño de 12x6 pulgadas.
plt.figure(figsize=(12, 6))
# Crea un gráfico de líneas que muestra cómo el precio de cierre.
sns.lineplot(x='Date', y='Close', data=stock_data)
plt.gca().xaxis.set_major_locator(ticker.MaxNLocator(integer=True, prune='both')
# Rota las etiquetas del eje x en 45 grados para facilitar la legibilidad de las
plt.xticks(rotation=45)
# Título del gráfico.
plt.title('Acciones del Copa en el momento del cierre a través del tiempo')
plt.xlabel('Fecha')
plt.ylabel('Precio de cierre')
# Muestra el gráfico
plt.show()
```

El precio de cierre es el último precio al que se negocia la acción durante un día habitual. El precio de cierre de una acción es el punto de referencia estándar utilizado por los inversores para seguir su desempeño a lo largo del tiempo.

Volumen de ventas

El volumen es la cantidad de un activo o valor que cambia de manos durante un período de tiempo, a menudo en el transcurso de un día. Por ejemplo, el volumen de negociación de acciones se referiría a la cantidad de acciones de valores negociadas entre su apertura y cierre diario. El volumen de operaciones y los cambios en el volumen a lo largo del tiempo son datos importantes para los operadores técnicos.

In [114…
```
plt.figure(figsize=(12, 6))
sns.lineplot(x='Date', y='Volume', data=stock_data)
plt.gca().xaxis.set_major_locator(ticker.MaxNLocator(integer=True, prune='both')
plt.xticks(rotation=45)
plt.title('Volumen de acciones de Copa negociado a lo largo del tiempo')
plt.xlabel('Fecha')
plt.ylabel('Volume')
plt.show()
```



2.) ¿Cuál fue el rendimiento diario de las acciones en promedio?

La media móvil (MA en inglés) es una herramienta de análisis técnico sencilla que suaviza los datos de precios creando un precio medio constantemente actualizado. El promedio se toma durante un período de tiempo específico, como 10 días, 20 días, 50 días o cualquier período de tiempo que elija el comerciante.

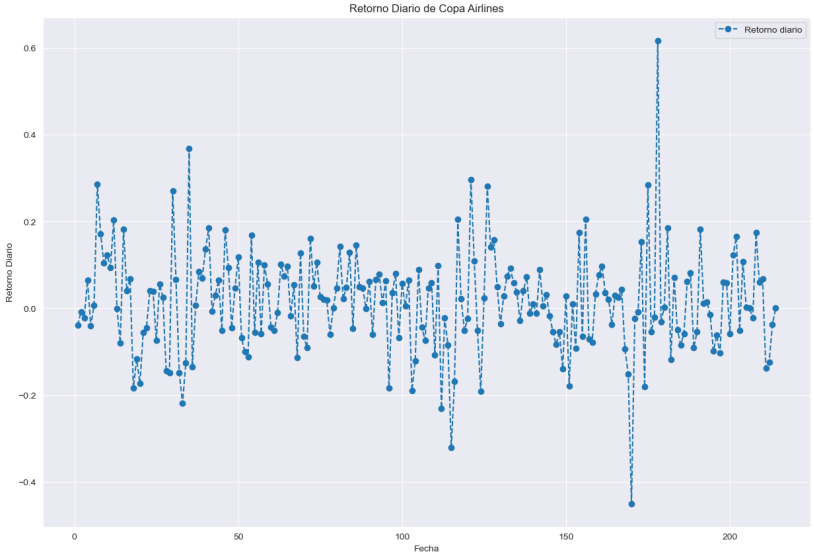
In [115…
```
ma_day = [10, 20, 50]

for ma in ma_day:
    nom_column = f"MA para {ma} días"
    stock_data[nom_column] = stock_data['Adj Close'].rolling(ma).mean()

fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(10, 6))

stock_data[['Adj Close', 'MA para 10 días', 'MA para 20 días', 'MA para 50 días'
axes.set_title('Precio de cierre ajustado y medias móviles de stock_data')
axes.set_xlabel('Fecha')
axes.set_ylabel('Precio de cierre')
axes.grid(True)

plt.tight_layout()
plt.show()
```

Vemos en el gráfico que los mejores valores para medir la media móvil son 10 y 20 días porque todavía capturamos tendencias en los datos sin ruido.

3.) ¿Cuál fue el promedio de movimiento de las distintas acciones?

Ahora que hemos realizado un análisis de referencia, sigamos y profundicemos un poco más. Ahora vamos a analizar el riesgo de la acción. Para ello necesitaremos observar más de cerca los cambios diarios de la acción, y no sólo su valor absoluto. Usaremos pandas para recuperar los rendimientos diarios de las acciones de Copa Airlines.

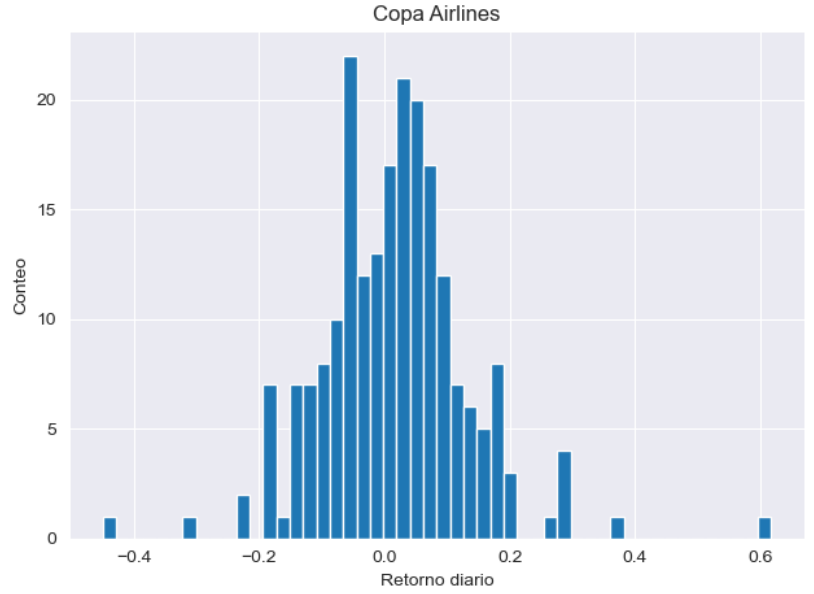
In [116…
```
# Crear el gráfico del precio de cierre ajustado
stock_data['Retorno diario'] = stock_data['Adj Close'].pct_change()

fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(15, 10))  # Asegurémonos de
stock_data['Retorno diario'].plot(ax=axes, legend=True, linestyle='--', marker='
axes.set_title('Retorno Diario de Copa Airlines')
axes.set_xlabel('Fecha')
axes.set_ylabel('Retorno Diario')
axes.grid(True)
```



Ahora usaremos un seaborn para mostrar un histograma y ver el retorno diario.

In [117…

```
stock_data['Retorno diario'].hist(bins=50)
plt.xlabel('Retorno diario')
plt.ylabel('Conteo')
plt.title('Copa Airlines')

plt.tight_layout()
```



La forma del histograma puede proporcionar información sobre la volatilidad y el riesgo de las acciones. Por ejemplo, si el histograma es simétrico y centrado en torno a cero, podría indicar que los retornos tienden a ser equilibrados entre ganancias y pérdidas. Si el histograma está sesgado hacia la izquierda (valores negativos), puede indicar una mayor tendencia a las pérdidas. Si está sesgado hacia la derecha (valores positivos), puede indicar una mayor tendencia a las ganancias.

4.) ¿Cuánto valor ponemos en riesgo al invertir en una acción en particular?

In [118…

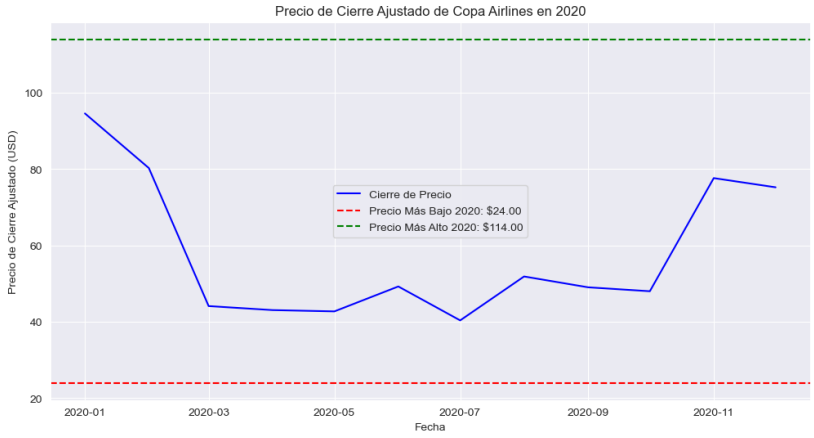
```
# Calcular el rendimiento promedio y la volatilidad
rendimiento_promedio = stock_data['Adj Close'].pct_change().mean()
volatilidad = stock_data['Adj Close'].pct_change().std()

# Crear un gráfico de dispersión
plt.figure(figsize=(10, 8))
plt.scatter(rendimiento_promedio, volatilidad, s=200)

# Etiquetar los puntos con el nombre de la acción (en este caso, la fecha)
for i, fecha in enumerate(stock_data['Date']):
    plt.annotate(fecha, (rendimiento_promedio, volatilidad), textcoords="offset

# Configurar el gráfico
plt.xlabel('Rendimiento Promedio')
plt.ylabel('Volatilidad')
plt.title('Rendimiento vs. Volatilidad de las Acciones')
plt.grid(True)

# Mostrar el gráfico
plt.show()
```

El punto desviado hacia la derecha, eso podría indicar que la mayoría de las acciones han tenido un rendimiento positivo en promedio. La desviación a la derecha también sugiere que hay cierta volatilidad en esos rendimientos, ya que el punto no está perfectamente en el centro.

5.) ¿Valor más bajo y alto de las acciones durante el 2020?

In [121…

```
# Asegúrate de que la columna 'Date' sea de tipo datetime
stock_data['Date'] = pd.to_datetime(stock_data['Date'])

# Filtrar los datos para el año 2020
stock_data_2020 = stock_data[stock_data['Date'].dt.year == 2020]

# Encontrar el precio más bajo y alto de las acciones del 2020
precio_bajo_2020 = stock_data_2020['Low'].min()
precio_alto_2020 = stock_data_2020['High'].max()

# Crear un gráfico del precio de cierre ajustado en 2020
plt.figure(figsize=(12, 6))
plt.plot(stock_data_2020['Date'], stock_data_2020['Adj Close'], label='Cierre de
plt.axhline(y=precio_bajo_2020, color='r', linestyle='--', label=f'Precio Más Ba
plt.axhline(y=precio_alto_2020, color='g', linestyle='--', label=f'Precio Más Al

plt.title('Precio de Cierre Ajustado de Copa Airlines en 2020')
plt.xlabel('Fecha')
plt.ylabel('Precio de Cierre Ajustado (USD)')
plt.legend()
plt.grid(True)
plt.show()
```



Si la diferencia entre el precio más alto y el precio de cierre ajustado es significativamente grande en el año 2020, es posible que la línea que representa el precio más alto (la línea verde en el gráfico) parezca estar muy por encima de la línea que muestra el precio de cierre. Esto puede deberse a movimientos bruscos en los precios de las acciones durante ese año.

6.) ¿Cómo podemos intentar predecir el comportamiento futuro de las acciones? (Predicción del precio de cierre de las acciones de Copa Airlines utilizando LSTM)

Explicación paso a paso de cómo se construye un modelo LSTM:

Importar bibliotecas: Debes importar las bibliotecas necesarias, como TensorFlow o Keras, que te permitirán construir y entrenar tu modelo LSTM.

Preparar los datos: Antes de alimentar los datos al modelo, debes asegurarte de que estén en el formato adecuado. Esto incluye la normalización de datos si es necesario y dividir los datos en conjuntos de entrenamiento y prueba.

Definir el modelo: Crear una instancia de un modelo secuencial (Sequential) o funcional en Keras. Luego, agregar capas LSTM al modelo. El número de capas y neuronas por capa dependerá de la complejidad de tu problema.

Compilar el modelo: Configurar el proceso de entrenamiento del modelo mediante la especificación del optimizador, la función de pérdida y las métricas que se utilizarán para evaluar el rendimiento del modelo.

Entrenar el modelo: Alimentar los datos de entrenamiento al modelo y ajustar los pesos de las capas para minimizar la pérdida. Esto implica múltiples épocas de entrenamiento.

Evaluar el modelo: Utilizar datos de prueba para evaluar el rendimiento del modelo. Esto suele incluir métricas como el error cuadrático medio (RMSE) para problemas de regresión o la precisión para problemas de clasificación.

Hacer predicciones: Una vez que el modelo está entrenado y evaluado, puedes usarlo para hacer predicciones en nuevos datos o secuencias.

Visualizar resultados: A menudo, es útil visualizar las predicciones del modelo junto con los datos reales para evaluar su calidad.

In [7]:
```
plt.figure(figsize=(16,6))
plt.title('Precio de cierre historico')
plt.plot(stock_data['Close'])
plt.xlabel('Fecha', fontsize=18)
plt.ylabel('Precio de cierre USD ($)', fontsize=18)
plt.show()
```

In [21]:
```
# Creamos un nuevo DataFrame con la columna 'Cierre'
datos = stock_data.filter(['Close'])

# Convertimos el DataFrame a un arreglo de NumPy
conjunto_datos = datos.values

# Obtenemos la longitud de los datos de entrenamiento
longitud_datos_entrenamiento = int(np.ceil(len(conjunto_datos) * 0.95))

longitud_datos_entrenamiento
```

Out[21]:
```
205
```

In [23]:
```
# Importar la clase MinMaxScaler del módulo preprocessing de scikit-learn
from sklearn.preprocessing import MinMaxScaler

# Crear una instancia de MinMaxScaler y definir el rango de escalado de 0 a 1
escalador = MinMaxScaler(feature_range=(0, 1))

# Usar el escalador para transformar (escalar) el conjunto de datos original (da
datos_escalados = escalador.fit_transform(dataset)

datos_escalados
```

Out[23]:
```
array([[0.01326664],
       [0.00637958],
       [0.00492968],
       [0.00130492],
       [0.01181672],
       [0.00347977],
       [0.0045672 ],
       [0.05183413],
       [0.08815428],
       [0.1140351 ],
       [0.14774539],
       [0.17681601],
       [0.24561403],
       [0.24539655],
       [0.2125562 ],
       [0.28062926],
       [0.29810063],
       [0.32673625],
       [0.23734957],
       [0.19066261],
       [0.12962157],
       [0.11338264],
       [0.10120343],
       [0.11164274],
       [0.12222705],
       [0.10105844],
       [0.1155575 ],
       [0.12266203],
       [0.08191968],
       [0.04342468],
       [0.09881107],
       [0.11606495],
       [0.07488763],
       [0.0231985 ],
       [0.        ],
       [0.05908366],
       [0.02950558],
       [0.03088299],
       [0.04712194],
       [0.061476  ],
       [0.09170654],
       [0.13520371],
       [0.13317385],
       [0.14209076],
       [0.1618095 ],
       [0.14542555],
       [0.20052198],
       [0.23415979],
       [0.21610845],
       [0.23357981],
       [0.28004929],
       [0.25018124],
       [0.20900391],
       [0.15985212],
       [0.21371612],
       [0.19290995],
       [0.23010004],
       [0.20704654],
       [0.24358417],
       [0.26584022],
       [0.24706394],
       [0.22589533],
       [0.22205306],
       [0.26083806],
       [0.29230101],
       [0.32311148],
       [0.31491954],
       [0.3403654 ],
       [0.28345657],
       [0.3400029 ],
       [0.30730751],
       [0.26460778],
       [0.33326083],
       [0.35812672],
       [0.4134406 ],
       [0.42873711],
       [0.4411338 ],
       [0.4372191 ],
       [0.4013339 ],
       [0.40205884],
       [0.42844713],
       [0.5121792 ],
       [0.52682322],
       [0.56024356],
       [0.63382628],
       [0.59627372],
       [0.70639409],
       [0.74967378],
       [0.79128611],
       [0.78983612],
       [0.84819485],
       [0.78737131],
       [0.84457008],
       [0.92337242],
       [0.93700157],
       [1.        ],
       [0.78679133],
       [0.82129916],
       [0.89183704],
       [0.81999419],
       [0.87545304],
       [0.87284331],
       [0.94026384],
       [0.73082501],
       [0.61707989],
       [0.68689283],
       [0.65057274],
       [0.5906191 ],
       [0.6186748 ],
       [0.66485426],
       [0.57126287],
       [0.64317818],
       [0.45759026],
       [0.4380165 ],
       [0.38683485],
       [0.21096129],
       [0.14325069],
       [0.20552414],
       [0.21371612],
       [0.18914019],
       [0.18073074],
       [0.28193418],
       [0.33043352],
       [0.3014354 ],
       [0.21313614],
       [0.2181383 ],
```

```
[0.32499637],
[0.39335943],
[0.47672901],
[0.507902  ],
[0.48354359],
[0.49775266],
[0.54603449],
[0.61128025],
[0.65303755],
[0.68326807],
[0.65876468],
[0.68747281],
[0.74880382],
[0.73872698],
[0.74206175],
[0.73234741],
[0.81209222],
[0.81114975],
[0.84210527],
[0.82499638],
[0.77178486],
[0.68870521],
[0.64252575],
[0.52522838],
[0.54494705],
[0.41880529],
[0.41808029],
[0.36436131],
[0.45570537],
[0.40988835],
[0.52689573],
[0.47846888],
[0.42366247],
[0.44287372],
[0.48890822],
[0.54661447],
[0.57220529],
[0.58728432],
[0.55516891],
[0.57684499],
[0.59511383],
[0.62280703],
[0.54944178],
[0.44200376],
[0.16760911],
[0.15977961],
[0.15724229],
[0.20581413],
[0.13969841],
[0.22531535],
[0.20421922],
[0.19653472],
[0.41706537],
[0.39915907],
[0.40017401],
[0.50384223],
[0.42496738],
[0.46636218],
[0.4356242 ],
[0.38538496],
[0.35334206],
[0.38480498],
[0.42924457],
[0.37545309],
[0.34616498],
[0.43852402],
[0.44519357],
[0.45432794],
[0.44562852],
[0.38567495],
[0.35181961],
[0.29868058],
[0.32659128],
[0.35544437],
[0.32506888],
[0.38466002],
[0.47455413],
[0.44222124],
[0.50681457],
[0.50877191],
[0.50877191],
[0.49405538],
[0.60091342],
[0.64093085],
[0.69486731],
[0.576555  ],
[0.48535597],
[0.45505293],
[0.45505293]])
```

In [25]:

```
dato_entrenados = datos_escalados[0:int(longitud_datos_entrenamiento), :]

x_entrenada = []
y_entrenada = []

for i in range(60, len(dato_entrenados)):
    x_entrenada.append(dato_entrenados[i-60:i, 0])
    y_entrenada.append(dato_entrenados[i, 0])
    if i<= 61:
        print(x_entrenada)
        print(y_entrenada)
        print()

x_entrenada, y_entrenada = np.array(x_entrenada), np.array(y_entrenada)

x_entrenada = np.reshape(x_entrenada, (x_entrenada.shape[0], x_entrenada.shape[1
```

```
[array([0.01326664, 0.00637958, 0.00492968, 0.00130492, 0.01181672,
       0.00347977, 0.0045672 , 0.05183413, 0.08815428, 0.1140351 ,
       0.14774539, 0.17681601, 0.24561403, 0.24539655, 0.2125562 ,
       0.28062926, 0.29810063, 0.32673625, 0.23734957, 0.19066261,
       0.12962157, 0.11338264, 0.10120343, 0.11164274, 0.12222705,
       0.10105844, 0.1155575 , 0.12266203, 0.08191968, 0.04342468,
       0.09881107, 0.11606495, 0.07488763, 0.0231985 , 0.        ,
       0.05908366, 0.02950558, 0.03088299, 0.04712194, 0.061476  ,
       0.09170654, 0.13520371, 0.13317385, 0.14209076, 0.1618095 ,
       0.14542555, 0.20052198, 0.23415979, 0.21610845, 0.23357981,
       0.28004929, 0.25018124, 0.20900391, 0.15985212, 0.21371612,
       0.19290995, 0.23010004, 0.20704654, 0.24358417, 0.26584022])]
[0.24706393905274804]

[array([0.01326664, 0.00637958, 0.00492968, 0.00130492, 0.01181672,
       0.00347977, 0.0045672 , 0.05183413, 0.08815428, 0.1140351 ,
       0.14774539, 0.17681601, 0.24561403, 0.24539655, 0.2125562 ,
       0.28062926, 0.29810063, 0.32673625, 0.23734957, 0.19066261,
       0.12962157, 0.11338264, 0.10120343, 0.11164274, 0.12222705,
       0.10105844, 0.1155575 , 0.12266203, 0.08191968, 0.04342468,
       0.09881107, 0.11606495, 0.07488763, 0.0231985 , 0.        ,
       0.05908366, 0.02950558, 0.03088299, 0.04712194, 0.061476  ,
       0.09170654, 0.13520371, 0.13317385, 0.14209076, 0.1618095 ,
       0.14542555, 0.20052198, 0.23415979, 0.21610845, 0.23357981,
       0.28004929, 0.25018124, 0.20900391, 0.15985212, 0.21371612,
       0.19290995, 0.23010004, 0.20704654, 0.24358417, 0.26584022]), array([0.00
637958, 0.00492968, 0.00130492, 0.01181672, 0.00347977,
       0.0045672 , 0.05183413, 0.08815428, 0.1140351 , 0.14774539,
       0.17681601, 0.24561403, 0.24539655, 0.2125562 , 0.28062926,
       0.29810063, 0.32673625, 0.23734957, 0.19066261, 0.12962157,
       0.11338264, 0.10120343, 0.11164274, 0.12222705, 0.10105844,
       0.1155575 , 0.12266203, 0.08191968, 0.04342468, 0.09881107,
       0.11606495, 0.07488763, 0.0231985 , 0.        , 0.05908366,
       0.02950558, 0.03088299, 0.04712194, 0.061476  , 0.09170654,
       0.13520371, 0.13317385, 0.14209076, 0.1618095 , 0.14542555,
       0.20052198, 0.23415979, 0.21610845, 0.23357981, 0.28004929,
       0.25018124, 0.20900391, 0.15985212, 0.21371612, 0.19290995,
       0.23010004, 0.20704654, 0.24358417, 0.26584022, 0.24706394])]
[0.24706393905274804, 0.2258953296658306]
```

In [26]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# Construir el modelo LSTM
modelo = Sequential()
modelo.add(LSTM(128, return_sequences=True, input_shape=(x_entrenada.shape[1], 1
modelo.add(LSTM(64, return_sequences=False))
modelo.add(Dense(25))
modelo.add(Dense(1))

# Compilar el modelo
modelo.compile(optimizer='adam', loss='mean_squared_error')

# Entrenar el modelo
modelo.fit(x_entrenada, y_entrenada, batch_size=1, epochs=1)
```

```
145/145 [==============================] - 7s 19ms/step - loss: 0.0310
```

Out[26]:
```
<keras.src.callbacks.History at 0x1bfae7ede50>
```

modelo: Esta es la instancia del modelo de red neuronal que se ha creado previamente.

compile: Este método compila el modelo y se debe llamar antes de entrenarlo. La compilación del modelo implica especificar cómo se llevará a cabo el proceso de entrenamiento.

optimizer='adam': Aquí se selecciona el optimizador que se utilizará durante el entrenamiento. 'Adam' es un optimizador popular que ajusta la velocidad de aprendizaje de manera adaptativa durante el proceso de entrenamiento para mejorar la convergencia.

loss='mean_squared_error': El parámetro de pérdida (loss) especifica la función de pérdida que se utilizará para evaluar qué tan bien se está desempeñando el modelo durante el entrenamiento. 'Mean squared error' (error cuadrático medio) es una función de pérdida comúnmente usada en problemas de regresión. Mide la diferencia al cuadrado entre las predicciones del modelo y los valores reales y busca minimizar esta diferencia durante el entrenamiento.

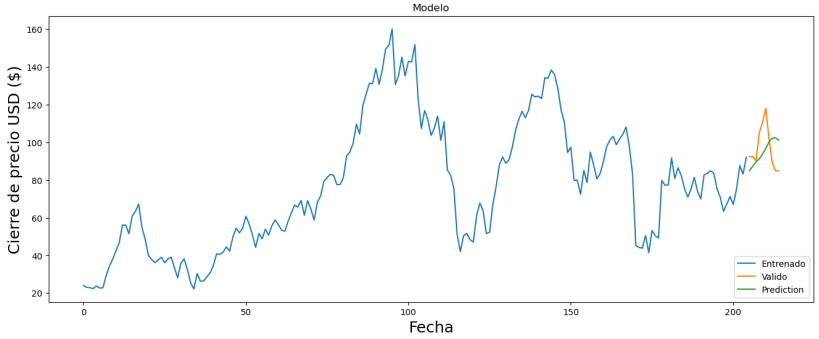
In [34]:
```
# Crear el conjunto de datos de prueba
# Selecciona los datos de prueba, comenzando 60 pasos antes del final
test_data = datos_escalados[longitud_datos_entrenamiento -60: , :]
# Lista para almacenar las secuencias de entrada
x_prueba = []
# Valores reales de prueba
y_prueba = dataset[longitud_datos_entrenamiento:, :]
for i in range(60, len(test_data)):
    x_prueba.append(test_data[i-60:i, 0])
x_prueba = np.array(x_test) # Convertir a un arreglo numpy

x_prueba = np.reshape(x_prueba, (x_prueba.shape[0], x_prueba.shape[1], 1))
# Realizar predicciones con el modelo
prediccion = model.predict(x_prueba)
prediccion = scaler.inverse_transform(prediccion)
# Calcular la raíz del error cuadrático medio (RMSE)

rmse = np.sqrt(np.mean(((prediccion - y_prueba) ** 2)))   # Calcular el RMSE par
rmse
```

```
1/1 [==============================] - 0s 34ms/step
```

Out[34]:
```
13.232994438324507
```

In [37]:
```
entrenar = data[:longitud_datos_entrenamiento]
validar = data[training_data_len:]

valid['Prediccion'] = prediccion

#visualizar los datos

plt.figure(figsize=(16,6))
plt.title('Modelo')
plt.xlabel('Fecha', fontsize=18)
plt.ylabel('Cierre de precio USD ($)', fontsize=18)
plt.plot(train['Close'])
plt.plot(valid[['Close', 'Prediccion']])
plt.legend(['Entrenado', 'Valido', 'Prediction'], loc='lower right')
plt.show()
```

```
C:\Users\Osvaldo G\AppData\Local\Temp\ipykernel_13056\1853764126.py:4: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  valid['Prediccion'] = prediccion
```



Explicación del modelo:

entrenar: Esta línea muestra la serie temporal de precios de cierre de entrenamiento. Estos son los datos que el modelo utilizó para aprender y ajustarse.

Validar: Esta línea representa la serie temporal de precios de cierre de validación. Estos son datos independientes que el modelo no ha visto durante el entrenamiento. La línea "Valido" muestra cómo se comparan las predicciones del modelo con los datos reales en este conjunto de validación. Cualquier discrepancia entre las predicciones del modelo y los datos reales se hará evidente en esta parte del gráfico.

Predicción: Esta línea muestra las predicciones del modelo para los precios de cierre. Estas predicciones se generan utilizando el modelo entrenado y se comparan con los datos reales en la línea "Valido". Si el modelo es preciso, la línea "Predicion" debería seguir de cerca la línea "Valido".

In [ ]: